

# cg\_full

## Vorwort

---

Diese Stoffsammlung/Zusammenfassung enthält den Stoff, der in der EVC Vorlesung der TU Wien im Sommersemester 2025 vorgetragen wurde, der auch in den jeweiligen Skripten und Slides zu finden ist. Die Struktur dieser Zusammenfassung basiert demnach auch der des Skriptums.

### Disclaimer

Vieles der Zusammenfassung wurde mit AI generiert, basiert allerdings nur auf Inhalten der Unterlagen. Die Stellen die mit AI generiert wurden, wurden von mir überprüft und mit den Unterlagen verglichen, aber auch ich kann Fehler machen.

Demnach, falls sich irgendwo Fehler befinden oder es Verbesserungsvorschläge gibt, bitte an [@xmozz](#) auf Discord wenden.

# Inhalt

---

- 1. Einführung
- 2. Graphikpipeline und Objektrepräsentationen
- 3. Transformationen
- 4. Farbe
- 5. Rasterisierung
- 6. Viewing
- 7. Clipping und Antialiasing
- 8. Sichtbarkeitsverfahren
- 9. Beleuchtung und Schattierung
- 10. Ray-Tracing
- 11. Globale Beleuchtung und Texturen
- 12. Kurven und Flächen
- 13. Computer Animation
- 14. Machine Learning für 3D Graphics

# **cv\_full**

## **Vorwort**

---

Diese Stoffsammlung/Zusammenfassung enthält den Stoff, der in der EVC Vorlesung der TU Wien im Sommersemester 2025 vorgetragen wurde, der auch in den jeweiligen Skripten und Slides zu finden ist. Die Struktur dieser Zusammenfassung basiert demnach auch der des Skriptums.

### **Disclaimer**

Vieles der Zusammenfassung wurde mit AI generiert, basiert allerdings nur auf Inhalten der Unterlagen. Die Stellen die mit AI generiert wurden, wurden von mir überprüft und mit den Unterlagen verglichen, aber auch ich kann Fehler machen.

Demnach, falls sich irgendwo Fehler befinden oder es Verbesserungsvorschläge gibt, bitte an [@xmozz](#) auf Discord wenden.

# Inhalt

---

- 1. Einführung in Computer Vision
- 2. Bildaufnahme
- 3. Bildcodierung und Kompression
- 4. Punktoperationen
- 5. Lokale Operationen
- 6. Kantenfilterung
- 7. Globale Operationen
- 8. Bildmerkmale - Interest Points
- 9. Multiskalenrepräsentationen
- 10. Stereo und Motion
- 11. Deep Learning
- 12. Computational Photography

# 1. Einführung in die Computergraphik

Quellen: [EVC\\_Skriptum\(CG, p.5\)](#), [EVC\\_Skriptum\(CG, p.6\)](#)

---

**Definition:** Computergraphik ist ein Teilgebiet der Informatik, das sich mit der künstlichen Erzeugung und Manipulation von Bildern beschäftigt, inklusive der digitalen Repräsentation, Erzeugung und Manipulation der zugehörigen Daten.

**Anwendungsbereiche:** Die Notwendigkeit künstlicher Bilder wächst mit der Computerisierung vieler Lebensbereiche. Typische Anwendungen:

- **Entertainment:**

- **Computerspiele:** Größter Markt, treibende Kraft in der Forschung.
- **Filmindustrie:** Erzeugung unmöglicher Szenen, Nachbearbeitung, Ergänzung von Inhalten; Verschmelzung mit klassischer Filmproduktion und Animation.

- **Computer Aided Design (CAD):**

- Industrielle Produktentwicklung und visuelle Inspektion (Geräte, Behältnisse, Sportartikel, Schmuck, Autos, Flugzeuge, Fenster).
- Architektur: Virtuelle Begehung vor Bau.
- Straßen- und Landschaftsplanung: Visuelle Vorwegnahme und Planung.

- **Werbung:**

- Langjährige Nutzung von Computergraphik.
- Finanzstarke Spotindustrie (Kurzanimationen, Manipulationen).
- Marketing mit interaktiven visuellen Methoden -> Imagevorteil.

- **Simulatoren:**

- Training in teuren oder gefährlichen Technologien (Flugzeugpiloten, Raumfahrer, Autosimulation).
- Simulation von Gefahren- und Katastrophensituationen zur Vorbereitung und Schulung.
- Nutzung von wahrnehmungsbasiertem Rendering (Berücksichtigung der Augenwahrnehmung).

- **Kulturerbe:**

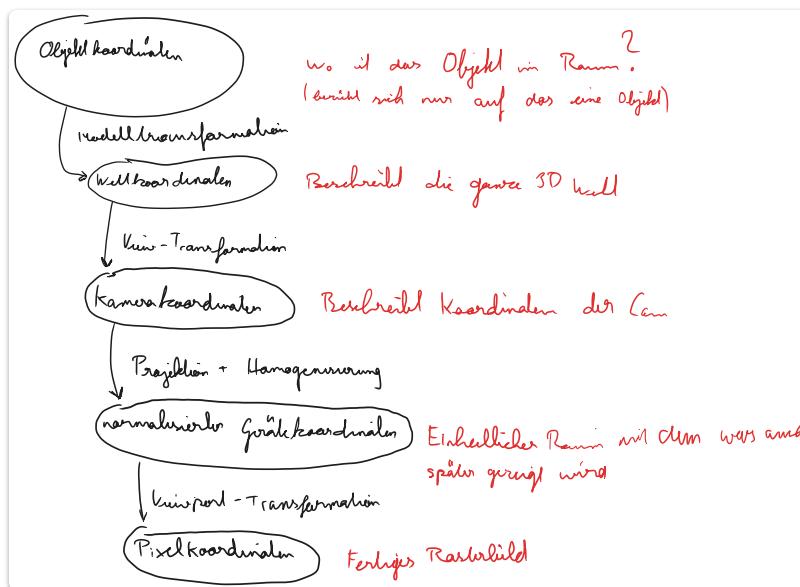
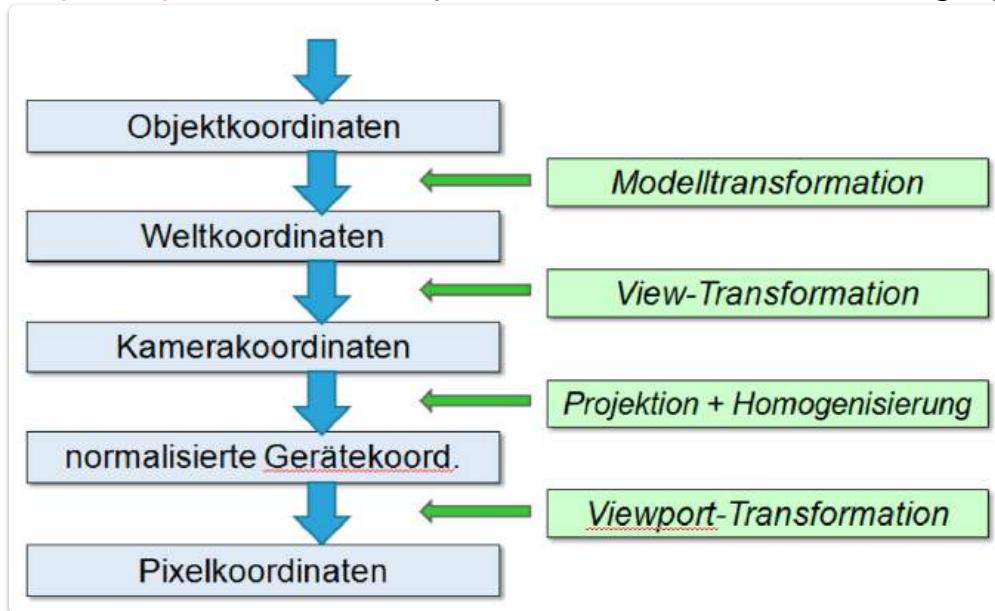
- Virtuelle Bewahrung/Wiederherstellung von geschädigten Dingen und Bauwerken.
- Unterstützung von Museen und Bildung (z.B. Geschichtsunterricht).

- **Wissenschaft:**

- **Visualisierung:** Finden von Strukturen und Informationen in unübersichtlichen/großen Datenmengen (explorativ und routinemäßig, z.B. Medizinische Bildgebung).

**Komponenten von Computergraphik-Software:** Ein Graphiksystem umfasst viele Schritte von der Datenmodellierung bis zur Bilddarstellung.

- **Graphik-Pipeline:** Kette von Operationen und Daten zur Bilderzeugung.



- [2. Graphikpipeline und Objektrepräsentationen](#)
- **Graphik-Primitive:** Einfache geometrische Grundformen (Linien, Kreise, Rechtecke) zur geräteunabhängigen Darstellung. [2. Graphikpipeline und Objektrepräsentationen](#)
- **Rasterisierung:** Umwandlung von Primitiven in Pixelinformationen. [5. Rasterisierung](#)
- **Objektmodellierung:** Kombination geometrischer Primitiven (inkl. Freiformflächen) und Speicherung in geeigneten geometrischen Datenstrukturen. [2. Graphikpipeline und Objektrepräsentationen](#)
- **Platzierung:** Anordnung der Objekte im Weltkoordinatensystem.
- **Projektion:** Festlegung der Ansicht durch Kameraparameter.
- **Geometrische Transformationen:** Homogene Matrizen für Platzierung und Projektion. [3. Transformationen](#)

- **Clipping:** Entfernen von Bildteilen außerhalb des Betrachtungsfensters. [7. Clipping und Antialiasing](#)
- **Sichtbarkeitsberechnung:** Entfernen verdeckter Bildinformationen. [8. Sichtbarkeitsverfahren](#)
- **Beleuchtungsmodelle:** Einfache Schattierung bis realistisches Ray-Tracing und globale Beleuchtung. [8. Sichtbarkeitsverfahren](#)
- **Texturen:** Zusätzliche Oberflächeninformationen, kombinierbar mit lokalen geometrischen Strukturen.
- **Anti-Aliasing:** Verfahren zur Reduktion des Rastereindrucks bei der Bildausgabe. [7. Clipping und Antialiasing](#)



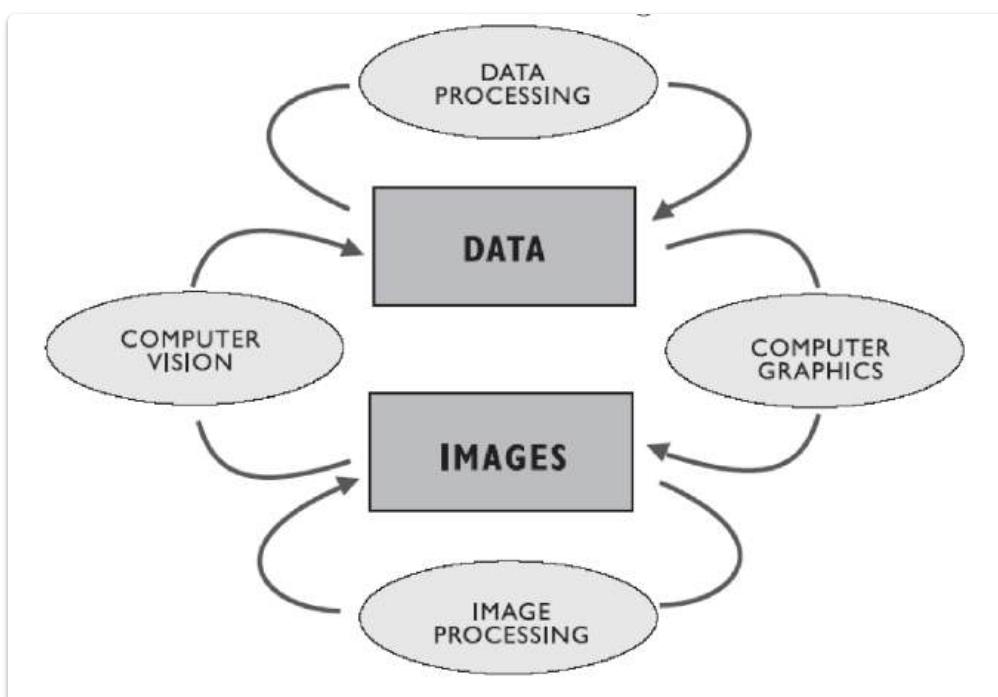
# 1. Einführung in Computer Vision

Quellen:

- [EVC\\_Skriptum\\_CV, p.4](#)
- [EVC\\_Skriptum\\_CV, p.5](#)
- [EVC\\_Skriptum\\_CV, p.6](#)
- [EVC\\_Skriptum\\_CV, p.7](#)

## Die drei Teilbereiche

- **Computergrafik:**
  - Erzeugt Bilder aus Daten (z.B. Diagramme, 3D-Modelle)
  - Ziel: Realistische Bildgenerierung.
- **Computer Vision (Maschinelles Sehen):**
  - Extrahiert Semantik aus Bildern (also genau solche Diagramme / Modelle).
  - Ziel: Nachbildung des Sehvorgangs, Erkennung von Objekten, Bewegung, etc.
  - Teilbereich: Mustererkennung (z.B. optische Zeichenerkennung).
- **Bildverarbeitung:**
  - Verbessert Bilder für bestimmte Aufgaben (z.B. Rauschunterdrückung, Qualitätsverbesserung).
  - Ergebnis: Ein verändertes Bild.
  - Unterschied zur Bildbearbeitung: Bildverarbeitung ist algorithmisch, Bildbearbeitung ist interaktiv.



# Definition Computer Vision

- Automatische Ableitung von Struktur und Eigenschaften einer 3D-Welt aus 2D-Bildern.
- Ziel: Computer sollen visuelle Daten wahrnehmen, verarbeiten und verstehen wie Menschen.
- Anwendungsbereiche:
  - Industrielle Bildverarbeitung (z.B. Flaschenerkennung).
  - Künstliche Intelligenz (z.B. autonome Roboter).
  - Gesichtserkennung.
- Computer Vision vs. Machine Vision: Computer Vision ist die Kerntechnologie zur Bildanalyse, Machine Vision kombiniert die Bildanalyse mit anderen Technologien.
- Teilgebiete: Szenenrekonstruktion, Objekterkennung, Video-Tracking, etc.
- Beispiel Gesichtserkennung:



Abbildung 2: Gesichtserkennung

## 3. Menschliches Sehen vs. Computer Vision

- Computer Vision versucht, menschliches Sehen zu reproduzieren, aber:
  - Menschliches Sehen ist Ergebnis von Millionen Jahren Evolution
  - Menschliches Sehen ist fehlerhaft (optische Täuschungen).
  - Wenig Wissen über die genauen Prozesse des menschlichen Sehens.
  - Die Frage besteht, ob es erstrebenswert ist, dass Maschinen genauso sehen wie Menschen.
- Das menschliche Sehen ist nicht unfehlbar. Optische Täuschungen und Mehrdeutigkeiten beweisen dies.
- In CV Herleitung durch geometrischen Eigenschaften, Materialeigenschaften und Beleuchtung
- **Begrenztes Wissen:**
  - Unser Verständnis der komplexen Prozesse im Gehirn, die nach der Reizaufnahme durch das Auge ablaufen, ist begrenzt.

- **Fehlerhaftigkeit des menschlichen Sehens:**
  - Optische Täuschungen, Mehrdeutigkeiten und Inkonsistenzen zeigen, dass unser Sehen nicht perfekt ist.
- **Subjektivität der Wahrnehmung:**
  - Die Drehung eines Bildes kann unsere Wahrnehmung komplett verändern.
- **Helmholtz' Theorie:**
  - Wir sehen, was wir erwarten oder was unter normalen Bedingungen zu erwarten wäre.
  - Unser Gehirn konstruiert aktiv unsere visuelle Realität.
- **Toleranzunterschiede:**
  - Wir akzeptieren die Unvollkommenheit des menschlichen Sehens, sind aber bei Maschinen weniger tolerant.
- **Die Frage:**
  - Wollen wir, dass Maschinen genauso "sehen" wie wir, mit all den damit verbundenen Fehlern und Verzerrungen?

## 2. Optische Täuschungen und ihre Bedeutung

- **Einblicke in die Natur des Sehens:**
  - Optische Täuschungen geben uns wertvolle Einblicke in die Funktionsweise unseres visuellen Systems.
- **Kontrollierte Halluzination?:**
  - Die Frage, ob unser Sehen eher einer "kontrollierten Halluzination" gleicht, wird aufgeworfen.
- **Zuverlässigkeit der Wahrnehmung:**
  - Die Täuschungen verdeutlichen, dass unser Vertrauen in unsere visuellen Fähigkeiten nicht immer gerechtfertigt ist.

## 3. Konsequenzen für die Computer Vision

- **Nachbildung des menschlichen Sehens:**
  - Die Schwierigkeiten bei der Nachbildung des menschlichen Sehens werden deutlich.
- **Erwartungen an Maschinen:**
  - Wir müssen uns fragen, welche Art von "Sehen" wir von Maschinen erwarten.
- **Grundlagenforschung:**
  - Die Erforschung optischer Täuschungen hilft, die Grundlagen des menschlichen Sehens besser zu verstehen, was wiederum in der Entwicklung von Computer Vision Systemen hilfreich ist.

## 4. Weitere Informationen:

- Weitere Infos zu wie wir sehen findet man hier:
  - [2. Bildaufnahme](#)

- 4. Farbe

## 4. Plenoptische Funktion

- Beschreibt die Menge aller Lichtstrahlen in einem Raum.
- Parameter zum Beschreiben der Beleuchtung:
  - Position ( $V_x, V_y, V_z$ ).
  - Einfallswinkel ( $\theta, \phi$ ).
  - Wellenlänge ( $\lambda$ ).
  - Zeit ( $t$ ).

Wir wollen nun die Parameter zur Beschreibung beleuchteter Umgebungen betrachten. Im ersten Schritt nehmen wir eine Schwarz-weißfotografie einer Lochkamera an, die uns darüber Auskunft gibt, wie groß die Intensität des Lichtes von einem einzigen Blickpunkt aus zu einem bestimmten Zeitpunkt, gemittelt über die Wellenlängen des Spektrums des sichtbaren Lichtes, ist. Diese Lichtintensitätsverteilung  $P$  kann beim Durchgang der Strahlenbündel durch die Linse gemessen und anhand von Kugelkoordinaten  $P(\theta, \phi)$  oder kartesischen Koordinaten  $P(x, y)$  parametrisiert werden.

Ein Farbbild enthält zusätzliche Informationen, welche die Veränderung der Lichtintensität bezüglich der Wellenlänge  $\lambda$  berücksichtigt (daher  $P(\theta, \phi, \lambda)$ ). Ein Farbvideo oder Film erweitert die Informationen um die Dimension der Zeit  $t$ :  $P(\theta, \phi, \lambda, t)$ . Schlussendlich zeigen farbige holografische Filme die komplette wahrnehmbare Lichtintensität von jeder Betrachtungsposition  $V_x, V_y$  und  $V_z$  an:  $P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$ . Solch eine vollständige Repräsentation impliziert die Beschreibung aller möglichen Bilder, die von einem bestimmten Raum-Zeit Stück der Welt aufgenommen werden können (unter Vernachlässigung der Polarisation und momentanen Phase des einfallenden Lichtes). Es ist zu beachten, dass die plenoptische Funktion keine zusätzlichen Parameter zur Spezifizierung der "Blickrichtung" des Auges benötigt. Die Veränderung der Blickrichtung ohne Veränderung der Position des Auges hat keine Auswirkung auf die Verteilung des Lichtes eines Strahlenbündels beim Auftreffen auf die Pupille. Nur die relative

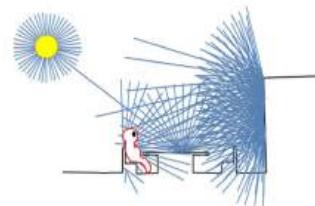


Abbildung 4: Plenoptische Funktion

Einfallposition des Lichtes auf der Netzhaut wird dadurch verändert. Die Messung der plenoptischen Funktion erfolgt durch die imaginäre Platzierung eines idealen Auges an jeder möglichen ( $V_x, V_y, V_z$ ) Position und beinhaltet die Messung der Intensität der Lichtstrahlen, für jeden möglichen Einfallswinkel ( $\theta, \phi$ ), für jede Wellenlänge  $\lambda$ , zu jedem Zeitpunkt  $t$ , die durch das Zentrum der Pupille gehen. Die Winkel ( $\theta, \phi$ ) können immer relativ zu einer optischen Achse, die parallel zur  $V_z$ -Achse liegt, beschrieben werden. Die resultierende Funktion hat die Form:  

$$p = P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$$
.

- Die plenoptische Funktion beschreibt alle möglichen Bilder, die von einem bestimmten Raum-Zeit-Abschnitt der Welt aufgenommen werden können.

## 5. Camera Obscura / Lochkamera

- Einfachstes Kameraprinzip.
- war die erste Spycam
- Funktionsweise: Lichtstrahlen werden durch eine kleine Öffnung auf eine Bildebene projiziert.
- Ergebnis: Verkleinertes, seitenverkehrtes Abbild.
- Diente als Grundlage für die Entwicklung der Fotografie.

Die Lochkamera besteht aus einer geschlossenen Box mit einer winzigen Öffnung an der Vorderseite ("Pinhole") und der Bildebene an der gegenüberliegenden Rückseite (siehe Abbildung 5). Lichtstrahlen, die von einem Objektpunkt vor der Kamera ausgehend durch die Öffnung einfallen, werden geradlinig auf die Bildebene projiziert, wodurch ein verkleinertes und seiterverkehrtes Abbild der sichtbaren Szene entsteht. Durch den Einsatz von Spiegeln kann das projizierte Bild in die richtige Position gedreht werden. Eine portable Version der Lochkamera stellte eine Kiste mit einem angewinkelten Spiegel dar, mit denen ein Bild in richtiger Ausrichtung auf ein durchscheinendes Papier, welches auf einer Glasoberfläche liegt, projiziert wird. Die Lochkamera kann als einfache Kamera ohne Linse und einer einzigen, kleinen Apertur (Öffnung) angesehen werden.

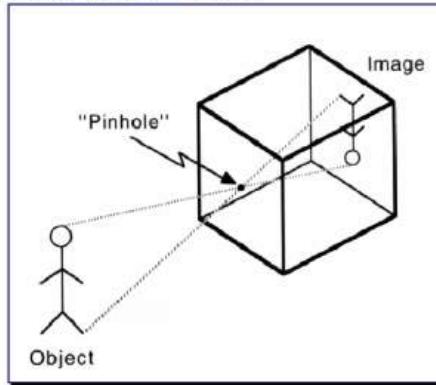


Abbildung 5: Modell der *pinhole camera*



## 2. Graphikpipeline und Objektrepräsentationen

### Graphikpipeline:

Quelle: [EVC\\_Skriptum\\_CG](#), p.7

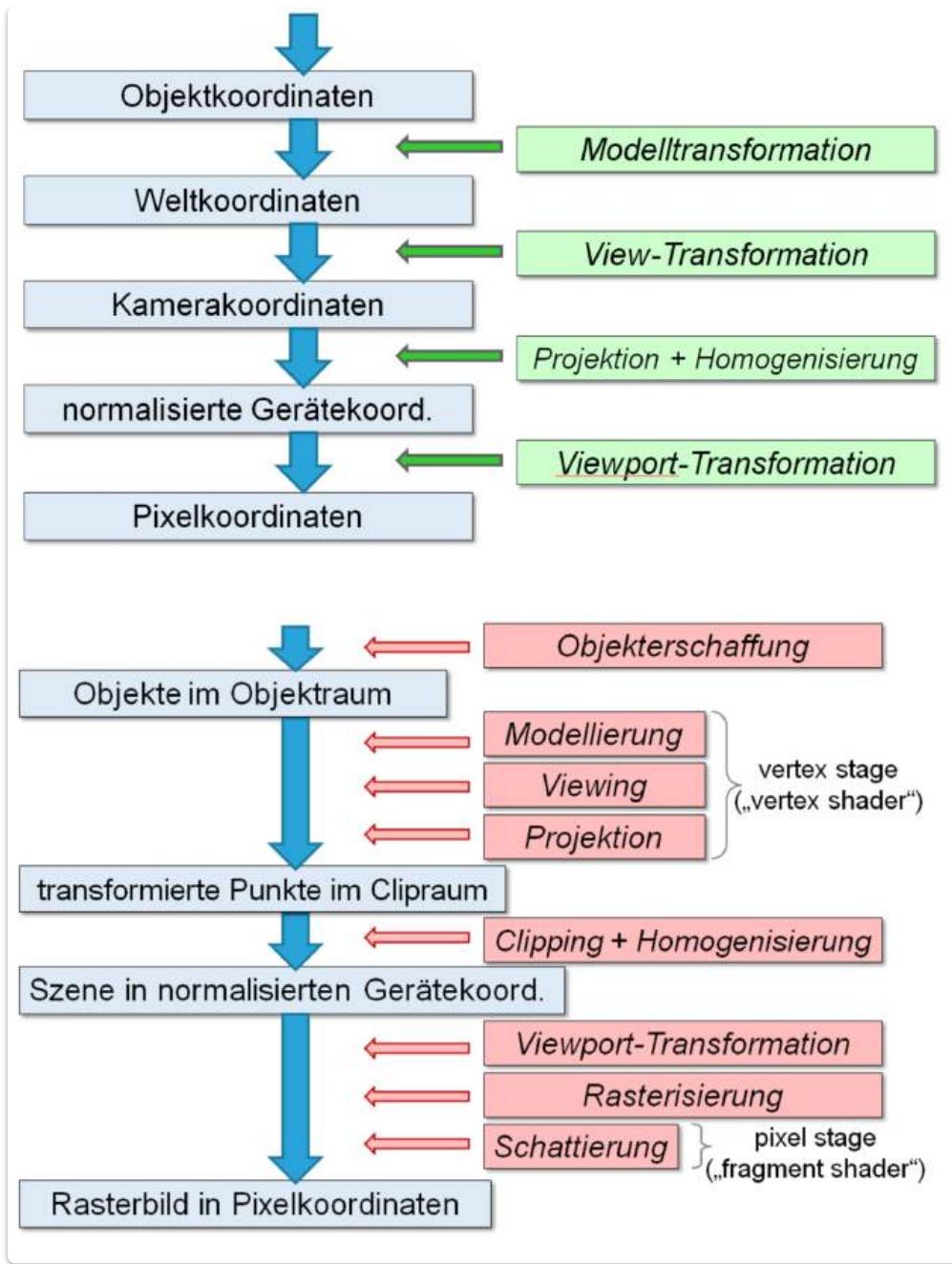
Informationen werden in aufeinanderfolgenden Schritten in ein Bild transformiert. Diese Abfolge wird als **Graphikpipeline** bezeichnet und kann je nach Fokus auch **Viewing-Pipeline**, **Transformationspipeline** oder **Rendering-Pipeline** genannt werden.

#### Schritte der Graphikpipeline:

1. **Objekt- und Szenenbeschreibung:** Die Objekte und ihre Anordnung in der Szene werden auf irgendeine Weise definiert. Die Objekterschaffung kann durch **Modellierung** oder **Scanning** erfolgen.
2. **Festlegung der Blickrichtung etc.:** Die Kameraparameter und die gewünschte Perspektive werden bestimmt.
3. **Projektion der Objekte:** Die 3D-Objekte werden auf eine 2D-Ebene projiziert.
4. **Umwandlung in Rasterpunkte:** Das projizierte Ergebnis wird in Pixel umgewandelt, um auf einem Bildschirm dargestellt zu werden.

#### Zusätzliche Informationen:

- **Vertex Shader und Fragment Shader:** Dies sind programmierbare Teile von Graphikkarten, die in der Pipeline eine Rolle spielen.
- **Grundprinzip:** Es existieren viele ähnliche Darstellungen der Graphikpipeline, die alle dasselbe grundlegende Prinzip beschreiben.
- **Einordnung weiterer Kapitel:** Die meisten weiteren Themen in diesem Skriptum zur Graphik können direkt in dieses Schema der Graphikpipeline eingeordnet werden.

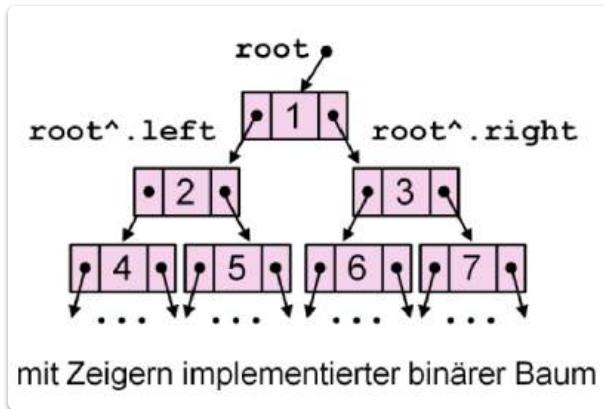
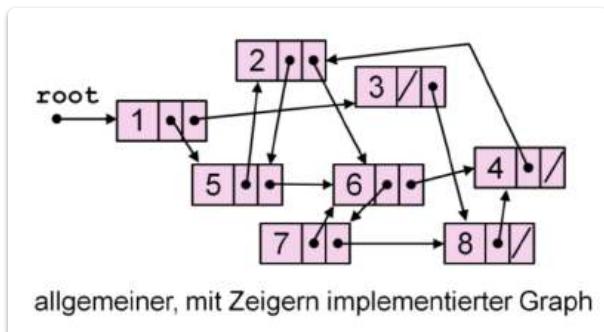


## Einschub: Graphen und Bäume

Quelle: [EVC\\_Skriptum\\_CG](#), p.7

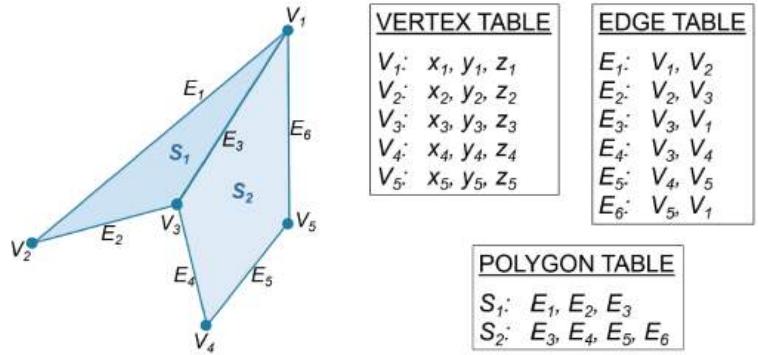
- **Repräsentation:** Ähnlich verketteten Listen können Graphen und speziell Bäume durch **zeigerverkettete Strukturen** dargestellt werden.
- **Knotenstruktur:** Einzelne Knoten benötigen ausreichend **Zeigerkomponenten**, um die gewünschte Struktur abzubilden (z.B. zwei Zeiger für binäre Bäume).
- **Operationen:** Einfügen und Entfernen von Knoten analog zu verketteten Listen.
- **Bearbeitungsreihenfolge:** Steuerung durch Aufrufe geeigneter Nachfolger.
- **Rekursive Abarbeitung binärer Bäume (Beispiele):**
  - **Pre-order:** Wurzel → linker Nachfolger → rechter Nachfolger (Beispielhafte Ausgabe: 1245367)

- **In-order:** linker Nachfolger → Wurzel → rechter Nachfolger (Beispielhafte Ausgabe: 4251637)
- **Post-order:** linker Nachfolger → rechter Nachfolger → Wurzel ( $\rightarrow 4526731$ )



## Polygon-Listen (B-Reps)

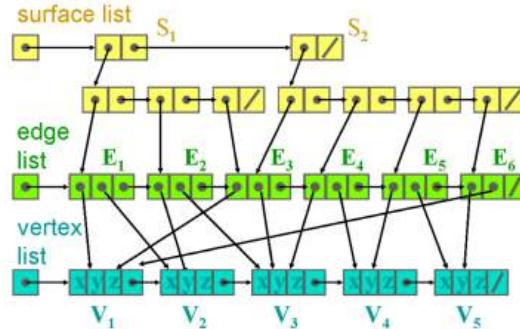
Dreidimensionale Objekte werden meist durch Polygonlisten repräsentiert (oft Dreiecke). Eine Menge von Polygone, die die Oberfläche eines Objektes beschreibt, nennt man *Boundary-Representation* („B-Rep“). Datenstrukturen für B-Reps enthalten neben geometrischer Information auch Attribute (Eigenschaften). Die Geometrie besteht aus Punktlisten, Kantenlisten, Flächenlisten und muss auf Konsistenz und Vollständigkeit überprüft werden.



Dieselbe Struktur lässt sich auch mit Zeigerlisten darstellen (siehe Abbildung rechts).

Die gesamte Koordinateninformation befindet sich in den **Punktknoten** ( $V$  steht für **Vertex**). Wenn ein Punkt an einen neuen Ort transformiert wird, so reicht es, die Koordinaten dieses Punktes zu verändern. Die Topologie bleibt dabei erhalten. Die lineare Verkettung der Kanten und Punkte erleichtert die Bearbeitung (z.B. alle Kanten einmal zeichnen, oder alle Punkte verschieben).

Das Beispiel links zeigt für eine ganz einfache Situation mit 2 Polygone, wie die Punktliste (**Vertex Table**), Kantenliste (**Edge Table**) und die Flächenliste (**Surface Table**) sich gegenseitig referenzieren. Nur in der **Punktliste** ist die tatsächliche **geometrische Information enthalten**, die anderen Listen beschreiben lediglich die Topologie.



## Wichtig!

- Das heißt der einzige Ort, wo Tatsächlich Koordinaten gespeichert sind, ist die Vertex Tabelle
- Die anderen Tabellen sind jeweils nur Referenzen auf sich gegenseitig und die VT
- Somit ist das anpassen verschiedener Attribute wie Position einfach durchzusetzen, da man nur in der jeweiligen Tabelle was ändern muss und sich der Rest anpasst.
- Außerdem hatten wir hier dann noch einen kleinen Wiederholung von [Vektoren und ihre Produkte \(Aus EVC\)](#) mehr dazu auch noch hier [7. Vektoren](#) und [8. Matrizen](#)

## Wichtige Begriffe

Quelle: [EVC\\_Skriptum\(CG\)](#), p.8

- Polygonfläche:**
  - Repräsentation beinhaltet die **Trägerebene** (definiert durch die Gleichung  $Ax + By + Cz + D = 0$ ) und die zugehörigen **Eckpunkte** ( $V_1$  bis  $V_n$ ).
  - Aus den Ebenenparametern ( $A, B, C, D$ ) lässt sich direkt der **Normalvektor der Ebene** ( $A, B, C$ ) ableiten.
- Backface:** Die Rückseite eines Polygons, die ins Innere des Objekts zeigt.
- Frontface:** Die Vorderseite eines Polygons, die die Außenseite des Objekts bildet.

Wenn man ein rechtshändiges Koordinatensystem vorausschickt und die Eckpunkte jedes Polygons (von vorne betrachtet) im mathematisch positiven Sinn (also gegen den Uhrzeigersinn) anordnet, dann gilt für einen Punkt  $(x, y, z)$ :

- wenn  $Ax + By + Cz + D = 0$  dann liegt der Punkt **auf** der Ebene
- wenn  $Ax + By + Cz + D < 0$  dann liegt der Punkt **hinter** der Ebene
- wenn  $Ax + By + Cz + D > 0$  dann liegt der Punkt **vor** der Ebene

- **Berechnung des nach außen gerichteten Normalvektors:** Aus drei aufeinanderfolgenden Eckpunkten ( $V_1, V_2, V_3$ ) eines Polygons (bei rechtshändigem Koordinatensystem und gegen den Uhrzeigersinn angeordnet) kann der nach außen gerichtete Normalvektor  $\mathbf{N}$  berechnet werden durch die Formel:  $N = (V_2 - V_1) \times (V_3 - V_1)$
- **Dreiecke als Polygone:** Sehr häufig verwendet, da sie Algorithmen vereinfachen (z.B. sind Dreiecke immer eben).
- **Dreiecks-Mesh:** Eine Datenstruktur, die ausschließlich aus Dreiecken besteht.
- **Dreiecks-Strip:** Eine lineare Abfolge von Dreiecken, bei der jedes weitere Dreieck durch Angabe nur eines neuen Punktes definiert wird (die vorherige Kante wird beibehalten).

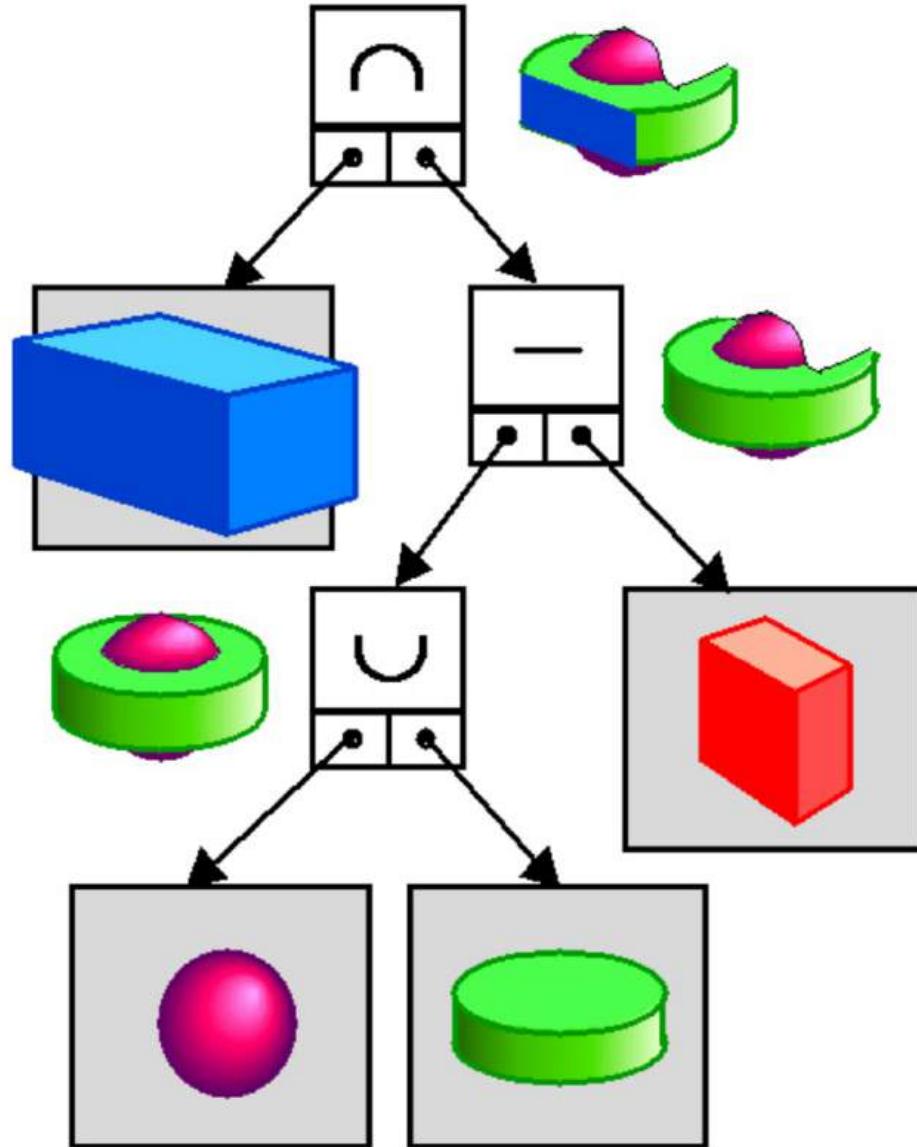
## Constructive Solid Geometry (CSG)

### Quellen:

- EVC\_Skriptum\_CG, p.8
- EVC\_Skriptum\_CG, p.9
- **Konstruktion:** Objekte werden durch **Mengenoperationen** (Vereinigung, Durchschnitt, Differenz) auf **dreedimensionale Primitive** (Kugel, Tetraeder, Würfel, Zylinder etc.) konstruiert.
- **Datenstruktur:** Anordnung in einer **hierarchischen Datenstruktur**, dem sogenannten **CSG-Baum** (eigentlich ein kreisfreier Graph).
- **Konsistenz:** CSG-Objekte sind immer **konsistent** (keine Löcher, wohldefiniertes Inneres), da Primitive trivialerweise konsistent sind und die Mengenoperationen Konsistenz bewahren.
- **Knoteninformation:** Jeder Knoten im CSG-Baum enthält zusätzlich **Transformationen (Matrizen)**, die auf den darunterliegenden Teilbaum angewendet werden.
- **Vorteile:**
  - **Exakte Repräsentation:** Primitive behalten ihre exakte geometrische Form (z.B. eine perfekte Kugel).
  - **Geringer Speicherbedarf.**
  - **Einfache Transformationen.**
- **Nachteile:**
  - **Aufwändiges Rendering:** Komplizierte Berechnung von Bildern.
  - **Lösungsmöglichkeiten für Rendering:**

- Umwandlung der CSG-Struktur in eine **B-Rep-Repräsentation** und herkömmliches Rendering.
- Direkte Bilderstellung mittels **Ray-Casting** oder **Ray-Tracing**.

Hier ein Beispiel:

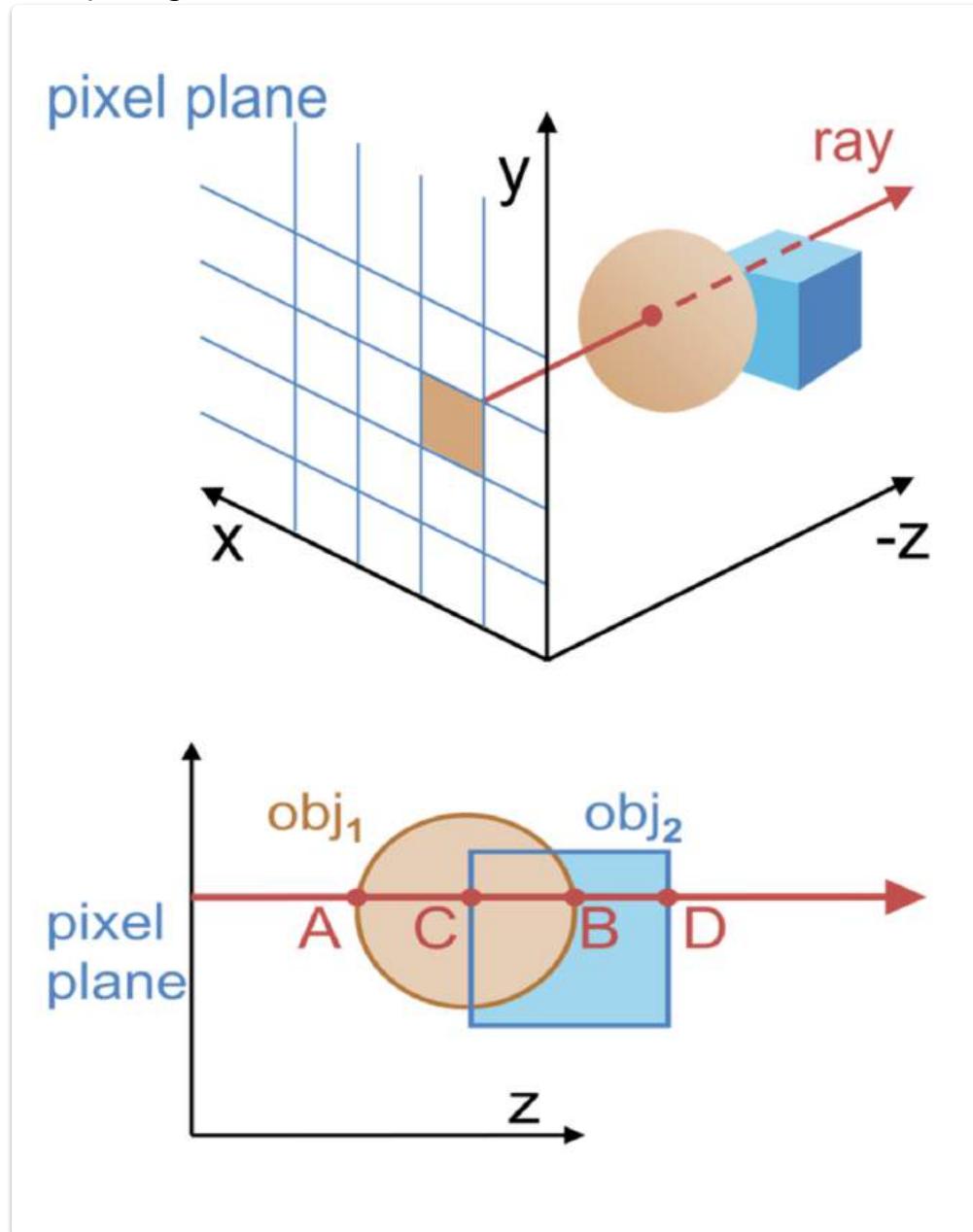


## Ray-Casting von CSG-Objekten

EVC\_Skriptum\_CG, p.9

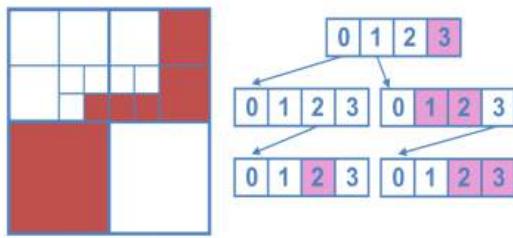
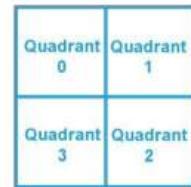
- **Methode:** Pixelweise Bildberechnung. Für jedes Pixel wird ein **Strahl (Ray)** in Blickrichtung ausgesendet und mit allen Objekten geschnitten. Der **vorderste Schnittpunkt** bestimmt das sichtbare Objekt und dessen Farbe für das Pixel.
- **Berechnung im CSG-Baum (rekursiv):**
  - **Endknoten (Primitive):** Berechnung aller Schnittpunkte ist einfach.
  - **Zwischenknoten (Operationen):** Verknüpfung der Schnittpunktlisten der beiden Nachfolger entsprechend dem Operator:

- **Vereinigung (Union):** Aus Listen (A,B) und (C,D) entsteht (A,D).
- **Durchschnitt (Intersection):** Aus Listen (A,B) und (C,D) entsteht (C,B).
- **Differenz (Difference):** Aus Listen (A,B) und (C,D) entsteht (A,C).
- **Wurzel:** Der **erste Punkt** der resultierenden verknüpften Schnittpunktliste wird ausgewählt.
- **Ray-Tracing:** Eine fortgeschrittenere Technik, die Ray-Casting als Teilmenge beinhaltet und später genauer behandelt wird.

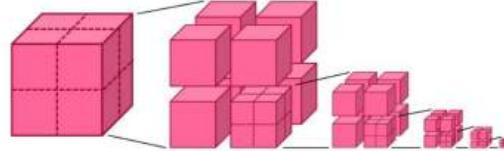


## Quadtrees und Octrees

Ein **Quadtree** ist eine Datenstruktur, die zur Repräsentation beliebiger zweidimensionaler Strukturen geeignet ist. Der relevante Bereich wird überall dort in vier Viertel geteilt, wo die Information noch zu kompliziert ist um einfach abgelegt zu werden, andernfalls wird die einfache Information abgelegt. Jedem Bildbereich entspricht ein Knoten eines Baumes, in dem jeder Knoten (maximal) vier Nachfolger hat („Quadtree“).



Das nebenstehende Beispiel zeigt einen **einfachen Quadtree**, der eine zweifarbige einfache Graphik repräsentiert. Der Wurzelknoten entspricht dem ganzen Bild, die Knoten in der zweiten Reihe entsprechen den oberen zwei Vierteln des Bildes und die letzten zwei Knoten entsprechen den zwei Bereichen, die am feinsten aufgelöst sind.



Ein **Octree** ist die Erweiterung dieses Konzeptes auf **drei Dimensionen**. Ein beliebig geformtes Objekt (oder auch eine ganze Szene) innerhalb eines Würfels wird dadurch repräsentiert, dass „einfache“ Teilwürfel (leer

## Grundprinzip

- Ein **Octree** ist die Erweiterung des Quadtrees auf **drei Dimensionen**.
- Der Raum (z.B. eine Szene oder ein Objekt) wird rekursiv in **acht Teilwürfel (Oktanten)** unterteilt.
- Jeder Knoten im Baum hat **acht Nachfolger**.
- Die Unterteilung wird fortgesetzt, bis:
  - der Teilwürfel **einfach** ist (komplett leer oder vollständig im Objekt),
  - oder eine festgelegte **Mindestgröße** erreicht wurde (z.B. ein Tausendstel der Gesamtausdehnung).

## Aufbau des Baums

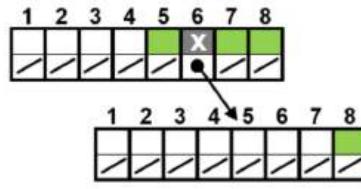
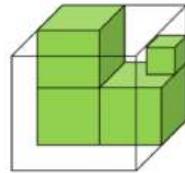
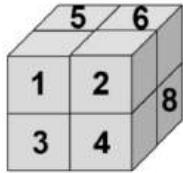
- **Einfache Teilwürfel** werden als **Blätter (Endknoten)** gespeichert.
- **Komplexe Teilwürfel** werden weiter in acht kleinere Würfel unterteilt.
- Die Struktur ergibt einen **rekursiven Baum**, in dem jeder Knoten bis zu acht Kinder hat.

## Vorteile

- Kann **beliebige 3D-Formen** repräsentieren.
- **Schnelle Abfragen** möglich (z.B. Inhalt an einem bestimmten Punkt).
- **Mengenoperationen** sind einfach umzusetzen durch die rekursive Struktur.

## Nachteile

- **Unpräzise Repräsentation** bei komplexen oder schrägen Oberflächen.
- **Hoher Speicherbedarf** bei feiner Auflösung.
- **Transformationen** (z.B. Drehungen) sind aufwändig, oft muss der Octree neu aufgebaut werden.



Linearisierung: X(EEEESX(EEEEEEES)SS)  
E ... Empty, S ... Solid, X ... Mixed

## Rendering (Darstellung)

- Ablauf bei Verwendung eines speicherbasierten Renderings:

```
if Knoten ist einfach
    then zeichne Knoten #d.h. tue nichts wenn Knoten leer ist
else rekursiver Aufruf der 8 Oktanten von hinten nach vorne
```

Quelle:

[EVC\\_Skriptum\(CG\)](#), p.10

# Szenengraphen

Quelle: [EVC\\_Skriptum\(CG\)](#), p.10

### Begriff:

- Objektorientierte Datenstruktur.
- Hierarchische Beschreibung der logischen und/oder räumlichen Anordnung von Elementen in 2D/3D-Szenen.
- Oberbegriff für verschiedene hierarchische Beschreibungsformen graphischer Objekte.

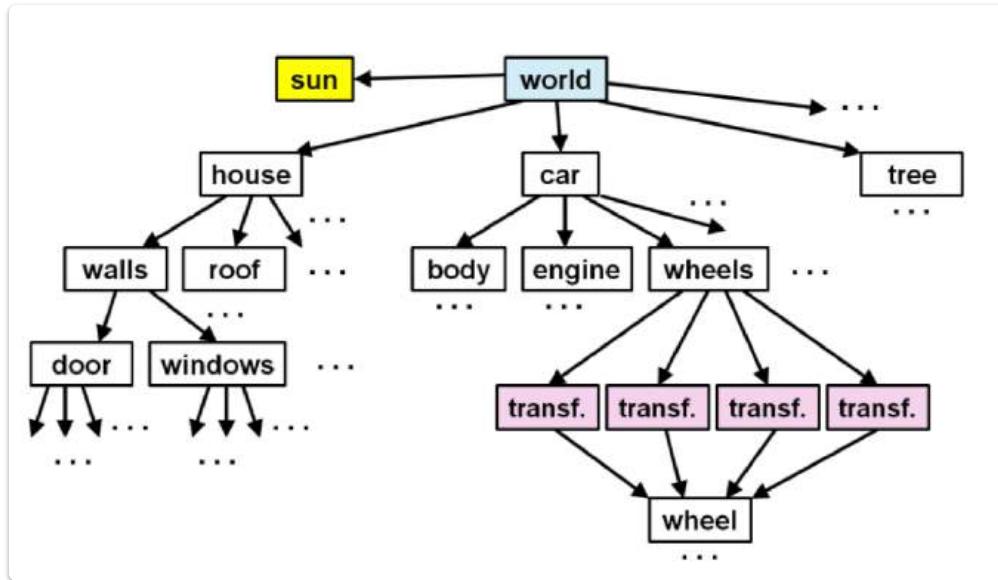
### Struktur:

- Graphentheoretisch: Baumähnlicher gerichteter kreisfreier Graph.
- Wurzelknoten:** Repräsentiert die gesamte Szene.
- Zwischenknoten:** Beschreiben Teilszenen (Wurzeln von Teilbäumen).
- Endknoten:** Repräsentieren die einfachsten Objekte der Szene (unterschiedliche Repräsentationen möglich).

### Beispiel (Stadt):

- Wurzel:** Stadt
- Zwischenknoten:** Haus, Fenster

- **Endknoten:** Fensterscheibe, Schraube



### Vorteile/Konzepte:

- **Mehrfachnutzung:** Wiederkehrende Elemente können im Graphen mehrfach referenziert werden.
- **Transformationen:** Zwischenknoten enthalten Informationen für den gesamten Teilgraphen:
  - Material, Farbe
  - Position, Lage im Raum
  - Größe
  - Verzerrung
- **Matrixdarstellung:** Geometrische Transformationen können elegant in Matrizen gespeichert werden.

### Relevanz:

- Zentrales Konzept in Systemen wie OpenSceneGraph, VRML und X3D.

## Andere Objektrepräsentationen

- es gibt noch viele andere Objektrepräsentationen und Datenstrukturen
- Oft für sehr spezielle Anwendungen

### Beispiele:

- **BSP-Bäume:**
  - Effiziente Kollisionserkennung.
  - Darstellung von komplexen Szenen.
- **Fraktale:**

- Naturnahe Formen (Landschaften, Wolken, Bäume).
- Kunst und Design.
- **Prozedurale Modelle:**
  - Generierung von Landschaften, Gebäuden, Pflanzen.
- **Partikelsysteme:**
  - Effekte wie Rauch, Feuer, Wasser, Explosionen.
- **Physikalisch basierte Modelle:**
  - Realistische Simulation von Objekten und Umgebungen.
- **3D-Volumendaten:**
  - Medizinische Bildgebung (CT, MRT).
  - Geologische Daten.

Quelle: [EVC\\_Skriptum\\_CG](#), p.10

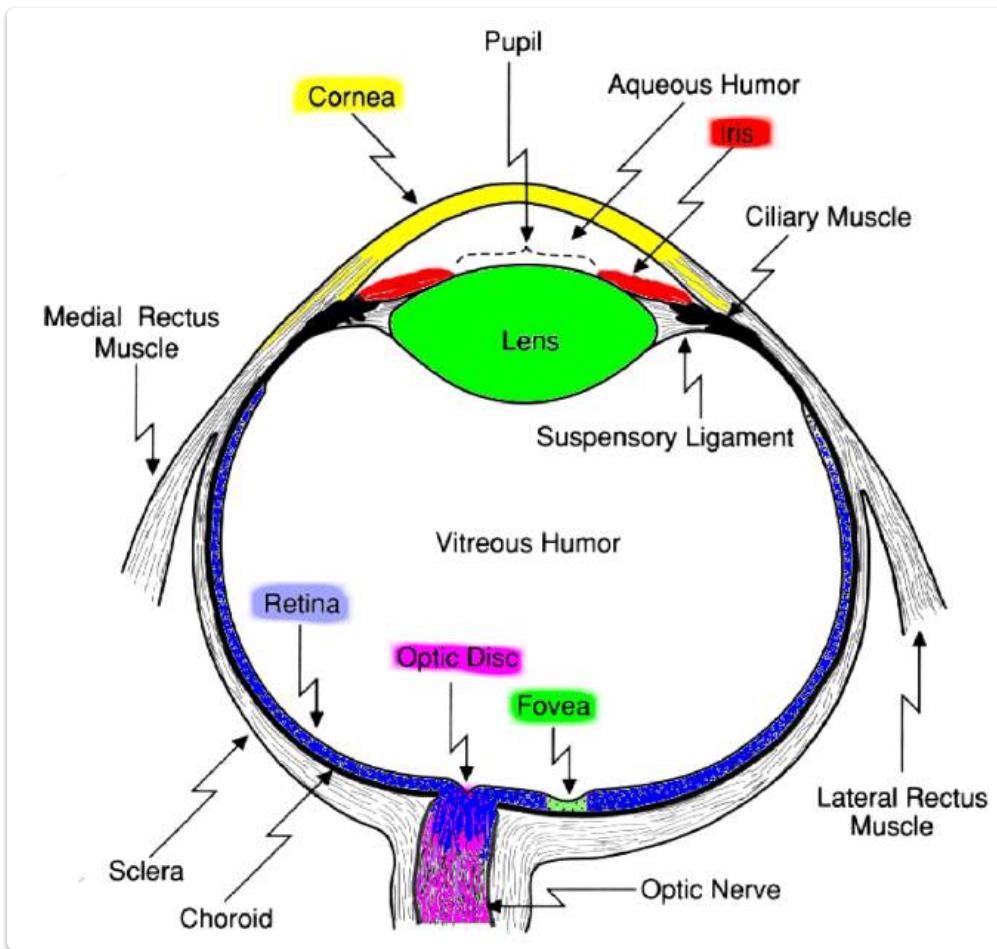


## 2. Bildaufnahme

Quellen:

- EVC\_Skriptum\_CV, p.8 bis EVC\_Skriptum\_CV, p.14

## Das Menschliche Auge:



- **Linse** des Auges fokussiert Licht aus der Umgebung
- Bild wird **seitenverkehrt & auf dem Kopf stehend** auf die Retina projiziert
- **Retina (Netzhaut)** = lichtempfindliche Membran auf der Rückseite des Auges

### Fotorezeptoren

- Spezialzellen, die Lichtreize in **neuronale Impulse** umwandeln
- Zwei Typen:
  - **Stäbchen**
    - Verantwortlich für **Schwarz-Weiß-Sehen**
    - Hohe Lichtempfindlichkeit → wichtig für das **Dämmerungssehen**

- **Zäpfchen**

- Zuständig für **Farbsehen**
- Funktionieren bei **Tageslicht**
- Drei Typen für unterschiedliche **Wellenlängenbereiche** (rot, grün, blau)

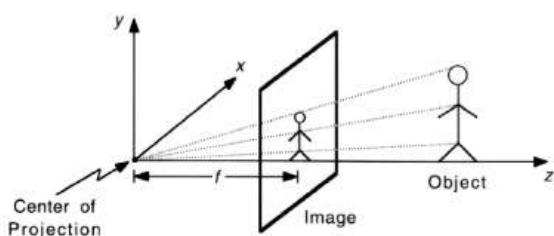
## Signalverarbeitung

- Reaktion der Fotorezeptoren: **chemisch-elektrisch**
- Entstehung eines **neuronalen Impulses**
- Weiterleitung über mehrere **Neuronenschichten**
- Schichten sind durch **Synapsen** miteinander verbunden

## Besondere Strukturen

- **Fovea centralis** (im Zentrum des gelben Flecks / *Macula lutea*):
  - Ort des **schärfsten Sehens**
  - Dichteste Anordnung von Zäpfchen
- **Sehnerv (Nervus opticus)**:
  - Leitet visuelle Impulse ans Gehirn
  - **50 % der Fasern**: Informationen aus der **Fovea**
  - **50 % der Fasern**: Informationen aus der **restlichen Retina**

## Perspektivische Projektion



## Grundprinzip

- Einfachstes Modell der Bildaufnahme ohne Linse, nur mit punktförmiger Blende
- Beschreibt perspektivische Projektion eines 3D-Punktes auf eine 2D-Bildecke
- Idealisiert: keine geometrischen Verzerrungen, keine Unschärfe durch Linsen

## Annahmen (zur Vereinfachung der Herleitung)

- Zentrum der Projektion (Punkt O) = Ursprung des Kamerakoordinatensystems
- Optische Achse (Kamaraachse) ist entlang der z-Achse der Kamera ausgerichtet

- Bildebene liegt vor dem Zentrum der Projektion ( $\rightarrow$  keine Bildumkehrung)

## Aufbau des Modells

- $O$  = Zentrum der Projektion (Punkt, durch den alle Lichtstrahlen verlaufen)
- Bildebene = Fläche, auf die das 3D-Bild projiziert wird
- $f$  = Brennweite (*Abstand von  $O$  zur Bildebene*) (*Bei Kameras: "Abstand zwischen Linse und Chip/Film"*)
- Optische Achse = Gerade durch  $O$ , senkrecht zur Bildebene

## Mathematische Beziehung (perspektivische Projektion)

- Gegeben: 3D-Punkt  $P = (X, Y, Z)$
- Projektionsgleichungen:
  - $x = \frac{f}{Z} \cdot X$
  - $y = \frac{f}{Z} \cdot Y$
- $x$  und  $y$ : Position des projizierten Punktes auf der Bildebene

## Eigenschaften der Projektion

- Nicht-linear: Bildgröße  $\propto 1/Z \rightarrow$  je näher ein Objekt, desto größer erscheint es
- 1:n-Abbildung: mehrere 3D-Punkte können auf denselben 2D-Punkt projiziert werden
- 3D-Linien  $\rightarrow$  2D-Linien (außer wenn parallel zur optischen Achse)
- Winkel und Abstände bleiben nicht erhalten
- Parallelle Linien bleiben nur erhalten, wenn sie parallel zur Bildebene liegen
- Informationsverlust: 3D  $\rightarrow$  2D nicht umkehrbar; Tiefeninformation geht verloren

## Praktische Anwendung

- Vereinfachtes Modell zur Beschreibung von Bildaufnahmeprozessen
  - Grundlage in Computervision, Grafik und Kamerakalibrierung
  - Idealisierung realer Kameras zur mathematischen Beschreibung der Abbildungsgeometrie
- 

# Linsen

## Motivation

- Die einfache Lochkamera hat **praktisch keine Bedeutung** in der realen Bildaufnahme
- Für eine **scharfe Projektion** benötigt man ein sehr kleines Loch  $\rightarrow$  lässt kaum Licht durch
- Folge: **lange Belichtungszeiten**, ungeeignet für bewegte Szenen oder praktische Anwendungen

## Lösung: optische Linsen

- Statt Lochblende werden **Linsen oder Linsensysteme** verwendet
- Diese bieten bessere Lichtbündelung und schärfere Abbildung
- Abbildungsverhalten ist komplexer, aber wesentlich **praxisnäher**

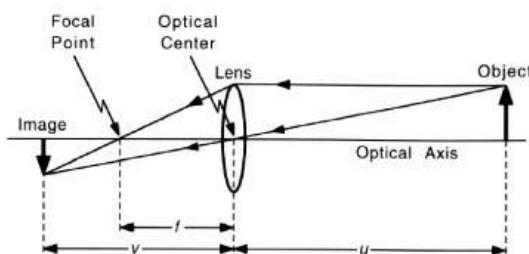
## Modell der dünnen Linse

- Vereinfachtes, idealisiertes Modell
- Ersetzt die Lochblende durch eine **symmetrische, unendlich dünne Linse**
- **Lichtstrahlen werden an einer virtuellen Ebene** (in der Linsenmitte) gebrochen
- Abbildungsgeometrie bleibt gleich wie bei der Lochkamera → perspektivische Projektion

## Brennpunktverhalten

- **Parallele Strahlen** zur optischen Achse werden im **Brennpunkt** (Abstand  $f$  zur Linse) gebündelt
- Die **Brennweite  $f$**  ist ein zentraler Parameter der Abbildung
- Fokuspunkt entsteht auf der Bildebene, wenn das Objekt korrekt positioniert ist

## Linsengleichung



$$\text{Thin-Lens Equation: } \frac{1}{u} + \frac{1}{v} = \frac{1}{f}$$

Abbildung 8: Linsengleichung

Liegt ein Objekt in einer endlichen Entfernung  $u$ , werden dessen **Lichtstrahlen** zu einem Bild hinter dem Brennpunkt in der Entfernung  $v$  fokussiert. Die zur Linsenachse senkrecht stehende Ebene wird **Bildebene** genannt. Die Beziehung zwischen der **Entfernung  $u$  vom Objekt zur Linse** und der **Entfernung  $v$  von der Linse zur Bildebene** wird durch die **einfache Linsengleichung** beschrieben, wie in Abbildung 8 dargestellt:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}.$$

Diese besagt, dass **Objekte, die sehr weit weg sind ( $u = \infty, v = f$ )**, im Brennpunkt scharf abgebildet sind, alle Objekte, die näher sind, dahinter.

## Tiefenschärfebereich (DOF (Depth of Field))

- Wegen einfacher Linsengleichung immer nur Objekte mit bestimmter Entfernung zur Kamera scharf

- alles andere unscharf
- In Praxis Objekte mit gewissen DOF scharf abgebildet.
- DOF = Entfernung zwischen dem nächsten und weitesten entfernten Objekt einer Szene, welches scharf dargestellt wird

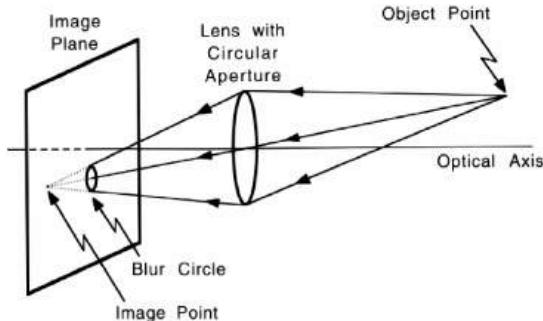
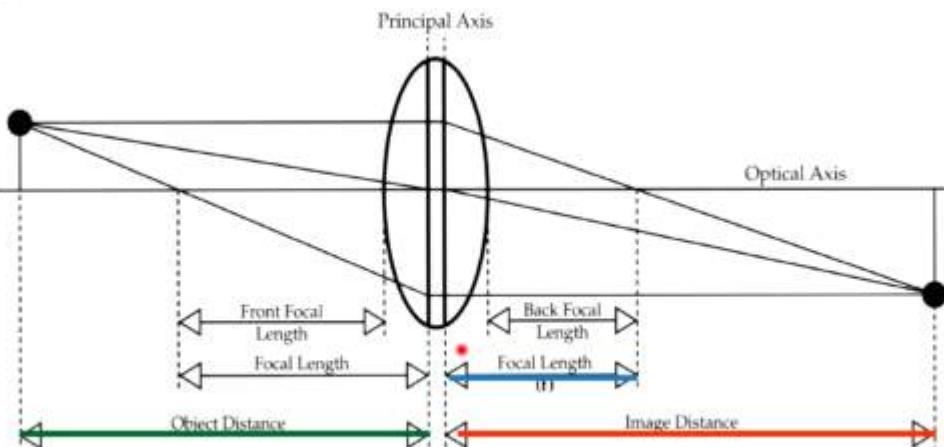


Abbildung 9: Tiefenschärfenbereich



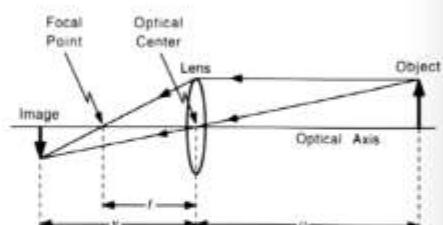
FOCAL LENGTH = Distance from focus point to principal axis.

Test-ähnliches Beispiel:

### Beispiel



- Eine Person steht 5m vor einer Kamera. Die fokale Länge der Linse beträgt 200mm. Wie muss der Bildabstand gewählt werden, damit die Person scharf auf der Bildebene erscheint?



$$\begin{aligned} \bullet u &= 5000 \text{ mm} & \frac{1}{u} + \frac{1}{v} &= \frac{1}{f} \Rightarrow \frac{1}{5000} + \frac{1}{v} &= \frac{1}{200} \Rightarrow \frac{1}{v} &= \frac{1}{200} - \frac{1}{5000} \Rightarrow \\ \bullet f &= 200 \text{ mm} & \frac{1}{v} &= \frac{24}{5000} \Rightarrow v &= \frac{5000}{24} \Rightarrow v &= 208,333 \\ \bullet v &=? & & & & \end{aligned}$$

### Einflussfaktoren

- **Sensorauflösung**
- **Größe der Linse / Blendenöffnung**
- **Brennweite der Linse**

## Blende und Lichtbündel

- Blende = ringförmige Öffnung vor der Linse
- Bestimmt den **Kegelwinkel** des Strahlenbündels, das auf die Bildebene trifft
- **Kleine Blende:**
  - Weniger Strahlen erreichen die Bildebene
  - Geringere Lichtmenge → dunkleres Bild
  - **Größerer DOF** (mehr vom Bild erscheint scharf)
- **Große Blende:**
  - Mehr Strahlen pro Objektpunkt → höhere Lichtausbeute
  - **Kleinerer DOF** (weniger Tiefenbereich, Bild weniger scharf)

## Unschärfekreise (Blur Circles)

- Wenn Objekt nicht im richtigen Abstand  $v$  zur Linse steht → Strahlen bündeln sich nicht exakt
- Statt eines Punktes entsteht ein **Unschärfekreis** auf der Bildebene
- Benachbarte Objektpunkte ergeben überlappende Kreise → **Bild wird unscharf**

## Zusammenhang: Blende – Blur – DOF

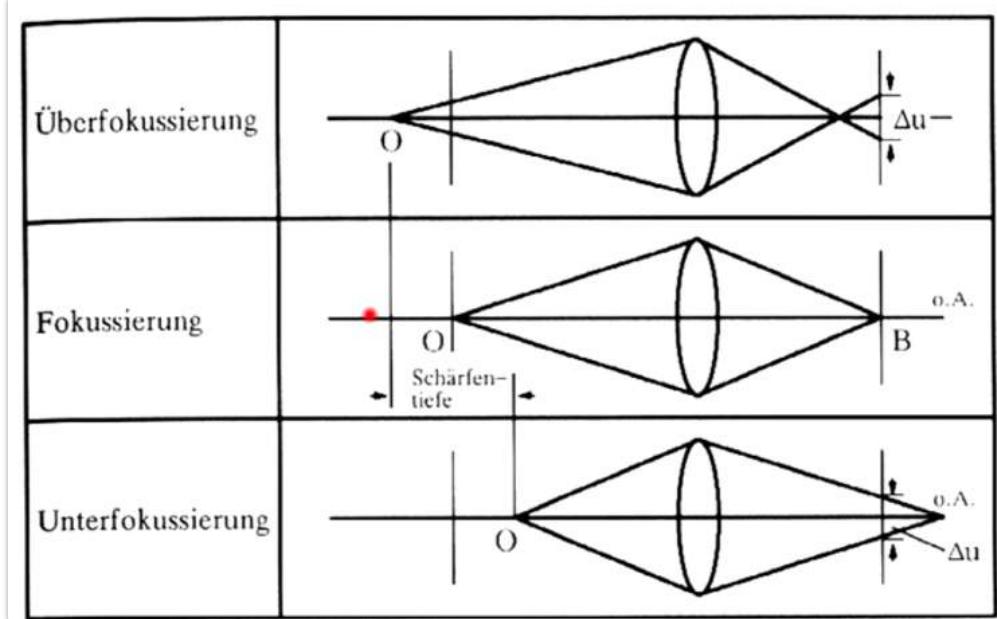
- **Größere Linse** = größere Unschärfekreise = **kleinerer DOF**
- **Kleinere Linse** = kleinere Unschärfekreise = **größerer DOF**

## Einfluss der Sensorauflösung

- **Höhere Auflösung:** Unschärfekreise können besser abgebildet werden → **kleinerer DOF**
- Geringere Auflösung: Unschärfen weniger deutlich → scheinbar größerer DOF

## Einfluss der Brennweite

- **Längere Brennweite (z. B. Teleobjektiv)** → **kleinerer DOF**
- **Kürzere Brennweite (z. B. Weitwinkelobjektiv)** → **größerer DOF**



# Radiometrie

## Grundlagen

- Radiometrie = **Messung elektromagnetischer Strahlung**, inkl. sichtbarem Licht
- Ziel: Beschreibung der Lichtmenge, die von der realen Welt auf ein Bild projiziert wird

## Strahldichte (Radiance)

- Beschreibt die **Menge des reflektierten Lichtes** eines Oberflächenpunktes
- Richtungsabhängige Größe
- Gibt an, wie viel Licht in eine bestimmte Richtung von einem Punkt abgestrahlt wird
- Entspricht der **Lichtmenge, die durch ein kleines Raumwinkel-Element ausgesendet wird**

## Bestrahlungsstärke (Irradiance)

- Gibt an, **wie viel Licht** von einer Oberfläche **empfangen** wird
- Entspricht der Helligkeit eines Bildpunktes
- Wird aus der Strahldichte berechnet, indem über alle Richtungen integriert wird

## Spektrum

- Radiometrische Größen beziehen sich auf **Strahlung aller Frequenzen**
- Betrachtet man **einzelne Frequenzbereiche**, spricht man vom **Spektrum**
- Wichtige spektrale Größe:
  - **Spektrale Bestrahlungsstärke (spectral irradiance)**
  - Gibt an, **wie viel Strahlungsenergie pro Wellenlänge** auf eine Fläche trifft

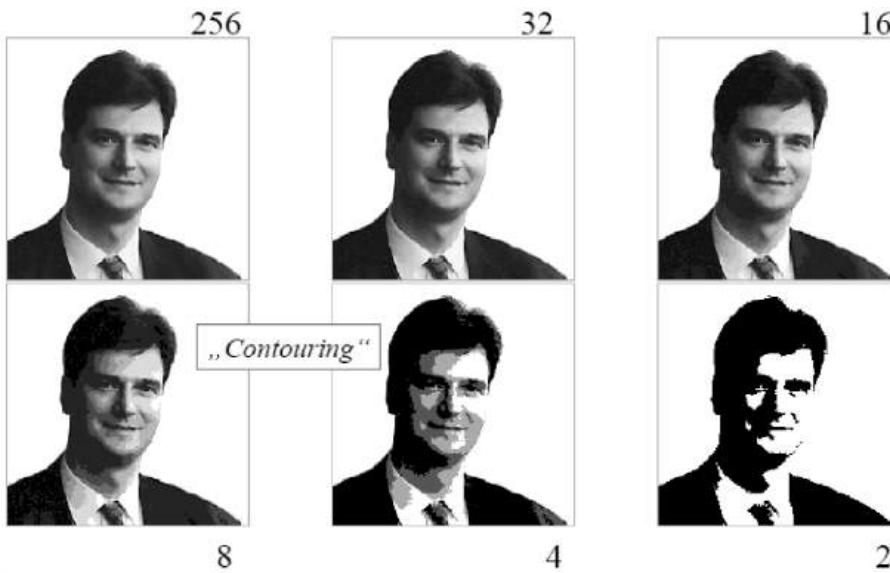
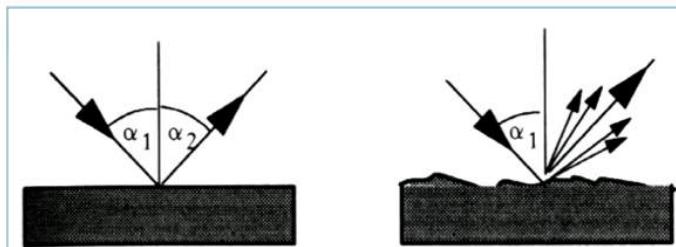


Abbildung 10: Einfluss der Bitanzahl auf die Bildqualität

The radiometric relation between the world and its projection is formed by:

- Amount of light that is reflected by a surface point = **Radiance**
- Amount of light that is projected from this point onto the image = **Irradiance**
- measured in watts per square meter ( $W/m^2$ ),



## Sampling

### Bildauflösung – Überblick

- **Bildauflösung** = Maß für den **Detailreichtum** eines Bildes
- Abhängig von:
  - **Radiometrischer Auflösung**
  - **Sensorauflösung**
  - **Räumlicher (geometrischer) Auflösung**
  - **Zeitlicher Auflösung**

### Radiometrische Auflösung

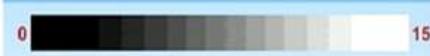
- Gibt an, wie fein ein System **Helligkeitsunterschiede unterscheiden** kann
- Beschrieben durch **Anzahl der Bits** (z. B. 8 Bit = 256 Graustufen)

- Mensch kann ~120 Grauwerte unterscheiden → 256 Graustufen genügen

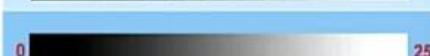
$$2^1 = 2 \text{ levels (0,1)}$$



$$2^2 = 4 \text{ levels (0,1,2,3)}$$



$$2^8 = 256 \text{ levels (0-255)}$$



$$2^{12} = 4096 \text{ levels (0-4095)}$$

- Weniger Graustufen → geringerer Speicherbedarf, aber:

- Qualitätsverlust

- Contouring-Effekte (sichtbare Tonwertsprünge)

(siehe hier:

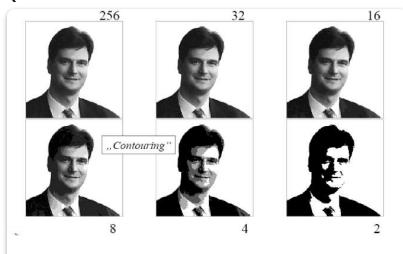


Abbildung 10: Einfluss der Bitanzahl auf die Bildqualität

)

## Sensorauflösung

- Anzahl der **Pixel (Bildpunkte)** eines digitalen Bildes
  - z. B.  $640 \times 480 = 307.200$  Pixel = ~0.3 MP
- Häufige Angabe: **Megapixel**
- Alternativ: **dpi / ppi** (dots/pixels per inch) – sinnvoll nur bei Scannern
- Für Kameras ungeeignet, da Objekt-Sensor-Abstand variabel

## Räumliche (geometrische) Auflösung

- Fähigkeit, **benachbarte Objektstrukturen trennt** zu erfassen
- Maßeinheiten:
  - Linien pro Millimeter (L/mm)** in Optik/Photographie
  - Definiert durch:
    - minimale Entfernung** zwischen zwei erkennbaren Merkmalen
    - minimale Größe** eines erkennbaren Merkmals

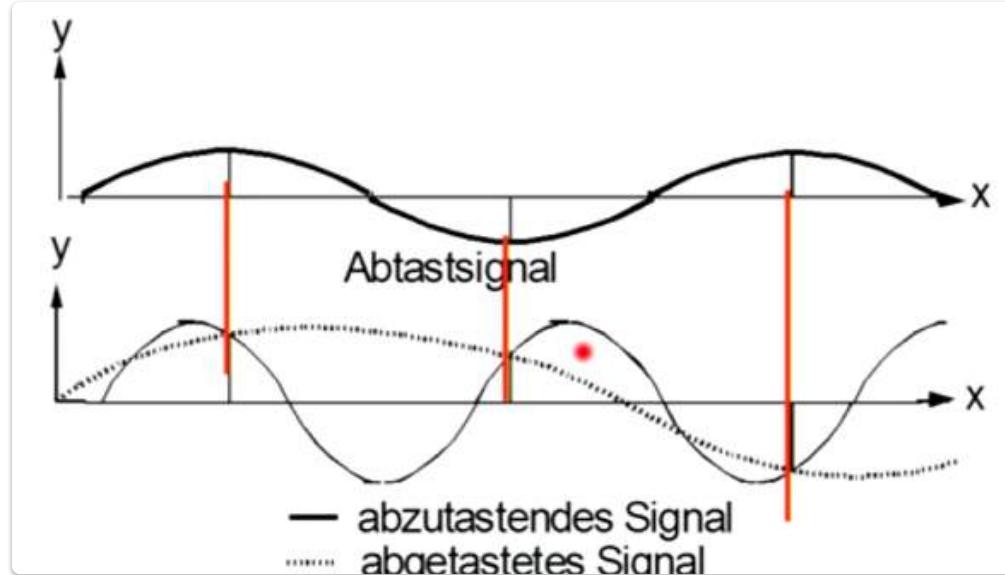
## Zeitliche Auflösung

- Anzahl der aufgenommenen Bilder **pro Sekunde**
  - z. B. 25 fps (frames per second)
- Wichtig bei Videoaufnahmen oder Bewegungsanalysen

## Sampling (Abtastung)

- Umwandlung** eines kontinuierlichen Signals (z. B. in Raum/Zeit) in eine **diskrete Folge**

- Zentrale Grundlage: **Nyquist–Shannon Sampling Theorem**
  - Ein bandbegrenztes Signal kann exakt rekonstruiert werden, wenn:
    - **Samplingrate >  $2 \cdot B$**  ( $B$  = höchste Frequenz im Signal)
    - d.h.  $f_{abtast} > 2 \cdot f_{maxSignal}$



- Anwendung bei **analoger/digitaler Wandlung** in digitalen Kameras:
  - Raum (Pixelraster)
  - Zeit (fps)
  - Spektrum (z.B. Farbbandsampling)

mehr Beispiele zu Nyquist: [7. Clipping und Antialiasing](#)

## Bildsensoren

### Allgemeines

- Ein **Bildsensor** wandelt ein **optisches Bild** in ein **elektronisches Signal** um
- Es gibt zwei Arten digitaler Pixelsensoren:
  - **CCD (Charge-Coupled Device)**
  - **CMOS (Complementary Metal-Oxide-Semiconductor)**

### CCD-Sensoren

- Bestehen aus einer **Matrix lichtempfindlicher Fotodioden**
  - Zusätzlich enthalten: **MOS-Kondensatoren, Feldeffekttransistoren, Steuerleitungen, Leitungspfade**
- **Belichtungszeit**: Licht erzeugt elektrische Ladung in der Fotodiode
- Ladung wird im Kondensator gespeichert (parallel zur Fotodiode geschaltet)
- Auslesung erfolgt **zeilenweise**
- Spannung wird in der Kamera oder extern in **digitale Information** umgewandelt

- Arbeitet **linear**, wie analoge Kameras → basiert auf **Flächenintegral-Prinzip**

## CMOS-Sensoren

- Besitzen **aktive Pixelsensoren**, jedes Pixel ist individuell auslesbar
  - Spannung, die der **Helligkeit** entspricht, liegt **ständig an**
  - Output ist **nicht-linear (logarithmisch)**
    - Ähnelt dem Prinzip der **menschlichen Bildwahrnehmung**
  - Vorteile gegenüber CCD:
    - **Niedriger Energieverbrauch**
    - **Schnellere Lesegeschwindigkeit**
    - **Günstigere Herstellung**
- 

## Farbe auf CCD-Sensoren

### Allgemeines

- **Bildsensoren** erfassen **nur Helligkeit**, nicht direkt Farbe
- Um **Farbinformation** zu erhalten, werden **Farbfilter** eingesetzt
- Drei gängige Methoden zur Farberfassung:
  - **Field Sequential Technik**
  - **3-Chip-Technik**
  - **Color Filter Array (CFA)**

### Field Sequential Technik

- Ursprünglich 1900er Jahre, von **Produkin Gorski** entwickelt
- Drei **Schwarz-Weiß-Bilder** nacheinander aufgenommen – je mit **Rot-, Grün- oder Blaufilter**
- Darstellung durch **Lichtprojektion** (Laterna Magica)
- Nachteil: **Geisterbilder** bei Bewegung, da jede Farbe zu einem **anderen Zeitpunkt** aufgenommen wird
- Heute noch bei **Mikroskopen** in Verwendung

## Sequential Color Three-Pass CCD Imaging System

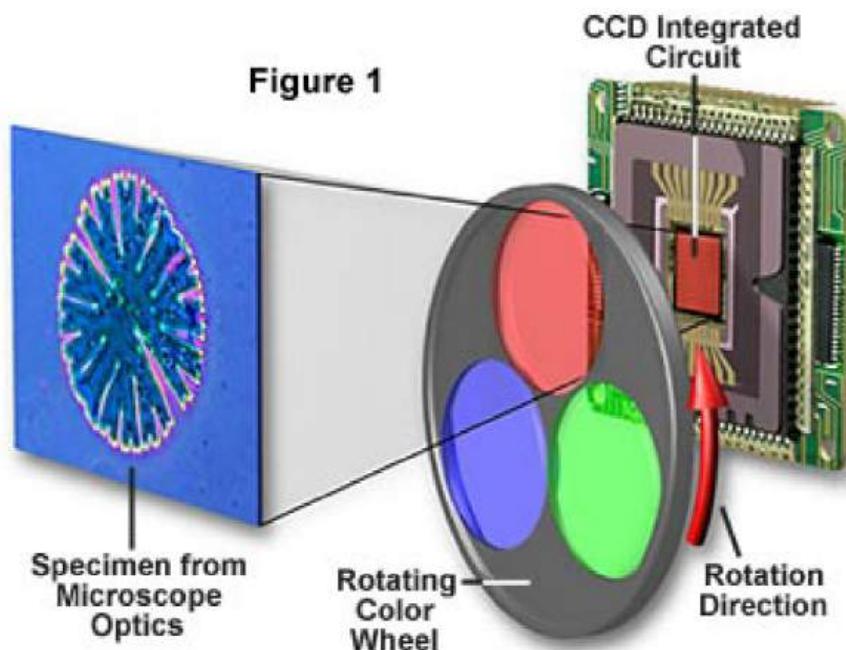


Abbildung 11: Farbaufnahme mittels Color Wheel

## 3-Chip-Technik

- Prisma trennt Licht in Rot, Grün, Blau (RGB)
- Jeder Farbkanal wird von eigenem CCD-Sensor aufgenommen
- Exakte Ausrichtung nötig (Der Lichtstrahl muss bei allen 3 Filtern den selben Pixel treffen), um Farbtreue zu garantieren
- nur eingesetzt bei Videokameras und Camcordern

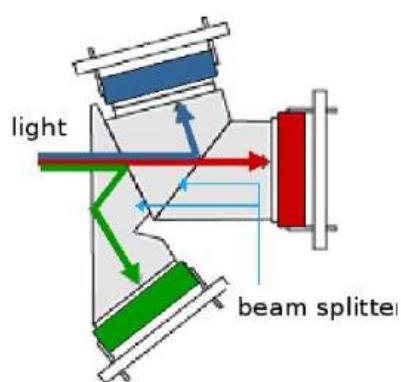


Abbildung 12: Farbtrennung durch Prisma

## Color Filter Array (CFA)

- Häufigste Methode bei einzelnen CCD- oder CMOS-Sensoren
- Über jedem Pixel liegt ein Farbfilter (Mosaikstruktur)
- Bayer-Filter ist Standard:

- 50 % Grün
- 25 % Rot
- 25 % Blau
- Anpassung an **menschliche Farbwahrnehmung** (grünempfindlichster Bereich)

## Bayer-Muster und Demosaicing

- Output eines Bayer-Sensors = **Rohdatenbild** (Bayer-Muster)
- Jedes Pixel enthält **nur eine Farbkomponente**
- **Demosaicing**: Interpolation der fehlenden Farben
  - Nutzt Annahme: **Geringe Unterschiede** zwischen benachbarten Farbwerten
  - Einfache Verfahren: **Durchschnittswerte** benachbarter Pixel gleicher Farbe
  - Erweiterte Verfahren: Interpolation **entlang von Kanten**
  - Farbverhältnisse (z.B. Rot-Grün) bleiben konstant → erst Grün interpolieren, dann Rot und Blau

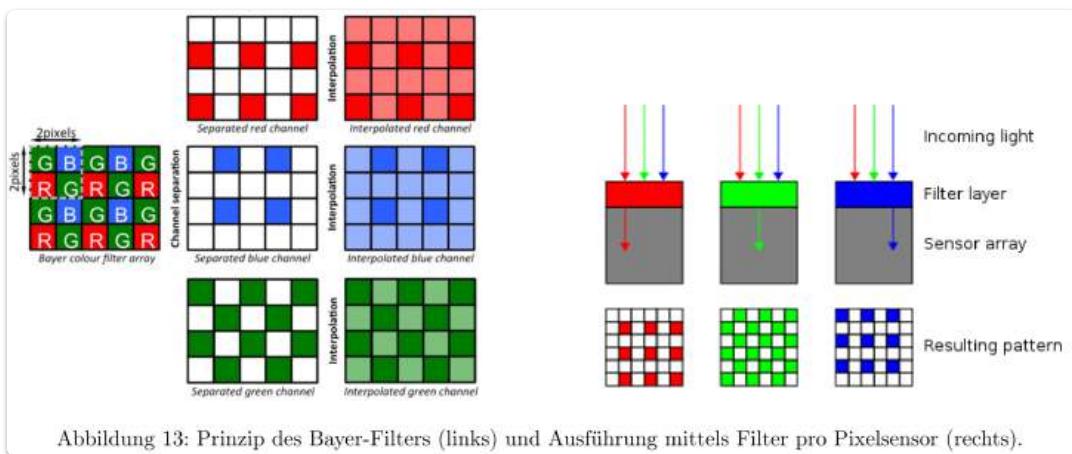


Abbildung 13: Prinzip des Bayer-Filters (links) und Ausführung mittels Filter pro Pixelsensor (rechts).

## Auflösung und Verarbeitung

- Effektive **geometrische Auflösung** des Sensors beträgt **nur ein Drittel**
- Demosaicing kann erfolgen:
  - **In der Kamera** (z.B. JPEG, TIFF-Export)
  - **Extern** durch Programme wie **nomacs** ([www.nomacs.org](http://www.nomacs.org))

## Foveon X3 Sensor

- Der **Foveon X3 Sensor** kombiniert die **3CCD-** und **CFA-Technik**
- Er nutzt ein **Array** von Fotodioden, wobei jede **vertikal gestapelt** ist und in einem **zweidimensionalen Raster** angeordnet wird
- Jede Fotodiode ist auf **verschiedene Wellenlängen** des Lichts empfindlich, was bedeutet, dass jede Diode eine **eigene spektrale Sensitivitätskurve** hat
- Das Signal der drei Fotodioden wird verarbeitet, um **drei additive Primärfarben** (Rot, Grün, Blau) zu erzeugen und so ein **fehlerfreies Farbbild** darzustellen

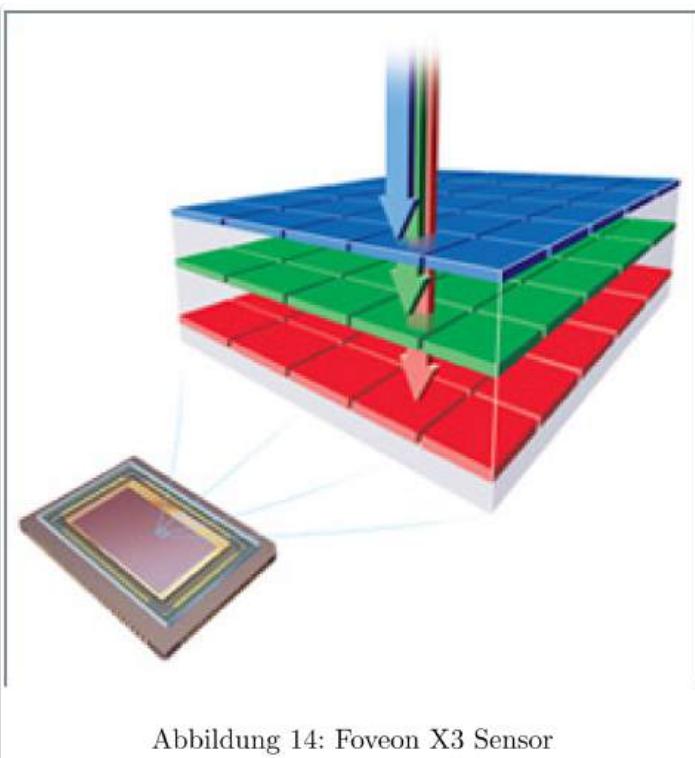


Abbildung 14: Foveon X3 Sensor

## Multi-Shot-Technologie

- Eine **weitere Möglichkeit** zur Erzeugung von **Echtfarbbildern** ist der Einsatz von **beweglichen Sensoren**
- Bei der **Multi-Shot-Technologie** wird der Sensor mithilfe eines **Piezomotors** präzise in **Einpixel-Schritten** bewegt
- Diese Technik ermöglicht es der Kamera, **mehr Farbinformationen** und **Daten** zu erfassen, als bei einer Einzelbildaufnahme
- Im **4-Shot-Modus** werden **vier Aufnahmen** gemacht:
  - **Sensorbewegung** erfolgt in vier Schritten (1 Pixel horizontal, 1 Pixel vertikal, erneut 1 Pixel horizontal und 1 Pixel vertikal)
  - Dieser Zyklus sorgt dafür, dass für jedes Pixel **echte RGB-Daten** erfasst werden
- Nach Beendigung des Zyklus kehrt der Sensor in seine **Ausgangsstellung** zurück

## Ideale vs. Reale Kamera

Reale Kameras weichen von idealen Kameramodellen ab, da sie Linsen mit optischen Phänomenen und nicht-ideale Bildsensoren mit Chipartefakten verwenden. Einige dieser Störungen sind:

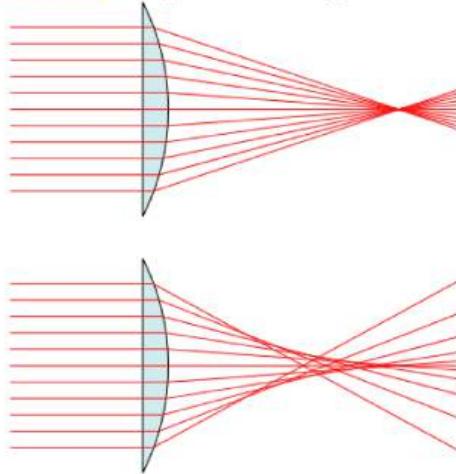


Abbildung 15: Optische Aberration

- Die **optische Aberration** tritt auf, wenn Licht von einem Punkt auf einem Objekt nicht in einem einzigen Punkt nach der Transmission durch die Linse zusammenfällt.
- Die **Chromatische Aberration** ist eine Art von Störung, bei der die Linse nicht im Stande ist, alle Farben im gleichen Konvergenzpunkt zu fokussieren. Diese Aberration tritt auf, da Linsen unterschiedliche Brechungsindizes für verschiedene Wellenlängen des Lichtes aufweisen. Chromatische Aberration manifestiert sich durch "Farbränder/-säume entlang der Abgrenzung zwischen dunklen und hellen Bildteilen."

- Die **sphärische Aberration** tritt auf, da Kugeloberflächen nicht die ideale Form für Linsen sind, aber beim Schleifen von Glas die einfachste Form darstellen. Diese Aberration führt dazu, dass Strahlen, die weiter am Rand der Linse auftreffen in einem etwas anderen Punkt fokussiert werden, als Strahlen die nahe der Achse liegen.
- Die **Linsenverzerrung** ist eine Abweichung einer geradlinigen Projektion, gerade Linien in einer Szene bleiben nicht gerade im Bild, diese können nach außen (tonnenförmig) oder nach innen (kissenförmig) verzerrt sein.

## Weißabgleich

- Jede Kamera** geht bei der Aufnahme eines Fotos zunächst von einem **Durchschnittsbild** aus
- Die Kamera versucht, ein Bild zu erstellen, das dem **Durchschnitt** in Bezug auf Belichtung, Farbigkeit und mehr entspricht – was nicht immer korrekt ist
- Weißabgleich** hilft, die **Farbstimmung** oder **Farbtemperatur** anzupassen
- Der Weißabgleich sagt der Kamera, welche Farbe als **weiß** bzw. als **18%-iges neutrales Grau** angesehen werden soll
- Farbtemperatur** wird in **Kelvin** gemessen, z.B. hat **Licht von Leuchstoffröhren** ca. 4000 Kelvin, **normale Glühbirnen** etwa 3200 Kelvin und im **Schatten** um 7000 Kelvin
- Automatischer Weißabgleich:**
  - Die Kamera sucht eine neutrale Fläche und passt alle Farben an diese an
  - Dies funktioniert nicht immer perfekt, daher gibt es **verschiedene Anpassungsmöglichkeiten**

## Optionen für den Weißabgleich

- Der **Weißabgleich** ist meist auf **Automatik** eingestellt
- **Voreingestellte Lichtsituationen** wie z.B. **Kunstlicht, Sonne, Schnee** oder **Schatten** stehen zur Auswahl
- **Manueller Weißabgleich:**
  - Ein weißes Blatt oder eine **18% Graukarte** wird fotografiert
  - Dieses Bild wird als **Referenz** verwendet, um den Weißabgleich korrekt zu erzeugen
- **Software:** Der Weißabgleich kann auch **nachträglich** angepasst werden
  - Hierbei wird ein **weißer Bereich** im Bild ausgewählt, und die Software berechnet den Weißabgleich
  - Beim **RAW-Format** funktioniert dies besonders gut, da der in der Kamera vorgenommene Weißabgleich nur als Vorschlag gespeichert wird und nachträglich geändert werden kann

No white balance



White balance



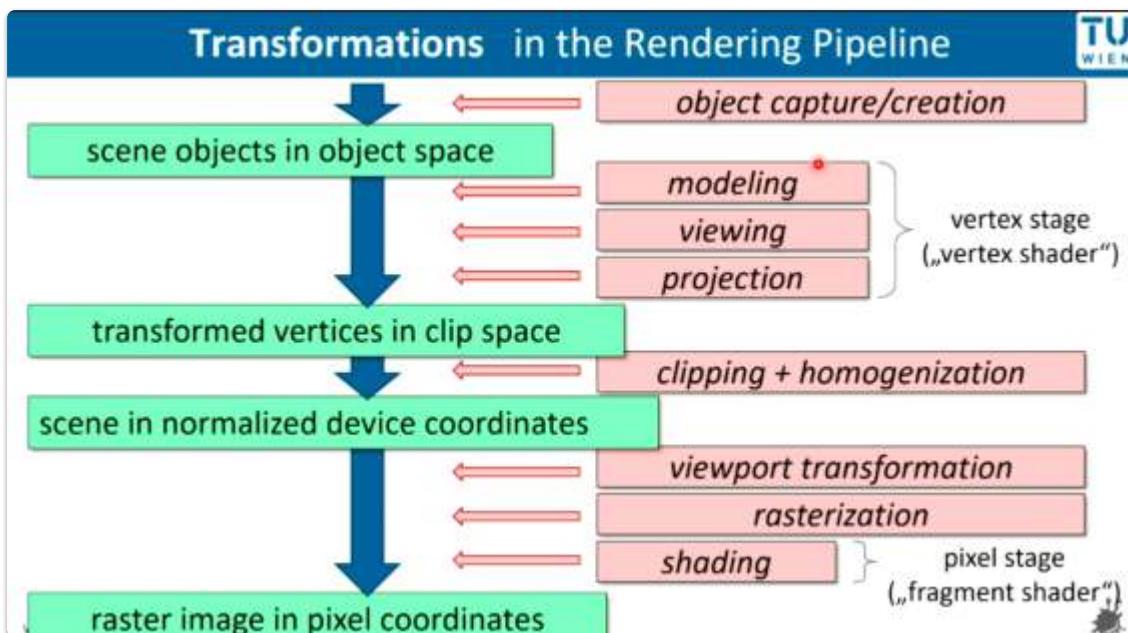


# 3. Transformationen

Behandelt:

- Verschieben
  - Vergrößern
  - Verkleinern
  - Drehen
  - Spiegeln
  - usw...
- ...innerhalb oder zwischen Koordinatensystemen.

In der Rendering Pipeline wäre das hier:



## Einfache 2D-Transformationen

### Translation (Verschiebung)

Das Verschieben eines Punktes  $(x, y)$  um den Vektor  $(t_x, t_y)$  liefert den transformierten Punkt:

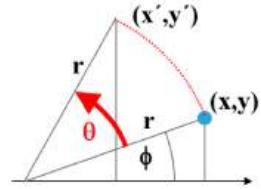
$$(x', y') = (x + t_x, y + t_y)$$

### Rotation (Drehung)

Durch das Drehen eines Objektes mit dem Winkel  $\theta$  um den Koordinatenursprung kommt der Punkt  $(x, y)$  wegen  $x = r \cdot \cos\theta$  und  $y = r \cdot \sin\theta \Rightarrow x' = r \cdot \cos(\phi + \theta) = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta = x \cdot \cos\theta - y \cdot \sin\theta$  (und  $y'$  analog) auf

$$(x', y') = (x \cdot \cos\theta - y \cdot \sin\theta, x \cdot \sin\theta + y \cdot \cos\theta)$$

zu liegen.



### Skalierung (Vergrößerung oder Verkleinerung)

Beim Skalieren eines Objektes um den Faktor  $s$  um den Ursprung  $(0, 0)$  wird ein Punkt  $(x, y)$  auf

$$(x', y') = (s \cdot x, s \cdot y)$$

abgebildet. Wenn in x- und y-Richtung unterschiedliche Skalierungsfaktoren  $s_x$  und  $s_y$  verwendet werden, dann erhält man

$$(x', y') = (s_x \cdot x, s_y \cdot y).$$

### Reflexion (Spiegelung)

Die Spiegelung an einer Koordinatenachse ist ein Sonderfall der Skalierung mit  $s_x = -1$  oder  $s_y = -1$ .

Alle anderen Transformationen können durch Hintereinanderausführen der beschriebenen einfachen Transformationen erreicht werden. Diese Abbildungen (mit Ausnahme der Translation) lassen sich auch durch Transformationsmatrizen darstellen. Dabei werden die Punkte als Vektoren dargestellt, um mit ihnen die Matrixoperationen durchführen zu können:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad (x', y') = (x + t_x, y + t_y) \quad \dots?$$

(a) Skalierung

(b) Rotation (gegen Uhrzeigersinn)

(c) Spiegelung um x-Achse

(d) Verschiebung

## Homogene Koordinaten

### Grundprinzip

- Ziel: Auch Translation soll in Matrixform darstellbar sein → **homogene Koordinaten**
- Erweiterung eines Punkts  $(x, y)$  zu  $(x, y, h)$ , meist mit  $\mathbf{h} = 1$
- Rückrechnung in 2D:
  - $x = \frac{x'}{h}$
  - $y = \frac{y'}{h}$

### Grundlegende Transformationen

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

(a) 2D-Skalierung

(b) 2D-Rotation

(c) 2D-Translation

### Vorteil einheitlicher Matrixform

- Alle Transformationen können als Matrizen dargestellt und kombiniert werden
- Durch **Assoziativität** der Matrizen: Vorab-Multiplikation möglich → eine Gesamtmatrix M → effizientere Berechnung

Warum ist es vorteilhaft, alle Transformationen in einheitlicher Matrixschreibweise zu formulieren? Meist werden größere Teile (Objekte, Bilder) als Ganzes transformiert, d.h. auf jeden Punkt dieser Gebilde wird die gleiche Folge von Transformationen angewendet. Dies entspricht einer sequenziellen Multiplikation eines Punktes  $P$  mit Matrizen  $M_1, M_2, M_3, \dots$ :  $P' = M_1 \cdot P, P'' = M_2 \cdot P', P''' = M_3 \cdot P'', \dots$  Nun kann man sich die Assoziativität der Matrixmultiplikation [ also  $(M_1 \cdot M_2) \cdot M_3 = M_1 \cdot (M_2 \cdot M_3)$  ] zunutze machen und den Rechenaufwand damit massiv reduzieren:

$$\text{Statt } P^{(n)} = M_n \cdot (M_{n-1} \cdot \dots \cdot (M_3 \cdot (M_2 \cdot (M_1 \cdot P)))) \dots \\ \text{schreibt man } P^{(n)} = (M_n \cdot M_{n-1} \cdot \dots \cdot M_3 \cdot M_2 \cdot M_1) \cdot P.$$

Nun kann man  $M = (M_n \cdot M_{n-1} \cdot \dots \cdot M_3 \cdot M_2 \cdot M_1)$  vorher ausrechnen und diese eine Gesamtmatrix dann auf alle Punkte anwenden.

## Kurznotation für Transformationen

- $T(tx, ty)$  = Translation um Vektor  $(tx, ty)$
- $R(\theta)$  = Rotation um Winkel  $\theta$
- $S(s_x, s_y)$  = Skalierung in x- und y-Richtung

## Inverse Transformationen

- $T^{-1}(tx, ty) = T(-tx, -ty)$
- $R^{-1}(\theta) = R(-\theta)$
- $S^{-1}(sx, sy) = S(1/sx, 1/sy)$

## Komplexere Transformationen

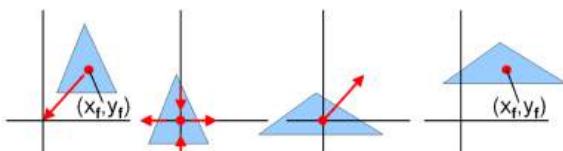
### Skalierung um einen anderen Punkt als den Koordinatenursprung:

1. Schritt = Verschieben des Skalierungszentrums in den Koordinatenursprung:  $T(-x_f, -y_f)$
2. Schritt = Skalieren des Objektes im Koordinatenursprung:  $S(s_x, s_y)$
3. Schritt = Zurückverschieben des Objektes an die ursprüngliche Stelle:  $T^{-1}(-x_f, -y_f) = T(x_f, y_f)$

Die allgemeine Matrix für die Skalierung mit  $(x_f, y_f)$  als Zentrum erhält man nun so:

$$S(x_f, y_f, s_x, s_y) = T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)$$

Als weiteres Beispiel betrachten wir die



### Spiegelung an einer beliebigen Achse $y = mx + b$ :

1. Schritt = Verschieben, so dass die Achse durch den Koordinatenursprung geht:  $T(0, -b)$
2. Schritt = Drehen, so dass die Achse z.B. mit der x-Achse zusammenfällt:  $R(-\theta)$  [  $m = \tan\theta$  ]
3. Schritt = Spiegeln an der x-Achse:  $S(1, -1)$
4. Schritt = Zurückdrehen, so dass Achse ursprünglichen Winkel hat:  $R^{-1}(-\theta) = R(\theta)$
5. Schritt = Zurückverschieben, so dass Achse an der ursprünglichen Stelle liegt:  $T^{-1}(0, -b) = T(0, b)$

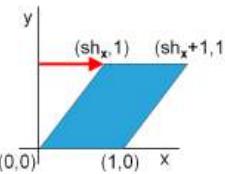
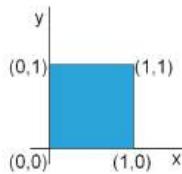
Die allgemeine Matrix für die Spiegelung an der Achse  $y = mx + b$  erhält man nun so:

$$X(m, b) = T(0, b) \cdot R(\theta) \cdot S(1, -1) \cdot R(-\theta) \cdot T(0, -b)$$

## Scherung

Eine weitere wichtige Transformation ist die Scherung, sie hat im einfachsten Fall in x-Richtung mit fixierter x-Achse die Form:

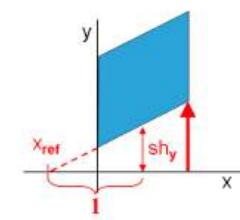
$$\begin{bmatrix} 1 & \text{sh}_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Etwas allgemeiner kann die Scherung auch an einer Parallelen zu einer Achse erfolgen, das sieht etwa in y-Richtung dann so aus:

Natürlich kann man jetzt auch wieder ganz leicht die allgemeine Matrix für eine Scherung ableiten, die nicht parallel zu einer der Koordinatenachsen erfolgt: zuerst Drehung in achsenparallele Lage, dann Scherung, und dann wieder zurückdrehen.

$$\begin{bmatrix} 1 & 0 & 0 \\ \text{sh}_y & 1 & -\text{sh}_y \cdot x_{\text{ref}} \\ 0 & 0 & 1 \end{bmatrix}$$



## Viewing-Transformation

- Transformation von Weltkoordinaten in **Kamerakoordinaten** (auch Viewport-Koordinaten genannt)
- Kombination aus:
  - Ursprung verschieben
  - Rotation zur Achsenausrichtung
  - Skalierung der Achsen
- Kein Rückverschieben/-drehen nötig

## Affine Transformationen

- Alle behandelten Transformationen = affine Transformationen
- Definition: Koordinaten werden über lineare Abbildung + Translation transformiert
- Eigenschaften:
  - Erhalten Kollinearität (3 Punkte auf Linie bleiben auf Linie)
  - Erhalten Verhältnis von Streckenlängen auf Geraden
  - Parallele Linien bleiben parallel
  - Endliche Punkte bleiben endlich
- Zusammensetbar aus: Skalierung, Rotation, Translation, Scherung, Spiegelung
- Transformationen mit nur Rotation, Translation, Spiegelung = längen- und winkelerhaltend

# 3D Transformationen

## 3D Transformationen

Alle 2D-Konzepte lassen sich leicht auf 3D erweitern. Man benötigt wieder eine **homogene Komponente**, so dass 4x4-Matrizen auf 4-dimensionalen Vektoren operieren. Später werden wir sehen, dass man auch Projektionen auf diese Art formulieren kann.

Hier sind einmal die **wichtigsten 3D-Transformationen**:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(a) 3D-Skalierung

(b) 3D-Translation

(c) Spiegelung um yz-/xz-/xy-Ebene

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(d) 3D-Rotation um x-Achse

(e) 3D-Rotation um y-Achse

(f) 3D-Rotation um z-Achse

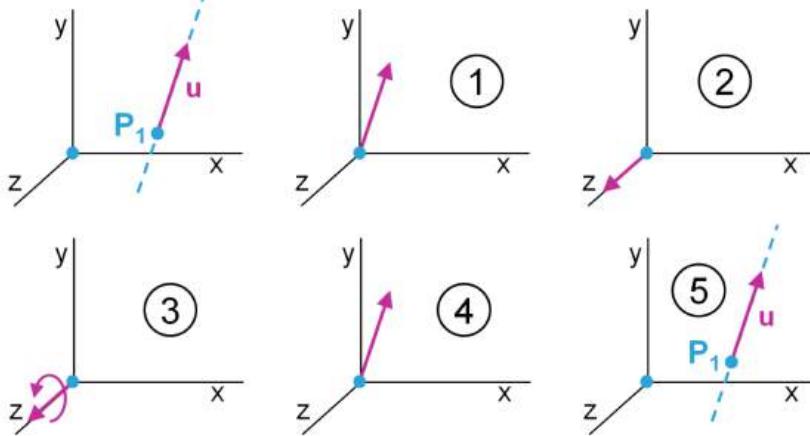
Die Namen für diese einfachen Transformationsmatrizen sehen nun so aus:

- $\mathbf{T}(t_x, t_y, t_z)$  = Translation um den Vektor  $(t_x, t_y, t_z)$
- $\mathbf{R}_x(\theta)$  = Rotation um den Winkel  $\theta$  um die x-Achse; y- und z-Achse analog
- $\mathbf{S}(s_x, s_y, s_z)$  = Skalierung um die Faktoren  $s_x, s_y$  und  $s_z$ .

Als Beispiel für eine komplexere Transformation wollen wir eine

**Drehung um den Winkel  $\theta$  um eine beliebige Achse im Raum**

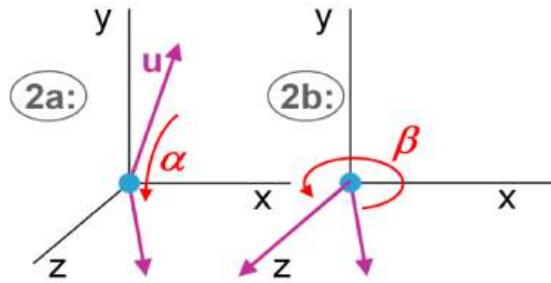
herleiten. Die Achse sei durch einen Punkt  $P_1(x_1, y_1, z_1)$  und einen Richtungsvektor  $u$  gegeben.



1. Schritt = Punkt  $P_1$  in den Koordinatenursprung verschieben:  $T(-x_1, -y_1, -z_1)$
2. Schritt = Vektor  $u$  in die z-Achse drehen
  - (a) Vektor  $u$  um die x-Achse in die xz-Ebene drehen:  $R_x(\alpha)$   
Sei  $u = (a, b, c)$ , dann ist  $u' = (0, b, c)$  die Projektion von  $u$  auf die yz-Ebene. Der Drehungswinkel  $\alpha$  um die x-Achse ergibt sich aus  $\cos\alpha = c/d$  mit  $d = \sqrt{(b^2 + c^2)}$
  - (b) Vektor  $u$  um die y-Achse in die z-Achse drehen:  $R_y(\beta)$   
Der Drehungswinkel  $\beta$  um die y-Achse ergibt sich aus  $\cos\beta = d$  (bzw.  $\sin\beta = -a$ )
3. Schritt = Drehung um  $\theta$  um die z-Achse ausführen:  $R_z(\theta)$
4. Schritt = Vektor  $u$  in die ursprüngliche Richtung zurückdrehen: zuerst  $R_y(-\beta)$ , dann  $R_x(-\alpha)$
5. Schritt = Punkt  $P_1$  an die ursprüngliche Position zurückverschieben:  $T(x_1, y_1, z_1)$

Die resultierende Matrix berechnet sich also so:

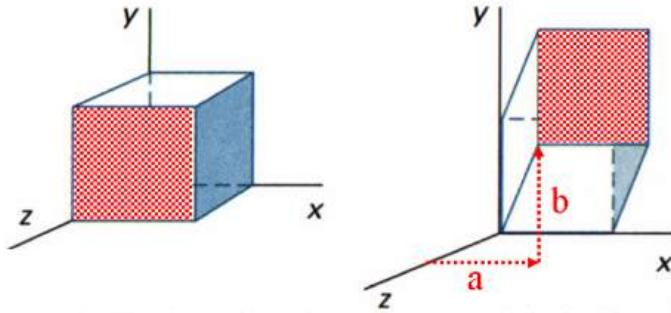
$$\begin{aligned} R(\theta) &= T^{-1}(-x_1, -y_1, -z_1) \cdot R_x - 1(\alpha) \cdot R_y - 1(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T(-x_1, -y_1, -z_1) = \\ &= T(x_1, y_1, z_1) \cdot R_x(-\alpha) \cdot R_y(-\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T(-x_1, -y_1, -z_1) \end{aligned}$$



Die **Scherung in 3D** ist ebenfalls einfach darstellbar:

Eine Scherung parallel zur xy-Ebene um den Wert  $a$  in x-Richtung und den Wert  $b$  in y-Richtung erreicht man mittels

$$\begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Eine Scherung mit einer anderen Fixebene als einer der Koordinatenhauptebenen kann man sich leicht ableiten.



# 3. Bildcodierung und Kompression

Quellen:

- EVC\_Skriptum\_CV, p.15 bis EVC\_Skriptum\_CV, p.19
- 

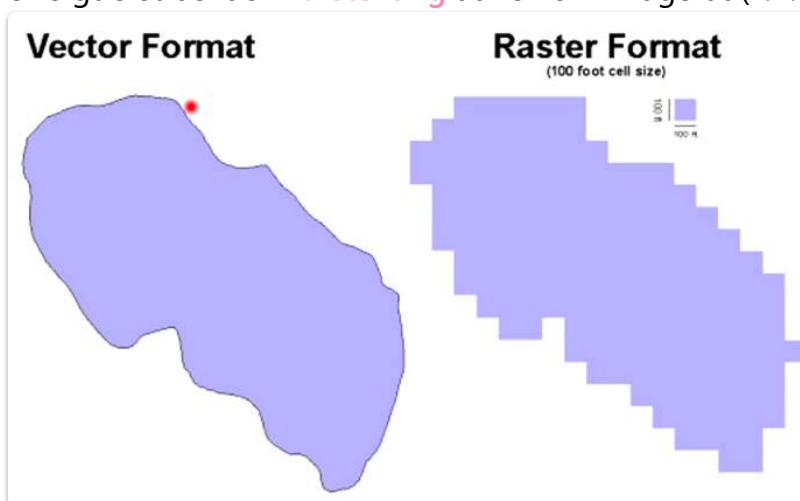
## Digitales Bild-Dateienformat

### Kontinuierliche vs. digitale Daten

- Wenn eine Zahl unendlich viele **mögliche Werte** annehmen kann, spricht man von **kontinuierlichen** oder **analogen** Daten
- **Computer** können mit analogen Daten nicht direkt arbeiten
- Daher müssen diese Daten **digitalisiert** werden, um sie für den Computer bearbeitbar zu machen
- Dieser Umwandlungsprozess erfolgt z.B. beim **Scannen von Fotos** oder beim Fotografieren mit **digitalen Kameras**

### Digitale Bilder

- Ein digitales Bild ist eine **numerische Repräsentation** eines zweidimensionalen Bildes
- Das Bild kann entweder aus **Vektorbeschreibungen** oder einem **Raster** bestehen
- **Rasterbild:** Ein Raster von **diskreten Werten** (Pixel), bei dem jeder **Bildpunkt** mit seinem **Helligkeitswert** oder **Farbwert** gespeichert wird
- **Vektorbild:** Der Bildinhalt wird in **geometrischen Objekten** dargestellt und die Rasterung erfolgt erst bei der **Darstellung** auf einem Endgerät (z.B. **Display** oder **Drucker**)



### Speicherung von Bilddaten

- Digitale Bildinformationen müssen in einem **Bilddatenformat** abgespeichert werden

- In der Frühzeit der digitalen Bildverarbeitung (bis etwa 1985) gab es eine große Anzahl unterschiedlicher **Dateiformate**, was zu vielen notwendigen **Konvertierungsprogrammen** führte
- Heute gibt es standardisierte **Dateiformate**, die den **Austausch** von Bilddaten erleichtern und die **langfristige Lesbarkeit** fördern

## Speichergröße eines Rasterbildes

- Ein Rasterbild enthält ein **Pixelraster**, das für jede Rasterzelle eine bestimmte Anzahl an **Bits** zur Farbgebung bereitstellt – dies entspricht der **Farbtiefe**
  - Ein Bild mit **LxN** (L Zeilen und N Spalten), **2B** (B = Anzahl der Bits pro Rasterzelle), **c** Farbkomponenten kann unkomprimiert als:
    - $L \times N \times B \times c$  gespeichert werden
    - Beispiel: Bei einer Bildgröße von **1024x768**, **3 Farbkomponenten (RGB)** und **256 (28) Graustufen** ergibt sich:
      - $1024 \times 768 \times 3 \times 8 = 18,87 \text{ MBit} \rightarrow 2,36 \text{ MByte}$
  - **Bilddatengröße** korreliert positiv mit der Anzahl der **Pixel** und der **Farbtiefe** (Bits pro Pixel)
  - **Prüfungs-ähnliches Beispiel** dazu:
    - Wie viel Speicherplatz benötigt man für die Speicherung des Bildinhaltes bei einem **RGB Farbbild** der Größe **1.024x768**, wenn pro Farbkanal **4.096** verschiedene Werte kodiert werden sollen?
      - $1.024 \times 768 = 786.432 \text{ Pixel}$
      - $4.096 = 2^{12} \Rightarrow 12 \text{ Bit/Pixel}$
      - RGB = **3 Farbkanäle**
    - ⇒  **$786.432 \cdot 12 \cdot 3 = 28.311.552 \text{ Bit}$**
    - ⇒  **$28.311.552 : 8 = 3.538.944 \text{ Byte}$**
    - ⇒  **$3.538.944 : 1.024 = 3.456 \text{ KiloByte (KB)}$**
- Size = LxNxBxc*
- 

## Raster-Bildformate

### Raw-Bildformat

- **Raw-Bildformat** ermöglicht es, auf die tatsächlich von der Kamera aufgenommenen Bilddaten zuzugreifen
- Speichert für **jeden Pixel** den entsprechenden **Farbwert** ohne **Nachbearbeitung** (bei 1-Chip-Kameras werden Rot-, Grün- und Blauwerte entsprechend dem **CFA-Muster** gespeichert)
- **RAW-Format** stellt kein „richtiges“ Farbbild dar, da **2/3 der Farbinformation** interpoliert werden müssen

- Muss zur **farbigen Anzeige** umgewandelt werden

## Konventionelle Raster-Bildformate

- Beispiele für konventionelle Raster-Bildformate:
  - **Bitmap (BMP)**
  - **Portable Network Graphics (PNG)**
- Ein modernes Raster-Bildformat ist in der Lage, **zweidimensionale digitale Bilder** beliebiger **Breite, Höhe** und **Auflösung** abzuspeichern

## Struktur von Rasterbilddateien

- Eine Rasterbilddatei besteht aus **Strukturen fixer Größe (Header)** und **variabler Größe** (bildabhängig)
  - Die Strukturen erscheinen in einer vordefinierten Sequenz
    - Beispiel: **BMP**: Der **Bitmap File Header** speichert allgemeine Informationen über die Bitmap-Datei (14 Bytes)
  - **Metainformationen** werden in jedem individuellen Dateiformat gespeichert
- 

## Vektor-Bildformate

- **Vektor-Bildformat** beinhaltet eine **geometrische Beschreibung**, die problemlos für jede gewünschte **Anzeigegröße** gerendert werden kann
- **Rasterisierung**: An einem bestimmten Punkt müssen **alle Vektorgrafiken** rasterisiert werden, um auf einem digitalen Bildschirm angezeigt werden zu können
  - siehe: [5. Rasterisierung](#)

## Plotter und Vektordaten

- **Plotter** sind Drucker, die Vektordaten zum **Zeichnen von Grafiken** verwenden

## Computer Graphics Metafile (CGM)

- **CGM** ist ein freier und offener internationaler Standard für die Speicherung von **2D-Vektorzeichnungen, Rasterbildern und Text**
- Der Standard wird in Bereichen wie **technische Illustration, Kartografie, Visualisierung** und **elektronische Publikationen** verwendet
- Alle grafischen Elemente werden in **Quelltextdateien** spezifiziert, die anschließend zu einer **Binärdatei** oder **Textdarstellung** kompiliert werden
- **CGM** stellt Instrumente für den Austausch von Grafikdaten bei der Darstellung zweidimensionaler grafischer Informationen zur Verfügung, unabhängig von einer bestimmten **Anwendung, Plattform, System** oder **Gerät**

## Windows-Metafile (WMF)

- WMF wurde 1990 entwickelt
  - WMF ermöglicht den **Datenaustausch** zwischen Anwendungen und beinhaltet sowohl **Vektorgrafiken** als auch **Bitmap-Komponenten**
- 

## Bildkompression

- **Ziel der Bildkompression:** Reduzierung irrelevanter und redundanter Bildinformationen, um die Daten effizient zu speichern oder zu übertragen.
  - **Arten der Kompression:**
    - **Verlustfrei (lossless):** Keine Daten gehen verloren, Bildqualität bleibt erhalten. Wird oft für medizinische Bilder, technische Zeichnungen oder Comics verwendet.
    - **Verlustbehaftet (lossy):** Daten gehen verloren, jedoch oft unmerklich. Wird für natürliche Bilder wie Fotografien verwendet, da es die Dateigröße stark reduziert.
  - **Verlustbehaftete Kompression:**
    - Produziert **kompressionsartefakte** bei niedriger Bitrate.
    - In vielen Fällen als **visuell verlustfrei** bezeichnet, wenn der Verlust für den menschlichen Betrachter nicht wahrnehmbar ist.
- 

## Verlustfreie Datenkompression

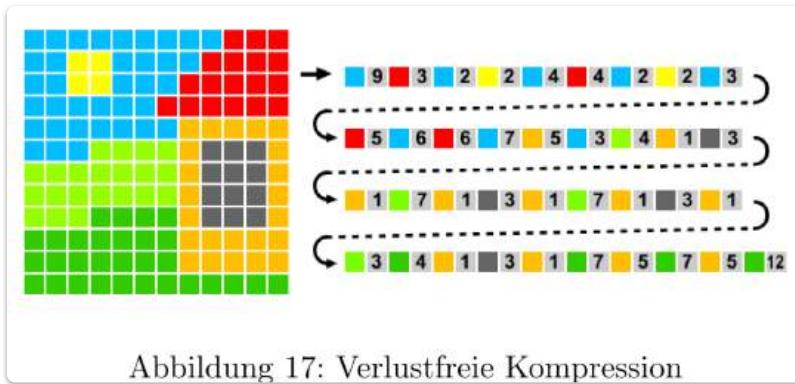


Abbildung 17: Verlustfreie Kompression

- Erlaubt eine exakte Rekonstruktion der Originaldaten. Wird in Bereichen genutzt, in denen die vollständige Datenintegrität wichtig ist (z. B. bei medizinischen Daten oder technischen Zeichnungen).

## Prozess der Kompression

1. **Erstellung eines statistischen Modells** der Eingabedaten.
2. **Abbildung der Daten auf eine Bitreihe**, wobei häufig vorkommende Daten kürzere Bitfolgen erzeugen als seltene.

## Run Length Encoding (RLE)

- Ein grundlegender Kompressionsalgorithmus, der **Datenwiederholungen** speichert.
- Beispiel: Eine lange Sequenz gleicher Farben (z. B. bei Icons, Linienzeichnungen).
- **Nachteil:** Für natürliche Bilder, die keine langen Wiederholungssequenzen haben, kann es zu einer **Vergrößerung der Dateigröße** führen.
- **Entropiecodierung:** Sie erstellt **kurze Codes für häufige Symbole** und **lange Codes für seltene Symbole**. Dies reduziert die durchschnittliche Länge der Codes und verbessert die Kompression.

## Huffman-Codierung

- **Binärbaum:** Wird erstellt, bei dem die **Blätter** die Symbole und deren **Wahrscheinlichkeit (Häufigkeit)** enthalten. Die **Knoten** sind entweder Blätter oder interne Knoten.
- **Codeerstellung:**
  1. Beginnt mit den **Blättern**, die die Häufigkeit jedes Symbols enthalten.
  2. Zwei Knoten/Blätter mit den **geringsten Wahrscheinlichkeiten** werden zu einem neuen Knoten zusammengeführt. Der neue Knoten erhält eine Wahrscheinlichkeit, die der Summe der beiden Kinder entspricht.
  3. Der Vorgang wird wiederholt, bis nur noch ein Knoten übrig bleibt – der **Huffman Tree**.

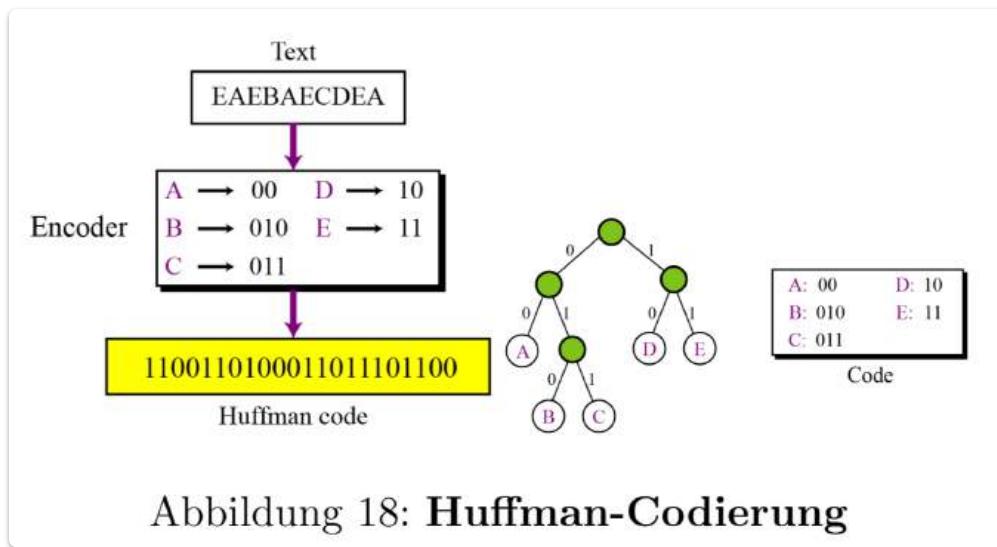


Abbildung 18: **Huffman-Codierung**

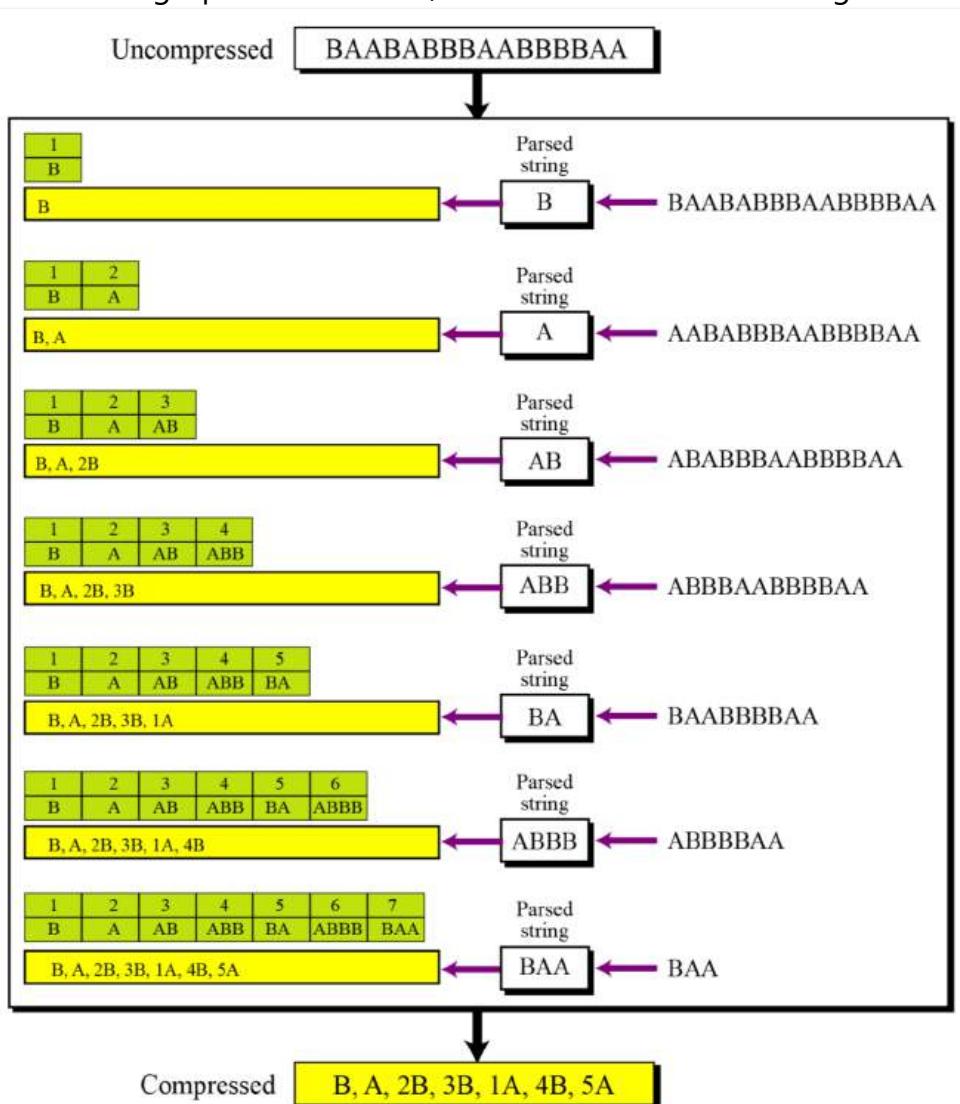
- **Ziel:** Der **Huffman Tree** ermöglicht die effiziente Zuordnung von **binären Codes** zu Symbolen, wobei häufige Symbole kürzere Codes erhalten und seltene Symbole längere.

## Lempel-Ziv (LZ) Kompressionsverfahren

Das **Lempel-Ziv (LZ) Kompressionsverfahren** basiert auf der **Wiederholung** von Daten und speichert diese Wiederholungen als Referenzen in einer **Tabelle**.

- **Codierung:**
  - 8-Bit Datensequenzen werden als 12-Bit Code komprimiert.
  - Codes 0-255 repräsentieren **einzelne Zeichen** (1-Zeichen-Sequenzen).

- Codes 256-4059 repräsentieren **Sequenzen**, die in einer Tabelle gespeichert sind.
- **Kompressionsprozess:**
  - Tabelle initialisieren:** Alle **1-Zeichen-Strings** werden zu Beginn als Einträge in der Tabelle gespeichert.
  - Längster String finden:** Der längste String **W**, der mit den aktuellen Eingabedaten übereinstimmt, wird identifiziert.
  - Tabellenindex ausgeben:** Der **Index für W** wird ausgegeben, und der String **W** wird vom Input entfernt.
  - Neuen String hinzufügen:** Der **neue String W + das nächste Symbol** wird zur Tabelle hinzugefügt.
  - Wiederholen:** Der Prozess geht weiter, bis das gesamte Eingabedaten verarbeitet sind.
- **Ziel:** Der Algorithmus **reduziert die Daten**, indem häufig wiederholte Sequenzen als Referenzen gespeichert werden, anstatt sie mehrfach abzulegen.

Abbildung 19: **LZ-Codierung**

Bildformate die Verlustfreie Kompression verwenden:

## GIF (Graphics Interchange Format):

- **Einführung:** 1987 von CompuServe.
- **Farbe:** Unterstützt bis zu **256 Farben** (8 Bit pro Pixel) aus dem 24-Bit RGB Farbraum.
- **Verwendung:** Besonders geeignet für **kleine Bilder** (z. B. Icons) und **Animationen**.
- **Einschränkungen:** Weniger geeignet für **Fotografien** aufgrund der begrenzten Farbpalette.

## PNG (Portable Network Graphics):

- **Entwicklung:** Entwickelt als Verbesserung und Ersatz für GIF.
- **Farbe:** Unterstützt **24-Bit RGB** oder **32-Bit RGBA** (mit Transparenz) sowie **Graustufen**.
- **Kompression:** Verlustfreie Kompression mittels **PKZIP**.
- **Verwendung:** Besonders für den **Webbereich** geeignet, aber nicht für **hochqualitative Druckgrafiken**.
- **Farträume:** Unterstützt nur **RGB**, keine CMYK.

## TIFF (Tagged Image File Format):

- **Verwendung:** Beliebt bei **Grafikern, Fotografen, Verlagen und Wissenschaftlern**.
- **Unterstützung:** Kann **Graustufenbilder, Indexbilder** und **Vollfarbenbilder** speichern.
- **Besonderheit:** Kann mehrere **Bilder** mit unterschiedlichen Eigenschaften in einer Datei speichern.
- **Kompression:** Unterstützt **verschiedene Kompressionsverfahren** (z. B. **LZW, ZIP, JPEG, CCITT**).
- **Verwendung:** Wird häufig für **Dokumentenarchivierung, wissenschaftliche Anwendungen** und in der **Digitalfotografie** verwendet.

## Verlustbehaftete Datenkompression

- **Prinzip:** Entfernung von Datenteilen, die für das menschliche Wahrnehmungssystem nicht oder nur schwer erkennbar sind.
- **Ziel:** Die Daten werden so komprimiert, dass möglichst wenig Qualität verloren geht, aber die Dateigröße erheblich reduziert wird.
- **Transformation:** Daten werden in eine neue Domäne umgewandelt, die die relevanten Informationen effizienter darstellt.

### JPEG-Kompression:

- **Entwickelt:** Von der **Joint Photographic Experts Group (JPEG)**, 1990 als **ISO-Standard** etabliert.
- **Ziel:** Durchschnittliche Datenreduktion von **1:16**, ideal für **fotografische Bilder**.

## Codierungsprozess in JPEG:

### 1. Farbraumkonversion und Downsampling:

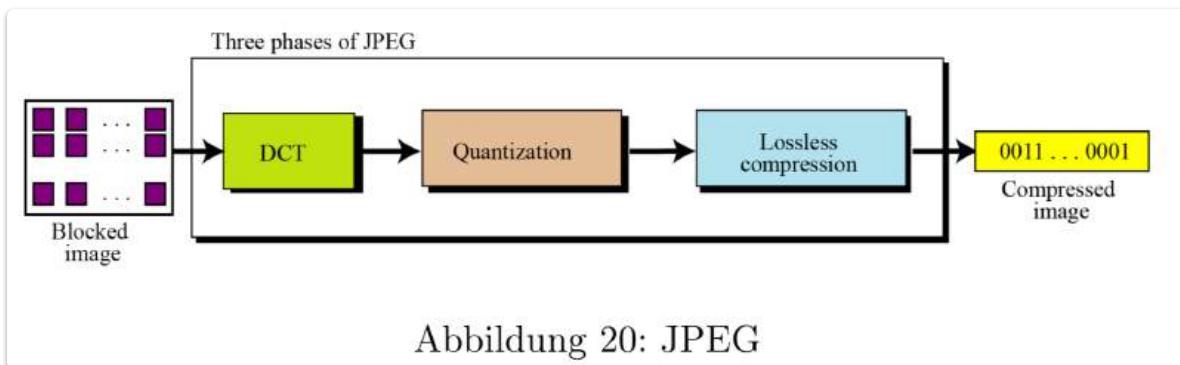
- Das Bild wird von **RGB** (Rot, Grün, Blau) in den **YCbCr-Farbraum** umgewandelt, wobei Y die Helligkeit (Luminanz) und Cb sowie Cr die Farbinformationen (Chrominanz) darstellen.
- Downsampling** der Chrominanzkanäle (Cb und Cr) erfolgt, da das menschliche Auge weniger empfindlich auf Farbdetails als auf Helligkeitsdetails reagiert.

### 2. Kosinustransformation und Quantisierung:

- Das Bild wird in **8x8 Blöcke** unterteilt, und für jeden Block wird eine **diskrete Kosinustransformation (DCT)** durchgeführt, um die Frequenzen des Bildes zu berechnen.
- Die resultierenden **Spektralkoeffizienten** werden **quantisiert**, wobei hohe Frequenzen stärker reduziert werden, da diese weniger zur Wahrnehmung der Bildschärfe beitragen.

### 3. Verlustfreie Kompression:

- Nach der Quantisierung wird der Datenstrom mittels **verlustfreier Kompression** (z.B. **Lauflängenkodierung** oder **Huffman-Kodierung**) weiter komprimiert, um die Dateigröße zu minimieren, ohne zusätzliche Informationen zu verlieren.



Das JPEG-Verfahren nutzt also **wahrnehmungspsychologische Erkenntnisse** zur Reduktion von Bilddaten und ermöglicht eine hohe Kompressionsrate bei gleichzeitig guter Bildqualität.

## Diskrete Cosinus Transformation (DCT)

### DCT (Diskrete Kosinustransformation):

- Die DCT ist eine Variante der **Fouriertransformation**, die Signale (hier Bildpixel) in **Cosinuswellen** unterschiedlicher **Frequenz** und **Amplitude** zerlegt.
- Ziel: **Frequenz- und Amplitudenverteilung** der Bildpixel sichtbar zu machen, um zu verstehen, welche Bildteile hohe oder niedrige Frequenzen enthalten.

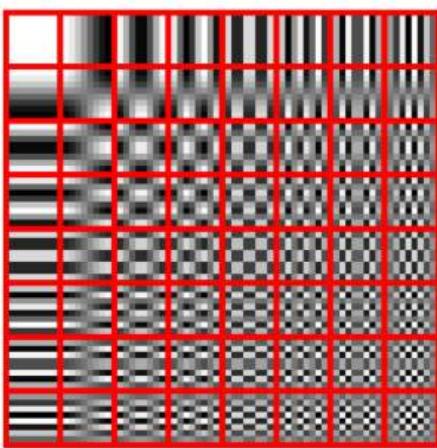


Abbildung 21: Die DCT ist eine Variation der Fouriertransformation.

- Formel für die 2D-DCT (für einen 8x8 Block):

$$F(u, v) = \alpha(u) \cdot \alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

- $u$  und  $v$ : **Horizontale** bzw. **vertikale Ortsfrequenz** ( $0 \leq u, v < 8$ ).
- $f(x, y)$ : Pixelwert am Punkt  $(x, y)$ .
- $F(u, v)$ : DCT-Koeffizient, der das Signal in Frequenzkomponenten zerlegt.
- $\alpha(u)$  und  $\alpha(v)$ : **Skalierungsfaktoren** zur Wahrung der Orthonormalität.

### JPEG-Kompression:

- **Quantisierung**: Die zentrale Methode der **verlustbehafteten** Kompression bei JPEG. Jeder DCT-Koeffizient wird durch einen vordefinierten Quantisierungswert geteilt und auf den nächsten Integer gerundet.
  - **Häufige Quantisierungswerte**: Niedrige Frequenzen erhalten **kleinere Quantisierungswerte** (präziser), während **hohe Frequenzen** größere Werte erhalten (weniger präzise), da das menschliche Auge weniger empfindlich gegenüber hohen Frequenzen ist.
  - Das führt dazu, dass **hochfrequente Komponenten auf 0 gerundet werden** und **niedrigere Frequenzen eine hohe Genauigkeit behalten**.
- **Ergebnis**: Die Quantisierung reduziert die Bildgröße, indem sie **unnötige Bilddetails entfernt**, insbesondere bei den hohen Frequenzen.
  - **Visuell kann die Kompression das Bild auf ein Fünftel seiner Originalgröße reduzieren**, ohne dass große visuelle Qualitätseinbußen sichtbar werden.

### Kompressionsartefakte:

- Bei zu starker **Kompression** (zu hoher Quantisierung) können **Blockartefakte** auftreten, da das Bild in 8x8-Blöcke unterteilt wird und hohe Frequenzen unzureichend dargestellt

werden.

- **Schwächen:**

- JPEG zeigt Schwächen bei **abrupten Übergängen** (wie Kanten oder Text).
- **Blockbildung:** Bei sehr starker Kompression können die 8x8 Blöcke sichtbar werden, was das Bild unnatürlich erscheinen lässt.

### Optimierung für natürliche Bilder:

- JPEG wurde speziell für **natürliche fotografische Bilder** entwickelt und ist nicht ideal für **Computergrafiken** oder **Bilder mit scharfen Kanten**, bei denen die Blockbildung besonders auffällt.
- 

## Video Kompression

- Ziel: **Reduzierung der Redundanz** in Videodaten, um die **Datenmenge** zu verringern und effizienter zu speichern oder zu übertragen.
- Kombination aus **räumlicher Bildkompression** (ähnlich wie bei Bildern) und **zeitlicher Bewegungskompensation** (bezieht sich auf die Bewegung zwischen den Frames).

### Techniken:

#### 1. Verlustbehaftete Kompression:

- Entfernt große Mengen an Daten, aber der visuelle Unterschied ist oft **kaum erkennbar**.
- Es gibt einen **Kompromiss** zwischen Videoqualität, Kompressionsaufwand und den Systemanforderungen.

#### 2. Räumliche Bildkompression:

- Komprimiert das Bild innerhalb eines einzelnen Frames (ähnlich wie JPEG).

#### 3. Zeitliche Bewegungskompensation:

- Verwendet **Makroblöcke** (quadratische Bildausschnitte), die Unterschiede zwischen Frames messen.
- Bei viel Bewegung im Video müssen mehr Daten codiert werden, da mehr Pixel sich zwischen den Frames ändern.

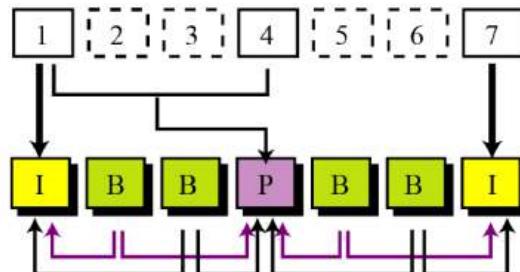
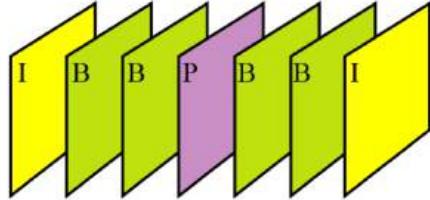
### Interframe- und Intraframe-Kompression:

- **Interframe-Kompression:**

- Verwendet **Frames davor und danach** (B/P-Frames) zur Kompression.
- Beispiel: Ein Frame wird nur dann gespeichert, wenn sich etwas verändert hat, ansonsten wird er durch ein Referenzbild ersetzt.

- **Intraframe-Kompression:**

- Komprimiert nur den aktuellen Frame (ähnlich wie Bildkompression, z.B. JPEG).



## Video-Kompressionstechniken:

- Veränderungen innerhalb eines Frames:** Wenn sich ganze Makroblöcke eines Frames verändern, kann der Kompressor Anweisungen wie **Verschieben, Rotieren oder Aufhellen** an den Dekompressor senden, um die Veränderung zu rekonstruieren.
- Interframe-Kompression:** Komprimiert Bereiche, die sich nicht verändert haben, durch **einfachen Verweis auf den vorherigen Frame**.

## MPEG-Kompression:

- MPEG** (Moving Picture Experts Group) ist eine weit verbreitete Technik zur Video- und Audiokompression.
- Asymmetrisch**: Codierung ist algorithmisch komplexer als Dekodierung (Vorteil im Broadcasting, da viele billige Dekodierer und wenige teure Codierer benötigt werden).

## MPEG-Standards:

- MPEG-1:**
  - Entwickelt für **Video CDs, SVCDs und DVDs** mit niedriger Videoqualität.
  - Ziel war es, Film und Ton auf die Bitrate einer **Compact Disc** zu kodieren.
- MPEG-2:**
  - Unterstützt **Zeilensprungverfahren** (Interlaced) und **High Definition** Auflösung.
  - Wichtig für **digitales Fernsehen, Kabelsignale und DVDs**.
- MPEG-4:**
  - Bietet **effizientere Codierung** und eignet sich auch für **Computergrafik-Applikationen**.
  - Wird neben MPEG-2 auch für **Blu-ray Discs** verwendet.



## 4. Farbe

Farbe ist eine der Grundesszenen der Computergraphik. Farbe richtig zu verstehen und zu handhaben ist ein Grundwerkzeug für Computergraphiker. Das häufig verwendete RGB-Farbmodell ist jedoch nicht in der Lage, alle Farben darzustellen, und auch sonst sehr approximativ. Viele Farbberechnungen werden meist nur näherungsweise gemacht (was oft reicht), und die exakte Farbenlehre ist sehr komplex

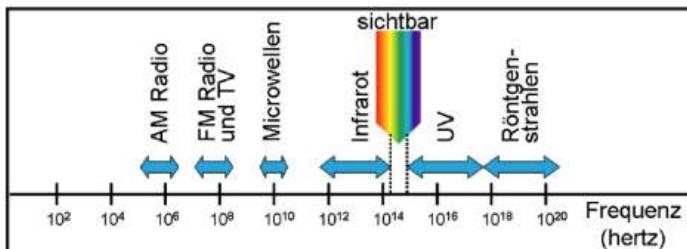
Ich habe zum Thema Farbe so gut wie keine PowerPoint slides eingebaut, bei Bedarf hier schauen: [EVC-CG04-Farbe\\_2025S\\_Slides.pdf](#)

## Was ist Farbe?

Unser Auge kann elektromagnetische Strahlung im eher engen Frequenz-bereich zwischen etwa  $3,8 \cdot 10^{14} \text{ Hz} (\approx 780 \text{ nm})$  und  $7,8 \cdot 10^{14} \text{ Hz} (\approx 380 \text{ nm})$  erkennen. Dabei empfinden wir diese Strahlung als Licht. Dieser sichtbare Bereich ist von Mensch zu Mensch leicht unterschiedlich, und viele Tiere haben andere Grenzen. Andere Frequenzbereiche dienen anderen Zwecken (siehe Diagramm).

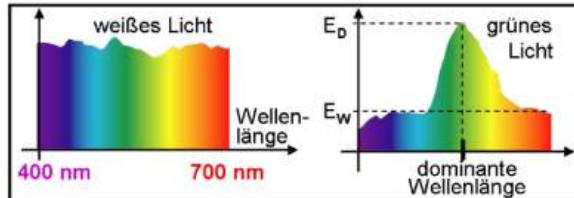
Unser Auge kann innerhalb des sichtbaren

Bereiches sogar unterscheiden, welche Frequenz die Strahlung hat, das empfinden wir dann als unterschiedliche Farben. Langwelligeres Licht (also niedrigere Frequenz) empfinden wir als rot, kurzwelligeres Licht (also höhere Frequenz) als blau bis violett. Dazwischen liegen alle Regenbogenfarben.



[zur Erinnerung:  $c = \lambda \cdot f$ , wobei  $c$  ... Lichtgeschwindigkeit,  $\lambda$  ... Wellenlänge,  $f$  ... Frequenz]

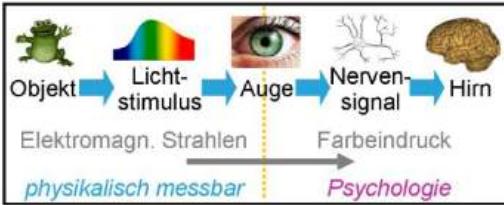
Tatsächlich kommt in der Natur aber höchst selten spektralreines Licht vor (das nur genau eine Wellenlänge hat), sondern meist sehen wir eine Mischung aus vielen Farben (Spektrum). Frequenzen mit mehr Energie bestimmen dann welche Farbe wir wahrnehmen, man spricht von dominanter Wellenlänge. Sind alle Anteile (ungefähr) gleich groß, so sehen wir ein farbloses Licht (also weiß oder grau). Wenn man mit  $E_D$  die Energie der dominanten Wellenlänge bezeichnet, und mit  $E_W$  die durchschnittliche Energie der anderen Wellenlängen, so nennt man  $(E_D - E_W)/E_D$  die Reinheit (purity) einer Farbe. Die Helligkeit ergibt sich als die Fläche (Integral) unter der Spektralkurve.



## Kolorimetrie

## Kolorimetrie

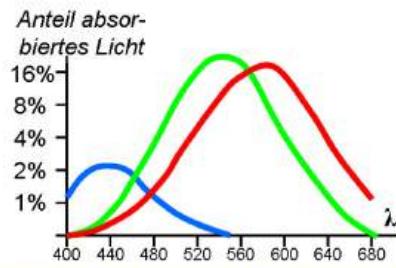
Die Kolorimetrie ist die Wissenschaft von der technischen Beschreibung von Farben. Man möchte also eine Farbe durch Zahlen, durch exakte Angaben beschreiben. Da aber eine Farbe ein empfundener Sinneseindruck ist, und keine physikalisch direkt messbare Größe, kann nur der visuelle Stimulus numerisch definiert werden (also das, was ein Mensch sieht), und zwar so dass



1. Stimuli mit den gleichen Spezifikationen unter gleichen Bedingungen gleich aussehen,
2. Stimuli die gleich aussehen die gleichen Spezifikationen haben,
3. die verwendeten Zahlen stetige Funktionen der physikalischen Parameter sind (d.h. kleine Änderungen der Zahlen bewirken kleine Änderungen der Farben und umgekehrt).

Kolorimetrie berücksichtigt also nur die *visuelle Unterscheidbarkeit* von elektromagnetischer Strahlung. Alle Spektren, die den gleichen Farbeindruck erzeugen, sind in diesem Sinn nicht unterscheidbar, bilden eine Äquivalenzklasse im Farbraum.

Die *Retina* des Auges, das ist die lichtempfindliche Schicht im hinteren inneren Bereich des Augapfels, enthält etwa 120 Millionen Stäbchen und Zapfen. Stäbchen können keine Farben unterscheiden, dafür sind sie sehr lichtempfindlich. Zapfen sind wesentlich weniger leicht aktivierbar, dafür gibt es drei verschiedene Arten, wobei jede Art in einem anderen Wellenlängenbereich empfindlich ist (die Empfindlichkeitskurven sind in der Graphik rechts abgebildet). Unser Farbempfinden setzt sich folglich aus der Kombination von drei getrennten „nicht-farbigen“ skalaren Signalen zusammen, daher bezeichnet man das menschliche Farbempfinden als *Tristimulus*. Die Empfindlichkeitskurven der drei Zapfenarten haben ihre Maxima bei Rot, Grün und Blau, es ist also durchaus angebracht, von Rot-, Grün und Blau-Zapfen zu sprechen. Erst das Gehirn mischt diese 3 Werte zu einer Farbe zusammen. Dies ist auch die Grundlage dafür, dass man dem Auge „alle“ Farben dadurch vorgaukeln kann, dass man eine Farbe aus nur 3 Grundfarben zusammensetzt. Wenn man kleine Lichtpunkte in rot, grün und blau nahe genug nebeneinander platziert, nehmen wir dies als einen Punkt in der so *additiv* gemischten Farbe wahr.



### The Human Eye

Werner Purgathofer

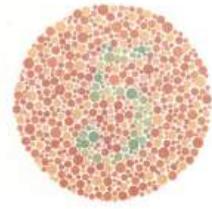
**TU WIEN**

retina contains:

- rods: black/white (grey)
- cones: color

## Farbfehlsichtigkeit:

Bei manchen Menschen fehlt erbbedingt eine Zapfenart (oder sogar zwei) oder es sind die Empfindlichkeitskurven der Zapfen nicht ausreichend verschieden, dann fehlt die Fähigkeit, so viele verschiedene Farben wie die meisten zu unterscheiden. Man spricht von *Farbschwäche* oder *Farblindheit*. Die häufigste Art ist Rot-Grün-Blindheit, bei der die Rot- und Grün-Zapfen auf zu ähnliche Wellenlängen reagieren. Etwa 8% aller Männer sind zumindest geringfügig farbfehlsehig! Testbilder, in denen die Information nur erkennbar ist, wenn man bestimmte (z.B. rötliche und grünliche) Töne gleicher Helligkeit unterscheiden kann (siehe Bild), dienen zur Diagnose von Farbfehlsehigkeit. Da man im Leben auch mit reduziertem Farbsehen sehr gut zurecht kommt, wissen viele Leute gar nichts von ihrer Einschränkung.



## Mögliche Beeinträchtigungen:

### **red/green blindness**

→ red & green cones too similar

### **blue blindness**

→ no blue cones

### **monochromatism**

→ all cones missing

mehr zum Auge: [2. Bildaufnahme](#)

## **Farbmodelle:**

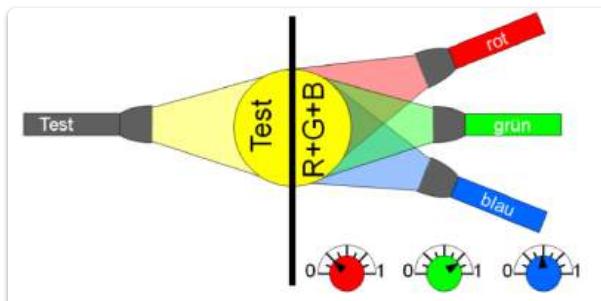
### **CIE 1931 XYZ-Farbmodell**

[EVC\\_Skriptum\\_CG, p.16](#)

**Grundlage:** Tristimulus-Theorie (Farbwahrnehmung durch 3 Zapfentypen).

**Experiment:** Farbvergleich mit 3 Grundfarben (Rot, Grün, Blau) zur Erzeugung einer Testfarbe.

- **Problem:** Negative Farbanteile nötig (Grundfarbe muss zur Testfarbe addiert werden).



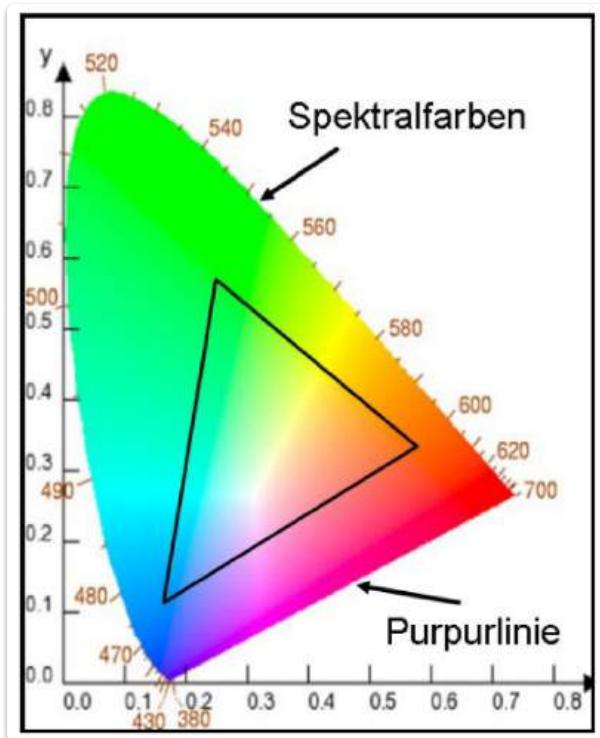
**Transformation:** Umwandlung in positive "imaginäre Grundfarben" X, Y, Z.

**CIE-Diagramm (1931):**

- Normierung auf Helligkeit 1.
- Projektion auf die **XY-Ebene** mit Koordinaten **(x, y)**.
- **z** ergibt sich aus  $x + y + z = 1$ .
- Vollständige Farbdefinition: **(x, y, Y)**, wobei **Y** die Helligkeit darstellt.

### Eigenschaften des CIE-Diagramms:

- **U-förmige Außenkante:** Spektralreine Farben (monochromatisches Licht).
- **Purpurlinie:** Verbindet die Endpunkte der U-Form, enthält Komplementärfarben spektraler Farben (keine reine Wellenlänge).
- **Jeder Punkt:** Représentiert eine andere Farbe.
- **Linearkombination zweier Farben:** Liegt auf der geraden Linie zwischen den Farben.
- **Weißpunkt:** Liegt etwa in der Mitte.
- **Komplementärfarben:** Liegen an entgegengesetzten Enden einer Geraden durch den Weißpunkt.
- **RGB-Monitor-Farbraum:** Darstellbare Farben liegen innerhalb des Dreiecks, das durch die Rot-, Grün- und Blau-Punkte des Monitors aufgespannt wird.
- **Begrenzung:** Kein Monitor kann alle sichtbaren Farben darstellen, da keine drei realen Farben das gesamte CIE-Diagramm abdecken.



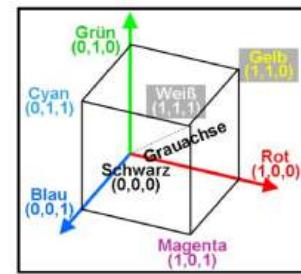
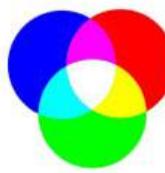
## RGB Farbmodell

---

## RGB-Farbmodell



Neben Farbräumen (eigentlich Farbraumbeschreibungen) wie dem CIE-Modell, die alle Farben zu beschreiben imstande sind, gibt es Farbräume zur Beschreibung der Farben eines Gerätes. Für Bildschirme wird fast immer



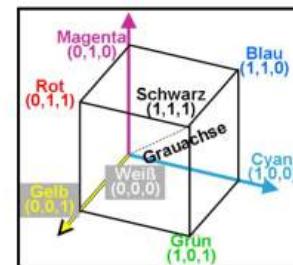
das RGB-Modell verwendet. Dabei wird ein Pixel aus drei kleinen Farb-punkten zusammengesetzt, deren Lichtsumme (*additive Farbmischung!* – siehe Skizze mit den Kreisen) einen Farbeindruck erzeugt. Je nach verwendeter Technologie und konkreten Materialien hat jeder Monitor geringfügig unterschiedliche Grundfarben, aus denen unterschiedliche Teilmengen aller Farben erzeugt werden können. Den Raum der Farben, die ein Gerät erzeugen kann, nennt man sein *Gamut*.

## CMY Farbmodell

### CMY-Farbmodell

Das *Mischen von farbiger Tinte auf einem Blatt Papier* unterliegt ganz anderen Regeln als die additive Farbmischung von Licht. Je mehr Tinte man verwendet, desto dunkler wird das Ergebnis, weil man ja eigentlich einen Filter vor das passiv reflektierende Papier bringt, daher spricht man von *subtraktiver Farbmischung* (siehe Skizze mit den Kreisen). Das CMY-Modell dazu ist das *Komplement des RGB-Raumes*. Für einfache Anwendungen gilt daher

$$[C, M, Y] = [1, 1, 1] - [R, G, B]$$

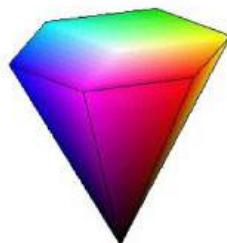


Vielfach kommt einem auch das *CMYK-Modell* unter. K steht dabei für Key, das entspricht der Farbe Schwarz. Beim Druck werden hierbei alle Grauanteile mit schwarzer Farbe extra gedruckt statt sie als Mischung gleicher Anteile von Cyan, Magenta und Yellow teurer und schlechter zu erzeugen.

## HSV- und HLS- Farbmodelle

Neben den für Geräte sinnvollen Farbräumen gibt es noch Beschreibungen der Farben in einer Weise, die dem *menschlichen Benutzer* entgegen kommt. Wir können nur sehr schwer und mit viel Übung eine Zielfarbe aus den Komponenten R, G, B oder C, M, Y beschreiben. Unsere üblichen Beschreibungen von Farben setzen sich aus Qualitäten wie einem Farbwort, einer *Helligkeit* und einer *Farbreinheit* zusammen. Daher werden für das User-Interface zur Farbdefinition solche Farbsysteme verwendet, die in diesen 3 Dimensionen funktionieren. Dazu gehören *HLS*, *HSV*, *Munsell*, *RAL*, *NCS*, *Coloroid* und einige andere.

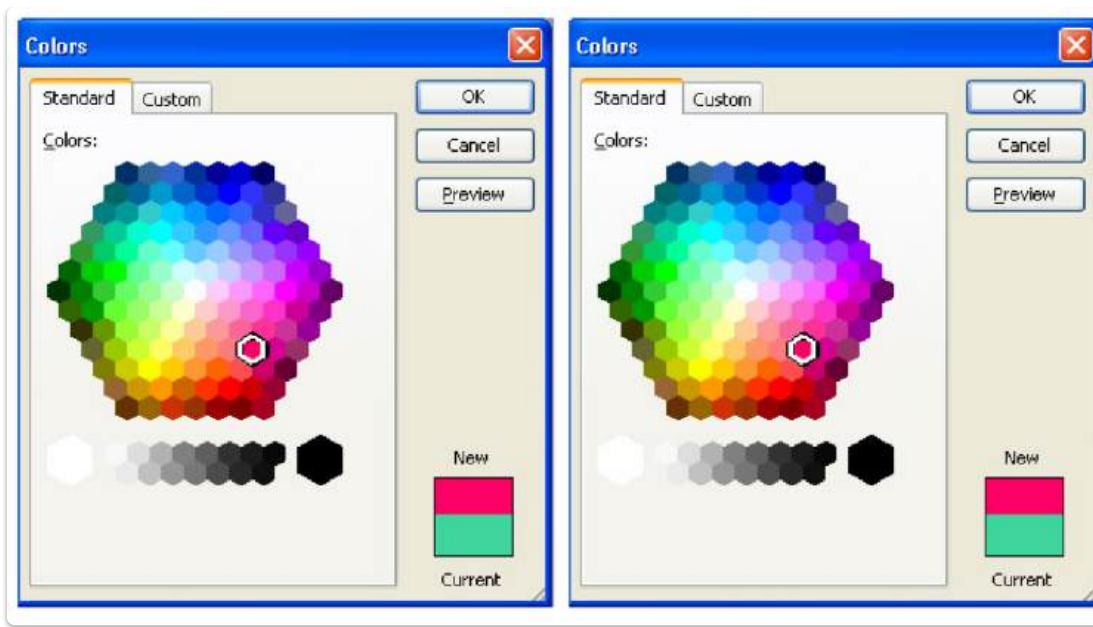
HSV steht für *Hue*, *Saturation* und *Value*. Hue heißt Bunnton oder Farbton, bezeichnet die Farbe entlang eines Farbkreises, der von Rot über Orange, Gelb, Grün, Cyan, Blau, Violet, Magenta wieder ins Rot geht. Wenn man den RGB-Würfel genau in Richtung seiner Grau-achse anschaut, so sieht man diesen Farbkreis als Grenze des entstehenden Sechsecks (Abbildung). Saturation heißt Sättigung und gibt an wie rein eine Farbe ist, wie stark sie sich also von Grau unterscheidet. Value heißt Wert und gibt so etwas wie die *Helligkeit* der Farbe an.



Je dunkler nun eine Farbe ist, desto weniger Abstufungen der Sättigung gibt es. Dadurch lassen sich alle Farben in einer Pyramide darstellen, deren Spitze schwarz ist und deren Grundfläche das Farb-Sechseck ist (Abb. links). Die Farbe wird in Grad entlang der Basiskante angegeben (Rot=0°, Grün=120°, Blau=240°), die Sättigung in Prozent des Abstandes von der Pyramidenachse und die Helligkeit als Prozent des Abstandes der Grundfläche von der Spitze. Ein mittelhelles gesättigtes Gelb hat damit den HSV-Wert (60, 1, 0.5).

Ganz ähnlich funktioniert das *HLS-System* (auch *HSL*), bei dem H=Hue, L=Lightness oder Luminance, S=Saturation heißen. Die Form des Modells ist jedoch diesmal ein Doppelkegel, der oben an der Spitze weiß ist und unten schwarz (Abb. rechts). Der Hintergrund ist die Annahme, dass Weiß viel heller ist als jede reine Farbe.





Meist hat man heutzutage in Desktopanwendungen mehrere verschiedene Farbmodelle zur Auswahl

---

## Farbsymbolik

Quelle: [EVC\\_Skriptum\\_CG](#), p.18

- Farben spielen in unserem Leben eine große Rolle.
- Die Verwendung von Farben kann zwischen verschiedenen Kulturen divergieren
- Manche Bedeutungen beziehen sich nur auf ein gewisses Gebiet.

## Sprachgebrauch

- **Grundvokabular:** Jede Sprache besitzt einen Kernbestand an Farbbezeichnungen.
- **Anzahl:** Variiert stark zwischen Sprachen (2 bis 20 grundlegende Termini).
- **Nuancen:** Zusätzlich existieren zahlreiche Bezeichnungen für Farbnuancen.
- **Deutsch:**
  - 6 bis 11 grundlegende Farbterme.
  - Ca. 150 bis 200 zusätzliche Bezeichnungen (z.B. oliv).
- **Andere Sprachen:**
  - **Italienisch:** Unterscheidung innerhalb einer Farbkategorie (z.B. *azzurro* für Himmelblau, *blu* für Dunkelblau).
  - **Ungarisch:** Unterscheidung innerhalb einer Farbkategorie (z.B. *piros* und *vörös* für Rot).
- **Fazit:** Die Kategorisierung und Benennung von Farben ist sprachabhängig und kann feiner oder gröber ausfallen.

## Farbe in der Religion

- **Symbolkraft:** Farben besitzen in der Spiritualität oft symbolische und kulturell/religiös bedeutsame Inhalte.
- **Heilige Farbe:** In vielen Weltreligionen (außer dem Christentum) existiert eine heilige Farbe.
- **Islam:**
  - **Grün:** Lieblingsfarbe des Propheten Mohammed.
  - **Bedeutung:** Oft auf islamischen Staatsflaggen vertreten (z.B. Saudi-Arabien).

## Farben in der Politik

---

- **Zuordnung:** Politische Strömungen und Parteien werden oft mit bestimmten Farben assoziiert.
- **Funktion:** Farben dienen als **einheitliches Erkennungsmerkmal**.
- **Beispiele:**
  - **Rot:** Marxismus-Leninismus, Sozialismus, Arbeiterbewegung.
  - **Grün:** Umweltorganisationen und -parteien.

## Kennzeichnung durch Farben

---

- **Alleinstehendes Merkmal:** Farbe selbst transportiert Bedeutung ohne zusätzliche Erklärung.
- **Beispiele:**
  - **Wasserhähne:** Rot (warm), Blau (kalt).

## Farben im Verkehr

---

- **Bewusste Information:** Farben in Verkehrszeichen, Lichtern und Ampeln sind codiert.
- **Rot/Weiß:** Verbots- und Gefahrenschilder.
- **Blau/Weiß:** Gebots- und Informationsschilder.
- **Ampel:** Rot (Stopp), Gelb (Vorsicht), Grün (Fahren).
- **Begrenzungslichter:** Rot (links vorbeifahren), Weiß (rechts vorbeifahren).

## Farben in der Technik

---

- **Beschreibung von Teilen:** Farben kennzeichnen spezifische Komponenten.
- **Beispiel (Elektrik):** Phase, Nullleiter, Erdung (durch Drahtfarbe).

## Farben in der Natur

---

- **Farbwirkung:** Natur nutzt Farben für verschiedene Zwecke.
- **Anlocken:** Buntes Balzgefieder/Schnäbel (Vögel).
- **Tarnung:** Anpassung an die Umgebung.

- **Warnung:** Abschreckung von Fressfeinden.

## Assoziationen zu Farben (kulturabhängig)

---

- **Blau:** Himmel, Weite, Ferne, Sehnsucht, Phantasie.
- **Rot:** Blut, Krieg, Tod, Lebenskraft, Leidenschaft, Liebe, Zorn.
- **Grün:** Wiesen, Wälder, Natur, "grüner Daumen", Hoffnung, Zuversicht.
- **Gelb:** Sommer, Sonne, Lebensfreude, Licht, Gold, aber auch Neid, Geiz, Eifersucht, Egoismus, Verlogenheit.
- **Schwarz:** Tod, Ende, Leere, Trauer (in "weißen" Kulturen), Freude (in manchen "dunklen" Hautfarben-Kulturen).
- **Weiß:** Vollkommenheit (in "weißen" Kulturen), Trauer (in manchen "dunklen" Hautfarben-Kulturen), Freude (in "weißen" Kulturen).



## 4. Punktoperationen

Quellen:

- [EVC\\_Skriptum\\_CV, p.20](#) bis [EVC\\_Skriptum\\_CV, p.23](#)

## Was sind Punktoperationen?

- **Definition:** Punktoperationen betreffen nur die Werte der einzelnen Bildelemente und verändern nicht die Größe, Geometrie oder Struktur des Bildes. Der neue Pixelwert hängt nur vom Wert des ursprünglichen Pixels an derselben Position ab.
- **Formel:** Der neue Wert  $I'(u, v)$  wird durch eine Funktion  $f$  des Originalwertes  $I(u, v)$  bestimmt:

$$I'(u, v) = f(I(u, v))$$

Der neue Wert hängt also nur von den ursprünglichen Pixelwerten und eventuell von konstanten Parametern ab.

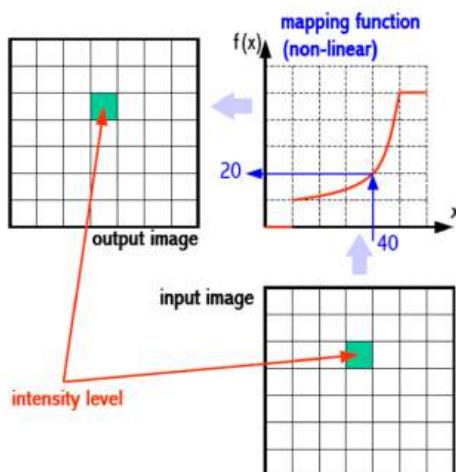


Abbildung 23: Beispiel Punktoperati-  
on

## Unterklassen der Punktoperatoren:

### 1. Homogene Punktoperatoren:

- Die Funktion  $f$  ist unabhängig von den Bildkoordinaten, d.h., die Operation ist für alle Bildpositionen gleich.
- Beispiele:
  - Helligkeits- und Kontraständerungen
  - Bildinvertierung

- Quantisierung der Bildhelligkeit
- Schwellwertbildung
- Gammakorrektur
- Farbtransformationen
- Diese Operationen verändern die Pixelwerte, aber nicht deren Position im Bild.

## 2. Inhomogene Punktoperationen:

- Diese Art der Punktoperation berücksichtigt zusätzlich die Bildkoordinaten  $(u, v)$ .
- Sie können beispielsweise bei der Verarbeitung von Bildteilen oder bei der Anwendung von Filtern mit Bildpositionen variieren.

### Affine Punktoperatoren:

- Diese sind eine spezielle Unterklasse der homogenen Punktoperatoren.
- Sie lassen sich durch eine lineare Gleichung beschreiben:

$$I'(u, v) = a \cdot I(u, v) + b$$

- $a$  und  $b$  sind Konstanten. Je nach Wahl von  $a$  und  $b$  kann man unterschiedliche Bildveränderungen erzielen:
  - **Helligkeitsveränderung:**  $a = 1, b \neq 0$
  - **Kontrastveränderung:**  $a \neq 1$

### Abbildungsfunktion (Mapping Function):

- Die Abbildungsfunktion ordnet jedem Pixelwert im Eingangsbild einen neuen Helligkeitswert im Ausgangsbild zu.
- Sie kann verschiedene Formen annehmen:
  - Linear
  - Stufenweise linear
  - Nichtlinear (z.B. Gammakorrektur, welche typischerweise eine nichtlineare Funktion darstellt).

### Beispiel einer nichtlinearen Abbildungsfunktion:

- In einem Beispiel könnte die Abbildungsfunktion dem Eingangswert 40 den Ausgangswert 20 zuordnen. Dies zeigt, dass die Zuordnung der Ausgangswerte nicht immer direkt proportional oder linear zum Eingangswert ist.

## Identitätsfunktion und Invertierung

Die einfachste Abbildungsfunktion ist die **Identitätsfunktion**, alle Werte behalten denselben Wert, die **Kennlinie verläuft von links unten nach rechts oben**. Die **Invertierung** ist eine einfache affine Punktoperation, die einerseits die Ordnung der Pixelwerte (durch **Multiplikation mit  $-1$** ) umkehrt und andererseits durch Addition eines konstanten Intensitätswerts dafür sorgt, dass das Ergebnis innerhalb des erlaubten Wertebereichs bleibt. Für ein Bild  $I(u, v)$  mit dem maximalen Wertebereich  $[0, q]$  ist die zugehörige Operation daher:  $I'(u, v) \rightarrow -I(u, v) + q = q - I(u, v)$ .



Abbildung 24: Identitätsfunktion und Invertierung

## Schwellwertoperation

Eine **Schwellwertoperation** (engl. *thresholding*) ist eine spezielle Form der Quantisierung, bei der die Bildwerte in zwei Klassen  $p_0$  und  $p_1$  getrennt werden, abhängig von einem vorgegebenen **Schwellwert** (engl. *threshold value*)  $p_{th}$ :

$$I'(u, v) \leftarrow f_{th}(I(u, v)) = \begin{cases} p_0 & \text{for } I(u, v) < p_{th} \\ p_1 & \text{for } I(u, v) \geq p_{th} \end{cases}$$

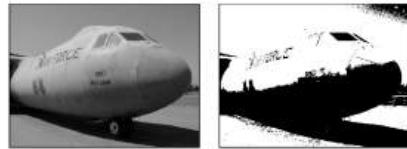


Abbildung 25: Schwellwertoperation

wobei  $0 < p_{th} \leq q$ . Eine häufige Anwendung ist die Binarisierung von **Grauwertbildern** mit  $p_0 = 0$  und  $p_1 = 1$ .

## Kontrast und Helligkeit

Um den **Kontrast in einem Bild um 50% (d.h. um den Faktor 1,5)** zu erhöhen, wird dies durch eine affine Punktoperation ausgedrückt:

$$I'(u, v) = I(u, v) \cdot 1.5$$

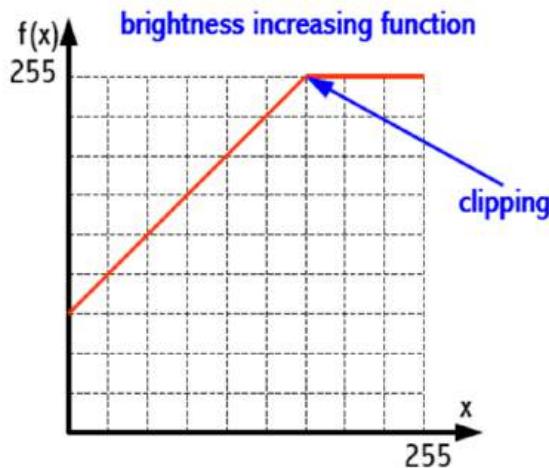
Hierbei wird der Pixelwert  $I(u, v)$  des ursprünglichen Bildes mit dem Kontrastfaktor 1.5 multipliziert, was den Kontrast im Bild entsprechend anhebt.

**Wichtiger Hinweis:** Bei der Anwendung solcher Operationen muss der vorgegebene Wertebereich der Bildpixel beachtet werden. Für **8-Bit-Grauwertbilder** liegt der zulässige Pixelwertbereich zwischen **0** und **255**.

## Problematik:

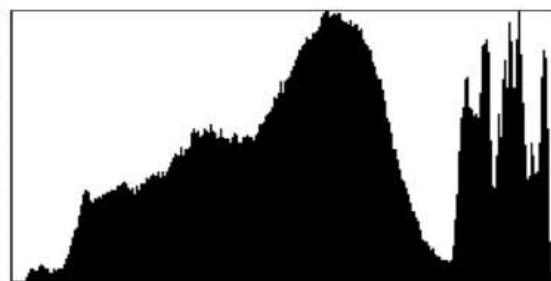
- **Clipping:** Wenn nach der Kontrastanhebung der berechnete Pixelwert **größer als 255** ist, wird dieser Wert auf den Maximalwert **255** begrenzt.
- Ebenso wird, wenn der Pixelwert **kleiner als 0** wird, dieser Wert auf den **Minimalwert 0** begrenzt, um negative Werte zu vermeiden.

Das Clipping verhindert, dass der Wertebereich überschritten wird, wodurch die Berechnungen im zulässigen Bereich bleiben. Diese Begrenzung (Clipping) ist notwendig, um eine korrekte Darstellung und Bildverarbeitung zu gewährleisten, da Bildpixel keine Werte außerhalb des zulässigen Bereichs annehmen können.



## Histogramm

- **Definition:** Ein Histogramm ist eine grafische Darstellung der Häufigkeit von Pixelwerten in einem Bild.
- **Verwendung:** Wird zur Analyse von **Belichtungsfehlern**, **Bildverarbeitungsschritten** und **Bildqualität** genutzt.
- **Funktionsweise:** Zeigt die Häufigkeit von Grauwerten oder Farbwerten auf der vertikalen Achse und die Grauwert- bzw. Farbintensitäten auf der horizontalen Achse.
- **Einschränkungen:** Es ist **nicht möglich**, das Originalbild ausschließlich aus dem Histogramm zu rekonstruieren, da viele verschiedene Bilder dasselbe Histogramm aufweisen können.



## Grauwert-Histogramme

- **Verwendung:** Besonders nützlich für **Graustufenbilder**.
- **Berechnung:** Zeigt, wie häufig jeder Grauwert  $x$  im Bild vorkommt.
- **Beispiel:** In einem 8-Bit-Bild ist das Histogramm von 0 bis 255 (für jeden Grauwert) unterteilt, wobei die vertikale Achse die Häufigkeit der Vorkommen dieses Grauwerts darstellt.

- **Verwendung in der Praxis:** Kann für die **Beurteilung von Belichtungsfehlern** oder die **Optimierung von Bildoperationen** verwendet werden.
- Definition:  $H(x) = \text{card}\{(u, v) | I(u, v) = x\}$  für jedes  $x \in \{0, \dots, q\}$

## Farbhistogramme

- **Verwendung:** Werden für **Farbbilder** verwendet, z. B. im **RGB-** oder **HSV-Farbmodell**.
- **Berechnung:** Hierbei wird die Häufigkeit der Pixelwerte für jeden Farbkanal berechnet.
- **Typen:** Häufig verwendete Varianten sind das **RGB-Histogramm** und das **HS-Histogramm**.
- **Einschränkungen:** Bei 24-Bit Farbbildern können die Histogramme sehr groß werden, daher werden oft **zweidimensionale** Histogramme verwendet (z. B. **HS-Histogramm** für den Farnton und die Sättigung).

## Auswirkungen von Bildoperationen auf Histogramme

- **Helligkeitserhöhung:** Verschiebt das gesamte Histogramm nach rechts.
- **Kontrasterhöhung:** Macht das Histogramm breiter und verteilt die Pixelwerte über einen größeren Bereich.
- **Invertieren des Bildes:** Spiegelt das Histogramm entlang der vertikalen Achse.
- **Verschmelzen von Histogrammlinien:** Führt zu einem Verlust von Bilddynamik und Information, wenn unterschiedliche Pixelwerte zusammengeführt werden.

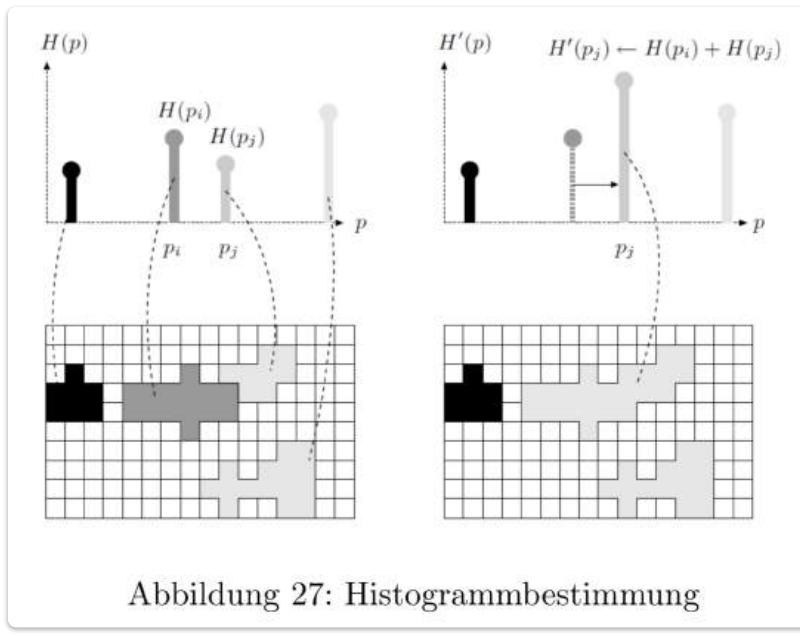


Abbildung 27: Histogrammbestimmung

## Verlust von Bildinformationen

- **Clipping:** Wenn Werte außerhalb des definierten Bereichs (z. B. 0-255 für 8-Bit-Graustufenbilder) berechnet werden, werden diese auf den maximalen oder minimalen Wert beschränkt.
- **Verlust durch Punktoperationen:** Wenn durch eine Bildoperation zwei Histogrammlinien zusammenfallen, werden diese Pixel nicht mehr voneinander unterschieden, was zu

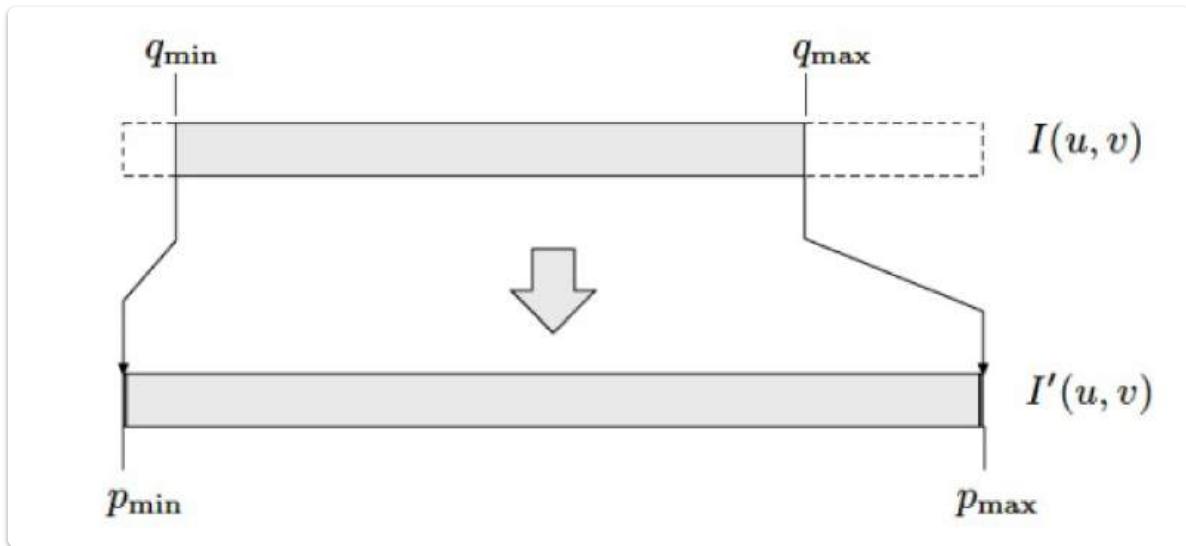
einem Verlust an Bilddynamik führen kann.

## Histogrammnormalisierung

- **Ziel:**
  - Erhöhung des Kontrasts durch lineare Umverteilung der Pixelwerte.
  - Maximale Ausnutzung des Intensitätsbereichs.
- **Prinzip:**
  - Dunkler Pixelwert ( $q_{min}$ ) → niedrigster Intensitätswert (0).
  - HELLSTER Pixelwert ( $q_{max}$ ) → höchster Intensitätswert (z. B. 255).
  - Lineare Verteilung der restlichen Pixelwerte dazwischen.
- **Formel zur Umrechnung:**

$$I'(u, v) = \frac{(I(u, v) - q_{min})}{(q_{max} - q_{min})} \cdot q$$

- $I(u, v)$ : Pixelwert im Originabild.
  - $q_{min}$ : Kleinster Pixelwert im Originalbild.
  - $q_{max}$ : Größter Pixelwert im Originalbild.
  - $q$ : Maximal möglicher Pixelwert im Zielbild (255 für 8-Bit Graustufenbilder).
  - $I'(u, v)$ : Neuer Pixelwert im Bild nach der Normalisierung.



- **Problem der geringen Robustheit:**
  - **Ausreißer** (ein Pixel mit Intensität 0 oder  $q$ ) beeinflussen die gesamte Streckung.
  - Das führt dazu, dass viele Pixelwerte in einem engen Bereich bleiben und die Transformation weniger effektiv wird.

## Vermeidung von Ausreißern durch Quantile

- **Lösung:** Bestimmung von  $q_{min}$  und  $q_{max}$  über **Quantile** statt extremen Werten.

- **p-Quantil:** Der Wert, unter dem  $p * 100\%$  der Werte einer Verteilung liegen.
  - Beispiel: **0.005-Quantil** (für 0.5% der Pixel an beiden Enden des Intensitätsbereichs).
- **Akkumuliertes Histogramm (Ha):**

$$Ha(x) = \sum_{k=0}^x H(k)$$

- **H(k):** Häufigkeit des Grauwerts **k** im Histogramm.
- **Anwendung:**
  - In Programmen wie **Adobe Photoshop** werden Ausreißer durch Quantile minimiert.
  - **Auto-Kontrast:** Häufige Anwendung der Histogrammnormalisierung mit Quantilen.

## Abbildung 28:

- Zeigt die Auswirkungen der Histogrammnormalisierung auf das Histogramm.
- **Regelmäßige Lücken** im Histogramm, die durch die lineare Streckung des Wertebereichs entstehen.

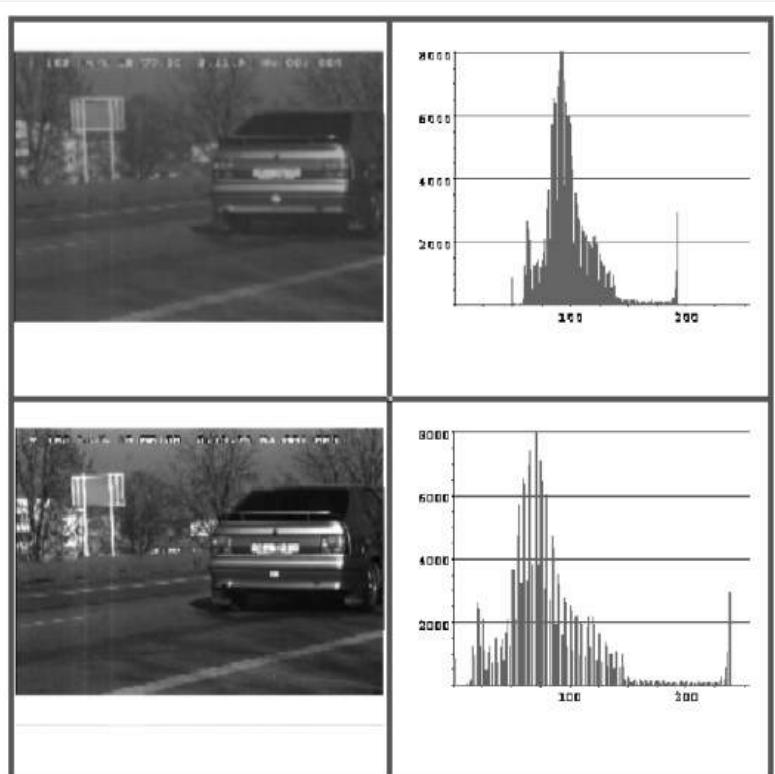


Abbildung 28: Histogrammnormalisierung

## Test-ähnliches Beispiel:

## Histogram Normalization Beispiel



- Angenommen, ein 10-Bit-Grauwertbild  $I(u,v)$  weist einen minimalen Intensitätswert von 50 und einen maximalen Intensitätswert von 200 auf. Wie lautet in diesem Fall die affine (lineare) Punktoperation, die den Kontrast des Bildes auf den gesamten Intensitätsbereich verstärkt?

$$I'(u,v) = q \cdot \frac{I(u,v) - q_{\min}}{q_{\max} - q_{\min}}$$

Einsetzen in Formel:

$q = 10\text{-bit} = 1024$  Grauwerte = **1023**

$q_{\min} = 50$

$q_{\max} = 200$

$$I'(u,v) = \frac{1023((I(u,v) - 50)}{150}$$

## Histogram Normalization Beispiel



- Angenommen, ein 10-Bit-Grauwertbild  $I(u,v)$  weist einen minimalen Intensitätswert von 50 und einen maximalen Intensitätswert von 200 auf. Wie lautet in diesem Fall die affine (lineare) Punktoperation, die den Kontrast des Bildes auf den gesamten Intensitätsbereich verstärkt?

2. Möglichkeit:

Lineares Gleichungssystem lösen:

$$y = kx + d$$

$$0 = k \cdot 50 + d$$

$$1023 = k \cdot 200 + d$$

$$\Rightarrow d = 341, k = -6,82$$

## Histogrammausgleich



- Ziel:**
  - Erzeugung eines **gleichmäßig verteilten Histogramms** im Ergebnisbild durch eine homogene Punktoperation.
  - Anwendung:** Verbesserung des Kontrasts in Bereichen mit stark vertretenen Grauwerten.
- Unterschied zur Histogrammnormalisierung:**
  - Histogrammausgleichung zielt darauf ab, **häufige Grauwertbereiche auseinanderzuziehen**

- Histogrammnormalisierung hingegen **streckt den gesamten Wertebereich gleichmäßig**.
- **Prinzip:**
  - **Ziel:** Annäherung an ein gleichverteiltes Histogramm (perfekte Gleichverteilung ist nicht möglich).
  - Einzelne **Spitzen im Histogramm** (stark konzentrierte Grauwertbereiche) können nicht vollständig entfernt werden, sondern werden nur auseinandergezogen.
  - **Veränderung des Kontrasts:** Es wird der Kontrast **in stark vertretenen Grauwertbereichen erhöht**.

## Berechnung des Histogrammausgleichs

- **Akkumuliertes Histogramm ( $H_a$ ):**
  - Das akkumulierte Histogramm wird verwendet, um das Bildkontrast zu modifizieren.
- **Normalisiertes, akkumuliertes Histogramm ( $H_n$ ):**
  - Berechnung:
$$H_n(x) = \frac{q}{H_a(q)} \cdot H_a(x)$$
  - **$H_a(q)$ :** Akkumuliertes Histogramm an der Stelle **q**.
  - **N:** Gesamtzahl der Pixel im Bild.
  - **q:** Maximaler Grauwert des Bildes (z. B. 255 bei 8-Bit).
- **Ziel:**
  - Durch diese Berechnung wird das akkumulierte Histogramm auf den **normalisierten Bereich zwischen 0 und q** abgebildet.

$H_n$  dient als Lookup-Table (Umsetzungstabelle: die Werte einer Funktion werden vorab ermittelt und als Tabelle abgelegt) für die **Neuzuordnung der Grauwerte**. Der Ausgleich bewirkt, dass jedem Pixel mit Grauwert  $x$  der  $p$ -te Anteil des maximal kodierbaren Grauwerts  $q$  zugeordnet wird. Dabei bezeichnet  $p$  die relative Häufigkeit, mit der alle Grauwerte von 0 bis einschließlich  $x$  im Eingabebild vorkommen. Die Histogrammequalisierung ist - im Gegensatz zur Spreizung und Histogrammdehnung - keine affine Punktoperation. In Abbildung 29 ist eine Histogrammequalisierung veranschaulicht. Im Gegensatz zur Histogrammnormalisierung sind die Lücken zwischen den Balken der Grauwerte nicht mehr gleichverteilt, sondern in Bereichen mit hoher Anzahl von Grauwerten im Eingabebild sind die Lücken größer als in Bereichen mit wenig Grauwerten. Das bewirkt eine **Kontrastverstärkung bei den Maxima und eine Kontrastabschwächung bei den Minima**, z.B. ist dadurch der Schatten der Bäume auf der Hausmauer in Abbildung 29 besser erkennbar.

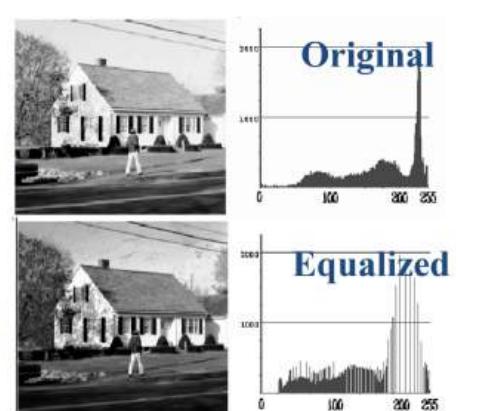


Abbildung 29: Histogrammausgleich  
Im Gegensatz zur Histogrammnormalisierung sind die Lücken zwischen den Balken der Grauwerte nicht mehr gleichverteilt, sondern in Bereichen mit hoher Anzahl von Grauwerten im Eingabebild sind die Lücken größer als in Bereichen mit wenig Grauwerten. Das bewirkt eine Kontrastverstärkung bei den Maxima und eine Kontrastabschwächung bei den Minima, z.B. ist dadurch der Schatten der Bäume auf der Hausmauer in Abbildung 29 besser erkennbar.



# 5. Rasterisierung

EVC\_Skriptum\_CG, p.19

## Was ist Rasterisierung?

- Prozess der **Umwandlung von Vektordaten** (z.B. Linien, Kurven) in **Pixel** zur Anzeige auf **bildschirmbasierten Ausgabegeräten**.
- Damit Grafiken angezeigt werden können, braucht es in der **Programmiersprache** entsprechende Befehle → sogenannte **Grafikprimitive**.

## Grafikprimitive – die Bausteine der Darstellung

### In 2D:

- **Punkte und Linien**
- **Polygone** (z.B. Dreiecke, Rechtecke)
- **Kreise, Ellipsen, Kurven** – auch in gefüllter Form
- **Bitmap-Operationen** (z.B. Bilder einfügen, verschieben)
- **Text** – Buchstaben, Zeichen, Zahlen

### In 3D:

- **Dreiecke und andere Polygone** – Grundelemente für 3D-Modelle
- **Freiformflächen** – z.B. Bézier-Flächen, Splines

## Zusätzliche Befehle zur Eigenschaftsdefinition

- Neben dem „Was zeichnen?“ braucht man auch „Wie?“
- Beispiele für **Eigenschaftsdefinitionen**:
  - **Farbe**
  - **Füllmuster**
  - **Textur** (Bild, das über Fläche gelegt wird)
  - **Materialeigenschaften** (für Beleuchtung, Glanz etc.)
  - **Transparenz**

### ⓘ Merke:

Diese Einstellungen gelten **global**, d.h. sie beeinflussen **alle danach gezeichneten Objekte**, bis sie erneut geändert werden.

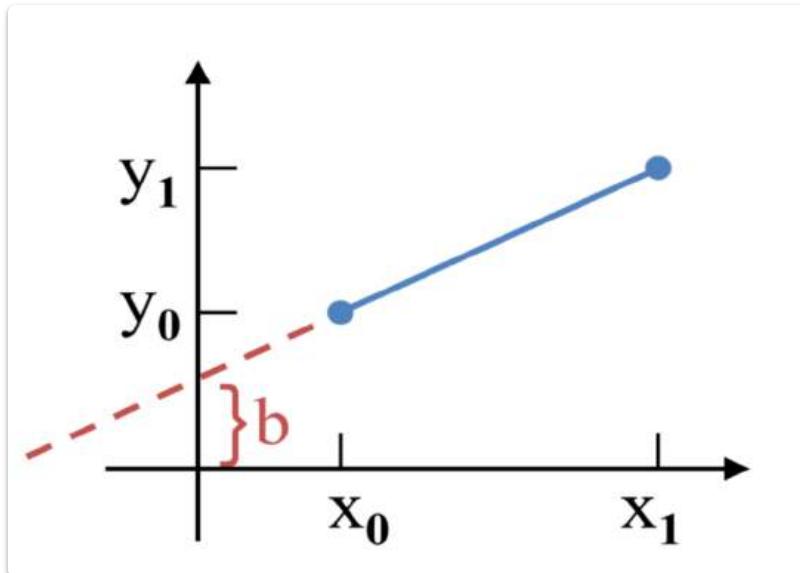
# Linienalgorithmen

Linien werden in der Form  $y = m * x + b$  angegeben wobei  $m$  den Anstieg beschreibt und  $(0, b)$  den Schnittpunkt der y Achse.

Aus Endpunkten  $(x_0, y_0)$  und  $(x_1, y_1)$  kann man sich  $m$  und  $b$  berechnen:

$$m = \frac{(y_1 - y_0)}{(x_1 - x_0)}$$

$$b = y_0 - m * x_0$$



## DDA-Verfahren

Der einfache DDA-Algorithmus für  $|m| < 1$  zählt zu  $y_0$  für jeden Schritt nach rechts ( $x+ = 1$ ) den Wert  $m$  dazu und rundet das Ergebnis danach auf ganze Zahlen. Dadurch entsteht eine Linie, bei der für jeden x-Wert genau ein Pixel für die Linie erzeugt wird.

```

1 dx = x1 - x0; dy = y1 - y0;
2 m = dy / dx;
3
4 x = x0; y = y0;
5 setPixel (round(x), round(y));
6
7 for (k = 0; k < dx; k++) {
8     x += 1; y += m;
9     setPixel (round(x), round(y))
10 }
```

Für  $|m| > 1$  werden  $x$  und  $y$  vertauscht, und das Verfahren wird in senkrechter Richtung durchgeführt. Auch der nachfolgende Bresenham-Algorithmus wird nur für  $0 < m < 1$  dargestellt, die anderen Richtungen erhält man durch Spiegelung und durch Rotation um  $90^\circ$ .

## DDA Line-Drawing Algorithm

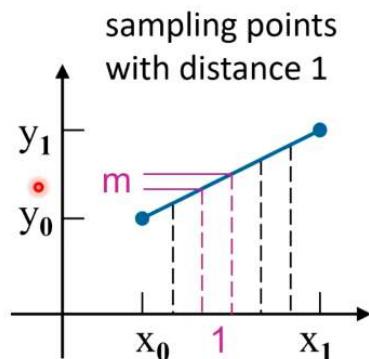


line equation:  $y = m \cdot x + b$

$$\delta y = m \cdot \delta x \quad \text{for } |m| < 1$$

$$\left( \delta x = \frac{\delta y}{m} \quad \text{for } |m| > 1 \right)$$

**DDA** (digital differential analyzer):



$$\text{for } \delta x=1, |m|<1 : y_{k+1} = y_k + m$$

## DDA – Based on Taylor Series Expansion



$$T_{f(x;a)} = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f'''(a)}{6}(x-a)^3 + \dots$$

$$f(x+1) = g(f(x), f'(x), f''(x), \dots) \quad (x+1)^n = \sum_{k=0}^n \binom{n}{k} x^k$$

$$f(x) = x^2, \quad f'(x) = 2x, \quad f''(x) = 2, \quad f'''(x) = 0$$

$$f(x+1) = (x+1)^2 = x^2 + 2x + 1 = f(x) + f'(x) + 1$$

$$f'(x+1) = 2x + 2 = f'(x) + 2$$

Eduard Gröller



## Bresenham-Verfahren

Man schaut ob man näher zum oberen oder unteren Pixel ist und setzt das dann so

Der **Bresenham-Algorithmus** erzeugt exakt dasselbe Ergebnis wie der einfache DDA, verwendet jedoch nur Integer-Arithmetik. Er ist dadurch schneller, leichter in Firm- oder Hardware zu implementieren, und überdies lässt er sich auch einfach für andere Kurven anpassen, z.B. Kreise, Ellipsen, Spline-Kurven usw.

Bei Linien lässt sich der nächste Punkt so berechnen:

$$y = m * (x_k + 1) + b$$

(lässt sich ganz einfach aus Linearen Funktionen erschließen)

Hier werden dann nicht die genauen  $y$  Werte berechnet sondern lediglich die Entscheidung getroffen, ob  $y_k$  oder  $y_{k+1}$  näher zum exakten  $y$ -Wert liegt.

Abstand zu  $y_k$  ist:

$$d_{lower} = y - y_k = m * (x_k + 1) + b - y_k$$

Abstand zu  $y_{k+1}$  ist:

$$d_{upper} = (y_k + 1) - y = y_k + 1 - m * (x_k + 1) - b$$

Nun berechnet man sich die Differenz zwischen  $d_{lower}$  und  $d_{upper}$ :

$$d_{lower} - d_{upper}$$

- Wenn diese Differenz negativ ist, dann nimmt man den unteren Punkt  $(x_{k+1}, y_k)$
- Wenn positiv den oberen  $(x_{k+1}, y_{k+1})$

## Optimierung durch Entscheidungsvariable

Um keine Fließkommaoperationen (Multiplikation/Division) durchführen zu müssen, wird eine **Entscheidungsvariable** eingeführt. Dazu setzt man:

$$m = \frac{\Delta y}{\Delta x} \quad \text{mit} \quad \Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0$$

Multipliziert man die obige Differenz mit  $\Delta x$ , ergibt sich:

$$p_k = \Delta x \cdot (d_{lower} - d_{upper}) = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

Diese Entscheidungsvariable hat dasselbe Vorzeichen wie  $d_{lower} - d_{upper}$ , benötigt aber keine Division mehr.

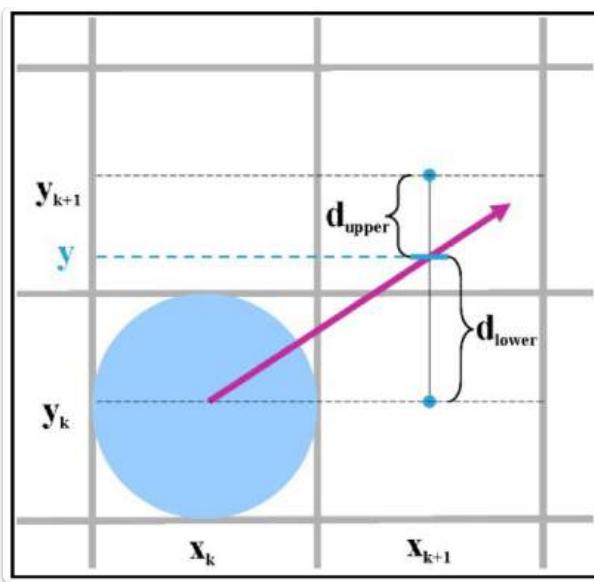
**Rekursive Berechnung** der nächsten Entscheidungsvariable:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x \cdot (y_{k+1} - y_k)$$

Das bedeutet: Die neue Entscheidungsvariable lässt sich **einfach aus der vorherigen berechnen**, je nachdem, ob  $y$  erhöht wurde oder nicht – also ganz ohne Neuberechnung des exakten  $y$ -Werts.

**Startwert:**

$$p_0 = 2\Delta y - \Delta x$$



## Attribute

Graphikprimitive können mit vielerlei Eigenschaften erzeugt werden, sogenannten Attributen.

### Attribute von Punkten und Linien

Neben allgemein bekannten Eigenschaften von Linien, wie Strichdicke, Strichlierungsmuster, Farbe oder Pinseltyp, gibt es noch ein paar Attribute, die einem oft weniger bewusst sind. Dazu gehören etwa die Liniendenenden bei breiteren Linien sowie die Form von Ecken bei breiten Linien:



Weiters ist Antialiasing auch für Linien ein Thema, dazu werden etwas weiter unten Details gebracht.

## Attribute von Text

[EVC\\_Skriptum\\_CG, p.21](#)

### Typische Eigenschaften von Text:

- **Font / Schriftart:**  
z. B. *Courier, Helvetica, Times, Fraktal*
- **Stil:**  
*normal, fett, kursiv, unterstrichen, durchgestrichen* etc.
- **Größe:**  
in Punkten angegeben (z. B. 12pt)
- **Richtung:**  
horizontal, vertikal, rotiert
- **Farbe:**  
z. B. RGB-Werte oder vordefinierte Farbnamen

- **Bündigkeit:**  
*links, rechts, zentriert, Blocksatz*

## Serifen vs. serifenos

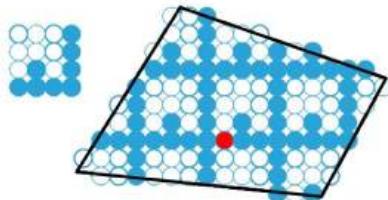
- **Fonts mit Serifen** (z.B. Times):
  - ✓ Besser geeignet für **Fließtext** – durch Serifen wird das Auge entlang der Zeile geführt.
- **Fonts ohne Serifen** (z.B. Helvetica):
  - ✓ Ideal für **plakativen Text**, Überschriften oder Bildschirmschirmdarstellung.



## Attribute von (2D-) Polygonen und Flächen

Klarerweise sind die **Attribute des Randes von Flächen** dieselben wie die von Linien. Dazu kommt nun die Fläche selbst, die mit einer Füllung versehen werden kann. Muster werden dabei gewöhnlich durch repetitive Aneinanderreihung eines Grundmusters ausgehend von einem Referenzpunkt (auch Seed-Point genannt) erzeugt.

In vielen Anwendungen ist es auch notwendig, eine **Kombination** des neu gezeichneten Musters mit dem Hintergrund zu erzeugen. Hier gibt es viele Varianten, die oft auf logischen Verknüpfungen aufbauen: **AND**, **OR**, **XOR**. Das Mischen von Farben erfolgt meist durch Linearkombination der vorhandenen Hintergrundfarbe  $B$  mit der zu zeichnenden Vordergrundfarbe  $F$ :  $P = t \cdot F + (1 - t) \cdot B$



## Baryzentrische Koordinaten

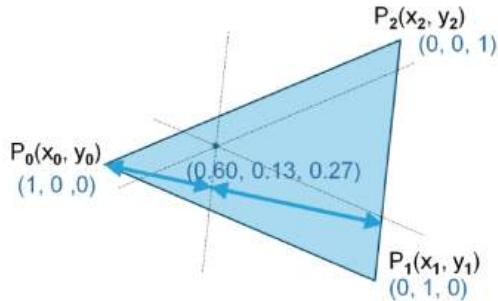
Baryzentrische Koordinaten sind eine Grundlage für die Interpolation von Pixeln in Dreiecken.

## Dreiecke rasterisieren

Um Dreiecke zu füllen verwendet man oft **baryzentrische Koordinaten**. Jeder Punkt der Ebene wird dabei als gewichtetes Mittel der drei Eckpunkte des Dreiecks dargestellt:

$$P = \alpha P_0 + \beta P_1 + \gamma P_2$$

$(\alpha, \beta, \gamma)$  nennt man dann die baryzentrischen Koordinaten des Punktes  $P$ , wobei immer gilt:  $\alpha + \beta + \gamma = 1$ . Alle Punkte mit  $(0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1)$  liegen innerhalb des Dreiecks; sobald einer dieser Werte negativ oder größer als 1 ist, liegt der Punkt außerhalb des Dreiecks.



Zum Füllen eines Dreiecks berechnet man für jedes Pixel einer (möglichst engen) Umgebung dessen baryzentrische Koordinaten und zeichnet alle für deren Mittelpunkt ( $0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1$ ) gilt. Dabei kann man sehr einfach beliebige Eckpunktattribute (z.B. Farbe) in jedem Pixel mit  $(\alpha, \beta, \gamma)$  gewichtet berechnen, dies entspricht einer linearen Interpolation dieser Werte.

**Baryzentrische Koordinaten** beschreiben einen Punkt  $P$  im Bezug auf die Eckpunkte eines Dreiecks  $P_0, P_1, P_2$ :

$$P = \alpha P_0 + \beta P_1 + \gamma P_2$$

mit der Bedingung:

$$\alpha + \beta + \gamma = 1$$

**Punkt liegt innerhalb des Dreiecks**, wenn gilt:

$$0 < \alpha < 1, \quad 0 < \beta < 1, \quad 0 < \gamma < 1$$

**Füllalgorithmus**:

- Für jedes Pixel im Bounding-Box-Rechteck des Dreiecks:
  - Baryzentrische Koordinaten  $(\alpha, \beta, \gamma)$  des Pixelmittelpunkts berechnen.
  - Liegt der Punkt **innerhalb** des Dreiecks? → **Pixel zeichnen**

**Vorteil**:

Mit  $\alpha, \beta, \gamma$  lassen sich beliebige **Attribute (z. B. Farbe, Tiefe, Textur)** linear interpolieren:

$$Attribut(P) = \alpha \cdot Attribut(P_0) + \beta \cdot Attribut(P_1) + \gamma \cdot Attribut(P_2)$$

## Beispiel der Baryzentrischen Koordinaten:

Angenommen, du hast ein Dreieck mit den Eckpunkten:

- A,
- B,
- C.

Dann sind die **baryzentrischen Koordinaten** für jeden Eckpunkt wie folgt:

**Eckpunkt A:**

→ (1, 0, 0)

Bedeutet: 100 % bei A, 0 % bei B, 0 % bei C → also exakt Punkt A.

## Eckpunkt B:

→ (0, 1, 0)

Bedeutet: 100 % bei B → also exakt Punkt B.

## Eckpunkt C:

→ (0, 0, 1)

Bedeutet: 100 % bei C → also exakt Punkt C.

Also:

- Die baryzentrischen Koordinaten  $(\alpha, \beta, \gamma)$  eines Punkts  $P$  im Dreieck erfüllen immer:  
 $P = \alpha A + \beta B + \gamma C$  mit  $\alpha + \beta + \gamma = 1$
- Für Punkte **innerhalb** des Dreiecks sind **alle drei Werte positiv**.
- Für Punkte **auf einer Kante** ist **eine Koordinate = 0**.
- Für die **Eckpunkte** ist **zwei Koordinaten = 0**.

General Polygon Fill Algorithms

**TU**  
WIEN

- *triangle rasterization*
- other polygons: what is inside?
- scan-line fill method
- flood fill method

barycentric  
coordinates!

clean borders  
between  
adjacent triangles?

Werner Purgathofer      31

## Berechnen der baryzentrischen Koordinaten

Sei  $g_{12}(x, y) = a_{12}x + b_{12}y + c_{12} = 0$  die Trägergerade durch die Punkte  $P_1$  und  $P_2$ , dann berechnet sich  $\alpha$  des Punktes  $P(x_p, y_p)$  zu  
 $\alpha = g_{12}(x_p, y_p) / g_{12}(x_0, y_0)$ .

$\beta$  und  $\gamma$  werden analog berechnet.

Um das doppelte Zeichnen der Kanten aneinander grenzender Dreiecke zu vermeiden, werden nur Pixel gezeichnet, deren Mittelpunkt innerhalb eines (exakten) Dreiecks liegen. Pixel genau auf einer Kante sind speziell zu behandeln, z.B. durch Regeln wie „Kanten unten und rechts werden gerendert, Kanten oben und links nicht“. Dadurch stellt man sicher, dass jedes Kantenpixel nur einmal behandelt wird.

Gegeben: ein Dreieck mit Eckpunkten  $P_0 = (x_0, y_0)$ ,  $P_1 = (x_1, y_1)$ ,  $P_2 = (x_2, y_2)$

Und ein beliebiger Punkt  $P = (x, y)$ , z.B. der Mittelpunkt eines Pixels

## Schritt 1: Trägergeraden der Dreiecksseiten

Zu jeder Seite des Dreiecks wird eine lineare Funktion  $g_{ij}(x, y)$  bestimmt, die null ist, wenn der Punkt auf der Geraden zwischen  $P_i$  und  $P_j$  liegt:

$$g_{ij}(x, y) = a_{ij}x + b_{ij}y + c_{ij}$$

Diese Gerade ist die Kante gegenüber von Punkt  $P_k$ .

Zum Beispiel:

- $g_{12}(x, y)$ : Gerade durch  $P_1$  und  $P_2 \rightarrow$  gegenüber von  $P_0 \rightarrow$  liefert  $\alpha$
- $g_{20}(x, y)$ : Gerade durch  $P_2$  und  $P_0 \rightarrow$  gegenüber von  $P_1 \rightarrow$  liefert  $\beta$
- $g_{01}(x, y)$ : Gerade durch  $P_0$  und  $P_1 \rightarrow$  gegenüber von  $P_2 \rightarrow$  liefert  $\gamma$

Die Formel für  $g_{ij}(x, y)$  ist:

$$g_{ij}(x, y) = (y_i - y_j)x + (x_j - x_i)y + (x_i y_j - x_j y_i)$$

## Schritt 2: Baryzentrische Koordinate berechnen

Um z.B.  $\alpha$  zu berechnen (also Anteil von  $P_0$ ), setzt man den Punkt  $P = (x, y)$  in  $g_{12}$  ein:

$$\alpha = \frac{g_{12}(x, y)}{g_{12}(x_0, y_0)}$$

Analog für  $\beta$  und  $\gamma$ :

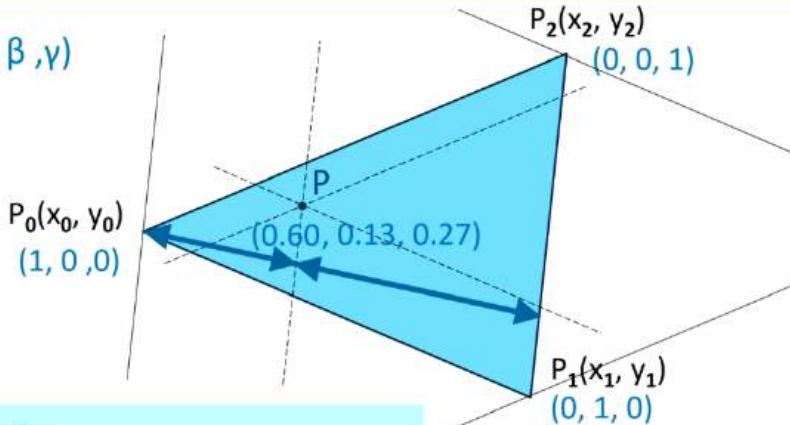
$$\beta = \frac{g_{20}(x, y)}{g_{20}(x_1, y_1)}, \quad \gamma = \frac{g_{01}(x, y)}{g_{01}(x_2, y_2)}$$

Der Nenner ist der jeweilige Wert, den die Gerade beim zugehörigen Eckpunkt liefert. Da die Punkte  $P_0, P_1, P_2$  nicht auf ihren gegenüberliegenden Seiten\*\* liegen, ist das kein Problem.

kompaktere Version hier: [5. FS - Rasterisierung > Barzentrische Koordinaten berechnen](#)

## Triangles: Barycentric Coordinates

notation:  $(\alpha, \beta, \gamma)$



$$P = \alpha P_0 + \beta P_1 + \gamma P_2$$

$$\text{triangle} = \{P \mid \alpha + \beta + \gamma = 1, 0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1\}$$

## Triangle Rasterization Algorithm



for all x

```

    for all y          /* use a bounding box! */
        {compute ( $\alpha, \beta, \gamma$ ) for (x,y) ;
         if ( $0 < \alpha < 1$ ) and ( $0 < \beta < 1$ ) and ( $0 < \gamma < 1$ )
         {
            draw pixel (x,y)
         }
    }
```

$$P = \alpha P_0 + \beta P_1 + \gamma P_2$$

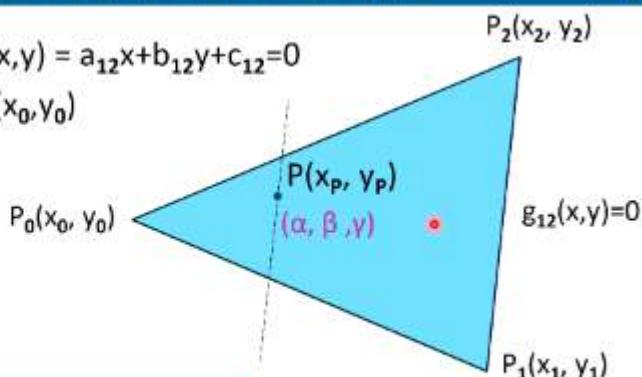
$$\text{triangle} = \{P \mid \alpha + \beta + \gamma = 1, 0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1\}$$

## Computing $(\alpha, \beta, \gamma)$ for $P(x_p, y_p)$



line through  $P_1, P_2$ :  $g_{12}(x, y) = a_{12}x + b_{12}y + c_{12} = 0$

then  $\alpha = g_{12}(x_p, y_p) / g_{12}(x_0, y_0)$

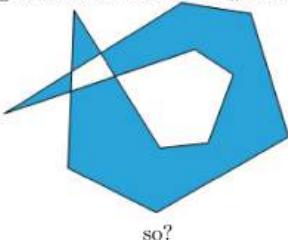


$$P = \alpha P_0 + \beta P_1 + \gamma P_2$$

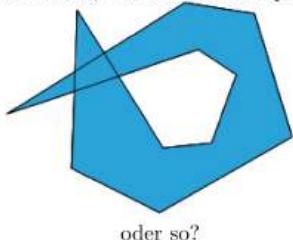
$$\text{triangle} = \{P \mid \alpha + \beta + \gamma = 1, 0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1\}$$

# Was ist in einem Polygon innen?

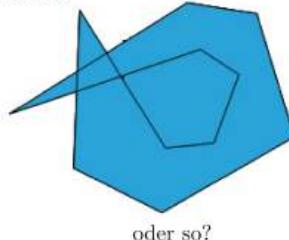
Bevor man mit dem Füllen von Flächen beginnt, muss man sich fragen, was denn zu füllen sei. Bei einer einfachen geschlossenen Kurve ist „innen“ leicht zu definieren, was aber bei komplizierteren Kurven?



so?

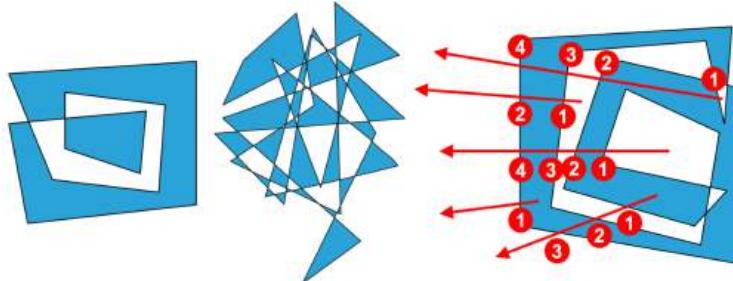


oder so?



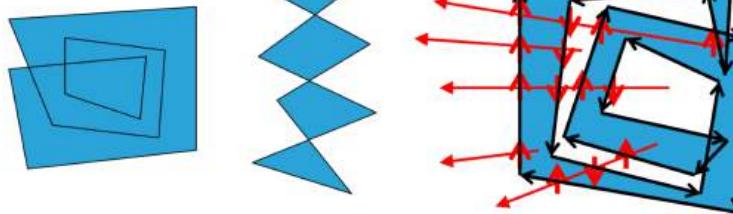
oder so?

**Odd-Even-Rule** Zieht man von einem Punkt aus einen beliebigen Halbstrahl, so ist der Punkt innerhalb, wenn die Zahl der Schnitte mit der Kurve ungerade ist, ansonsten ist der Punkt außerhalb (in Abb. oben links, sowie alle Bilder rechts). Jede Kante hat also eine Seite innen und die andere außen.



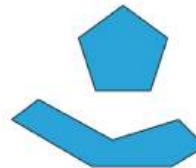
#### Nonzero-Winding-Number-Rule

Punkte sind außerhalb, wenn sich auf einem beliebigen Halbstrahl gleich viele im Uhrzeigersinn und gegen den Uhrzeigersinn verlaufende Kurvenkanten befinden, ansonsten innerhalb (in Abb. oben Mitte, sowie alle Bilder rechts).

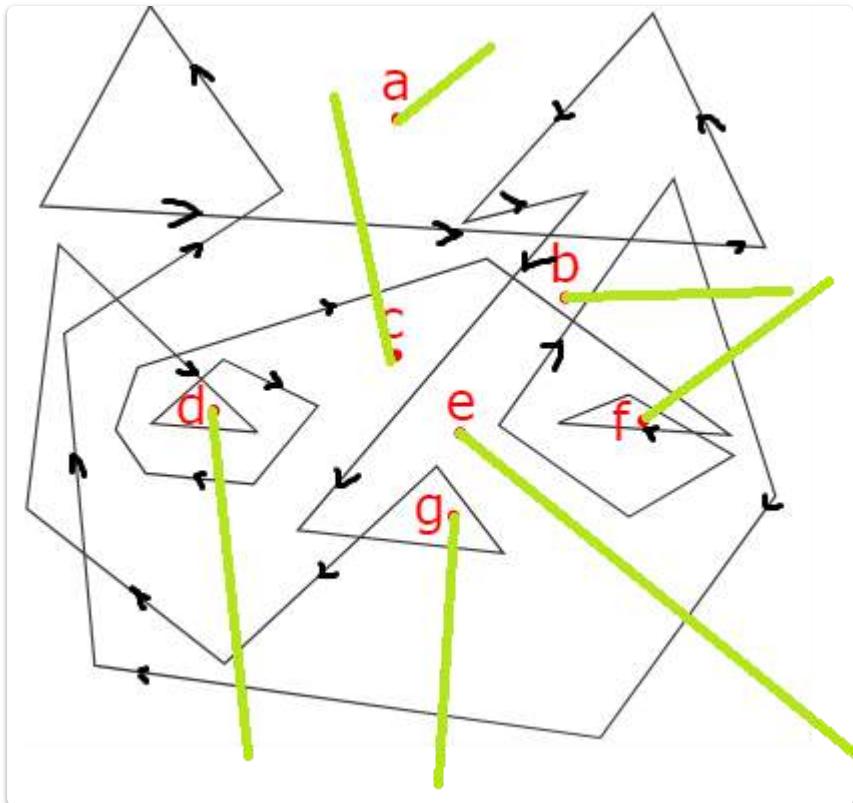


**All-In-Rule** Alles, was irgendwie umschlossen ist, ist innen. Wird selten verwendet, meist beim Pokern  $\odot$  (in Abb. oben rechts).

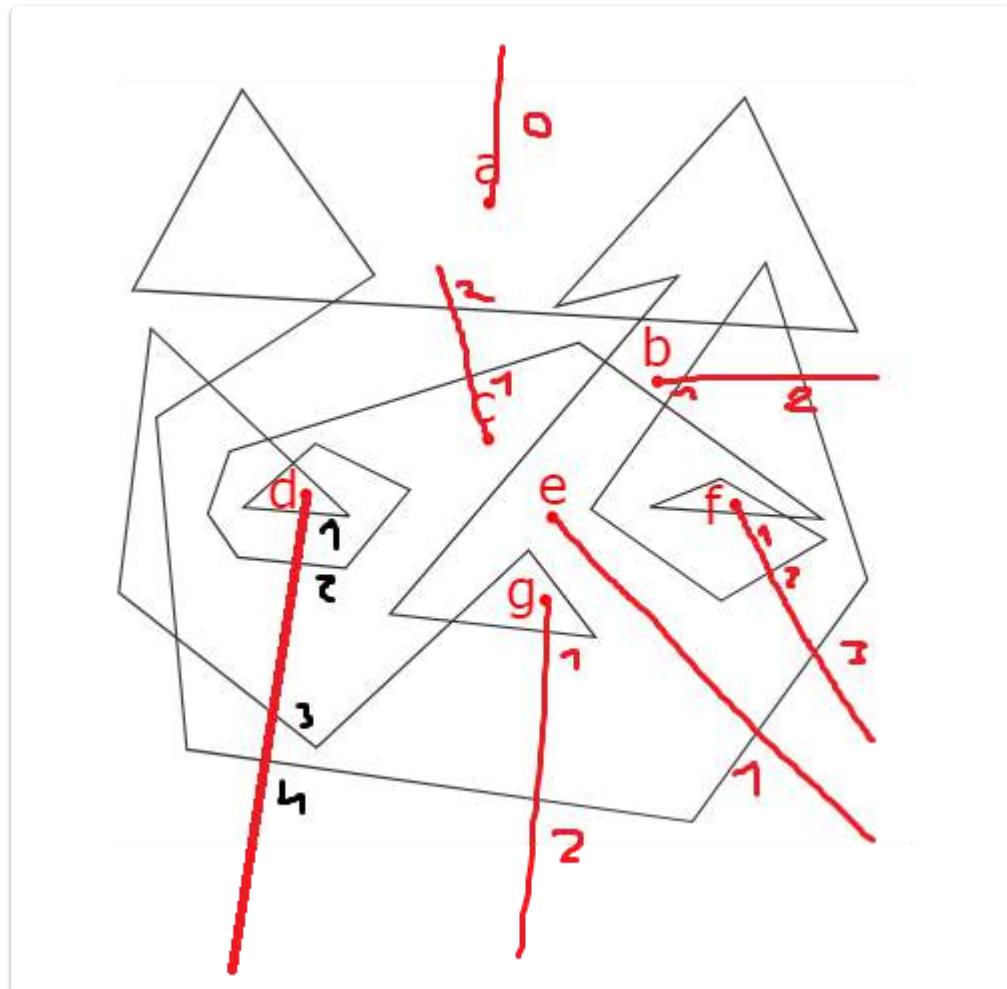
Ein Polygon heißt *konvex* wenn alle inneren Winkel kleiner als  $180^\circ$  sind (oberes Bild), andernfalls *konkav* (unteres Bild). Da konvexe Polygone viel weniger Sonderfälle erzeugen, sind viele Algorithmen für konvexe Polygone ausgelegt (oft sogar nur für Dreiecke). Daher braucht man auch Methoden um konkave Polygone in mehrere konvexe Polygone zu zerlegen (oft in Dreiecke).



35



Hier muss man Pfeile einzeichnen, wie man das Polynom als Oneliner zeichnen könnte, dann Linien von den Punkten nach draußen machen und schauen ob gleich viele in die eine Richtung wie in die andere Richtung gehen... Wenn nein dann innerhalb.



36.

Hier muss man zählen wie viele Kanten man überqueren muss um eine Linie ganz nach

außen ziehen zu können. Wenn diese Anzahl ungerade ist, dann liegt der Punkt in dem Polygon drinnen. Wenn nicht außerhalb

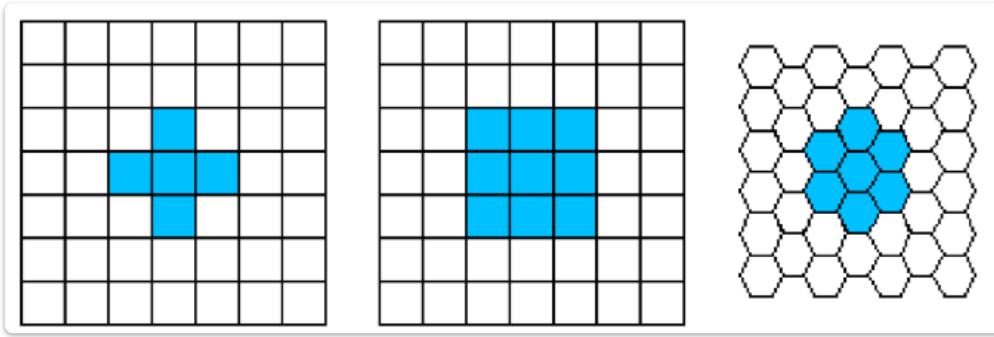


# 5. Lokale Operationen

Quellen:

- [EVC\\_Skriptum\\_CV, p.24](#) bis [EVC\\_Skriptum\\_CV, p.28](#)

## Nachbarschaften:



- Eine Nachbarschaft bezeichnet eine kleine, definierte Bildregion um ein Pixel, um Bildverarbeitungsoperationen durchzuführen.

### Vierer-Nachbarschaft

- Jedes Pixel  $P$  hat **2 horizontale** und **2 vertikale** Nachbarn.
- Koordinaten des Pixels  $P$ :  $(u, v)$ .
- Koordinaten der vier D-Nachbarn:  
 $(u - 1, v), (u + 1, v), (u, v - 1), (u, v + 1)$
- Eine Vierer-Nachbarschaft besteht aus **5 Punkten** (Pixel  $P$  + 4 D-Nachbarn).

### Achter-Nachbarschaft

- Neben den D-Nachbarn hat jedes Pixel  $P$  auch **4 diagonale Nachbarn**.
- Koordinaten der diagonalen Nachbarn:  
 $(u - 1, v - 1), (u + 1, v + 1), (u - 1, v + 1), (u + 1, v - 1)$
- Eine Achter-Nachbarschaft besteht aus **9 Punkten** (Pixel  $P$  + 4 D-Nachbarn + 4 diagonale Nachbarn).

### Abstand der Nachbarn

- Der Abstand der Nachbarn wird durch die **Metrik** festgelegt:
  - **Euklidische Metrik**: Abstand beträgt  $\sqrt{2}$ .
  - **Manhattan-Metrik**: Abstand beträgt 2.

# Was sind lokale Operationen

## Punktoperationen

- Der neue Wert eines Bildelements hängt ausschließlich vom ursprünglichen Bildwert an derselben Position ab.
- siehe [4. Punktoperationen](#)

## Lokale Operationen (Filter)

- **Ähnlichkeit zu Punktoperationen:** Auch hier besteht eine **1:1-Abbildung** der Bildkoordinaten, d. h., die Geometrie des Bildes bleibt unverändert.
- **Unterschied zu Punktoperationen:** Das Ergebnis wird nicht nur aus einem einzigen Ursprungspixel berechnet, sondern aus mehreren Pixeln des Originalbildes.
- Die Koordinaten der Quellpixel sind bezüglich der aktuellen Position  $(u, v)$  definiert und bilden eine zusammenhängende Region.

## Filterregion

- Die **Größe der Filterregion** bestimmt, wie viele ursprüngliche Pixel zur Berechnung des neuen Pixelwerts beitragen und damit das **räumliche Ausmaß des Filters**.
- Eine gängige Filtergröße ist  $3 \times 3$ , zentriert in der Achter-Nachbarschaft um die Koordinate  $(u, v)$ .
- Die Form der Filterregion muss nicht quadratisch sein, sondern kann beliebige Formen annehmen.

# Lineare Filter

- **Bezeichnung:** Lineare Filter verbinden die Pixelwerte innerhalb der Filterregion in **linearer Form**, d. h., durch eine gewichtete Summation.
- **Beispiel:** Die **lokale Mittelwertbildung** ist ein einfaches Beispiel, bei dem alle neun Pixel der  $3 \times 3$  Filterregion mit der Gewichtung 1/9 summiert werden.

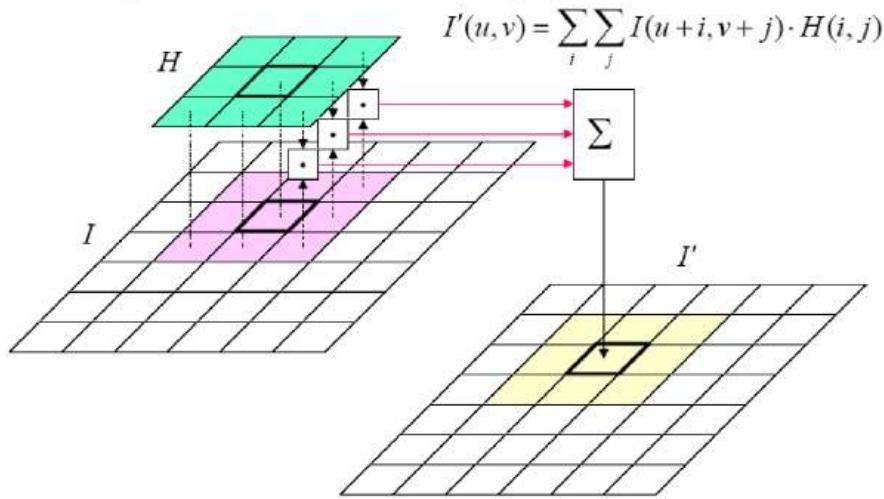


Abbildung 30: Berechnung einer Faltung (Convolution)

## Filtermatrix

- **Definition:** Eine **Filtermatrix** oder **Filtermaske**  $F(i, j)$  spezifiziert die Größe, Form und die zugehörigen **Gewichte** der Filterregion.
  - Die Größe der Matrix entspricht der Größe der Filterregion.
  - Jedes Element  $F(i, j)$  der Matrix definiert das Gewicht des entsprechenden Pixels.
- **Einzigartigkeit:** Das Ergebnis eines linearen Filters ist eindeutig und vollständig durch die Koeffizienten in der Filtermatrix bestimmt.

## Anwendung des Filters

- Die Anwendung eines linearen Filters auf ein Bild erfolgt durch folgende Schritte:
  1. **Positionierung der Filterfunktion  $F$ :** Die Filtermatrix  $F$  wird so über das Bild  $I$  positioniert, dass ihr Koordinatenursprung  $F(0, 0)$  auf das aktuelle Bildelement  $I(u, v)$  fällt.
  2. **Multiplikation und Summation:** Alle Bildelemente in der Filterregion werden mit den jeweils darüber liegenden Filterkoeffizienten multipliziert und die Ergebnisse werden summiert.
  3. **Speichern des Ergebnisses:** Die resultierende Summe wird an der entsprechenden Position im Ergebnisbild  $I'(u, v)$  gespeichert.

## Berechnung für den $3 \times 3$ Filter

- Für einen  $3 \times 3$  Filter des neuen Bildes  $I'(u, v)$  wird der Wert für jedes Pixel wie folgt berechnet:
  - Die Schritte 1–3 werden an jeder Position  $(u, v)$  im Bild wiederholt, um das gefilterte Bild zu erhalten.

# Tiefpassfilter

## Unterscheidung zwischen Tiefpass- und Hochpassfiltern

- **Tiefpassfilter:**
  - Filtern **hohe Frequenzen** heraus und lassen **niedrige Frequenzen** passieren.
  - Eignen sich für **Rauschunterdrückung** bzw. als **Glättungsoperatoren**.
  - Bekannte Tiefpassfilter: **Mittelwertfilter** und **Gauß-Filter**.
- **Hochpassfilter:**
  - Filtern **tiefe Frequenzen** heraus und lassen **hohe Frequenzen** passieren.
  - Eignen sich z. B. für die **Kantendetektion**. (siehe [8. Bildmerkmale - Interest Points](#))

## Mittelwertfilter (Box-Filter)

- **Filtermaske:** Besteht aus lauter gleichen Gewichten (1), einfachste Form aller Tiefpassfilter.
- **Nachteile:**
  - Scharf abfallende Ränder und unoptimales Frequenzverhalten.
  - Alle Bildelemente haben das gleiche Gewicht, wodurch das Zentrum nicht stärker gewichtet wird als die Ränder.

## Gauß-Filter

- **Filtermaske:** Entspricht einer diskreten, zweidimensionalen **Gauß-Funktion**.
  - Beispiel für eine Filtermaske (für  $\sigma = 0.5$ ):
$$F_{Gauss} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$
- **Eigenschaften:**
  - Das mittlere Bildelement erhält das **maximale Gewicht**.
  - Die Werte der übrigen Koeffizienten nehmen mit zunehmender Entfernung zur Mitte **kontinuierlich und gleichmäßig** ab (isotrop).
  - **Standardabweichung  $\sigma$**  bestimmt den „Radius“ der glockenförmigen Funktion und beeinflusst die Stärke der Glättung.

## Filtermaske für einen $3 \times 3$ Gauß-Filter mit $\sigma = 0.5$

- Die resultierende Filtermaske lautet:
 
$$F_{Gauss} = \begin{bmatrix} 0.011 & 0.084 & 0.011 \\ 0.084 & 0.619 & 0.084 \\ 0.011 & 0.084 & 0.011 \end{bmatrix}$$
- **Summe der Koeffizienten** muss 1 betragen, was durch Division aller Koeffizienten durch deren Summe erreicht wird.

## Wichtige Hinweise

- Größere Filtermasken führen zu einer besseren Approximation der Gauß-Funktion, aber ändern nicht das Glättungsverhalten.
  - Die Stärke der Glättung kann durch die Standardabweichung  $\sigma$  variiert werden.
  - Ein Mittelwertfilter mit einer  $3 \times 3$  Filtermaske führt zu einem befriedigenden Ergebnis, aber der Gauß-Filter wird im Allgemeinen bevorzugt.
- 

## Differenzfilter

### Interpretation negativer Filterkoeffizienten

- Wenn einzelne Filterkoeffizienten negativ sind, kann die Filteroperation als Differenz zweier gewichteter Summen verstanden werden:  
Summe positiver Gewichtungen – Summe negativer Gewichtungen
- Innerhalb der Filterregion  $R$  werden:
  - Positive Koeffizienten → positiv gewichtete Pixel.
  - Negative Koeffizienten → negativ gewichtete Pixel.

### Beispiel: Laplace-Filter

- Filtermatrix:
 
$$F_{Laplace} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
- Berechnet die Differenz zwischen dem zentralen Pixel ( $-4$ ) und den 4 umliegenden Pixeln (Vierer-Nachbarschaft).
- Übrige 4 Pixel (diagonale Nachbarn) haben Koeffizienten 0 → werden nicht berücksichtigt.

### Wirkung der Differenzbildung

- Gegenteil zur Durchschnittsbildung:
  - Durchschnitt → Glättung von Intensitätsunterschieden.
  - Differenz → Verstärkung von Intensitätsunterschieden.
- Anwendungen:
  - Kanten- und Konturverstärkung
  - Bildschärfung
- → Differenzfilter sind Hochpassfilter.

### Mathematische Grundlage

- Hochpassfilter basieren auf Ableitungen der Bildfunktion  $g(x, y)$ :
  - Erste Ableitung → Gradientenfilter

- Zweite Ableitung → Laplace-Filter

## Nachbearbeitung des Ergebnisbildes

- Ergebnis enthält oft **positive und negative Grauwerte**.
- Mögliche Nachbearbeitungen:
  - Normierung auf z.B. [0, 255]
  - Betragsbildung:  $|g(x, y)|$

## Anwendung in Software (z.B. Adobe Photoshop)

- Umsetzung durch sogenannte „**Custom Filter**“
  - Filter mit:
    - **Ganzzahligen Koeffizienten**
    - **Skalierungsfaktor (Scale)**
    - **Offset-Wert**, um negative Ergebnisse in den **sichtbaren Intensitätsbereich** zu verschieben.
- 

## Bildrandproblem

- Beim Anwenden von Filtern kann es zu **Problemen an den Bildrändern** kommen.
- Ursache: Die **Filterregion überschreitet den Bildbereich**, es fehlen passende Pixelwerte → das Filterergebnis **kann nicht berechnet werden**.
- Es gibt keine mathematisch exakte Lösung für das Problem
- andere Probleme mit Bildrand siehe: [7. Clipping und Antialiasing](#)

## Methoden zum Umgang mit dem Randproblem

### 1. Einsetzen eines konstanten Werts

- Beispiel: 0 (schwarz)
- **Nachteil**: Verkleinert den sichtbaren Bildbereich.
- **Nicht akzeptabel** in den meisten Anwendungen.

### 2. Beibehalten der ursprünglichen (unfiltrierten) Bildwerte

- Filter wird **nicht auf die Randpixel angewendet**.
- **Nachteil**: Inkonsistente Bildverarbeitung; ebenfalls nicht ideal.

### 3. Annahme künstlicher Pixelwerte außerhalb des Bildbereichs:

- (a) **Konstanter Wert** außerhalb des Bildes (z.B. schwarz oder grau)
  - Kann bei großen Filtern zu **starken Verfälschungen an den Rändern** führen.
- (b) **Fortsetzung der Randpixel**
  - Randwerte des Bildes werden **nach außen hin fortgeführt**.
  - **Geringe Verfälschung** → bevorzugte Methode

- (c) **Zyklische Wiederholung des Bildes**
    - Das Bild wird horizontal und vertikal periodisch fortgesetzt.
- 

# Formale Eigenschaften lineare Filter

## Ursprung und Definition

- **Lineare Filter** basieren auf dem mathematischen Konzept der **linearen Faltung** (engl. *linear convolution*).
- Sie verknüpft zwei **Funktionen gleicher Dimensionalität**, kontinuierlich oder diskret.
- Für diskrete, 2D-Funktionen  $I$  (Bild) und  $F$  (Filtermatrix) ist die Faltung definiert als:  
 $I' = I * FI'$
- Dabei gilt (mit Berücksichtigung der Koordinatenumkehr):

$$I'(u, v) = \sum_i \sum_j I(u - i, v - j) \cdot F(-i, -j)$$

- Die ursprüngliche lineare Filterdefinition entspricht einer **linearen Korrelation**, da hier **keine Spiegelung** der Filtermatrix erfolgt.

## Eigenschaften der linearen Faltung

### 1. Kommutativität:

$$I * F = F * I$$

→ Reihenfolge von Bild und Filter spielt **keine Rolle**.

### 2. Linearität:

- Skalierung eines Bildes:

$$(a \cdot I) * F = a \cdot (I * F)$$

- Addition zweier Bilder:

$$(I_1 + I_2) * F = (I_1 * F) + (I_2 * F) \quad (I_1 + I_2) * F = (I_1 * F) + (I_2 * F)$$

### 3. Assoziativität:

$$A * (B * C) = (A * B) * C$$

→ Filter können **beliebig kombiniert** und **umgruppiert** werden.

## Konsequenz: Separierbarkeit

- Ein Filterkern  $F$  kann als **Faltungsprodukt kleinerer Filterkerne** beschrieben werden:  
 $F = F_1 * F_2 * \dots * F_n$
- Besonders nützlich: Trennung in **zwei eindimensionale Filter**:

Beispiel:

- $F_x = [1 \ 2 \ 1]$

- $F_y = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

Kombiniert:

$$F_{xy} = F_x * F_y$$

- Vorteil: **Reduktion der Rechenkomplexität**
    - Normal:  $3 \times 5 = 15$  Operationen pro Pixel
    - Separiert:  $5 + 3 = 8$  Operationen pro Pixel
- 

## Nicht lineare Filter

### Nachteil linearer Filter

- Lineare Filter **glätten** nicht nur **Störungen**, sondern auch **gewollte Bildstrukturen** wie:
  - Punkte
  - Kanten
  - Linien
- **Bildqualität leidet**: Strukturen werden verwischt.
- Einschränkung ihrer Anwendung bei Struktur- oder Kantenerhaltung.

### Rangordnungsfilter (engl. *rank value filters*)

- Nichtlineare Filteroperationen**
- Kombination von benachbarten Pixeln durch **Vergleichen und Selektieren**, statt **Gewichten und Addieren**.
- Funktionsweise:
  - Alle Grauwerte innerhalb der Filtermaske werden **sortiert** (aufsteigend).
  - Es wird **ein bestimmter Rang** aus dieser Liste ausgewählt.
  - Dieser Wert ersetzt das zentrale Pixel.

### Typen von Rangordnungsfiltern

- Medianfilter**:
  - Wählt den **mittleren Wert** (Median) der sortierten Grauwerte.
  - Besonders wirksam bei der **Entfernung von Ausreißern** (z. B. Salz-und-Pfeffer-Rauschen).
- Minimumfilter**:
  - Wählt den **kleinsten** Grauwert in der Region.
- Maximumfilter**:
  - Wählt den **größten** Grauwert in der Region.

## Vorteile

- Besser geeignet zur **Erhaltung von Kanten und feinen Strukturen**.
- Besonders effektiv bei **nicht-gausschem Rauschen**.

## Definition dieser Filter:

$I(u, v) = \min\{I(u + i, v + j) \text{ für } (i, j) \in R\} = \min(R_{u,v})$  bzw.  $I(u, v) = \max\{I(u + i, v + j) \text{ für } (i, j) \in R\} = \max(R_{u,v})$ , wobei  $R_{u,v}$  die Region der Bildwerte bezeichnet, die an der aktuellen Position  $(u, v)$  von der Filterregion überdeckt werden. Die Abbildung 31 zeigt die Anwendung von 3x3-Min- und -Max-Filters auf ein Grauwertbild, das künstlich mit SSalt-and-PepperSStörungen versehen wurde (zufällig platzierte weiße und schwarze Punkte). Der

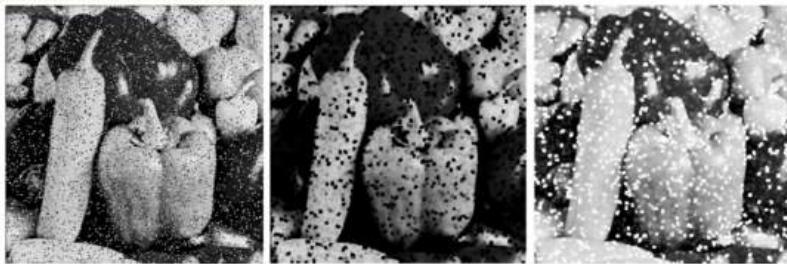
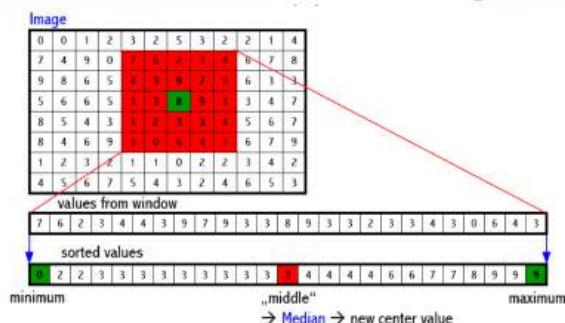


Abbildung 31: Min-/Max-Filter Anwendung

Min-Filter entfernt die weißen (Salt) Punkte, denn ein einzelnes, weißes Pixel wird innerhalb der 3x3-Filterregion  $R$  immer von kleineren Werten umgeben, von denen einer den Minimalwert liefert. Gleichzeitig werden durch das Min-Filter aber dunkle Strukturen räumlich erweitert. Der Max-Filter hat genau den gegenteiligen Effekt.



## Grundproblem bei der Filterung

- Kein Filter kann *automatisch* unterscheiden zwischen:
  - wichtigen Strukturen (z.B. Kanten, Details)
  - unerwünschten Störungen (z.B. Rauschen)
- → Perfekter Filter existiert nicht.
- Jeder Filter trifft eine "blinde Entscheidung", ob ein Pixel zur Struktur oder zur Störung gehört.

## Medianfilter – ein sinnvoller Kompromiss

- Ziel: **Störungen entfernen**, aber **Strukturen besser erhalten** als bei linearen Filtern.
- **Definition:**  
Jedes Bildelement  $I(u, v)$  wird durch den **Median** der Pixelwerte innerhalb einer Filterregion  $R$  ersetzt:  $I(u, v) = median(R_{u,v})$

- Der **Median** aus einer sortierten Liste von  $2K + 1$  Pixelwerten  $p_i$  ist der mittlere Wert:  

$$\text{median}(p_0, \dots, p_{2K}) = p_K \text{ (sofern } p_i \leq p_{i+1} \text{)}$$

## Beispielhafte Wirkung

- Abbildung 32** (gedanklich):
  - Linkes Bild:** Originalbild mit **Salt-and-Pepper-Rauschen**.
  - Mittleres Bild:** Nach Anwendung eines **Mittelwertfilters** – Störungen teilweise noch sichtbar.
  - Rechtes Bild:** Nach Anwendung eines **Medianfilters** – Störungen **besser entfernt**, Strukturen **erkennbar erhalten**.



Abbildung 32: Anwendung von Mittelwert- und Medianfilter

## Vorteile des Medianfilters

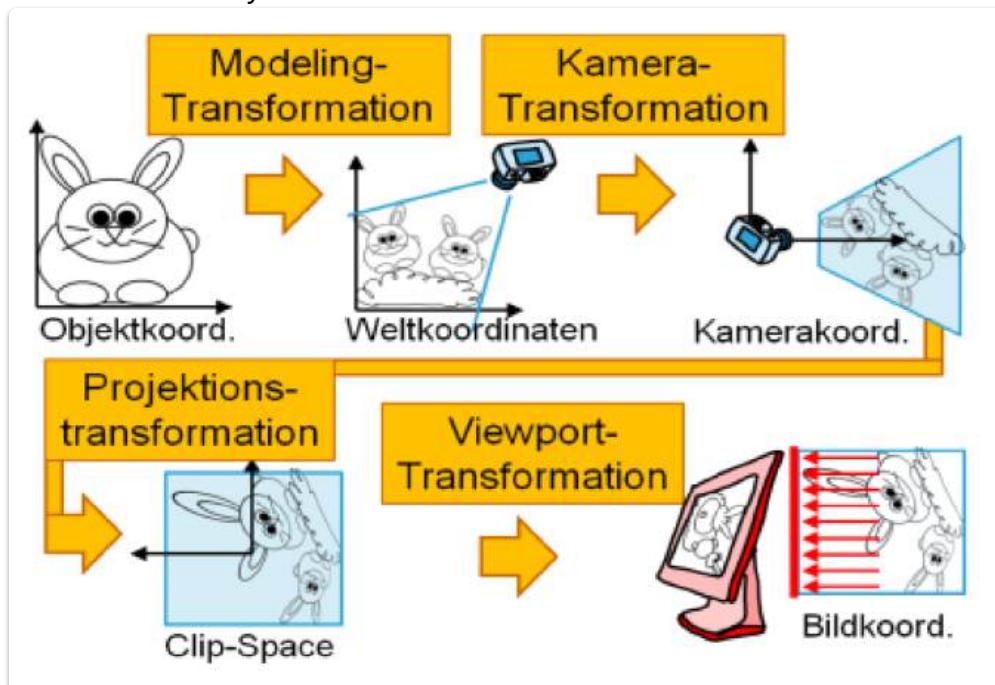
- Robust gegenüber Ausreißen**
  - Besonders effektiv bei impulsartigem Rauschen wie **Salt-and-Pepper-Noise**
- Erhält Kanten besser** als der Mittelwertfilter
- Nichtlinear**, daher nicht anfällig für lineare Glättungsverluste



## 6. Viewing

# Viewing in der Graphik-Pipeline

- Szenenmodellierung erfolgt in **Weltkoordinaten**
- Objekte werden aus **lokalen Koordinatensystemen** über **geometrische Transformationen (Matrizen)** in Weltkoordinaten überführt
- Nach Festlegung der **Kameraparameter**:
  - Transformation der Koordinaten in **Kamerakoordinaten**
  - Anschließende **Projektion** der Kamerakoordinaten
- Ergebnis der Projektion liegt in einem **normierten Würfel** (meist mit Seitenlänge 2)
- Danach erfolgt die **Viewport-Transformation**:
  - Überführung der normierten Koordinaten in **Gerätekordinaten** des Ausgabemediums
- In der Geometrie existieren verschiedene Projektionen
- In der **Computergraphik** sind hauptsächlich zwei Projektionen relevant:
  - **Parallelprojektion**
  - **Perspektivische Projektion**
- Zuerst wird die **Parallelprojektion** angenommen
- Danach wird die **Integration der perspektivischen Projektion** in die Pipeline betrachtet
- Vorgehensweise erfolgt **von hinten nach vorne** (vom Endergebnis zurück), da dies einfacher zu analysieren ist

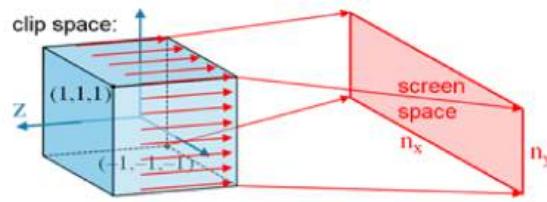


# Viewport-Transformation

Das ist das was eben vom Clip-Space in die Bildkoordinaten des Ausgabegeräts umwandelt

Wir nehmen also an, dass die Szene bereits im Clip-Space vorhanden ist, d.h. alle relevanten Koordinaten befinden sich in einem achsenparallelen Würfel der Seitenlänge 2 mit Mittelpunkt (0,0,0). Wir wollen eine orthographische Abbildung (parallele Normalprojektion) mit Blickrichtung -z auf einen Bildschirm mit Abmessungen  $n_x \times n_y$  (Pixel) durchführen. Es müssen also alle Punkte  $(-1, -1, z)$  auf  $(0, 0)$  abgebildet werden und alle Punkte  $(1, 1, z)$  auf  $(n_x \times n_y)$ . Diese lineare Abbildung wird durch die Matrix Mvp bewerkstelligt:

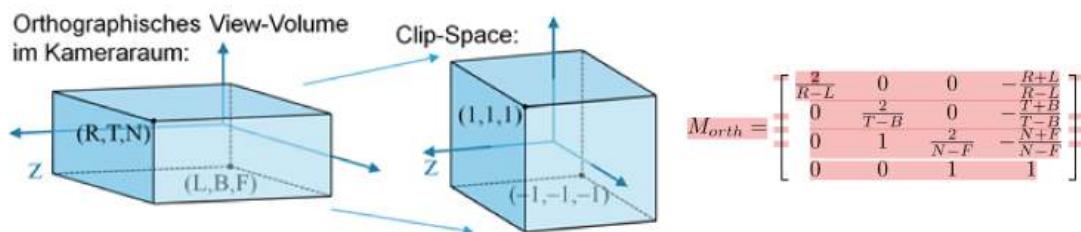
$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Deren Richtigkeit kann sofort nachgewiesen werden, indem man die Eckpunkte einsetzt. Die Matrix weist aber in den roten Zahlen noch eine Besonderheit auf: die z-Werte werden erhalten! Dies ist im Moment ohne Belang, wird aber bei späteren Schritten (vor allem bei der Berechnung der Sichtbarkeit) noch von großem Wert sein.

# Projektionstransformation:

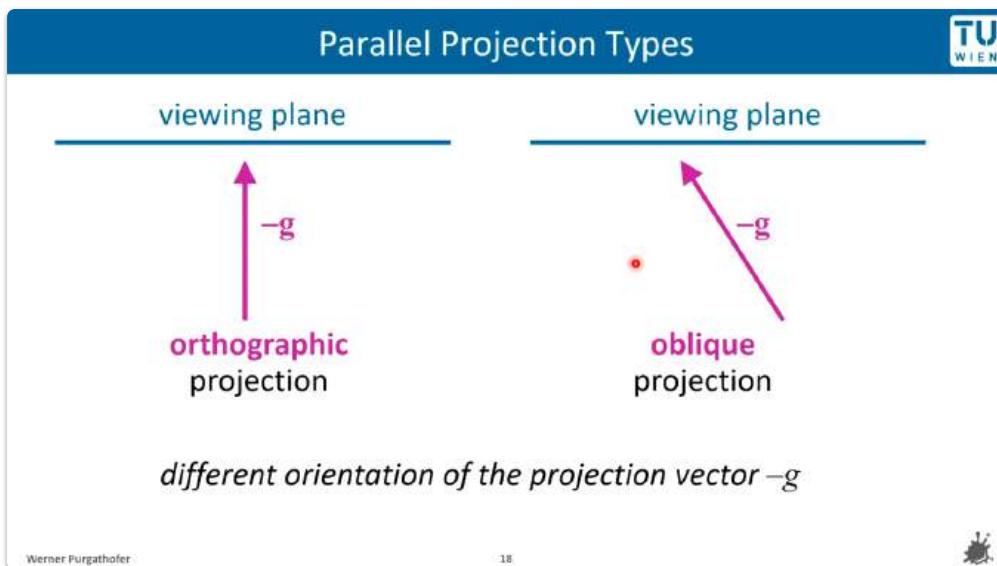
- Annahme: Orthographische Projektion
- Ziel: Vereinfachte Transformation durch achsenparallelen Quader
- Gegeben: Quader mit Grenzen:
  - L (Left)
  - R (Right)
  - B (Bottom)
  - T (Top)
  - N (Near)
  - F (Far)
- Transformation dieses Quaders in den normierten Würfel  $[-1, 1]^3$
- Dabei gilt:
  - Punkt  $(L, B, F) \rightarrow (-1, -1, -1)$
  - Punkt  $(R, T, N) \rightarrow (1, 1, 1)$
- Umsetzung erfolgt über eine Transformationsmatrix



Bemerkung: Eine Parallelprojektion kann auch schräg auf eine Abbildungsebene erfolgen (zum Beispiel beim Schattenwurf), diese Variante wird hier nicht berücksichtigt.

Warum ist das wichtig?

- **Anzeige auf 2D-Bildschirmen:** Unsere Monitore und Bildschirme sind zweidimensional. Um 3D-Grafiken darauf darstellen zu können, müssen die 3D-Objekte in 2D projiziert werden.



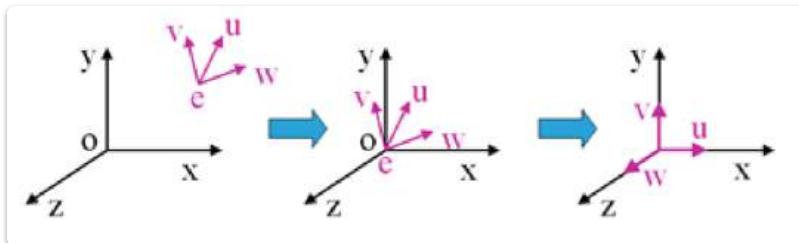
## Kamera Transformation

- Beim Festlegen der Kamerawerte bestehen mehrere **Freiheitsgrade**:
  1. **Position der Kamera** im Raum
  2. **Blickrichtung** von dieser Position aus
  3. **Orientierung** der Kamera (Definition von „oben“)
  4. **Größe des Bildausschnittes** (analog zur Brennweite oder Zoomfaktor)



- Aus den ersten drei Werten ergibt sich das **Viewing-Koordinatensystem** mit den Achsen  $u, v, w$
- Eigenschaften des Viewing-Koordinatensystems:
  - Die **uv-Ebene** ist **normal** zur Hauptblickrichtung
  - Die Blickrichtung verläuft entlang der **negativen w-Achse**
- In **Animationen** wird die Kameradefinition oft automatisch aus Bedingungen berechnet:
  - z.B. Kamerafahrt um ein Objekt
  - z.B. Kamerasteuerung in Flugsimulationen
  - Ziel: **Unkomplizierte Erzeugung gewünschter Effekte**
- Umwandlung von **Weltkoordinaten** in **Viewingkoordinaten** erfolgt durch eine Kette von **einfachen Transformationen**:

- **Translation**, um Koordinatenursprünge aufeinander abzustimmen
- **Drei Rotationen**, um Koordinatenachsen zur Deckung zu bringen:
  - Zwei Drehungen zur Ausrichtung der ersten Achse
  - Eine weitere Drehung für die zweite Achse
  - Die dritte Achse ergibt sich automatisch korrekt
- Zusammensetzung dieser Transformationen in eine **Transformationsmatrix**:  
 $M_{WC \rightarrow VC} = R_z \cdot R_y \cdot R_x \cdot T$
- Zur vollständigen **Projektionsbeschreibung** werden zusätzlich die **Grenzen des darzustellenden Bereichs** benötigt
- Mehr zu Transformations findet man hier: [3. Transformationen](#)



Ausgehend von der Kameraposition geht man prinzipiell folgendermaßen vor, um das Viewing-Koordinatensystem festzulegen:

1. Wahl einer Kameraposition (auch Augpunkt oder Viewing-Punkt genannt).
2. Wahl einer Blickrichtung, die negative Blickrichtung ergibt die w-Achse.
3. Wahl einer Richtung t „nach oben“; aus dieser lassen sich dann die u- und v-Achsen berechnen.
4. Da die Abbildungsebene normal auf die Blickrichtung liegt, ergibt das Vektorprodukt  $t \times w$  die Richtung der u-Achse.
5. Berechnung der v-Achse als Vektorprodukt der w- und u-Achsen:  $v = w \times u$ .
6. Die Wahl von minimalen und maximalen u-, v- und w-Werten zur Eingrenzung des Ausschnittes der Szene, der abgebildet wird: L(eft), R(ight), B(ottom), T(op), N(ear), F(ar).

Hier nochmal die Formeln:

$e$  ... eye position

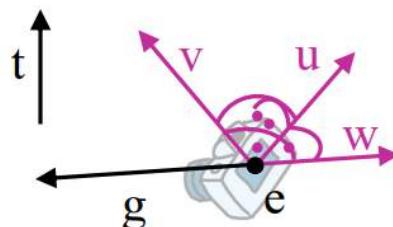
$g$  ... gaze direction (positive w-axis points to the viewer)

$t$  ... view-up vector

$$w = -\frac{g}{|g|}$$

$$u = \frac{t \times w}{|t \times w|}$$

$$v = w \times u$$



## Orthographisches Viewing

Für die orthographische Projektion (**Kamera bildet parallel ab**) haben wir nun alle Schritte durch Matrizen beschrieben, diese können wir wie bei den geometrischen Transformationen zu einer einzigen Matrix zusammensetzen (**multiplizieren**), die dann die gesamte Viewing-Transformation durchführt:

$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ z \\ 1 \end{bmatrix} = (\mathbf{M}_{\text{vp}} * \mathbf{M}_{\text{orth}} * \mathbf{M}_{\text{cam}}) * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Man beachte: die rechteste Matrix wird zuerst mit dem Punkt  $(x,y,z)$  multipliziert. Wenn man das Assoziativgesetz anwendet, dann kann man aber auch zuerst die 3 Matrizen miteinander multiplizieren, und kann damit alle Punkte nur mit dieser einen Ergebnismatrix direkt von Weltkoordinaten in Gerätekordinaten transformieren!

## Viewing: Camera + Projection + Viewport



$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ z \\ 1 \end{bmatrix} = \mathbf{M}_{\text{vp}} \cdot \mathbf{M}_{\text{orth}} \cdot \mathbf{M}_{\text{cam}} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

pixels on the screen

world coordinates

viewport transformation

projection transformation

camera transformation

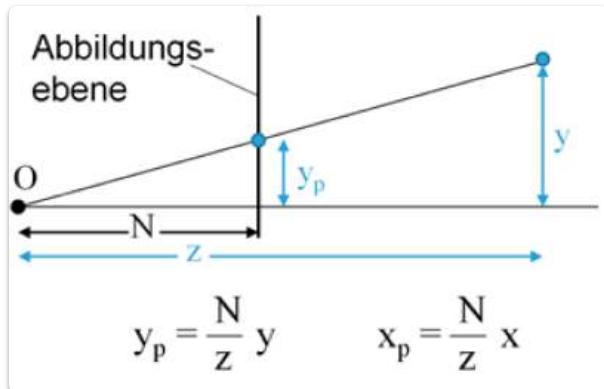
## Perspektive

- Perspektivische Projektion ist **keine affine Transformation**:
  - Affine Eigenschaften wie Parallelität bleiben **nicht** erhalten
  - Kann **nicht** mit einer  $3 \times 3$ -Matrix dargestellt werden
- Lösung: Verwendung von **homogenen Koordinaten**
  - Einziger Fall, in dem die **homogene Komponente  $h \neq 1$**
  - Erfordert einen **Divisionsschritt** am Ende der Transformation: Koordinaten werden durch  $h$  geteilt
- Grundlagen der perspektivischen Transformation:
  - $O$ : **Projektionszentrum**
  - **Blickrichtung**: entlang der **negativen  $z$ -Achse**
  - **Abbildungsebene**: normal zur  $z$ -Achse im Abstand  $N$  (Near)
- Abbildung eines Punktes  $(x, y, z)$  auf die Ebene:

$$(x, y, z) \rightarrow \left( \frac{x \cdot N}{z}, \frac{y \cdot N}{z}, N \right)$$

- Das lässt sich durch eine Matrix  $P$  darstellen:

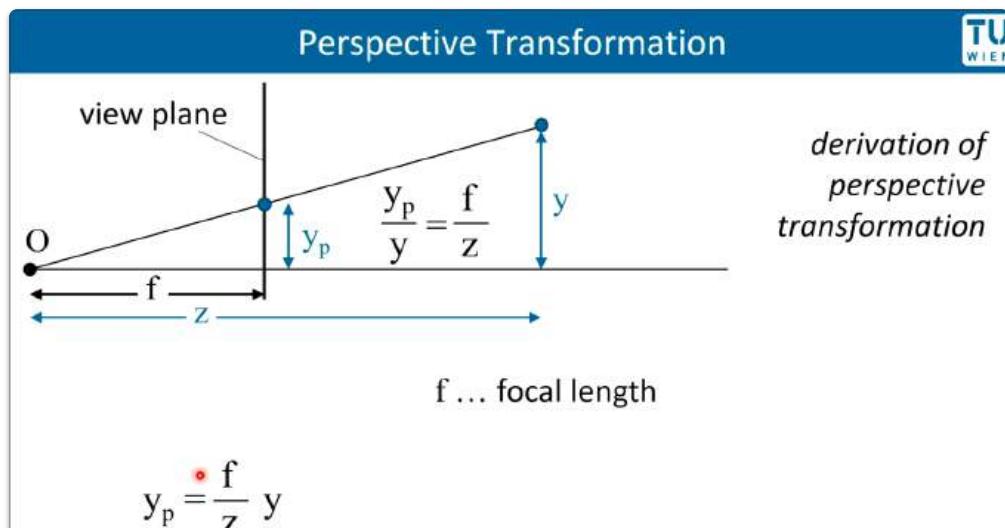
$$P = \begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 1 & N + F & -F * N \\ 0 & 0 & N & 1 \end{bmatrix}$$



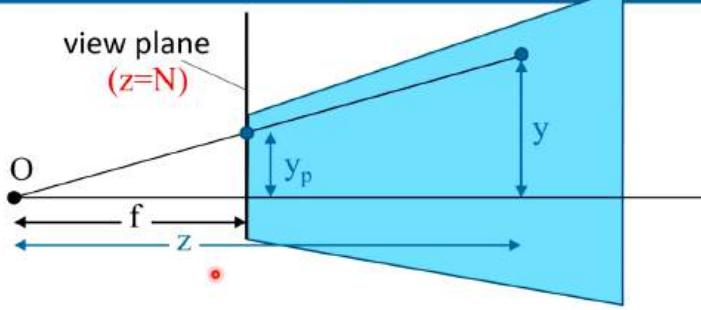
- Ein Punkt  $(x, y, z, 1)$  wird mit der **Projektionsmatrix  $P$**  multipliziert
- Ergebnis der Multiplikation:  $(x \cdot N, y \cdot N, z \cdot (n + F) - F \cdot N, z)$
- Danach erfolgt das **Homogenisieren** (Normalisieren): Division durch die letzte Komponente  $z$
- Ergebnis nach Division:

$$\left( \frac{x \cdot N}{z}, \frac{y \cdot N}{z}, (N + F) - \frac{F \cdot N}{z}, 1 \right)$$

Hier der Inhalt der Folien zu dem:



## Perspective Transformation



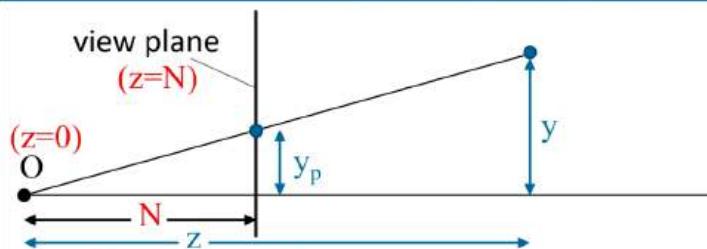
derivation of perspective transformation

$$y_p = \frac{f}{z} y$$

Werner Purgathofer

38

## Perspective Transformation



derivation of perspective transformation

$$x_p = \frac{N}{z} x$$

analogous:

$$y_p = \frac{N}{z} y$$

$$\begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & N+F & -F \cdot N \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Werner Purgathofer

38

## Perspective Transformation

$$P = \begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & N+F & -F \cdot N \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \cdot N \\ y \cdot N \\ z \cdot (N+F) - F \cdot N \\ z \end{bmatrix}$$

derivation of perspective transformation

homogenization: divide by z

$$x_p = \frac{N}{z} x$$

$$y_p = \frac{N}{z} y$$

$$\leadsto \begin{bmatrix} x \cdot N/z \\ y \cdot N/z \\ (N+F) - F \cdot N/z \\ 1 \end{bmatrix}$$

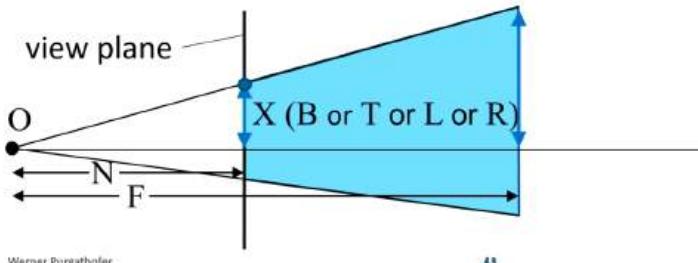
Werner Purgathofer

39

mit homogenization ist gemeint, dass man alles durch z dividiert, sodass die z-Koordinate = 1 ist.

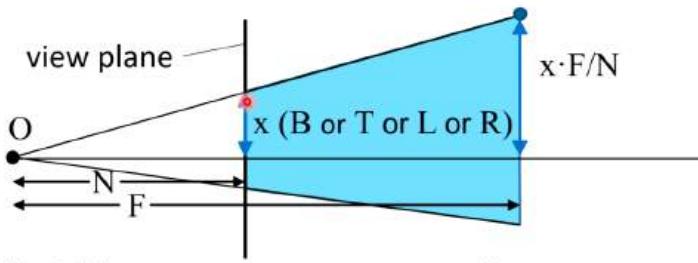
## Example: (Right) Top Near Corner

$$\begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & N+F & -F \cdot N \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R \\ T \\ N \\ 1 \end{bmatrix} = \begin{bmatrix} R \cdot N \\ T \cdot N \\ N \cdot (N+F) - F \cdot N \\ N \end{bmatrix} \rightsquigarrow \begin{bmatrix} R \\ T \\ N \\ 1 \end{bmatrix}$$



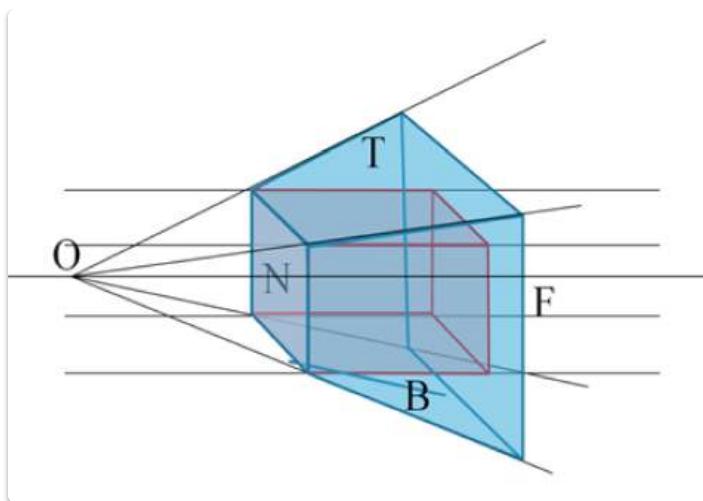
## Example: (Left) Top Far Corner

$$\begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & N+F & -F \cdot N \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} L \cdot F/N \\ T \cdot F/N \\ F \\ 1 \end{bmatrix} = \begin{bmatrix} L \cdot F \\ T \cdot F \\ F \cdot (N+F) - F \cdot N \\ F \end{bmatrix} \rightsquigarrow \begin{bmatrix} L \\ T \\ F \\ 1 \end{bmatrix}$$



- Die perspektivische Projektion führt zu einer **Verzerrung des Szenebereichs**:
  - Dieser Bereich wird als „**View Frustum**“ bezeichnet
  - Das View Frustum ist ein **Pyramidenstumpf**, der auf einen **achsparallelen Quader** abgebildet wird
  - In diesem Quader liefert die **orthographische Projektion** dasselbe Bild wie die **perspektivische Projektion** im View Frustum
- Nach der Verzerrung kann die bereits erarbeitete **Parallelprojektion** angewendet werden, um die perspektivische Matrix  $M_{per}$  zu berechnen
- Alternative Methode:
  - **Einsetzen** der Projektionsmatrix  $P$  an der richtigen Stelle in der Gesamtviewingberechnung
  - Dadurch wird eine **Gesamtmatrix** erzeugt, die die Transformation von Modellkoordinaten  $(x, y, z)$  zu Gerätekordinaten  $(x_{screen}, y_{screen})$  in einem Schritt ausführt

$$\begin{bmatrix} x_{screen} \\ y_{screen} \\ z \\ 1 \end{bmatrix} = M_{vp} * \underbrace{(M_{orth} * P * M_{cam} * M_{mod})}_{M_{per}} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



## Clip Space vs. NDC Space

**TU**  
WIEN

- Clip Space:  $[-w, w] \times [-w, w] \times [-w, w]$ ,  $w > 0$
- Normalized Device Coordinates (NDC):  $[-1,1] \times [-1,1] \times [-1,1]$

Clip Space	NDC Space
<b>Constraints</b> $x_{\min} = -w$ $x_{\max} = w$ $y_{\min} = -w$ $y_{\max} = w$ $z_{\min} = -w$ $z_{\max} = w$ $w > 0$	$(x_{\min}/w, y_{\min}/w, z_{\min}/w)$ $(x_{\max}/w, y_{\max}/w, z_{\max}/w)$ $(x_{\min}/w, y_{\max}/w, z_{\min}/w)$ $(x_{\max}/w, y_{\max}/w, z_{\max}/w)$ $(x_{\min}/w, y_{\min}/w, z_{\max}/w)$ $(x_{\max}/w, y_{\min}/w, z_{\max}/w)$ $(x_{\min}/w, y_{\max}/w, z_{\max}/w)$ $(x_{\max}/w, y_{\max}/w, z_{\max}/w)$
Pre-perspective divide puts the region surviving clipping within $-w \leq x \leq w, -w \leq y \leq w, -w \leq z \leq w$	
Post-perspective divide puts the region surviving clipping within the $[-1,+1]^3$	

Image Source: Mark Kilgard, University of Texas, CS 354 Graphics Math (2012)

- **Homogenisierung:**

- Wenn eine **perspektivische Abbildung** beteiligt ist, muss das Ergebnis am Ende durch die homogene Komponente  $z'$  dividiert werden.
- Im **Clipraum** wird nicht nur das **Clipping** durchgeführt, sondern auch die **Homogenisierung**.
- Nach der Homogenisierung wird die **Viewport-Matrix** als letzter Schritt angewendet.

- **Wichtige Eigenschaften der Projektionstransformation:**

1. **Gerade Strecken bleiben gerade Strecken:**

- Um eine Strecke (z.B. Seite eines Polygons) abzubilden, reicht es, nur die beiden Endpunkte zu transformieren.

2. **Relative Ordnung der z-Werte bleibt erhalten:**

- Der Abstand der Punkte von der Kamera bleibt relativ zueinander erhalten, jedoch nicht die **Abstandswerte selbst**.
- Diese Eigenschaft ist wichtig für die **Sichtbarkeitsberechnung**:

$$z_1, z_2, N, F < 0$$

$$z_1 < z_2$$

$$\frac{1}{z_1} > \frac{1}{z_2}$$

$$| * (-F * N)(< 0)$$

$$-F * \frac{N}{z_1} < -F * \frac{N}{z_2}$$

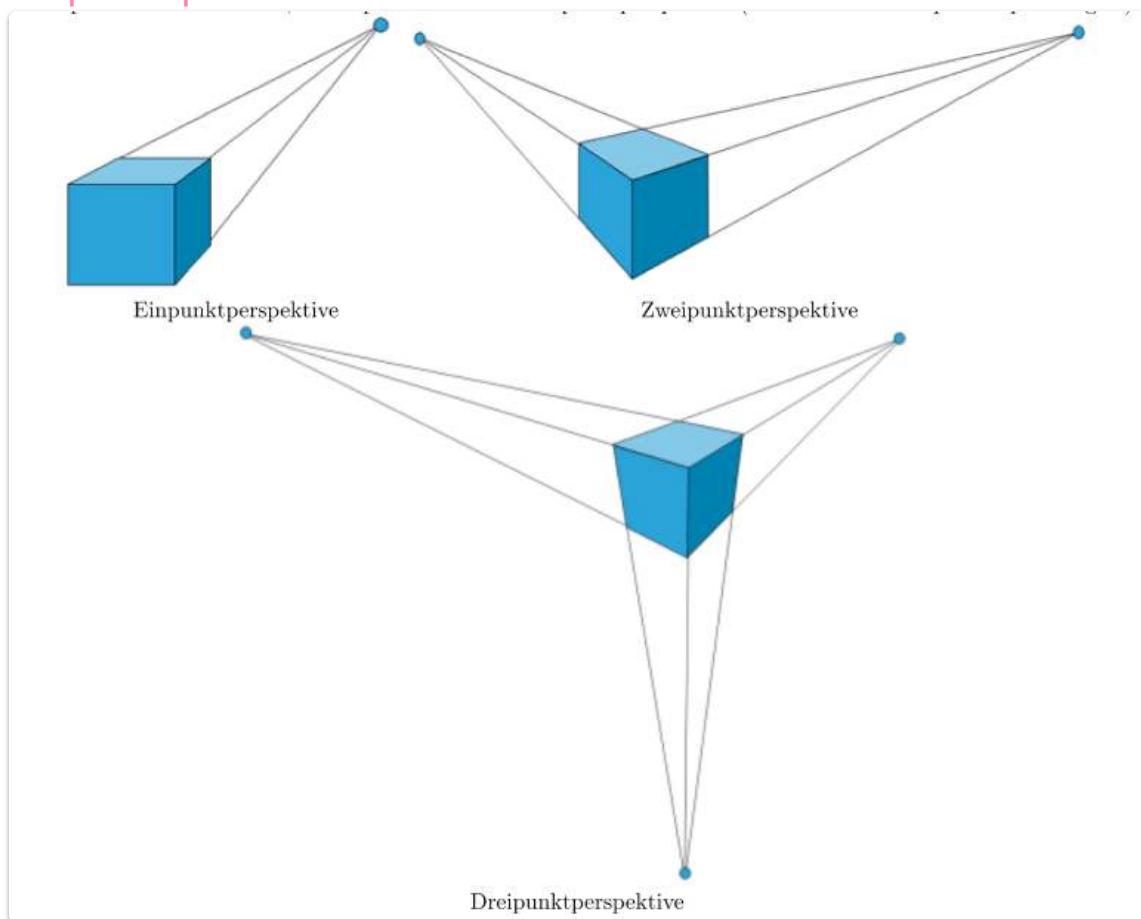
$$| + (N + F)$$

$$(N + F) - F * \frac{N}{z_1} < (N + F) - F * \frac{N}{z_2}$$

## Fluchtpunkte:

Anzahl der Hauptfluchtpunkte hängt von der Lage der **Bildebene** zum **Koordinatensystem** ab:

1. **Einpunktperspektive**: Zwei Achsen sind parallel zur Bildebene.
2. **Zweipunktperspektive**: Eine Achse ist parallel zur Bildebene.
3. **Dreipunktperspektive**: Keine Achse ist parallel zur Bildebene, es gibt **3 Hauptfluchtpunkte**.





# 6. Kantenfilterung

Information:

- [EVC\\_Skriptum\\_CV, p.29](#) bis [EVC\\_Skriptum\\_CV, p.34](#)
- ist quasi ein Unterkapitel von [5. Lokale Operationen](#)

## Kantenfilterung

- wichtiger Schritt in Bildverarbeitung
- um Kanten hervorzuheben
- Ansatz dafür ist Verwendung von Gradienten die man sich so berechnen kann:

$$\nabla I = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix}$$

- $G_x$  und  $G_y$  sind die Gradientenkomponenten in horizontaler bzw. vertikaler Richtung
- Stärke davon hängt durch Betrag von  $|\nabla I|$  und dem Winkel  $\phi$  ab

$$|\nabla I| = \sqrt{G_x^2 + G_y^2} \quad \text{und} \quad \theta = \text{atan2}(G_y, G_x)$$

- Es gibt verschiedene Filter die zur Kantenfilterung verwendet werden z.B.:
  - Sobel-Operator
  - Prewitt-Operator
  - ...

## Bildschärfung



## Ziel der Bildschärfung

- Nachträgliches Schärfen von Bildern dient dazu, Unschärfen zu kompensieren, die durch:
  - Scannen
  - Skalieren
  - Druck
  - Bildschirmanzeige entstanden sind oder entstehen können.

## Methode

- Hochfrequente Bildanteile enthalten die Bilddetails, die den Schärfeeindruck ausmachen.
- Beim Schärfen werden diese hochfrequenten Anteile angehoben.

## Laplace-Operator

- Im zweidimensionalen Fall erfolgt das Schärfen mit dem Laplace-Operator.
- Der Laplace-Operator  $\nabla^2$  einer Funktion  $f(x, y)$  ist definiert als:

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

wobei:

$$\frac{\partial f}{\partial^2 x}(x, y) = f(x + 1, y) + f(x - 1, y) - 2f(x, y)$$

in x Richtung und analog in y Richtung.

- Er nutzt die zweiten Ableitungen ( $\partial^2$  steht für 2. Partielle Ableitung) in horizontaler und vertikaler Richtung zur Hervorhebung von Kanten und feinen Strukturen.
- Daraus entsteht in der Kombination von x- und y-Richtung mit  $[1 - 21]$  und  $[1 - 21]^T$  der Laplace Operator
- Fürs Filtern von einem Bild, zuerst Laplace Filter und dann das Ergebnis vom Ursprungsbild subtrahieren:

$$I' = I - w * (H^L * I)$$

- $H^L$  ist Laplace Filter
- $w$  bestimmt Intensität der Schärfung

## Kanten

### Bedeutung von Kanten

- **Umgangssprachlich:** Kanten = abrupte Änderungen der Oberflächennormale (z.B. Tischkante).
- **In Bildern:** Kanten = Stellen mit **lokalen Änderungen der Intensität oder Farbe** (Diskontinuitäten).
- **Visuelle Wahrnehmung:**
  - Subjektive Schärfe hängt mit **Deutlichkeit von Diskontinuitäten** zusammen.
  - Kanten sind **strukturell wichtig** für die Interpretation von Bildern.
  - Unser Sehsinn erkennt Inhalte oft schon an **kantenförmigen Strukturen** (z.B. in Karikaturen).

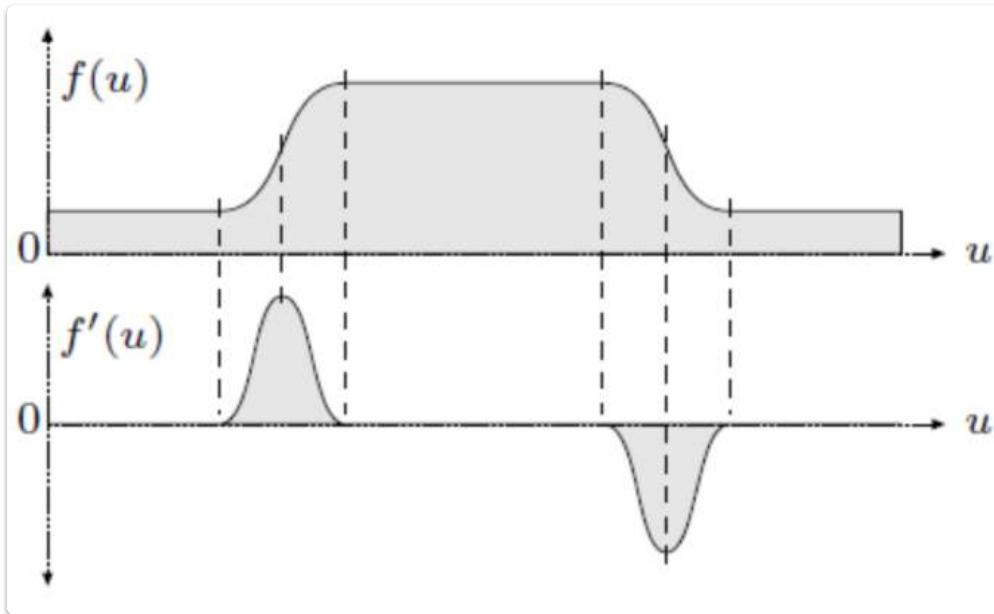
## Bedeutung in der Bildverarbeitung

- Kanten enthalten **nicht-redundante Information**, d.h. wichtige Strukturinformationen.
- Definition: Kanten = Orte im Bild mit **starker Intensitätsänderung auf kleinem Raum** und entlang **ausgeprägter Richtung**.

## Mathematische Beschreibung

- **Stärke der Änderung pro Distanz** = **erste Ableitung** der Intensitätsfunktion.
- Je größer die Ableitung, desto **deutlicher die Kante**.

## Gradientenbasierte Kantendetektion



## Motivation

- Betrachten wir ein eindimensionales **Intensitätsprofil** einer Bildzeile.
- Beispiel: Helle Region im Zentrum, dunkler Hintergrund → typische Kante im Bild.

## Erste Ableitung – Bedeutung

- Bezeichnet als:  
 $f'(u) = \frac{df}{du}(u)$
- **Positive Werte** → Intensität steigt (z.B. dunkler zu heller Übergang).
- **Negative Werte** → Intensität fällt (z.B. heller zu dunkler Übergang).

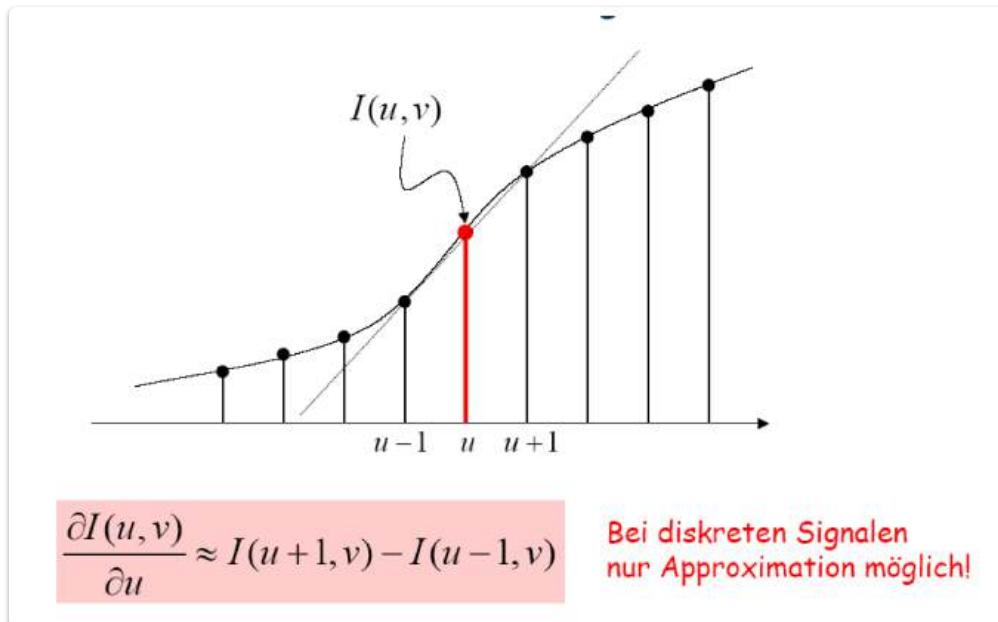
## Problem bei diskreten Bildern

- In digitalen Bildern ist die Funktion **diskret** → Ableitung **nicht direkt definiert**.
- Wir benötigen eine **Schätzung** (Approximation).

## Lösung: Differenzenapproximation

- Ableitung an Position  $u$  kann geschätzt werden durch:

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2}$$



- Dies entspricht dem **Anstieg einer Geraden**, die durch die benachbarten Abtastwerte verläuft.

## Interpretation

- Diese Approximation entspricht einem **Kantenfilter**.
- Wird oft in **Kantendetektion** (z.B. Sobel-Operator) verwendet.
- Erlaubt es, **Kanten als starke Intensitätsänderungen** zu erkennen – genau das, was unser Auge als „Kante“ wahrnimmt.

## Gradienten und Kanten in zweidimensionalen Bildern

- Ein Bild wird als zweidimensionale Funktion  $I(u, v)$  betrachtet
- Kanten im Bild = abrupte lokale Änderungen in Intensität oder Farbe
- Starke Änderungen = hohe Ableitungswerte → Hinweis auf Kante

- Erste Ableitung misst Stärke der Intensitätsänderung
- Für diskrete Funktionen ist die Ableitung nicht definiert → Approximation notwendig
- In 1D wird die erste Ableitung durch den Differenzenquotienten geschätzt:

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2}$$

- In 2D: partielle Ableitungen entlang der Koordinatenachsen:

$$\frac{\partial I}{\partial u}(u, v), \quad \frac{\partial I}{\partial v}(u, v)$$

- Gradient Vektor:

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial u} \\ \frac{\partial I}{\partial v} \end{pmatrix}$$

- Betrag des Gradienten (Kantenstärke):

$$\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial u}\right)^2 + \left(\frac{\partial I}{\partial v}\right)^2}$$

- Betrag des Gradienten ist rotationsinvariant
- Horizontale Ableitung kann geschätzt werden durch linearen Filter:

$$H_{Dx} = \frac{1}{2} \cdot [-1 \quad 0 \quad 1]$$

- Vertikale Ableitung entsprechend:

$$H_{Dy} = \frac{1}{2} \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

- 
- Filterantwort ist richtungsabhängig
  - Horizontale Filter erkennen vertikale Kanten, vertikale Filter horizontale Kanten
  - In flachen Bildregionen (konstante Intensität) ist die Filterantwort null

## Kantendetektionsfilter

### Allgemeine Eigenschaften von Ableitungsfiltern

- Ableitungsfilter beliebiger Ordnung dürfen keine Antwort auf konstante Intensitätswerte oder Signal-Offsets zeigen
- **Bedingung:** Summe der Filterkoeffizienten muss 0 sein

- Für gute Kantendetektion: Operatorantwort soll richtungsunabhängig sein → **isotrope Kantendetektion**

## Arten von Kantenfiltern

- **Isotrope Filter**: Reagieren unabhängig von der Richtung, z. B. **Laplace-Operator**
- **Richtungsabhängige Filter**: Reagieren auf u- oder v-Richtung, z. B. **Sobel-** und **Prewitt-Operator**

## Unterschiede der Kantenfilter

- Differenzieren sich durch:
  - die Filter zur Schätzung der Richtungsanteile
  - die Art der Kombination der Richtungsanteile zur Gesamtkanteninformation

## Gradient und Kanteninformation

- Gradient enthält:
  - **Betrag** → Kantenstärke
  - **Richtung** → Kantenverlauf

## Prewitt- und Sobel-Operator

- Beide nutzen Filterkerne mit 3 Zeilen bzw. 3 Spalten → reduzierte Rauschempfänglichkeit
- **Prewitt-Operator**:
  - Filter:  $HP_x, HP_y$
  - Einfache Box-Glättung vor der Ableitung
- **Sobel-Operator**:
  - Fast identische Filter, aber stärkere Gewichtung der mittleren Zeile/Spalte durch doppelte Glättung

## Mathematische Formulierungen

- Gradienten in x- und y-Richtung:

$$D_x(u, v) = H_x * I$$

$$D_y(u, v) = H_y * I$$

- Kantenstärke:

$$E(u, v) = \sqrt{D_x(u, v)^2 + D_y(u, v)^2}$$

- Kantenrichtung:

$$\Phi(u, v) = \tan^{-1} \left( \frac{D_y(u, v)}{D_x(u, v)} \right)$$

## Ablauf der Kantendetektion

1. Bild  $I$  wird mit Filtern  $H_x$  und  $H_y$  gefaltet
2. Aus  $D_x$  und  $D_y$  werden:
  - Kantenstärke  $E(u, v)$
  - Kantenrichtung  $\Phi(u, v)$  berechnet

## Weitere Kantenoperatoren

- **Roberts-Operator** (ältester Kantenoperator)
  - Sehr kleine Filtergröße:  $2 \times 2$
  - Schätzt Ableitungen entlang der Diagonalen
  - $H_{R1} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad H_{R2} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

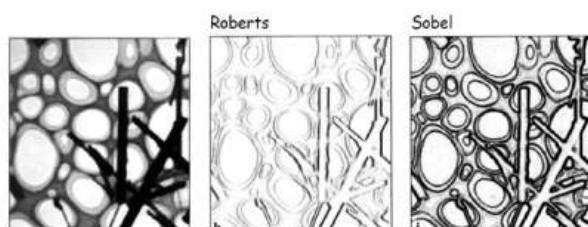
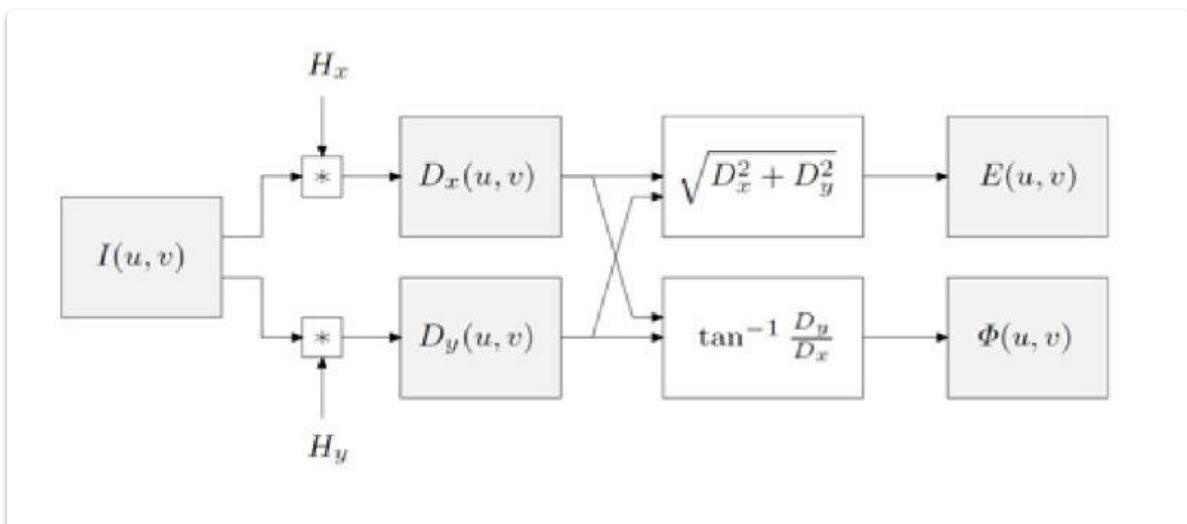


Abbildung 34: Roberts- und Sobelfilter

## Roberts-Filter

- **Starke Reaktion auf diagonale Kanten**
- Filter sind **weniger richtungsselektiv**: Reagieren über breites Orientierungsband ähnlich stark
- **Mängel bei horizontalen und vertikalen Kanten**:
  - Schlechte Filterantwort bei diesen Kanten, wodurch die Kantendetektion ungenau wird

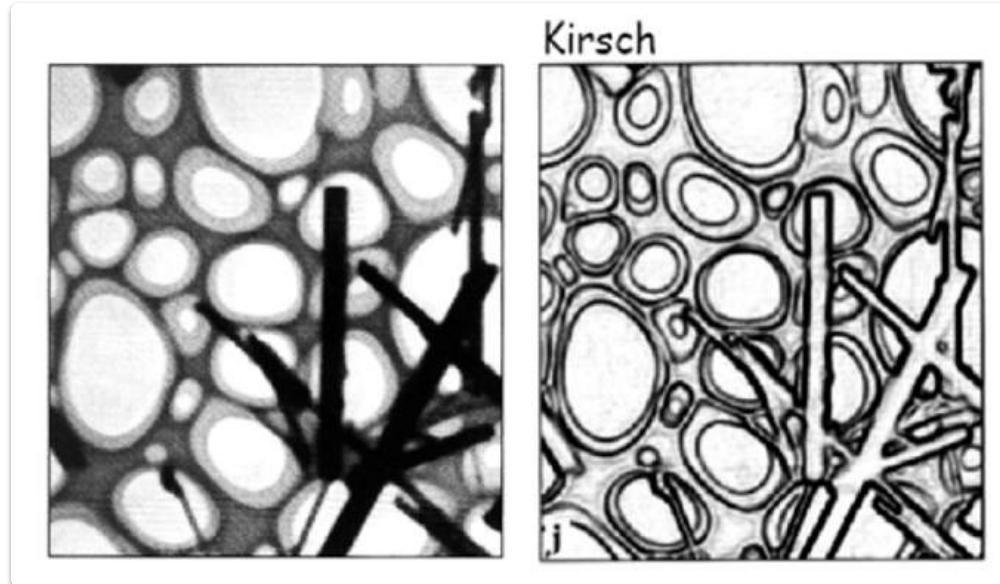
## Kompromiss im Design von Kantenfiltern

- **Besseres Filterdesign:**

- Je besser ein Filter auf kantenartige Bildstrukturen reagiert, desto **richtungsabhängiger** wird es
- **Breitere Filter** reagieren auf viele Orientierungen, aber mit geringerer Präzision in der Richtung
- **Engere Filter** reagieren stärker in spezifischen Richtungen, bieten jedoch bessere Kantenerkennung

## Kirsch-Operator

- **Kirsch-Operator:** Beispiel für den Einsatz mehrerer Filter in verschiedenen Richtungen



- Filter für **acht verschiedene Richtungen** im Abstand von  $45^\circ$
- Bietet höhere Präzision durch mehrere **enger aufgestellte Filter** für spezifische Richtungen
- Diese Acht Richtungen sehen so aus:

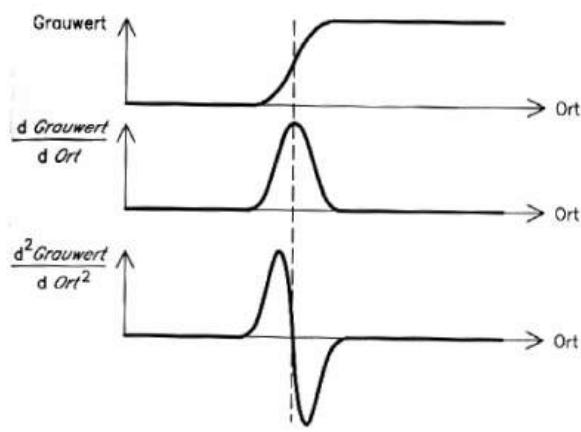
$$H_1^K = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, H_2^K = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}, H_3^K = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, H_4^K = \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix},$$

$$H_5^K = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}, H_6^K = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{pmatrix}, H_7^K = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}, H_8^K = \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

Die **Kantenstärke  $E^K$**  an der Stelle  $(u, v)$  ist als Maximum der einzelnen Filterergebnisse definiert, d.h.  $E^K(u, v) = \max(D_1(u, v), D_2(u, v), \dots, D_8(u, v))$ , und der am stärksten ansprechende Filter bestimmt auch die zugehörige **Kantenrichtung**. Derartige **Kompass-Operatoren** bieten allerdings kaum **Vorteile** gegenüber einfacheren Operatoren, wie z.B. dem Sobel-Operator. Ein Vorteil des Kirsch-Operators ist, dass er **keine Wurzelberechnung** benötigt.

## Kantendetektion mit 2. Ableitungen

## 6. Kantenfilterung, [xmozz](#)



## Problem der ersten Ableitung bei Kantendetektion

- **Breite der Kanten:** Bei der ersten Ableitung wird die Kante genauso breit wie der Anstieg der Bildfunktion, was eine genaue Positionsbestimmung erschwert.

## Nutzung der zweiten Ableitung

- **Zweite Ableitung** misst die **lokale Krümmung** der Funktion
- **Nulldurchgänge der zweiten Ableitung:** Diese stellen die tatsächlichen Kantenpositionen dar
  - **Vorteil:** Genauere Bestimmung der Kantenposition als bei der ersten Ableitung

## Laplace-Operator

- Bekannter und einfachster Kantenfilter basierend auf der zweiten Ableitung
- **Richtungsunabhängig** (Vorteil)
- **Rauschanfällig** (Nachteil)
- **Lösung:** Vor Anwendung des Laplace-Operators sollte ein Glättungsfilter zur Rauschunterdrückung eingesetzt werden

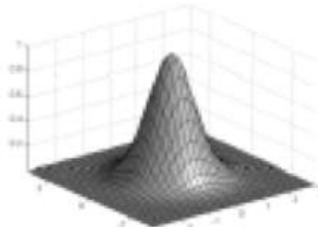
## Laplacian-of-Gaussian-Operator (LoG)

- **LoG-Operator (Laplacian of Gaussian)** oder **Marr-Hildreth-Operator:**
  - Kombiniert **Glättung mit einem Gauß-Filter** und die **zweite Ableitung (Laplace-Filter)** in einem einzigen linearen Filter
  - Aufgrund seiner Form (ähnelt einem mexikanischen Sombrero) auch als **Mexican Hat** oder **Sombrerofilter** bekannt
  - **Diskrete Umsetzung:**
    - Zuerst wird das Bild mit einer **Binomialmaske** geglättet
    - Anschließend erfolgt die Anwendung des **diskreten Laplace-Operators**

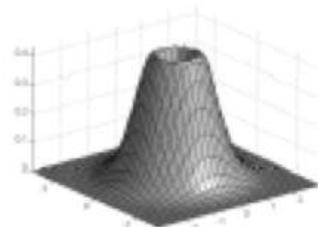
## 2D-Gauß-Funktion

- Ausgangspunkt für die Erzeugung des **LoG-Filterkernels** ist die **2D-Gauß-Funktion**

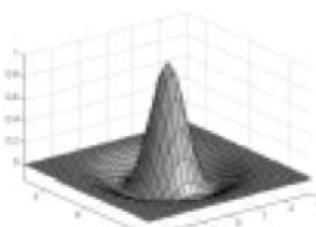
$$f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Gauss' Bell  
Function



First derivative



Minus second derivative  
„Mexican Hat“

Wendet man den Laplace-Operator auf die Gauß-Funktion an, erhält man die kontinuierliche Repräsentation des LoG:

$$g(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = -\frac{1}{\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(1 - \frac{x^2+y^2}{2\sigma^2}\right)$$

Die diskrete Approximation sollte für Kernel ungerader Kantenlänge durchgeführt werden, wobei der Ursprung des Kernels jeweils in der Mitte liegt. Für die Kantenlänge 5 entspricht dies dem unten abgebildeten Filter  $H^{\text{LoG}}$ . Die Abbildung zeigt ein Anwendungsbeispiel, die detektierten Kanten sind jetzt Ein Pixel breit.

$$H^{\text{LoG}} = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

## Canny Operator



### Kleinere Kantendetektoren (z. B. Sobel-Operator)

- **Begrenzte Reaktion:** Erkennen nur Intensitätsunterschiede innerhalb ihrer kleinen Filterregionen (z. B.  $3 \times 3$ )
- **Lösungsansatz für größere Unterschiede:**
  - Verwendung größerer Filter (größere Kantenoperatoren)
  - Anwendung der Operatoren auf **skalierte Bilder** (Mehrskalentechniken)

### Multi-Resolution bzw. Multi-Scale-Techniken

- **Grundidee:**

- Kanten werden auf verschiedenen Auflösungsebenen (Skalen) erkannt
- Für jede Bildposition wird entschieden, welche Kante auf welcher Ebene dominant ist

## Canny-Operator

- **Ziel:**

- Verwendet einen Satz großer, **gerichteter Filter** auf mehreren Auflösungsebenen
- Ergebnisse werden zu einem **gemeinsamen Kantenbild** (Edge Map) zusammengeführt

- **Ziele des Canny-Operators:**

- Minimierung der **falschen Kantenmarkierungen**
- **Optimale Lokalisierung** von Kanten
- Sicherstellung, dass pro Kante nur **eine Markierung** erfolgt

## Funktionsweise des Canny-Operators

- **Gradientenverfahren:**

- Der Canny-Operator basiert auf dem **Gradientenbetrag** des Bildes
- **Lokale Maximums-Suche**: Sichert, dass Kanten nur **ein Pixel breit** sind

## Nutzung von Single-Scale

- In der Praxis wird der Canny-Operator häufig in einer **Single-Scale-Version** angewendet
  - **Glättungsparameter**  $\sigma$  (Filterradius des Gaußfilters) wird angepasst, um die Kantendetektion zu verbessern
- **Vorteil:**
  - Verbesserte Kantendetektion im Vergleich zu einfachen Operatoren (wie Sobel)

Der Canny-Operator ist also besonders aufgrund seiner Fähigkeit, Kanten präzise zu erkennen, der Verwendung von Multi-Scale-Ansätze und der Optimierung durch die lokale Maximums-Suche gegenüber anderen Detektoren bevorzugt.



**Sobel**



**LOG**



**Canny**

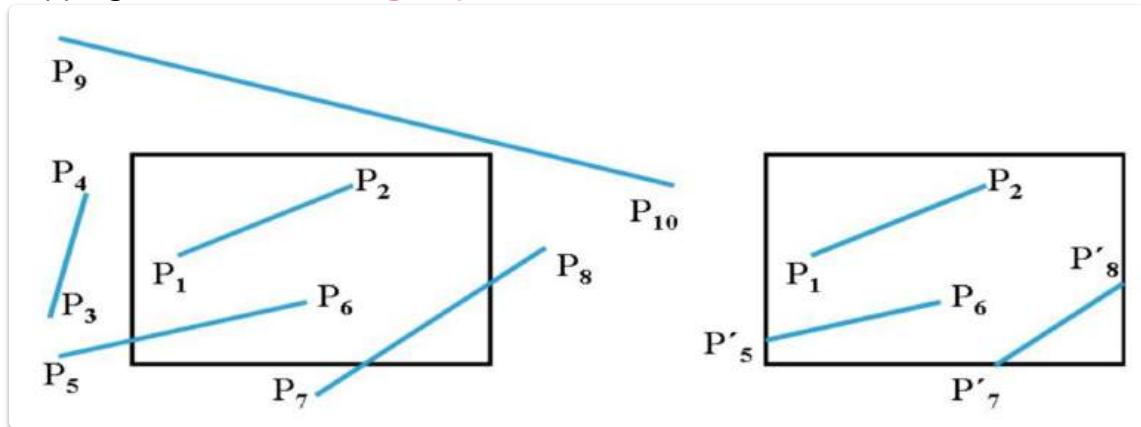
Kantenerkennung ist generell auch sehr wichtig für [8. Bildmerkmale - Interest Points](#)



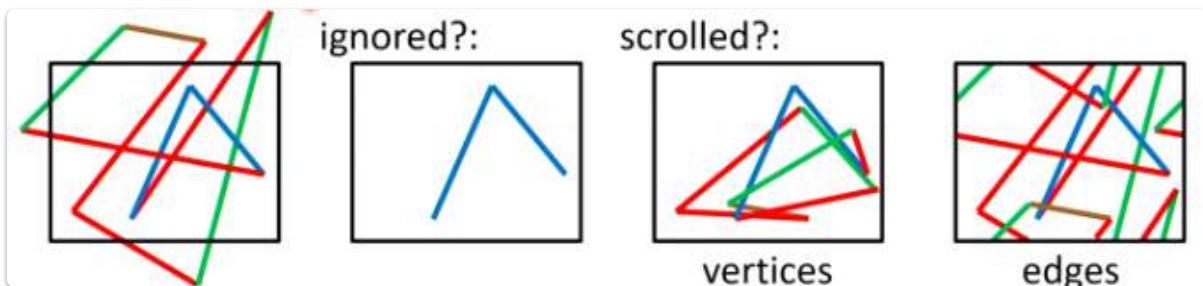
# 7. Clipping und Antialiasing

## Line Clipping:

- Clipping bezeichnet das **Abschneiden von Bildteilen**, die außerhalb des Darstellungsfensters liegen.
- Clipping wird durchgeführt, um **unnötige nachfolgende Umformungen** von nicht sichtbaren Teilen zu vermeiden:
  1. Clipping in Weltkoordinaten:
    - Analytische Berechnung zum frühestmöglichen Zeitpunkt.
  2. Clipping in Clipkoordinaten:
    - Analytische Berechnung an **achsenparallelen Grenzen** (einfacher).
  3. Clipping bei der Rasterkonversion:
    - Clipping erfolgt innerhalb des Algorithmus, der ein **Grafikprimitiv** in **Punkte umwandelt**.
- Clipping ist eine sehr **häufige Operation**, daher muss es **einfach und schnell** sein.



Wichtige Fragen: Was soll abgeschnitten werden?



## Clippen von Linien: Cohen-Sutherland-Verfahren

- Algorithmen zum Clippen von Linien nutzen die Tatsache aus, dass jede Linie in einem rechteckigen Fenster höchstens **einen sichtbaren Teil** besitzt.
- Wichtige Grundprinzipien der **Effizienz**:
  1. Häufige einfache Fälle früh eliminieren.

## 2. Teure Operationen wie Schnittpunkt-Berechnungen vermeiden.

- Ein einfaches Linieneckling könnte so aussehen:

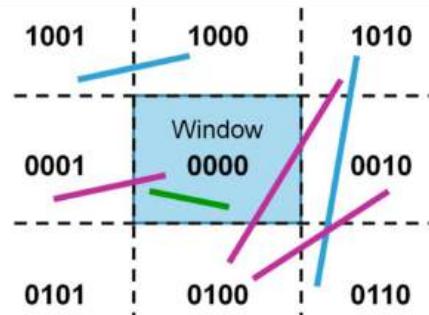
```

1 for endpoints (x0,y0), (xend,yend)
2 intersect parametric representation
3   x = x0 + u * (xend - x0)
4   y = y0 + u * (yend-y0)
5 with window borders:
6   intersection <= 0 < u < 1

```

Der Cohen-Sutherland-Algorithmus klassifiziert zuerst die Endpunkte einer Linie hinsichtlich ihrer Lage zum Clippingfenster: oben, unten, links, rechts, und codiert diese Information in 4 Bit. Nun kann man schnell überprüfen:

1. OR der beiden Codes = 0000  $\Rightarrow$  Linie ganz sichtbar
2. AND der beiden Codes  $\neq$  0000  $\Rightarrow$  Linie ganz unsichtbar
3. andernfalls mit einer relevanten Fensterkante schneiden, und den weggescnittenen Punkt durch den Schnittpunkt ersetzen.  
GOTO 1.



- Schnittpunktberechnungen mit vertikalen Fensterkanten:

- Für die linke Kante:

$$y = y_0 + m(x_{wmin} - x_0) \quad y = y_0 + m(x_{wmin} - x_0)$$

- Für die rechte Kante:

$$y = y_0 + m(x_{wmax} - x_0) \quad y = y_0 + m(x_{wmax} - x_0)$$

- Schnittpunktberechnungen mit horizontalen Fensterkanten:

- Für die untere Kante:

$$x = x_0 + \frac{(y_{wmin} - y_0)}{m} x = x_0 + m(y_{wmin} - y_0)$$

- Für die obere Kante:

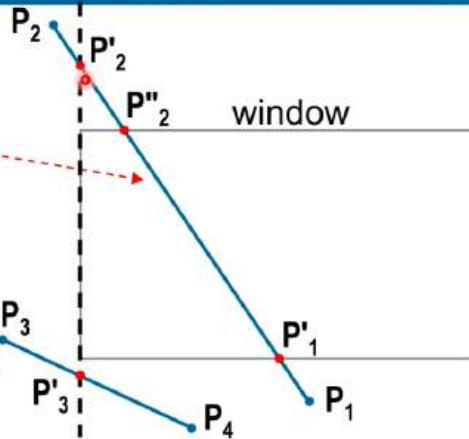
$$x = x_0 + \frac{(y_{wmax} - y_0)}{m} x = x_0 + m(y_{wmax} - y_0)$$

- Punkte, die genau auf den Fensterkanten liegen, gelten als innerhalb des Fensters.
- Es sind höchstens 4 Schleifendurchläufe erforderlich, da es höchstens 4 Schnittpunkte gibt.
- Effizienz:** Schnittpunktberechnungen werden nur durchgeführt, wenn sie wirklich notwendig sind.
- Clipping von Kreisen:**
  - Ähnliches Verfahren wie für Linien.
  - Kreise können beim Clipping in mehrere Teile zerfallen.

## Cohen-Sutherland Line Clipping

*passes through clipping window*

*intersects boundaries without entering clipping window*



vertical:  $y = y_0 + m(xw_{\min} - x_0)$ ,

horizontal:  $x = x_0 + (yw_{\min} - y_0)/m$ ,

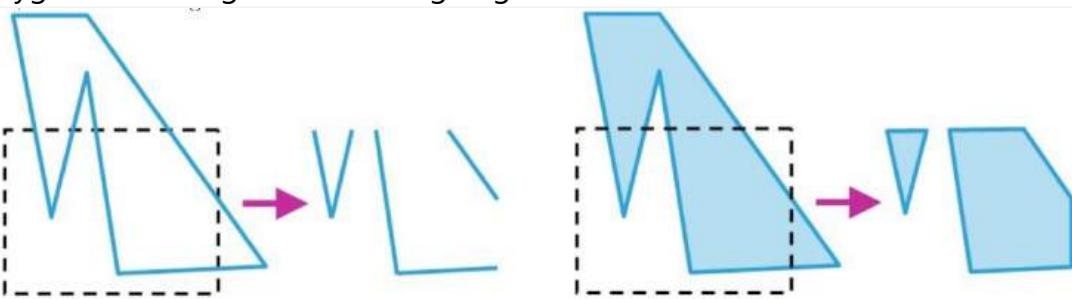
Werner Purgathofer

13



## Polygon Clipping:

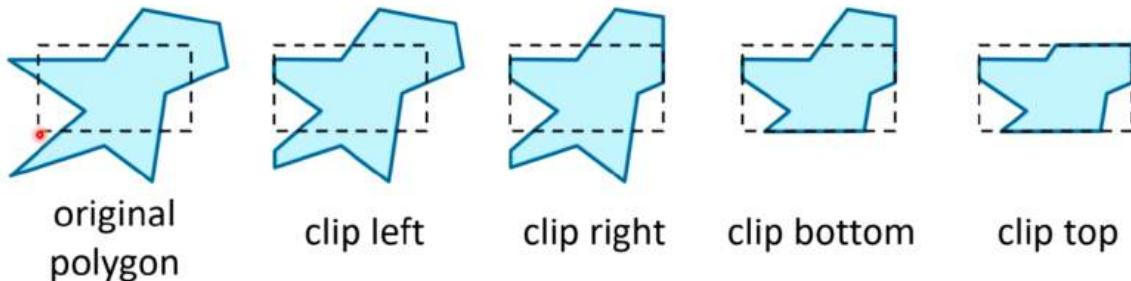
- **Polygon-Clipping** muss sicherstellen, dass nach dem Clipping ein **gültiges Polygon** entsteht, auch wenn der Clipping-Vorgang mehrere Teile erzeugt.
- Beispiel:
  - **Linien-Clipping-Algorithmus:**
    - Das Ergebnis zeigt ein **unvollständiges Polygon**, bei dem nicht mehr erkennbar ist, was innen und was außen liegt.
  - **Korrekte Polygon-Clipping-Verfahren:**
    - Das Polygon zerfällt in **mehrere Teile**, die alle korrekt **gefüllt** werden können.
- **Wichtig:** Auch wenn mehrere Teile entstehen, müssen alle Teile des resultierenden Polygons **korrekt** gefüllt und als gültige, sichtbare Bereiche behandelt werden.



Bei einfachem Linien Clipping könnten Linien zurückkommen deshalb gibt es extra Verfahren für das Polynomclipping:

## Sutherland-Hodgman Polygon Clipping

processing polygon boundary as a whole against each window edge  
 → output: list of vertices



clipping a polygon against successive window boundaries

Man zerlegt das hier in **4 Schritten** die meist rekursiv aufgerufen werden

## Clippen von Dreiecken:

- **Geometrische Daten** bestehen in der Praxis häufig nur aus **Dreiecken**. Der **Renderingprozess** hat dabei kein Wissen mehr über den Zusammenhang der Dreiecke und behandelt diese als eine „**Triangle Soup**“ (Dreiecks-Suppe).
- **Wichtig:**
  - Beim Clipping von Dreiecken muss immer darauf geachtet werden, dass nur Dreiecke entstehen – keine anderen Primitives.
- Beim **Clippen eines Dreiecks** gegen eine Kante gibt es vier mögliche Fälle:
  1. In einigen Fällen kann auch ein **Viereck** entstehen.

**Clipping of Triangles**

often b-reps are “triangle soups”  
 clipping a triangle → triangle(s)

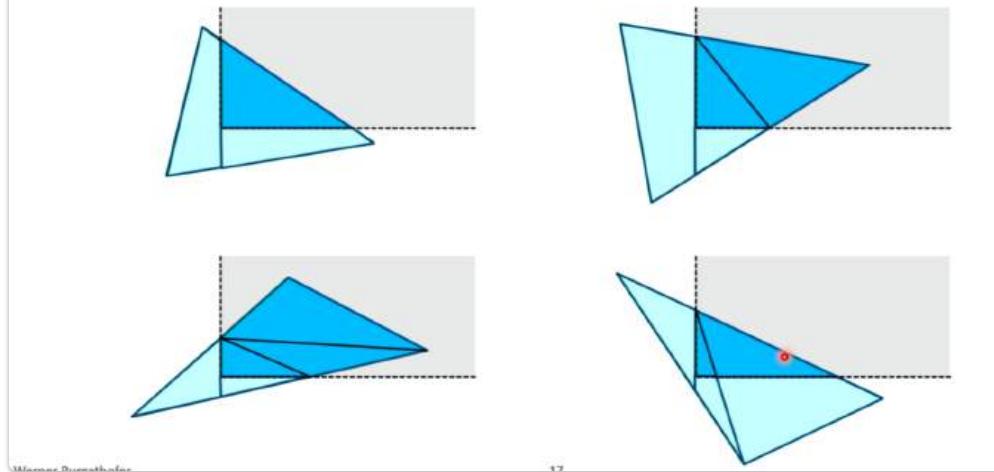
4 possible cases:

- (1) inside
- (2) outside
- (3) triangle
- (4) quadrilateral → 2 triangles

(inside)

2. Das Viereck muss sofort in **zwei Dreiecke** zerteilt werden, um die Weiterverarbeitung zu ermöglichen.

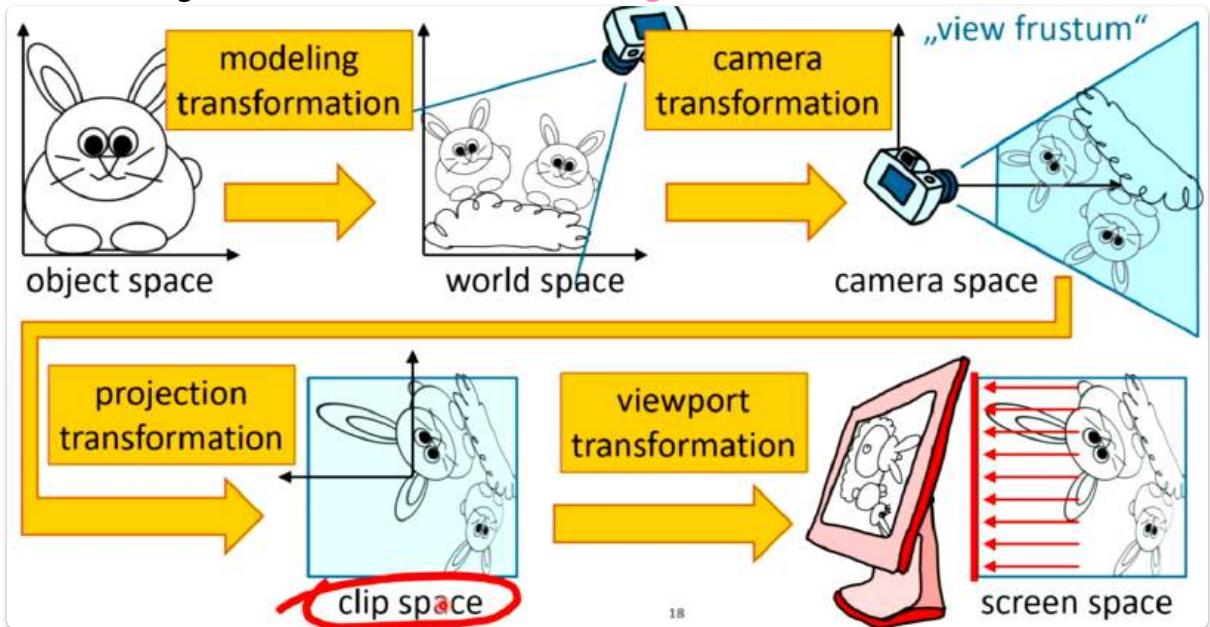
corner cases need no extra handling!



- An den **Ecken des Clip-Fensters** kann es vorkommen, dass mehr Dreiecke erzeugt werden, als tatsächlich notwendig wären (siehe Beispiel links), aber dies wird durch die **Einfachheit des Algorithmus** mehr als kompensiert.

## Clipping in Clipkoordinaten:

- Clip-Space:**
  - Die Begrenzungsflächen des **View-Frustums** sind achsenparallel (d.h., die Grenzen sind bei  $x = \pm 1, y = \pm 1, z = \pm 1$ ).
  - Dies vereinfacht die Feststellung, ob ein Punkt **innerhalb oder außerhalb** des Frustums liegt, da es nur einen **einfachen Vergleich** zwischen zwei Zahlen erfordert.



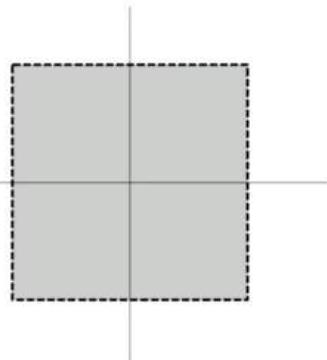
- Clipping vor der Homogenisierung:**
  - Um zu vermeiden, dass Punkte hinter dem Kamerapunkt projiziert werden, wird das Clipping schon **vor der Homogenisierung** der Punktkoordinaten durchgeführt.
  - Dabei wird an den **Ebenen  $x = \pm h, y = \pm h, z = \pm h$**  geclipppt (was ebenso einfach ist).
- Vorteile:**
  - Punkte, die hinter dem Kamerapunkt liegen, werden **nicht projiziert**.

2. **Ersparnis der Homogenisierungsdivision** für Punkte, die außerhalb des Clipbereiches liegen.

clipping against  $x = \pm 1, y = \pm 1, z = \pm 1$

( $x, y, z$ ) inside?

→ only compare one value per border!



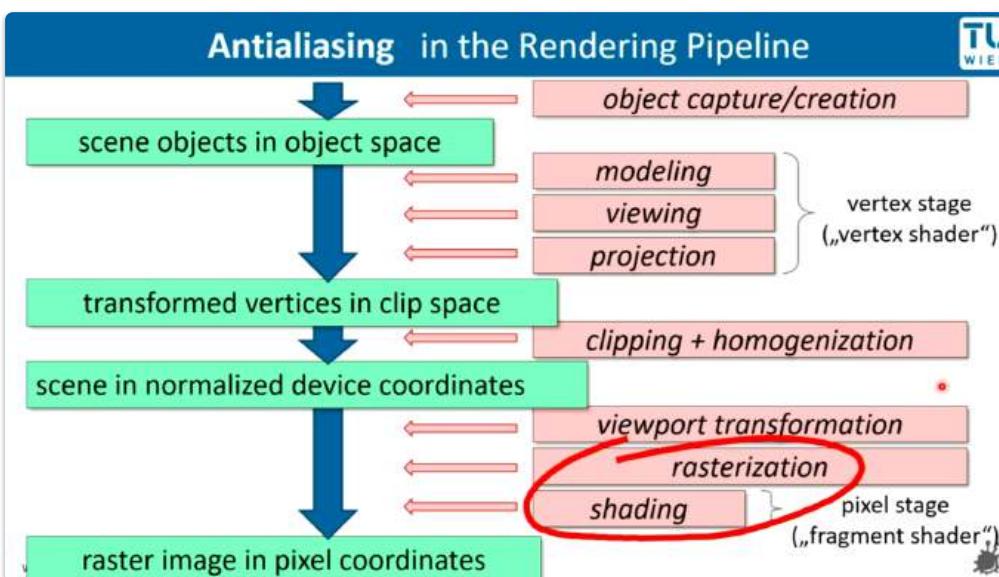
is done *before* homogenization:

clipping against  $x = \pm h, y = \pm h, z = \pm h$

clips points that are behind the camera!

reduces homogenization divisions

## Aliasing und Antialiasing:



### Aliasing

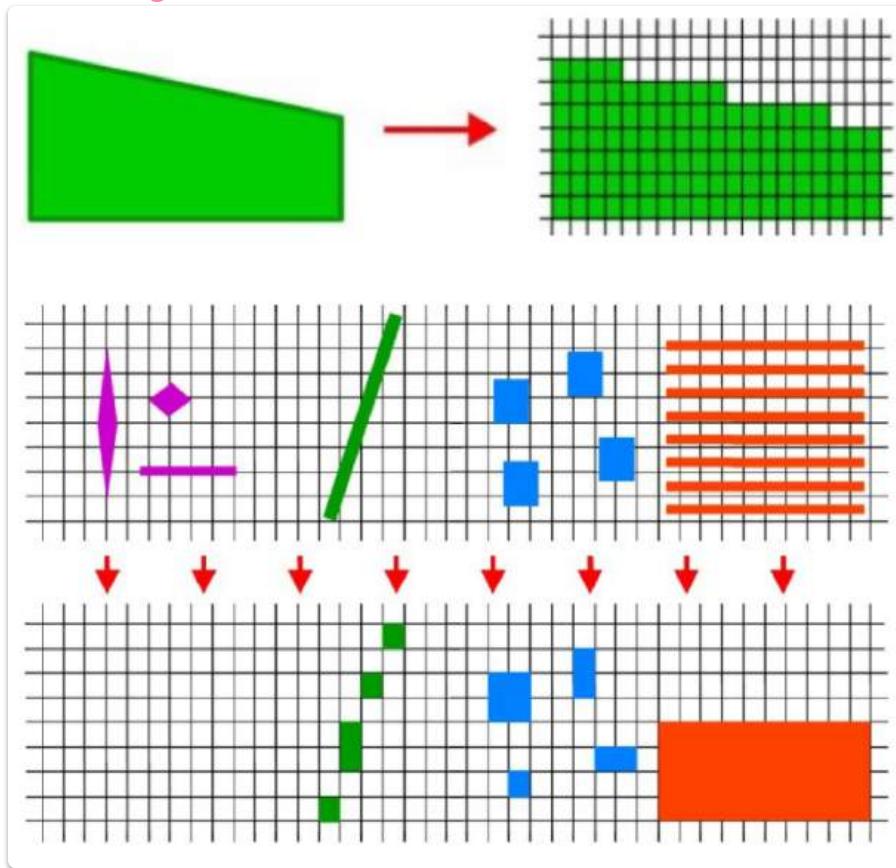
- **Aliasing-Effekte** sind Fehler, die bei der **Umwandlung (Diskretisierung)** von analogen in digitale Informationen auftreten.
- **Ursachen für sichtbare Aliasing-Effekte:**
  1. Zu geringe **Auflösung**
  2. Zu wenige **verfügbare Farben**
  3. Zu wenige **Bilder pro Sekunde** (Frames per second)
  4. **Geometrische Fehler**
  5. **Numerische Fehler**
- Ein Beispiel für **Aliasing**: Ein Pixel hat nur einen Wert, stellt aber tatsächlich eine **kleine Fläche** dar, was zu Unschönheiten in Rasterbildern führen kann.

## Antialiasing

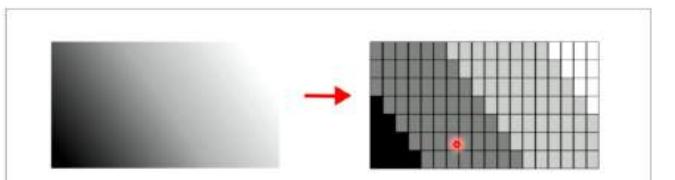
- **Antialiasing** bezeichnet Methoden zur **Reduktion** unerwünschter Aliasing-Artefakte.
- Verbesserung der **Hardware** ist meist unrealistisch, daher kommen hauptsächlich **Software-Methoden** zum Einsatz.
- Der Fokus liegt oft auf **Anti-Aliasing** zur Behandlung des **Auflösungsproblems**.

## Bekannte Aliasing-Effekte neben dem Treppeneffekt

1. **Verschwinden kleiner Objekte**
2. **Unterbrochene, schmale Objekte**
3. **Unterschiedliche Größen gleicher Objekte**
4. **Zerstörung feiner Texturen**.



Begriff Aliasing kommt von Alias: Das was angezeigt wird, zeigt sich an etwas anderem...

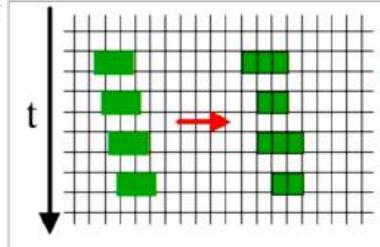


artificial color borders can appear

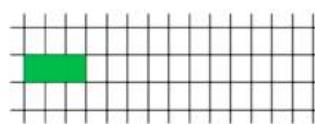
Aliasing kann auch in Animationen vorkommen:

- jumping images

- "worming"



•



- backwards rotating wheels



je nach dem welchen Pixel Mittelpunkt man anschaut bekommt man verschiedene Ergebnisse. Außerdem bekommt man dann einen Worming Effekt (also das Objekt wird immer größer und dann wieder kleiner). Ein anderer Effekt wäre, dass sich zum Beispiel in Filmen es so aussieht als würden sich die Räder rückwärts bewegen.

## Antialiasing von Linien:

### Ursache von Aliasing

- **Aliasing** entsteht durch **ungenügend feine Abtastung** des wahren Bildes, was zu Fehlern in der Rekonstruktion führt.
- **Nyquist-Shannon-Abtasttheorem:**
  - **Theoretische Grundlage:** Eine Information kann nur korrekt rekonstruiert werden, wenn die **Abtastfrequenz (sampling rate)** mindestens doppelt so hoch ist wie die höchste zu übertragende **Informationsfrequenz**.
  - Diese Grenze wird als **Nyquist-Limit** bezeichnet. ([2. Bildaufnahme](#))

### Beispiel und Fehlerbehebung

- **Beispiel:** Eine zu grobe Abtastrate eines Signals führt zu einer **falschen Rekonstruktion** (z.B. eine Kurve wird zu einem Polygonzug, siehe Abbildung).
- Fehler können reduziert werden durch:
  1. **Vorfilterung des Signals.**
  2. **Nachbearbeitung des fertigen Bildes** (jedoch unterliegt diese der Vorfilterung und ist weniger effektiv).

### Antialiasing für Linien

- **Pixel, die von einer Linie weiter durchkreuzt werden,** sollen mehr Linienfarbe erhalten als Pixel, die nur leicht gestreift werden.
- **Prozess:**

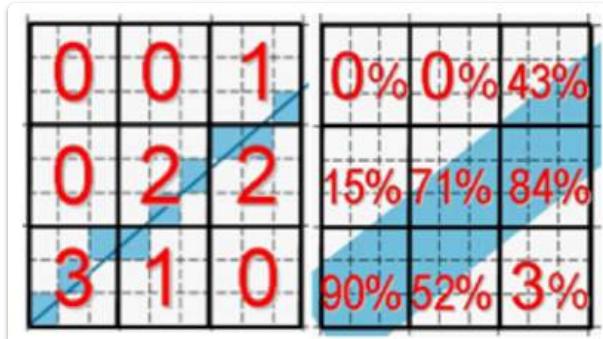
1. **Unterteilung jedes Pixels in Subpixel:** Für jedes Pixel wird gezählt, wie viele Subpixel von der Linie durchkreuzt werden.
2. **Intensitätswahl:** Die Intensität der Linienfarbe wird **proportional zur Anzahl** der Subpixel gewählt, die von der Linie durchquert werden.

## Breitere Linien

- Für **breitere Linien** wird der **Prozentsatz der Überdeckung** des Pixels durch die Linie berechnet und die Intensität der Linienfarbe entsprechend angepasst.

## Weighted Oversampling

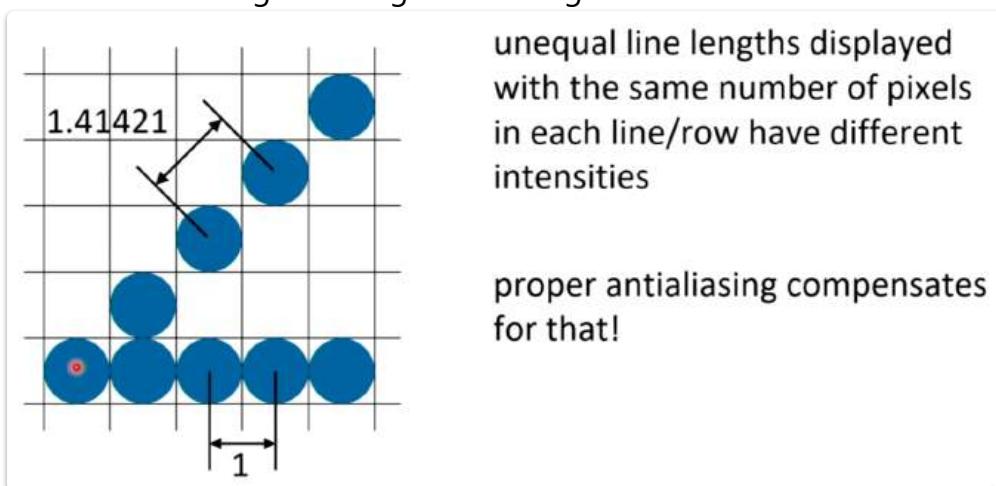
- Basierend auf der Erkenntnis, dass die **Mitte eines Pixels wichtiger** ist als der Rand, werden in einigen Fällen die **Subpixel in der Mitte stärker gewichtet** als die am Rand.
- Diese Technik wird als „**weighted oversampling**“ bezeichnet.



Hier haben wir ein 3 mal 3 Pixelfeld welches dann am Bildschirm angezeigt wird, und die kleinen Pixel im Pixel sind die Subpixel, welche gerendert werden wollen. Man kann allerdings nur die großen Pixel an- und abschalten.

## Andere Effekte von Antialiasing:

Verschiedene Längen trotz gleicher Pixelgröße:



Das kann man mit Area Boundaries behoben:

The diagram illustrates the process of adjusting pixel intensities along an area boundary. On the left, a grid shows a diagonal boundary line segment from (x, y) to (x+1, y+1). Below it, two small images show the effect of supersampling: the first shows a yellow-to-black gradient with visible steps, while the second shows a smoother result where the gradient is sampled at multiple points along the boundary.

**alternative 1:  
supersampling**

Surface Boundary  
Scan Line 1  
Scan Line 2  
Subdivided Pixel Area

## Antialiasing von Polygonkanten:

### 1. Ziel des Antialiasings

- **Ziel:** Reduktion der aliasing-bedingten Treppeneffekte an den Kanten von Polygonen.



### 2. Methoden zur Berechnung des Antialiasings

- **Alternativen:**
  - **Supersampling:** Mehrere Proben pro Pixel werden verwendet.
  - **Überdeckungsgradberechnung:** Der Überdeckungsgrad eines Pixels durch das Polygon wird direkt berechnet.

### 3. Berechnung des Überdeckungsgrads

- **Rasterkonversion:** Der Überdeckungsgrad wird während der Rasterkonversion berechnet, also beim Erzeugen der Randlinie und Füllung des Polygons.
- **Scanlinien-Füllverfahren:**

- Bei der Berechnung der Endpunkte der Scanlinien (Spans) im Füllverfahren fallen genug Informationen an, um den Überdeckungsgrad fast kostenfrei zu berechnen.

## 4. Verwendung der Entscheidungsvariable aus dem Bresenham-Algorithmus

- Bresenham-Linien-Algorithmus:**

(siehe [5. Rasterisierung](#))

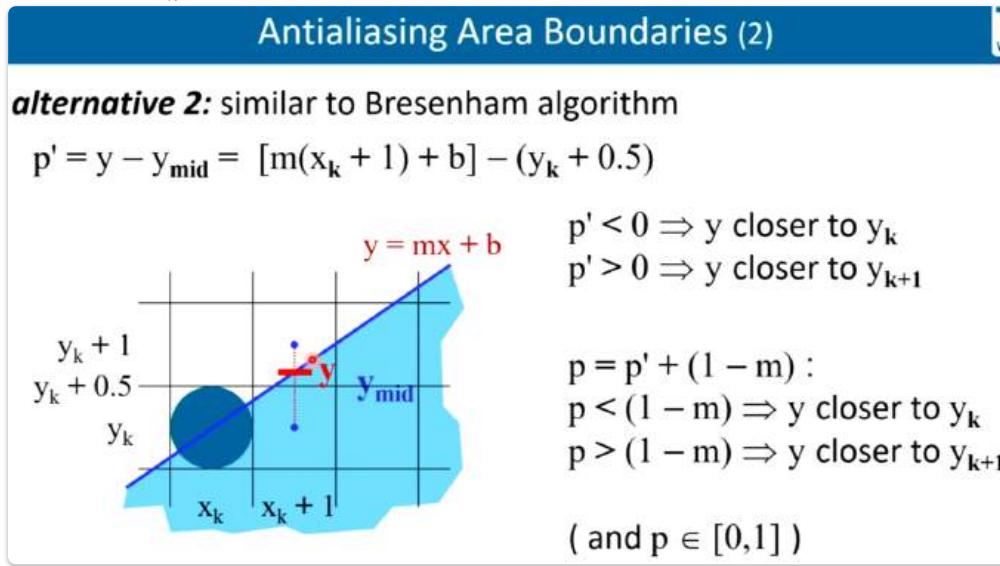
- Die Entscheidungsvariable  $p_k$  gibt an, welches Pixel als nächstes gezeichnet wird.
- Diese Variable kann so umgewandelt werden, dass ihr Wert den Überdeckungsgrad des letzten Pixels darstellt.
- Transformation:**
  - $p' = y - y_{mid}$ , wobei:

$$y_{mid} = \frac{y_k + y_{k+1}}{2}$$

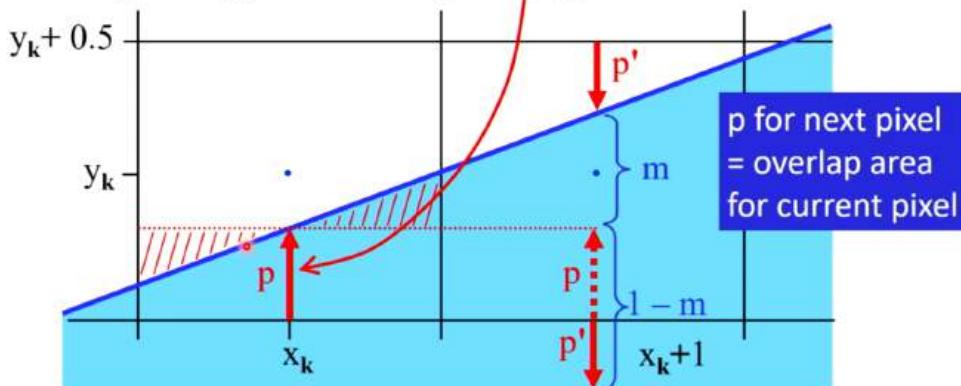
- Das Vorzeichen von  $p'$  hat die gleiche Bedeutung wie das Vorzeichen von  $p_k$ .

- Berechnung des Überdeckungsgrads:**

- $p = p' + (1 - m)$ , wobei  $m$  der Steigungsfaktor ist.
- Der Wert von  $p$  liegt im Bereich  $0 \leq p \leq 1$  und entspricht dem Überdeckungsgrad an der Stelle  $x_k$ .



$$\begin{aligned} p &= p' + (1 - m) = [m(x_k + 1) + b] - (y_k + 0.5) + (1 - m) = \\ &= mx_k + b - y_k + 0.5 = mx_k + b - (y_k - 0.5) \end{aligned}$$



## 5. Effizienz

- **Inkrementelle Berechnung:** Das Antialiasing lässt sich sehr schnell und inkrementell berechnen, was zu einer effizienten Verarbeitung führt.

## 6. Anpassungen für andere Winkel

- **Drehungen und Spiegelungen:** Für andere Winkel werden Drehungen um 90° und/oder Spiegelungen des Verfahrens verwendet.



## Abtastung und Fouriertransformation:

### 1. Fouriertransformation und Frequenzraum

- **Fouriertransformation:** Beschreibt ein Signal im Ortsraum (z.B. eine Scanlinie in einem Bild) als Summe von Sinusschwingungen im Frequenzraum.
  - **Sinusschwingung:** Durch **Frequenz**, **Phase** und **Amplitude** beschrieben.
  - **Spektrum:** Im Frequenzraum wird ein Signal durch sein Spektrum spezifiziert, also Phase und Amplitude in Abhängigkeit von den Frequenzen  $\omega$ .
  - **Euler-Identität:**  $e^{ix} = \cos(x) + i \sin(x)$ , mit der Phase und Amplitude effizient durch imaginäre Zahlen beschrieben werden.
- **Inverse Fouriertransformation:** Wandelt das Spektrum im Frequenzraum zurück in das Signal im Ortsraum.

### 2. Faltung

- **Faltung:** Kombiniert zwei Funktionen und ergibt das integralgewichtete Summenprodukt der beiden.
  - **Formel:**  $f_1 * f_2(x) = \int_{-\infty}^{\infty} f_1(\tau) f_2(x - \tau) d\tau$
- **Faltungstheorem:**
  - Multiplikation zweier Funktionen im Ortsraum entspricht der Faltung ihrer Spektren im Frequenzraum:  

$$f_1 f_2 = F_1 * F_2$$
  - Faltung im Ortsraum entspricht der Multiplikation der Spektren im Frequenzraum:  

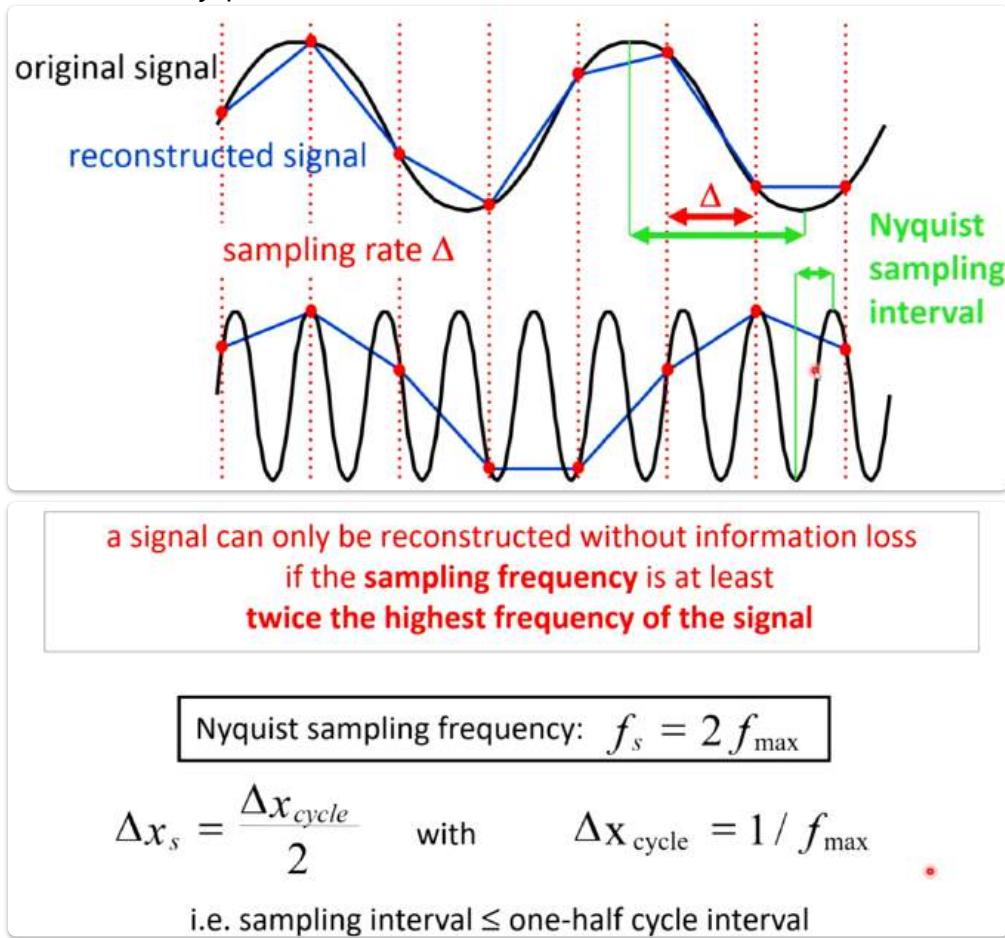
$$f_1 * f_2 = F_1 F_2$$

### 3. Abtasten und Diskretisierung

- **Abtasten im Ortsraum:** Multiplikation des Signals mit einer Kammfunktion  $\text{comb}_T$ .
  - **Frequenzraum:** Entspricht einer Faltung mit der Kammfunktion  $\text{comb}_{1/T}$ .
  - Die Zahnabstände in  $\text{comb}_T$  und  $\text{comb}_{1/T}$  sind invers proportional zueinander.
- **Faltung mit der Kammfunktion:**
  - Führt zu einer **Replikation des Spektrums** im Frequenzraum (Schattenspektra).

### 4. Nyquist-Limit und Aliasing

- **Nyquist-Limit:** Bei zu niedriger Abtastfrequenz (Abtastintervall  $T$  zu groß) sind die Zähne der Kammfunktion  $\text{comb}_T$  im Ortsraum zu weit auseinander und die Replikationen im Frequenzraum (durch  $\text{comb}_{1/T}$ ) zu nah beieinander.
    - Dies führt zu **Aliasing**, da die Schattenspektra sich überlappen und eine fehlerfreie Rekonstruktion unmöglich wird.
- weiteres zu Nyquist: [2. Bildaufnahme](#)



### 5. Rekonstruktion und Schattenspektra

- **Exakte Rekonstruktion:** Um die durch die Diskretisierung entstandenen Schattenspektra zu entfernen, wird das Spektrum der diskretisierten Funktion im Frequenzraum mit einer **Rechteckfunktion** multipliziert.
  - Das ursprüngliche Spektrum bleibt übrig.
- **Rekonstruktion im Ortsraum:**

- **Faltung mit Sinc-Funktion:** Die exakte Rekonstruktion erfolgt durch Faltung im Ortsraum mit der **Sinc-Funktion**:  $\text{Sinc}(x) = \frac{\sin(x)}{x}$

## 6. Praktische Rekonstruktion

- Da die Sinc-Funktion über einen unendlichen Bereich nicht null ist, wird für eine praktikable Rekonstruktion:
  - **Rechteckfunktion** (Nächster-Nachbar-Interpolation) oder
  - **Dreiecksfunktion** (lineare Interpolation) verwendet.

### Fourier Transform

**TU WIEN**

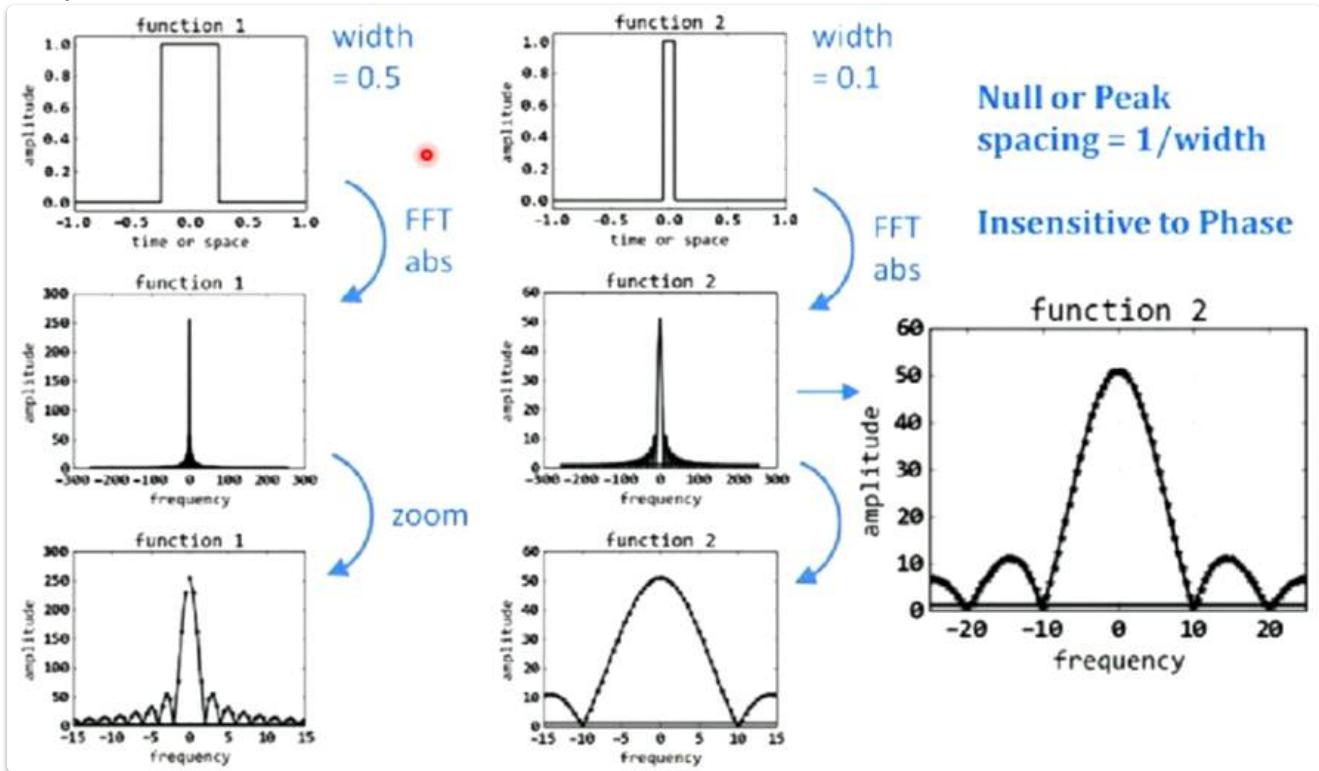
■ Link between spatial  $f(x)$  and frequency  $F(\omega)$  domain

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \omega x} dx$$

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i \omega x} d\omega$$

$$e^{ix} = \cos x + i \sin x$$

Beispiel zur Fourier Transformation:



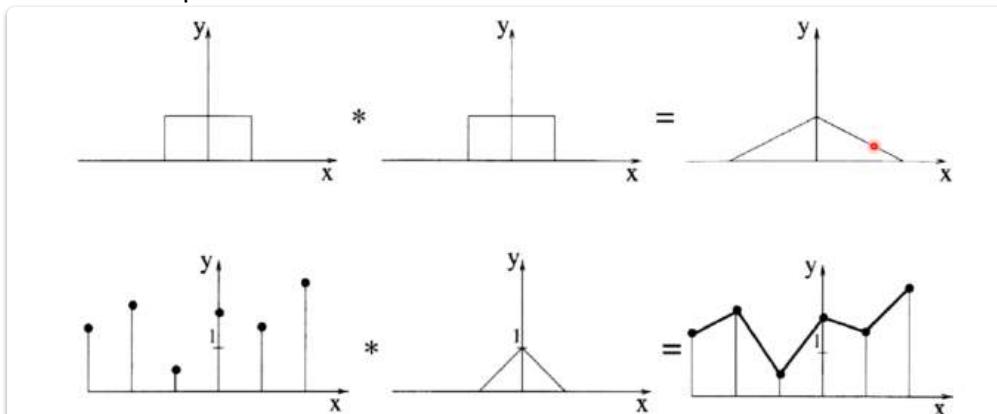
- The spectrum of the convolution of two functions is equivalent to the product of the transforms of both input signals, and vice versa
- Convolution  $f_1 * f_2(x) = \int_{\mathbb{R}} f_1(\tau) f_2(x - \tau) d\tau$

Convolution theorem  $f_1 * f_2 \equiv F_1 F_2$

$$F_1 * F_2 \equiv f_1 f_2$$

"Wenn ich eine Faltung im einen Raum mache (Orts oder Frequenzraum), ist es im anderen Raum dann zu multiplizieren"

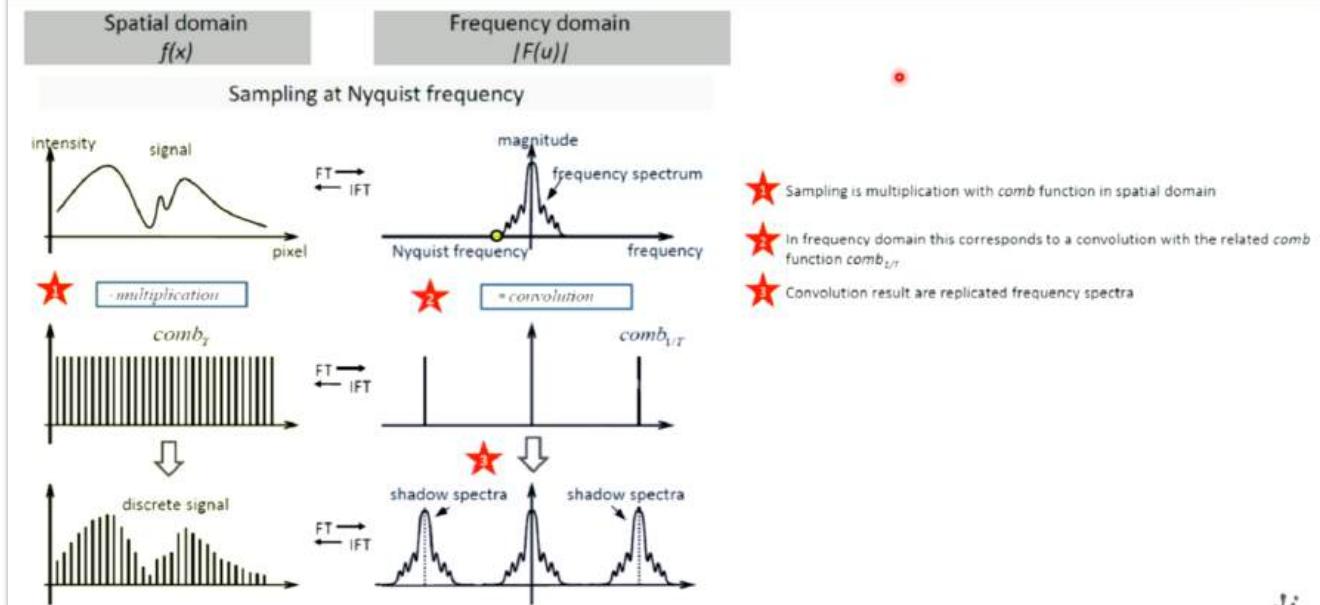
Hier ein Beispiel zu dem Convolution theorem: (Mehr dazu siehe: [7. Globale Operationen](#))



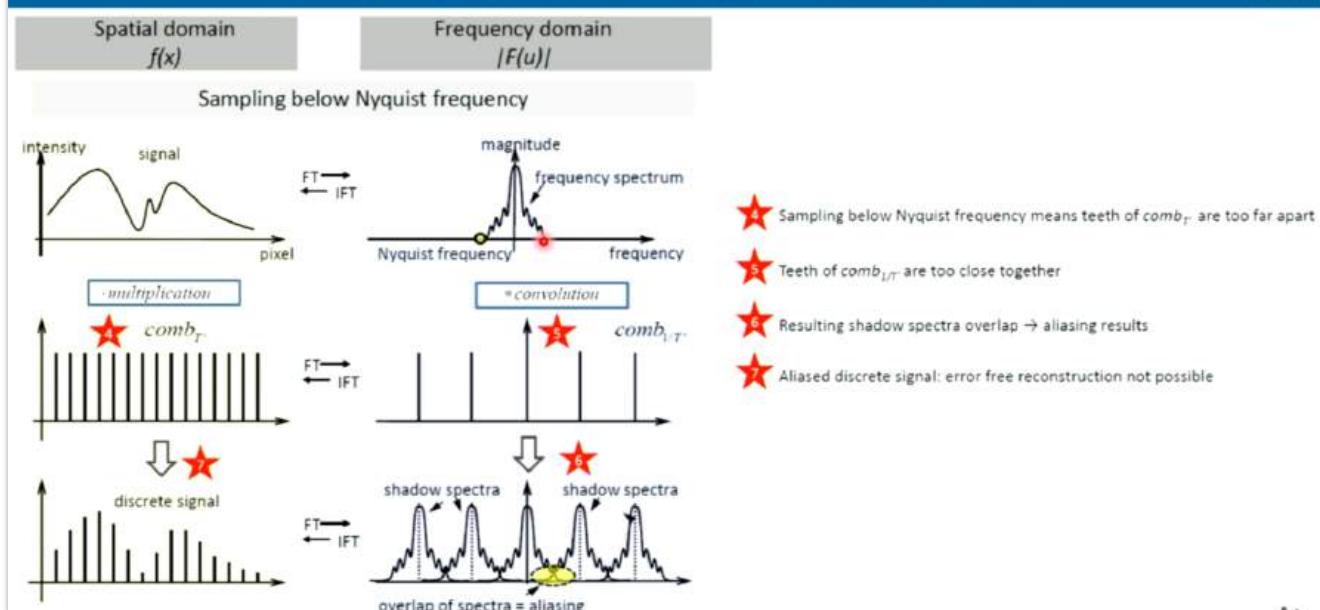
damit das 2. geht, muss der Abstand der Samplepunkten genau doppelt so groß sein wegen Nyquist (das bitte nochmal Fact-checken, er hat schnell gesprochen)

**Cheatsheet für alles was Sampling betrifft:**

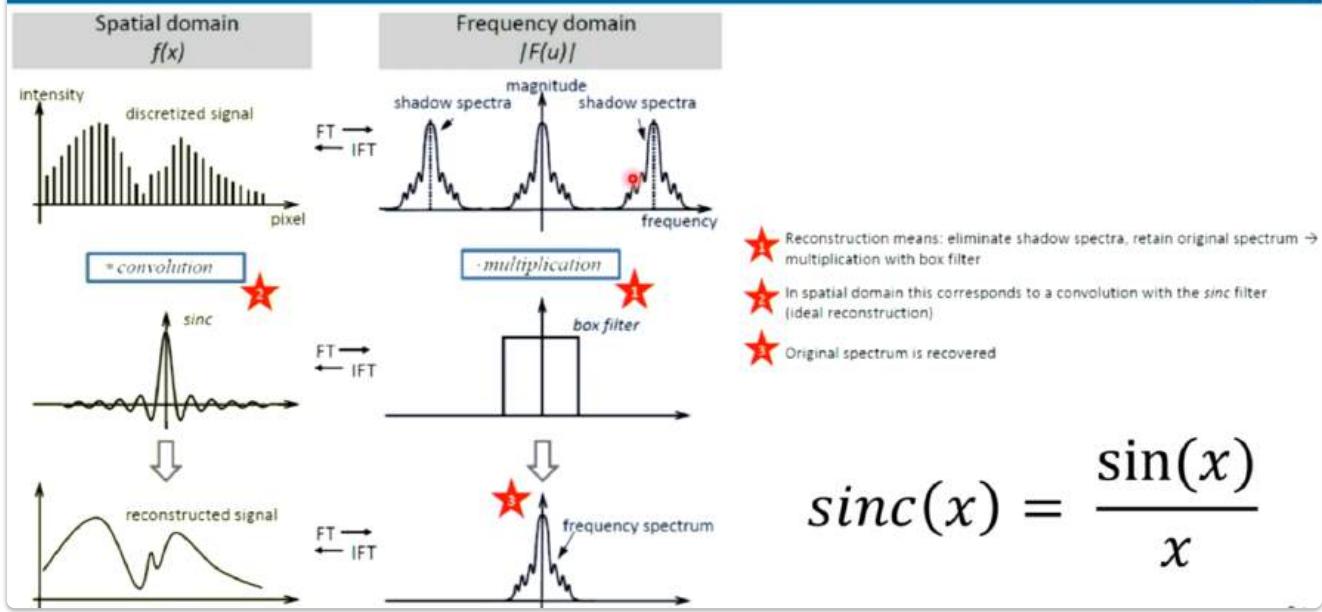
## Sampling at the Nyquist Frequency



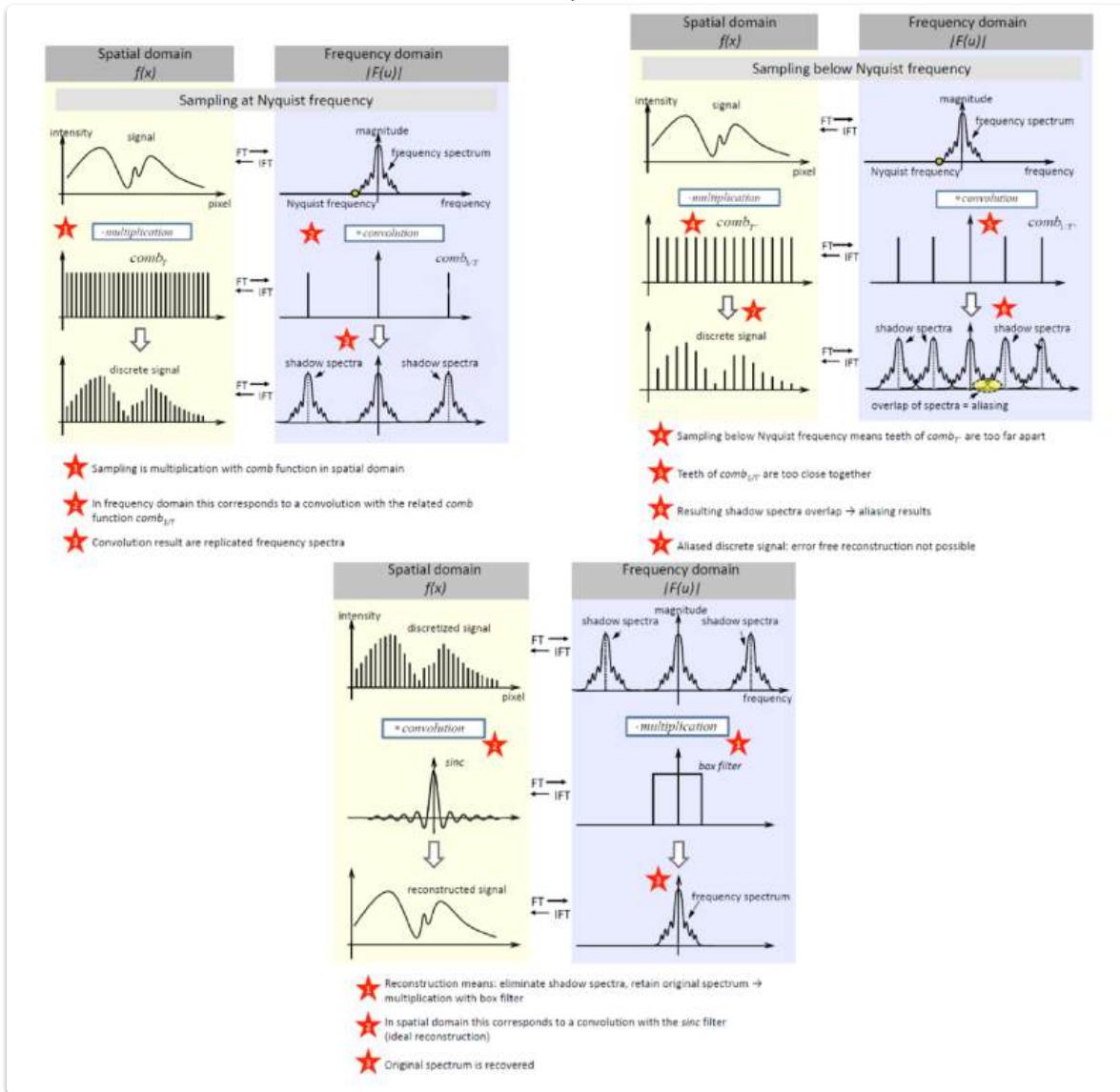
## Sampling Below the Nyquist Frequency



# Reconstruction



hier nochmal hochauflösend aus dem Skriptum:





# 7. Globale Operationen

Information:

- EVC\_Skriptum\_CV, p.35 bis EVC\_Skriptum\_CV, p.39
- 

## Bildtransformationen

### Punkt- und lokale Operationen

- **Bisherige Betrachtung:** Fokus auf **Punkt-** und **lokale Operationen**
  - Arbeitsweise: Entweder **einzelnes Pixel** oder **Pixelumgebung** wird bearbeitet
  - Siehe: [4. Punktoperationen](#), [5. Lokale Operationen](#)

### Globale Operationen

- Nutzen das gesamte **Bild** als Ausgangsbasis
  - Bild wird aus dem **Bildraum** (Ortsraum) in einen anderen Raum (z. B. **Frequenzraum**) transformiert
  - **Vorteile** der Transformation:
    - Bessere Sichtbarkeit von bestimmten Bildmerkmalen
    - Bessere **Daten-Dekorrelation** und Anpassung an das menschliche visuelle System

### Bild-Frequenzraum-Transformationen

- **Ziel:** Darstellung von Bilddaten in einem anderen Raum (z. B. Frequenzraum) zur besseren Analyse
  - Häufig verwendete Transformationen:
    - **Fourier-Transformation**
    - **Cosinus-Transformation**
    - Weitere Transformationen: **Wavelet-, Laplace-Transformation**

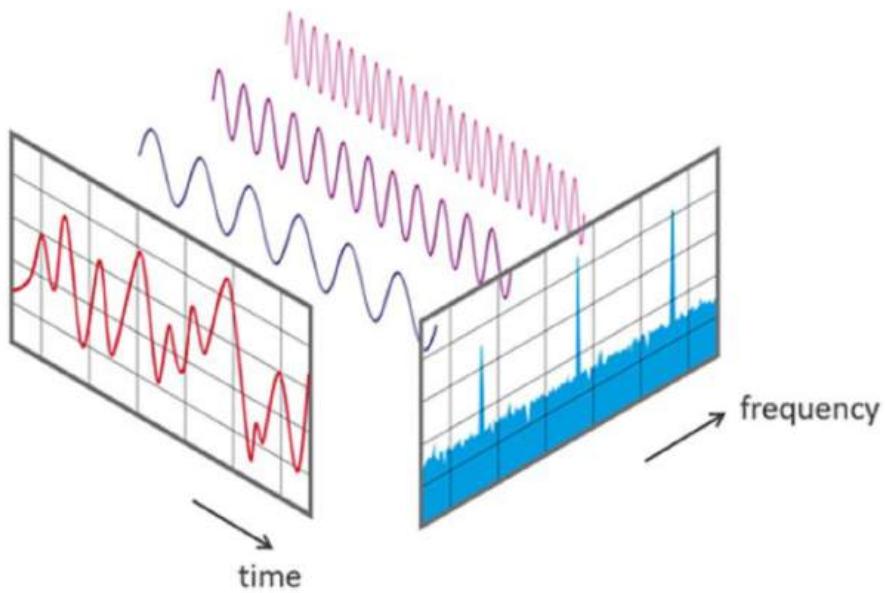
### Fourier- und Cosinustransformation

- **Transformationen im eindimensionalen, kontinuierlichen Bereich:**
  - Betrachtung einer Funktion  $y = f(x)$
  - Anwendung dieser Transformationen auf **zweidimensionale Bilder** (mit den Bildkoordinaten  $(x, y)$ )
  - Bei der **diskreten** Betrachtung:

- **Integrale** werden zu **Summen** (diskrete Werte)

Die Fourier- und Cosinus-Transformation sind die wichtigsten Methoden, um Bilddaten von ihrem Ortsraum in den Frequenzraum zu überführen, was viele Vorteile in der Analyse und Verarbeitung von Bildern bietet.

## Fouriertransformation



### Wichtige Info vorab!

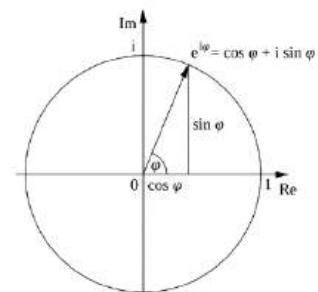
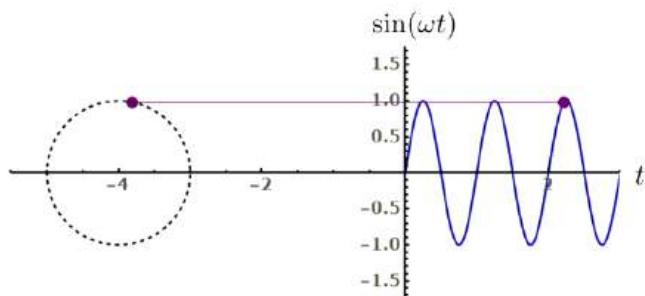
Diese Zusammenfassung bezieht sich hauptsächlich aufs [EVC\\_Skriptum\\_CV.pdf](#) und wird deshalb auch die im Skriptum verwendete Notation verwenden. In den Vorlesungsfolien wurde auf die Eulersche Identität zurückgegriffen um die  $\cos/\sin$  Ausdrücke abzukürzen. Falls beim Test die Notation aus den Slides verwendet wird hier das wichtigste zum herleiten:

### Before we start - some math: Euler's Identity



Leonhard Euler  
1707 - 1783

$$r e^{i \omega t} = r(\cos(\omega t) + i \sin(\omega t))$$



## Frequenzbereich und harmonische Funktionen

- **Zerlegung von Bildsignalen:** Die Darstellung und Analyse von Bildern im Frequenzbereich basiert auf der Zerlegung von Bildsignalen in **harmonische Funktionen** (Sinus- und Kosinusfunktionen).

## Sinus- und Kosinusfunktionen

- **Mathematische Darstellung:**
  - Sinusfunktion:  $f(t) = A \sin(\omega t + \phi)$
  - Kosinusfunktion:  $f(t) = A \cos(\omega t + \phi)$
- **Parameter:**
  - **A:** Amplitude (maximale Auslenkung)
  - **$\phi$ :** Phase (Verschiebung entlang der Zeitachse)

## Beziehung zwischen Frequenz und Kreisfrequenz

- **Kreisfrequenz ( $\omega$ )** und **Frequenz ( $f$ )** sind miteinander verknüpft:  

$$\omega = 2\pi f$$
  - $f$ : Frequenz in Zyklen pro Raum- oder Zeiteinheit, z. B. **1.000 Zyklen pro Sekunde** (Hertz)

## Entstehung des Frequenzkonzepts

- **Ursprung des Frequenzbegriffs:** Das Konzept der Frequenzen und der Zerlegung von Schwingungen in harmonische Funktionen entstand ursprünglich aus der **Akustik** (Schall, Töne und Musik), da das menschliche Ohr ähnlich arbeitet.
  - **Schall:** Befindet sich in der **Zeitdomäne**
  - **Töne:** Haben eine **Lautstärke** (Amplitude) und eine **Frequenz** (Schwingungen pro Sekunde)

## Fouriertransformation

- **Zentrale Aussage:**
  - Wellen, wie z.B. **Schallwellen**, **Wasserwellen** oder **elektromagnetische Wellen**, können mit beliebiger **Frequenz**, **Amplitude** und **Phasenlage** als **Summe von gewichteten Kosinus- und Sinusfunktionen** dargestellt werden.

## Kosinus- und Sinusfunktionen

- **Kosinusfunktion:**
  - $\cos(x)$  hat den Wert **1** am Ursprung ( $\cos(0) = 1$ )
  - Durchläuft von  $x = 0$  bis  $x = 2\pi$  eine volle Periode
- **Sinusfunktion:**
  - $\sin(x)$  hat den Wert **0** am Ursprung ( $\sin(0) = 0$ )

- Durchläuft ebenfalls von  $x = 0$  bis  $x = 2\pi$  eine volle Periode

## Frequenz und Perioden

- **Periodenzahl:**
  - Für  $\cos(x)$  innerhalb einer Strecke der Länge  $T = 2\pi$  ist die Anzahl der Perioden 1:  
 $\omega = \frac{2\pi}{T} = 1$

## Verschiebung der Kosinusfunktion

- **Verschiebung:**
  - Wenn eine Kosinusfunktion entlang der x-Achse um eine Distanz  $\phi$  verschoben wird ( $\cos(x) \rightarrow \cos(x - \phi)$ ), ändert sich die **Phase** der Funktion.
  - $\phi$  bezeichnet den **Phasenwinkel** der resultierenden Funktion.
- **Sinus als verschobene Kosinusfunktion:**
  - Die **Sinusfunktion** ist eine **Kosinusfunktion**, die um  $\frac{\pi}{2}$  (ein Viertelperiode) nach rechts verschoben ist.

## Orthogonalität von Sinus- und Kosinusfunktionen

- **Orthogonalität:**
  - Kosinus- und Sinusfunktionen sind **orthogonal**, was bedeutet, dass sie in ihrer Form unabhängig voneinander sind.
  - Durch die Kombination von Kosinus- und Sinusfunktionen können neue sinusoidale Funktionen mit beliebiger **Frequenz**, **Amplitude** und **Phase** erzeugt werden.

## Addition von Kosinus- und Sinusfunktionen

- **Addition:**
  - Wenn eine Kosinus- und eine Sinusfunktion mit der gleichen Frequenz  $\omega$  und den Amplituden  $A$  bzw.  $B$  addiert werden, entsteht eine neue Sinusfunktion mit der gleichen Frequenz  $\omega$ .
  - **Resultierende Amplitude**  $C$  und **Phasenwinkel**  $\phi$  sind durch die Amplituden  $A$  und  $B$  bestimmt:

$$C = \sqrt{A^2 + B^2}, \quad \phi = \tan^{-1} \left( \frac{B}{A} \right)$$

## Erweiterung auf beliebige Funktionen

- **Jean Baptiste Joseph de Fourier** (1768–1830) zeigte, dass jede **periodische Funktion**  $g(x)$  mit einer Grundfrequenz  $\omega_0$  als **Summe von harmonischen Sinus- und**

Kosinusfunktionen dargestellt werden kann:

$$g(x) = \sum_{k=0}^{\infty} A_k \cos(k\omega_0 x) + B_k \sin(k\omega_0 x)$$

- **Fourierkoeffizienten:**
  - $A_k, B_k$ : Bestimmen das Gewicht der jeweiligen Kosinus- und Sinusfunktionen.
  - Frequenzen der beteiligten Funktionen: Ganzzahlige Vielfache der Grundfrequenz  $\omega_0$
  - **Fourieranalyse**: Berechnung der Fourierkoeffizienten aus der gegebenen Funktion  $g(x)$ .

## Fourierintegral – Nicht periodische Funktionen

- **Fourierintegral**: Erweiterung auf **nicht periodische Funktionen**, die ebenfalls als Summe von Sinus- und Kosinusfunktionen dargestellt werden können:

$$g(x) = \int_0^{\infty} A_{\omega} \cos(\omega x) + B_{\omega} \sin(\omega x) d\omega$$

- **Koeffizienten**  $A_{\omega}$  und  $B_{\omega}$ :
  - Beschreiben die Amplitude der entsprechenden **Kosinus- bzw. Sinusfunktion** bei der Frequenz  $\omega$ .
  - Bestimmung der Koeffizienten:

$$A_{\omega} = \frac{1}{\pi} \int_{-\infty}^{\infty} g(x) \cos(\omega x) dx$$

$$B_{\omega} = \frac{1}{\pi} \int_{-\infty}^{\infty} g(x) \sin(\omega x) dx$$

- **Spektrum der Funktion**:
  - $A(\omega)$  und  $B(\omega)$  sind **kontinuierliche Funktionen**, die das **Spektrum** der Frequenzen im Signal repräsentieren.

## Fouriertransformation und Fourier-Spektrum

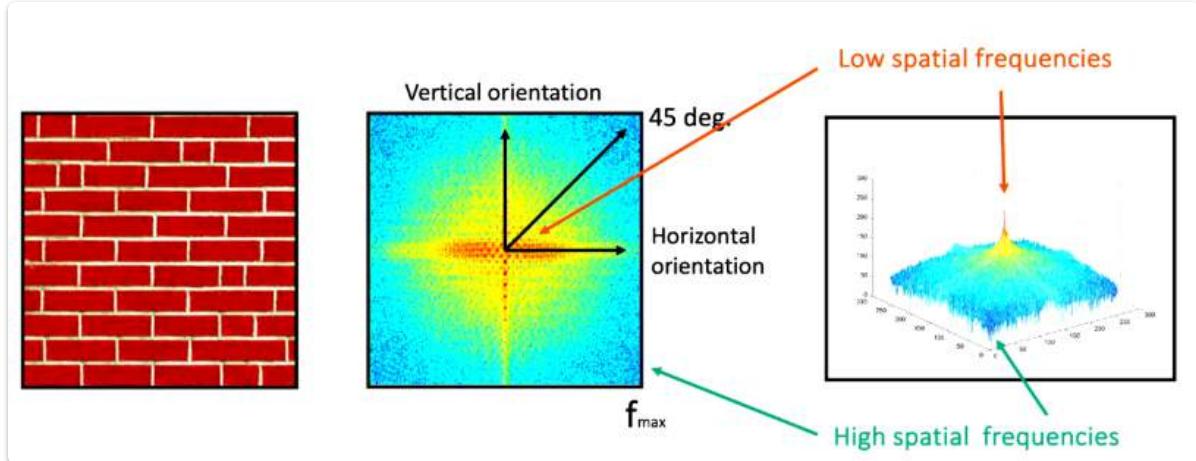
- **Fouriertransformation**:
  - Vereinfacht die Darstellung, indem die Ausgangsfunktion  $g(x)$  und das Spektrum als **komplexwertige Funktionen** betrachtet werden.
- **Fourierspektrum**  $G(\omega)$ :
  - Wird als **komplexe Funktion** dargestellt:

$$G(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (\cos(\omega x) - i \sin(\omega x)) g(x) dx$$

- **Inverse Fouriertransformation**:
  - Umkehrung der Fouriertransformation zur Rekonstruktion der ursprünglichen Funktion  $g(x)$  aus ihrem Spektrum  $G(\omega)$ .

- **Dualität von Orts- und Frequenzraum:**

- Die Darstellung von Funktionen im Ortsraum und im Frequenzraum ist dual – sie beschreiben das gleiche Signal, jedoch auf unterschiedliche Weise.



Da das abstrakte Prinzip von Fouriertransformation Anfangs vielleicht bisschen unübersichtlich ist empfehle ich folgende Videos:

- [https://www.youtube.com/watch?v=7IIR\\_BRkfPI](https://www.youtube.com/watch?v=7IIR_BRkfPI) (Kleine Einführung)
- <https://www.youtube.com/watch?v=spUNpyF58BY&t=477s> (Wurde auch in Vorlesung gezeigt)

## Diskrete Fourier-Transformation (DFT)

### Notwendigkeit der DFT

- **Kontinuierliche Fouriertransformation (FT)** ist für die numerische Berechnung am Computer nicht geeignet, da Bilder immer diskrete Daten enthalten.
- Die **Diskrete Fourier-Transformation (DFT)** stellt sowohl das Signal als auch sein Spektrum als **endliche Vektoren** dar, die eine **diskrete, periodische Signal** repräsentieren.

### DFT: Vorwärtstransformation

Für ein diskretes Signal  $g(u)$  der Länge  $M$  (mit  $u = 0, \dots, M - 1$ ), wird das **Fourierspektrum**  $G(m)$  für  $m = 0, \dots, M - 1$  durch die **Vorwärtstransformation** berechnet:

$$G(m) = \frac{1}{\sqrt{M}} \sum_{u=0}^{M-1} g(u) e^{-i \frac{2\pi m u}{M}}$$

### DFT: Inverse Transformation

Die **inverse DFT** zur Rekonstruktion des Signals  $g(u)$  aus dem Spektrum  $G(m)$ :

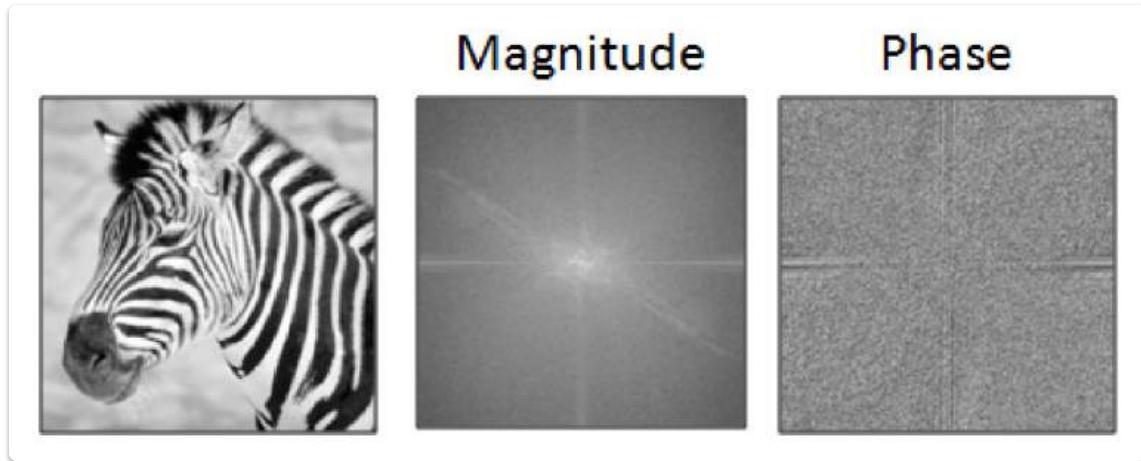
$$g(u) = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} G(m) e^{i \frac{2\pi mu}{M}}$$

- Beide Transformationen sind identisch: Vorwärts- und inverse DFT sind mathematisch symmetrisch.

## Eigenschaften des Fourierspektrums

- Sowohl das Signal  $g(u)$  als auch das Fourierspektrum  $G(m)$  sind komplexwertige Vektoren der Länge  $M$ .
- Betrag des Fourierspektrums (Magnitude):

$$\|G(m)\| = \sqrt{G_{\text{real}}^2(m) + G_{\text{imag}}^2(m)}$$



Wird als Leistungsspektrum (Powerspectrum) bezeichnet und beschreibt die Energie oder Leistung, die jede Frequenzkomponente zum Signal beiträgt.

- Leistungsspektrum:
  - Reellwertig und positiv.
  - Wird häufig zur grafischen Darstellung der Fouriertransformierten verwendet.
  - Phaseninformation geht verloren – das Signal kann daher nicht allein aus dem Leistungsspektrum rekonstruiert werden.
  - Unbeeinflusst von Verschiebungen des Signals: Ein zyklisch verschobenes Signal hat das gleiche Leistungsspektrum wie das ursprüngliche Signal.

## Fast Fourier Transform (FFT)

- Schnelle Berechnung der DFT mit der Fast Fourier Transform (FFT):
  - Reduzierte Zeitkomplexität von  $O(M^2)$  auf  $O(M \log_2 M)$ .
  - Wesentliche Zeitersparnis bei größeren Signallängen, z.B., bei  $M = 10^6$  wird die Berechnungszeit um den Faktor 10.000 reduziert.

## Convolution Theorem:

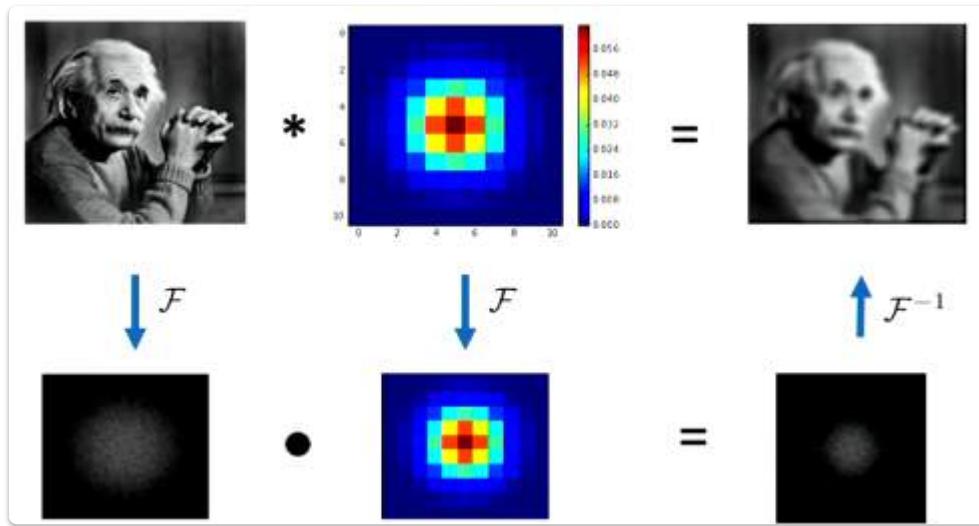
Das **Convolution Theorem** besagt, dass die Fouriertransformierte der Faltung zweier Funktionen im Zeitbereich gleich der Punktweise-Multiplikation ihrer Fouriertransformierten im Frequenzbereich ist. (Mehr dazu siehe [7. Clipping und Antialiasing](#))

## Mathematisch:

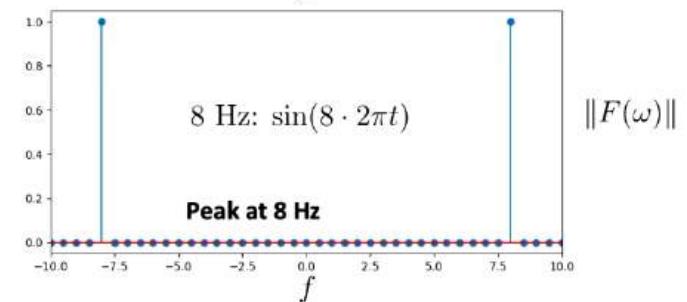
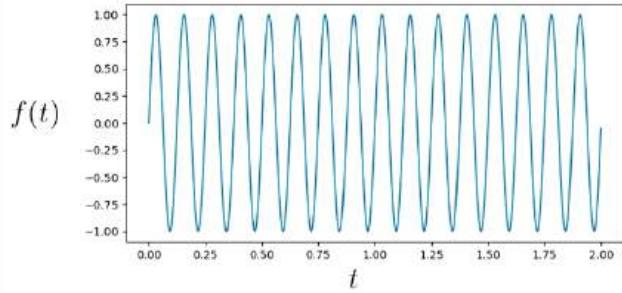
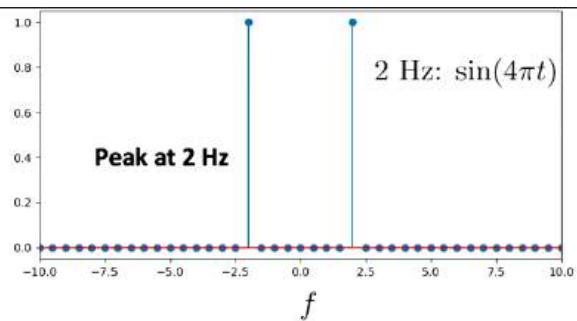
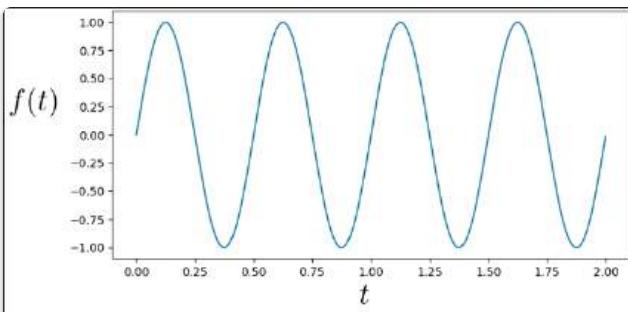
$$\mathcal{F}\{f * g\}(\omega) = F(\omega) \cdot G(\omega)$$

## Das bedeutet:

- Im Zeitbereich: Faltung der Funktionen  $f(t)$  und  $g(t)$ .
- Im Frequenzbereich: Punktweise Multiplikation der Fouriertransformierten  $F(\omega)$  und  $G(\omega)$



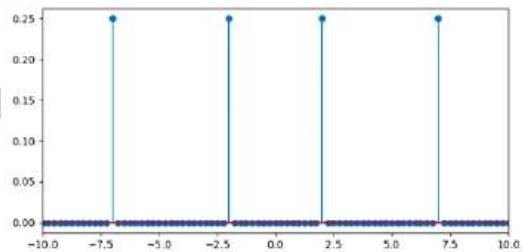
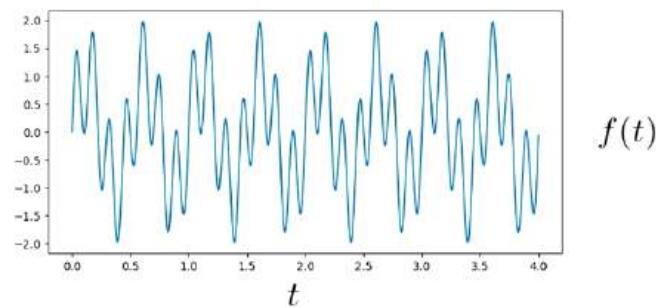
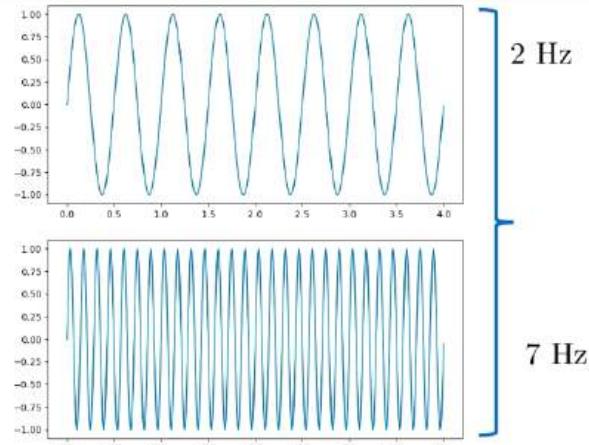
## Hier noch einige Beispiele für Fourier Transformationen:



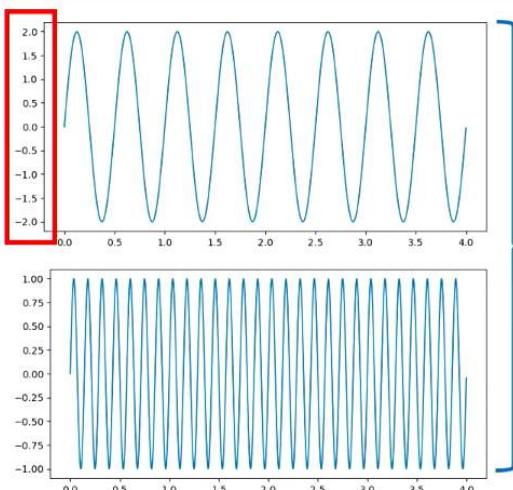
*Let's look at some fourier transformations*

$$\omega = 2\pi f$$

21



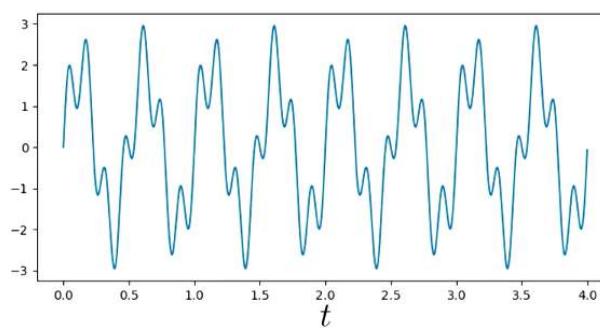
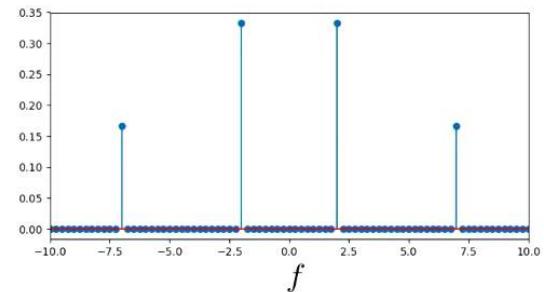
*What about a sum of sines?*



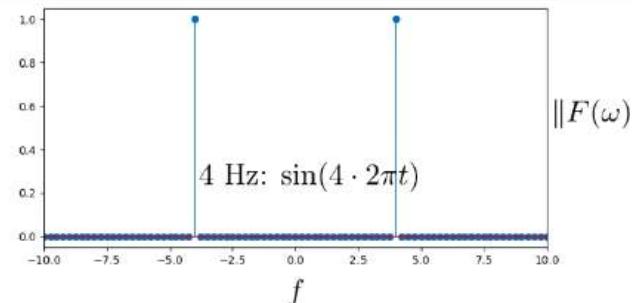
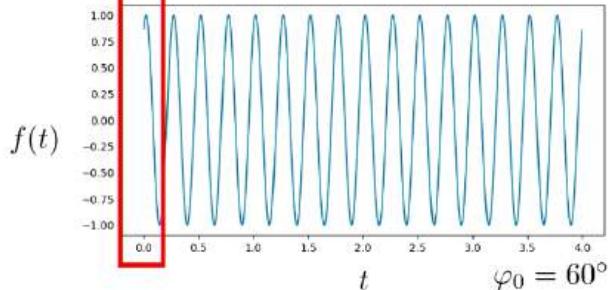
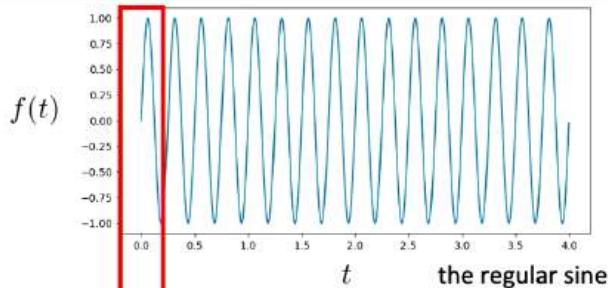
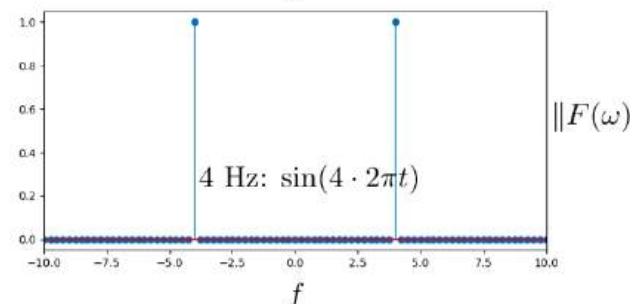
Amplitude 2!

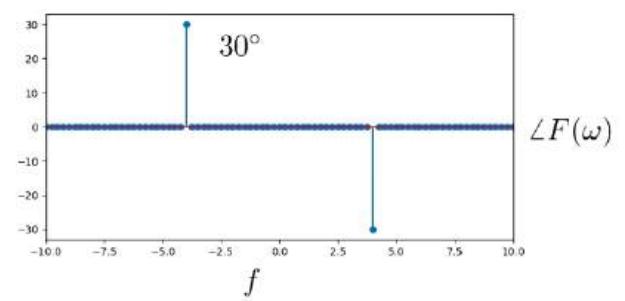
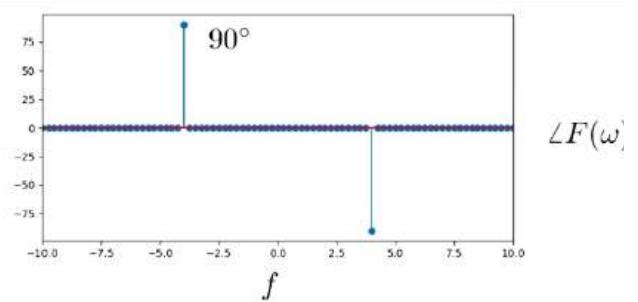
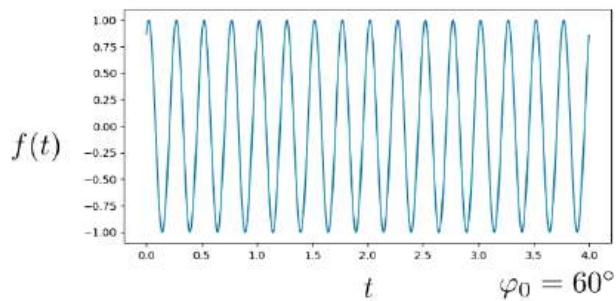
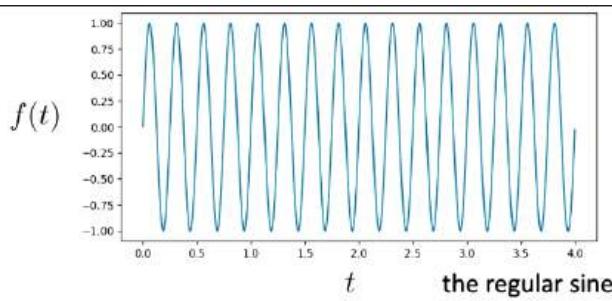
2 Hz

7 Hz

 $\|F(2\pi f)\|$ 

What about a sum of sines?

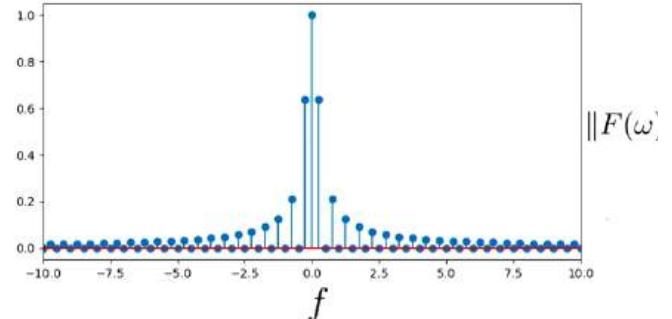
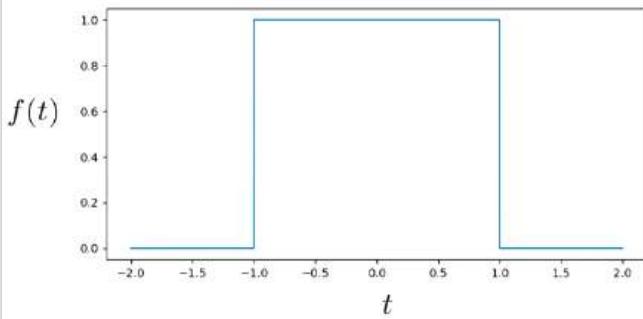
4 Hz:  $\sin(4 \cdot 2\pi t)$ 4 Hz:  $\sin(4 \cdot 2\pi t)$ Let's look at a sine which starts at  $\varphi_0 = 60^\circ$



Let's look at a sine which starts at  $\varphi_0 = 60^\circ$

We can reconstruct offsets via the phase!

25



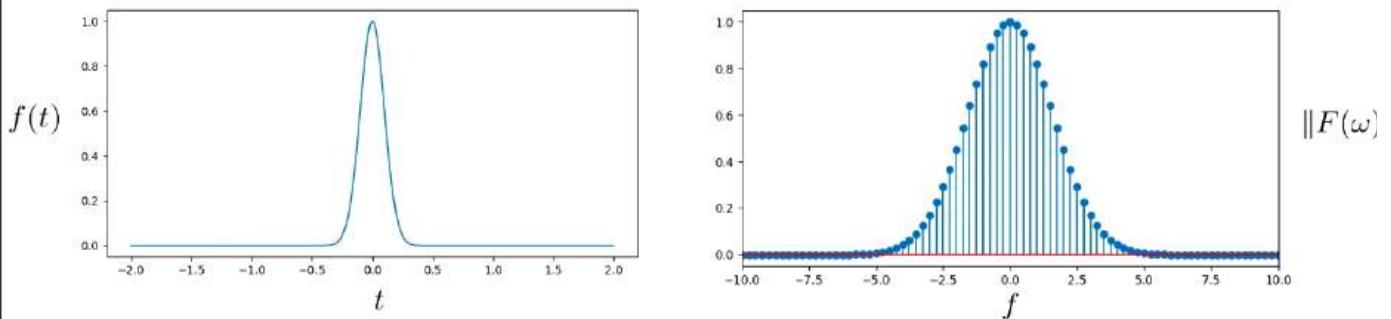
„bumping“ (harmonics) - approaches zero at  $\infty$

Sinc-function (Spaltfunktion):

$$\text{si}(x) = \frac{\sin(\pi x)}{\pi x}$$

Fourier Transformation of a Rectangle

## Gaussian with $\mu = 0, \sigma = 0.1$

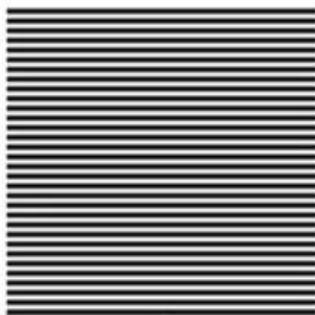


A Gaussian's spectrum is a Gaussian again! (but with  $\sigma$  rescaled)

## *Fourier Transformation of a Gaussian*

Jetzt hier 2D Beispiele:

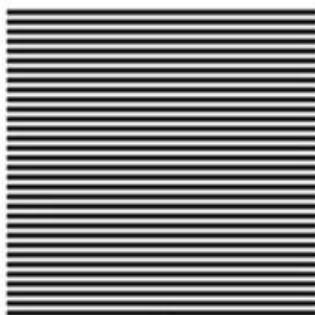
This image exclusively has **32 cycles in vertical direction.**



This image exclusively has **8 cycles in horizontal direction.**



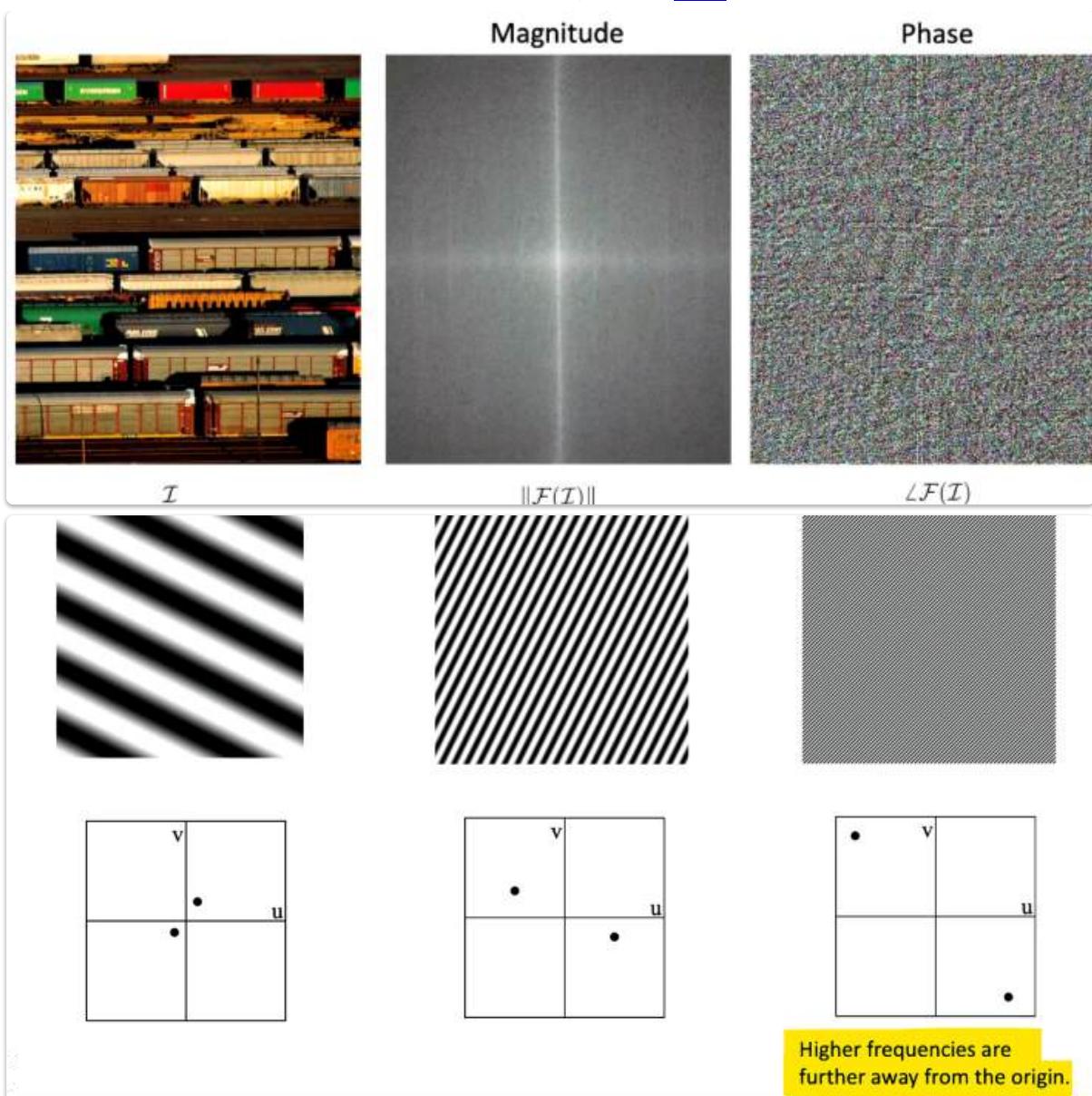
This image exclusively has **32 cycles in vertical direction.**

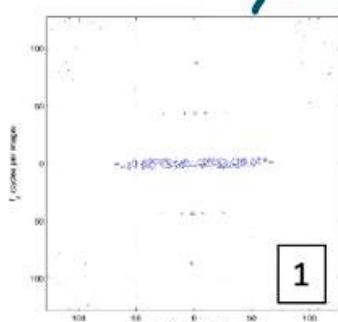
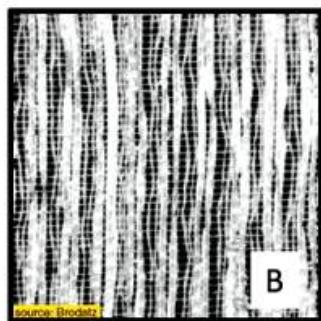


This image exclusively has **8 cycles in horizontal direction.**

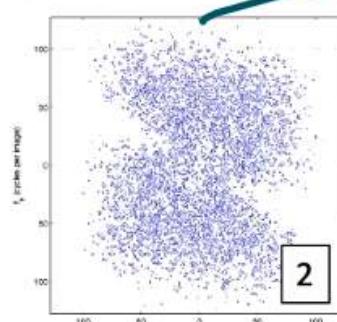


Jetzt hier Beispiel für ein weiteres Bild:

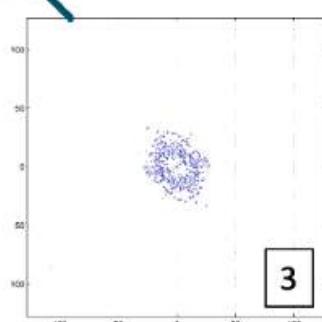


**Testähnliches Beispiel:**

fx(cycles/image pixel size)



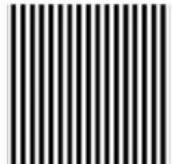
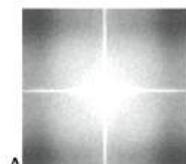
fx(cycles/image pixel size)



fx(cycles/image pixel size)

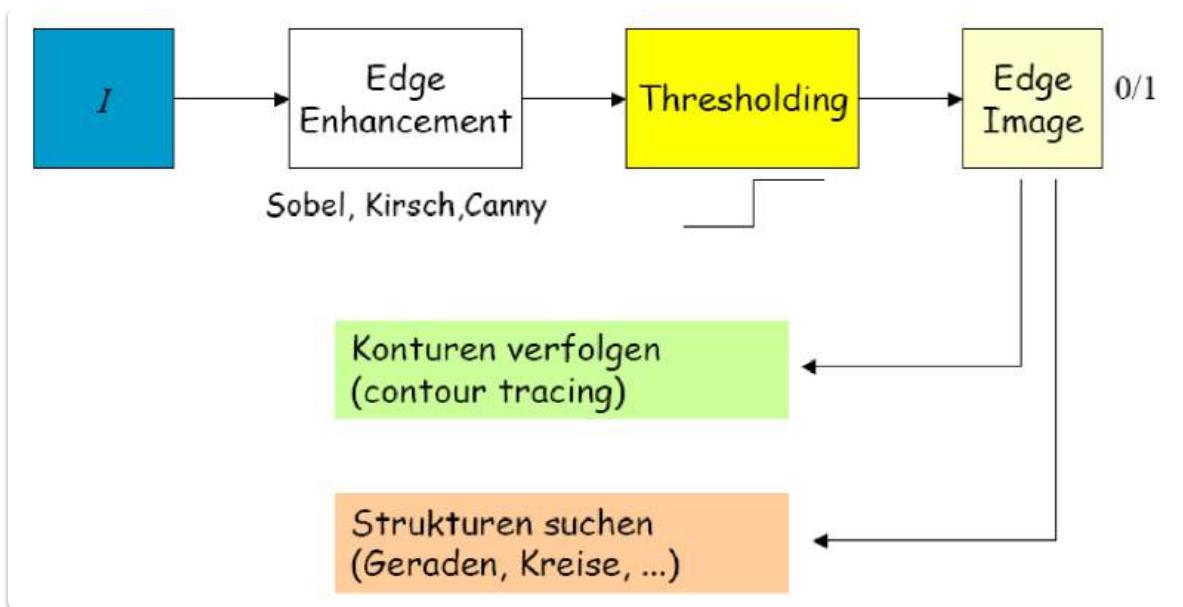
**Beispiel Globale Operationen**

- Die Bilder A-D zeigen den logarithmierten Betrag des Fourier-Spektrums eines Bildes. Ordnen Sie die Eingabebilder  $I_1$  bis  $I_4$  dem richtigen Spektrum aus A bis D zu.

DFT( $I_1$ ) = \_\_\_\_\_DFT( $I_2$ ) = \_\_\_\_\_DFT( $I_3$ ) = \_\_\_\_\_DFT( $I_4$ ) = \_\_\_\_\_

Richtige Antwort: BCAD

**Hough Transformation**



## Einführung in die Hough-Transformation

- **Ziel:** Analyse von Bildinhalten nach Kantendetektion, um größere Strukturen zu finden.
- **Problem der Kantendetektion:** Viele vermeintliche Kantenpunkte gehören in Wirklichkeit nicht zu echten Kanten und umgekehrt fehlen echte Kantenpunkte.
- **Lösungsansatz:** Kantenpunkte werden zu größeren Strukturen zusammengeführt (Konturen von Objekten), jedoch gibt es bei Unterbrechungen und Verzweigungen Schwierigkeiten.

## Prinzip der Hough-Transformation

- **Begriff:** Methode zur Erkennung von geometrischen Formen in Punktverteilungen (z.B. Geraden, Kreise, Ellipsen).
- **Besonderheit:** Eignet sich besonders für die Analyse von Bildern mit geometrischen Formen, die in menschlich geschaffenen Objekten häufig vorkommen.
- **Geradenerkennung in binären Kantenbildern:**
  - Eine Gerade in 2D lässt sich mit zwei Parametern beschreiben (z.B.  $y = kx + d$  mit Steigung  $k$  und Schnittpunkt  $d$ ).
  - Ziel: Finden der Geradenparameter  $k$  und  $d$ , auf denen viele Kantenpunkte liegen.
  - **Problem:** Alle möglichen Geraden zu berechnen, die durch einen Punkt laufen, wäre ineffizient.

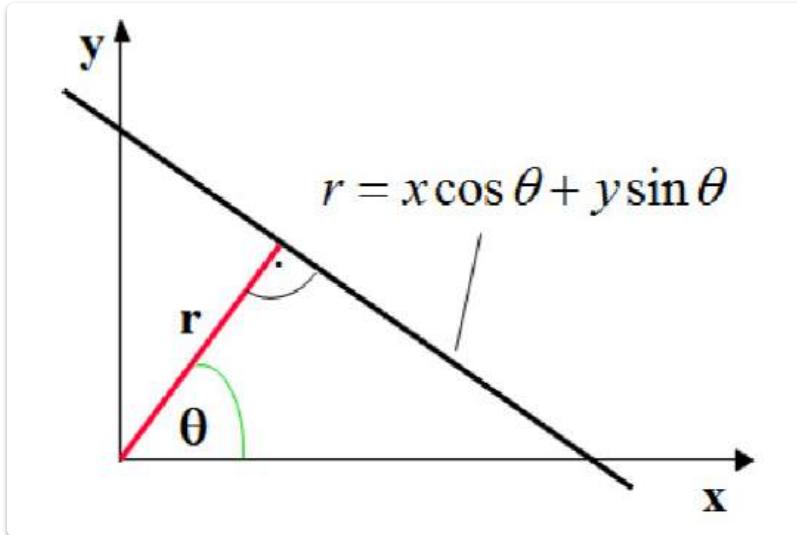
## Umgekehrter Ansatz der Hough-Transformation

- **Geradendarstellung:** Statt klassischer Form  $y = kx + d$  wird die Parameterform (Hessesche Normalform) verwendet:

$$r = x \cos(\theta) + y \sin(\theta)$$

- **Parameter:**

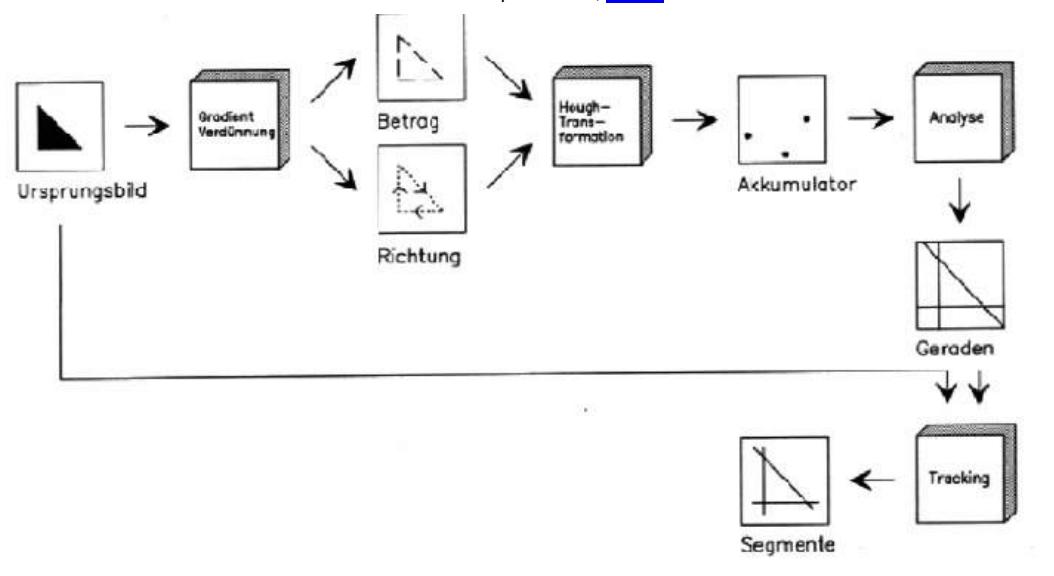
- $r$ : Normalabstand der Geraden zum Ursprung.
- $\theta$ : Winkel des Normalabstands zur x-Achse.



- **Erklärung:** Jede Gerade, die durch einen Punkt  $p = (x, y)$  läuft, erfüllt die Gleichung. Dadurch entstehen unendlich viele mögliche Geraden, die durch den Punkt verlaufen, mit variierenden Parametern  $r$  und  $\theta$ .
- **Hough-Raum:** Der Parameterraum  $(r, \theta)$  wird als Hough-Raum bezeichnet. Jeder Punkt im Hough-Raum entspricht einer Geraden im Bild.

## Vorgehensweise bei der Hough-Transformation

- **Kantenbild erstellen:** Aus dem Eingangsbild  $I(u, v)$  wird ein Kantenbild erzeugt, wobei der Betrag und die Richtung des Gradienten (z.B. durch Sobel-Filter) berechnet wird.
- **Akkumulator-Array:**
  - Ein zweidimensionaler Akkumulator (Array) wird erzeugt, um den  $(r, \theta)$ -Raum zu diskretisieren.
  - Für  $r$  und  $\theta$  werden festgelegte Schritte (z.B.  $r$  in Pixeln,  $\theta$  in  $1^\circ$  oder  $5^\circ$ ) verwendet.
- **Eintragen der Kantenpunkte:**
  - Jeder Kantenpunkt wird in den Hough-Raum überführt und im Akkumulator eingetragen. Die Zellen des Akkumulators werden entsprechend inkrementiert.
- **Schwellwertanalyse:**
  - Wenn der Wert einer Akkumulator-Zelle einen bestimmten Schwellwert überschreitet, repräsentiert diese Zelle eine mögliche Gerade im Bild.



## Nachbearbeitung der Ergebnisse

- **Tracking der Geraden:** Um die Anfangs- und Endpunkte der gefundenen Geraden zu ermitteln, wird ein Tracking-Verfahren angewendet.
  - Dazu wird das Originalbild entlang der gefundenen Geraden abgefahren, und es wird nach Stellen gesucht, an denen die Grauwertdifferenz den Schwellwert überschreitet.
  - An diesen Stellen befinden sich mit hoher Wahrscheinlichkeit Objektkonturen.

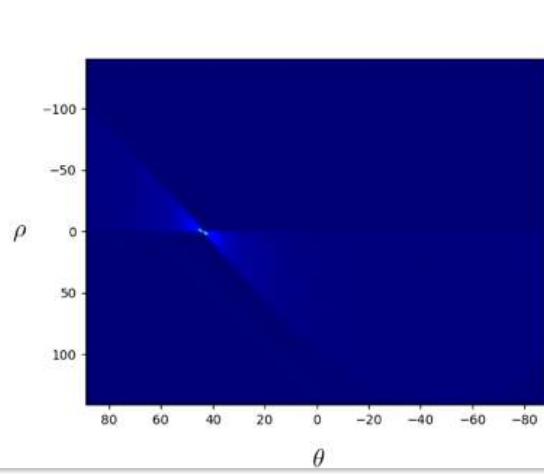
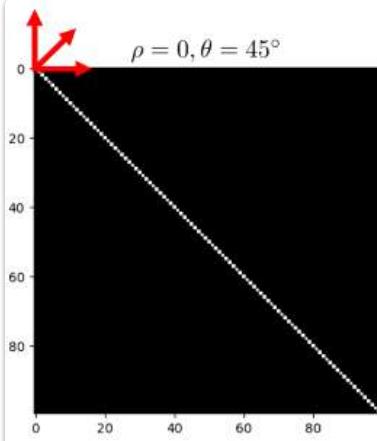
## Vorteile der Hough-Transformation

- **Berücksichtigung der Kantenrichtung:** Die Richtung der Kanten im Originalbild ist bereits bekannt, was die Effizienz steigert.
- **Verwendung von Gradienteninformationen:** Die Hough-Transformation nutzt den Betrag und die Richtung der Kantenpixel für eine genauere Analyse.

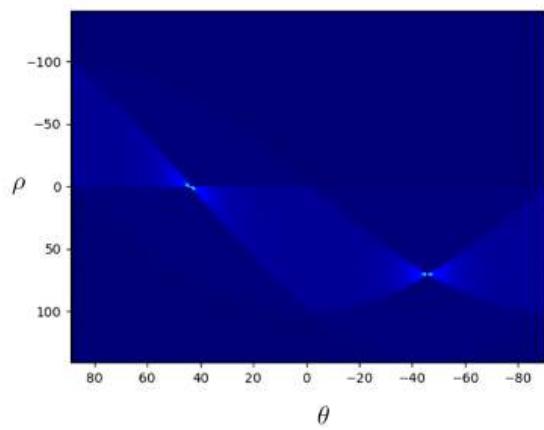
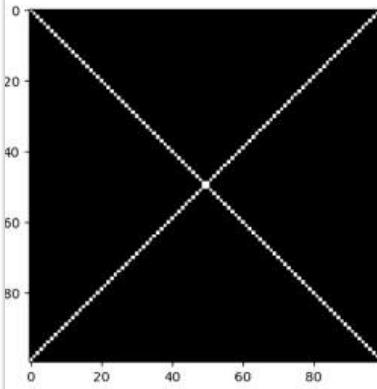
## Einschränkungen

- **Komplexität:** Wenn Filter höherer Ordnung verwendet werden, müssen alle möglichen Geraden durch jeden Kantenpunkt eingetragen werden, was zu einer sinusförmigen Linie im Akkumulator führt und die Berechnungen erheblich verlangsamen kann.

## Beispiele aus vo:

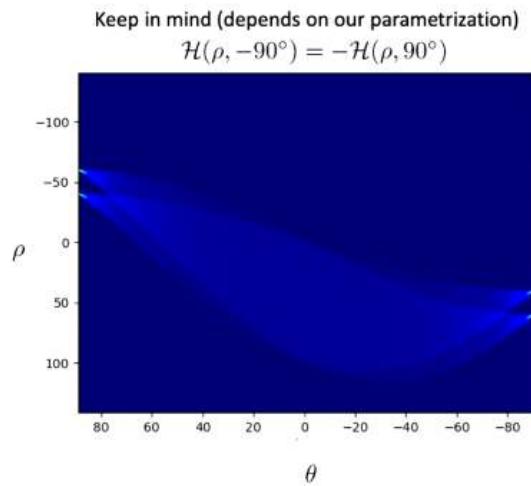
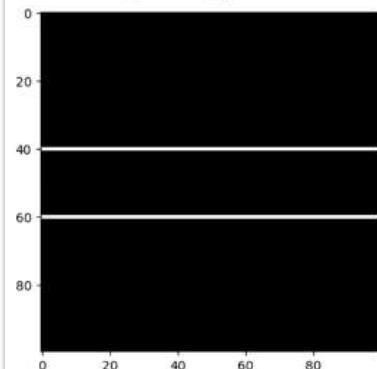


$$\rho_2 = \sqrt{(50^2 + 50^2)} \approx 70, \theta_2 = -45^\circ$$

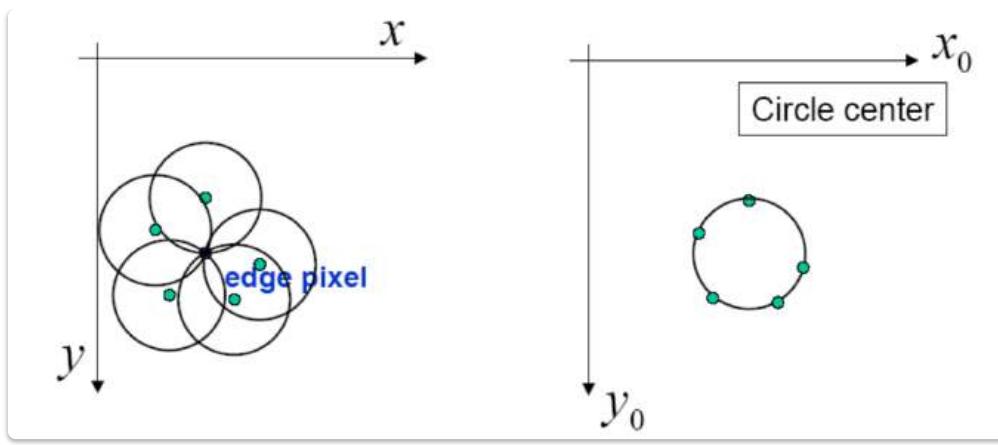


$$\theta_1 = \theta_2 = 90^\circ$$

$$\rho_1 = 40, \rho_2 = 60$$



## Weitere Hough Transformationen



## 1. Prinzip der Kreiserkennung

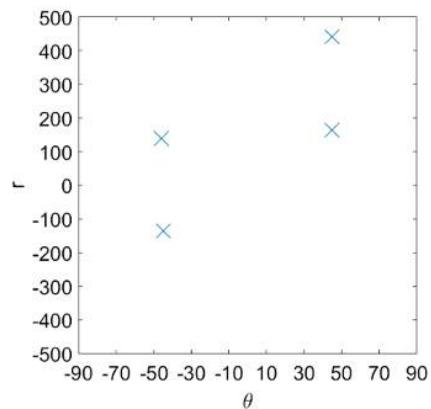
- **Ziel:** Erkennung von Kreisen im Bild.
- **Ähnlichkeit zur Geradenerkennung:**
  - Durch jeden Punkt im Originalbild wird ein Kreis mit einem festgelegten Radius gelegt.
  - Die Kreise im Hough-Transformationsraum schneiden sich in bestimmten Punkten, die Kandidaten für die Mittelpunkte der Kreise im Originalbild sind.
- **Bestimmung der Kreismittelpunkte:**
  - Die Punkte im Hough-Raum, an denen sich die meisten Kreise schneiden, entsprechen den Kreismittelpunkten.
  - Diese Punkte werden zurück in den Originalbereich transformiert, um die tatsächlichen Mittelpunkte der Kreise zu ermitteln.

## 2. Erweiterung auf Ellipsenerkennung

- **Ellipsenparameter:**
  - Eine Ellipse wird nicht nur durch den Mittelpunkt  $(x, y)$  beschrieben, sondern auch durch zwei Halbachsen  $a$  und  $b$ .
  - Dadurch ergibt sich ein 4-dimensionaler Hough-Raum, der berücksichtigt werden muss.
- **Hough-Raum:**
  - Für die Ellipsenerkennung müssen vier Dimensionen im Hough-Raum verwendet werden: die beiden Koordinaten des Mittelpunkts und die beiden Halbachsen.
  - Die Transformation ist komplexer als bei der Kreiserkennung, da sie mehr Parameter berücksichtigt.

**Testähnliches Beispiel Hough Transformationen:**

- Bei der Hough-Transformation zur Detektion von Linien werden diese in Hessescher Normalform repräsentiert:  $r = x \cos(\theta) + y \sin(\theta)$ . Im unten stehenden Diagramm sind 4 detektierte Linien im Hough-Raum (Akkumulator-Array) mit einem "X" markiert. Welche der folgenden Aussagen sind hier wahr bzw. falsch?

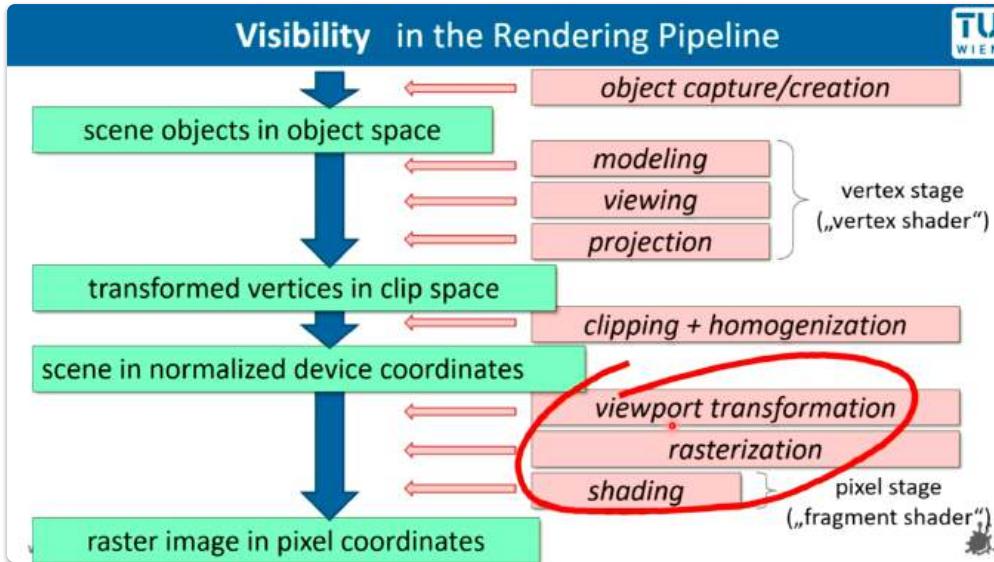


Alle 4 Linien sind parallel zueinander	<input type="checkbox"/> wahr	<input checked="" type="checkbox"/> falsch
Mindestens eine Linie verläuft horizontal	<input type="checkbox"/> wahr	<input checked="" type="checkbox"/> falsch
Mindestens eine Linie verläuft vertikal	<input type="checkbox"/> wahr	<input checked="" type="checkbox"/> falsch
Die Start- und Endpunkte der Linien lassen sich aus dem Hough-Raum nicht bestimmen	<input checked="" type="checkbox"/> wahr	<input type="checkbox"/> falsch
Linien werden durch lokale Maxima im Hough-Raum repräsentiert	<input checked="" type="checkbox"/> wahr	<input type="checkbox"/> falsch



# 8. Sichtbarkeitsverfahren

Das Sichtbarkeitsverfahren kommt hier in dem Bereich der Rendering Pipeline vor:



## 1. Ziel von Sichtbarkeitsverfahren

- **Ziel:** Korrekte und glaubwürdige Darstellung von Szenen, indem unsichtbare Teile der Objekte weggelassen werden.
  - **Unsichtbare Teile:** Rückseiten von Objekten und Teile, die von anderen Objekten verdeckt werden.
- **Begriff:** Hidden-Line- oder Hidden-Surface Eliminierung.

## 2. Ansätze in der Sichtbarkeitsberechnung

- **Objektraum-Methoden:**
  - **Vorgehen:** Vergleichen der Lage der Objekte miteinander.
  - **Ziel:** Nur die vorderen (sichtbaren) Teile der Objekte werden gezeichnet.
- **Bildraum-Methoden:**
  - **Vorgehen:** Für jeden Bildteil wird separat berechnet, was an dieser Stelle sichtbar ist.

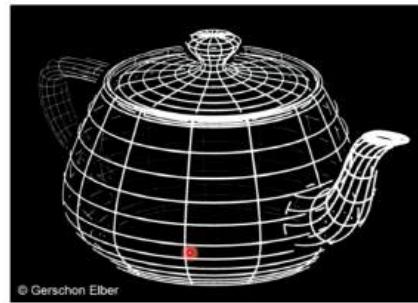
## 3. Berücksichtigung von Transparenz

- Die Erläuterungen zu den Sichtbarkeitsverfahren berücksichtigen **nicht** transparente Objekte.

## Depth Cueing

## 3D Display: Depth Cueing + Visibility

- only visible lines
- intensity decreases with increasing distance



Da geht's darum bei Objekten unsichtbare Polygone anders darzustellen

## Backface Detection (Backface Culling)

### 1. Ziel von Backface Culling

- **Ziel:** Elimination von Polygonen, die sicher nicht sichtbar sind, um den Aufwand nachfolgender Verarbeitungsschritte zu reduzieren.
  - **Nicht sichtbare Polygone:** Polygone, deren Oberflächennormale vom Betrachter weg zeigen.

### 2. Funktionsweise

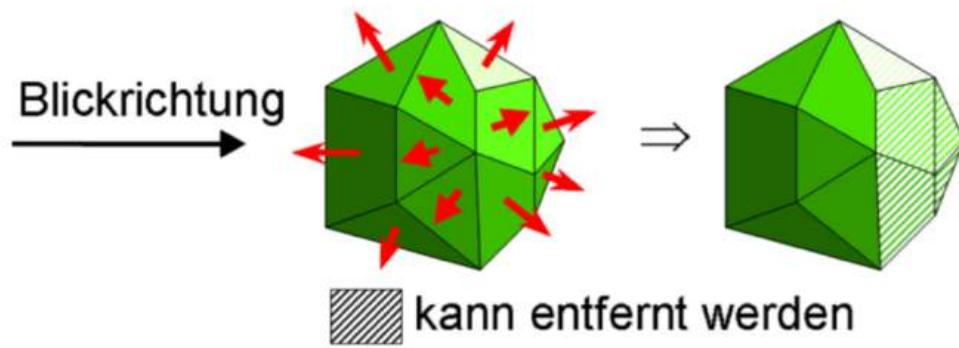
- **Backface Culling** ist kein vollständiges Sichtbarkeitsverfahren, sondern dient als Optimierungsschritt.
  - **Durchschnittliche Reduktion:** Etwa 50% der Polygone werden entfernt.

### 3. Berechnungsverfahren

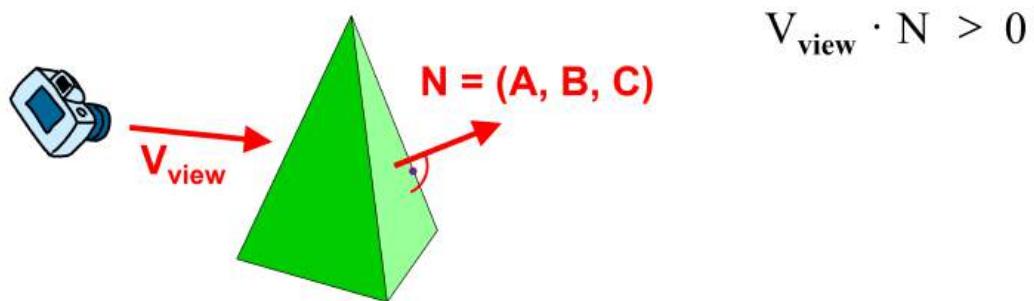
- **Orthographische Projektion:**
  - **Berechnung:** Das Skalarprodukt des Blickrichtungsvektors  $V_{view}$  und der Oberflächennormale  $N$  wird berechnet.
  - **Formel:**  $V_{view} \cdot N > 0 \Rightarrow$  Polygon ist unsichtbar.
- **Perspektivische Projektion:**
  - **Berechnung:** Der Blickpunkt  $(x, y, z)$  wird in die Ebenengleichung eingesetzt.
  - **Formel:**  $Ax + By + Cz + D < 0 \Rightarrow$  Polygon ist unsichtbar.

### 4. Annahmen

- Die gleichen Annahmen wie bei der Verwendung von „Polygonlisten“ werden zugrunde gelegt.



eliminating back faces of closed polyhedra  
 view point  $(x, y, z)$  “inside” a polygon surface if  
 $Ax + By + Cz + D < 0$   
 or polygon with normal  $N=(A, B, C)$  is a back face if

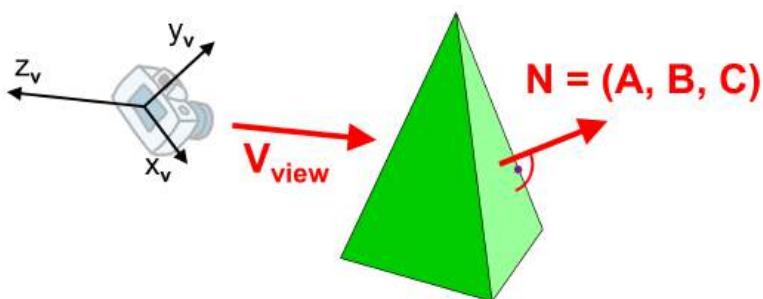


object description in viewing coordinates  $\Rightarrow V_{\text{view}} = (0, 0, V_z)$

$$V_{\text{view}} \cdot N = V_z C$$

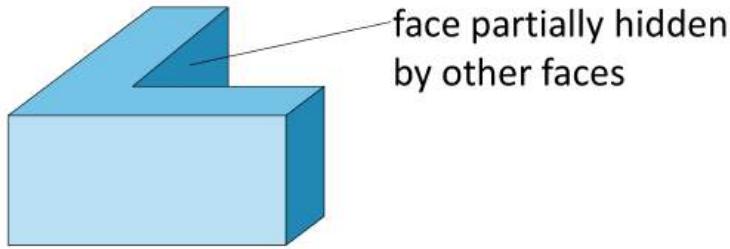
sufficient condition: if  $C \leq 0$  then back-face

↑  
negative !



Der Vektor hat in x und y Koordinaten 0, 0 und nur die Z Koordinate ist relevant.

complete visibility test only for non-overlapping convex polyhedra



... is a preprocessing step for other objects:

about **??** of surfaces are eliminated

## Depth Buffer Verfahren (Z Puffer Verfahren)

[EVC\\_Skriptum\\_CG](#), p.32

### Problemstellung: Sichtbarkeitsproblem

- Ziel: Bestimmung, welche Objekte in einer 3D-Szene für den Betrachter sichtbar sind und welche verdeckt werden.
- Lösung des Z-Puffer-Algorithmus erfolgt pixelweise für eine gegebene Bildauflösung.

### Kernidee des Z-Puffer-Algorithmus

- **Zusätzlicher Speicher:** Neben dem Framebuffer (Farbinformation pro Pixel) wird ein Z-Puffer (oder Tiefenpuffer) benötigt.
- **Z-Wert pro Pixel:** Der Z-Puffer speichert für jedes Pixel die Tiefeninformation (z-Koordinate) des bisher gezeichneten Objekts.
- **Blickrichtung:** Normalerweise in z-Richtung, daher Speicherung des einzelnen z-Wertes ausreichend.

### Speicherbereiche

- **Framebuffer:** Speichert die Farbinformation jedes Pixels.
- **Z-Puffer (Depth Buffer):** Speichert den z-Wert (Tiefe) jedes Pixels.

### Ablauf des Algorithmus

```

for all (x,y)                      // Initialisierung des Hintergrundes
    depthBuff(x,y) = -1             // größtmögliche Entfernung
    frameBuff(x,y) = backgroundColor
for each polygon P                  // Schleife über alle Polygone
    for each position (x,y) on polygon P
        calculate depth z
        if z > depthBuff(x,y) then
    
```

```

depthBuff(x,y) = z
frameBuff(x,y) = surfColor(x,y)
else
    // else nichts !

```

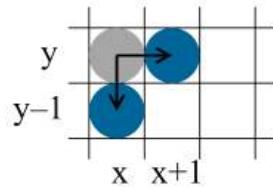
polygons with corresponding z-values

image  
back-ground

depth-buffer


## Effizienz

- Inkrementelle Berechnung:** Für ebene Polygone lassen sich die z-Werte natürlich wieder inkrementell effizienter berechnen.



$$Ax + By + Cz + D = 0$$

depth at  $(x,y)$ :

$$z = \frac{-Ax - By - D}{C} \quad \text{constants !}$$

depth at  $(x+1,y)$ :

$$z' = \frac{-A(x+1) - By - D}{C} = z - \frac{A}{C}$$

depth at  $(x,y-1)$ :

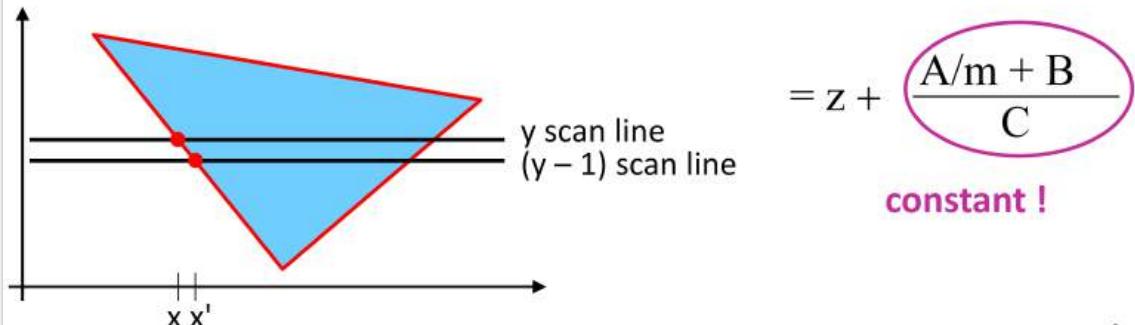
$$z'' = \frac{-Ax - B(y-1) - D}{C} = z + \frac{B}{C}$$

## Vorteile

- Keine Sortierung der Objekte notwendig:** Polygone können in beliebiger Reihenfolge gezeichnet werden.

$$z = \frac{-Ax-By-D}{C}$$

$$y' = y - 1 \quad \Rightarrow \quad z' = \frac{-A(x-1/m)-B(y-1)-D}{C}$$



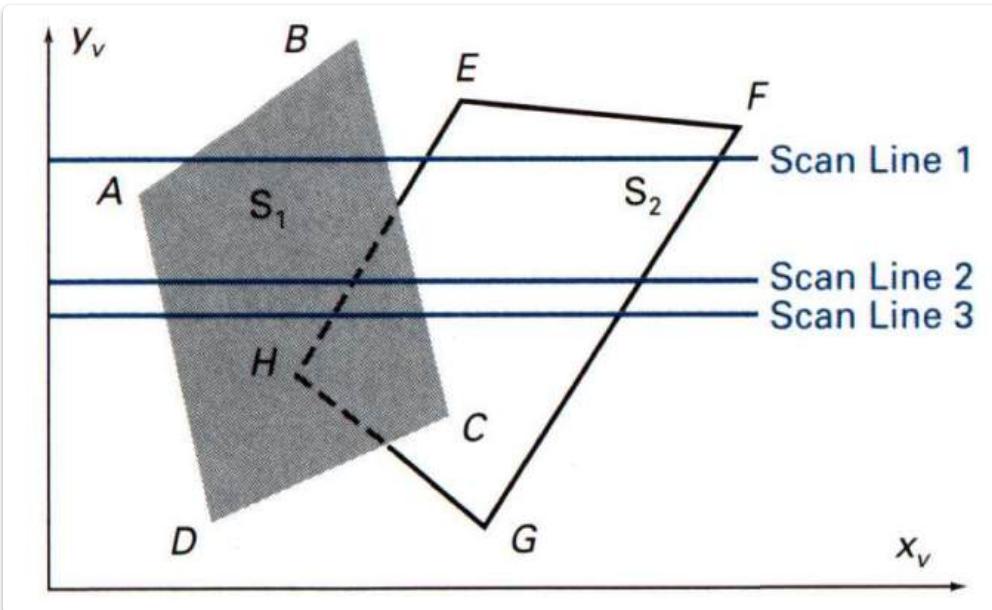
## Anmerkung

- **Viewport-Transformation:** Oft wird  $z$  mit  $-1$  multipliziert bevor das  $z$ -Puffer-Verfahren angewendet wird.

## Scanline-Methode

EVC\_Skriptum\_CG, p.32

- **Ziel:** Korrekte Sichtbarkeitsberechnung pixelzeilenweise.
- **Ablauf:** Zeilenweises Vorgehen (z.B. von oben nach unten,  $y$  fallend).
- **Vorteil:** Nutzt die Kohärenz zwischen aufeinanderfolgenden Scanlines (geringe Änderung des Sichtbarkeitsverhaltens).



## Depth-Sorting-Methode (Painter's Algorithm)

EVC\_Skriptum\_CG, p.33

- **Grundprinzip:**

- Sortiere alle Polygone nach ihrer Tiefenlage (von hinten nach vorne).
- Zeichne die Polygone in dieser sortierten Reihenfolge.
- **Logik:** Spätere (weiter vorne liegende) Polygone übermalen frühere (weiter hinten liegende) und erzeugen so korrekte Sichtbarkeit.
- **Hauptaufwand:** Sortierung der Polygone.
- **Herausforderung:** Sicherstellen, dass kein Polygon ein anderes verdeckt, das in der Sortierreihenfolge später kommt (weiter vorne liegt).
- **Vorgehensweise:**
  1. **Grobsortierung:** Schnelle erste Sortierung der Polygone.
  2. **Überprüfung:** Testen, ob die Sortierung korrekt ist (keine fehlerhaften Verdeckungen).
  3. **Umsortierung (ggf.):** Bei Bedarf Anpassung der Reihenfolge, um korrekte Sichtbarkeit zu gewährleisten.

surfaces sorted in order of decreasing depth (viewing in  $-z$ -direction)

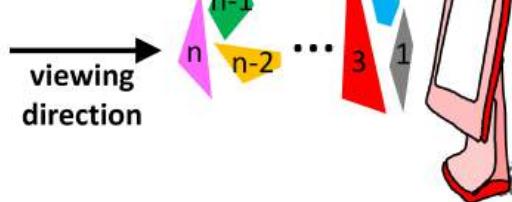
- (1) "approximate"-sorting using smallest  $z$ -value (greatest depth)
- (2) fine-tuning to get correct depth order

surfaces scan converted in order

sorting both in image and object space

scan conversion in image space

also called "painter's algorithm"



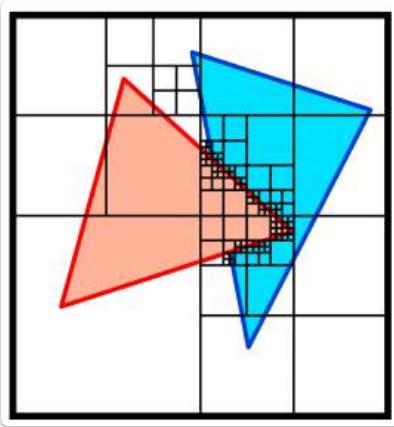
Werner Purgathofer

23

## Area-Subdivision Methode

[EVC\\_Skriptum\\_CG](#), p.33

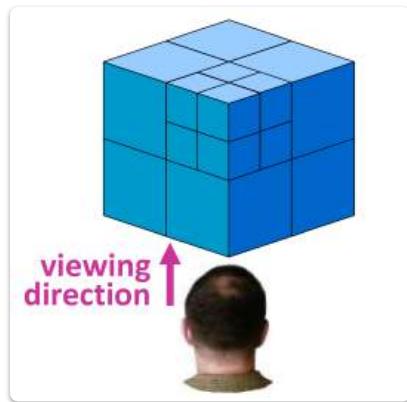
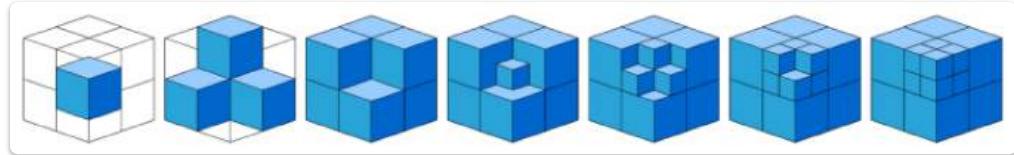
- **Grundidee:** Divide-and-Conquer-Ansatz für das Sichtbarkeitsproblem.
- **Analogie:** Ähnlich der Quadtree-Repräsentation von Bildern.
- **Vorgehensweise:**
  1. **Einfache Fälle:** Sichtbarkeitsproblem wird in grober Auflösung gelöst.
  2. **Komplizierte Fälle:**
    - Unterteilung der aktuellen Bildfläche in vier gleich große Viertel.
    - Rekursive Anwendung der Methode auf jede der vier Teilstufen.
- **Garantie:** Die rekursive Unterteilung bis zur maximalen Bildauflösung stellt eine pixelgenaue Lösung des Sichtbarkeitsproblems sicher.



## Octree-Methode

EVC\_Skriptum\_CG, p.33

- **Datenstruktur:** Repräsentation der Szene als Octree (raumorientierter Baum).
- **Vorteil der Datenstruktur:** Implizites Wissen über die Tiefenordnung (was vorne und hinten ist) für jede Blickrichtung.
- **Rendering-Strategien:**
  - **Von hinten nach vorne:**
    - Rekursives Durchlaufen der Octree-Knoten.
    - Rendern der Teilwürfel in der Reihenfolge: *entferntester* -> 3 nächstnäheren -> nächste 3 -> vorderster.
    - Beispiel für frontale Blickrichtung

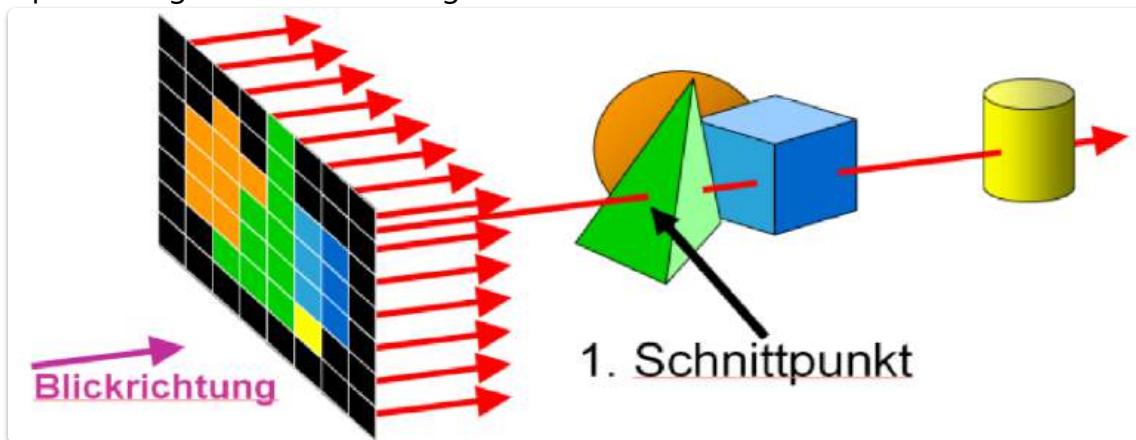


- **Von vorne nach hinten:**
  - Rekursives Durchlaufen der Octree-Knoten.
  - Zeichnen nur der sichtbaren Bereiche.
  - Notwendigkeit, sich bereits gezeichnete Bereiche zu merken, um Overdraw zu vermeiden.
- **Genereller Vorteil:** Das inhärente Wissen über die Tiefenordnung im Vergleich zu anderen Datenstrukturen vereinfacht die Sichtbarkeitsbestimmung.

# Ray-Casting

EVC\_Skriptum\_CG, p.33, 10. Ray-Tracing

- **Grundprinzip:** Berechnung der Sichtbarkeit für jedes Pixel einzeln.
- **Ablauf:**
  1. **Blickstrahl:** Vom Pixel in Blickrichtung wird eine gerade Linie (Blickstrahl) in die Szene projiziert.
  2. **Schnittpunktberechnung:** Der Blickstrahl wird mit allen Objekten/Polygonen in der Szene geschnitten.
  3. **Nächster Schnittpunkt:** Aus der Menge der Schnittpunkte wird derjenige ausgewählt, der am nächsten zum Betrachter liegt.
  4. **Pixelfarbe:** Die Farbe der Oberfläche am nächsten Schnittpunkt bestimmt die Farbe des betrachteten Pixels.
- **Vorteile:**
  - Ermöglicht das Rendern verschiedenster Oberflächen (nicht nur Polygone), sofern der Schnitt mit einer Geraden berechenbar ist (z.B. Freiformflächen).
  - **Benötigte Information (für Schattierung):** Oberflächennormale am Auftreffpunkt des Strahls.
- **Nachteile:**
  - **Hoher Rechenaufwand:** Für jedes Pixel müssen Schnittpunktberechnungen mit potenziell vielen Objekten durchgeführt werden (Millionen von Pixeln und möglicherweise tausende bis Millionen von Objekten).
  - **Notwendigkeit:** Effiziente Implementierung der Schnittpunkttests und weitere Optimierungen zur Reduzierung des Rechenaufwands sind unerlässlich.



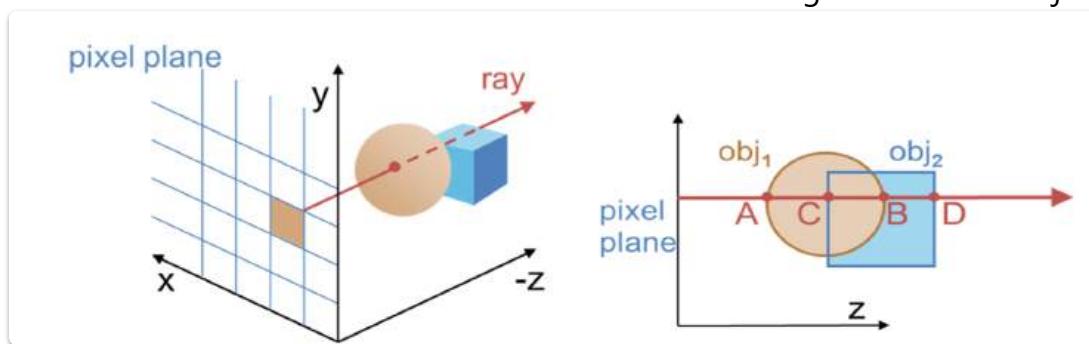
## Ray-Casting von CSG-Objekten

EVC\_Skriptum\_CG, p.33

- **Gebräuchliche Methode:** Pixelweise Berechnung des Bildes durch Ray-Casting.
- **Ablauf pro Pixel:**
  1. **Ray-Erzeugung:** Ein Blickstrahl (Ray) wird in Blickrichtung "ausgeworfen".
  2. **Schnitt mit allen Objekten:** Der Ray wird mit allen Objekten der Szene geschnitten.

3. **Vorderster Schnittpunkt:** Der Schnittpunkt, der am nächsten zum Betrachter liegt, bestimmt das sichtbare Objekt.

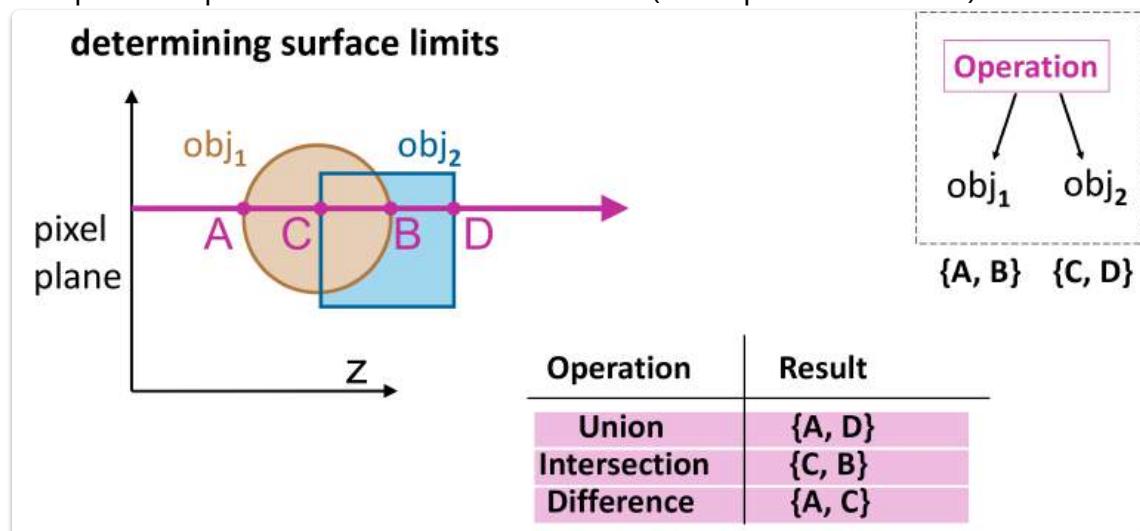
4. **Pixelfarbe:** Das Pixel erhält die Farbe des am vordersten geschnittenen Objekts.



- **Rekursive Berechnung bei CSG-Bäumen:**

- **Endknoten (Primitive Objekte):** Direkte und einfache Berechnung aller Ray-Objekt-Schnittpunkte.
- **Zwischenknoten (Boolesche Operationen):**
  - Die Schnittpunktlisten der beiden Kindknoten werden entsprechend dem Booleschen Operator verknüpft:
    - **Vereinigung (Union):** Kombination der Schnittpunktlisten (Beispiel: A, D).
    - **Durchschnitt (Intersection):** Schnittmenge der Schnittpunktlisten (Beispiel: C, B).
    - **Differenz (Subtraction):** Schnittpunkte des ersten Objekts, die nicht im zweiten Objekt liegen (Beispiel: A, C).
- **Wurzelknoten:** Der erste Punkt (der dem Betrachter am nächsten liegt) der resultierenden verknüpften Schnittpunktliste wird ausgewählt.

- **Relation zu Ray-Tracing:** Ray-Casting ist eine vereinfachte Version von Ray-Tracing, das komplexere optische Effekte simulieren kann (wird später behandelt).



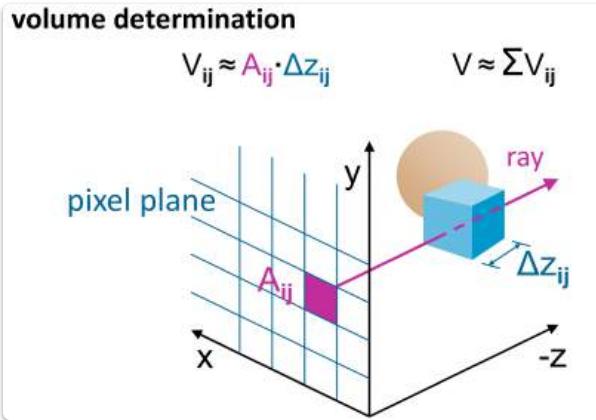
Ray-Casting ist eine vereinfachte Version von Ray-Tracing, mit dem noch viele weitere optische Effekte simuliert werden können. Dazu kommen wir in einem späteren Kapitel.

**Ray-Casting** = für jedes Pixel der Darstellungsfläche:

- erzeuge eine Gerade durch das Pixel in Blickrichtung („Blickstrahl“)
- schneide den Blickstrahl mit allen Objekten
- wähle aus der Schnittpunktliste den zum Betrachter nächsten Punkt
- färbe das Pixel mit der Farbe der Oberfläche dieses Punktes

## Ergänzung von den Slides

Man kann das auch verwenden um das Volumen zu berechnen



## Klassifizierung der Verfahren

[EVC\\_Skriptum\\_CG, p.34](#)

Wir wollen nun noch überlegen, welche Verfahren im Objektraum arbeiten und welche im Bildraum. Dies ist nicht immer ganz eindeutig klassifizierbar, aber im Großen und Ganzen gilt:

### Objektraum-Verfahren:

- Backface Detection
- Depth Sorting
- Octree-Methode

### Bildraum-Verfahren:

- Z-Puffer
- Scanline-Methode
- Area Subdivision
- Ray Casting

## 8. Bildmerkmale - Interest Points

Quellen:

- [EVC\\_Skriptum\\_CV, p.40](#) bis [EVC\\_Skriptum\\_CV, p.45](#)
- 

## Interessenspunkte (Interest Points)

### Definition

- Interessenspunkte sind charakteristische Punkte in einem Bild.
- Sie dienen der Beschreibung und dem Matching (Abgleich) von Bildern.

### Eigenschaften

- Hervorstechend durch **einzigartige und unterscheidbare Merkmale**
- Basieren auf **lokalen Intensitätsunterschieden**, z. B.:
  - Ecken
  - Kanten
  - Strukturen

### Zweck

- Werden zur **Bildbeschreibung** verwendet
  - Besonders geeignet für **Vergleiche und Wiedererkennung** zwischen Bildern
- 

## Harris Corner Detection

### Allgemeines

- Harris Corner Detection ist ein verbreiteter Algorithmus zur **Eckendetektion** in Bildern.
- Ziel: **Identifikation von Interessenspunkten**, die Ecken darstellen.

### Kernidee

- Berechnung der **Harris-Matrix**  $M$ :

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- $I_x, I_y$  sind die **Ableitungen des Bildes** entlang der x- bzw. y-Achse.

## Harris Response-Funktion

- Dient zur Bewertung der Matrix  $M$  an jedem Pixel:

$$R = \det(M) - k \cdot (\text{tr}(M))^2$$

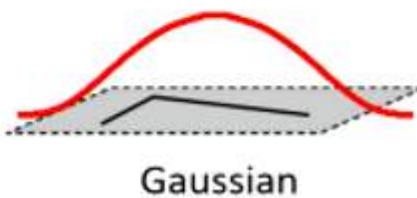
- $\det(M)$ : Determinante von  $M$
- $\text{tr}(M)$ : Spur von  $M$  (Summe der Diagonaleinträge)
- $k$ : Empirischer Parameter (typisch  $0,04 \leq k \leq 0,06$ )

## Eckendetektion

- Für **jeden Pixel** im Bild wird  $R$  berechnet.
- Ein Pixel wird als **potenzielle Ecke** klassifiziert, wenn:
  - $R$  ist größer als ein **vorgegebener Schwellenwert**

$$w(x, y) = \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right)$$

Window function  $w(x, y) =$



## SIFT (Scale-Invariant Feature Transform)

### Allgemeines

- SIFT ist ein Verfahren zur **Detektion und Beschreibung lokaler Merkmale** in Bildern.
- Besonders **robust gegenüber**:
  - Skalierung
  - Rotation
  - Helligkeitsänderungen

### Ablauf des Algorithmus

- Besteht aus mehreren Schritten wie der Skalenraumextrema-Detektion, Keypoint lokalisierung und Deskriptionsextraktion

# SURF (Speeded-Up Robust Features)

## Allgemeines

- SURF ist eine Weiterentwicklung von SIFT
- Ziel: Schnellere und effizientere Berechnung lokaler Merkmale

## Eigenschaften

- Robust gegenüber:
  - Skalierung
  - Rotation
  - Helligkeitsänderungen

## Technische Unterschiede zu SIFT

- Haar Wavelet-Responses werden zur Merkmalserkennung verwendet:
    - Vereinfachen und beschleunigen die Berechnungen
    - Geringerer Rechenaufwand bei vergleichbarer Robustheit
  - SURF ist im Vergleich zu SIFT schneller, insbesondere bei großen Bildmengen oder Echtzeitanwendungen
- 

# Bildmerkmale

## Definition

- Bildmerkmale sind mathematische Beschreibungen eines Bildes oder seiner Teile.
- Ziel: Repräsentation von Informationen, die besser unterscheidbar und nützlicher als reine Pixelwerte sind.
- Unterstützen die Extraktion relevanter Informationen für verschiedene Anwendungen.

## Arten von Bildmerkmalen

- Globale Merkmale:
  - Beschreiben das gesamte Bild
  - Beispiel: Grauerthistogramm
  - Nachteile:
    - Schlecht unterscheidbar
    - Nicht robust genug für komplexe Anwendungen
- Lokale Merkmale:
  - Beschreiben kleine Bildregionen

- Ziel: Mathematische Beschreibung von **interessanten Bildbereichen** und ihrer lokalen Nachbarschaft
- Robust gegenüber Skalierung, Rotation, etc.
- Anwendungen:
  - **Bildregistrierung**
  - **Panorama-Stitching**
  - **3D-Modellierung**
  - **Objekterkennung**

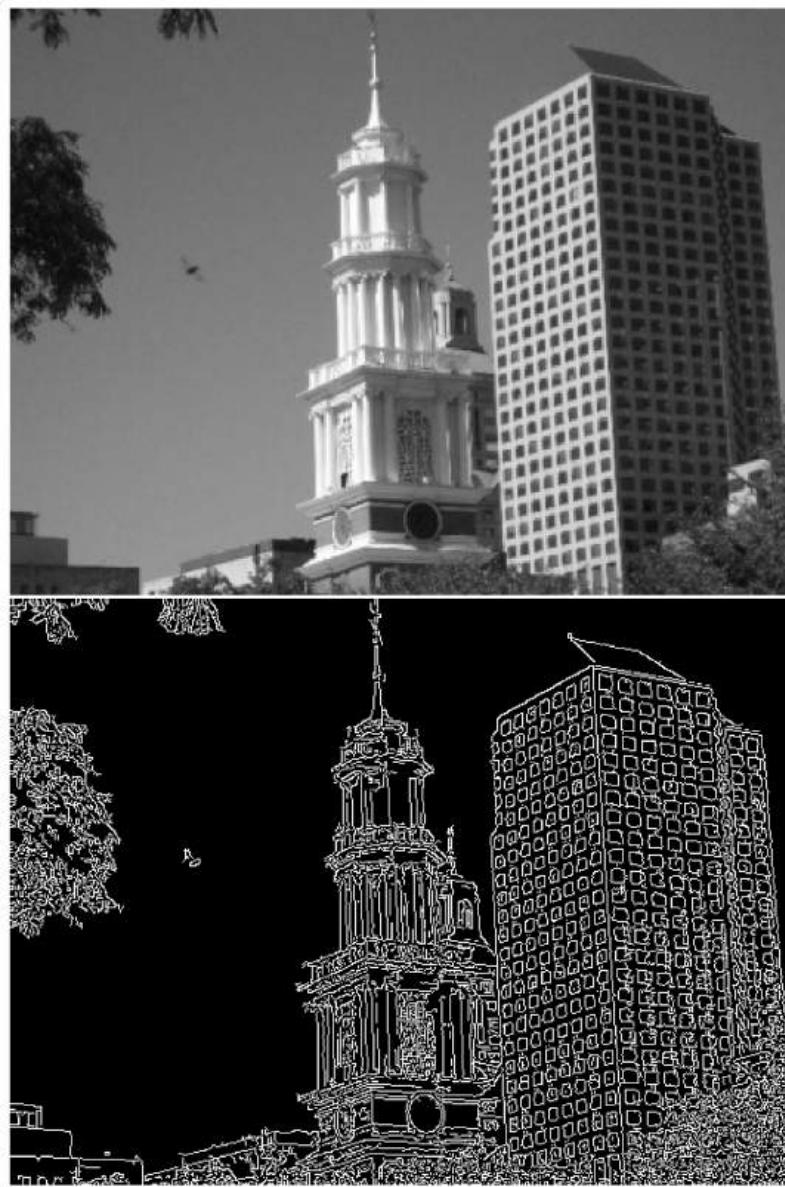
## Schritte der Merkmalsextraktion

### 1. Merkmalsdetektion

- Identifikation von **Bildbereichen mit hohem Informationsgehalt**
- Bestimmung der **räumlichen Lage und Skalierung**
- Ergebnis: **Interest Points / Keypoints**

### 2. Merkmalsbeschreibung

- Erstellung eines **Merkmalsvektors**, der die **lokale Bildstruktur** beschreibt
- Die Gesamtheit aller Vektoren definiert den **Merkmalsraum** (engl. *feature space*)



## Kantenerkennung

mehr siehe [6. Kantenfilterung](#)

### Bedeutung von Kanten

- Kanten enthalten **nicht redundante** Informationen eines Bildes.
- Sie stellen **Unstetigkeiten** in der Helligkeit dar – also **starke Helligkeitsveränderungen**.

### Ziel der Kantendetektion

- Finden von **Punkten mit plötzlichen Intensitätsänderungen** im Bild.
- Ergebnis: **Menge von Pixeln**, an denen sich **Unstetigkeitsstellen** befinden.

### Vorteile

- **Reduktion der Datenmenge**:

- Unwichtige, redundante Informationen werden **herausgefiltert**
- Nur die **strukturell relevanten Bildinformationen** (hohe Frequenzen) bleiben erhalten

## Einschränkungen

- **Nicht rotationsinvariant**
  - **Nicht skalierungsinvariant**
  - Dadurch sind gleiche Merkmale in unterschiedlichen Bildern **schwer vergleichbar**
- 

# Interest Points

## Eigenschaften von Eckpunkten

- **Auffällige Bildstellen** – sowohl für Menschen als auch für Algorithmen



- **Robuste Merkmale:**
  - Entstehen **nicht zufällig** in 3D-Szenen
  - **Zuverlässig lokalisierbar** bei:
    - unterschiedlichen Blickwinkeln
    - verschiedenen Beleuchtungsbedingungen

## Anforderungen an einen guten Corner Detector

- **Zuverlässige Erkennung** trotz Bildrauschen
- **Hohe Lokalisierungsgenauigkeit**
- **Effiziente Berechnung**

## Grundlagen der Eckpunkt detektion

- **Kante:** Bereich mit starkem Helligkeitsgradient in **einer Richtung**, orthogonal dazu gering
- **Ecke:** Punkt mit **starkem Helligkeitsunterschied in mehreren Richtungen gleichzeitig**

## Allgemeine Eigenschaften eines Interest Points

1. **Mathematisch eindeutig definierbar**
  2. **Eindeutige Position** im Bildraum
  3. **Hoher lokaler Informationsgehalt**
  4. **Stabilität** gegenüber:
    - Lokalen Störungen
    - Globalen Störungen (z. B. perspektivische Verzerrung, Beleuchtungsvariation)
  5. **Skalierungsinviananz**
- 

## Eckendetektion

### Detektionsprinzip

- Eckendetektoren reagieren nicht nur auf **Ecken**, sondern auch auf Bildregionen mit **hoher Variabilität in alle Richtungen**.
- Ein **Eckpunkt liegt vor**, wenn der **Gradient in mehreren Richtungen** stark ausgeprägt ist.

### Abgrenzung zu Kanten

- **Kanten:** Gradient stark **in einer Richtung** → **kein Eckpunkt**
- **Eckpunkt:** Starke Gradienten in **mehreren Richtungen gleichzeitig**

### Definition eines Eckpunkts

- **Schnittpunkt zweier Kanten**
- **Punkt mit zwei dominanten Kantenrichtungen** in seiner lokalen Umgebung
- Kann auch sein:
  - **Endpunkt einer Linie**
  - **Punkt auf einer Kurve mit maximaler Krümmung**

### Anforderungen an die Detektion

- **Isotrope Erkennung:** Unabhängig von Orientierung
- **Robustheit** gegenüber:
  - **Unterschiedlichen Lichtverhältnissen**
  - **Geometrischen Transformationen** wie:

- Translation
- Rotation

## Qualitätskriterium für Eckendetektoren

- **Wiedererkennung derselben Ecke** unter verschiedenen Bedingungen (Beleuchtung, Transformation)

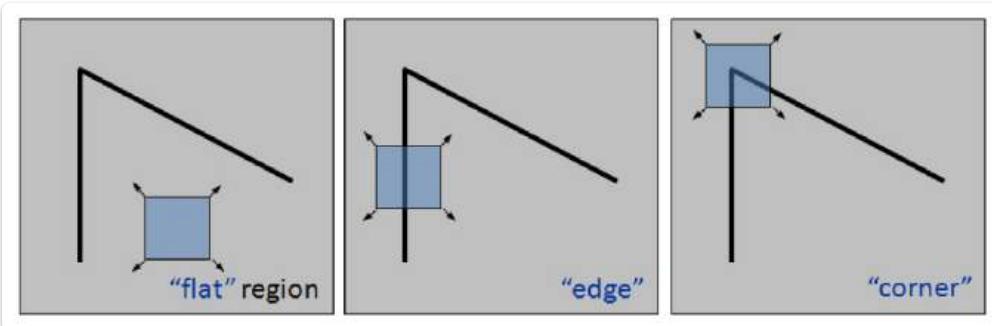
## Moravec-Eckendetektor

### Ursprung

- Einer der **ersten Eckendetektoren**
- Entwickelt von **Hans P. Moravec**
- Einführung des Begriffs "**Points of Interest**"

### Grundidee

- Eine **Ecke wird als Punkt mit geringer Ähnlichkeit** zum verschobenen Fensterinhalt definiert.
- Verwendet ein **lokales Fenster**  $w(x, y)$  zur Analyse der Intensitätsveränderungen.
- Fenster wird in **vier Richtungen** leicht verschoben:
  - Für jede Verschiebung wird die **Intensitätsveränderung** berechnet:
    - $I(x + u, y + v)$ : Intensität an verschobener Stelle
    - $I(x, y)$ : ursprüngliche Intensität



$$E(u, v) = \sum_{x, y} w(x, y) [I(x+u, y+v) - I(x, y)]^2$$

Window function      Shifted intensity      Intensity

Window function  $w(x, y) =$

## Interest Value

- Für **jede Pixelposition** wird ein **Interest Value** berechnet:
  - Entspricht der **minimalen Intensitätsänderung** bei allen Verschiebungen

## Interpretation der Interest Values

- **Geringe Veränderungen in alle Richtungen** → **flache Region**
- **Hohe Veränderung nur in einer Richtung** → **Kante**
- **Hohe Veränderung in mehreren Richtungen** → **Ecke**

## Nachteile des Moravec-Operators

- **Anisotropische Antwort:**
  - Kanten, die **nicht exakt** in Verschieberichtungen verlaufen, können fälschlich als Ecken erkannt werden
- **Empfindlich gegenüber Bildrauschen:**
  - Besonders entlang von Kanten
  - Betrachtet nur **Minimale Intensitätsänderungen**, nicht deren Verteilung

## Harris Eckendetektor

---

### Verbesserung gegenüber Moravec

- Statt **absoluter Pixeldifferenzen** (wie bei Moravec) wird die **Variation der lokalen Bildstruktur** analysiert.
- Verwendet die **Bildgradienten** zur Erkennung von Ecken:
  - An einem **Eckpunkt** sind die Gradienten sowohl in der **Hauptrichtung** als auch **orthogonal dazu** signifikant

### Vorteile

- Zuverlässigere Detektion und Wiedererkennung von Ecken
- **Isotrope Antwort:**
  - **Rotationsinvariant**
  - Kein bevorzugter Richtungsverlauf wie bei Moravec

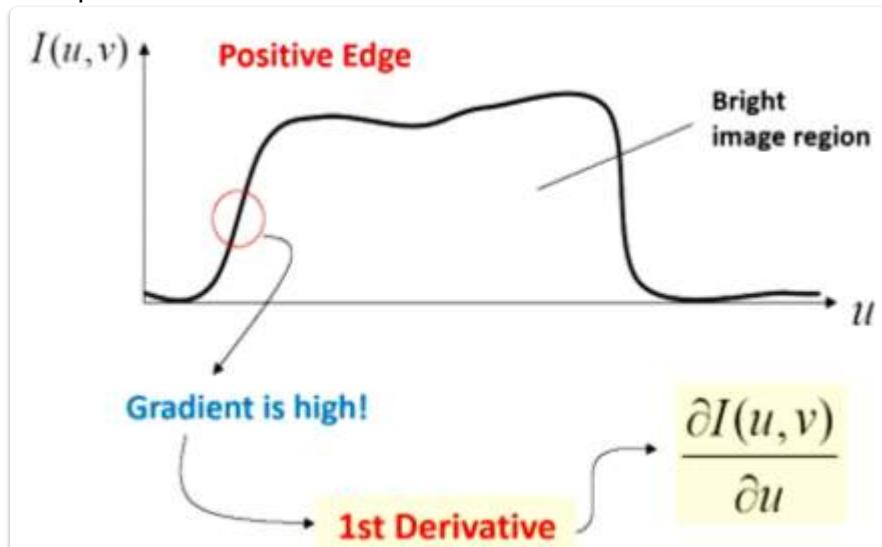
### Nachteile

- **Höherer Rechenaufwand** durch komplexere Berechnungen
  - **Nicht skalierungsinvariant** (wie auch der Moravec-Detektor)
- 

## Skaleninvariante Merkmalsumwandlung (SIFT)

### Allgemeines

- **SIFT (Scale-Invariant Feature Transform)**: Algorithmus zur **Dektection und Beschreibung** lokaler Merkmale in Bildern.
  - Entwickelt von **David Lowe** im Jahr **1999** und in den **USA patentiert**.
  - **Eigenschaften**:
    - **Skalierungs invariant**: Unabhängig von der Bildgröße
    - **Rotationsinvariant**: Unabhängig von der Orientierung des Bildes
    - **Robust gegenüber**:
      - Affinen Transformationen
      - Beleuchtungsveränderungen
- Was passiert an einer Ecke?

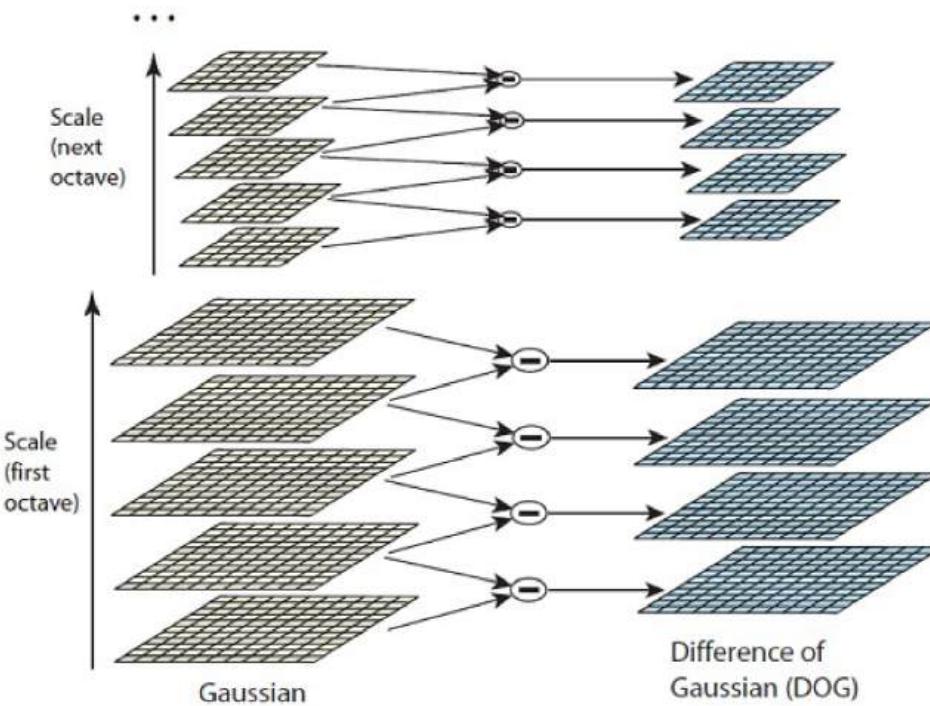


- Die Umwandlung erfolgt in **4 Schritten**:

## 1. Finden von Interest Points – Skalierung

### Difference of Gaussians (DoG)

- SIFT verwendet eine **Difference-of-Gaussians (DoG)**-approximierte **Laplacepyramide**.
  - Der Hauptunterschied zur **ursprünglichen Laplacepyramide**:
    - Es erfolgen **keine Größenänderungen** zwischen den Ebenen, die Auflösung bleibt konstant.



- Functions for determining scale

$$f = \text{Kernel} * \text{Image}$$

Kernels:

$$L = \sigma^2 (G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma))$$

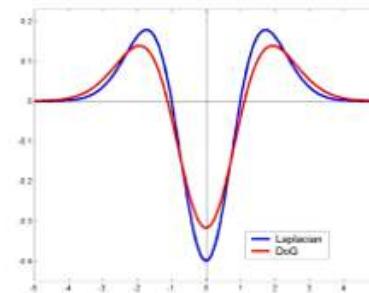
(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)

where Gaussian:

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



## DoG Filtering



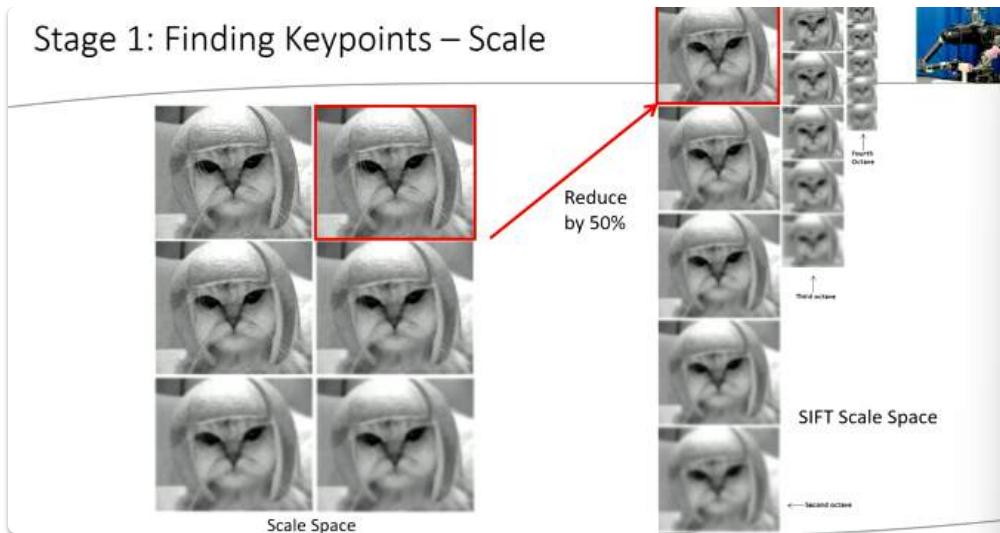
- Convolution with a variable-scale Gaussian

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

$$G(x, y, \sigma) = 1/(2\pi\sigma^2) \exp^{-(x^2+y^2)/\sigma^2}$$

- Difference-of-Gaussian (DoG) filter  $G(x, y, k\sigma) - G(x, y, \sigma)$

- Convolution with the DoG filter  $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$



## Oktaven und Anwendung der Gaußfilter

- **Oktave:** Eine Reihe von **5 aufeinanderfolgenden Gaußfiltern**, die insgesamt **4 bandpassgefilterte Ergebnisse** erzeugen.
- Nach der ersten Oktave wird die **nächste Ebene der Gaußpyramide** (2. Oktave) auf die gleiche Weise behandelt.
  - Jede Oktave reduziert die **Auflösung** des Bildes:
    - Erste Oktave: **Originalauflösung**
    - Zweite Oktave: **halbe Auflösung** in beiden Richtungen, und so weiter.

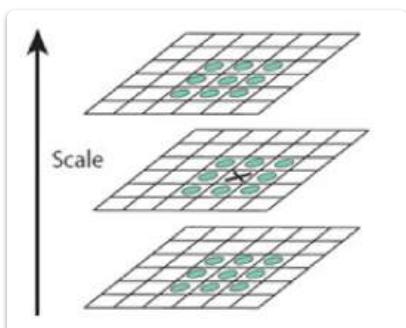
## DoG und Lokalisierung von Merkmalen

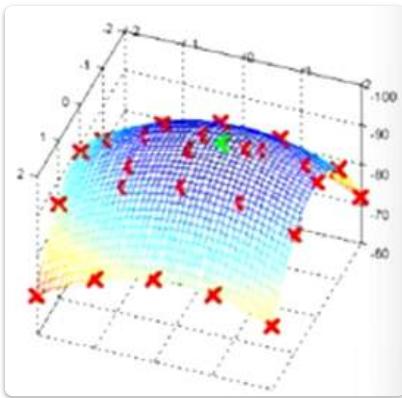
- **DoG Bilder** (oder auch **Laplacebilder**):
  - Trennen die **Frequenzen** im Bild
  - Dies hilft bei der **Lokalisation von Kanten** und **Ecken**

## Rolle der Oktaven

- **Extrema** werden durch das **Vergleichen von Bildpunkten** zwischen verschiedenen Oktaven lokalisiert.
- Die **verschiedenen Oktaven** sind entscheidend für die Erkennung von **Interest Points** auf unterschiedlichen Skalen.

## 2. Finden von Interest Points - Position





## Dreiteiliger Prozess zur Positionierung von Interest Points

### a) Lokalisierung der Extrema (Maxima/Minima) in den DoG-Bildern

- **Erster Schritt:** Grobe Lokalisierung der **Maxima und Minima**.
- **Vorgehensweise:**
  - Iterative Überprüfung der **Nachbarn** im DoG-Skalenraum.
  - Es werden **26 Abfragen** für jedes Pixel durchgeführt:
    - 9 Punkte in der gleichen Ebene
    - 9 Punkte in der Ebene darüber
    - 8 Punkte in der Ebene darunter
- **Kriterien:** Ein Interest Point ist **größer** oder **kleiner** als alle 26 benachbarten Punkte.

### b) Bestimmung der Position der Extrema mit Subpixel-Genauigkeit

- **Ziel:** Die genauen Positionen der **Maxima und Minima** zu bestimmen, da sie nicht unbedingt auf einem Pixel liegen, sondern auch zwischen den Pixeln.
- **Vorgehensweise:**
  - Nutzung der **Taylorreihenentwicklung**, um Subpixelwerte um den approximierten Interest Point zu berechnen.

### c) Eliminierung ungeeigneter Interest Points

- **Gründe für Eliminierung:**
  - Punkte liegen entlang von **Kanten**.
  - Punkte besitzen **nicht genügend Kontrast**.
- **Schritte zur Eliminierung:**
  1. **Kanten eliminieren:**
    - **Ähnlich wie der Harris-Detektor:**
      - Berechnung der **zwei Gradienten**, die senkrecht zueinander stehen.
      - **Möglichkeiten:**
        - **Flache Region:** Beide Gradienten sind klein.

- **Kante:** Ein Gradient ist groß (senkrecht zur Kante), der andere klein (entlang der Kante).
- **Ecke:** Beide Gradienten sind groß (wird als guter Interest Point betrachtet).

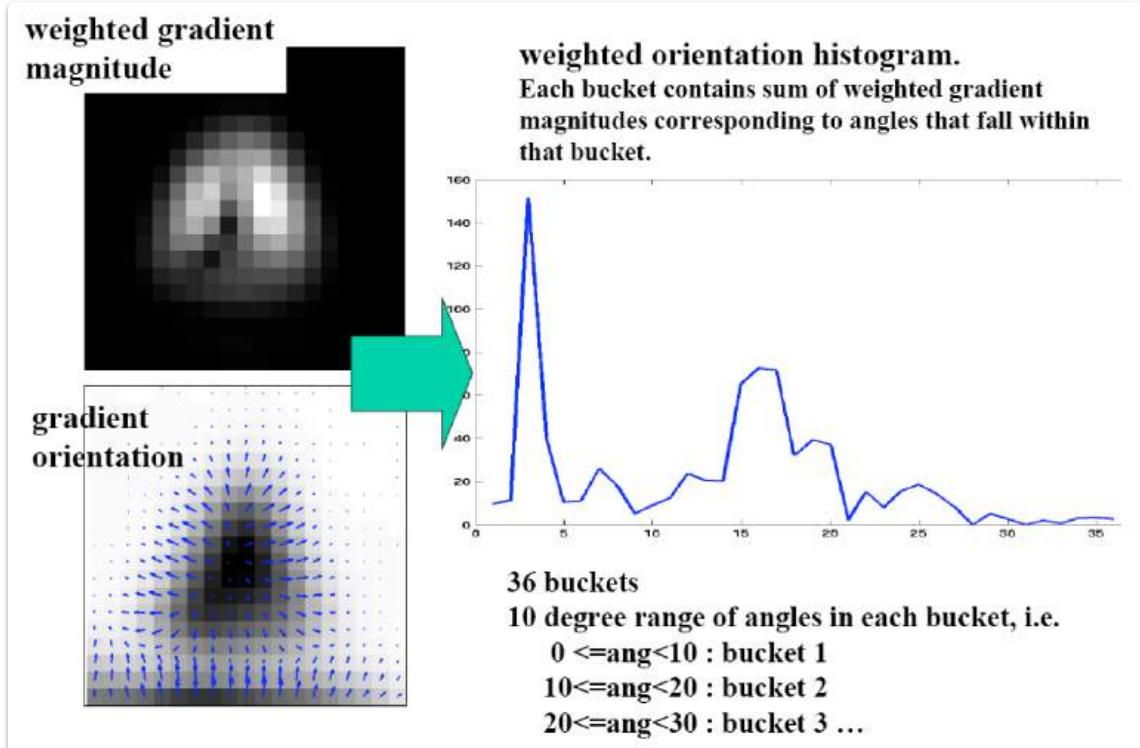
## 2. Geringer Kontrast:

- Kontrolle der **Intensitätswerte** im DoG-Bild.
- Wenn der Wert unter einem bestimmten **Schwellenwert** liegt, wird der Interest Point verworfen.

## 3. Finden von Interest Points – Orientierung

### Ziel

- **Rotationsinvarianz:** Der SIFT-Merkalsvektor für denselben Interest Point soll auch in einem rotierten Bild denselben Wert haben.



### Vorgehensweise

#### 1. Erhebung der Gradienten:

- Um den **rotationsinvarianten Merkmalsvektor** zu erstellen, werden die **Längen (Gradientenbeträge)** und **Richtungen der Gradienten** rund um den Interest Point gesammelt.

#### 2. Bestimmung der dominanten Gradientenrichtung:

- Die **dominante Gradientenrichtung** einer Region wird durch Bestimmung der Richtung des stärksten Gradienten in der Umgebung des Interest Points ermittelt.

#### 3. Histogramm der Gradientenrichtungen:

- Ein Histogramm wird erstellt, das die **360 Grad** der Gradientenrichtungen in **36 Bins** aufteilt (jeweils **10 Grad** pro Bin).
  - Beispiel: Eine Gradientenrichtung von **18,7 Grad** fällt in den Bin von **10-19 Grad**.
  - Der Wert des jeweiligen Bins entspricht der **Länge des Gradienten** an diesem Punkt.

#### 4. Dominante Richtung:

- Der **Bin** mit dem höchsten Wert im Histogramm wird als die **dominante Gradientenrichtung** des Interest Points festgelegt.

## Ergebnis

- Der Interest Point erhält eine **dominante Orientierung**, wodurch der SIFT-Merkalsvektor **rotationsinvariant** wird.

## 4. Erstellen einer Beschreibung der Merkmale

---

- Erstellung einer **eindeutigen Signatur** (Merkmalsvektor) für jeden Interest Point, die **robust gegenüber Störungen** und **rotations- sowie beleuchtungsinvariant** ist.

### Schritte zur Signaturerstellung

#### 1. 16x16 Fenster um den Interest Point:

- Ein **16x16 Fenster** wird um den Interest Point in seiner jeweiligen **Skalierung** definiert.
- Dieses Fenster wird in **16 4x4 Subfenster** unterteilt (siehe Abbildung 37).

#### 2. Berechnung der Gradienten in den Subfenstern:

- Innerhalb jedes **4x4 Subfensters** werden die **Richtungen** und **Längen der Gradienten** berechnet.
- Die Gradientenrichtungen werden in einem **Histogramm mit 8 Bins** zusammengefasst:
  - 0–44 Grad → 1. Bin
  - 45–89 Grad → 2. Bin
  - usw.

#### 3. Gewichtung der Gradienten:

- Der Wert, der einem Bin hinzugefügt wird, hängt von:
  - Der **Länge des Gradienten** (größere Gradienten haben mehr Einfluss).
  - Der **Entfernung zum Interest Point**:
    - Entferntere Gradienten werden mit einer **Gaußschen Gewichtungsfunktion** weniger stark gewichtet (siehe blauer Kreis in der Abbildung).

#### 4. Berechnung des Merkmalsvektors:

- Wiederhole den Prozess für alle **16 4x4 Subfenster**.
- Der resultierende Vektor hat  **$4 \times 4 \times 8 = 128$  Werte**.
- Diese 128 Werte bilden den **Merkmalsvektor** für den Interest Point.

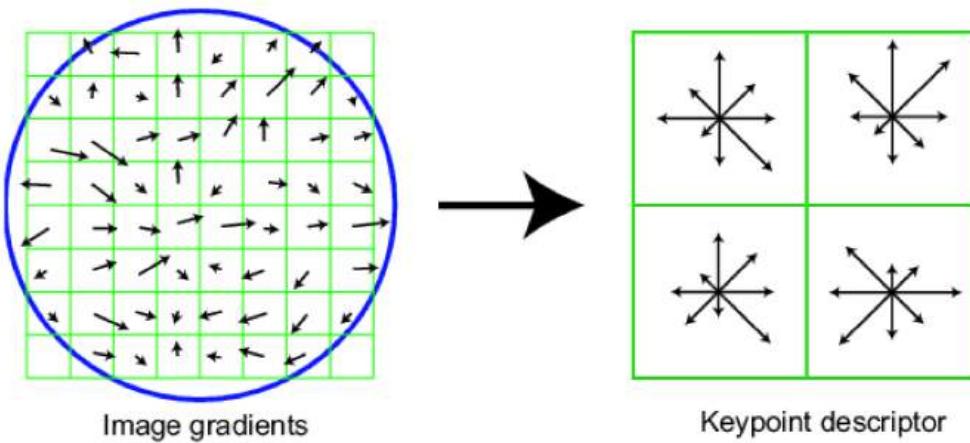
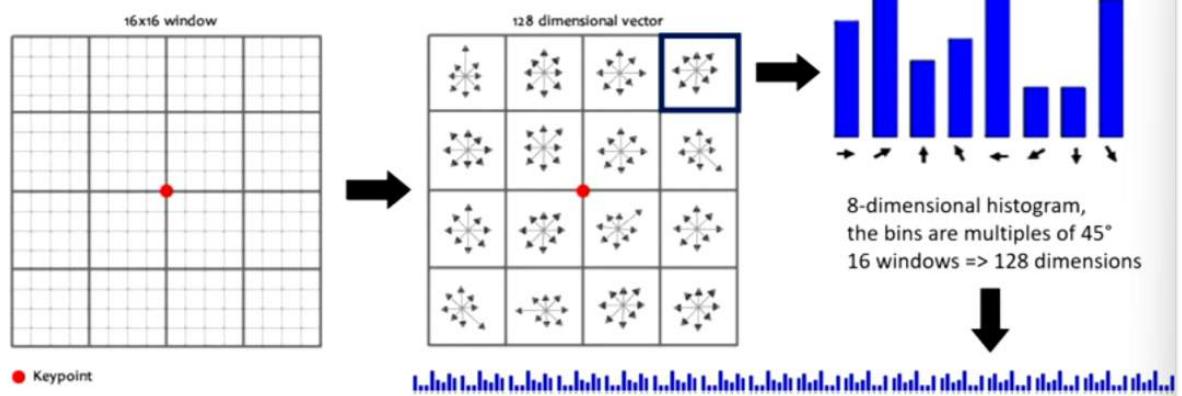


Abbildung 37: Berechnung der SIFT-Deskriptoren

- 8 orientations x 4x4 histogram array = **128 dimensions**



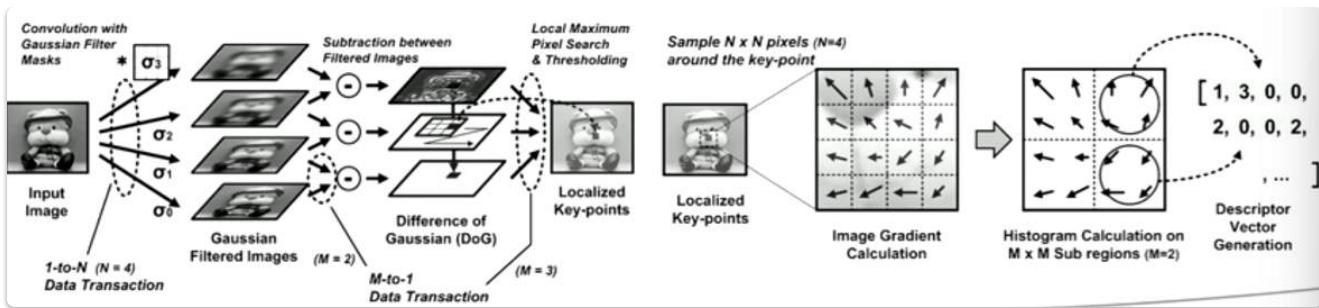
## Einschränkungen und Anpassungen

### 1. Rotationsabhängigkeit:

- Da der Merkmalsvektor auf **Gradientenrichtungen** basiert, verändern sich diese bei einer **Bildrotation**.
- Um **Rotationsinvarianz** zu erreichen, wird die dem Interest Point zugewiesene **dominante Orientierung** von jeder Gradientenrichtung subtrahiert. Somit wird jede Gradientenrichtung relativ zur Orientierung des Interest Points angegeben.

### 2. Beleuchtungsabhängigkeit:

- Lichteffekte können die **Länge einzelner Gradienten** beeinflussen.
- Um die Sensitivität gegenüber Beleuchtung zu verringern:
  - Alle Werte im Merkmalsvektor, die größer als **0.2** sind, werden auf **0.2** gesetzt.
- Der Merkmalsvektor wird anschließend normalisiert, wodurch er weniger anfällig für Beleuchtungsvariationen wird.

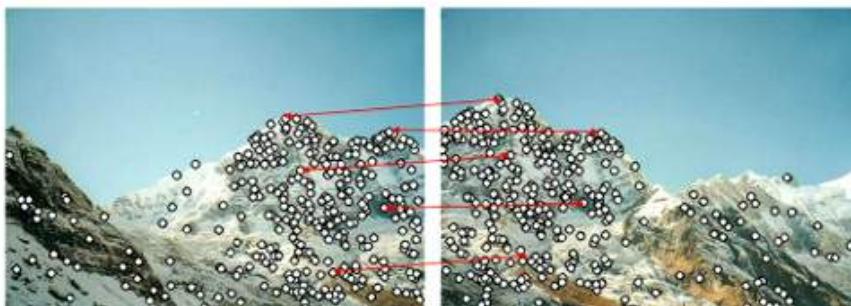


## Möglicher Anwendungsfall: Panoramaerstellung

- We need to match (align) images
- Global methods are sensitive to occlusion, lighting, and parallax effects. So, look for local features that match well.
- How would you do it by eye?



- Detect feature points in both images
- Find corresponding pairs



- Use these pairs to align images



# Testähnliches Beispiel

Beispiel: Moravec-Eckendetektor



- Gegeben ist ein 5x5 großer Bildausschnitt, auf den der Moravec-Eckendetektor angewendet werden soll. Berechnen Sie die Veränderungen der Intensitäten  $E$  für die mit Stern \* markierte Stelle (3,3) und die 4 Verschiebungen (1,0), (1,1), (0,1) und (-1,1). Verwenden Sie dazu eine Fenstergröße von 3x3 und die Summe der quadrierten Differenzen (SSD, Sum of Squared Differences). Bestimmen Sie des weiteren aus den Veränderungen der Intensitäten den „Interest Value“.

v	1	70	60	70	60	60	$E(1,0)$ = _____
1	80	80	90	80	80	80	$E(1,1)$ = _____
2	80	90	100*	100	100	100	$E(0,1)$ = _____
3	80	90	100	100	100	100	$E(-1,1)$ = _____
4	70	80	100	100	100	110	Interest value: _____
x	1	2	3	4	5		

$$E(u, v) = \sum_{x,y} w(x, y) [I(x+u, y+v) - I(x, y)]^2$$

$E(1,0)$ : SSD der Pixelwerte von  $\square$  und  $\square$ :

23 Robert Sablatnig, Computer Vision Lab, EVC-CV8: Image Features - Interest Points

CVL TU Informatics

Beispiel: Moravec-Eckendetektor



- Gegeben ist ein 5x5 großer Bildausschnitt, auf den der Moravec-Eckendetektor angewendet werden soll. Berechnen Sie die Veränderungen der Intensitäten  $E$  für die mit Stern \* markierte Stelle (3,3) und die 4 Verschiebungen (1,0), (1,1), (0,1) und (-1,1). Verwenden Sie dazu eine Fenstergröße von 3x3 und die Summe der quadrierten Differenzen (SSD, Sum of Squared Differences). Bestimmen Sie des weiteren aus den Veränderungen der Intensitäten den „Interest Value“.

v	1	70	60	70	60	60	$E(1,0)$ = _____
1	80	80	90	80	80	80	$E(1,1)$ = _____
2	80	90	100*	100	100	100	$E(0,1)$ = _____
3	80	90	100	100	100	100	$E(-1,1)$ = _____
4	70	80	100	100	100	110	Interest value: _____
x	1	2	3	4	5		

$$E(u, v) = \sum_{x,y} w(x, y) [I(x+u, y+v) - I(x, y)]^2$$

$E(1,0)$ : SSD der Pixelwerte von  $\square$  und  $\square$ :

$$(90-80)^2 + (80-90)^2 + (80-80)^2 + \\ (100-90)^2 + (100-100)^2 + (100-100)^2 + \\ (100-90)^2 + (100-100)^2 + (100-100)^2 = 400$$

23 Robert Sablatnig, Computer Vision Lab, EVC-CV8: Image Features - Interest Points

CVL TU Informatics

## Beispiel: Moravec-Eckendetektor



- Gegeben ist ein 5x5 großer Bildausschnitt, auf den der Moravec-Eckendetektor angewendet werden soll. Berechnen Sie die Veränderungen der Intensitäten  $E$  für die mit Stern \* markierte Stelle (3,3) und die 4 Verschiebungen (1,0), (1,1), (0,1) und (-1,1). Verwenden Sie dazu eine Fenstergröße von 3x3 und die Summe der quadrierten Differenzen (SSD, Sum of Squared Differences). Bestimmen Sie des weiteren aus den Veränderungen der Intensitäten den „Interest Value“.

$y$	1	70	60	70	60	60
1	70	60	70	60	60	
2	80	80	90	80	80	
3	80	90	100*	100	100	
4	80	90	100	100	100	
5	70	80	100	100	110	
$x$	1	2	3	4	5	

$$E(u, v) = \sum_{x,y} w(x, y) [I(x+u, y+v) - I(x, y)]^2$$

$E(1,0)$ : SSD der Pixelwerte von  und :

$$E(1,0) = \underline{\hspace{2cm}}$$

$$E(1,1) = \underline{\hspace{2cm}}$$

$$E(0,1) = \underline{\hspace{2cm}}$$

$$E(-1,1) = \underline{\hspace{2cm}}$$

$$\text{Interest value: } \underline{\hspace{2cm}}$$

## Beispiel: Moravec-Eckendetektor



- Gegeben ist ein 5x5 großer Bildausschnitt, auf den der Moravec-Eckendetektor angewendet werden soll. Berechnen Sie die Veränderungen der Intensitäten  $E$  für die mit Stern \* markierte Stelle (3,3) und die 4 Verschiebungen (1,0), (1,1), (0,1) und (-1,1). Verwenden Sie dazu eine Fenstergröße von 3x3 und die Summe der quadrierten Differenzen (SSD, Sum of Squared Differences). Bestimmen Sie des weiteren aus den Veränderungen der Intensitäten den „Interest Value“.

$y$	1	70	60	70	60	60
1	70	60	70	60	60	
2	80	80	90	80	80	
3	80	90	100*	100	100	
4	80	90	100	100	100	
5	70	80	100	100	110	
$x$	1	2	3	4	5	

$$E(u, v) = \sum_{x,y} w(x, y) [I(x+u, y+v) - I(x, y)]^2$$

**Interest Value:** Minimum der 4 Werte

$$E(1,0) = \underline{\hspace{2cm}}$$

$$E(1,1) = \underline{\hspace{2cm}}$$

$$E(0,1) = \underline{\hspace{2cm}}$$

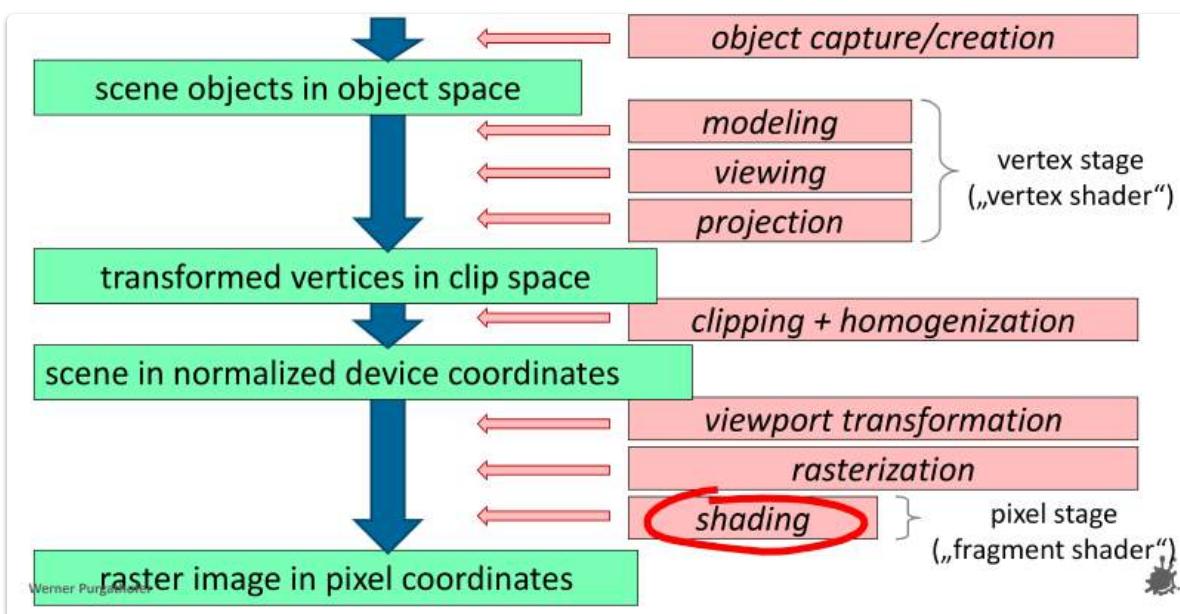
$$E(-1,1) = \underline{\hspace{2cm}}$$

$$\text{Interest value: } \underline{\hspace{2cm}}$$

# 9. Beleuchtung und Schattierung

EVC\_Skriptum\_CG, p.35

- **Definition:** Ein Beleuchtungsmodell (oder Schattierungsmodell, Illumination/Lighting/Shading Model) berechnet die wahrgenommene Farbe/Helligkeit eines Objekts für den Betrachter basierend auf:
  - Lichtverhältnissen in der Szene.
  - Oberflächeneigenschaften des Objekts.
- **Ziel:** Bestimmung der Farbe, die das entsprechende Pixel im Bild erhalten soll.
- **Bedeutung:** Zusammen mit der perspektivischen Projektion der wichtigste Faktor für realistisch aussehende Computergraphik-Bilder.
- **Vereinfachung:** Die folgenden Betrachtungen und Formeln konzentrieren sich zunächst auf die **Helligkeit** der Beleuchtung.
- **Farbbehandlung:** Um Farben zu berücksichtigen, müssen die Berechnungen für verschiedene Wellenlängen durchgeführt werden (im einfachsten Fall für Rot, Grün und Blau).



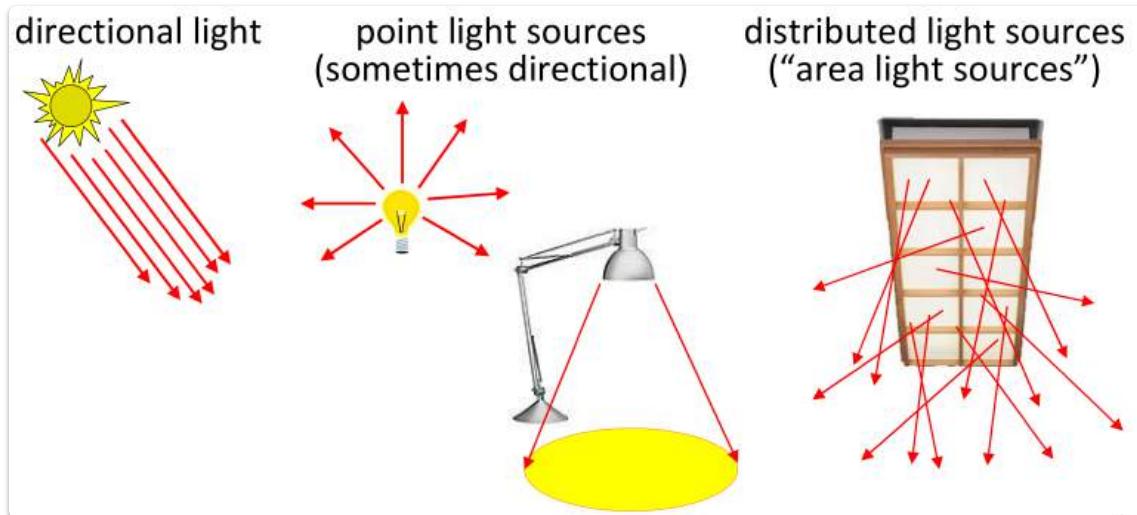
## Lichtquellen und Oberflächen

### Lichtquellen

EVC\_Skriptum\_CG, p.35

- **Notwendigkeit:** Voraussetzung zur Berechnung von Beleuchtungseffekten in einer Szene.
- **Merkmale von Lichtquellen:**
  - **Form:**

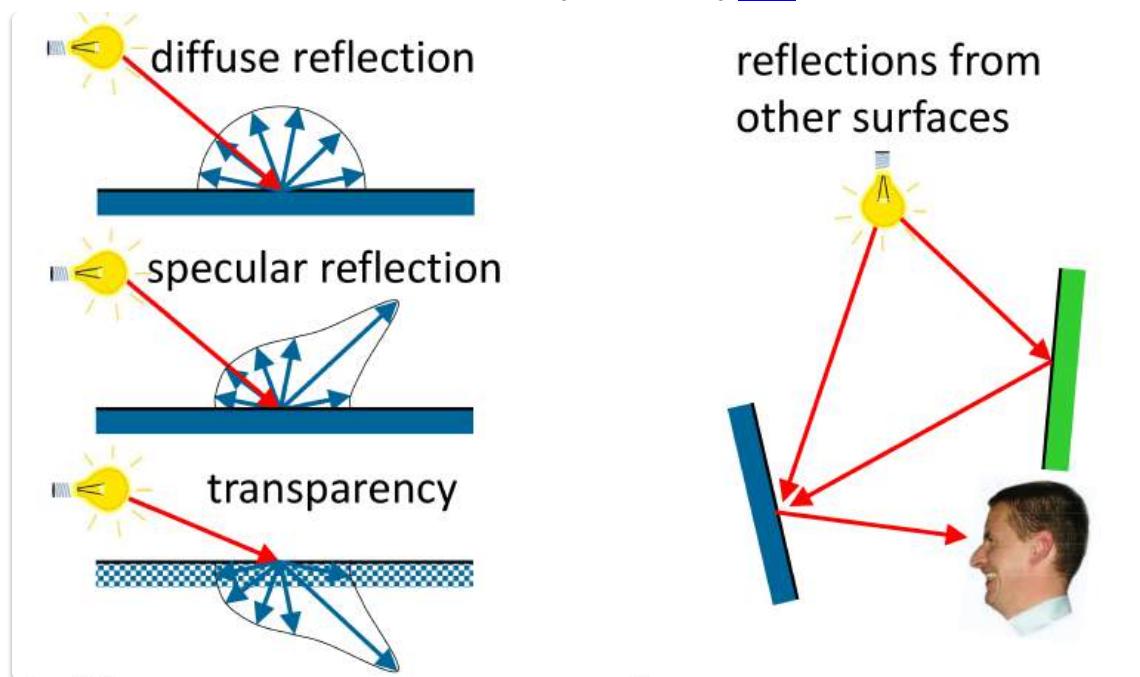
- Lichtrichtung (z.B. paralleles Sonnenlicht)
- Punktlichtquellen (strahlen Licht von einem einzelnen Punkt in alle Richtungen)
- Gerichtete Punktlichtquellen (kombinieren Punktlichtquelle mit einer kegelförmigen oder rechteckigen Lichtverteilung)
- Flächige Lichtquellen (emittieren Licht von einer ausgedehnten Oberfläche)
- ... (weitere Formen sind möglich)
- **Eigenschaften:**
  - Helligkeit (Intensität des Lichts)
  - Farbe (Spektrale Zusammensetzung des Lichts)
  - Entfernung (bei einigen Lichtquellen relevant für die Intensitätsabnahme)
  - ... (weitere Eigenschaften können definiert werden)



## Objektoberflächen

EVC\_Skriptum\_CG, p.35

- **Wechselwirkung mit einfallendem Licht:** Oberflächen können Licht auf verschiedene Weisen beeinflussen:
  - **Diffuse Reflexion:** Licht wird in alle Richtungen gleichmäßig reflektiert (Beispiele: Papier, Kreide).
  - **Spiegelnde Reflexion:** Licht wird bevorzugt in die Spiegelungsrichtung reflektiert (Beispiele: Lack, Metall).
  - **Transparenz:** Licht durchdringt die Oberfläche und tritt auf der anderen Seite wieder aus (Beispiele: Glas, Wasser).
- **Realität:** Die meisten Oberflächen weisen eine Kombination dieser Eigenschaften auf.
- **Indirekte Beleuchtung:** Es ist wichtig zu beachten, dass Licht nicht nur direkt von Lichtquellen auf Oberflächen trifft, sondern auch von anderen Oberflächen reflektiert wird und somit zur Beleuchtung beiträgt.



## Ein einfaches Beleuchtungsmodell

EVC\_Skriptum\_CG, p.35

- **Hintergrund:** Die physikalisch genaue Simulation von Licht und seiner Interaktion mit Oberflächen ist sehr aufwendig.
- **Ansatz in der Praxis:** Verwendung vereinfachter, empirischer Beleuchtungsmodelle.
- **Grundstruktur (ungefähre Darstellung):** (Die genaue Struktur wird in den folgenden Abschnitten detaillierter erläutert.)
- **Ziel:** Eine visuell plausible Beleuchtung mit überschaubarem Rechenaufwand zu erzielen.

## Hintergrundlicht (Ambientes Licht)

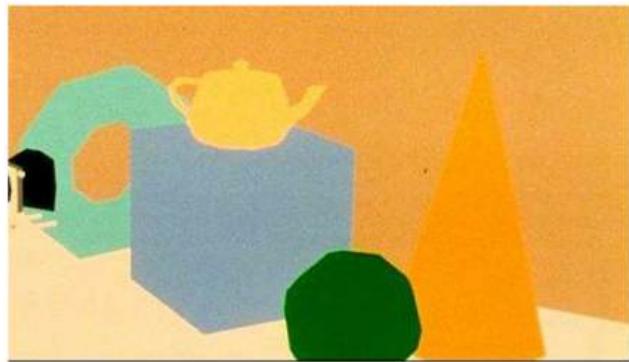
EVC\_Skriptum\_CG, p.35

- **Realitätsbezug:** Objekte strahlen einen Teil des auf sie treffenden Lichts ab, wodurch es auch in Bereichen ohne direkte Beleuchtung nicht vollständig dunkel ist.
- **Definition:** Dieses überall vorhandene, indirekte Basislicht wird als **ambientes Licht** oder **Hintergrundlicht** bezeichnet.
- **Implementierung in einfachen Beleuchtungsmodellen:** Ein konstanter Helligkeitswert ( $I_a$ ) wird zu jeder Beleuchtungsberechnung addiert, um diesen globalen Lichtanteil zu approximieren.

- ambient light (background light)  $I_a$

- constant over a surface
- independent of viewing direction
- diffuse-reflection coefficient  $k_d$  ( $0 \leq k_d \leq 1$ )
- approximation of global diffuse lighting effects

$$L_{\text{ambdiff}} = k_d I_a$$



## Lambert'sches Gesetz (Diffuse Reflexion)

[EVC\\_Skriptum\\_CG, p.35](#), [EVC\\_Skriptum\\_CG, p.36](#)

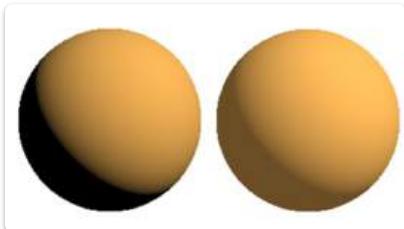
- **Kernaussage:** Die Helligkeit einer diffus reflektierenden Oberfläche ist proportional zum Kosinus des Winkels zwischen der Oberflächennormale und der Richtung zur Lichtquelle. Flacher Lichteinfall führt zu dunkleren Oberflächen.
- **Bedeutung:** Erzeugt den Eindruck räumlicher Form durch Helligkeitsvariationen.
- **Formel für die resultierende Helligkeit (L) durch diffuse Reflexion:**

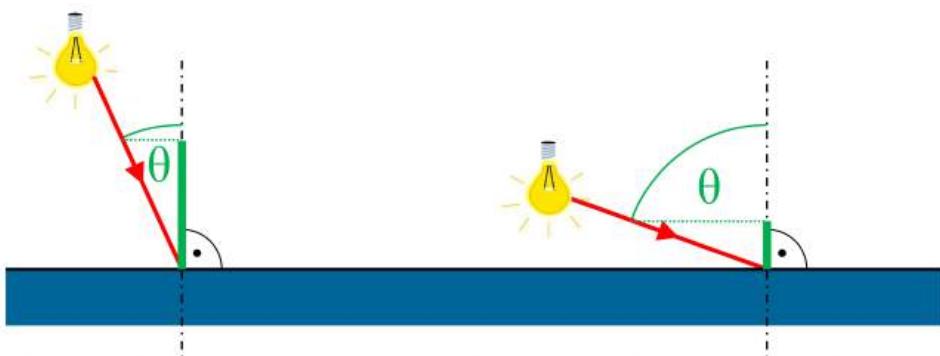
$$L = k_d \cdot I \cdot \cos \theta$$

oder in Vektorform:

$$L = k_d \cdot I \cdot (\mathbf{n} \cdot \mathbf{l})$$

- $I$ : Helligkeit der relevanten Lichtquelle.
- $k_d$ : Diffuser Reflexionskoeffizient der Oberfläche ( $0 \leq k_d \leq 1$ ), gibt den Anteil des einfallenden Lichts an, der diffus reflektiert wird.
- $\theta$ : Winkel zwischen der Oberflächennormale ( $\mathbf{n}$ ) und der Richtung zur Lichtquelle ( $\mathbf{l}$ ).
- $\mathbf{n} \cdot \mathbf{l}$ : Skalarprodukt zwischen dem Normalenvektor und dem Lichtrichtungsvektor.
- **Kombination mit ambientem Licht:**
  - Gesamte Helligkeit = Beitrag durch diffuse Reflexion + Beitrag durch ambientes Licht ( $I_a$ ).
  - Führt bereits zu einer ansprechenden Darstellung (siehe Beispiel der Kugeln)





$$L = I \cdot \cos \theta$$

when considering  
the material:

I ... light source intensity

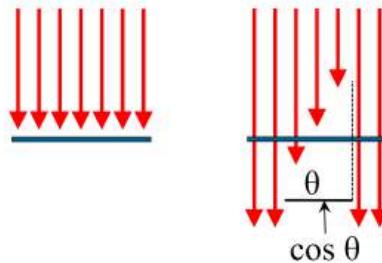
L ... pixel color

$k_d$  ... diffuse coefficient

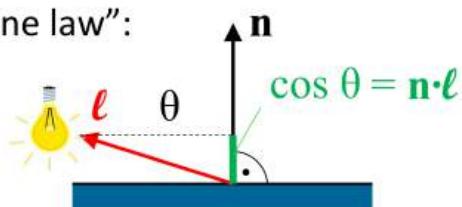
$$L = k_d \cdot I \cdot \cos \theta$$

for ideal diffuse reflectors (Lambertian reflectors)

brightness depends on  
orientation of the surface:



"Lambert's cosine law":

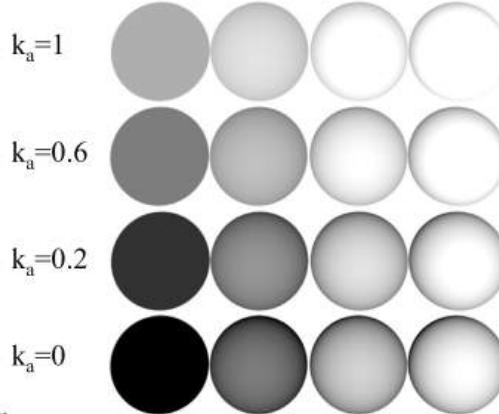


$$L_{\text{diff}} = k_d \cdot I \cdot (\mathbf{n} \cdot \mathbf{l})$$

total diffuse reflection:

$$L_{\text{diff}} = k_a I_a + k_d I(\mathbf{n} \cdot \mathbf{l})$$

$$k_d=0 \quad k_d=0.3 \quad k_d=0.7 \quad k_d=1$$

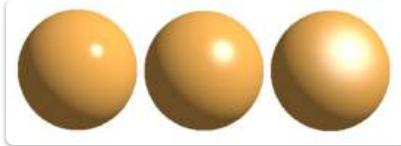


(sometimes extra  $k_a$   
for ambient light)

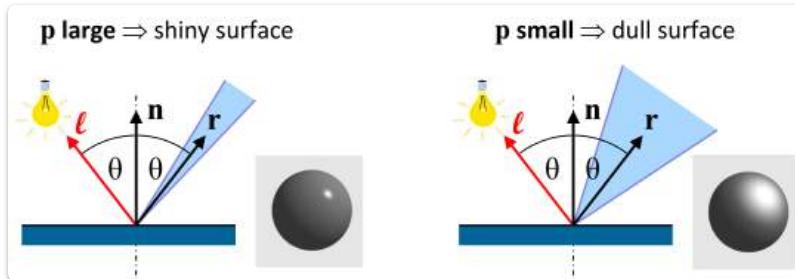
Werner Purgathofer

## Glanzpunkte (Specular Highlights)

- Fast jede Oberfläche ist auch etwas spiegelnd. Wenn man diesen Aspekt nicht mitmodelliert, dann wirken alle Materialien gleich stumpf.



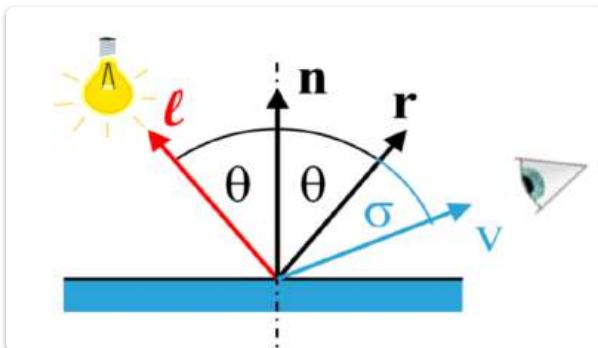
- Da die exakte spiegelnde Reflexion äußerst kompliziert zu berechnen ist, behilft man sich mit einer einfachen Funktion, die einen ähnlichen Verlauf hat wie das Highlight:  $\cos^p(\alpha)$ .
- Mit dem freien Parameter  $p$  lässt sich dabei die „Poliertheit“ der Oberfläche steuern:
  - Je größer  $p$  ist, desto kleiner wird der Glanzpunkt und desto glatter wirkt die Oberfläche** (linke Kugel im Bild).
  - Je kleiner  $p$  ist, desto matter wirkt die Oberfläche** (rechte Kugel im Bild).



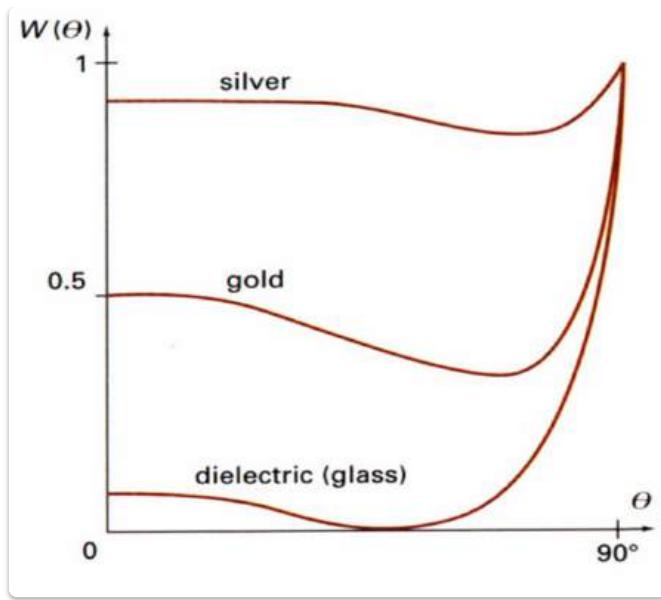
- Um diesen Effekt im richtigen Ausmaß zur Beleuchtung hinzufügen zu können, wird noch ein weiterer Faktor eingeführt, der spiegelnde Reflexionskoeffizient  $k_s$ .
- Der Glanz berechnet sich dann nach diesem sogenannten **Phong-Beleuchtungsmodell** so:

$$I_{spec} = k_s * I_L * \cos^p(\alpha)$$

- $I_L$ : Intensität des Lichts.
- $\alpha$ : Winkel zwischen dem exakten Reflexionsstrahl  $r$  und der Richtung zum Auge  $v$ .
- Etwas näher an der Wahrheit ist die Verwendung des **Fresnel'schen Reflexionsgesetzes**, das beschreibt, dass der Spiegelungsgrad auch vom Lichteinfallswinkel  $\theta$  abhängt.
- Also ist der Koeffizient  $k_s$  eigentlich eine Funktion  $W(\theta)$  der Lichteinfallsrichtung  $l$ .
- Für die meisten Materialien ist dieser Wert aber fast konstant. Daher wird auf diesen Aufwand verzichtet, wenn man nicht gerade ein Material darstellen will, bei dem der Effekt auffällt.

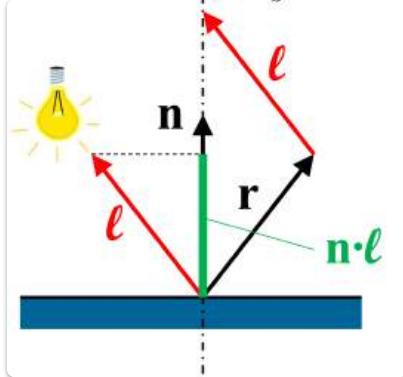


- Das Bild zeigt die Abhängigkeit dieser Funktion  $W(\theta)$  vom Winkel zwischen Lichteinfall und Normale auf der Oberfläche für drei verschiedene Materialien (Silber, Gold, dielektrisches Material).



- Bei der Berechnung des Reflexionsvektors  $r$  muss man noch bedenken, dass es sich hier um **Vektoren im 3D-Raum** handelt, wobei  $l$  (Lichtrichtung),  $n$  (Oberflächennormale) und  $r$  in einer Ebene liegen müssen und alle Länge eins haben sollen.
  - $r$  ergibt sich zu:

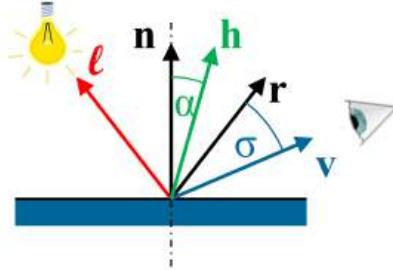
$$r = (2n \cdot l)n - l$$



- Weil die Glanzfunktion sowieso nur eine grobe Näherung ist, verwendet man auch häufig eine einfachere Formel, in der  $r \cdot v$  (Winkel zwischen Reflexionsrichtung und Blickrichtung) durch  $n \cdot h$  ersetzt wird.
  - $h$ : Halbierungsvektor zwischen  $l$  und  $v$ .

simplified Phong model with halfway vector  $\mathbf{h}$

$$L_{\text{spec}} = k_s \cdot I \cdot (v \cdot r)^p \quad \rightarrow \quad L_{\text{spec}} = k_s \cdot I \cdot (n \cdot h)^p$$



$$\mathbf{h} = \frac{\ell + \mathbf{v}}{\|\ell + \mathbf{v}\|}$$

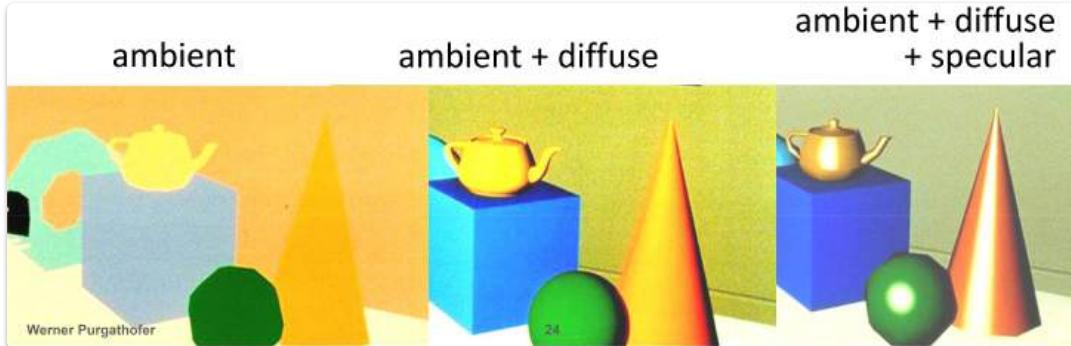
this revised model is called  
“Blinn-Phong Shading”

- Der Winkel zwischen  $n$  und  $h$  ist oft sehr ähnlich dem Winkel zwischen  $r$  und  $v$ .

- Das resultierende Modell nennt man **Blinn-Phong-Beleuchtungsmodell**.
- Wenn wir alle bisherigen Komponenten zusammensetzen, erhalten wir ein einfaches komplettes Beleuchtungsmodell:

$$L = k_a * I_a + \sum_{i=1,\dots,N} (k_d * I_i * (n \cdot l_i) + k_s * I_i * (n \cdot h_i)^p)$$

- $L$ : Gesamte Beleuchtung.
- $k_a$ : Ambienter Reflexionskoeffizient.
- $I_a$ : Intensität des ambienten Lichts.
- $N$ : Anzahl der Lichtquellen.
- $k_d$ : Diffuser Reflexionskoeffizient.
- $I_i$ : Intensität der  $i$ -ten Lichtquelle.
- $n$ : Oberflächennormale.
- $l_i$ : Richtung zur  $i$ -ten Lichtquelle.
- $k_s$ : Spekularer Reflexionskoeffizient.
- $h_i$ : Halbierungsvektor zwischen  $l_i$  und der Blickrichtung  $v$ .
- $p$ : Glanz-Exponent (Polierheit).



- Es gibt noch viele weitere Aspekte, die man berücksichtigen muss, um der Realität näher zu kommen, aber diese werden hier nicht näher beschrieben: Farbverschiebungen in Abhängigkeit der Blickrichtung, Einfluss der Entfernung der Lichtquelle, anisotrope Oberflächen und Lichtquellen, Transparenz, atmosphärische Effekte, Schatten und so weiter.

## Schattierung von Polygonen

### Flat-Shading

[EVC\\_Skriptum\\_CG, p.37](#)

- Beim Schattieren eines Polygons hat klarerweise jeder Punkt die gleichen Oberflächeneigenschaften, vor allem auch den gleichen Normalvektor.
- Beim einfachen Ausfüllen jedes Polygons mit einer Farbe werden die Grenzen zwischen den Polygone deutlich störend erkennbar.
- Der sogenannte **Mach-Band-Effekt**, ein kantenverstärkender Mechanismus des Auges, macht das Problem dabei noch ärger als es ist.

- Dieser Effekt lässt uns Kanten die dunklere Seite dunkler wahrnehmen als sie ist, und die hellere Seite heller als sie ist.
- Die einfachste Lösung dieses Problems ist das Interpolieren der Schattierung zwischen den Polygonen. Dazu sind zwei Verfahren üblich: **Gouraud-Schattierung** und **Phong-Schattierung**.



## Gouraud-Schattierung

[EVC\\_Skriptum\\_CG](#), p.37

Die Gouraud-Schattierung interpoliert die berechneten Helligkeitswerte über die Polygonflächen. Dazu werden an den Eckpunkten der Polygone Helligkeitswerte berechnet und von diesen aus durch lineare Interpolation jedes Polygon gefüllt.

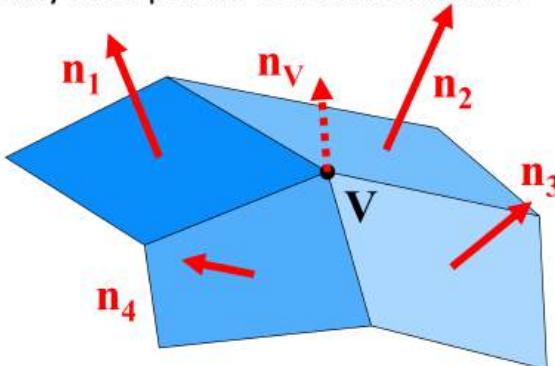


Konkret geht das so:

1. **Berechnung der Eckennormalen:** An jedem Eckpunkt wird eine Normale als Mittelwert der Normalen aller angrenzenden Polygone berechnet. Dies ist natürlich nur ein Näherungswert der Normale der echten zugrundeliegenden Fläche.
2. **Berechnung der Eckpunktintensitäten:** Aus den Eigenschaften der Oberfläche, der Normale (der gemittelten Eckennormalen) und der Lichteinfallsrichtung wird für jeden Eckpunkt ein Helligkeitswert („Schattierung“) berechnet. Beachte, dass dadurch angrenzende Polygone an diesen Eckpunkten alle die gleichen Werte erhalten.

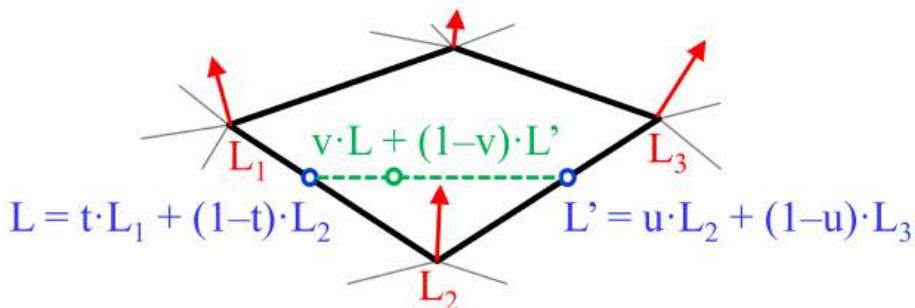
### intensity-interpolation:

- determine average unit normal vector at each polygon vertex
- apply illumination model to each vertex
- linearly interpolate vertex intensities



$$\mathbf{n}_v = \frac{\sum_{k=1}^N \mathbf{n}_k}{\left\| \sum_{k=1}^N \mathbf{n}_k \right\|}$$

- Interpolation entlang der Polygonkanten:** Entlang der Polygonkanten werden die Helligkeitswerte linear interpoliert, d.h. es wird für jeden Schnittpunkt mit einer Scanline ein Wert ermittelt. Beachte, dass dadurch für aneinander grenzende Polygone entlang der gemeinsamen Kante die gleichen Werte entstehen.
- Interpolation entlang der Scanlines:** Entlang jeder Scanline wird von der linken bis zur rechten Polygongrenze wieder linear interpoliert. Dadurch haben nebeneinander liegende Pixel immer eine sehr ähnliche Helligkeit und es kommt zu keinen sichtbaren Kanten (im Idealfall).

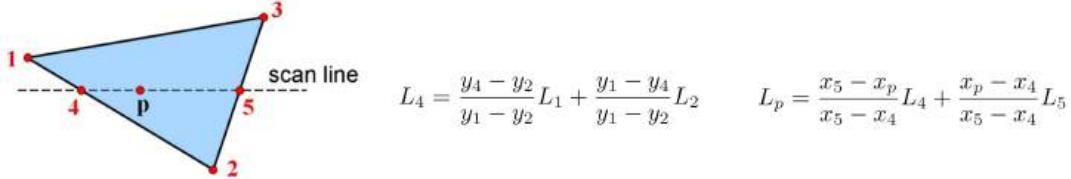


- find normal vectors at corners and calculate shading (intensities) there:  $L_i$
- interpolate intensities along edges linearly:  $L, L'$
- interpolate intensities along scanlines linearly:  $L_p$

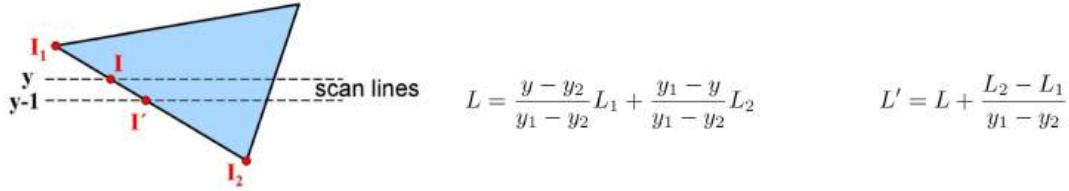
Dennoch verbleiben **Fehlerquellen**. So wird die Silhouette natürlich nicht verändert, dadurch verbleiben störende Polygonkanten sichtbar:



Einfache lineare Interpolation zur Berechnung eines Pixelwertes  $L_p$ :



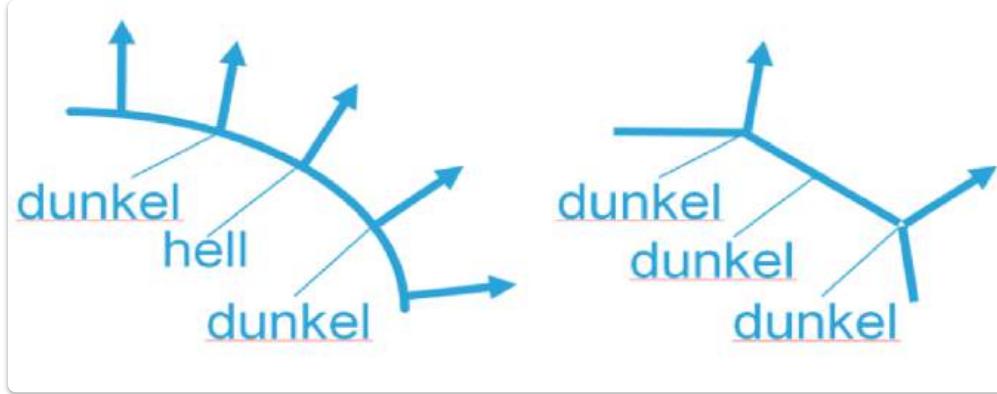
Die *lineare (!) Interpolation* der Intensitäten kann natürlich wieder inkrementell erfolgen, z.B.:



## Probleme bei Gouraud-Schattierung (Glanzpunkte)

[EVC\\_Skriptum\\_CG, p.37](#)

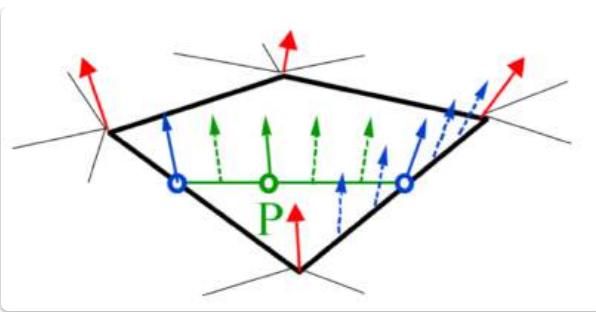
- Zufällige Interpolationsergebnisse bei Glanzpunkten möglich.
- Abhängig davon, ob Eckennormale zufällig Glanzpunkt erzeugt.
- Störend bei bewegten Objekten (Glanzpunkt "wandert").



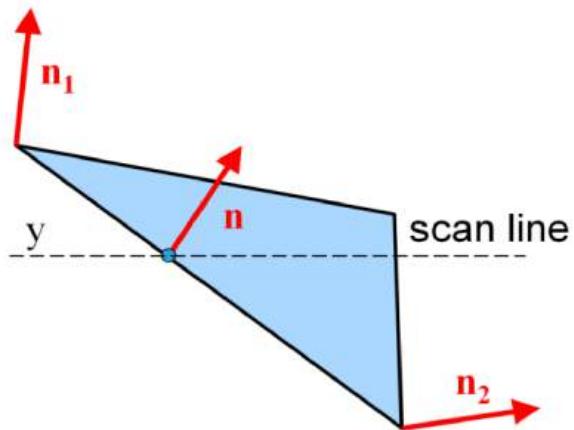
## Phong-Schattierung

[EVC\\_Skriptum\\_CG, p.38](#)

- Alternative zu Gouraud-Schattierung für konsistenter Glanzpunkte.
- **Ablauf:**
  1. Normalen an Polygon-Eckpunkten berechnen.
  2. Diese Normalen entlang Polygonkanten interpolieren.
  3. Entlang Scanlines interpolierte Normalen weiter interpolieren (pro Pixel).
  4. Pro Pixel mit interpolierter Normalen Helligkeit nach Beleuchtungsmodell berechnen.
- **Unterschied:** Wir drehen Punkt 2 und 3 um. Also ich interpoliere die Vektoren und schattiere dann.
- **Vorteil:** Konsistenter Glanzpunkte.
- **Nachteil:** Höherer Aufwand (Beleuchtung pro Pixel).



Normalvektorinterpolation:



$$n = \frac{y - y_2}{y_1 - y_2} n_1 + \frac{y_1 - y}{y_1 - y_2} n_2$$

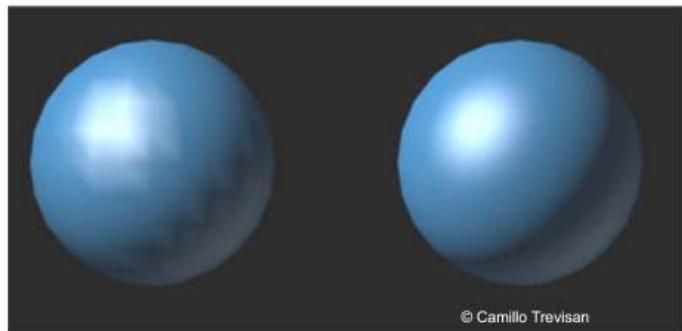
### ZUR BEACHTUNG:

- Phong-Beleuchtungsmodell und Phong-Schattierung sind zwei unabhängige Konzepte!

### Vergleich zwischen Gouraud und Phong:

comparison to Gouraud shading

- better highlights
- less Mach banding
- more costly
- wrong silhouette stays!



© Camillo Trevisan

# 9. Multiskalenrepräsentationen

EVC\_Skriptum\_CV, p.46

- **Abbildung 38:** Beispielbild zur Gesichtserkennung auf mehreren Skalen



- **Nachbarschaftsoperationen:**
  - Sind ein erster Schritt in der Bildanalyse.
  - Extrahieren lediglich **lokale Merkmale** (Größenordnung von maximal einigen Pixeln).
- **Großkalige Information:**
  - Bilder enthalten auch Information über größere Bereiche.
  - Zur Extraktion sind **größere Filtermasken** notwendig.
  - **Problem:** Erhöhter Rechenaufwand bei großen Filtermasken (Verdopplung der Filtergröße  $\Rightarrow$  vierfacher Rechenaufwand bei 2D-Bildern).
- **Grauwertänderungen und Skalenfehlanpassung:**
  - Grauwertänderungen durch Kontrastunterschiede zwischen Objekten und Hintergrund können schwer zu bestimmen sein.
  - **Ursache:** Skalenfehlanpassung - Grauwerte ändern sich über größere Distanzen, die die detektierenden Operatoren nicht erfassen.
- **Abhängigkeit der Merkmalsdetektion von der Skala:**
  - Die Detektion bestimmter Merkmale in einem Bild hängt von der **richtigen Skala** ab.
  - Die richtige Skala hängt von den **charakteristischen Größen** im zu detektierenden Objekt ab (z.B. Größe der Gesichter bei Gesichtserkennung).
- **Multiskalenanalyse als Lösung:**

- Für die optimale Verarbeitung muss ein Bild in **unterschiedlichen Skalen** vorliegen.
- Dies erfordert eine Darstellung in **mehreren Auflösungsstufen**.
- **Definition:** Multiskalenanalyse ist ein mathematisches Konzept, das die Signalanalyse auf unterschiedlichen Auflösungsstufen beschreibt.
- **Vorteile der Multiskalenanalyse:**
  - **Feine Details:** Benötigen die maximale verfügbare Auflösung.
  - **Grobe Strukturen:** Können mit geringerem Aufwand bei **reduzierter Auflösung** analysiert werden.

## Abtastung

---

[EVC\\_Skriptum\\_CV, p.46, p.47](#)

- **Diskreteisierung von Signalen:** Bei der Bildaufnahme wird eine kontinuierliche Funktion in eine diskrete Funktion umgewandelt, was als **Abtastung (Sampling)** bezeichnet wird.
- **Definition:** Abtastung ist die Entnahme von Abtastwerten der kontinuierlichen Funktion an bestimmten Punkten in der Zeit oder im Raum, üblicherweise in regelmäßigen Abständen.
- **Formale Beschreibung mit der Impulsfunktion (Deltafunktion oder Dirac-Impuls)  $\delta(x)$ :**
  - Modelliert einen kontinuierlichen, "idealen" Impuls.
  - Ist überall null mit Ausnahme des Ursprungs, wo ihr Wert zwar ungleich null, aber undefiniert ist.
  - Ihr Integral ist eins:

$$\int_{-\infty}^{\infty} \delta(x) dx = 1$$

- **Interpretation von  $\delta(x)$ :** Kann man sich als einzelnen Impuls an der Position null vorstellen, der unendlich schmal ist, aber endliche Energie aufweist.
- **Modellierung der Abtastung mit der Impulsfunktion:**
  - Eine kontinuierliche Funktion  $g(x)$  wird mit der Impulsfunktion  $\delta(x)$  punktweise multipliziert.
  - Dadurch erhält man einen einzelnen, diskreten Abtastwert der Funktion  $g(x)$  an der Stelle  $x = 0$ .
- **Abtastung an beliebigen Stellen durch Verschieben der Impulsfunktion:**
  - Durch Verschieben der Impulsfunktion um eine Distanz  $x_0$  kann  $g(x)$  an jeder beliebigen Stelle  $x = x_0$  abgetastet werden (Multiplikation mit  $\delta(x - x_0)$ ).
- **Abtastung einer kontinuierlichen Funktion  $g(x)$  an einer Folge von  $N$  Positionen  $x_i = 1, 2, \dots, N$ :**
  - Kann als Summe der  $N$  Einzelabtastungen dargestellt werden:

$$g_s(x) = g(x) \cdot [\delta(x - 1) + \delta(x - 2) + \dots + \delta(x - N)]$$

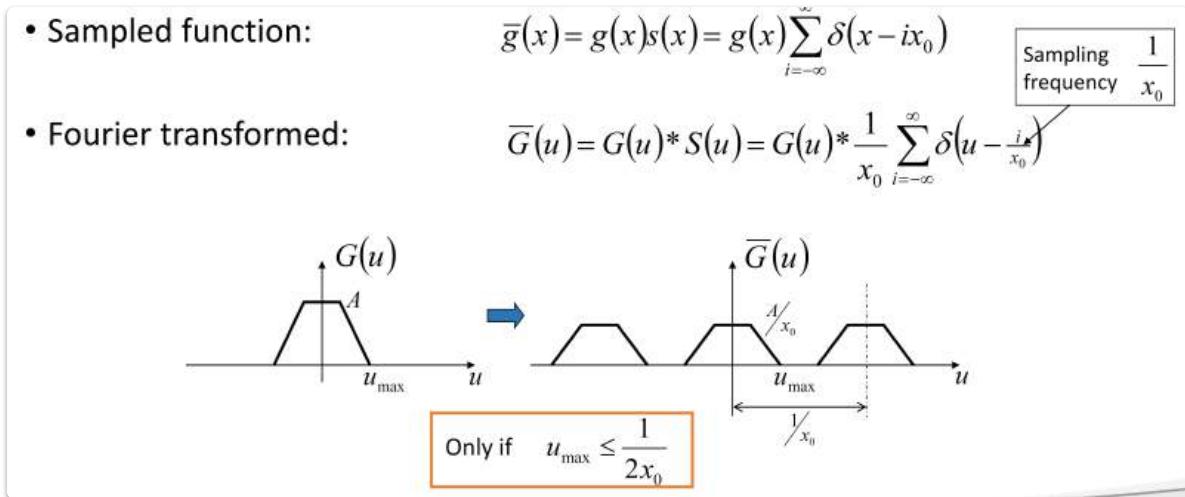
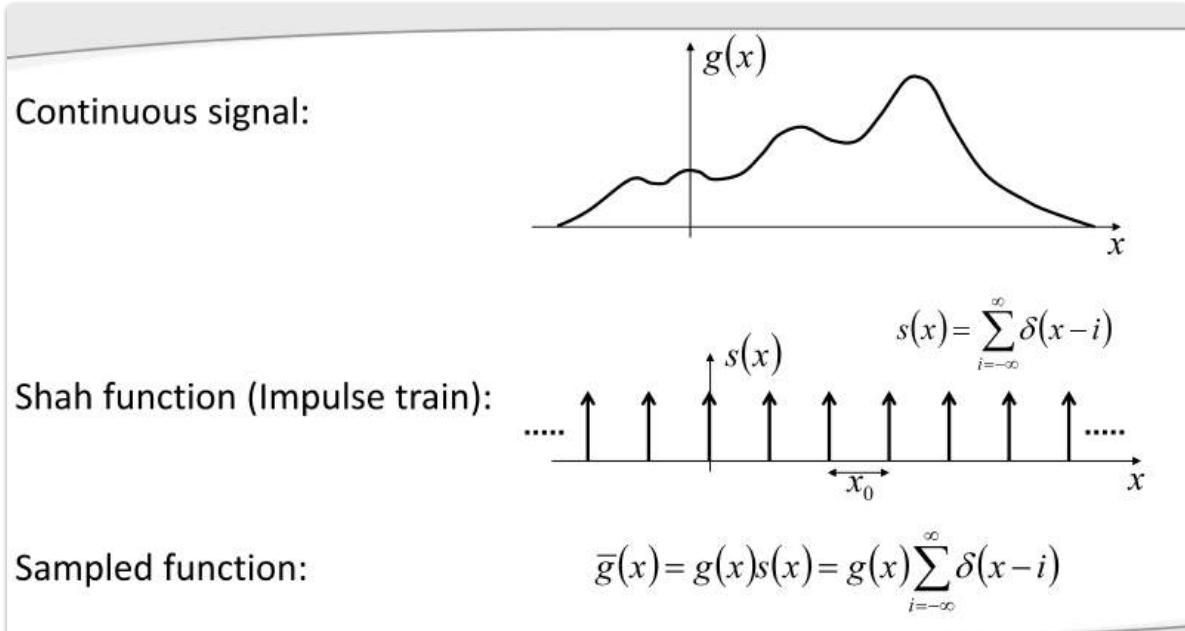
$$g_s(x) = g(x) \cdot \sum_{i=1}^N \delta(x - i)$$

- **Pulsfolge:** Diese Summe von verschobenen Einzelimpulsen wird als "Pulsfolge" bezeichnet.
- **Kammfunktion oder Shah-Funktion:** Wenn die Pulsfolge in beiden Richtungen ins Unendliche erweitert wird, erhalten wir die Kammfunktion oder Shah-Funktion:

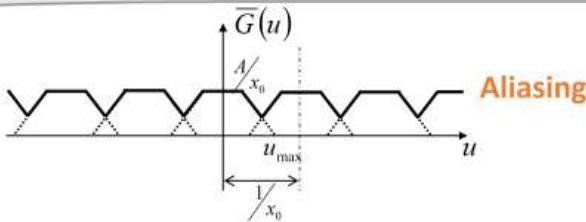
$$S(x) = \sum_{i=-\infty}^{\infty} \delta(x - i)$$

- **Auswirkungen der Abtastung auf das Frequenzspektrum:**
  - Die Abtastung einer kontinuierlichen Funktion hat auch Auswirkungen auf das Frequenzspektrum des resultierenden (diskreten) Signals.
  - Der Einsatz der Kammfunktion als formales Modell des Abtastvorgangs macht es einfacher, diese spektralen Auswirkungen zu interpretieren.
- **Eigenschaften der Kammfunktion im Frequenzbereich:**
  - Die Kammfunktion besitzt die seltene Eigenschaft, dass ihre Fouriertransformierte wiederum eine Kammfunktion ist, also den gleichen Funktionstyp hat.
  - Das Produkt zweier Funktionen in einem Raum (entweder im Orts- oder im Spektralraum) entspricht einer linearen Faltung im jeweils anderen Raum.
- **Spektrum des abgetasteten Signals:**
  - Das Fourierspektrum der Abtastfunktion (Kammfunktion im Zeitbereich) ist wiederum eine Kammfunktion im Frequenzbereich mit einem Impuls bei jeder ganzzahligen Frequenz.
  - Die Faltung einer beliebigen Funktion mit einem Impuls  $\delta(x)$  ergibt aber wieder die ursprüngliche Funktion:  $f(x) * \delta(x) = f(x)$ .
  - Das hat zur Folge, dass im Fourierspektrum des abgetasteten Signals  $\tilde{g}(u)$  das Spektrum  $G(u)$  des ursprünglichen, kontinuierlichen Signals unendlich oft, nämlich an jedem Puls im Spektrum der Abtastfunktion, repliziert wird.
  - Das daraus resultierende Fourierspektrum ist daher periodisch mit der Periodenlänge  $1/x_0$ , also im Abstand der Abtastfrequenz  $1/x_0$ .
- **Vermeidung von Aliasing:**
  - Solange sich die durch die Abtastung replizierten Spektralkomponenten in  $\tilde{g}(u)$  nicht überlappen, kann das ursprüngliche Spektrum  $G(u)$  - und damit auch das ursprüngliche, kontinuierliche Signal  $g(x)$  - ohne Verluste aus einer beliebigen Replika von  $G(u)$  aus dem periodischen Spektrum  $\tilde{g}(u)$  rekonstruiert werden.
- **Nyquist-Shannon-Abtasttheorem (impliziert):**
  - Zur Diskretisierung eines kontinuierlichen Signals  $g(x)$  mit Frequenzanteilen im Bereich  $0 \leq |u| \leq u_{max}$  benötigen wir eine Abtastfrequenz  $u_s$ , die mindestens doppelt so hoch wie die maximale Signalfrequenz  $u_{max}$  ist ( $u_s > 2u_{max}$ ).
- **Entstehung von Aliasing:**

- Wird diese Bedingung nicht eingehalten, dann überlappen sich die replizierten Spektralteile im Spektrum des abgetasteten Signals, und das Spektrum wird verfälscht mit der Folge, dass das ursprüngliche Signal nicht mehr fehlerfrei aus dem Spektrum rekonstruiert werden kann.
- Dieser Effekt wird als **Aliasing** bezeichnet.



$$\text{If } u_{\max} > \frac{1}{2x_0}$$



When can we recover  $G(u)$  from  $\bar{G}(u)$ ?

$$\text{Only if } u_{\max} \leq \frac{1}{2x_0} \text{ (Nyquist Frequency)}$$

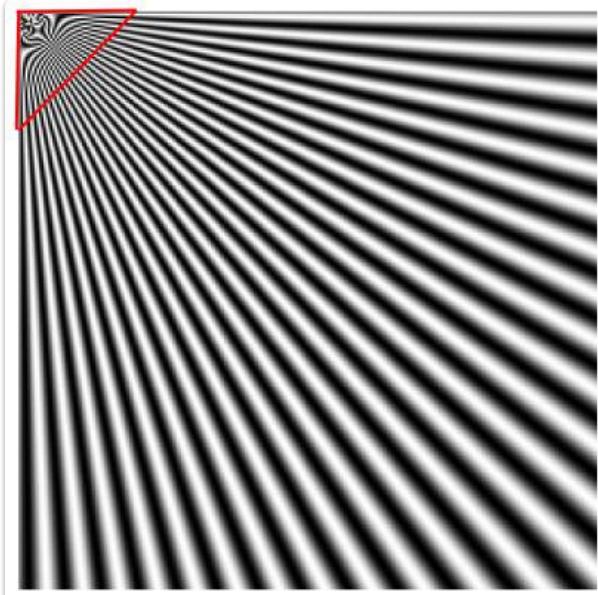
We can use

$$C(u) = \begin{cases} x_0 & |u| < \frac{1}{2x_0} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Then } G(u) = \bar{G}(u)C(u) \text{ and } f(x) = \text{IFT}[G(u)]$$

Sampling frequency must be greater than  $2u_{\max}$

Beispiel für Aliasing:

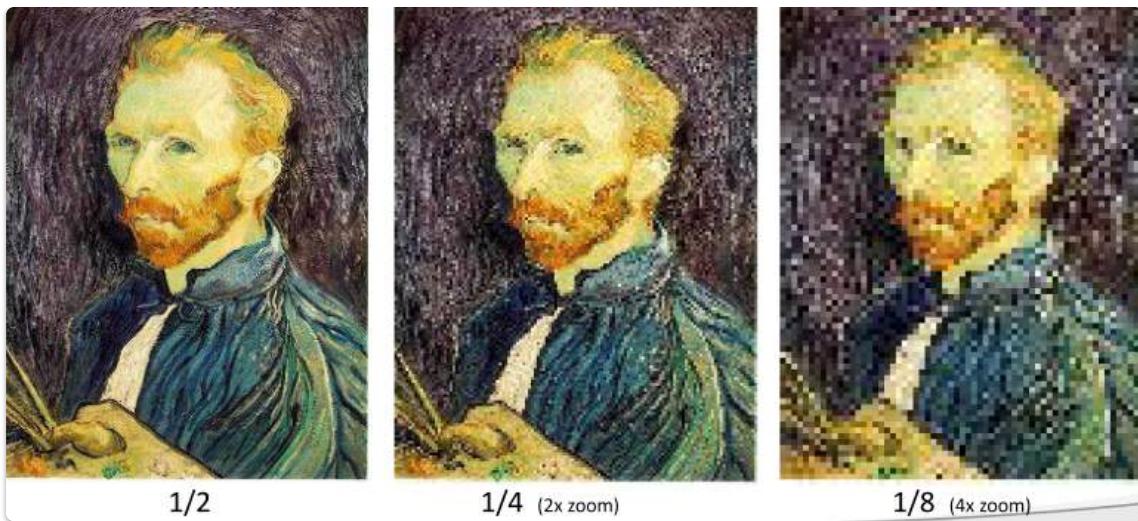


Das was wir hier machen zieht darauf ab dass wir nicht solche Bildfehler haben

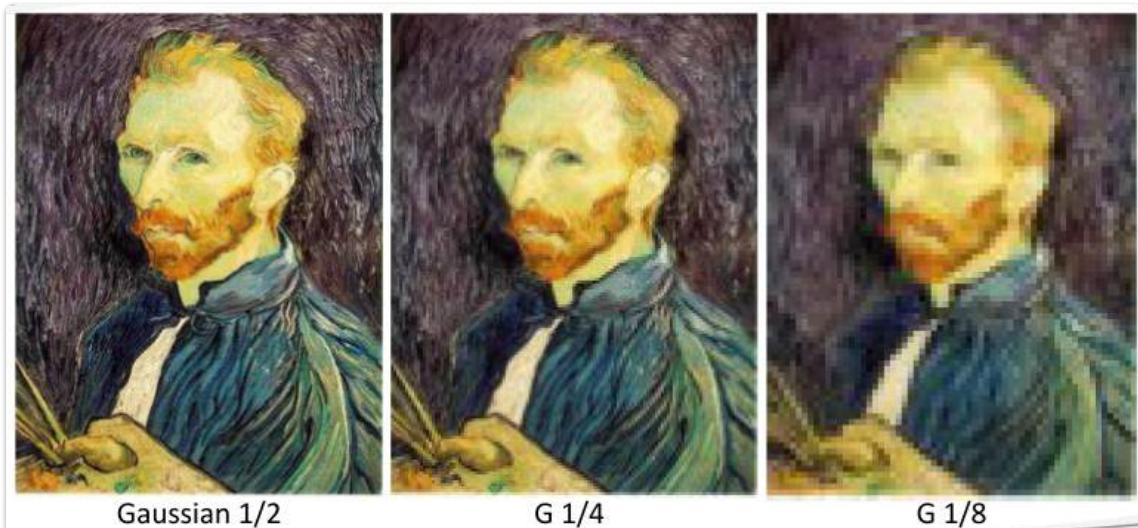
## Bildskalierung

[EVC\\_Skriptum\\_CV](#), p.47, p.48

- **Motivation für Bildreskalierung (Subsampling):** Wie können Bilder größengemäß angepasst werden, wenn ihre Originalgröße z.B. nicht auf den Bildschirm passt?
- **Beispiel für Subsampling:** Erzeugen eines Bildes, das nur ein Viertel so groß ist wie das Original.
- **Naives Subsampling (Löschen jeder zweiten Zeile und Spalte):** Führt zu einem Abtastproblem und erzeugt **Aliasing-Effekte**.
- Wenn man einfach nur Pixel steichen würde:



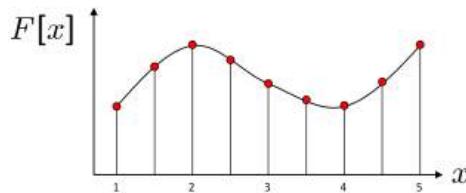
- **Korrekte Vorgehensweise für Subsampling:**
  1. Das Bild muss zuerst mit einem Gauß-Filter gefiltert werden.
  2. Anschließend wird das gefilterte Bild verkleinert.
  3. **Bedingung für den Filter:** Der Filter muss nach dem Abtasttheorem doppelt so groß wie die gewünschte Verkleinerung sein (bezogen auf die Frequenzen).
- **Grundlage für Multiskalenanalyse:** Dieser Theorie (des korrekten Subsamplings unter Berücksichtigung des Abtasttheorems) folgen im Weiteren auch alle Betrachtungen der Multiskalenanalyse.
- Hier das selbe Bild mit Gaußfilter:



## Resampling/Upsampling

- **Notwendigkeit:** Bei der Vergrößerung eines Bildes müssen neue Datenpunkte hinzugefügt werden.
- **Interpolation:** Die Werte dieser neuen Datenpunkte werden aus den benachbarten Pixeln des ursprünglichen Bildes interpoliert.

- What about arbitrary scale reduction?
- How can we increase the size of the image?



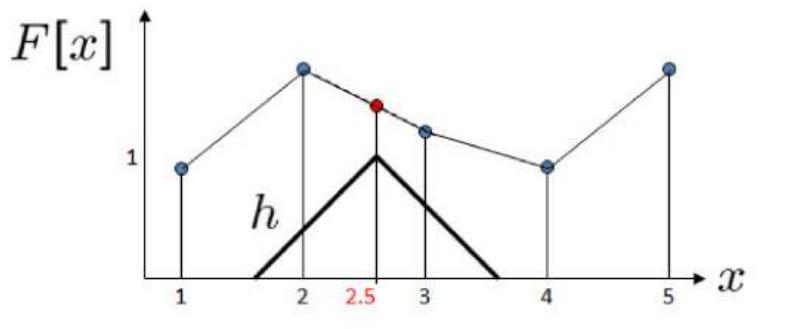
- Recall how a digital image is formed:

$$F[x, y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

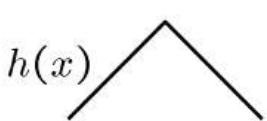
- **Grauwerte:** Die diskreten Grauwerte der Matrix des vergrößerten Bildes werden aus benachbarten Pixeln des Ausgangsbildes interpoliert.
- **Geometrie:** Die neue Matrix ist geometrisch durch das gewählte Bezugssystem definiert.
- **Neue Bildelemente:** Setzen sich aus Teilbereichen von Bildelementen der Eingabebildmatrix zusammen.
- **Filterkern:** Zur Grauwertzuweisung muss ein Filterkern definiert werden.
- **Gängige Resamplingverfahren:**

- **Bilineare Interpolation:** Berechnet den Wert einfach aus den Nachbarwerten (

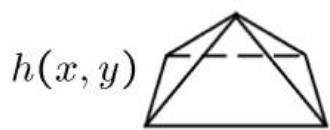


).

- **Filterfunktion:** Die Filterfunktion  $h(x)$  wird über  $F(x)$  gelegt und erzeugt den neuen Wert.
- **Bikubische Interpolation:** Geht von der vergrößerten neuen Bildmatrix aus und rechnet mit Hilfe von Transformationsgleichungen von den Mitten der dortigen Bildelemente in das Eingabebild zurück.

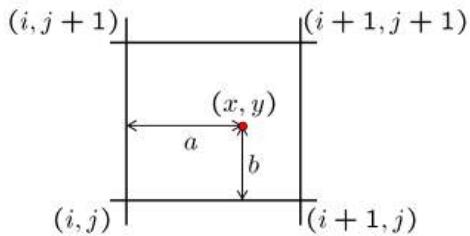


performs  
linear interpolation



performs  
bilinear interpolation

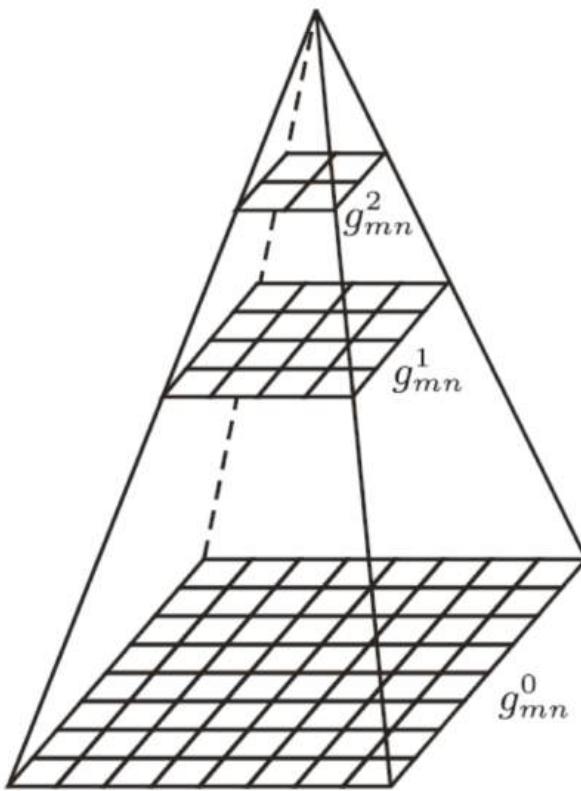
- A common method for resampling images



$$\begin{aligned}
 F(x, y) = & (1-a)(1-b) F(i, j) \\
 & + a(1-b) F(i+1, j) \\
 & + ab F(i+1, j+1) \\
 & + (1-a)b F(i, j+1)
 \end{aligned}$$

## Bildpyramiden zur effizienten Multiskalenanalyse

- **Motivation:**
  - Analyse von Strukturen auf verschiedenen Skalen für unterschiedliche Objekteigenschaften.
  - Erreichen einer gewissen Invarianz bezüglich der Objektgröße.
- **Problem naiver Multiskalenanalyse:** Beträchtlicher Rechenaufwand.
  - **Beispiel Korrelationsfilter:**
    - Bildgröße:  $N \times N$
    - Objektgröße:  $L \times L$
    - Aufwand Faltung im Ortsbereich:  $O(N^2 \cdot L^2)$
- **Lösung: Bildpyramiden**
  - **Konzept:** Mehrgitterdarstellung zur Verarbeitung von Signalen in unterschiedlichen Skalen.
  - **Effizienz:**
    - Feine Skalen → volle Auflösung erforderlich.
    - Grobe Strukturen → niedrigere Auflösung ausreichend.
  - **Aufbau:** Folge von Bildern, die von Stufe zu Stufe kleiner werden.
  - **Erzeugung:** Iterative Filterung und Unterabtastung des Bildes.
  - **Ergebnis:** Pyramide von Bildern, wobei jedes Bild eine bestimmte Skala repräsentiert.



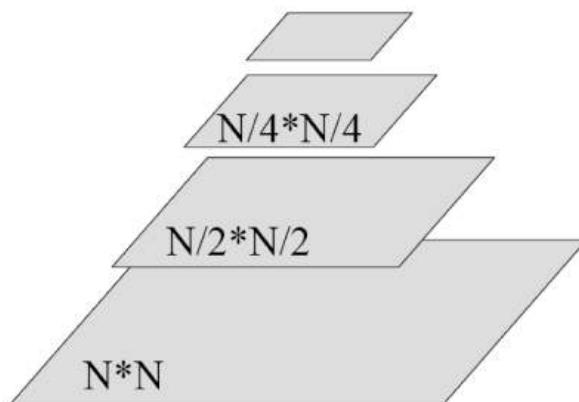
- **Problem einfache Abtastung:** Das einfache Entfernen jedes zweiten Bildpunkts in jeder zweiten Zeile führt zu Aliasing-Effekten.
- **Notwendigkeit Glättungsfilter:** Vor der Reduktion muss eine Glättung der höherfrequenten Strukturen durch einen passenden Glättungsfilter erfolgen.
- **Erzeugung des Skalenraums:** Größenreduktion muss mit einer angemessenen Glättung einhergehen.
- **Aufbau einer Bildpyramide:**
  - Start mit einem (o. B. d. A.) quadratischen Bild  $g_{mn}$  der Größe  $N \times N$  (Ebene  $v = 0$ ).
  - Berechnung der darüber liegenden Ebene mit reduzierter Größe (z.B. ein Viertel der Originalgröße).
  - **Schritt 1: Tiefpassfilterung** von  $g_{mn}$  auf halbe Bandbreite.
  - **Schritt 2: Unterabtastung** um den Faktor 2 (vermeidet Verletzung des Abtasttheorems durch vorherige Filterung).
  - **Ergebnis:** Gefiltertes und unterabgetastetes Bild  $g'_{mn}$  der Größe  $\frac{N}{2} \times \frac{N}{2}$  (Ebene  $v = 1$ ).
  - **Iteration:** Wiederholung des Vorgangs zur Erstellung der fehlenden Ebenen der Pyramide der Höhe  $K$ :  $v = 2, \dots, K; K = \log_2 N$ .

## Analyse auf verschiedenen Auflösungen in der Pyramide

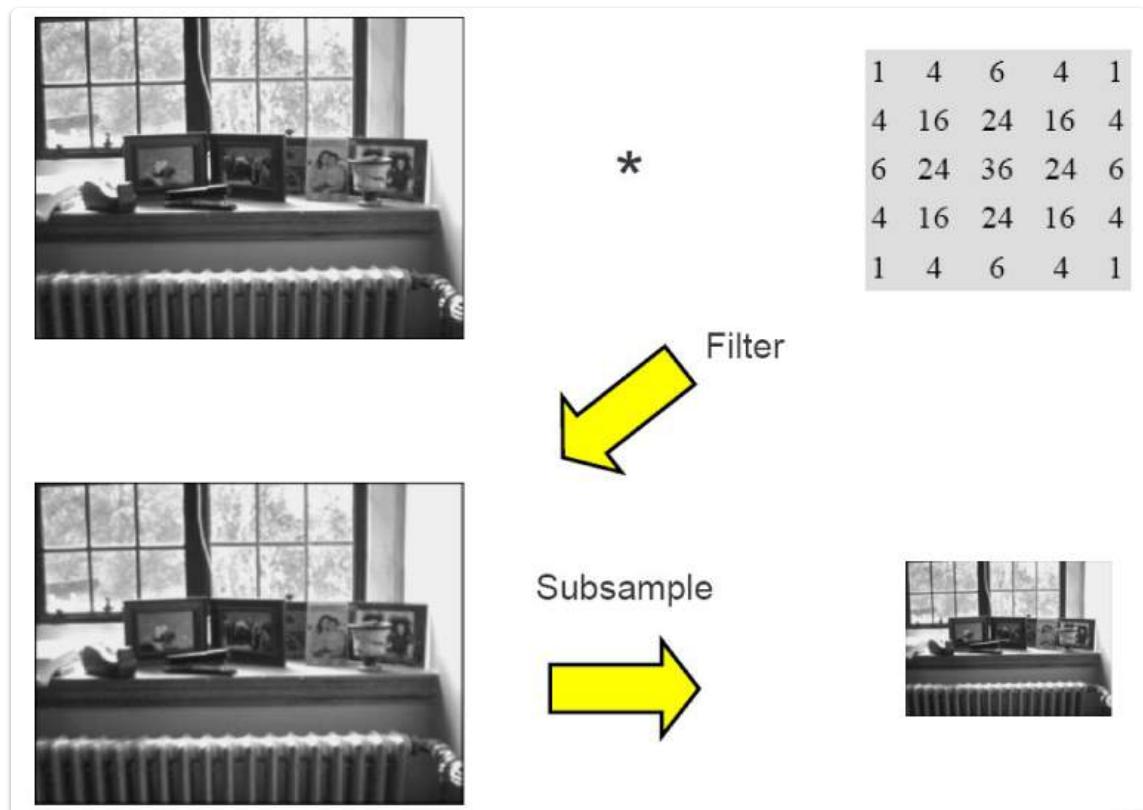
- **Vorteil:** Das Bild liegt in verschiedenen Auflösungen vor.
- **Analyse:** Sowohl grobe als auch feine Bildstrukturen können mittels kleiner lokaler Operatoren ausgewertet werden.
- **Äquivalenz:** Die Anwendung von Operatoren auf einer oberen Stufe der Pyramide ( $v > 0$ ) entspricht näherungsweise der Anwendung eines virtuell entsprechend vergrößerten

Operators auf das Originalbild ( $v = 0$ ).

- **Effiziente Detektion großer Strukturen:** Durch das Pyramidenkonzept gelangen grobe Strukturen sozusagen in Reichweite der lokalen Operatoren.
- **Reduzierter Aufwand:**
  - Betrachtet man die Detektionsaufgabe, so sinkt der Aufwand auf der Stufe  $v > 0$  der Pyramide auf  $O((\frac{N}{2^v})^2 \cdot (\frac{L}{2^v})^2)$ .
  - Die Abmessungen von Bild und Objekt verringern sich jeweils um den Faktor  $2^v$  auf  $\frac{N}{2^v} \times \frac{N}{2^v}$  bzw.  $\frac{L}{2^v} \times \frac{L}{2^v}$ .
  - Der Rechenaufwand verringert sich somit mit wachsendem  $v$  erheblich.
- **Kompromiss:** Die Detektion großer Objekte wird effizienter, allerdings auf Kosten des Verlusts feiner Details aufgrund der Tiefpassfilterung zwischen den Pyramidenstufen.

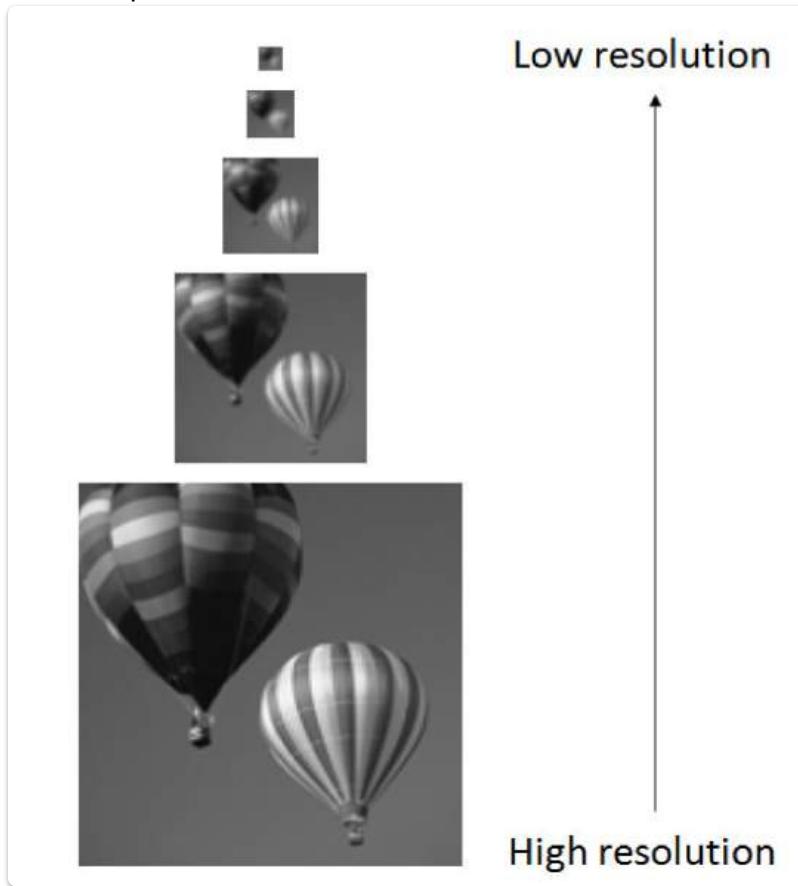


$$N^2 + \frac{1}{4}N^2 + \frac{1}{16}N^2 + \dots = N^2 + \frac{1}{3}N^2 = \frac{4}{3}N^2$$

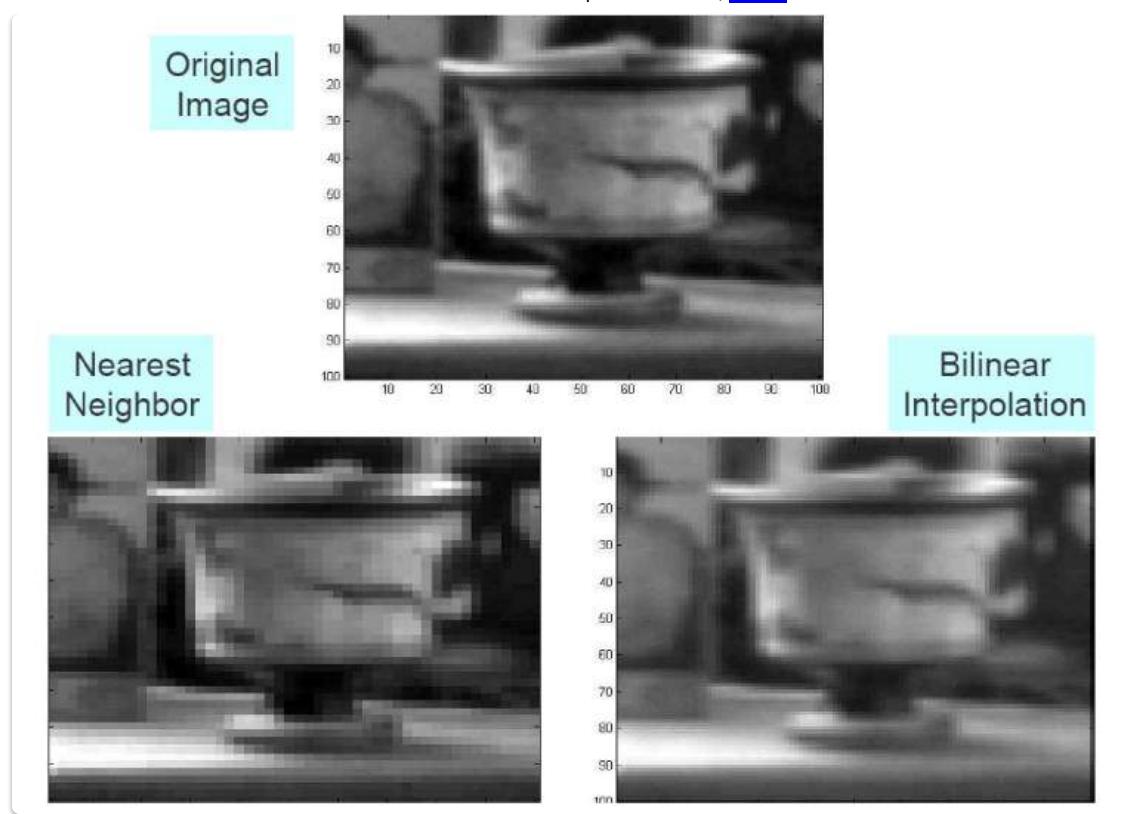


## Speicheraufwand und Berechnungsaufwand von Bildpyramiden

- **Specheraufwand:** Trotz der Speicherung von Bildinformationen auf mehreren Skalen bleibt der Specheraufwand für eine Bildpyramide überschaubar.
  - Das unterabgetastete Bild  $g'_{mn}$  hat die Größe  $\frac{N}{2} \times \frac{N}{2}$ .
  - Das zweite unterabgetastete Bild hat die Größe  $\frac{N}{4} \times \frac{N}{4}$  usw.
  - Der Gesamtzahl der Bildpixel beträgt  $\sum_{i=0}^K (\frac{N}{2^i})^2 = N^2 \sum_{i=0}^K (\frac{1}{4})^i < N^2 \cdot \frac{1}{1-1/4} = \frac{4}{3} N^2$ .
  - Der Gesamtspeicherbedarf ist also nur ca.  $\frac{4}{3}$  des Speicherbedarfs des Originalbildes.
- **Berechnungsaufwand:** Ebenso effektiv ist die Berechnung der Pyramide.
  - Derselbe Glättungsfilter wird auf jede Ebene der Pyramide angewandt.
  - Damit erfordert die Berechnung der gesamten Pyramide lediglich  $\frac{4}{3}$  der Rechenoperationen für ein zweidimensionales Bild.



- **Nachbarschaftsoperationen:** Wenn die Pyramide einmal berechnet ist, können wir Nachbarschaftsoperationen mit großen Skalen in den oberen Ebenen der Pyramide durchführen.

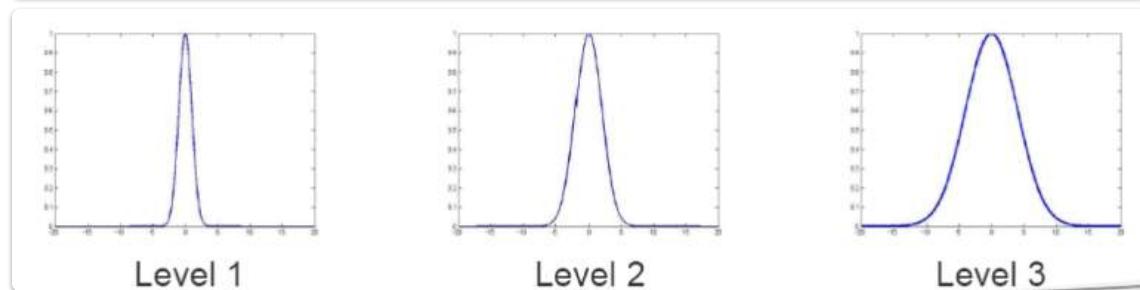
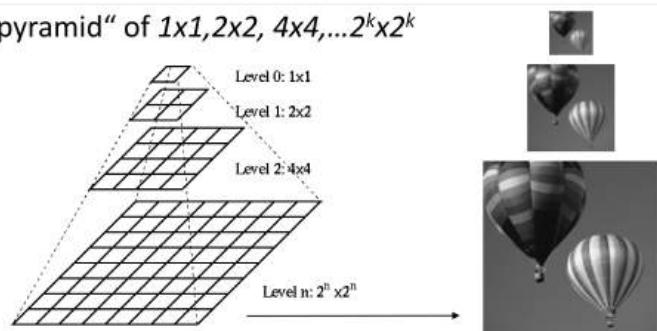


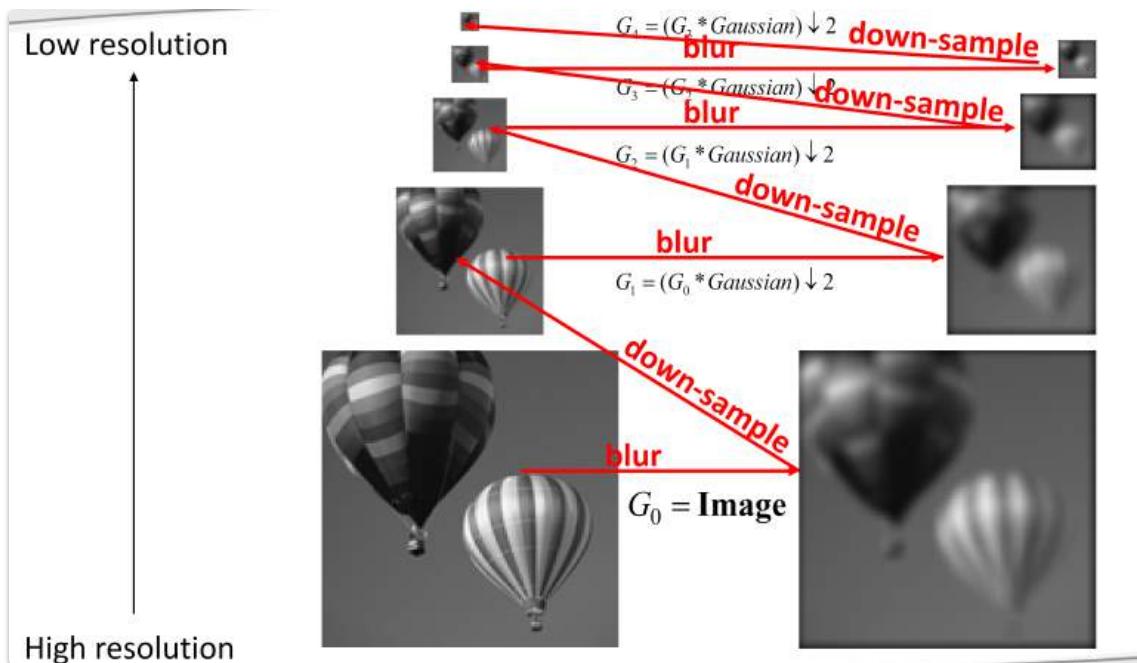
## Gaußpyramide

[EVC\\_Skriptum\\_CV](#), p.49, p.50

- **Definition:** Eine mit einem Tiefpass (Gaußfilter) konstruierte Bildpyramide heißt Gaußpyramide.

- **Idea:** Represent  $N \times N$  image as a „pyramid“ of  $1 \times 1, 2 \times 2, 4 \times 4, \dots 2^k \times 2^k$  images (assuming  $N = 2^k$ )





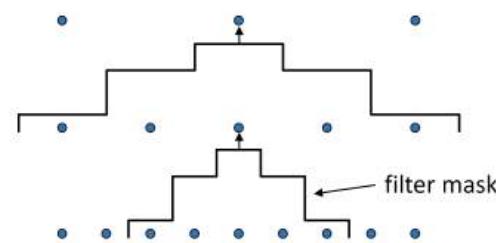
- **Erzeugung:**

- Tiefpassfilterung und Unterabtastung um den Faktor 2 an einem zweidimensionalen Signal  $g_m$ .
- Jeder abgeleitete Pixel wird jeweils aus fünf darunterliegenden Pixeln durch eine Gewichtung des  $1 \times 5$  Gaußfilters  $[1/16 \ 4/16 \ 3/8 \ 1/4 \ 1/16]$  erzeugt.
- Die kombinierte Glättung und Größenreduktion bei der Berechnung der  $(v+1)$ -ten Pyramidenebene aus der  $v$ -ten Ebene kann mit einem einzigen Operator ausgedrückt werden:

$$G^{(v+1)} = B \downarrow_2 G^{(v)}$$

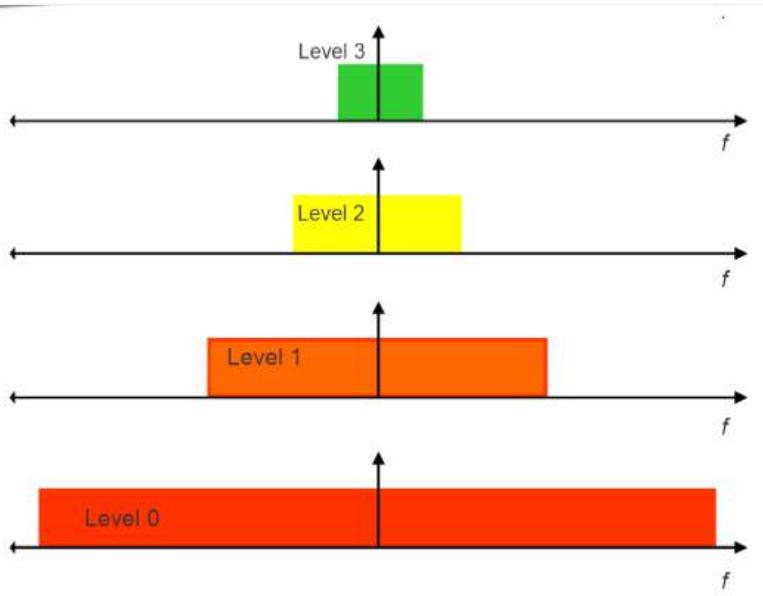
$\downarrow_2$  bezeichnet die Unterabtastung.

Die nullte Ebene ( $G^{(0)}$ ) ist das Originalbild.



- Repeat
  - Filter
  - Subsample
- Until minimum resolution reached
  - can specify desired number of levels (e.g., 3-level pyramid)
- The whole pyramid is only  $4/3$  the size of the original image!

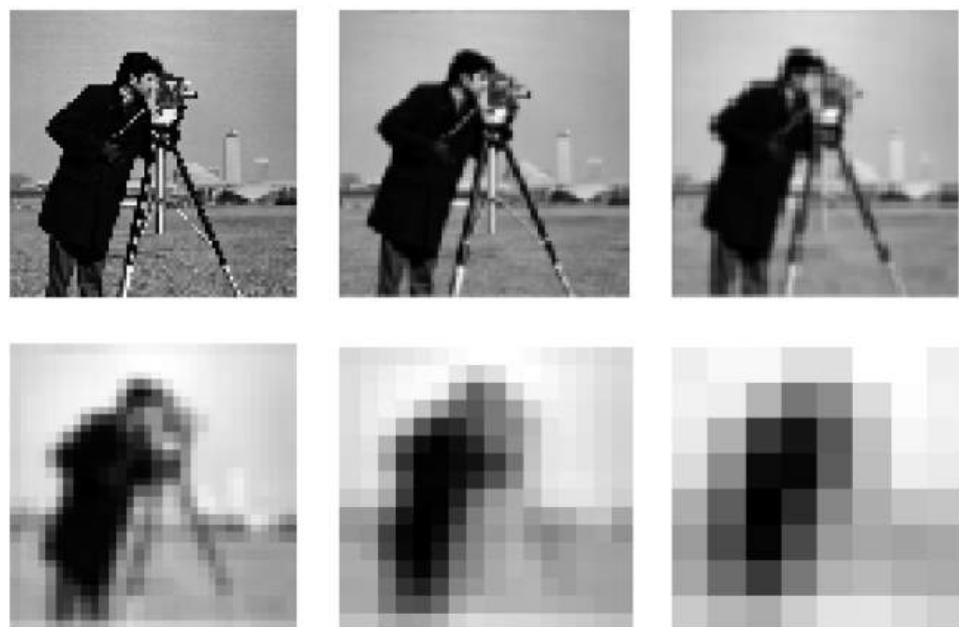
- **Struktur:** Die durch mehrmalige Anwendung dieses Operators erhaltene Bildserie wird als Gaußpyramide bezeichnet.
  - Von einer Ebene zur nächsten nimmt die Auflösung auf die Hälfte ab.
  - Man kann sich die Bildserie ebenfalls in Form einer Pyramide angeordnet vorstellen.



- **Eigenschaften:**

- Die Gaußpyramide stellt eine Serie von tiefpassgefilterten Bildern dar.
- Bei den oberen Ebenen nimmt die Grenzfrequenz von Ebene zu Ebene auf die Hälfte (eine Oktave) abnimmt.
- Damit verbleiben zunehmend gröbere Details im Bild.
- Um den gesamten Bereich der Frequenzen zu umspannen, sind nur wenige Pyramidenebenen erforderlich.
- Aus einem  $N \times N$ -Bild können wir eine Pyramide mit maximal  $\log_2(N) + 1$  Ebenen berechnen.
- Das kleinste Bild besteht nur aus einem Bildpunkt.

```
![[EVC_Skriptum_CV.pdf#page=50&rect=168,302,438,448|EVC_Skriptum_CV,  
p.49|500]]
```



- **Darstellung in Abbildung 42:** Jede Ebene wurde wieder auf die Originalauflösung skaliert, um Details an den oberen Ebenen zu zeigen.

## Anwendungsbeispiele

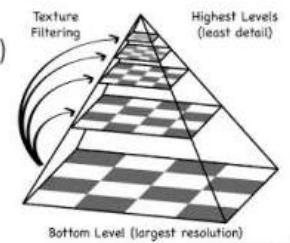
- **Improve Search**

- Search over translations: Classic coarse-to-fine strategy
- Search over scale: Template matching, e.g., find a face at different scales



- **Pre-computation**

- Need to access the image at different blur levels
- Useful for texture mapping at different resolutions (mip-mapping)



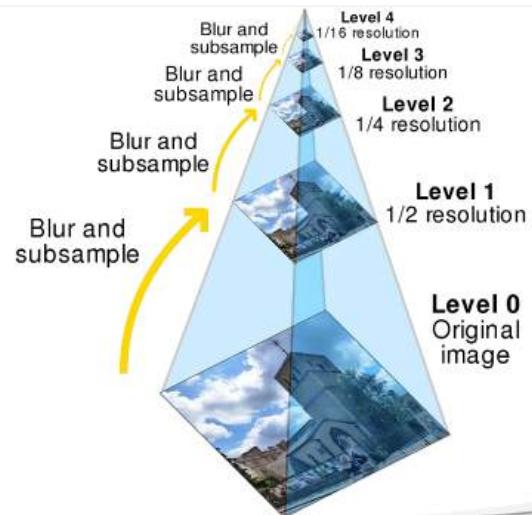
- **Image Processing**

- Editing frequency bands separately
- E.g., image blending...

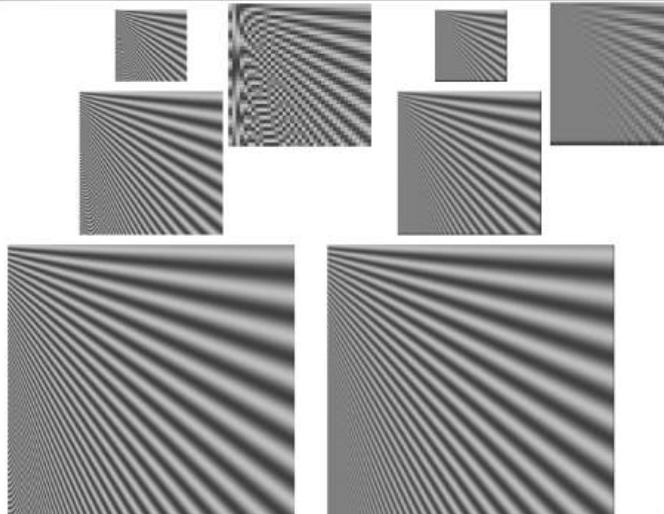
- **Up- or down-sampling images**

- **Multi-resolution image analysis**

- Look for an object over various spatial scales
- Coarse-to-fine image processing: from blur estimate or the motion analysis on a very low-resolution image, up-sample and repeat.
- Often a successful strategy for avoiding local minima in complicated estimation tasks.



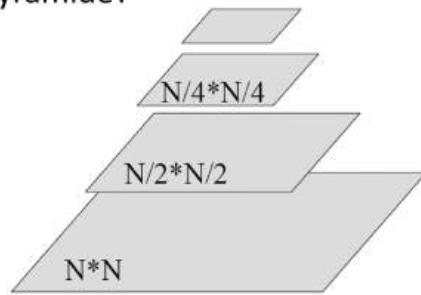
- A Gaussian pyramid (on the right) compared with a pyramid obtained by sampling without smoothing



## Testähnliches Beispiel

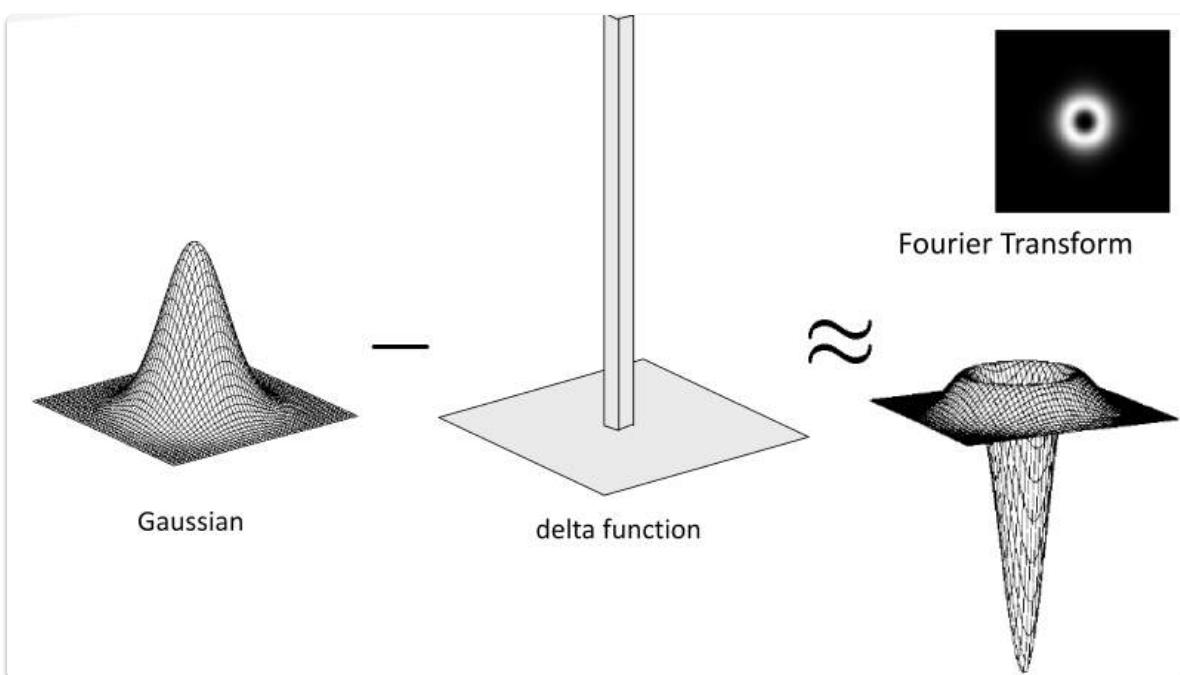
- Angenommen, ein Bild der 1. Ebene einer Gaußpyramide besteht aus 24.336 Pixeln ( $156 \times 156$ ), was der Originalauflösung des Bildes entspricht. Wie viele Pixel hat das Bild auf der 3. Ebene der Pyramide?

1. Ebene:  $156 \times 156$
2. Ebene:  $156/2 \times 156/2 = 78 \times 78$
3. Ebene:  $78/2 \times 78/2 = 39 \times 39 = \mathbf{1.521 \text{ Pixel}}$



## Laplacepyramide (Difference of Gaussians)

EVC\_Skriptum\_CV, p.50

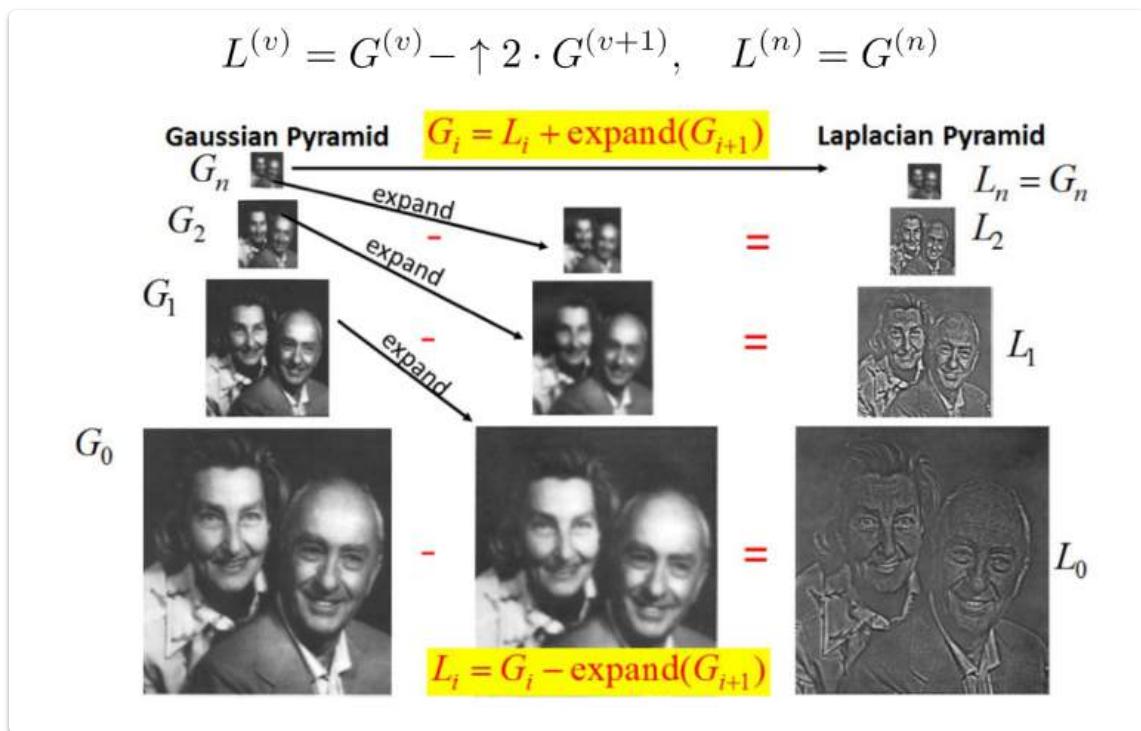


- **Konstruktion:**
  - Wird durch Differenzbildung mittels einer Tiefpasspyramide konstruiert.
  - In jedem Schritt werden Signalanteile mit hoher Ortsfrequenz herausgefiltert, sodass auf den oberen Stufen der Pyramide nur noch niedrige Ortsfrequenzen vorhanden sind.
- **Alternative Konstruktion:**
  - Ähnlich wie die Gaußpyramide konstruiert, aber jede Stufe ist das Ergebnis einer Bandpassfilterung des Originalbildes.
  - Eine einfache Möglichkeit zur Bandpassfilterung: Differenz zweier aufeinanderfolgender Bilder einer Gaußpyramide (Difference-of-Gaussians).
  - **Expansion:** Das Bild in der größeren Ebene wird zuerst expandiert.
    - **Expansionsoperator:**  $\uparrow_2$  durchgeführt.
    - **Reduzierung Glättungsoperator:** Grad der Glättung durch die Zahl nach dem  $\uparrow$ -Zeichen im Index angegeben.

- **Interpolation:** Der Expansion ist schwieriger als die Größenreduktion, da fehlende Information interpoliert werden muss.
- **Vergrößerung:** Um den Faktor zwei in allen Richtungen muss jeder zweite Bildpunkt in jeder Zeile interpoliert und dann jede zweite Zeile.
- **Erzeugung der  $v$ -ten Ebene der Laplacepyramide:**

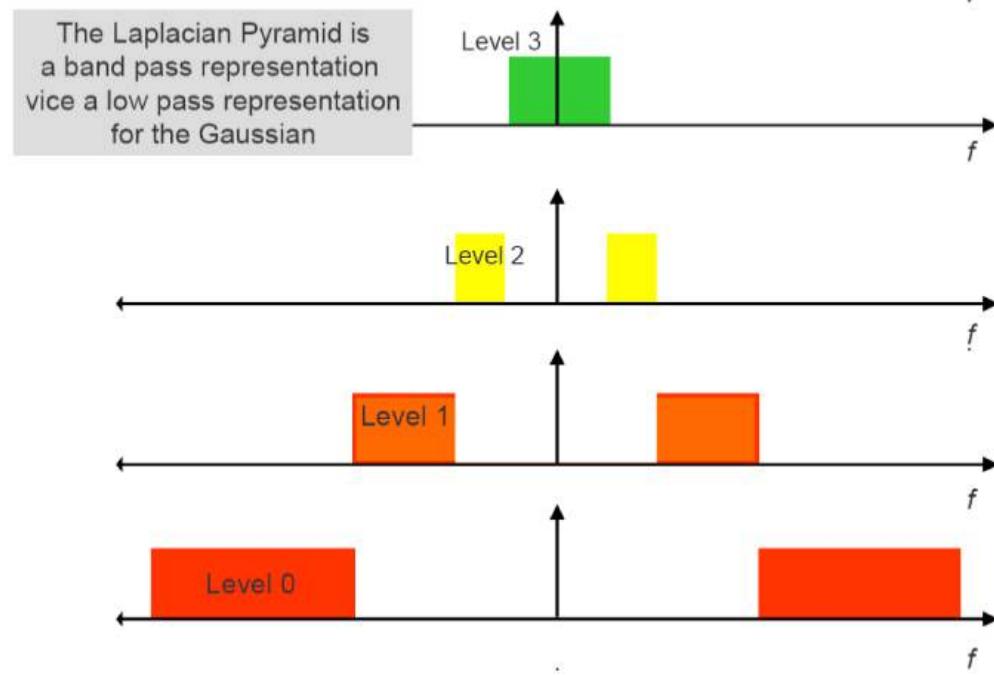
$$L^{(v)} = G^{(v)} - \uparrow_2 G^{(v+1)}, \quad L^{(n)} = G^{(n)}$$

- $G^{(v)}$ :  $v$ -te Ebene der Gaußpyramide.
- $\uparrow_2 G^{(v+1)}$ : Expandierte  $(v+1)$ -te Ebene der Gaußpyramide.
- $L^{(v)}$ :  $v$ -te Ebene der Laplacepyramide.
- Die oberste Ebene der Laplacepyramide ist die oberste (kleinste) Ebene der Gaußpyramide.



- **Eigenschaften der Laplacepyramide:**

- Stellt die Approximation der zweiten Ableitung des Originalbildes mit unterschiedlichen Glättungsparametern dar.
- Auf den ersten Stufen der Laplace-Pyramide werden feine Kantenstrukturen hervorgehoben.
- Während auf den höheren Stufen gröbere Kantenstrukturen des Originalbildes sichtbar sind.
- Die Laplacepyramide ist somit eine Zerlegung des Bildsignals in Bandpassbereiche mit logarithmischer Frequenzstaffelung.
- Das Originalbild kann aus der Laplace-Pyramide sowie dem obersten Bild der Gauß-Pyramide exakt rekonstruiert werden.
- Dies geschieht einfach durch eine Umkehrung des Konstruktionsschemas.



- **Vergleich zur Fouriertransformation:**

- Die Laplacepyramide ist lediglich zu einer groben Frequenzzerlegung ohne Richtungszerlegung.
- Alle Frequenzen innerhalb des Bereiches von ungefähr einer Oktave (Faktor 2) sind unabhängig von ihrer Richtung in einer Ebene der Pyramide enthalten.

## Anwendungen von Bildpyramiden

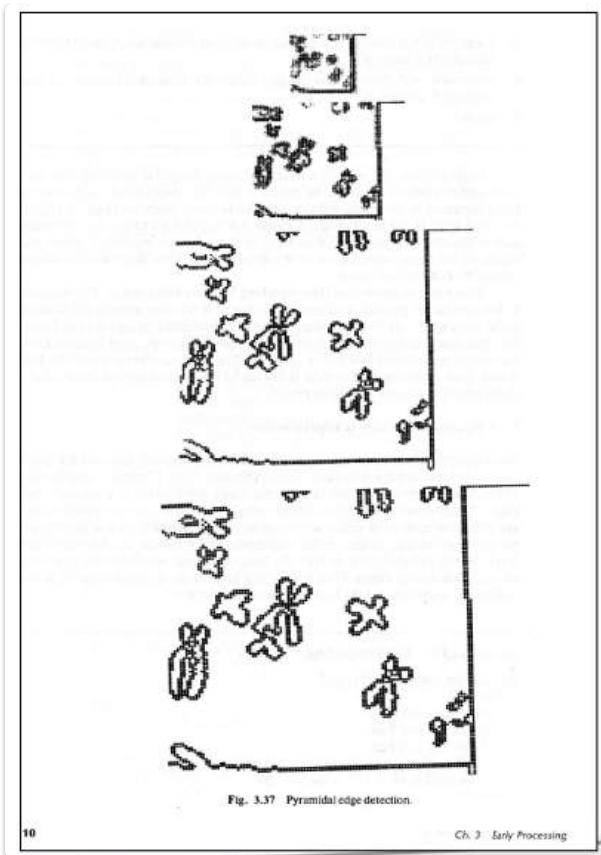
- **Coarse-to-Fine Strategien für Recheneffizienz:**

- **Suche nach Korrespondenzen:**
  - Suche auf groben Skalen, dann Verfeinerung mit feineren Skalen.
- **Kantenverfolgung (Edge Tracking):**
  - Eine "gute" Kante auf einer feinen Skala hat Eltern auf einer größeren Skala.
- **Kontrolle von Detail und Rechenaufwand beim Matching:**
  - Z.B. Finden von Streifen.
  - Sehr wichtig bei Texturerkennung.
- **Bildmischung und Mosaikbildung (Image Blending and Mosaicking).**
- **Datenkompression (Laplace-Pyramide).**

## Kantendetektion mit Pyramiden

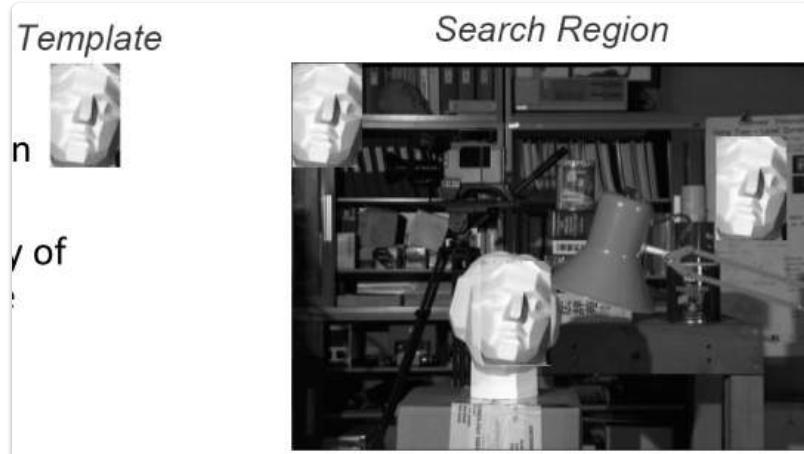
- **Coarse-to-Fine Strategie:**

- Kantendetektion auf einer höheren Ebene durchführen.
- Kanten feinerer Skalen nur in der Nähe der Kanten höherer Skalen berücksichtigen.



## Schnelles Template Matching

- Für ein  $m \times n$  Bild...
- Für ein  $p \times q$  Template...
- Die Komplexität der 2D Mustererkennungsaufgabe ist  $O(m \cdot n \cdot p \cdot q)$ .
- Dies wird noch schlimmer für eine Familie von Templates (z.B. um Skalierungs- und/oder Rotationseffekte zu berücksichtigen).

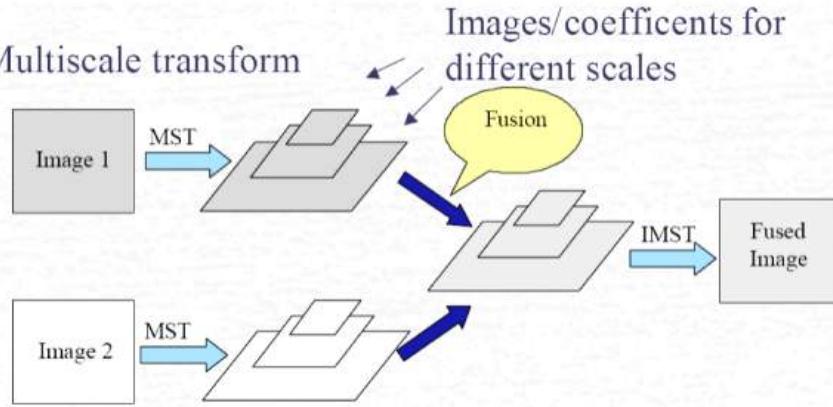


*Template**Search Region*

Original Image

**Image Fusion**

MST = Multiscale transform

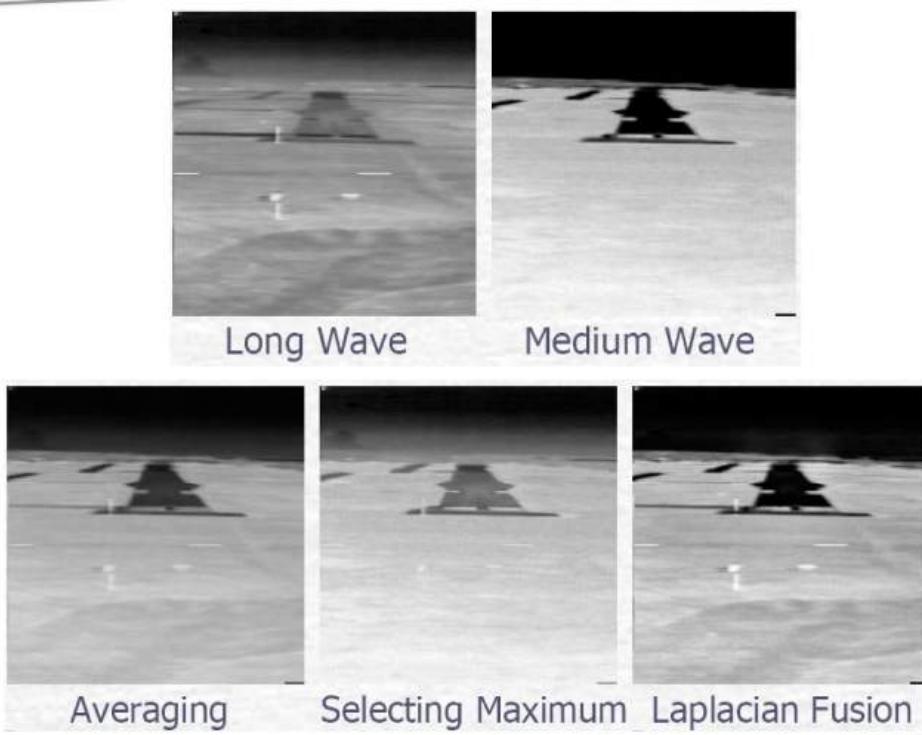


Multi-Scale Transform (MST)

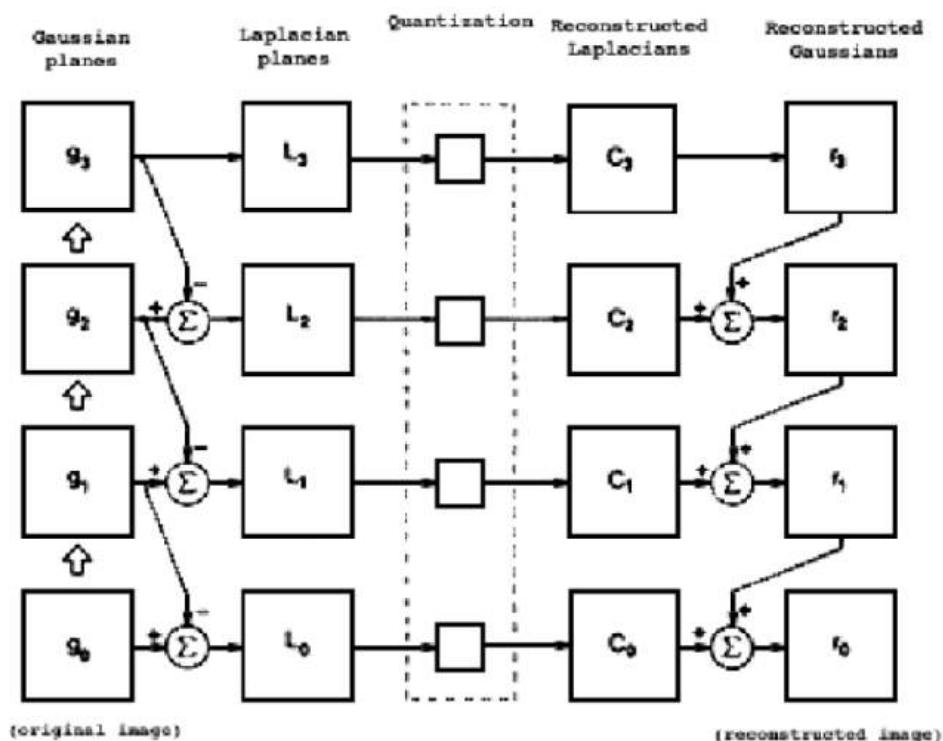
= Obtain Pyramid from Image

Inverse Multi-Scale Transform (IMST) = Obtain Image from Pyramid

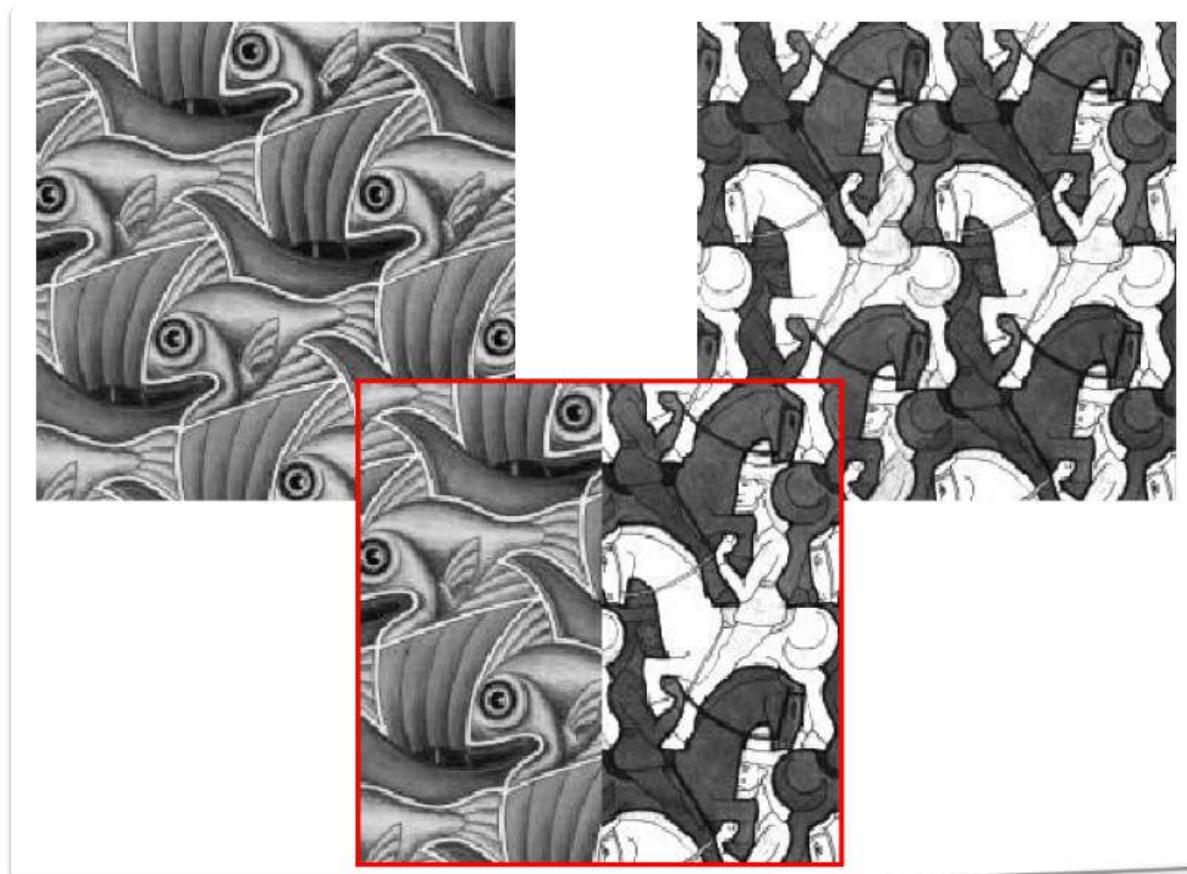
**Multi-Sensor Fusion**



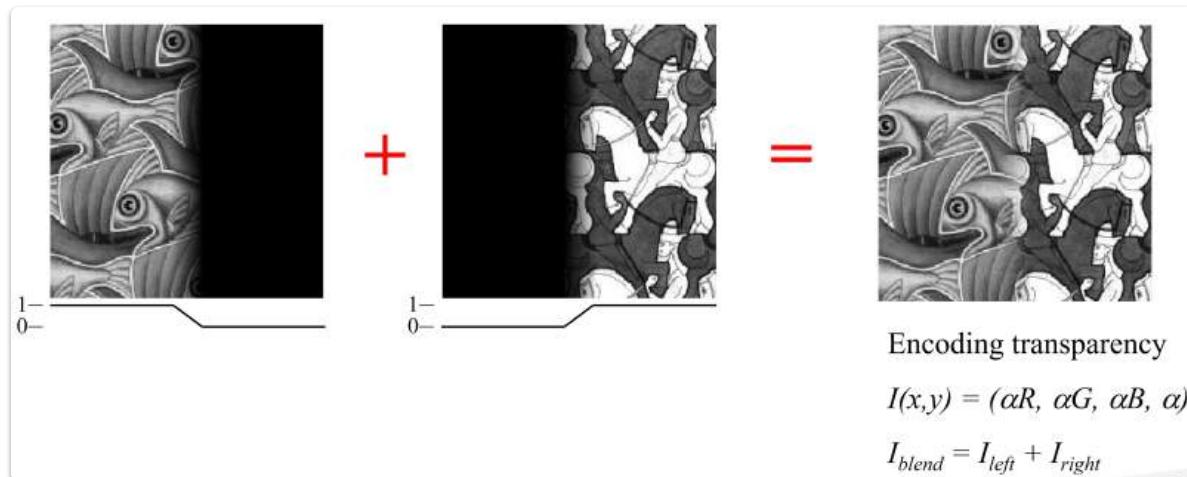
## Image Compression



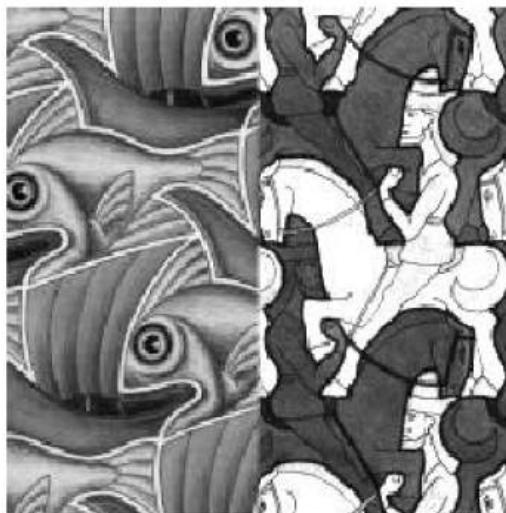
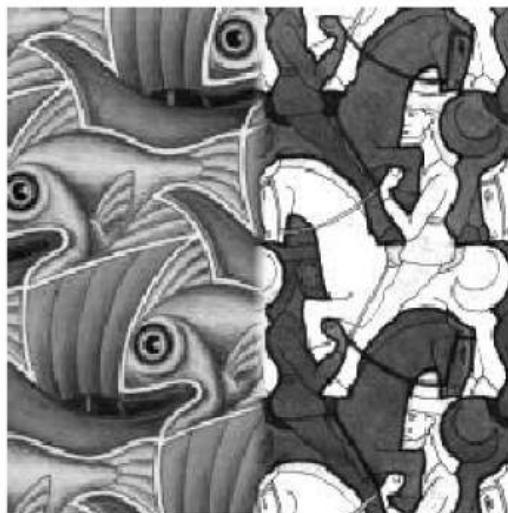
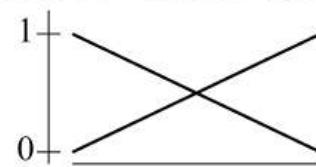
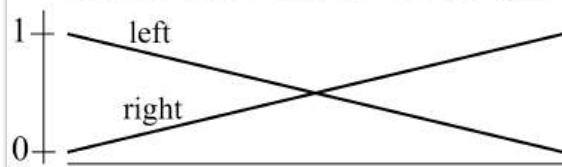
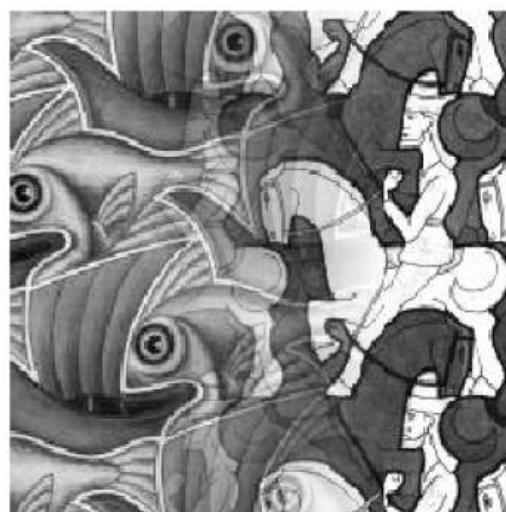
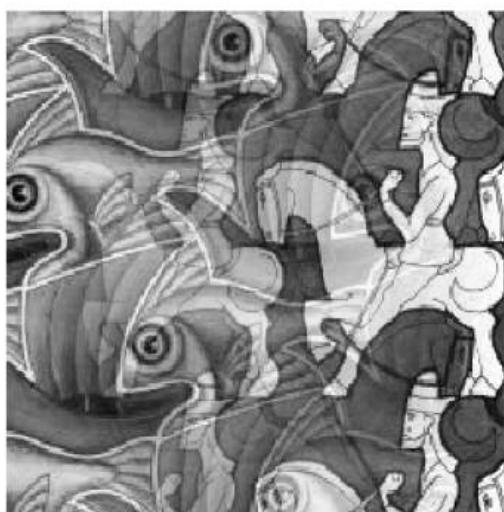
## Image Blending



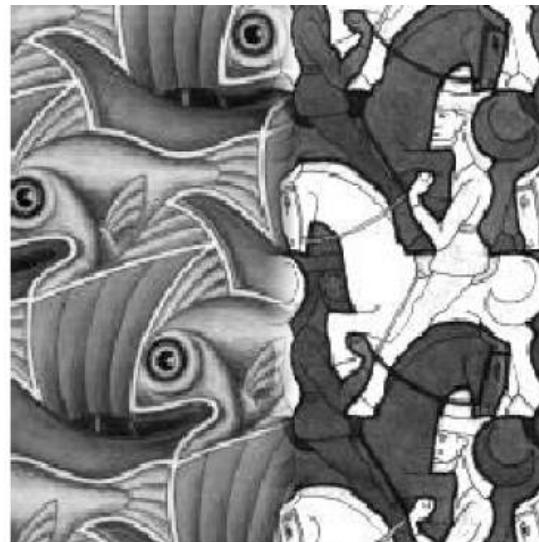
## Feathering



## Affect of Window Size



Gute Bildgröße:

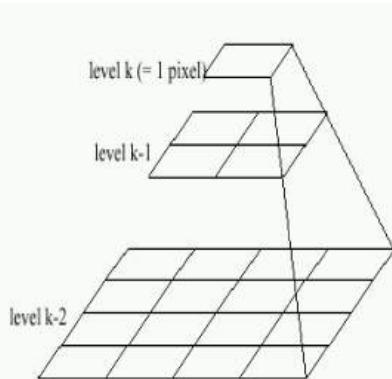


## “Optimal” Window: smooth but not ghosted

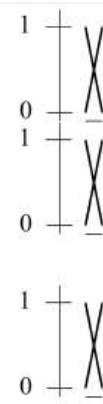
Was ist ein gutes Window?

- **Vermeidung von Nähten (Seams):**
  - Fenstergröße  $\geq$  Größe des größten prominenten Features.
- **Vermeidung von Geisterbildern (Ghosting):**
  - Fenstergröße  $\leq 2 \times$  Größe des kleinsten prominenten Features.

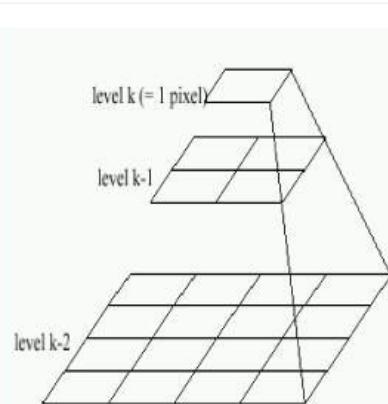
## Pyramid Blending



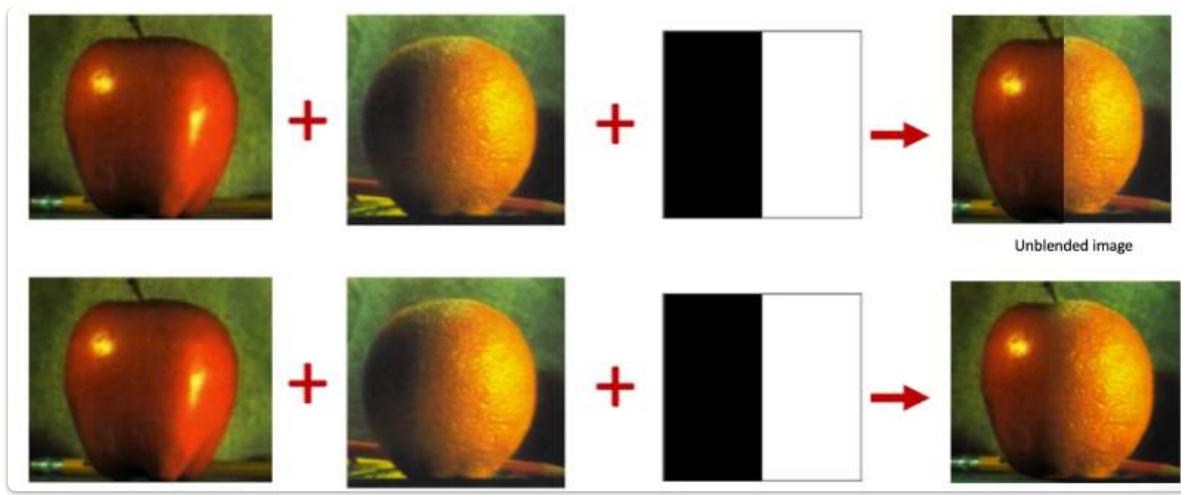
Left pyramid



blend



Right pyramid



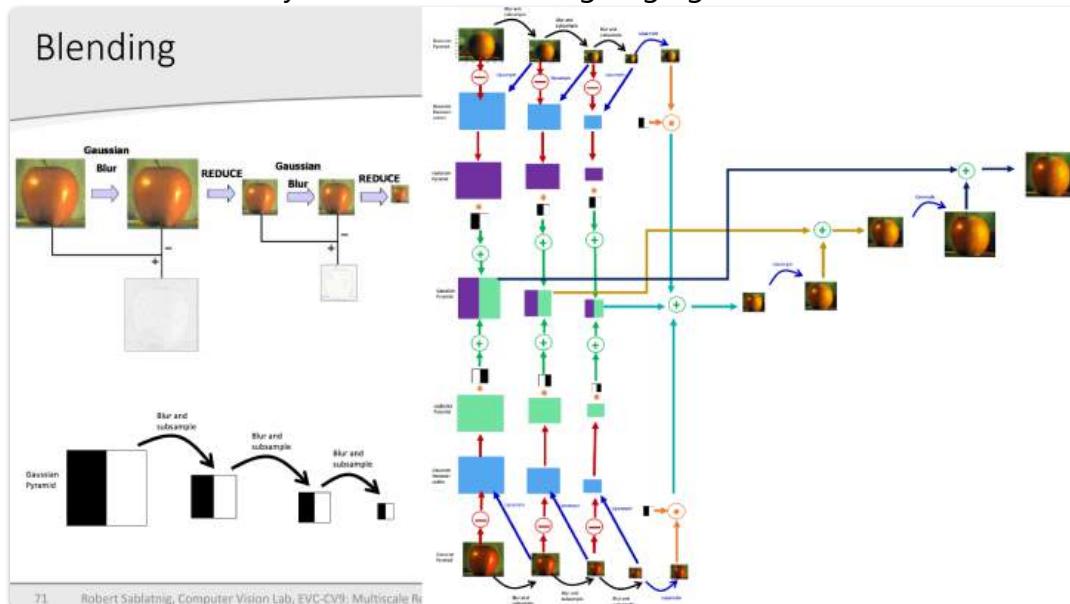
## Laplace-Pyramide: Bildmischung

- Genereller Ansatz:

- Erstelle Laplace-Pyramiden  $LA$  und  $LB$  von den Bildern  $A$  und  $B$ .
- Erstelle eine Gauß-Pyramide  $GR$  von der ausgewählten Region  $R$ .
- Bilde eine kombinierte Pyramide  $LS$  aus  $LA$  und  $LB$  unter Verwendung der Knoten von  $GR$  als Gewichte:

$$LS(i, j) = GR(i, j) \cdot LA(i, j) + (1 - GR(i, j)) \cdot LB(i, j)$$

- Kollabiere die  $LS$ -Pyramide, um das endgültige gemischte Bild zu erhalten.



## Blending Regions



waltuh

# 10. Ray-Tracing

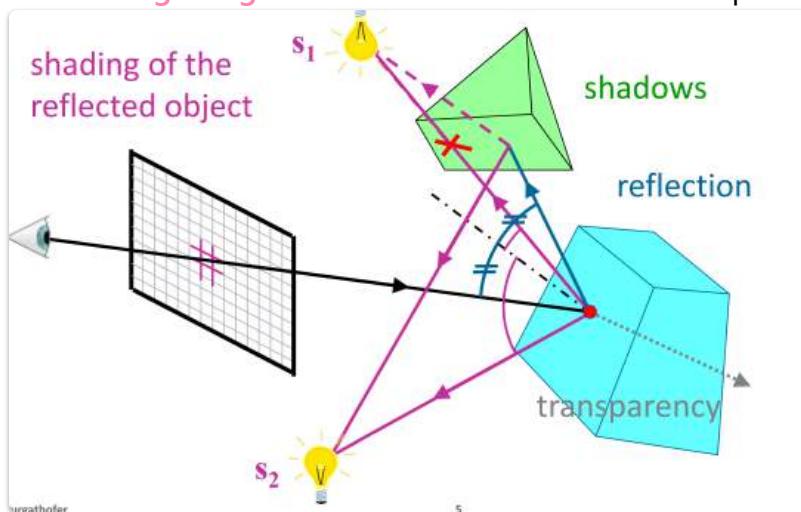
EVC\_Skriptum\_CG, p.39

- **Ray:** Strahl.
- **to trace:** Eine Spur verfolgen.
- **Ray-Tracing:** "Verfolgen von Strahlspuren".
- **Lichtstrahlen:** Durchlaufen in **verkehrter Richtung** (vom Auge zur Lichtquelle).
- **Aufbauend auf Ray-Casting:** Mächtige Methode zur Simulation wichtiger optischer Effekte:
  - Schattierung
  - Schatten
  - Spiegelbilder
  - Lichtbrechung
- **Einfachheit des Verfahrens:** Ermöglicht Darstellung komplexer Objekte:
  - Freiformflächen
  - Fraktale Oberflächen
  - Mathematische Funktionen aller Art
  - usw.

## Das Ray-Tracing Prinzip

EVC\_Skriptum\_CG, p.39

- **Basisidee:** Licht, das auf einen Bildpunkt trifft, in umgekehrter Richtung verfolgen.
- **Ziel:** Untersuchen, woher das Licht kommt.
- **Schlussfolgerung:** Daraus das Aussehen dieses Bildpunktes (Pixels) bestimmen.



## Korrekte Sichtbarkeit und Schattierung

## EVC\_Skriptum\_CG, p.39

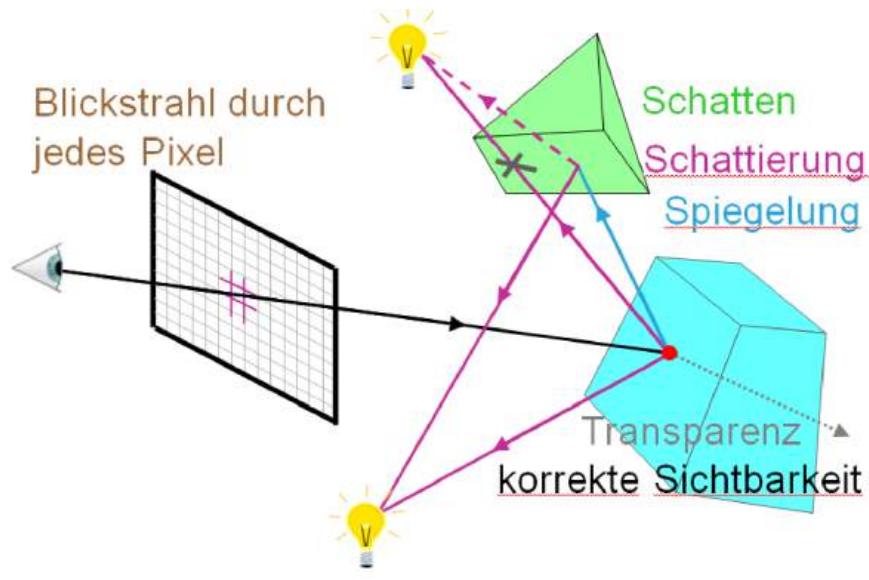
- **Blickstrahl (Primärstrahl):** Durch jeden Bildpunkt legen.
- **Schnitt:** Blickstrahl mit allen Oberflächen der Szene schneiden.
- **Nächster Schnittpunkt:** Denjenigen auswählen, der am nächsten zum Bild liegt.
- **Schattierung:** Schattierung dieses Objektpunktes (aus Blickrichtung) als Pixelwert.
- **Wiederholung:** Für alle Bildpunkte (Pixel).
- **Ergebnis:** Abbildung der Szene mit korrekter Sichtbarkeit.
- **Schattierungsmodell:** Beliebig wählbar (z.B. Phong-Modell).



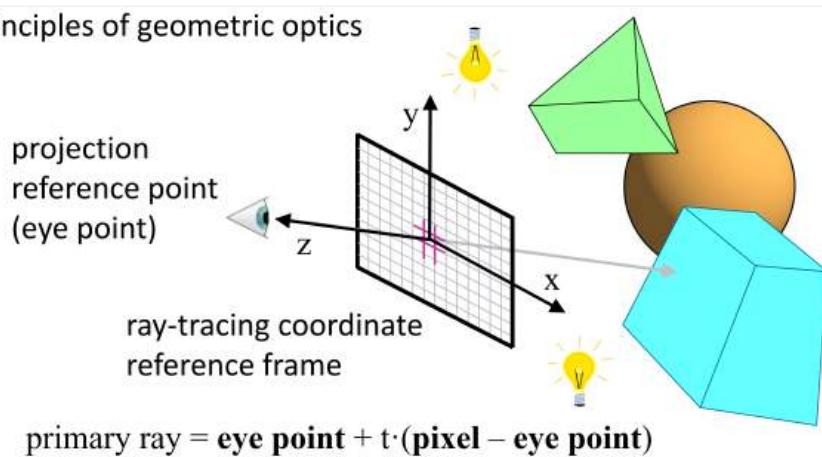
## Schatten

### EVC\_Skriptum\_CG, p.39

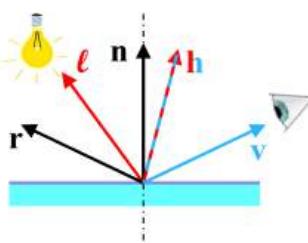
- **Schattierungsberechnung:** Benötigt Oberflächennormale und Richtungen zu allen Lichtquellen.
- **Direkter Lichteinfluss:** Nur wenn Lichteinfall nicht durch andere Objekte verdeckt ist.
- **Schattenfühler (Sekundärstrahl):** Vom zu schattierenden Punkt zur Lichtquelle legen.
- **Schnittprüfung:** Schattenfühler mit allen Objekten der Szene schneiden.
- **Schattenwurf:** Lichtquelle nicht berücksichtigen, wenn Schnittpunkt zwischen Objekt und Lichtquelle besteht.
- **Ergebnis:** Objektteile im Schatten erhalten weniger Lichteinfluss → automatischer Schattenwurf.



principles of geometric optics

ambient light  $k_a I_a$ shadow ray along  $\ell$ 

$$I_d = k_a I_a + k_d(\mathbf{n} \cdot \ell) + k_s(\mathbf{h} \cdot \mathbf{n})^p$$

diffuse reflection  $k_d(\mathbf{n} \cdot \ell)$ specular reflection  $k_s(\mathbf{h} \cdot \mathbf{n})^p$ 

## Spiegelbilder

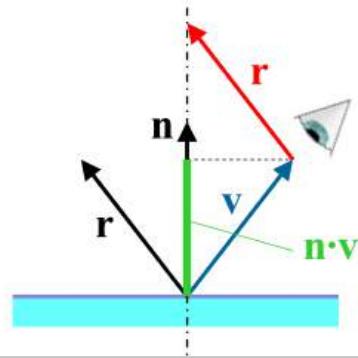
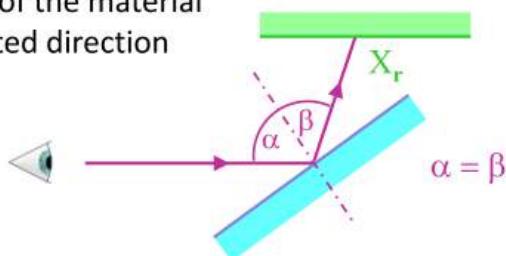
[EVC\\_Skriptum\\_CG](#), p.39

- **Spiegelndes Objekt getroffen:** Nicht das Objekt selbst sichtbar, sondern das, was in Spiegelungsrichtung sichtbar ist.
- **Reflexionsgesetz:** Einfallswinkel = Ausfallwinkel (Symmetrie).
- **Reflexionsstrahl (Sekundärstrahl):** Blickstrahl an der Oberfläche spiegeln und in Spiegelungsrichtung verfolgen.

- **Schnitt:** Reflexionsstrahl mit allen Objekten schneiden.
- **Nächster Schnittpunkt:** Auswählen.
- **Farbe/Schattierung:** Schattierung dieses weiteren Auftreffpunktes (aus Richtung des Reflexionsstrahls) ist die Farbe, die der ursprüngliche Blickstrahl sieht.
- **Lokale Berechnung:** Reflexionsverhalten wird lokal berechnet → einfache Erzeugung gekrümmter Spiegel.

$$I_r = k_r \cdot X_r$$

$I_r$  ... illumination caused by reflection  
 $k_r$  ... reflection coefficient of the material  
 $X_r$  ... shading in the reflected direction



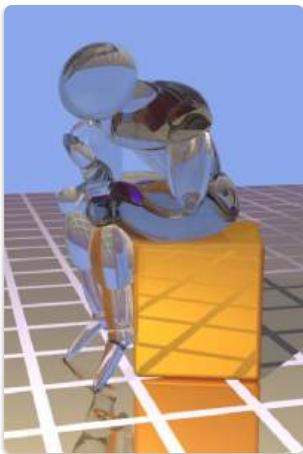
$$\mathbf{r} + \mathbf{v} = 2(\mathbf{n} \cdot \mathbf{v})\mathbf{n}$$

$$\mathbf{r} = 2(\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v}$$

## Transparenz

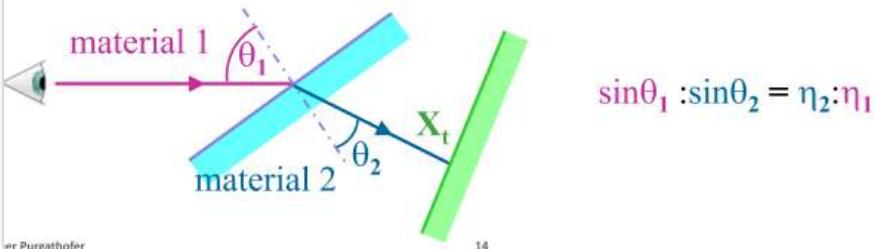
EVC\_Skriptum\_CG, p.39, p.40

- **Transparentes Objekt getroffen:** Man sieht, was der durch das Objekt verlaufende Transparenzstrahl trifft.
- **Transparenzstrahl (Sekundärstrahl):** Vom Auftreffpunkt durch das transparente Objekt verfolgen.
- **Brechungsgesetz:** Richtung des Transparenzstrahls so legen, dass das Material das Licht bricht.
- **Schnitt:** Transparenzstrahl mit allen Objekten schneiden.
- **Nächster Schnittpunkt:** Auswählen.
- **Farbe/Schattierung:** Schattierung dieses weiteren Auftreffpunktes (aus Richtung des Transparenzstrahls) ist, was der ursprüngliche Blickstrahl sieht.



$$I_t = k_t \cdot X_t$$

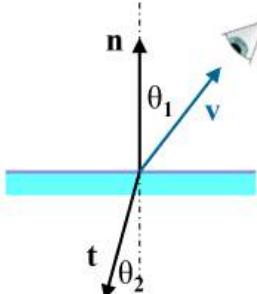
$I_t$  ... illumination caused by transparency  
 $k_t$  ... transparency coefficient of the material  
 $X_t$  ... shading in the transparency direction



calculation of transparency ray

$$\frac{\sin \theta_2}{\sin \theta_1} = \frac{n_1}{n_2} \quad \sin \theta_2 = \frac{n_1}{n_2} \sin \theta_1$$

$$t = -\frac{n_1}{n_2} v - (\cos \theta_2 - \frac{n_1}{n_2} \cos \theta_1) n$$



## Rekursion

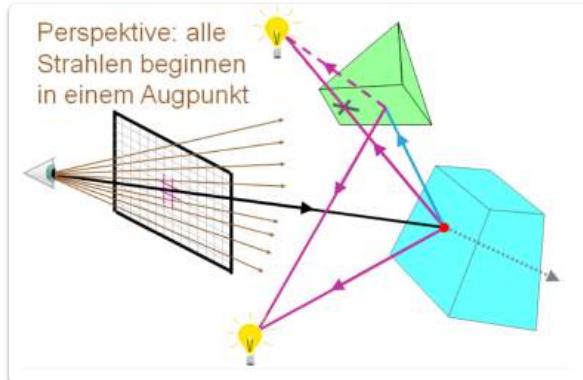
EVC\_Skriptum\_CG, p.40

- **Gleichwertigkeit der Strahlen:** Jeder Strahl (außer Schattenfühler) ist gleichwertig (Primär- oder Sekundärstrahl).
- **Aktion am Auftreffpunkt:** Unabhängig davon, ob Primär- oder Sekundärstrahl.
- **Ermöglicht:**
  - Mehrfachspiegelungen.
  - Spiegelungen hinter transparentem Material.
  - Usw.

## Perspektive

EVC\_Skriptum\_CG, p.40

- **Erzeugung primärer Blickstrahlen:** Bestimmt die Abbildung der Szene auf die Bildebene.
- **Parallele Strahlen (normal zur Bildebene):** Orthogonale Parallelprojektion.
- **Strahlen von fiktivem Augpunkt:** Perspektivische Projektion (natürliche Entstehung ohne Mehraufwand).



## Ray-Tracing Implementierung

[EVC\\_Skriptum\\_CG](#), p.40

Einen Ray-Tracer zu schreiben ist also ganz einfach. Man braucht eine Funktion, die eine Gerade mit allen Objekten schneidet und den vordersten Schnittpunkt zurückliefert.

Ray-Tracing Pseudocode:

```

FOR alle Pixel  $p_0$  DO
  1. lege Blickstrahl vom Auge  $e$  aus durch  $p_0$ ,
     schneide mit allen Objekten und wähle den nähesten Schnittpunkt  $p$ 
  2. FOR alle Lichtquellen  $s$  DO
      schneide Schattenführer  $p \rightarrow s$  mit allen Objekten
      IF kein Schnittpunkt zwischen  $p$ ,  $s$  THEN Schattierung += Einfluss von  $s$ 
  3. IF Oberfläche von  $p$  ist spiegelnd
      THEN verfolge Sekundärstrahl; Schattierung += Einfluss der Reflexion
  4. IF Oberfläche von  $p$  ist transparent
      THEN verfolge Sekundärstrahl; Schattierung += Einfluss der Transparenz
    
```

## Viewing-Koordinatensystem und Strahldarstellung

- **Viewing-Koordinatensystem (Standard):**
  - xy-Ebene = Bildebene.
  - Hauptblickrichtung = negative z-Achse.
- **Strahlen (parametrisierte Form):**

$$p(t) = p_0 + t \cdot d$$

- $p_0$ : Startpunkt.
- $t$ : Parameter.
- $d$ : Richtungsvektor.
- **Primärstrahlen:**

- $e$ : Augpunkt.
- $p_0$ : Pixelkoordinate.
- Schattenfühler:

$$p + t \cdot (s - p)$$

- $p$ : Oberflächenpunkt.
- $s$ : Lichtquellenposition.
- Reflexionsstrahlen:

$$p + t \cdot r$$

- $r$ : Reflexionsrichtung des Blickstrahls  $v$ .

$$r = (2n \cdot v)n - v \text{ (aus Reflexionsgesetz)}$$

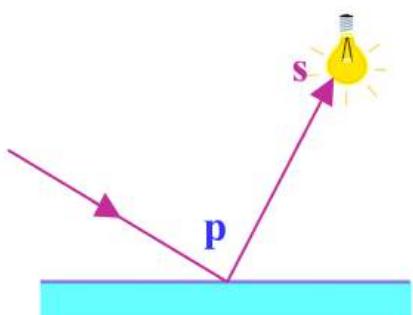
- $n$ : Oberflächennormale.
- $v$ : Richtung des einfallenden Strahls.
- $|r| = 1$  (garantiert durch Berechnung).

ray = intersection point +  $t \cdot$  vector to light source

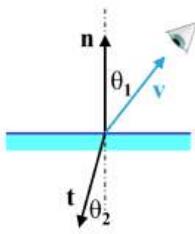
$$\text{ray} = \mathbf{p} + t \cdot (\mathbf{s} - \mathbf{p})$$

**p** ... intersection point

**s** ... light source position



a light source influences the result only if  
there is no intersection with  $0 < t < 1$

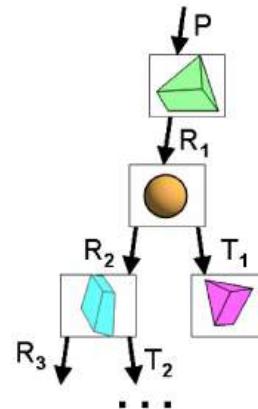
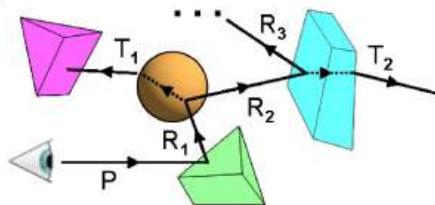


**Transparenzstrahlen** sind  $p + t \cdot t$ , wobei  $t$  sich aus dem Snellius'schen Brechungsgesetz  $\sin \theta_1 : \sin \theta_2 = \eta_2 : \eta_1$  berechnet ( $\eta_i$  ist der Brechungsindex des Materials  $i$ ):

$$t = -\frac{\eta_1}{\eta_2}v - (\cos \theta_2 - \frac{\eta_1}{\eta_2} \cos \theta_1)n$$

Auch der Vektor  $t$  hat wieder die Länge 1 (siehe linke Skizze).

Wenn man nun solcherart die Lichtstrahlen in verkehrter Richtung „verfolgt“, so entsteht eine **rekursive Aufruffolge** (siehe Skizze unten), die einem Strahlenbaum entspricht (rechte Skizze). Normalerweise wird dieser Baum aber nicht in dieser Form gespeichert, sondern ist nur eine symbolische Darstellung der Rekursionsaufruffolge.



## Schnitte zwischen Strahlen und Objekten (Ray-Tracing)

[EVC\\_Skriptum\\_CG, p.41](#)

- **Bedingungen für darzustellende Objekte:**
  - Schnittpunkt mit Gerade muss berechenbar sein.
  - Oberflächennormale am Schnittpunkt muss bekannt sein.
  - Materialeigenschaften am Schnittpunkt müssen vorhanden sein.
- **Erfüllung der Bedingungen:**
  - BReps (Boundary Representations): Einfach.
  - CSG-Bäume (Constructive Solid Geometry): Durch rekursive Evaluation.
  - Viele andere Datenformate (z.B. Freiformflächen).
- **Notwendigkeit:** Funktionen zur Schnittberechnung mit Strahl für jede Primitivart.
- **Beispiele (folgen):**
  - Kugel
  - Polygon

## Schnitt Strahl-Kugel

[EVC\\_Skriptum\\_CG, p.41](#)

**Kugelgleichung:**  $|p - c|^2 - R^2 = 0$

In diese setzt man den Strahl ein:  $|(e + td) - c|^2 - R^2 = 0$

Dann wird zur besseren Lesbarkeit  $\Delta p$  eingeführt:  $\Delta p = c - e$

Und man erhält eine quadratische Gleichung in  $t$ :  $t^2 - 2(d \cdot \Delta p)t + (|\Delta p|^2 - R^2) = 0$

Die 2 Lösungen entsprechen den beiden Schnittpunkten mit der Kugel:

$$t = d \cdot \Delta p \pm \sqrt{(d \cdot \Delta p)^2 - |\Delta p|^2 + R^2}$$

In Fällen, wo  $R^2 \ll |\Delta p|^2$  ist (das ist durchaus häufig), entstehen in dieser Formel Rundungsfehler. Um diese zu vermeiden, kann man  $d^2 = 1$  ausnutzen und die Formel umformen, so dass Rundungsfehler unwahrscheinlicher werden:

$$t = d \cdot \Delta p \pm \sqrt{|\Delta p|^2 - (d \cdot \Delta p)^2 - R^2}$$

ray equation:

$$\mathbf{p}(t) = \mathbf{p}_0 + t \cdot \mathbf{d}$$

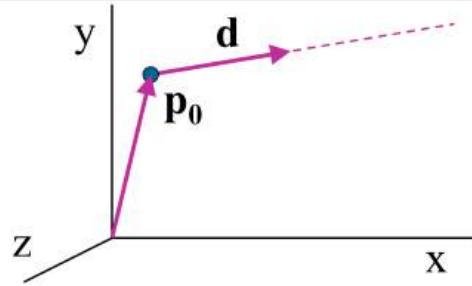
for primary rays:

$$\mathbf{d} = \frac{\mathbf{p}_0 - \mathbf{e}}{|\mathbf{p}_0 - \mathbf{e}|}$$

for secondary rays:

$$\mathbf{d} = \mathbf{r}$$

$$\mathbf{d} = \mathbf{t}$$



$\mathbf{p}_0$  ... initial-position vector  
 $\mathbf{d}$  ... unit direction vector  
 $\mathbf{e}$  ... eye-point vector  
 $\mathbf{r}$  ... reflection vector  
 $\mathbf{t}$  ... transparency vector

parametric ray equation  
 inserted into sphere equation

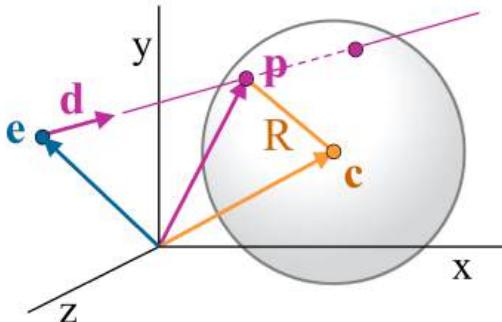
$$|\mathbf{p} - \mathbf{c}|^2 - R^2 = 0$$

$$|(\mathbf{e} + t\mathbf{d}) - \mathbf{c}|^2 - R^2 = 0$$

$$\Delta \mathbf{p} = \mathbf{c} - \mathbf{e}$$

$$t^2 - 2(\mathbf{d} \cdot \Delta \mathbf{p})t + (|\Delta \mathbf{p}|^2 - R^2) = 0 \quad (\mathbf{d}^2 = 1)$$

$$t = \mathbf{d} \cdot \Delta \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \Delta \mathbf{p})^2 - |\Delta \mathbf{p}|^2 + R^2}$$



if discriminant is negative  $\Rightarrow$  no intersections

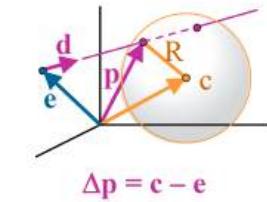
$$t = \mathbf{d} \cdot \Delta \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \Delta \mathbf{p})^2 - |\Delta \mathbf{p}|^2 + R^2}$$

$$\Rightarrow t = \mathbf{d} \cdot \Delta \mathbf{p} \pm \sqrt{R^2 - |\Delta \mathbf{p} - (\mathbf{d} \cdot \Delta \mathbf{p})\mathbf{d}|^2}$$

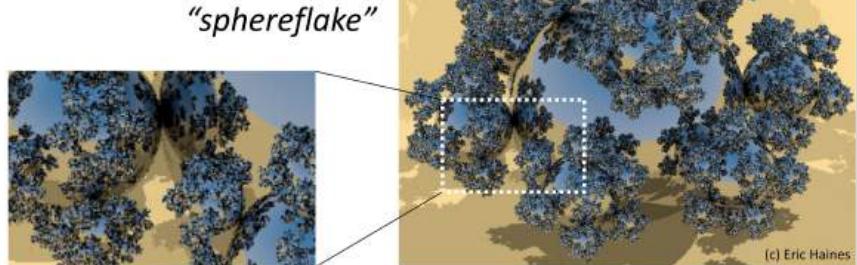
$$|\Delta \mathbf{p} - (\mathbf{d} \cdot \Delta \mathbf{p})\mathbf{d}|^2 = |\Delta \mathbf{p}|^2 - 2\mathbf{d}^2|\Delta \mathbf{p}|^2 + (\mathbf{d} \cdot \Delta \mathbf{p})^2\mathbf{d}^2$$

$$\mathbf{d}^2 = 1 \quad \mathbf{d}^2 = 1$$

(to avoid roundoff errors  
when  $R^2 \ll |\Delta \mathbf{p}|^2$ )

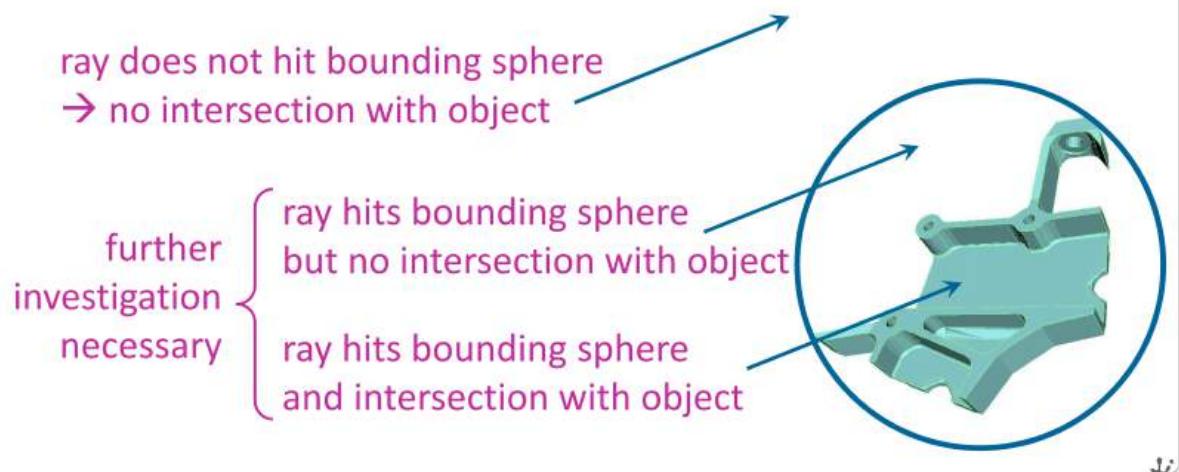


Werner Purgathofer



(c) Eric Haines

use **bounding sphere** to eliminate easy cases



## Schnitt Strahl-Polygon

[EVC\\_Skriptum\\_CG](#), p.41, p.42

- **Ziel:** Schnittpunkt eines Strahls mit einem Polygon bestimmen.
- **Schritt 1: Backface Detection**
  - Überprüfen, ob das Polygon in die richtige Richtung schaut. (siehe Backface Detection)
- **Schritt 2: Strahlengleichung**
  - Strahl definiert als:  $\mathbf{p} = \mathbf{p}_0 + t \cdot \mathbf{d}$ 
    - $\mathbf{p}$ : Punkt auf dem Strahl
    - $\mathbf{p}_0$ : Ursprung des Strahls
    - $\mathbf{d}$ : Richtung des Strahls
    - $t$ : Parameter entlang des Strahls
- **Schritt 3: Ebenengleichung des Polygons**
  - Form:  $Ax + By + Cz + D = 0$

- Kann auch in Normalenform geschrieben werden:  $\mathbf{n} \cdot \mathbf{p} = -D$ 
  - $\mathbf{n} = (A, B, C)$ : Normalenvektor der Ebene
- **Schritt 4: Schnittpunkt mit der Ebene berechnen**
  - Setze die Strahlengleichung in die Ebenengleichung ein:
    - $\mathbf{n} \cdot (\mathbf{p}_0 + t \cdot \mathbf{d}) = -D$
    - $\mathbf{n} \cdot \mathbf{p}_0 + t(\mathbf{n} \cdot \mathbf{d}) = -D$
  - Löse nach  $t$  auf:
    - $t(\mathbf{n} \cdot \mathbf{d}) = -(D + \mathbf{n} \cdot \mathbf{p}_0)$
    - $t = -\frac{D + \mathbf{n} \cdot \mathbf{p}_0}{\mathbf{n} \cdot \mathbf{d}}$
- **Schritt 5: Überprüfung des Schnittpunkts**
  - Liegt der Schnittpunkt innerhalb des Polygons?
  - Oder neben dem Polygon?
  - Kann durch Projektion auf eine Hauptebene in 2D überprüft werden.

1. use bounding sphere to eliminate easy cases

2. locate front faces       $\mathbf{d} \cdot \mathbf{n} < 0$

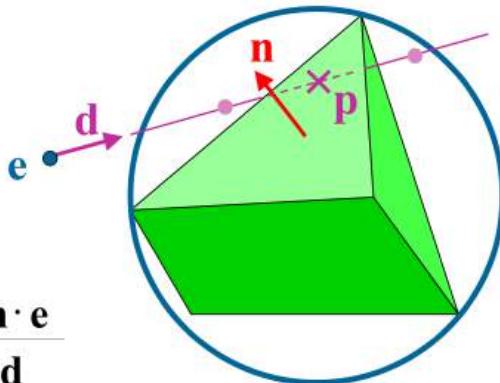
3. solve plane equation

$$Ax + By + Cz + D = 0$$

$$\mathbf{n} = (A, B, C)$$

$$\mathbf{n} \cdot \mathbf{p} = -D$$

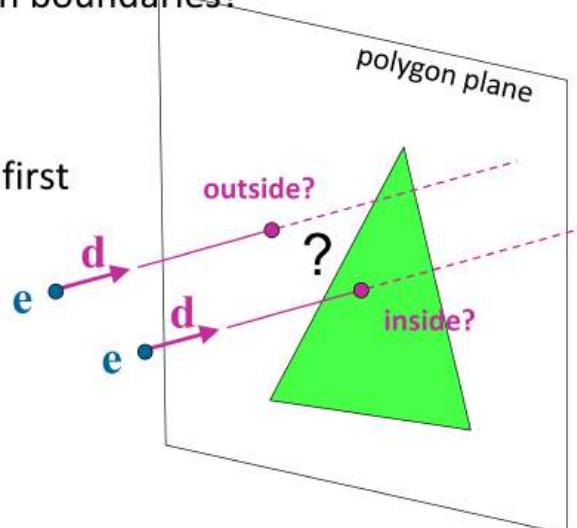
$$\mathbf{n} \cdot (\mathbf{e} + t\mathbf{d}) = -D \quad \Rightarrow \quad t = -\frac{D + \mathbf{n} \cdot \mathbf{e}}{\mathbf{n} \cdot \mathbf{d}}$$



4. intersection point inside polygon boundaries?

perform inside-outside test

→ smallest  $t$  to an inside point is first intersection point of polyhedron



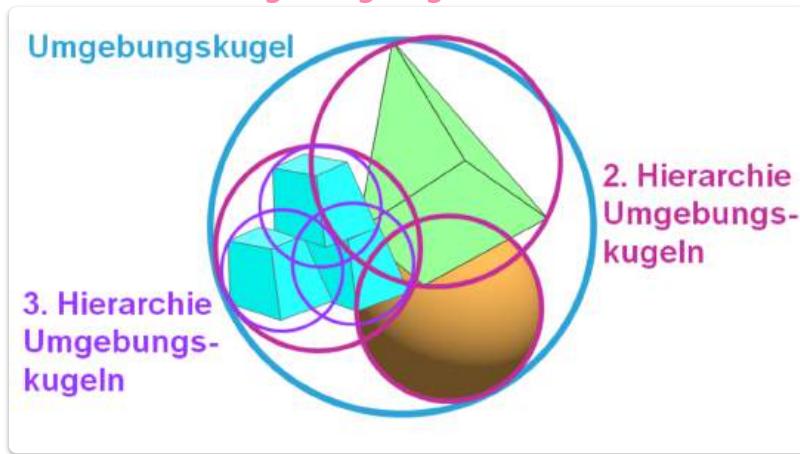
## Ray-Tracing Beschleunigung

- **Problem:** Ray-Tracing ist rechenintensiv.
  - Beispiel: Szene mit 1000 Polygonen auf 1000x1000 Pixel Fläche  $\Rightarrow 10^9$  Schnittberechnungen (nur für Primärstrahlen, ohne Optimierungen).
- **Notwendigkeit:** Signifikante Beschleunigung des Verfahrens ist erforderlich.
- **Wichtigste Methode:** Reduktion der Anzahl notwendiger Schnittberechnungen.
  - Ansatz: Ausnutzung von Kohärenz.

## Objektumgebungen

EVC\_Skriptum\_CG, p.42

- **Problem:** Direkter Schnitt komplexer Objekte mit Strahlen ist ineffizient.
- **Idee:** Umgebungen (Bounding Volumes) für Objekte definieren, um schnelle Vorabtests durchzuführen.
- **Umgebungskugeln (Bounding Spheres)**
  - Einfache geometrische Form (Kugel).
  - Umschließt das gesamte Objekt.
  - **Vorteil:** Schnelle Schnittprüfung mit dem Strahl.
  - **Funktionsweise:**
    - Trifft der Strahl die Umgebungskugel?
      - **Nein:** Dann trifft er auch nicht das eingeschlossene Objekt  $\Rightarrow$  keine detaillierte Schnittberechnung notwendig (Performancegewinn).
      - **Ja:** Detaillierte Schnittprüfung mit dem Objekt erforderlich.
- **Hierarchische Umgebungskugeln**



- Komplexe Objekte können hierarchisch in kleinere Teillojekte mit eigenen Umgebungskugeln unterteilt werden.
- **Vorteil:** Verfeinerte Tests ermöglichen früheres Ausschließen von irrelevanten Teilen der Szene.
- **Prinzip:**
  1. Große Umgebungskugel für das gesamte Objekt.
  2. Unterteilung in kleinere Gruppen mit eigenen Umgebungskugeln (2. Hierarchie).
  3. Weiter Unterteilung bis zu einfachen Objekten (3. Hierarchie).

- **Komplexität:**

- Ohne Umgebungskugeln:  $O(n)$  Schnittversuche (wobei  $n$  die Anzahl der Objekte/Teile ist).
- Mit hierarchischen Umgebungskugeln: Reduktion auf etwa  $O(\log n)$ .

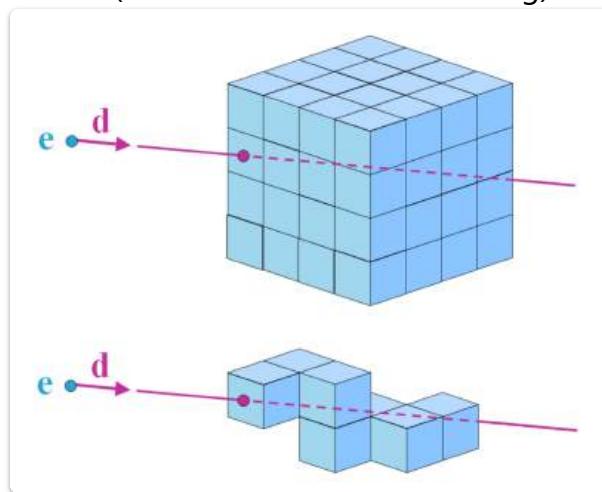
- **Alternative Objektumgebungen:**

- Statt Umgebungskugeln können auch andere Formen verwendet werden, z.B. Umgebungsquader (Bounding Boxes).
- **Abwägung:** Mehraufwand für komplexere Formen vs. genauere Umschließung (engeres Anliegen) und potenzieller Gewinn bei der Schnittprüfung.

## Raumteilungs-Methoden

EVC\_Skriptum\_CG, p.42

- **Alternative zu Objektumgebungen:** Aufteilung des gesamten Raumes, in dem sich die Szene befindet.
  - Unabhängig von der Objektverteilung.
  - **Methoden:**
    - Regelmäßiges Raster (Array von Würfeln/Voxeln).
    - Octree (hierarchische Raumauftteilung).



- **Vorteile:**

- Man muss nur die Objekte in den Teilräumen betrachten, durch die der Strahl geht.
- Schnelle Berechnung des nächsten Teilraums auf dem Strahlpfad.
- Sobald ein Schnittpunkt innerhalb eines Teilwürfels gefunden wurde, kann die Suche beendet werden.

- **Algorithmen:** Ähnlich dem 3D-Bresenham-Verfahren zur schnellen Durchquerung der Teilräume.

- **Vorgehensweise für einzelne Teilwürfel:**

- Strahlgleichung:  $\mathbf{p}(t) = \mathbf{p}_0 + t \cdot \mathbf{d}$
- Eintrittspunkt  $\mathbf{p}_{in}$  des Strahls in den Teilwürfel.
- Normalenvektoren der Würfelflächen sind  $(\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1)$ .

- Für drei Flächen mit  $\mathbf{d} \cdot \mathbf{n} > 0$  (d.h. der Strahl bewegt sich auf die Fläche zu), bestimmt man den Schnittpunkt mit dem Strahl und wählt den vordersten Schnittpunkt (kleinstes  $t$ ) aus.
- Diese Methode funktioniert auch, wenn die Würfel unterschiedlich groß sind (z.B. in einem Octree).
- Berechnung des Austrittspunkts und des zugehörigen  $t$ -Wertes:
  - Austrittspunkt:  $\mathbf{p}_{out} = \mathbf{p}_{in} + t_k \mathbf{d}$
  - Ebene der Austrittsfläche:  $\mathbf{n}_k \cdot \mathbf{p} = -D_k$
  - Da  $\mathbf{p}_{out}$  auf der Ebene liegt:  $\mathbf{n}_k \cdot \mathbf{p}_{out} = -D_k$
  - Einsetzen der Gleichung für  $\mathbf{p}_{out}$ :  $\mathbf{n}_k \cdot (\mathbf{p}_{in} + t_k \mathbf{d}) = -D_k$
  - Auflösen nach  $t_k$ :
    - $\mathbf{n}_k \cdot \mathbf{p}_{in} + t_k(\mathbf{n}_k \cdot \mathbf{d}) = -D_k$
    - $t_k(\mathbf{n}_k \cdot \mathbf{d}) = -D_k - \mathbf{n}_k \cdot \mathbf{p}_{in}$
    - $t_k = \frac{-D_k - \mathbf{n}_k \cdot \mathbf{p}_{in}}{\mathbf{n}_k \cdot \mathbf{d}}$

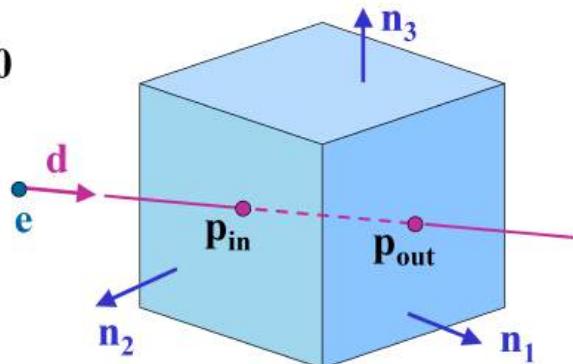
Slides:

ray direction  $\mathbf{d}$  & ray entry position  $\mathbf{p}_{in}$

→ potential exit faces  $\mathbf{d} \cdot \mathbf{n}_k > 0$

→ normal vectors

$$\mathbf{n}_k = \begin{cases} (\pm 1, 0, 0) \\ (0, \pm 1, 0) \\ (0, 0, \pm 1) \end{cases}$$



→ check signs of components of  $\mathbf{d}$

calculation of exit positions, select smallest  $t_k$

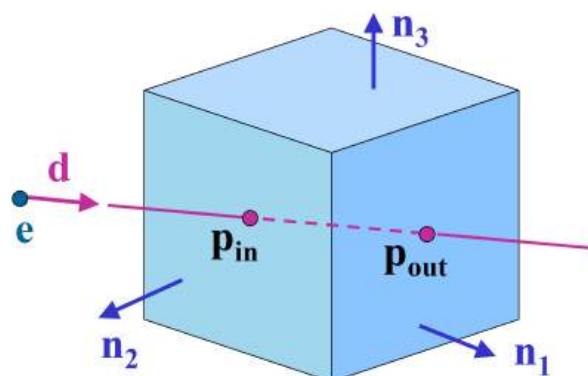
$$\mathbf{p}_{out,k} = \mathbf{p}_{in} + t_k \mathbf{d}$$

$$\mathbf{n}_k \cdot \mathbf{p}_{out,k} = -D_k$$

$$t_k = \frac{-D_k - \mathbf{n}_k \cdot \mathbf{p}_{in}}{\mathbf{n}_k \cdot \mathbf{d}}$$

example:  $\mathbf{n}_k = (1, 0, 0)$

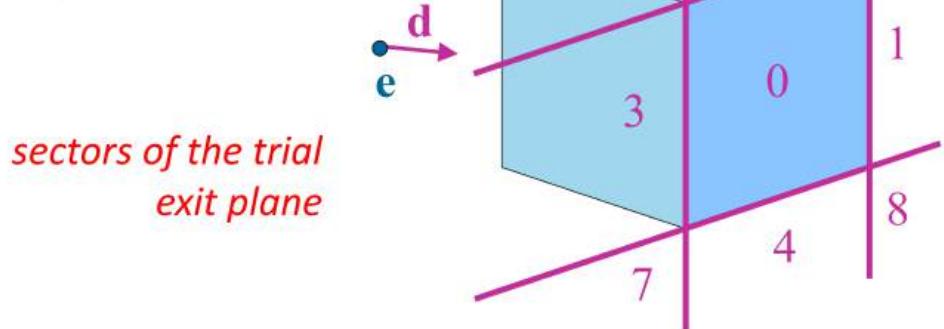
$$x_k = -D_k \Rightarrow t_k = \frac{x_k - x_0}{x_d}$$



variation: trial exit plane

perpendicular to largest component of  $\mathbf{d}$

- (a) exit point in 0  $\Rightarrow$  done
- (b)  $\{1, 2, 3, 4\} \Rightarrow$  side clear
- (c)  $\{5, 6, 7, 8\} \Rightarrow$  extra calc.



# 10. Stereo und Motion

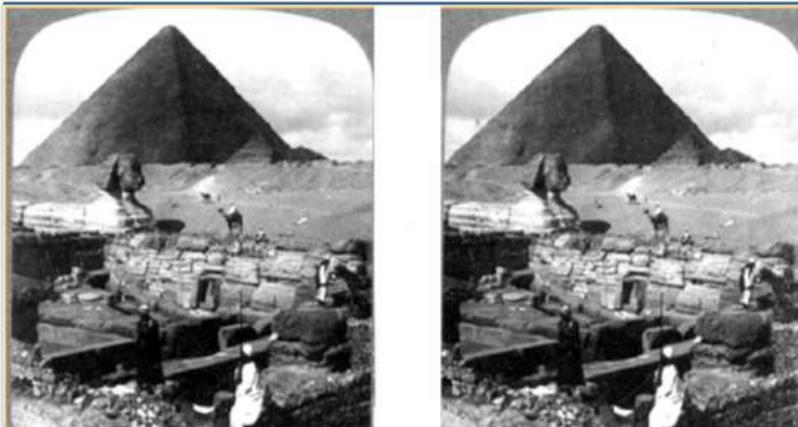
## Stereo Vision

[EVC\\_Skriptum\\_CV, p.51](#)

- Zusammengesetzt aus griechisch "stereos" (räumlich, fest) und lateinisch "videre" (sehen).
- Bezeichnet räumliches Sehen bzw. binokulares Sehen.
- Ziel: Erstellung eines Tiefenbildes aus zwei Bildern einer Szene.
- Bilder sind 2D-Projektionen einer 3D-Szene, wobei eine Dimension (die Tiefe) verloren geht.
- Der Mensch kann Tiefeninformationen aus seiner Umgebung durch binokulares Sehen gewinnen.

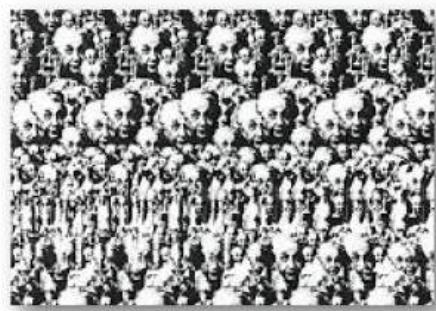
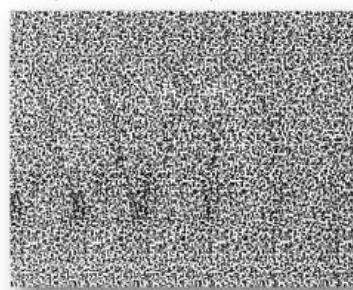
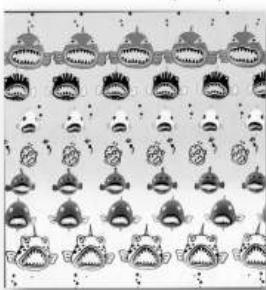
Prinzip der Stereo Vision:

- Nutzt zwei Kameras mit bekanntem Abstand zueinander.
- Anwendung geometrischer Prinzipien (Triangulation und Epipolargeometrie) zur Berechnung korrespondierender Bildpunkte.
- Rekonstruktion der Tiefenwerte.



For some people, Random Dot Stereograms are complicated to view:  
Autostereograms [Tyler77]:

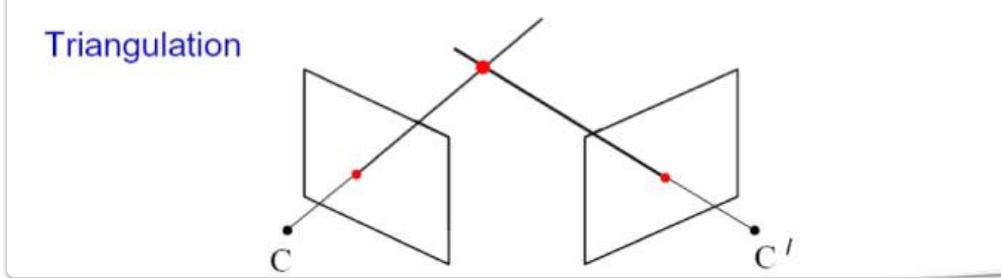
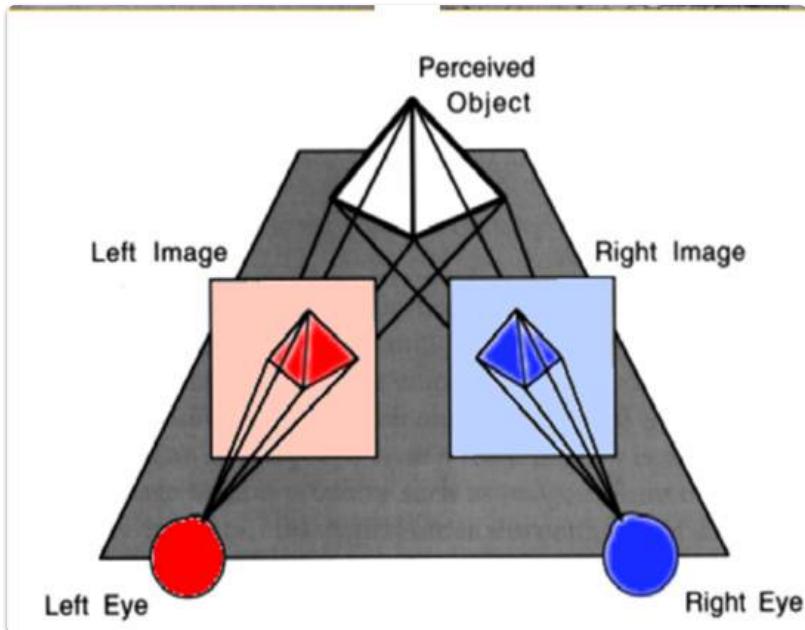
- Also called single image Stereogram
- Form of Art
- Is formed by repetition of patterns in specific intervals



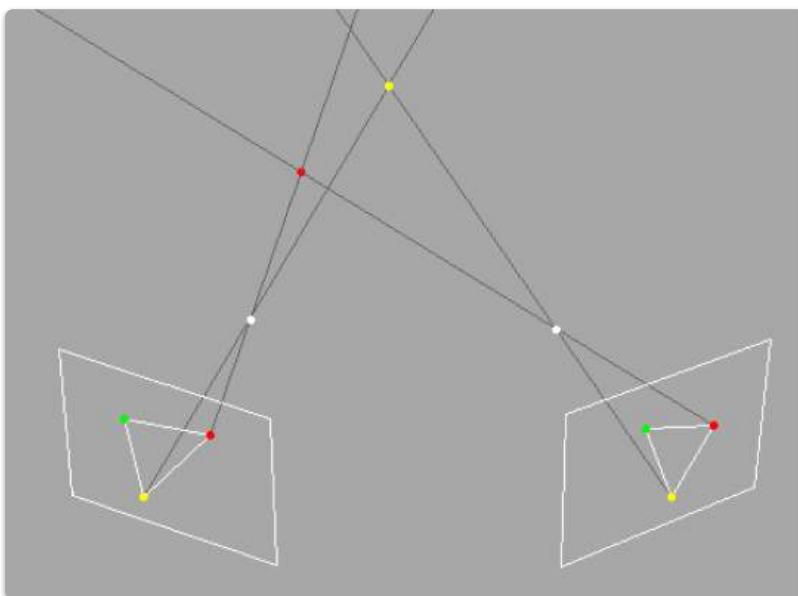
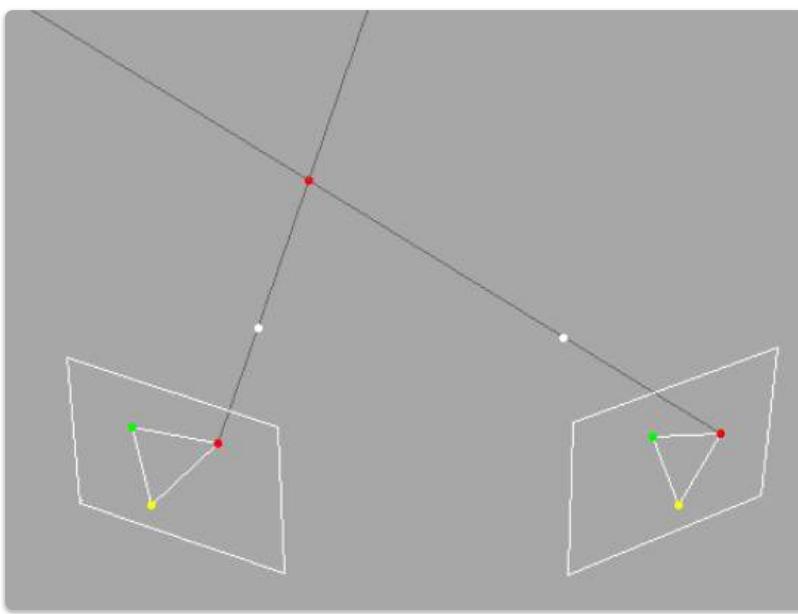
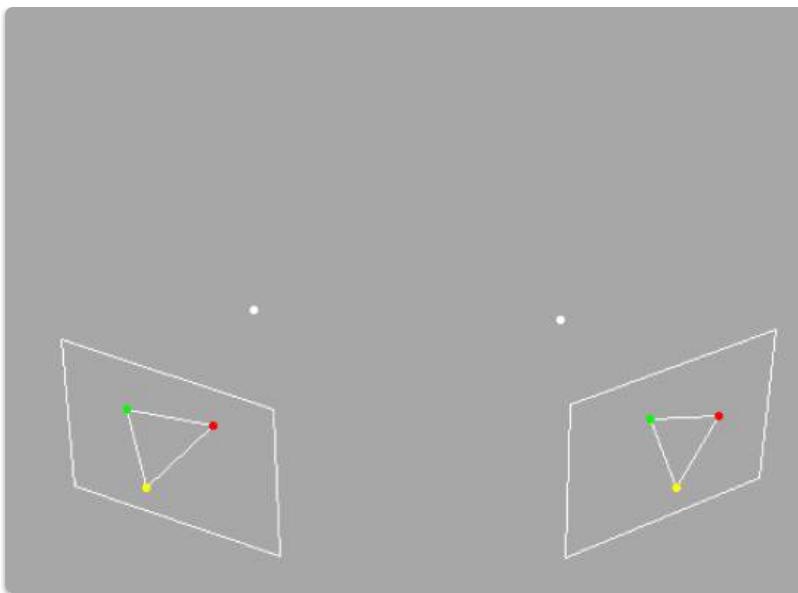
Tyler C.W. and Chang J.J. (1977) [Visual echoes: The perception of repetition in quasi-random patterns](#). *Vision Res.* 17, 109-116.

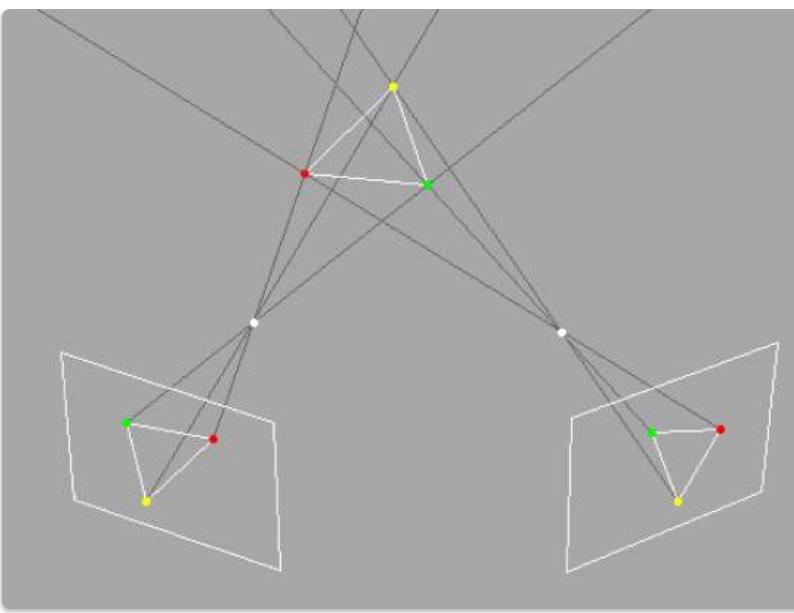
## Disparität:

- Leichter Versatz der beiden Kameras erzeugt unterschiedliche Bilder.
- Der Versatz in den Kamerabildern ist geringer für entfernte Objekte und größer für nahe Objekte.
- Dieser Versatz wird als Disparität bezeichnet.
- Durch Zuweisung der unterschiedlichen Disparitäten zu jedem Bildpunkt wird eine Disparitätsmatrix erstellt.
- Aus der Disparitätsmatrix lässt sich für jeden Bildpunkt die Tiefe ableiten.



**Triangulation: Fundament von Stereo Vision**

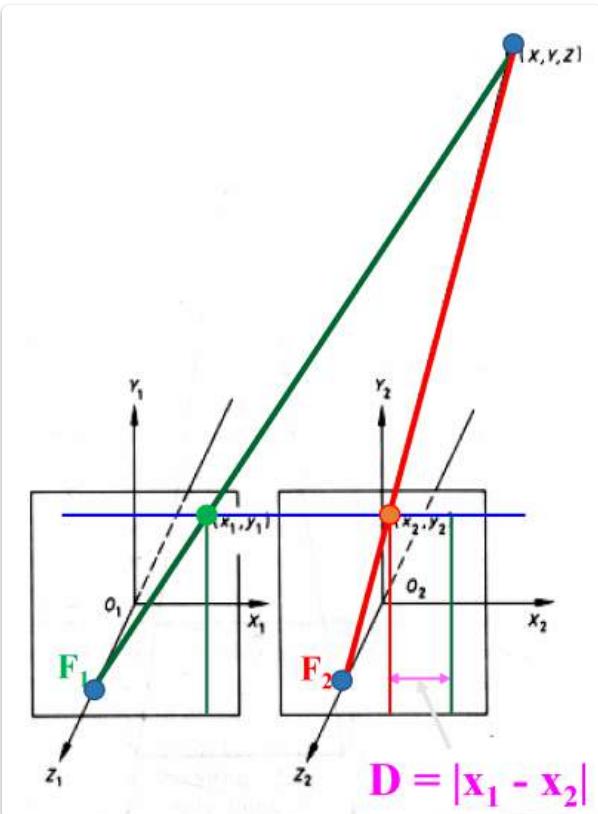




## Stereo Geometry

Grundlagen:

- Betrachtet werden *Perspektivische Projektionen* von Objektpunkten.
- **Spezifischer Fall (vereinfacht):** Beide Bildebenen sind parallel zueinander ausgerichtet.
  - Dieser spezifische Fall kann zur Erklärung des generellen Falls herangezogen werden.
- Durch die Nutzung der geometrischen Position der Bildebenen und der Information über die Tiefenprojektion des Objekts kann die Tiefe des Objekts berechnet werden.



Wichtige Elemente in der Abbildung:

- $(X, Y, Z)$ : Ein Punkt im 3D-Raum.
- $F_l$ : Projektionszentrum der linken Kamera.
- $F_r$ : Projektionszentrum der rechten Kamera.
- Bildebene (parallel zueinander).
- $x'_l$ : Projektion des 3D-Punktes auf der linken Bildebene.
- $x'_r$ : Projektion des 3D-Punktes auf der rechten Bildebene.
- $D = |x'_l - x'_r|$ : Die *Disparität* – der horizontale Abstand zwischen den korrespondierenden Bildpunkten.

Zusammenfassend: Stereo Vision nutzt zwei leicht versetzte Kameras, um durch die Analyse der Disparität zwischen den beiden resultierenden Bildern Tiefeninformationen zu gewinnen. Die geometrische Beziehung zwischen den Kameras ermöglicht die Berechnung der 3D-Positionen der Punkte in der Szene.

## Stereoskopie

---

[EVC\\_Skriptum\\_CV, p.51](#)

Stereoskopie (griechisch "skopeo" = betrachten):

- Wiedergabe von Bildern mit einem räumlichen Eindruck von Tiefe.
- Die Tiefe ist physikalisch nicht vorhanden.
- Umgangssprachlich fälschlich als "3D" bezeichnet, obwohl es sich um zweidimensionale Abbildungen handelt, die einen räumlichen Eindruck vermitteln.

Prinzip des räumlichen Sehens:

- Bereits im 3. Jh. v. Chr. von dem griechischen Mathematiker Euklid beschrieben.
- Viele Wissenschaftler (u.a. Leonardo da Vinci) beschäftigten sich mit diesem Phänomen.

Geschichte der Stereoskopie:

- 19. Jh.: Charles Wheatstone entdeckte die Stereoskopie.
  - Hielt 1838 einen bahnbrechenden Vortrag über "einige merkwürdige und bisher nicht beobachtete Erscheinungen beim beidäugigen Sehen".
  - Berechnete und zeichnete Bildpaare.
  - Konstruierte das Stereoskop, ein Apparat, um diese Bildpaare räumlich betrachten zu können



Weitere Entwicklung:

- 1849: David Brewster stellte die erste Stereokamera vor.
  - Ermöglichte erstmals, ein bewegtes Motiv aufzunehmen (allerdings noch nicht für Fotografie im heutigen Sinne).
- 1851: Durchbruch auf der Weltausstellung in London.

- Hardware for Viewing (3D-TV sets):
  - Anaglyph
  - Polarized
  - Field-sequential (Active shutter)
  - Lenticular display

Anaglyph

Polarizing

Active shutter

Lenticular

Zusammenfassend: Die Stereoskopie erzeugt einen räumlichen Eindruck aus zwei leicht unterschiedlichen, zweidimensionalen Bildern. Historisch gesehen reicht die Beobachtung dieses Phänomens bis in die Antike zurück, die eigentliche Erfindung des Stereoskops und der Stereokamera erfolgte jedoch im 19. Jahrhundert.

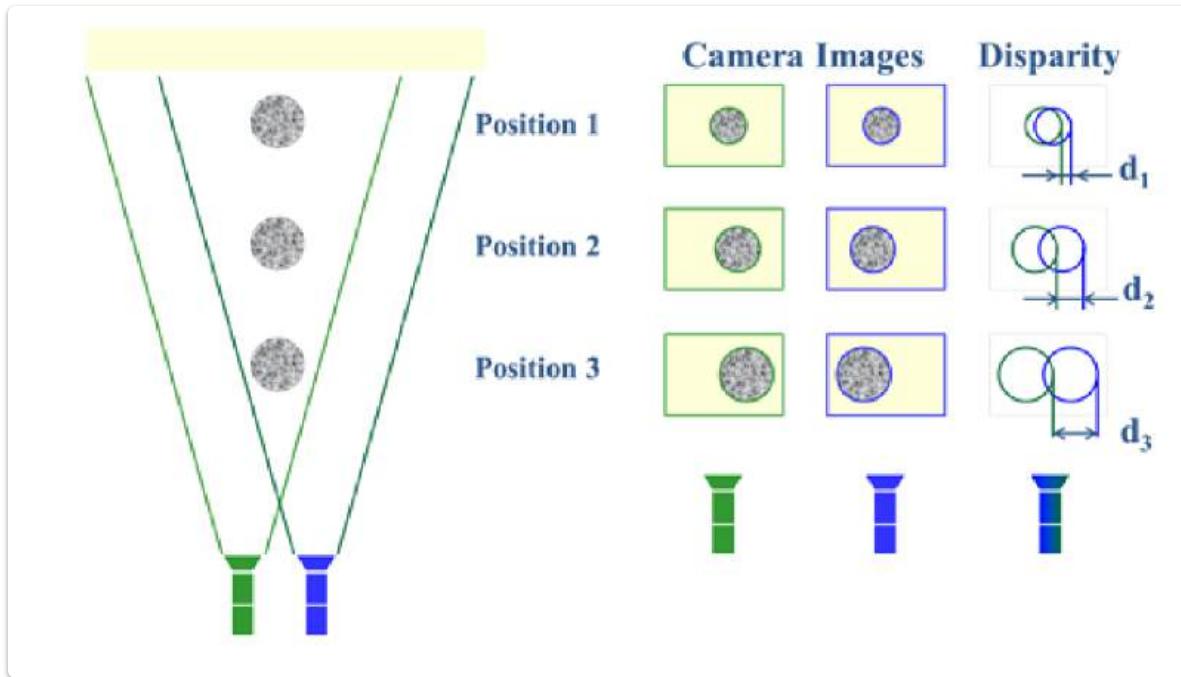
## Disparität

[EVC\\_Skriptum\\_CV, p.52](#)

Definition:

- Ein einzelner Punkt wird in beiden Bildern des Stereosystems auf unterschiedliche Bildkoordinaten abgebildet.

- Die **horizontale Differenz** zwischen diesen Bildkoordinaten nennt man *Disparität*.



Erläuterung anhand paralleler Kameras (siehe Abbildung 44):

- Eine Kugel wird an drei verschiedenen Positionen (Abständen zum Kamerapaar) aufgenommen.
- In der Überlagerung der beiden Kamerabilder kann die Disparität beobachtet werden.
- Position 1 (weit entfernt):**
  - Unterschied der Position der Kugel im überlagerten Bild ( $d_1$ ) ist klein.
- Position 2 und 3 (näher):**
  - Die Bilder der Kugel werden größer, da sie näher sind.
  - Der Unterschied der Punkte in den überlagerten Bildern ( $d_2, d_3$ ) wird größer.
  - Die Disparität wird größer ( $d_3 > d_2 > d_1$ ).

Zusammenhang zwischen Disparität und Tiefe:

- Je näher das Objekt zur Kamera ist, desto größer ist die Disparität.
- Im Unendlichen ist die Disparität 0.
- Die Disparität ist somit *umgekehrt proportional* zur Tiefe.

## Normalfall (Achsparalleles Stereosystem)

[EVC\\_Skriptum\\_CV, p.52](#)

Definition:

- Zeichnet sich durch zwei Kameras aus, die horizontal verschoben sind.
- Deren Koordinatensysteme sind nicht gegeneinander verdreht.
- Der Abstand zwischen den beiden optischen Zentren wird *Basislinie* ( $B$ ) genannt.

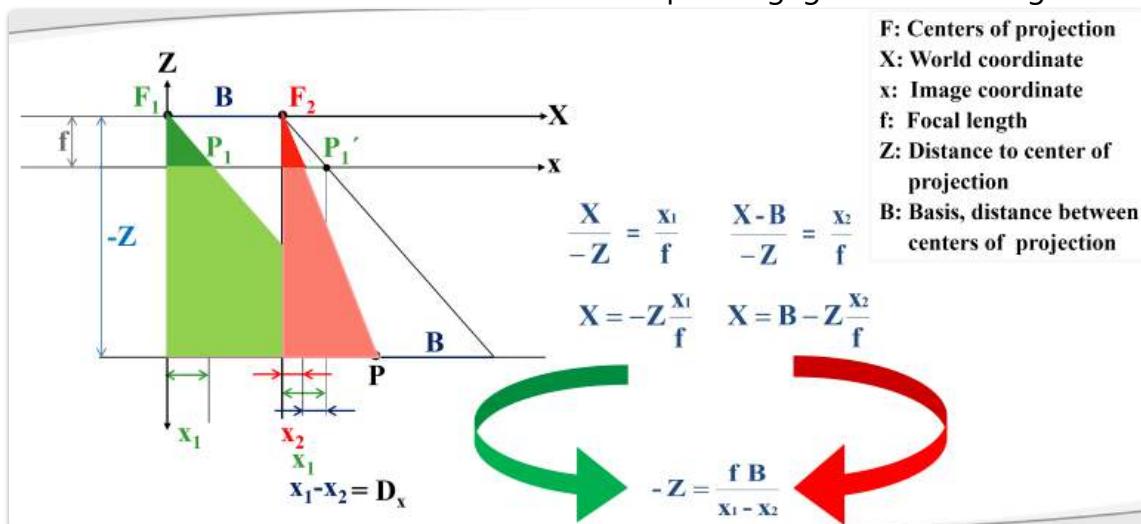
- Die Brennweite ( $f$ ) legt den Abstand der beiden Brennpunkte zu ihren Bildebenen fest und ist für beide Kameras als identisch vorausgesetzt.
- Ein 3D-Punkt  $X$  wird somit über die beiden optischen Zentren in den Abbildungen  $x$  und  $x'$  projiziert.
- Beim achsparallelen Stereosystem sind die Bildzeilen identisch, was für die unterschiedliche Perspektive der Kameras hinsichtlich des 3D-Punktes  $X$  nur zu einer horizontalen Disparität  $D$  in der Abbildung führt.
- Die Disparität wird im Allgemeinen in Bildkoordinaten berechnet, sodass die Einheit Pixel ist:  $D = |x_l - x_r|$ .

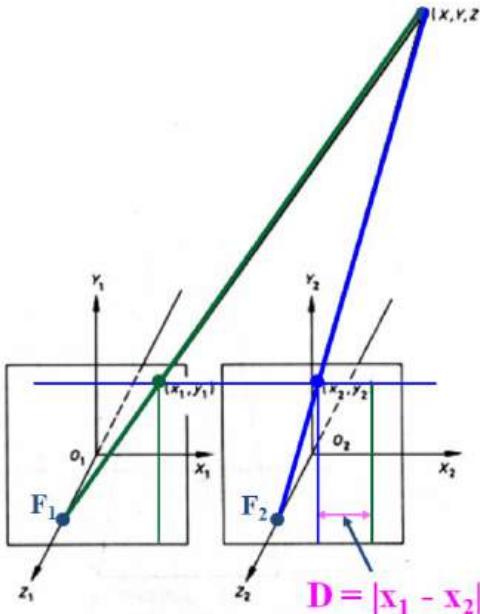
Tiefenberechnung:

- Der Abstand  $Z$  eines Punktes  $X$  von der Kamera lässt sich aus den bekannten konstanten Kameraparametern  $f$ ,  $B$  sowie der Disparität  $D$  berechnen:

$$Z = \frac{B \cdot f}{D}$$

- Damit stellt die Disparität ein Maß für die Raumtiefe des 3D-Punktes  $X$  dar und verhält sich *umgekehrt proportional* zu ihr.
- Für Punkte im Unendlichen muss daher die Disparität gegen Null konvergieren.



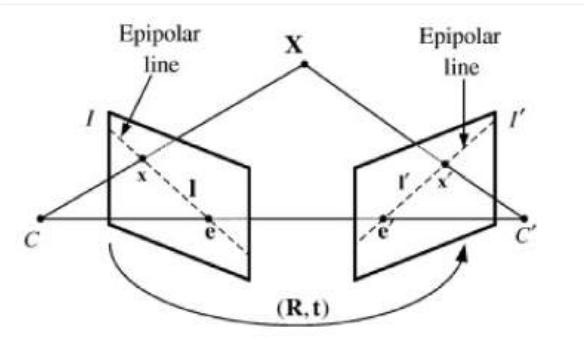


## Epipolargeometrie

EVC\_Skriptum\_CV, p.52

Kameraanordnungen mit zwei Kameras im Stereosystem:

- Können bezüglich ihrer räumlichen Anordnung in zwei grundlegende Klassen eingeteilt werden:
  - Das bereits vorgestellte *achsparallele Stereosystem (Normalfall)*.
  - Die *konvergente Anordnung* (Ausrichtung der optischen Achsen auf einen Konvergenzpunkt).
- Bei der allgemeineren Stereogeometrie, auch *Epipolargeometrie* genannt, sind die beiden Kameras nicht nur zueinander verschoben, sondern auch zueinander gedreht.



Wichtige Begriffe:

- **Epipole (e und e')**: Die Schnittpunkte der Verbindungsgeraden der beiden Kamerazentren (Basislinienebene) mit den jeweiligen Bildebenen. Die Epipole können auch als Projektion des optischen Zentrums der einen Kamera in der Bildebene der anderen Kamera aufgefasst werden.

- **Epipolarebene:** Die Ebene, die durch den 3D-Punkt  $X$  und die beiden Brennpunkte  $C$  und  $C'$  aufgespannt wird.
- **Epipolarlinien ( $l$  und  $l'$ ):** Die Schnittlinien der Epipolarebene mit den beiden Bildebenden. Betrachtet man die beiden Sehstrahlen des 3D-Punktes in den beiden Kameras als Gummiband, so bewegt man diesen Punkt innerhalb der Epipolarebene, wobei diese jedoch immer auf den Epipolarlinien liegen.

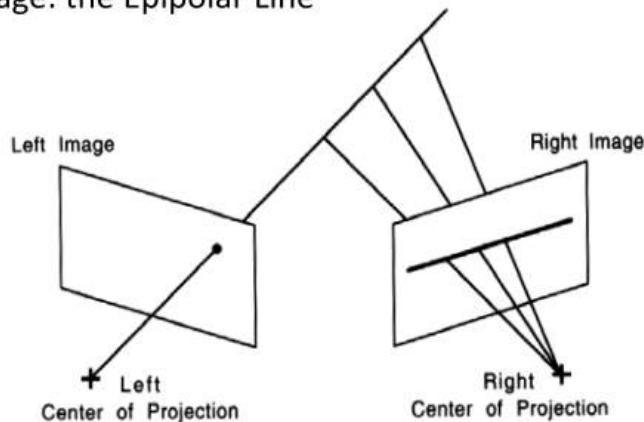
Bedeutung der Epipolarlinien:

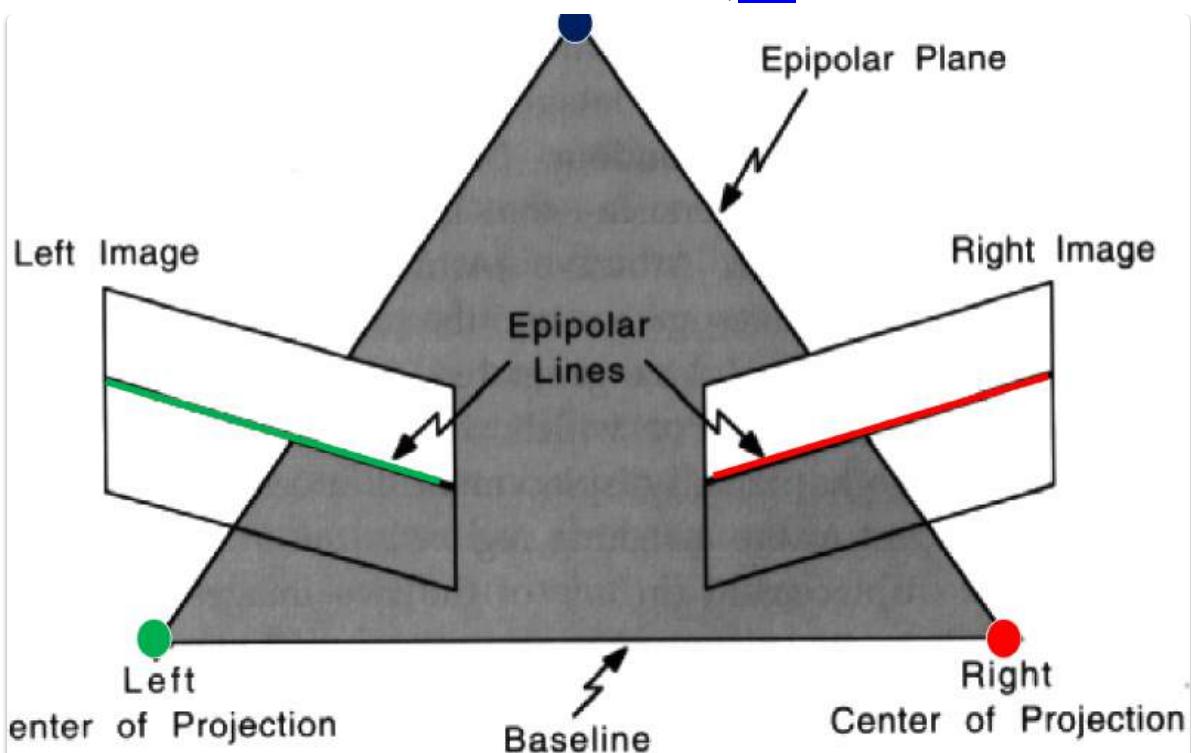
- Lässt man den 3D-Punkt  $X$  entlang seines Sehstrahles (z.B. in Richtung Kamera 1) laufen, so ergibt sich immer die gleiche Abbildung  $x$  in Kamera 1, während in Kamera 2 die Abbildung  $x'$  entlang der Epipolarlinie  $l'$  wandert.



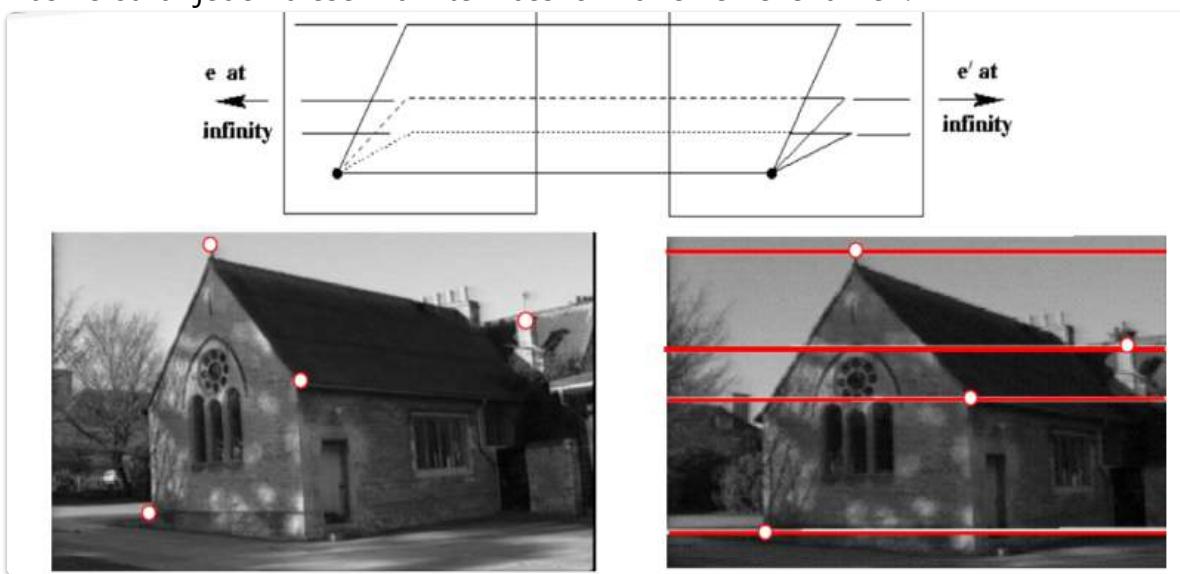
- Der Sehstrahl von jedem 3D-Punkt in einer Kamera liefert somit als Projektion in der anderen Kamera die entsprechende Epipolarlinie.
- Folglich muss auch für jeden Bildpunkt in einer Kamera der korrespondierende Punkt auf einer der Epipolarlinien in der anderen Kamera liegen.

- **Epipolar Constraint:** Each point of the left image can lie only on a specific line in the right image: the Epipolar Line





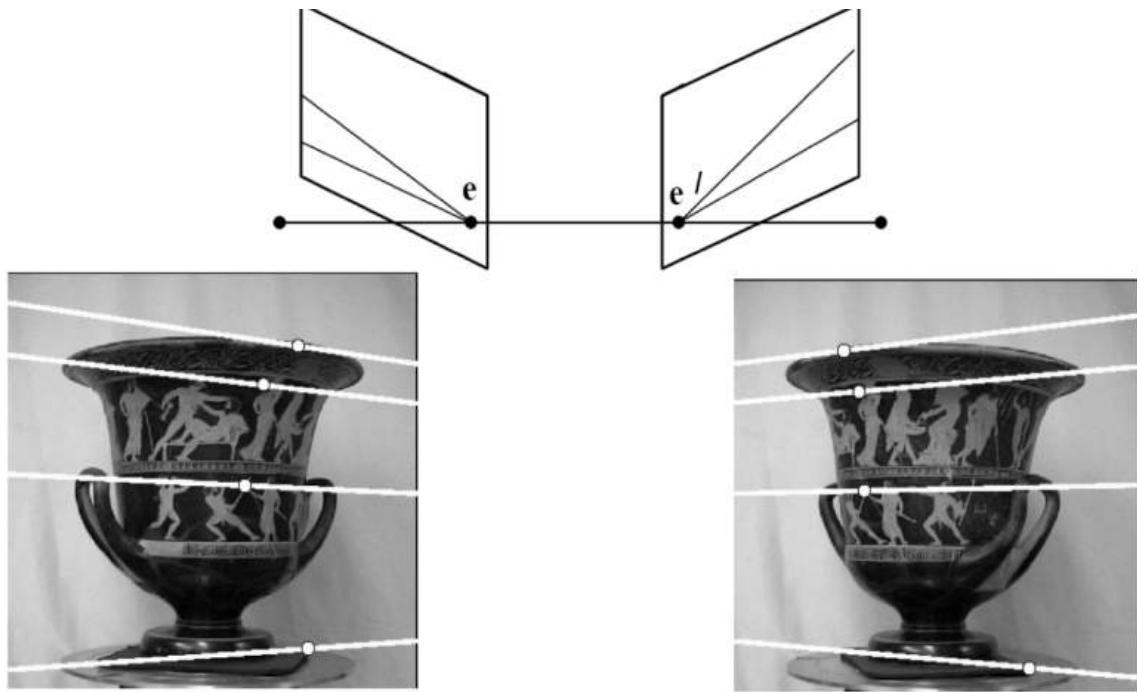
Das heißt für jeden dieser Punkte muss ich nur eine Zeile fahren:



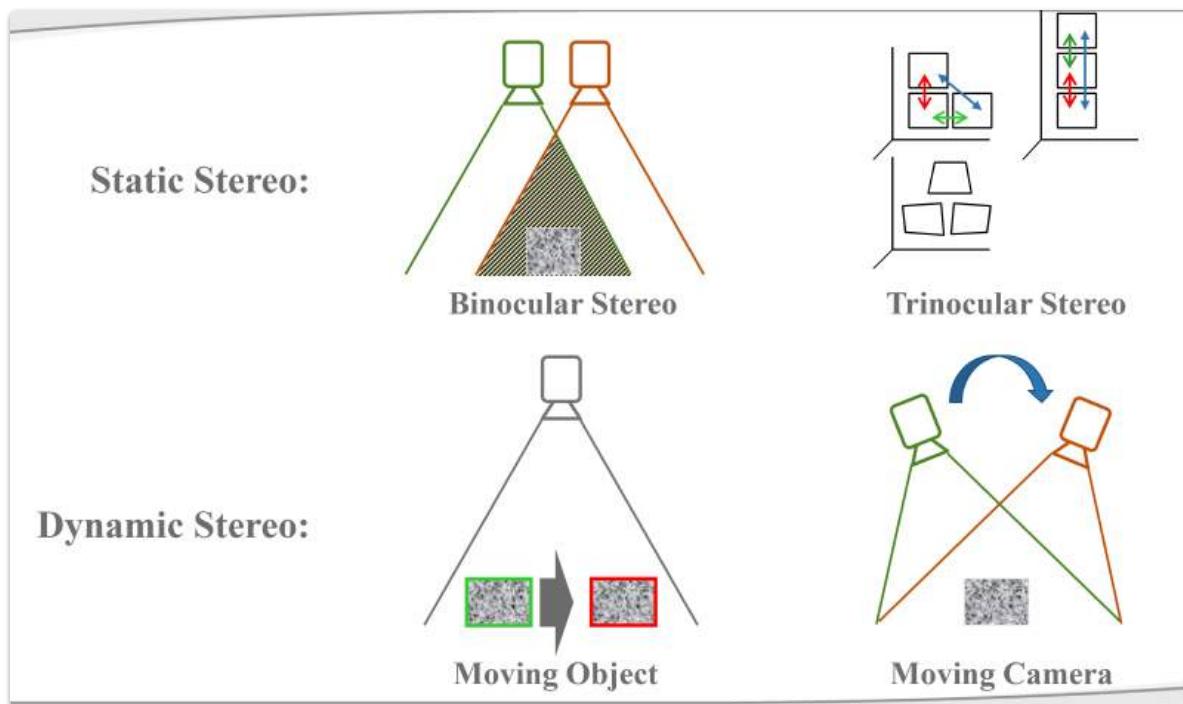
Wichtige Eigenschaften:

- Die Epipolargeometrie hängt **nur** von der relativen Pose (Position und Orientierung) und den internen Parametern der beiden Kameras ab.
  - Dazu gehören die Position der Kamerazentren und der Bildebenen.
- Sie hängt **nicht** von der Szenenstruktur ab (den 3D-Punkten außerhalb der Kameras).

Im Normalfall sind diese Epipole nicht parallel:

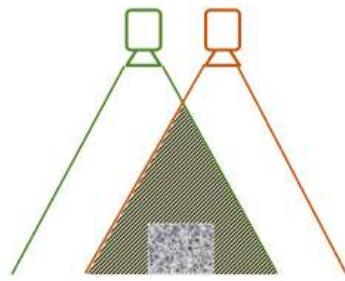


## Camera Setup



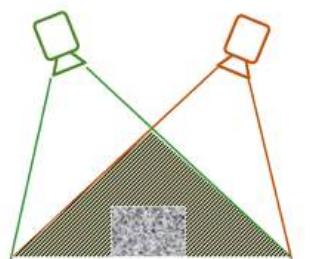
- Baseline

- Distance between cameras (focal points)



- Trade-off

- Small baseline: Matching easier
- Large baseline: Depth precision better

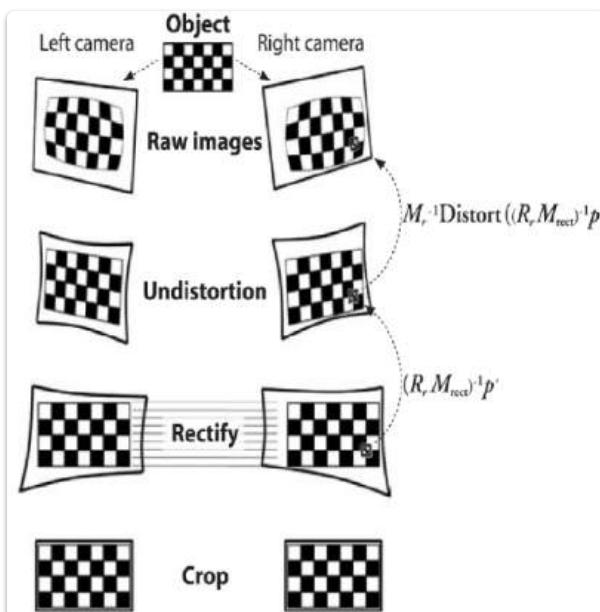


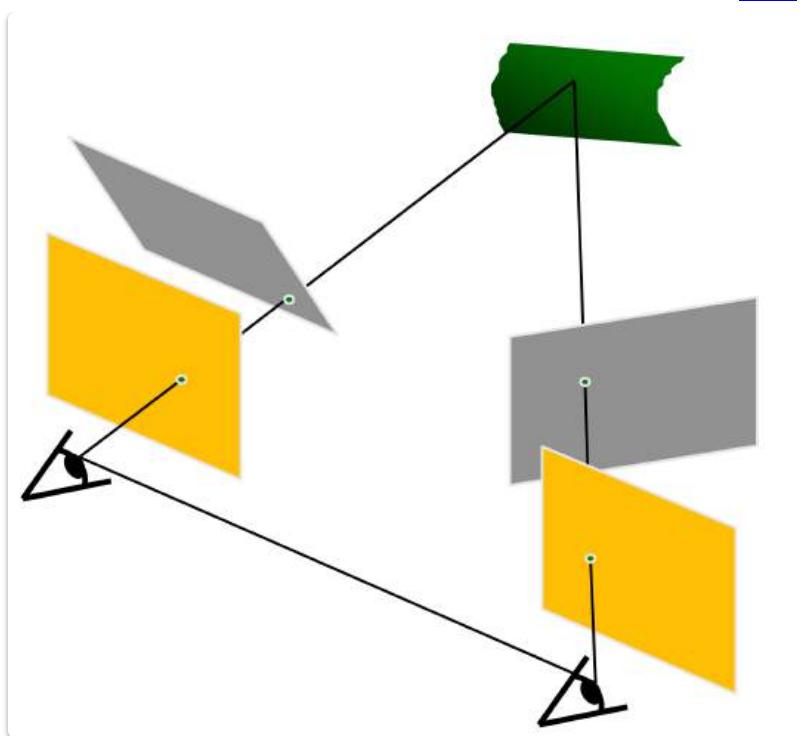
## Image Rectification

Ziel: Erreichen einer vereinfachten Stereo-Geometrie (ähnlich dem Normalfall) durch Bildrektifikation.

Image Re-projection:

- Die Bildebenen werden auf eine gemeinsame Ebene re-projiziert.
- Diese gemeinsame Ebene ist parallel zur Linie zwischen den optischen Zentren (Basislinie).
- Wichtig: Es ist zu beachten, dass hauptsächlich der Brennpunkt der Kamera relevant ist.





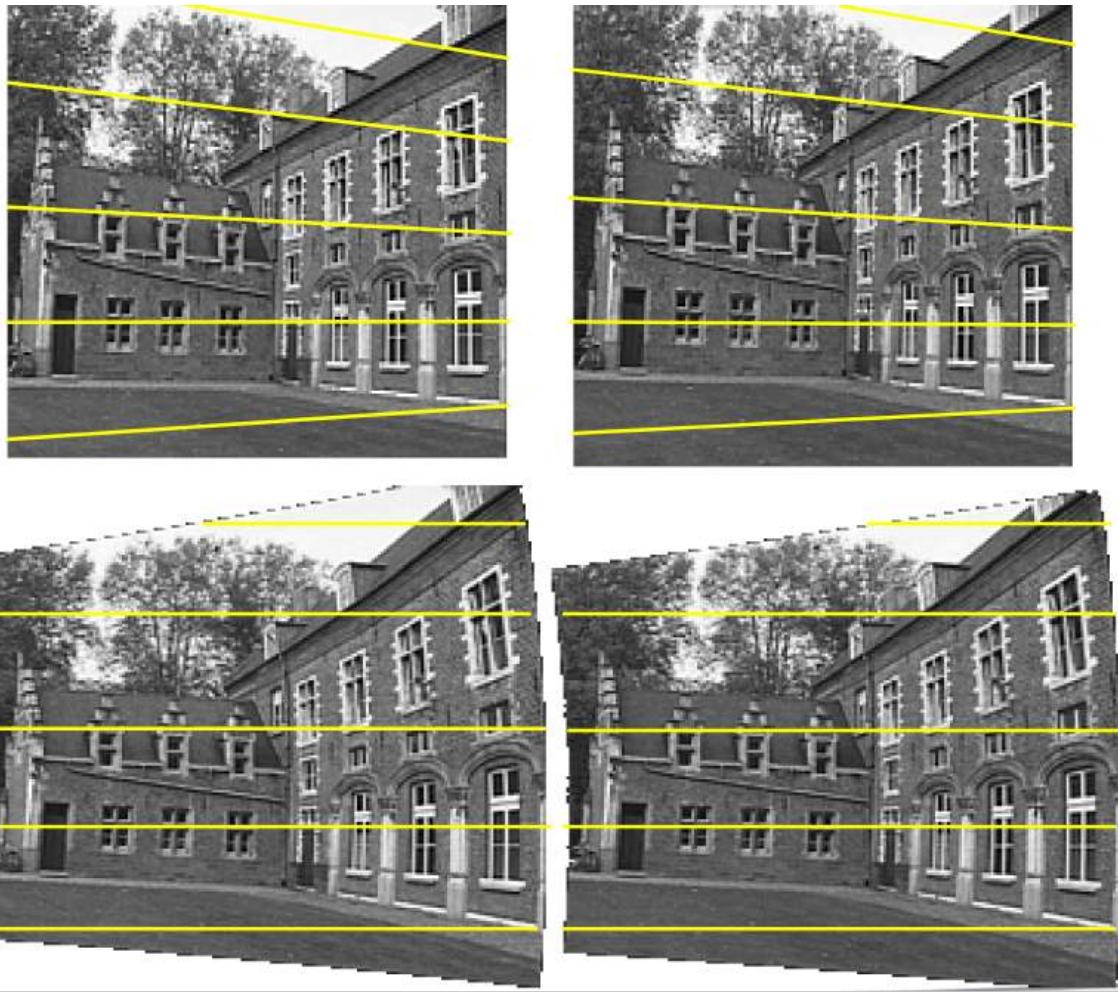
Prozess der Bildrektifikation (illustriert in der Abbildung):

1. **Raw Images:** Die ursprünglichen, möglicherweise verzerrten Bilder der linken und rechten Kamera.
2. **Undistortion:** Korrektur der Linsenverzerrung in den Rohbildern.
3. **Rectify:** Transformation der undistorrierten Bilder, sodass korrespondierende Punkte in den beiden Bildern auf der gleichen horizontalen Zeile liegen. Dies entspricht der Geometrie mit parallelen Bildebenen.
4. **Crop (optional):** Beschneidung der rektifizierten Bilder, um Bereiche ohne gültige Informationen zu entfernen.

Vorteil der Bildrektifikation:

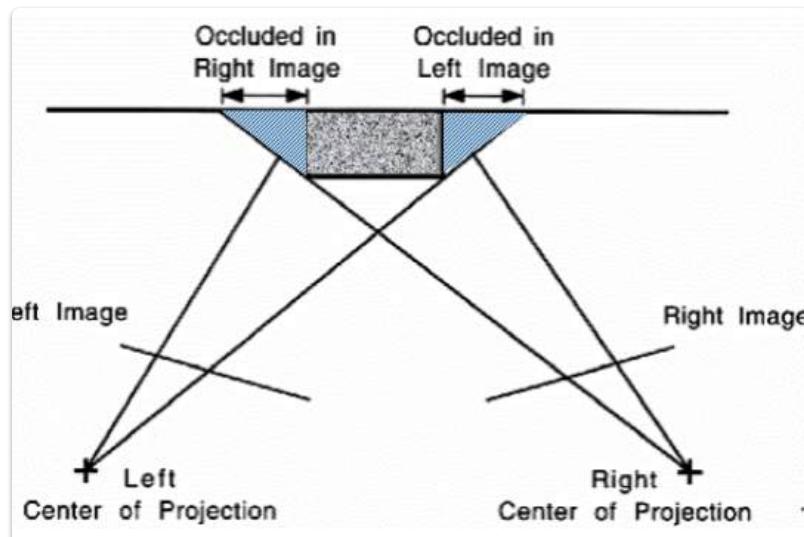
- Vereinfacht die Suche nach korrespondierenden Punkten erheblich, da diese nun auf horizontalen Epipolarlinien liegen (die zu horizontalen Zeilen im Bild werden).
- Ermöglicht die direkte Anwendung von Algorithmen, die für den Normalfall der Stereo-Geometrie entwickelt wurden.

**Beispiel:**



## Okklusionen

- **Ansichtsabhängig (View dependent):** Okklusionen treten auf, weil verschiedene Kameras unterschiedliche Perspektiven auf die Szene haben.
- **Verdeckte Punkte können nicht berechnet werden:** Bereiche, die in einem Bild verdeckt sind, liefern keine direkten Korrespondenzen im anderen Bild und somit keine Tiefeninformationen durch Stereo-Matching.



Erläuterung anhand der Abbildung:

- Ein Objekt (grauer Kasten) verdeckt einen bestimmten Bereich der Szene für die rechte Kamera (als "Occluded in Right Image" markiert).
- Gleichzeitig verdeckt das Objekt einen anderen Bereich der Szene für die linke Kamera (als "Occluded in Left Image" markiert).
- Diese verdeckten Bereiche können in den jeweiligen Bildern nicht mit korrespondierenden Punkten im anderen Bild abgeglichen werden.

Folge von Okklusionen:

- Führt zu fehlenden Tiefeninformationen in den Bereichen, die in mindestens einer der Kameras verdeckt sind.
- Die Größe und Position der okkludierten Bereiche hängen von der relativen Position und Orientierung der Kameras sowie der Geometrie der Szene ab.

## Testähnliches Beispiel

- Eine Szene wird mit einem Stereo-Setup aufgenommen. Der Abstand der beiden Kameras mit einer fokalen Länge von **450 Pixeln** beträgt **8cm**. Für einen Bildpunkt wird eine Disparität von **10 Pixeln** festgestellt. Wie weit ist der zugehörige Szenenpunkt entfernt?

$$\begin{aligned} \bullet & f = 450 \\ \bullet & B = 8\text{cm} \quad Z = \frac{f * B}{x_1 - x_2} \quad Z = \frac{450 * 8\text{cm}}{10} = 360\text{cm} \\ \bullet & D = 10 \end{aligned}$$

- Welche Disparität hat ein Szenenpunkt, der doppelt so weit entfernt ist?

$$D = \frac{f * B}{Z} \quad D = \frac{450 * 8\text{cm}}{720\text{cm}} = \frac{45}{9} = 5$$

## Korrespondenzproblem

[EVC\\_Skriptum\\_CV, p.53](#)

Definition:

- Das Korrespondenzproblem stellt das zentrale Problem der Stereo Vision dar.
- Es bezeichnet die Aufgabe, für jeden Bildpunkt im linken Bild jenen Punkt im rechten Bild zu finden, der denselben Objektpunkt abbildet.
- Entsprechende Suchverfahren werden als *Korrespondenzanalyse* oder auch als *Stereo Matching* bezeichnet.

Vereinfachung durch Epipolarkorrektur (Bildrektifikation):

- Aufgabe der Epipolarkorrektur ist es, Stereobildpaare anhand der Epipolargeometrie so zu transformieren, dass zusammengehörige Bildpunkte auf derselben horizontalen Linie liegen.

- Statt in zwei Dimensionen muss ein korrespondierender Punkt hier nur mehr entlang einer einzigen Scanline gesucht werden.
- Unter dieser Voraussetzung wird das Korrespondenzproblem wesentlich vereinfacht und die Korrespondenzanalyse somit beschleunigt.
- Gängige Stereo Matching Verfahren gehen in der Regel davon aus, dass die Bildpaare in *rektifizierter* Form vorliegen.

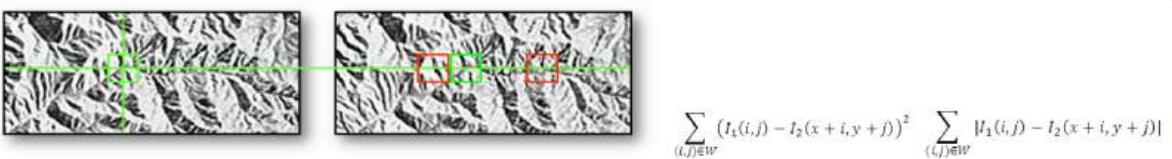
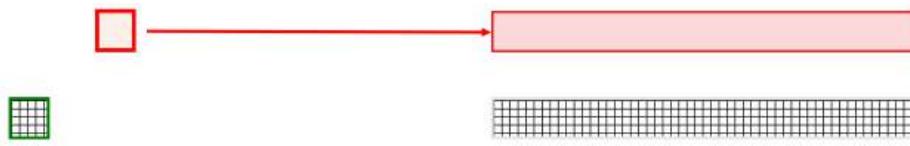
## Regionenbasiertes Matching (Area-Based Matching - ABM)

EVC\_Skriptum\_CV, p.53

Grundprinzip:

- Vergleicht kleine Ausschnitte (Beobachtungsfenster) zwischen den *rektifizierten* Bildern.
- Für jede Position eines Beobachtungsfensters im linken Bild wird das entsprechende Beobachtungsfenster im rechten Bild entlang der Epipolarlinie (jetzt eine horizontale Scanline) bewegt.
- Für jede Position wird geprüft, wie gut die Grauwerte mit den zu vergleichenden Grauwerten im linken Bild übereinstimmen.
- Dies geschieht durch eine Ähnlichkeitsmessung.

- The observation window in the left image is fixed
- For each position in the right image, the correlation function is calculated
- The window will be "slid" from left to right across the image
- Is performed for each position in the left image



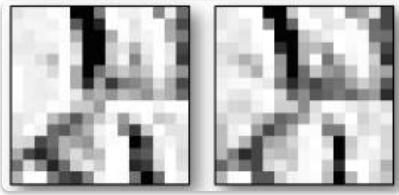
### Einfluss der Fenstergröße:

- **Zu kleine Fenster:** Beinhaltet zu wenig Information für eine korrekte Korrespondenzzuordnung, was zu erhöhten Fehlzuordnungen führen kann.
- **Zu große Fenster:** Erhöht die Rechenzeit.

### Gängige Ähnlichkeitsmaße:

- Summe der absoluten Differenzen (Sum of Absolute Differences - SAD)

- Summe der quadrierten Differenzen (Sum of Squared Differences - SSD)
- Normalisierte Kreuzkorrelation (Normalized Cross Correlation - NCC)



Formeln für Ähnlichkeitsmaße:

- **SSD (Sum of Squared Differences):**

$$SSD(\Delta m, \Delta n) = \sum_{i,j \in R} [I_1(i, j) - I_2(i - \Delta m, j - \Delta n)]^2$$

- $I_1(i, j)$ : Pixelintensität am Koordinaten  $(i, j)$  im ersten Fenster.
- $I_2(i - \Delta m, j - \Delta n)$ : Pixelintensität am verschobenen Koordinaten im zweiten Fenster.
- $R$ : Bereich des Fensters.
- $\Delta m, \Delta n$ : Verschiebung des zweiten Fensters relativ zum ersten.

- **CC (Cross-Correlation):**

$$CC(\Delta m, \Delta n) = \sum_{i,j \in R} [I_1(i, j) \cdot I_2(i - \Delta m, j - \Delta n)]$$

- Hier wird die Korrelation (Ähnlichkeit im Muster) direkt berechnet. Höhere Werte deuten auf größere Ähnlichkeit hin.

Anmerkung: Die gezeigte Formel für CC ist die unnormalisierte Kreuzkorrelation. Oft wird eine normalisierte Version (NCC) verwendet, um die Ergebnisse robuster gegenüber Helligkeits- und Kontrastunterschieden zu machen.

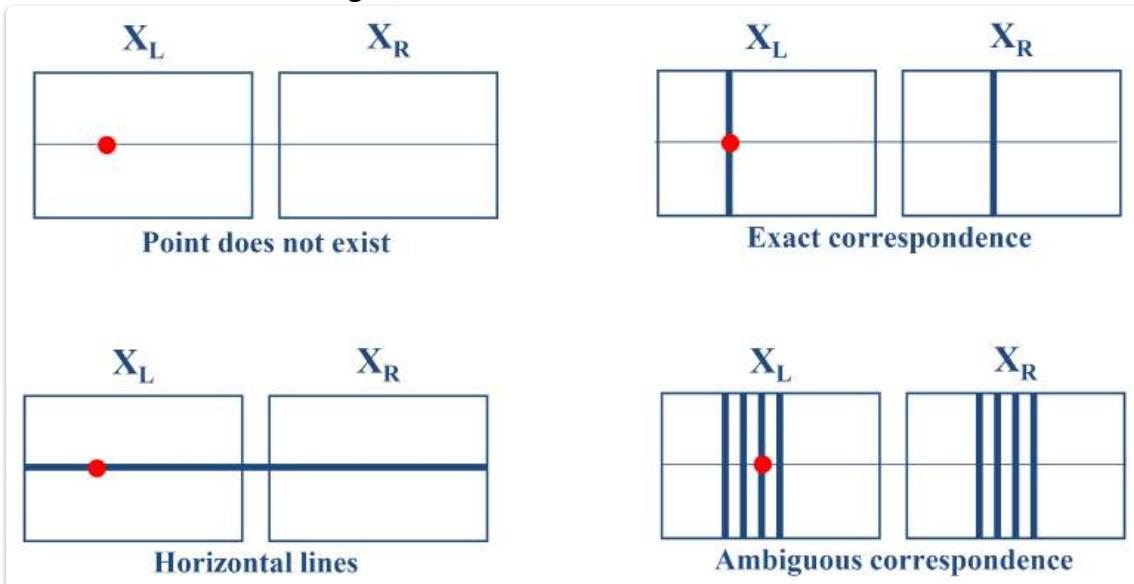
### Funktionsweise der Ähnlichkeitsmessung:

- Prinzipiell wird bei allen diesen Verfahren die Ähnlichkeit durch einen Vergleich von Pixeln innerhalb einer quadratischen Nachbarschaft zwischen dem linken und dem rechten Bild berechnet.
- Wenn das linke und rechte Bild exakt aufeinander passen, erhält man als Resultat ein Maximum (bei NCC) oder Minimum (bei SAD und SSD) in der Ähnlichkeitsfunktion.
- Mit Hilfe der Position des Maximums oder Minimums der Ähnlichkeitsfunktion wird die Position des korrespondierenden Punktes bestimmt (Disparität).

### Probleme:

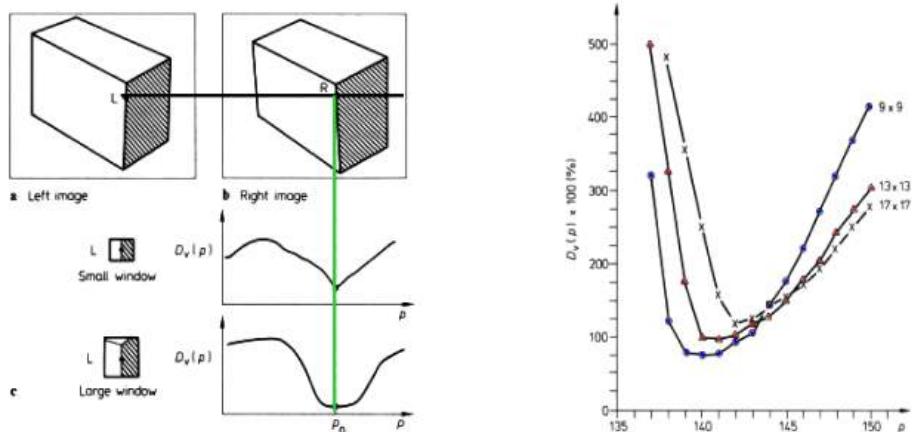
- Es ist möglich, dass kein eindeutiges Minimum oder Maximum gefunden werden kann.
- Dies kann zum Beispiel durch **Verdeckungen** passieren, wenn ein Punkt von einer Kamera aus sichtbar ist und von der anderen nicht.

- Intensitäten in den beiden Bildern müssen nicht zwingend passen, der korrespondierende Punkt ist nicht notwendig.

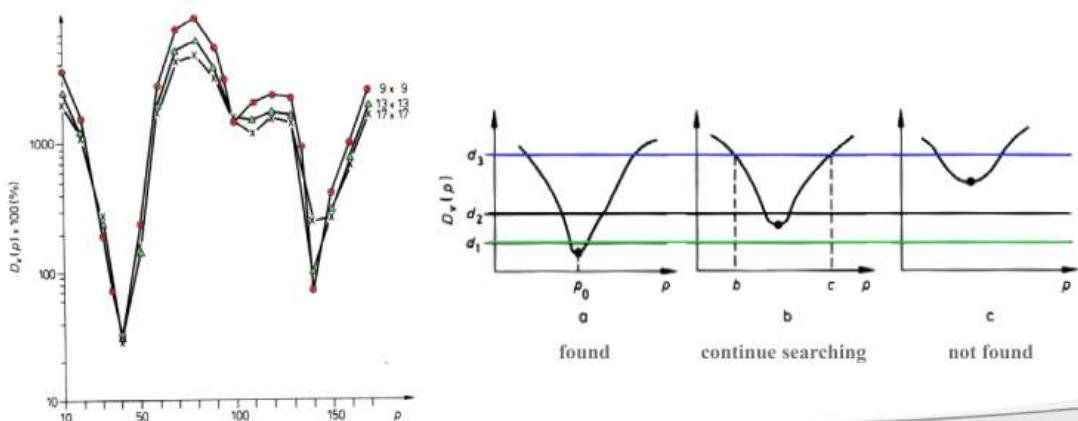


- Correspondence is strongly dependent on window size used

- Different algorithms like adaptive matching, pyramids ....



- Solution of ambiguity with threshold or additional constraints
- Different threshold values



Vorteil:

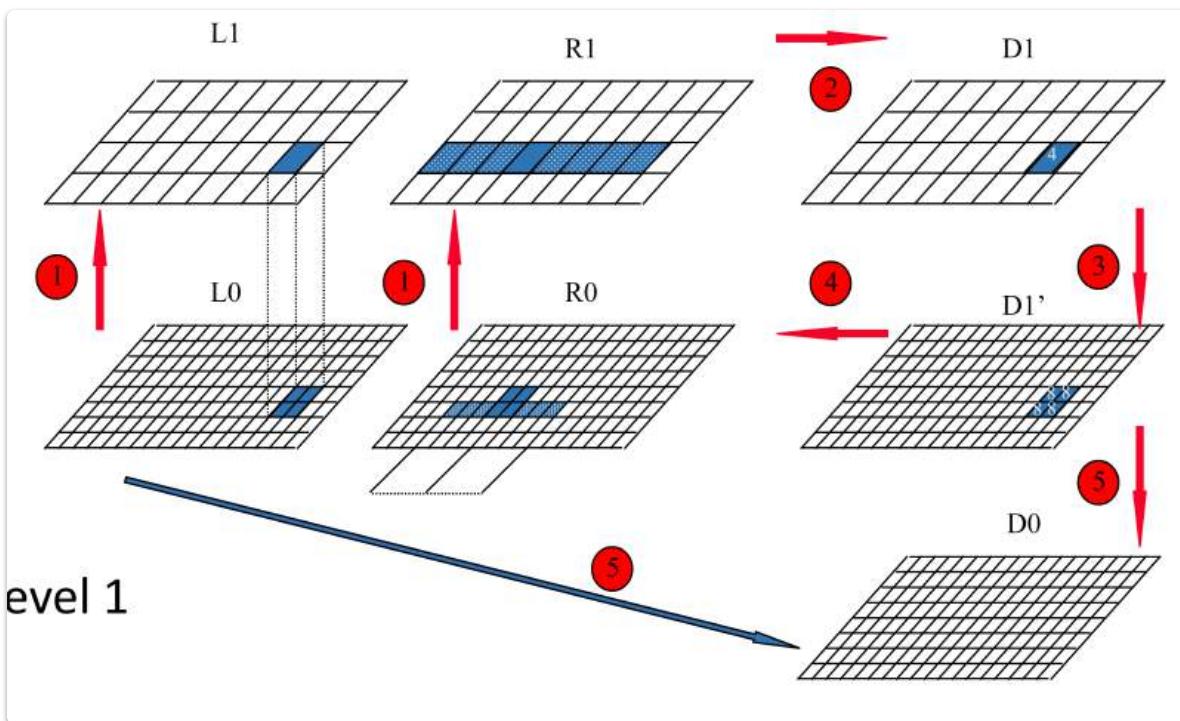
- Der Hauptvorteil des regionenbasierten Matching gegenüber den merkmalbasierten Verfahren ist ein dichteres Tiefenbild.
- Hier werden die Tiefenwerte für alle Pixel direkt ausgerechnet, nicht nur für einige ausgewählte Merkmalspunkte.

### Nachteil:

- Eine höhere Komplexität und ein entsprechend höherer Rechenaufwand.

### Hierarchical Matching

Ansatz: Verwendet Bildpyramiden (z.B. Gaussian Pyramids), um das Korrespondenzproblem effizienter zu lösen.

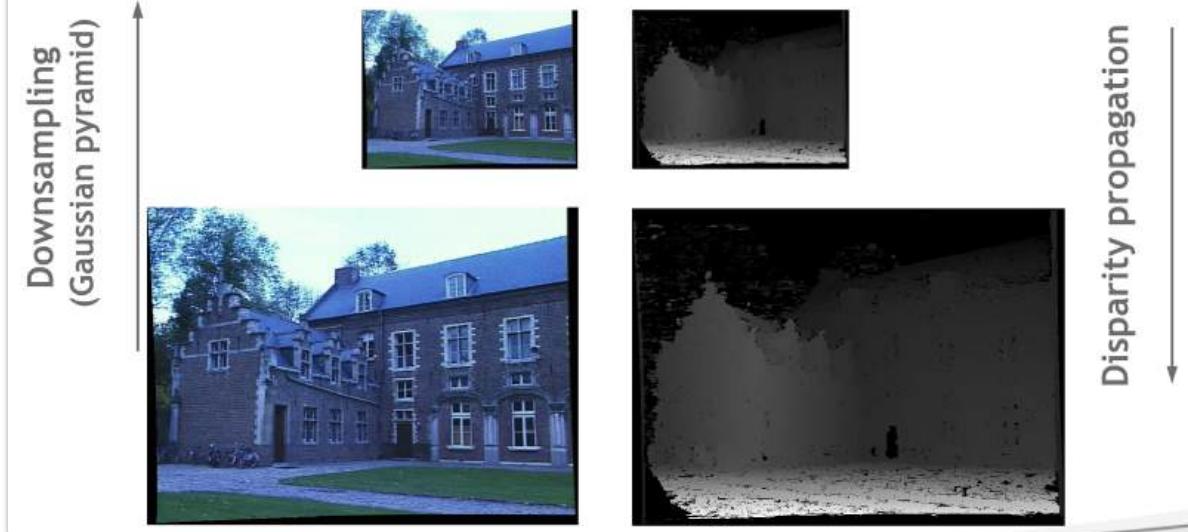


Schritte des hierarchischen Matchings (basierend auf der oberen Abbildung):

- Gaussian Pyramids:** Erstellung von Bildpyramiden für das linke und rechte Bild. Höhere Ebenen sind dabei niedrigere Auflösungsversionen der Originalbilder.
- Compute disparities of level 1:** Berechnung der Disparitäten auf der obersten (niedrigsten Auflösungs-) Ebene der Pyramide. Dies ist recheneffizienter und kann größere Disparitätsbereiche abdecken.
- Project values:** Die auf der höheren Ebene berechneten Disparitäten werden auf die nächstniedrigere Ebene projiziert und dienen dort als initiale Schätzwerte für die Disparitätssuche.
- Compute disparities of lower level:** Die Disparitäten werden auf der niedrigeren Ebene (mit höherer Auflösung) verfeinert, indem um die projizierten Schätzwerte herum gesucht wird. Dieser Prozess wird für alle Ebenen der Pyramide wiederholt, bis zur Originalauflösung.

tion

ity ranges



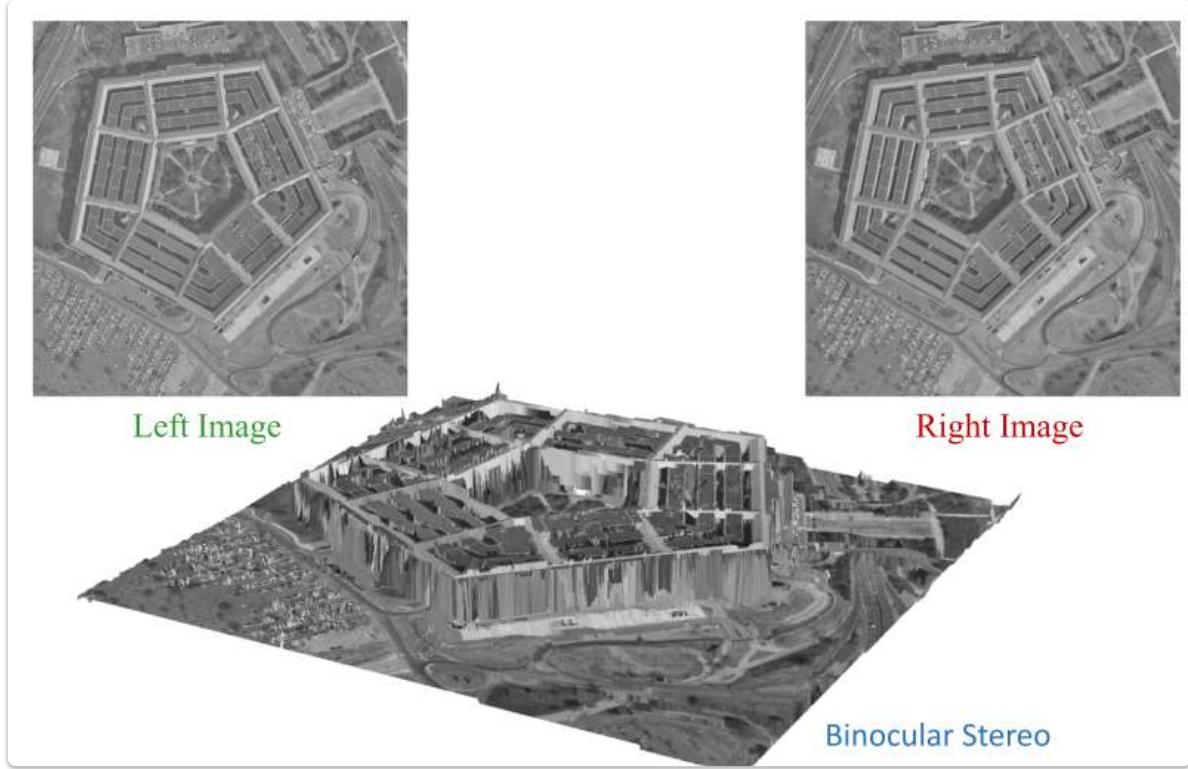
Vorteile des hierarchischen Stereo Matchings (basierend auf der unteren Abbildung):

- **Allows faster computation:** Die Suche nach Korrespondenzen beginnt auf einer niedrigen Auflösung, was die Anzahl der zu vergleichenden Pixel reduziert und somit die Berechnungszeit beschleunigt.
- **Deals with large disparity ranges:** Die grobe Suche auf niedriger Auflösung kann große Disparitätsunterschiede erfassen. Die anschließende Verfeinerung auf höheren Auflösungen ermöglicht eine präzisere Disparitätsschätzung.

Zusammenfassend: Hierarchisches Matching nutzt den Pyramidenansatz, um die Effizienz und den Suchbereich des Stereo Matchings zu verbessern. Durch die schrittweise Verfeinerung der Disparitäten von groben zu feinen Auflösungen können sowohl Rechenzeit reduziert als auch größere Disparitätsbereiche akkurat behandelt werden.

## Beispiele





## Merkmalbasiertes Matching

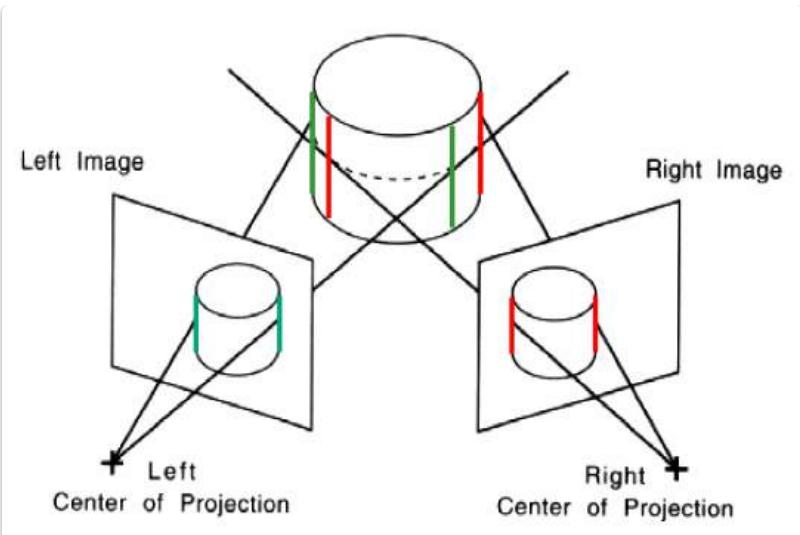
[EVC\\_Skriptum\\_CV, p.53](#)

Problem des ABM (Regionenbasiertes Matching):

- Homogene Bildbereiche enthalten sehr wenig Information.
- Werden aber in die Berechnung miteinbezogen und können zu Fehlern führen.

Ansatz des merkmalbasierten Matching:

- Verwendet einzelne, ausgewählte Pixel (Merkmale), die sich gut zuordnen lassen.
- Merkmale werden aus jedem Bild individuell extrahiert, bevor sie verglichen werden.
- Lokale Merkmale können Ecken, Kanten oder andere *Interest Points* sein.
- Diese Interest Points werden durch lokale Operatoren wie z.B. Moravec oder SIFT extrahiert.



Vorteil des merkmalbasierten Matching:

- Der eigentliche Korrespondenzvergleich kann schneller durchgeführt werden, da bei der Merkmalsextraktion eine wesentliche Datenreduktion stattfindet.

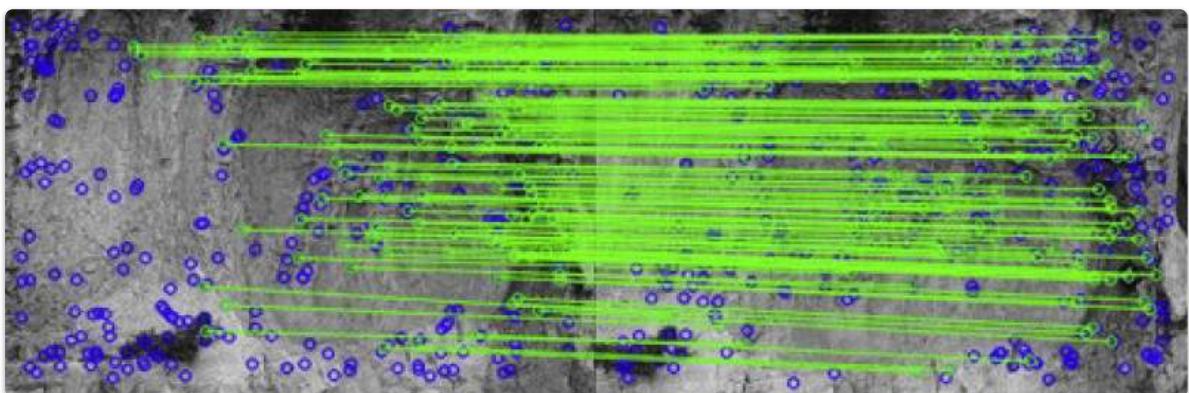
Nachteil des merkmalbasierten Matching:

- Der Hauptnachteil dieser Verfahren liegt aber darin, dass man nur zuverlässige Tiefeninformationen für diese ausgewählten Merkmale erhält (kein dichtes Tiefenbild wie bei ABM).
- Die Bereiche zwischen den Interest Points bleiben zunächst unberücksichtigt und müssen ggf. weiteren Verarbeitungen unterzogen werden (z.B. Interpolation).

### Matching Criteria (Merkmalsbasiertes Matching)

Verwendete Merkmale:

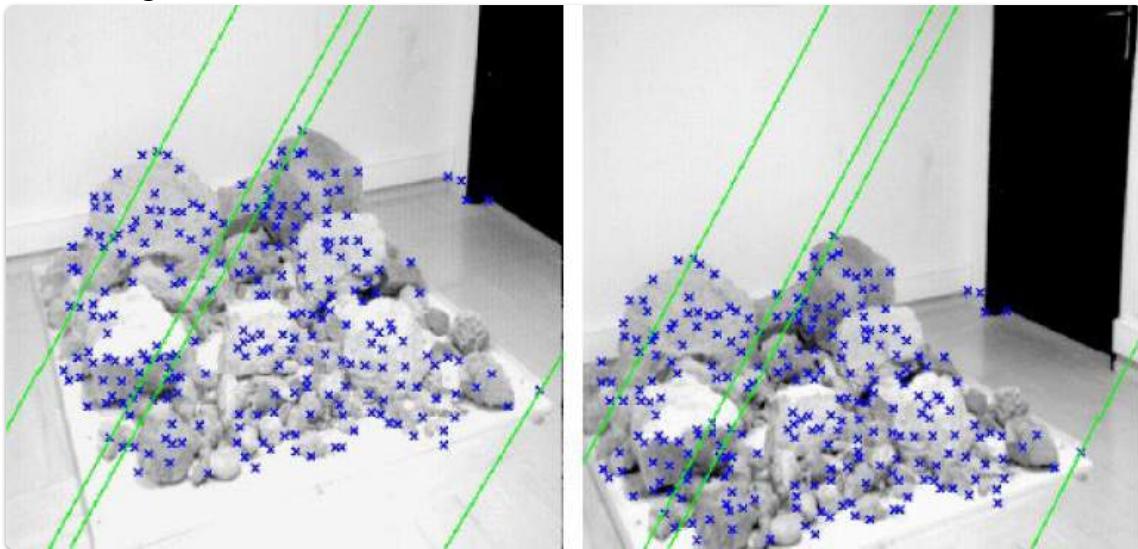
- "Corner"-artige Features (z.B. aus [Zhang, ...])
- Kanten (Edges) [viele Personen...]
- Gradienten [Seitz 89; Scharstein 94]
- Interest Points (z.B. SIFT)



### Feature-based Stereo

Prozess:

- **Match "corner" (interest) points:** Zuerst werden markante Punkte in beiden Bildern detektiert und einander zugeordnet (gematcht).
- **Interpolate complete solution:** Da nur für die Merkmale Tiefeninformationen vorhanden sind, muss die Tiefenkarte für die übrigen Bildbereiche interpoliert werden, um eine vollständige Tiefenkarte zu erhalten.



## Structure-from-Motion (SfM)

[EVC\\_Skriptum\\_CV, p.54](#)

Definition:

- Bezeichnet den Prozess der Gewinnung dreidimensionaler Informationen von Objekten oder einer ganzen Szene durch die Auswertung einer zeitlichen Folge von Bildern.

Zentrales Problem:

- Bestimmung der **Richtung** und des **Ausmaßes** der Kamerabewegung.
- Ansätze zur Lösung:
  - **Motion Field estimation:** Schätzung des Bewegungsfeldes der Punkte im Bild über die Zeit.
  - **Motion Field analysis:** Analyse des geschätzten Bewegungsfeldes zur Bestimmung der Kamerabewegung und der Szenenstruktur.
- Verschiebung des Blickpunkts führt zur scheinbaren Bewegung von Objekten in der Szene.

Unterschied zur Stereo Vision:

- Im Gegensatz zur Stereo Vision ist die Kamerageometrie (die relative Position und Orientierung der Kameras zueinander) zunächst **nicht bekannt**.

Vorgehensweise:

- Anhand der gefundenen Korrespondenzen werden Bewegungsfelder geschätzt.
- Diese Bewegungsfelder können dazu verwendet werden, die Kamerabewegungen zu bestimmen.
- Aus den Kamerabewegungen und den korrespondierenden Punkten kann dann die 3D-Struktur der Szene rekonstruiert werden.

Informationen, die aus der Bewegung gewonnen werden können:

- Information über die Bewegung des Betrachters (der Kamera).
- Tiefeninformationen der Umgebung (vgl. Stereo Vision).
- *Motion Parallax* = *Motion Disparity*: Nahe Objekte scheinen sich bei Bewegung des Betrachters schneller relativ zu entfernten Objekten zu bewegen.



## Bewegungsfeld

[EVC\\_Skriptum\\_CV, p.54](#)

Beobachtung:

- Fixiert ein Beobachter den Horizont, so scheinen sich Mond, Sterne und die gesamte obere Gesichtssphäre zu bewegen.
- Welt und Erdboden scheinen in einem kontinuierlichen Strom vorbeizuziehen.

Relativgeschwindigkeit und Entfernung (Helmholtz, 1950):

- Projiziert man die Umgebung auf eine Abbildungsebene vor dem Betrachter, so ist die relative Geschwindigkeit eines Objekts *umgekehrt proportional* zu seiner Entfernung vom Betrachter.
- Je weiter ein Objekt vom Betrachter entfernt ist, desto geringer ist dessen Bewegung.
- Sterne und der Mond bewegen sich nicht, die Straße direkt neben dem Beobachter bewegt sich sehr schnell.

Richtung des Bewegungsvektors:

- Die Richtung des Bewegungsvektors eines Ortspunktes ist abhängig von der Lage dieses Ortspunktes zum Betrachter.

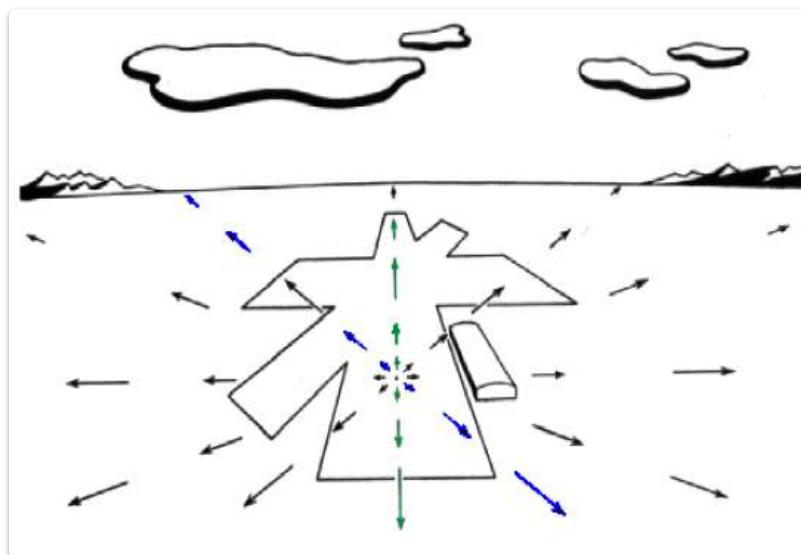
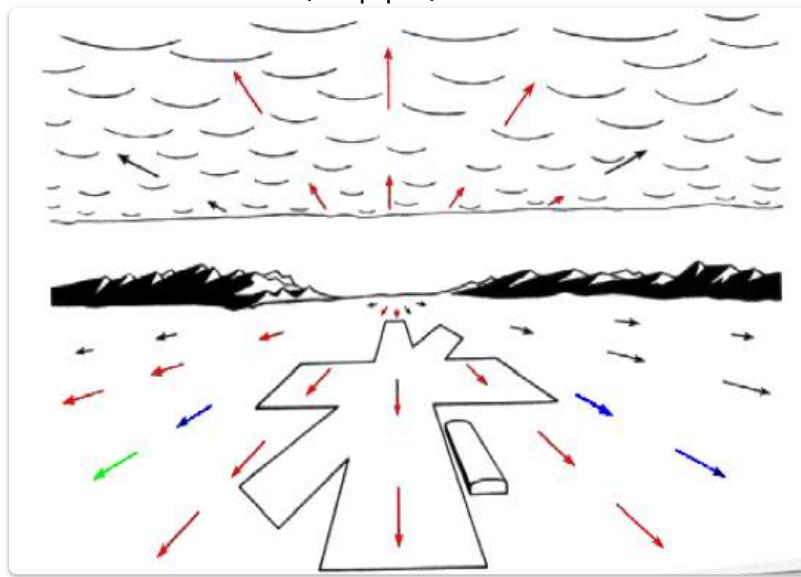
- In einer Umgebung, in der sich die Objekte zueinander nicht bewegen, kann die Eigenbewegung berechnet und eine relative Tiefenkarte der Umgebung erstellt werden.

Bestandteile des Bewegungsfeldes:

- Das Bewegungsfeld besteht aus den einzelnen Bewegungsvektoren aller Punkte im Bild.

Kamera ohne Rotation (Translation):

- Bewegt sich die Kamera ohne zu rotieren, bewirkt dies ein "nach außen" oder "nach innen" Zeigen aller Vektoren zu einem einzigen Punkt.
- Dieser Punkt wird *Focus of Expansion (FoE)* (bei Vorwärtsbewegung) oder *Focus of Contraction (FoC)* (bei Rückwärtsbewegung) genannt.
- Dieser Punkt befindet sich dort, wo sich die Verschiebungsvektor der Kamera mit der Bildebene schneidet (= Epipol).



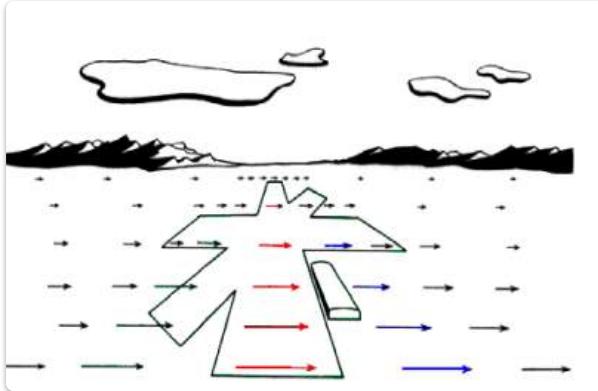
Größe der Bildbewegung eines Szenenpunktes (bei Kamerabewegung):

- Hängt *umgekehrt proportional* von der Entfernung eines Punktes zur Kamera ab.

- Hängt *direkt proportional* vom Sinus des Winkels zwischen der Richtung, in der dieser Szenenpunkt liegt, und der Richtung, in welcher die Kamera verschoben wird, ab.

Berechnung von Kamerabewegung:

- Damit können somit die Richtung der Kamerabewegung (FoE bzw. FoC) und der Betrag der Bewegung berechnet werden.
- Dies führt dann über die Epipolargeometrie zur Stereorekonstruktion (obwohl hier nur eine einzelne bewegte Kamera verwendet wird, nicht zwei gleichzeitig).



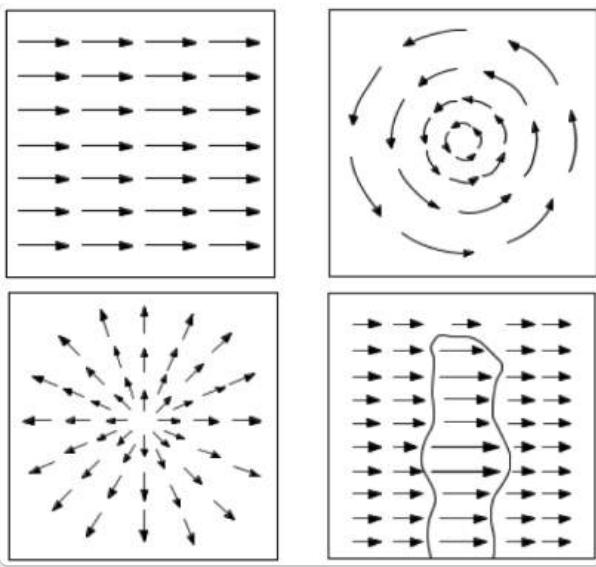
## Bewegungsfeld: Schätzung

Herausforderungen:

- **Sparsely occupied vector field:** Das resultierende Bewegungsfeld kann spärlich sein, d.h., nicht für jeden Pixel ist eine Bewegungsinformation vorhanden (insbesondere in homogenen Bereichen).
- **Same problem as stereo - just the moving direction of the camera is not known:** Das grundlegende Problem der Korrespondenzfindung ist ähnlich wie beim Stereo Matching, jedoch ist die relative Bewegung (die "Basislinie" und Ausrichtung) der Kamera zwischen den Zeitpunkten unbekannt.
- **Epipolar line not known in the beginning:** Da die Kamerabewegung unbekannt ist, sind auch die Epipolarlinien zunächst nicht bekannt, was die Suche nach korrespondierenden Punkten erschwert.

Ansätze zur Korrespondenzfindung zwischen Bildern in einer Sequenz:

- **High temporal sampling = low differences:** Bei einer hohen Bildrate (geringer Zeitabstand zwischen den Bildern) sind die Unterschiede zwischen aufeinanderfolgenden Bildern geringer, was die Korrespondenzsuche erleichtern kann.
- **Either unchanged intensities in both images or unchanged edges in both images:** Annahme, dass entweder die Intensitäten der korrespondierenden Punkte oder deren lokale Struktur (z.B. Kanten) sich zwischen den aufeinanderfolgenden Bildern nicht wesentlich ändern. Diese Annahme kann zur Einschränkung der Suche verwendet werden.



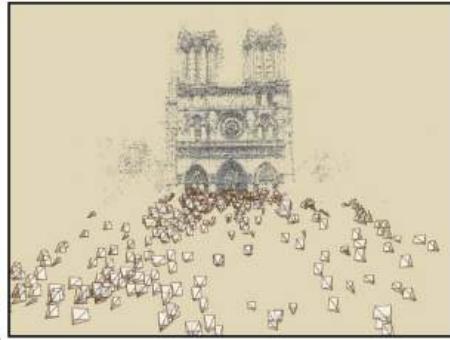
## Multi View Geometry

[EVC\\_Skriptum\\_CV, p.54](#)

### Definition:

- Eine weitere Möglichkeit zur 3D-Rekonstruktion, bei der mehr als zwei beliebige Bilder verwendet werden, um die Relation der Bildpunkte und der Punkte im 3D-Raum zu errechnen.

Structure from Motion (SfM)

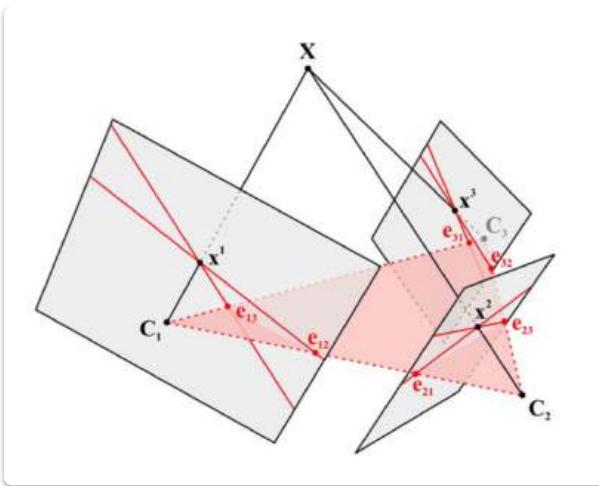


Dense multiview stereo



### Vorteil gegenüber Stereo mit nur zwei Bildern:

- Betrachtet man den Fall mit 3 Bildern: Wenn ein Objektpunkt in zwei Bildern identifiziert wurde, so kann seine geometrische Position im dritten Bild durch den Schnitt der entsprechenden Epipolar-Geraden vorhergesagt werden (siehe Abbildung 45).
- Im Unterschied zum Bildpaar existiert beim Bildtripel jedoch ein eindeutiges Ergebnis.
- Von den 2 oder 3 Basisbildern ausgehend, können bei der Mehrbildgeometrie dann stufenweise mehr Bilder der Szene aus mehreren von verschiedenen Orten aus aufgenommenen Bildern hinzugenommen werden.
- Durch Hinzunahme von mehr Bildern entsteht eine stabilere 3D-Punktwolke.
- Die Durch Bündelausgleich verbessert werden kann.



## Bündelausgleich (Bundle Adjustment):

- Stammt aus der Photogrammetrie.
- Erlaubt die gleichzeitige Bestimmung der internen und externen Kameraparameter sowie der 3D-Struktur der Szene aus mehreren verschiedenen Ansichten.
- Grundlegende Annahmen für den Bündelausgleich sind die Starrheit der Szene zwischen den einzelnen Ansichten und die Erfüllung der Kollinearitätsgleichung (Collinearity Condition).
- Kollinearitätsgleichung: Besagt, dass ein betrachteter 3D-Objektpunkt, sein zugehöriger Bildpunkt und das Projektionszentrum der Kamera auf einer gemeinsamen Gerade liegen müssen.

## Zielsetzung des Bündelausgleichs:

- Gleichzeitige Variation der 3D-Koordinaten der einzelnen Ansichten und der Transformationsparameter der einzelnen Ansichten.
- Ziel ist, eine möglichst gute Übereinstimmung zwischen erwarteten und gemessenen Bildpunkten zu erreichen.

## Zentralprojektion:

- Im Falle einer Zentralprojektion erzeugt jeder Szenenpunkt einen Strahl durch das Projektionszentrum.
- Für die Menge aller Punkte entsteht ein Strahlenbündel, welches im Projektionszentrum geschnürt wird.

## Structure from Motion

### Grundlagen:

- Gegeben viele korrespondierende Punkte über mehrere Bilder hinweg,  $\{(u_{ij}, v_{ij})\}$ .
- Ziel ist die simultane Berechnung der 3D-Positionen der Punkte  $\mathbf{x}_i$  und der Kamera- (oder Bewegungs-) Parameter  $(K, R_j, \mathbf{t}_j)$ .
  - $K$ : Intrinsische Kameraparameter (Kalibrierungsmatrix).

- $R_j$ : Rotationsmatrix der Kamera im  $j$ -ten Bild.
- $\mathbf{t}_j$ : Translationsvektor der Kamera im  $j$ -ten Bild.
- Die Bildkoordinaten  $(u_{ij}, v_{ij})$  sind eine Funktion der Kameraparameter und der 3D-Punktpositionen:

$$u_{ij} = f(K, R_j, \mathbf{t}_j, \mathbf{x}_i)$$

$$v_{ij} = g(K, R_j, \mathbf{t}_j, \mathbf{x}_i)$$

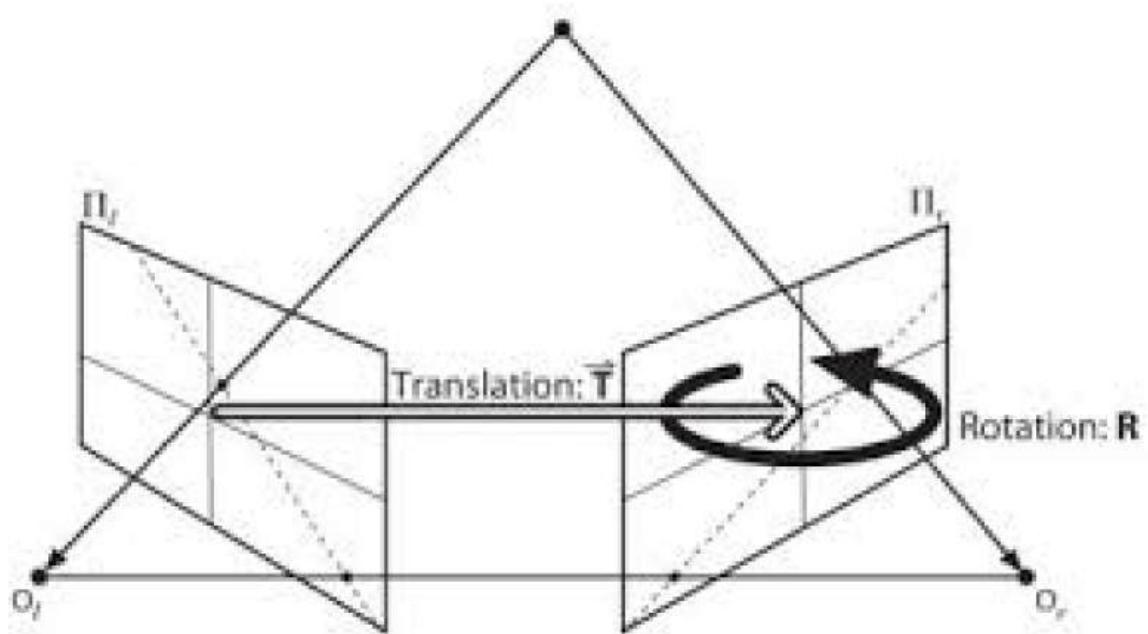
Hauptvarianten:

- **Kalibriert:** Die intrinsischen Kameraparameter  $K$  sind bekannt.
- **Unkalibriert:** Die intrinsischen Kameraparameter  $K$  sind unbekannt (manchmal assoziiert mit euklidischen und projektiven Rekonstruktionen).

## Automatische Berechnung von F (Fundamentale Matrix - typisch für unkalibrierte SfM)

Schritte:

1. **Interest points:** Detektion von markanten Punkten in den Bildern.
2. **Putative correspondences:** Erstellung initialer, potenzieller Korrespondenzen zwischen den Bildern (können fehlerhaft sein).
3. **RANSAC (RANdom SAmple Consensus):** Robustes Schätzverfahren zur Entfernung von Ausreißern (falschen Korrespondenzen) und zur Schätzung der Fundamentalen Matrix  $F$ .
4. **Non-linear re-estimation of F:** Nicht-lineare Optimierung zur Verfeinerung der Schätzung der Fundamentalen Matrix  $F$ .
5. **Guided matching:** Verwendung der geschätzten Epipolargeometrie (aus  $F$ ) zur Verbesserung der Korrespondenzsuche.
6. **Repeat (4.) and (5.) until stable:** Iterativer Prozess der Verfeinerung der Fundamentalen Matrix und der Korrespondenzen, bis eine stabile Lösung erreicht ist.



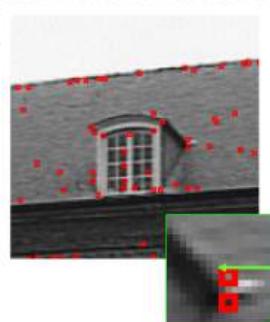
## Beispiel



Select strongest features (e.g. 1000/image)

Evaluate SSD and SAD for all features with similar coordinates

e.g.  $(x', y') \in [x - \frac{w}{10}, x + \frac{w}{10}] \times [y - \frac{h}{10}, y + \frac{h}{10}]$



Keep mutual best matches  
Still many wrong matches!



## RANSAC



Step 1. Extract features

Step 2. Compute a set of potential matches

Step 3. do

Step 3.1 select minimal sample (i.e. 7 matches)

Step 3.2 compute solution(s) for F

Step 3.3 determine inliers

(verify hypothesis)

}

(generate hypothesis)

until  $\Gamma(\#inliers, \#samples) < 95\%$

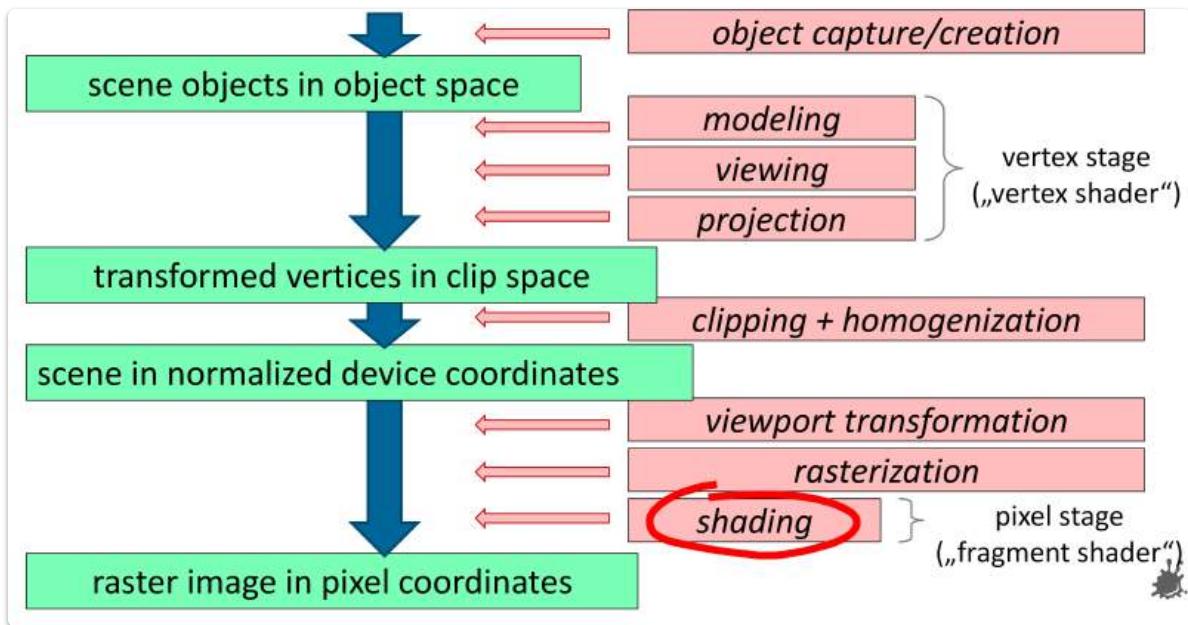
- Step 4. Compute F based on all inliers
- Step 5. Look for additional matches
- Step 6. Refine F based on all correct matches

$$\Gamma = 1 - \left(1 - \left(\frac{\#inliers}{\#matches}\right)^7\right)^{\#samples}$$

#Inliers	90%	80%	70%	60%	50%
#samples	5	13	35	106	382

# 11. Globale Beleuchtung und Texturen

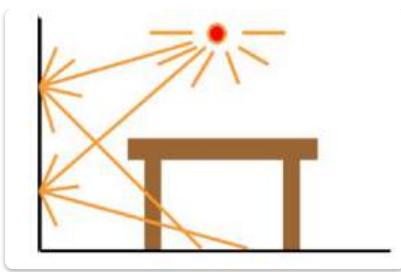
## Rendering Pipeline



## Radiosity

EVC\_Skriptum\_CG, p.43

- **Ursprung**: Wärmeübertragung.
- **Modellierung**: Lichtausbreitung unter Beachtung des Energiegleichgewichts in einem geschlossenen System.
- **Beschreibung**: Physikalischer Vorgang der Ausbreitung von Licht in einer diffus reflektierenden Umgebung.
- **Ziel**: Berechnung der Helligkeit aller Flächen einer Szene unter Berücksichtigung der gegenseitigen Beeinflussung.
- **Effekt**: Auch Flächen, die nicht direkt beleuchtet sind, erhalten eine gewisse Helligkeit (indirekte Beleuchtung).
- **Sekundäre Lichtquellen**: Jeder beleuchtete Gegenstand wirkt als sekundäre Lichtquelle und strahlt Licht in die Umgebung ab.
- **Bildgenerierung**:
  - Zuerst wird die Lichtausbreitung im Raum berechnet, **ohne** dass die Kameraposition bekannt ist.
  - Vereinfachende Annahme: Der Beobachter beeinflusst die Ausbreitung des Lichts nicht.
  - **Vorteil**: Objekte können dann aus verschiedenen Richtungen dargestellt werden, ohne dass die Lichtausbreitung jedes Mal neu berechnet werden muss.



## Die Radiosity Gleichung

EVC\_Skriptum\_CG, p.43

- **Szenenbeschreibung:** Besteht aus  $n$  ebenen Polygonen, beim Radiosity-Verfahren als Patches bezeichnet.
- **Patch-Eigenschaften:**
  - Jedes Patch  $P_i$  ist homogen.
  - Perfekt diffuse Oberfläche (Licht wird in alle Richtungen gleichmäßig abgestrahlt).
  - Lichtquellen sind ebenfalls Patches.
- **Radiosity  $B_i$  von Patch  $P_i$ :**
  - Gesamte abgestrahlte Energie pro Flächeneinheit.
  - Summe aus Eigenemission und reflektierter Leistung pro Flächeneinheit.
  - Lichtenergiedichte ist proportional zur wahrgenommenen Helligkeit.
- **Vereinfachende Annahme:** Radiosity ist für alle Positionen auf einem Patch gleich.
- **Die Radiosity Gleichung:**

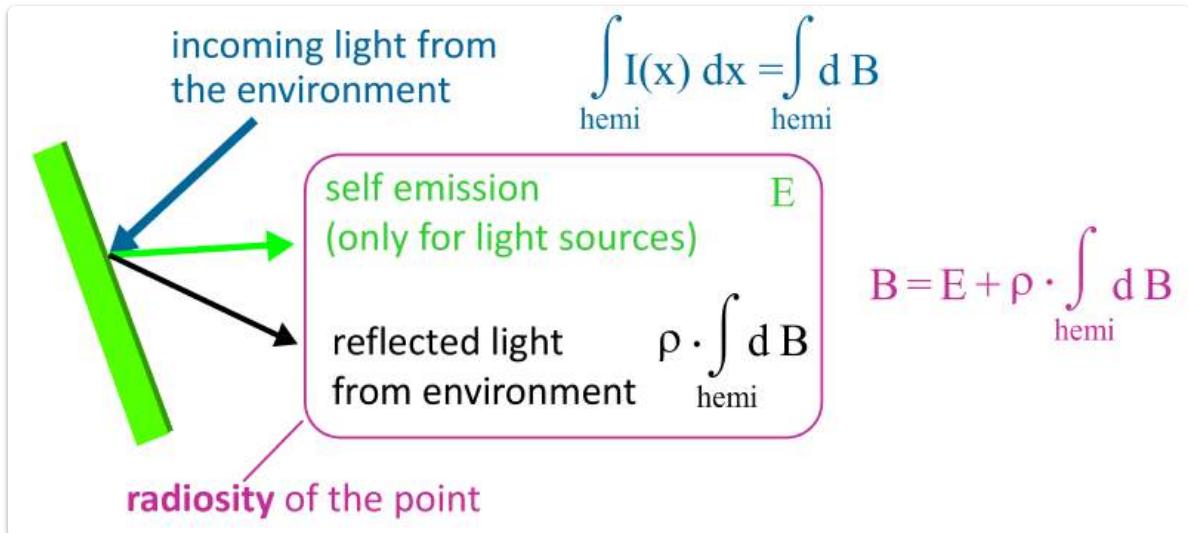
$$B_i = E_i + \rho_i \sum_{j=1}^n B_j F_{ij}$$

- $B_i$ : Radiosity des Patches  $i$ .
- $E_i$ : Eigenemission des Patches  $i$ .
- $\rho_i$ : Diffuser Reflexionskoeffizient der Oberfläche von Patch  $i$  (gibt an, wie viel % des einfallenden Lichts diffus reflektiert wird, auch Albedo genannt).
- $n$ : Anzahl der Patches in der Szene.
- $B_j$ : Radiosity aller anderen Patches  $j$ .
- $F_{ij}$ : Formfaktor (view factor) zwischen Patch  $i$  und Patch  $j$ .
  - Gibt an, welcher Anteil der Radiosity von Patch  $j$  auf Patch  $i$  wirkt (ist gleich dem Anteil der Radiosity von  $i$ , die auf  $j$  trifft, aufgrund des Reziprozitätsgesetzes).
  - Geometrische Größe, unabhängig von Lichtquellen oder Radiositywerten.
- **Gleichungssystem:** Die Radiosity-Formeln für  $n$  Patches ergeben ein lineares Gleichungssystem mit  $n$  Unbekannten  $B_i$ . (kann nur iterativ gelöst werden)
- **Matrix-Form des Gleichungssystems:**

$$B_i - \rho_i \sum_{j \neq i} B_j F_{ij} = E_i$$

$$\begin{pmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

## Herleitung der Gleichung



to calculate the light influence between surfaces

**Radiosity = total light leaving a surface point**

$$B = E + \rho \cdot \int_{hemi} d B$$

$B$  ... radiosity       $\text{hemi}$  ... half space over point

$E$  ... self emission       $\rho$  ... reflection coefficient

“radiosity = self emission + reflection property · sum of all incoming light”

- diffuse interreflections in a scene
- radiant energy transfers
- conservation of energy, closed environments
- subdivision of scene into *patches* with constant radiosity  $B_i$

$$B = E + \rho \cdot \int_{\text{hemi}} d B$$

the scene is discretized into **n "patches"** (plane polygons)  $P_i$ , for each of these patches a constant radiosity  $B_i$  is assumed:

$$B = E + \rho \cdot \int_{\text{hemi}} d B \quad \Rightarrow \quad B_i = E_i + \rho_i \cdot \sum_{j=1}^n B_j \cdot F_{ij}$$

$\rho_i$  diffuse reflection coefficient of patch **i**

$F_{ij}$  “form factor”: describes what % of the influence on patch **i** comes from patch **j**;  
= geometric size !

$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

$B_i$  ... radiosity of patch **i**

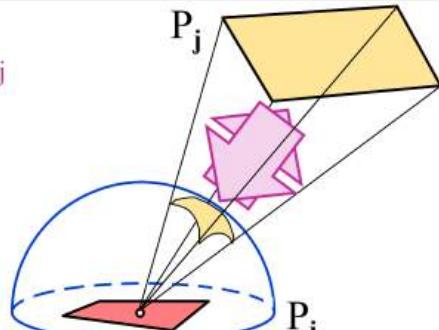
$E_i$  ... self-emission of patch **i**

$\sum B_j F_{ij}$  ... contribution of other patches

$F_{ij}$  ... form factor, defines

- contribution of  $B_i$  on patch **j** - which is equal to
- contribution of patch **j** to  $B_i$

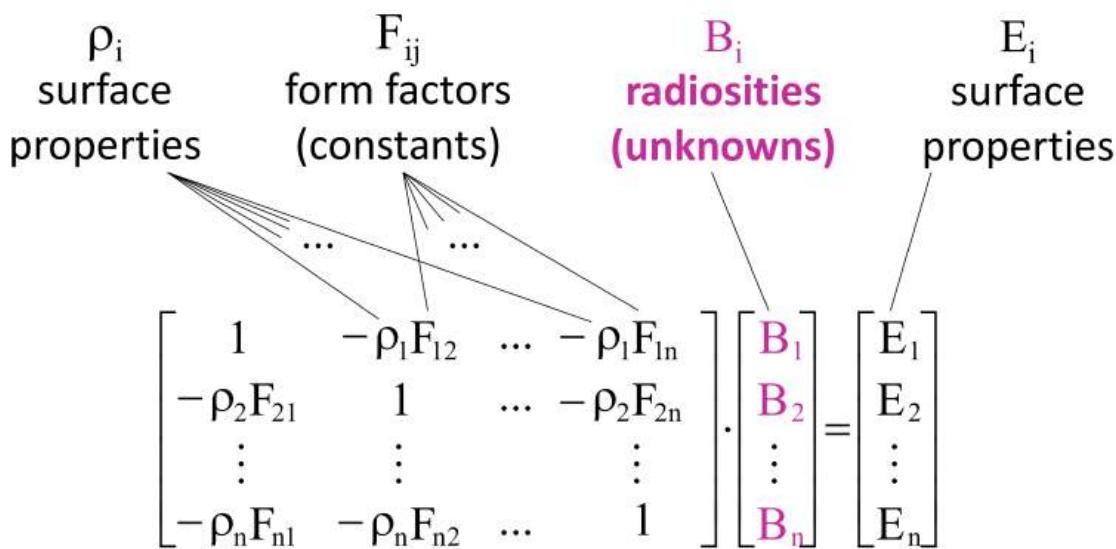
$\rho_i$  ... reflectivity coefficient of patch **i** (“*albedo*”)



$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

$$B_i - \rho_i \sum_{j \neq i} B_j F_{ij} = E_i$$

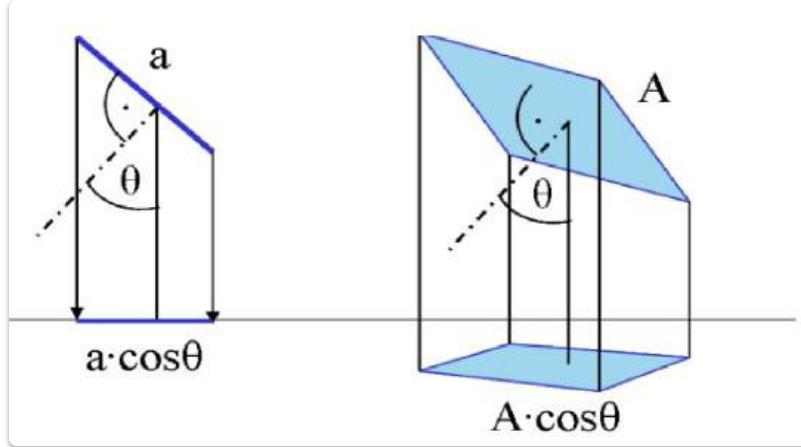
$$\begin{bmatrix} 1 & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \dots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$



## Berechnung der Formfaktoren

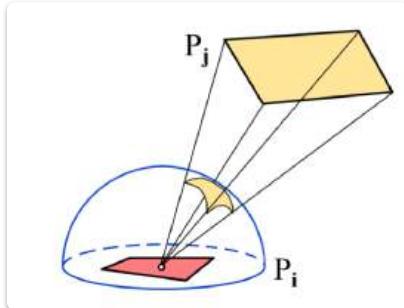
[EVC\\_Skriptum\\_CG, p.43, p.44](#)

- **Geometrischer Zusammenhang:** Die Fläche der Normalprojektion einer Fläche  $A$  auf eine andere Fläche verkleinert sich um den Kosinus des Winkels  $\theta$  zwischen den Flächen:  $A \cdot \cos \theta$ .



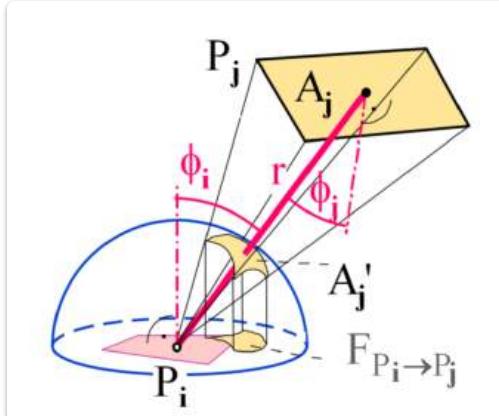
- **Definition des Formfaktors  $F_{ij}$ :**

- Anteil der vom Patch  $P_j$  ausgehenden Strahlungsenergie, die den Patch  $P_i$  trifft.
- Oder: Wie viel Prozent der Energie von  $P_j$  trifft auf  $P_i$ ?
- Reziprok: Gibt auch an, wie viel Prozent der auf  $P_i$  eingehenden Energie von  $P_j$  kommt.



- **Herleitung der Formel für  $F_{ij}$  (vereinfachte Annahme: Patches klein im Verhältnis zum Abstand  $r$ ):**

1. Betrachte Patch  $P_i$  mit Fläche  $A_i$ .
2. Stelle dir über  $P_i$  eine Halbkugel (Hemisphäre) mit Radius 1 vor, auf die  $P_j$  projiziert wird.
3. Die projizierte Fläche  $A'_j$  hat die Größe  $A_j \cos \phi_j$ , wobei  $\phi_j$  der Winkel zwischen der Normalen von  $P_j$  und der Verbindungsgeraden zwischen den Patches ist.



4. Energie, die unter einem Winkel  $\phi_i$  auf  $P_i$  einfällt, ist proportional zu  $\cos \phi_i$ . Multiplikation mit  $\cos \phi_i$  (entspricht einer Projektion auf die Grundfläche der Halbkugel) liefert den korrekten Anteil des Einflusses von Patch  $P_j$  auf  $P_i$ .
5. Normierung durch die Größe der Grundfläche der Halbkugel ( $1^2\pi = \pi$ ).

- **Formel für den Formfaktor  $F_{ij}$ :**

$$F_{ij} = \frac{\cos \phi_i \cos \phi_j A_j}{\pi r^2}$$

- **Wichtige Voraussetzung:** Diese Formel gilt unter der Annahme, dass sich keine Hindernisse zwischen den beiden Patches befinden und das Licht ungehindert von  $P_j$  nach  $P_i$  gelangen kann. Korrekte Formfaktoren müssen auch die gegenseitige Sichtbarkeit berücksichtigen.
- $\phi_i$ : Winkel zwischen der Normalen von  $P_i$  und der Verbindungsgeraden.
- $\phi_j$ : Winkel zwischen der Normalen von  $P_j$  und der Verbindungsgeraden.
- $A_j$ : Fläche des Patches  $P_j$ .
- $r$ : Abstand zwischen den Patches.
- **Reziprozitätsprinzip:** Die Abhängigkeit der Formfaktoren zwischen zwei Patches setzt in Beziehung:

$$A_i F_{ij} = A_j F_{ji}$$

form factor  $F_{ij}$ : contribution of patch  $P_j$  to  $B_i$  = contribution of  $B_i$  to patch  $P_j$

### form factor properties:

■ conservation of energy

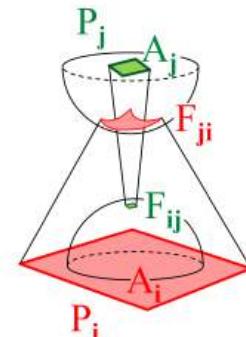
$$\sum_{j=1}^n F_{ij} = 1$$

■ uniform light reflection

$$A_i F_{ij} = A_j F_{ji}$$

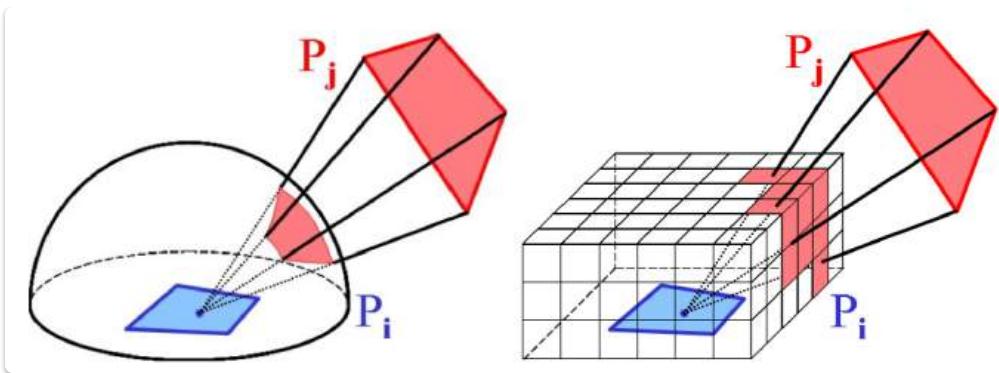
■ no self-incidence

$$F_{ii} = 0$$



### Praktische Berechnung:

- **Hemicube-Methode**
  - Abschätzung der Formfaktoren statt einer Halbkugel über  $P_i$  wird ein Halbwürfel (Hemicube) verwendet.
  - Die gesamte Szene wird auf diesen Hemicube abgebildet (z-Buffer-Technologie).
  - Die Oberfläche des Halbwürfels wird in Pixel aufgeteilt.
  - Alle anderen Patches werden von  $P_i$  aus mit dem Würfelmittelpunkt als Projektionszentrum auf diese Pixel abgebildet.
  - Für jedes Pixel wird sein Formfaktor bestimmt.
  - Für jedes Patch wird die Summe der Formfaktoranteile seiner Pixel gebildet.
- **Ray-Tracing-Verfahren:** Alternative Methode zur Berechnung von Formfaktoren.



## Fortschreitende Verfeinerung (Progressive Refinement)

EVC\_Skriptum\_CG, p.44

$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

"shooting" → select brightest patch  $P_i$  and distribute its radiosity  $B_i$

$$B_{j \text{ due to } B_i} = \rho_j B_i F_{ij} \frac{A_i}{A_j}$$

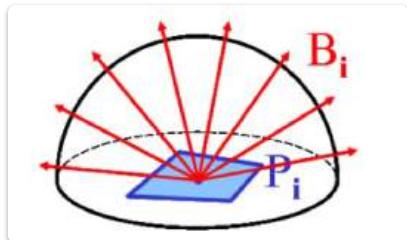
- **Ziel:** Iterative Lösung des Radiosity-Gleichungssystems.
- **Grundidee ("Shooting"):** Man wählt das jeweils hellste Patch aus und "schießt" (verteilt) dessen noch nicht abgestrahlte Energie auf alle anderen Patches.
- **Vorteil:**
  - In jedem Schritt werden alle Patches etwas besser beleuchtet.
  - Das Verfahren konvergiert schnell, da immer die Energie der hellsten Patches zuerst verteilt wird.
- **Radiosity-Anteil von  $P_i$ , der von  $P_j$  verursacht wird ( $B_{(i \text{ von } B_j)}$ ):**
  - Dies beschreibt, wie viel Radiosity von Patch  $j$  zu Patch  $i$  beiträgt.
  - Es gilt:  $B_{(i \text{ von } B_j)} = \rho_i B_j F_{ij}$
  - **Symmetrie:** Der Einfluss von  $P_i$  auf  $P_j$  ist symmetrisch zum Einfluss von  $P_j$  auf  $P_i$ .
  - Es gilt auch:  $B_{(j \text{ von } B_i)} = \rho_j B_i F_{ji}$
- **Verknüpfung mit dem Reziprozitätsprinzip:**
  - Aus dem Reziprozitätsprinzip  $A_i F_{ij} = A_j F_{ji}$  folgt, dass  $F_{ij} = F_{ji} \frac{A_j}{A_i}$ .
  - Setzt man dies in die Gleichung für  $B_{(i \text{ von } B_j)}$  ein:  

$$B_{(i \text{ von } B_j)} = \rho_i B_j F_{ij} = \rho_i B_j F_{ji} \frac{A_j}{A_i}$$
  - Diese Beziehung ermöglicht es, den Formfaktor  $F_{ij}$  direkt zu nutzen, auch wenn die Berechnung des Beitrags von  $P_j$  zu  $P_i$  erfolgt.
- **Speicherhaltung pro Patch:**

- **Gesammelte Radiosity ( $B_i$ ):** Der bisher beste Schätzwert für die Radiosity des Patches.
- **Noch nicht verschossene Radiosity ( $\Delta B_i$ ):** Die Energie, die dieses Patch noch abgeben muss und die als Basis für die Auswahl des nächsten "hellsten" Patches dient.
- **Initialisierung:**

- Am Anfang werden  $B_i$  und  $\Delta B_i$  mit der Eigenemission  $E_i$  des Patches initialisiert:  

$$B_i = \Delta B_i = E_i \text{ für alle Patches } i.$$



## Iterationsschritt des Progressive Refinement Algorithmus

p.45

```

select patch i with highest  $A_i * \Delta B_i$ 
FOR selected patch i {
    set up hemicube
    calculate form factors  $F_{ij}$ 
}
FOR each patch j {
     $\Delta rad := \rho_j * \Delta B_i * F_{ij} * A_i / A_j$ 
     $\Delta B_j := \Delta B_j + \Delta rad$ 
     $B_j := B_j + \Delta rad$ 
}
 $\Delta B_i := 0$ 

```

- **Bezeichnung:** Dieses Verfahren wird auch als "Progressive Refinement" (schrittweise Verfeinerung) bezeichnet.

Verschiedene Beispiele: [EVC-CG11-Globale Beleuchtung+Texturen\\_2025S\\_Slides](#), p.19

## Aspekte von Radiosity

[EVC\\_Skriptum\(CG\)](#), p.45

- **Blickpunktunabhängigkeit:** Radiosity ist eine Methode zur Berechnung der Helligkeit von einzelnen diffusen Patches, die unabhängig von der Kameraposition (Blickpunkt) ist.
  - **Vorteil:** Die Lichtausbreitung muss nur einmal berechnet werden, danach können Ansichten aus beliebigen Blickpunkten generiert werden, ohne die Beleuchtung neu zu berechnen.

- **Nachfolgender Rendering-Schritt:** Nach der Radiosity-Berechnung ist meist noch ein zusätzlicher Rendering-Schritt notwendig, um das finale Bild zu erzeugen.
  - Oft wird hierfür ein einfaches Polygon-Verfahren mit Gouraud-Schattierung verwendet, um die berechneten Helligkeiten darzustellen.
- **Kombination mit Ray-Tracing:** Die durch Radiosity gewonnenen diffusen Schattierungswerte können als Basiswerte für ein Ray-Tracing-Verfahren verwendet werden.
  - **Vorteil:** Dadurch lassen sich zusätzlich Effekte wie Spiegelungen und scharfe Schatten, die Radiosity allein nicht modelliert, gut darstellen.

radiosity is viewpoint-independent  
needs a rendering step to display

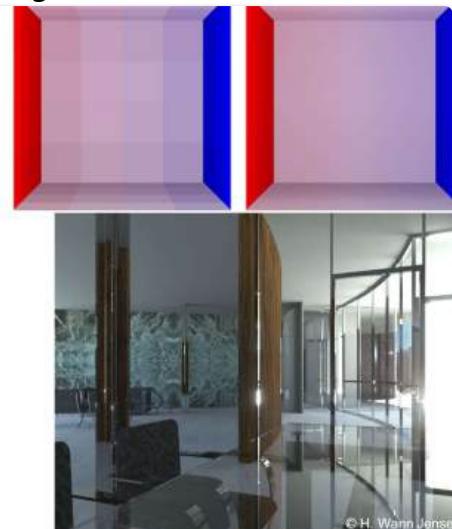
- polygon rendering
- Gouraud shading
- ray-tracing
- ...

combination with ray-tracing enables

- reflections
- shadows
- ...

Werner Purgathofer

23.



© H. Wann Jensen

## Erweiterungen des Radiosity-Prinzips

[EVC\\_Skriptum\\_CG](#), p.45

Das vorgestellte Radiosity-Prinzip ist erweiterbar, um Effizienz und Qualität zu verbessern:

- **Hierarchische Strukturierung von Patches:**
  - **Ziel:** Reduzierung der Anzahl der zu berechnenden Patches.
  - **Methode:** Patches können hierarchisch strukturiert werden. Das bedeutet, dass entfernte Patches nicht einzeln behandelt werden müssen, sondern als Gruppen zusammengefasst werden können. Dies reduziert den Rechenaufwand erheblich.
- **Discontinuity-Meshing:**
  - **Problem:** An Stellen, an denen sich die Beleuchtung abrupt ändert (z.B. Schattenkanten), können zu große Patches eine falsche "Verschmierung" der Beleuchtung verursachen. Umgekehrt können zu kleine Patches den Rechenaufwand unnötig erhöhen.
  - **Lösung:** Discontinuity-Meshing passt die Größe der Patches an die Beleuchtungssituation an. In Bereichen mit starken Helligkeitsunterschieden (z.B. Schattenübergängen) werden kleinere Patches verwendet, um eine präzisere Darstellung zu ermöglichen. In Bereichen mit gleichmäßiger Beleuchtung können größere Patches verwendet werden.

# Path Tracing

---

[EVC\\_Skriptum\\_CG, p.45](#)

- **Grundlage:** Path Tracing ist eine Erweiterung des [Ray-Tracing Verfahrens](#).
- **Alternativname:** Wird auch "*Monte Carlo Ray-Tracing*" genannt.
- **Kernprinzip:**
  - Beim klassischen Ray-Tracing werden an jedem Auftreffpunkt Sekundärstrahlen in alle relevanten Richtungen gelegt (z.B. Reflexion, Brechung, Schatten).
  - Beim Path Tracing wird stattdessen an jedem Auftreffpunkt **zufällig nur eine Richtung** entsprechend einer gültigen Verteilungsfunktion ausgewählt.
  - **Intuition:** Es wird ein "Lichtpfad" (Path) von der Kamera bis zu einer Lichtquelle oder ins Unendliche verfolgt, wobei an jeder Oberfläche nur eine zufällige Richtung für die Weiterverfolgung gewählt wird.
- **Vorteile:**
  - **Inklusion komplexer Lichtverhältnisse:** Ermöglicht die realistische Simulation von Szenen, in denen viele verschiedene Lichtrichtungen relevant sind.
  - **Diffuse Reflexion:** Kann diffuse Reflexionen sehr gut modellieren (im Gegensatz zu klassischem Ray-Tracing, das eher auf spiegelnde oder brechende Oberflächen spezialisiert ist).
  - **Ausgedehnte Lichtquellen:** Kommt besser mit ausgedehnten (nicht punktförmigen) Lichtquellen zurecht.
- **Herausforderungen & Lösungen:**
  - **Rauschen im Ergebnisbild:** Da pro Pixel nur eine zufällige Richtung verfolgt wird, führt dies zu Rauschen im Ergebnis.
  - **Lösung:** Um starkes Rauschen zu reduzieren, müssen pro Pixel **viele Strahlen** (Pfade) berechnet und gemittelt werden. Dies ist rechenintensiv.
- **Mathematischer Hintergrund:**
  - Grundsätzlich entspricht das Vorgehen des Path Tracings der **Monte-Carlo-Integration** eines mehrdimensionalen Integrals.
  - Dieses Integral wird als "Rendering Equation" bezeichnet und beschreibt die vollständige Lichtausbreitung im Raum.
  - **Rendering Equation (vereinfachtes Verständnis):** Eine komplexe mathematische Gleichung, die die gesamte Beleuchtung eines Punktes in einer Szene beschreibt, indem sie alle eingehenden Lichtstrahlen und deren Wechselwirkungen mit der Oberfläche berücksichtigt. Monte Carlo Integration ist eine Methode, um solche komplexen Integrale durch zufällige Stichproben zu approximieren.

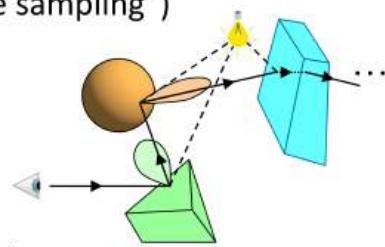
also called Monte Carlo ray tracing

ray directions selected randomly according to

distribution functions ("importance sampling")

uses Monte Carlo integration to solve

$$B = E + \rho \cdot \int_{\text{hemi}} d B$$



**B** ... radiosity      **hemi** ... half space over point

**E** ... self emission       **$\rho$**  ... reflection coefficient

- Qualitätsverbesserung:

- Die Verwendung von **Quasi-Zufallszahlen** (anstelle von Pseudo-Zufallszahlen) reduziert die Varianz (d.h., das Rauschen) im Ergebnisbild merklich.
- Quasi-Zufallszahlen:** Sind keine echten Zufallszahlen, sondern Sequenzen, die eine gleichmäßige Verteilung im Definitionsbereich aufweisen als Pseudo-Zufallszahlen. Dies hilft, die Stichproben für die Monte-Carlo-Integration besser über den gesamten Raum zu verteilen und somit das Ergebnis genauer zu machen.

## Photon Mapping

[EVC\\_Skriptum\\_CG, p.46](#)

- Grundidee:** Eine Methode zur Berechnung globaler Beleuchtungseffekte, die besonders gut mit komplexen Lichtinteraktionen wie Spiegelungen und Kaustiken (Lichtbündel, die durch Brechung oder Reflexion erzeugt werden) umgehen kann.

- Vorgehensweise:

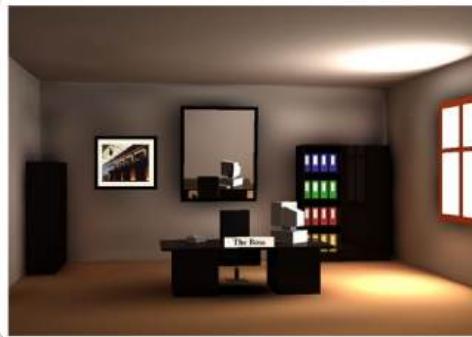
- "Vorwärts"-Verfolgung von Lichtstrahlen:** Im Gegensatz zum Ray-Tracing, das Strahlen von der Kamera verfolgt, werden beim Photon Mapping Lichtstrahlen (genannt "Photonen") von den Lichtquellen ausgesendet und in der Szene verfolgt. Dies ist eine "Vorwärts"-Richtung.
- Speichern der Lichtwirkung an Auftreffpunkten:** Wenn ein Photon auf eine Oberfläche trifft, wird die Wirkung des Lichts (z.B. Energie, Farbe) an diesem Auftreffpunkt gespeichert. Diese gespeicherten Punkte werden als "Photonen" in einer "Photon Map" abgelegt.
- Interpolation für das Aussehen des Objekts:** Später, während des Renderings, werden diese gespeicherten Photonen genutzt. An einem Punkt, für den die Beleuchtung berechnet werden soll, werden die Informationen von mehreren nahegelegenen Photonen interpoliert, um das Aussehen des Objekts zu bestimmen.

- Vorteile:

- Korrekte Berechnung komplexer Lichtsituationen:** Ermöglicht die akkurate Simulation von:
  - Spiegelungen der Lichtquellen:** Wie sich Lichtquellen in spiegelnden

Oberflächen abbilden.

- **Kaustiken:** Die Fokussierung von Licht durch Brechung (z.B. durch Glas) oder Reflexion (z.B. durch eine gewölbte, spiegelnde Oberfläche), die zu hellen Lichtmustern auf anderen Oberflächen führt.
- **Kombination mit anderen Verfahren:**
  - Eine Kombination aus **Path Tracing** und **Photon Mapping** kann nahezu alle Lichteffekte in einem Bild integrieren und so sehr realistische Renderings erzeugen. Path Tracing ist gut für direkte und diffuse Beleuchtung, während Photon Mapping seine Stärken bei Kaustiken und komplexen spiegelnden/refraktiven Pfaden hat.

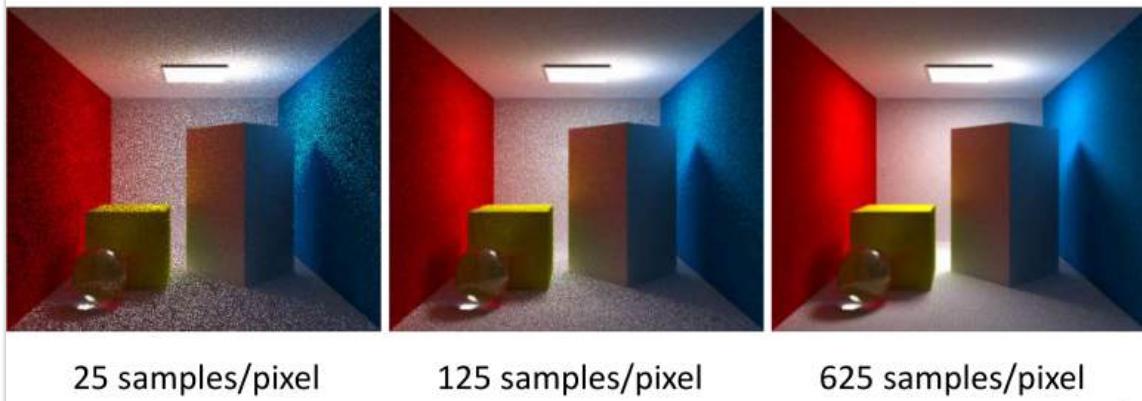


## Optische Eigenschaften natürlicher Oberflächen: Mapping-Methoden

EVC\_Skriptum\_CG, p.46

- **Problem:** Natürliche Oberflächen weisen diverse Unregelmäßigkeiten auf, die entweder durch die Umgebung, eine variable Färbung der Oberfläche oder durch Oberflächenunebenheiten (Schattierung) verursacht werden.
- **Simulationsmethoden ("Mapping"):** Für diese drei Ursachen lassen sich die Effekte mit speziellen "Mapping"-Methoden simulieren:
  - **Environment Mapping:** Simuliert die Reflexion der Umgebung auf einer Oberfläche. (Beispiel: Eine glänzende Kugel spiegelt die Umgebung wider, ohne dass die Umgebung geometrisch modelliert werden muss.)
  - **Texture Mapping:** Bringt ein 2D-Bild (Textur) auf eine 3D-Oberfläche auf, um deren Farbe und Muster zu definieren. (Beispiel: Eine Ziegelmauertextur auf einer Wand.)
  - **Bump Mapping:** Simuliert Oberflächenunebenheiten, indem es die Normalenvektoren der Oberfläche modifiziert. Dies beeinflusst die Schattierung und erzeugt den Eindruck von Unebenheiten, ohne die Geometrie tatsächlich zu verändern. (Beispiel: Eine grobe Steintextur, die Risse und Vertiefungen simuliert.)
- **"Mapping" Bedeutung:** Im Kontext der Computergrafik bedeutet "Mapping" das Abbilden von Informationen (wie Farben, Helligkeiten, Normalen) von einem Raum (oft 2D-Bild) auf eine 3D-Oberfläche.

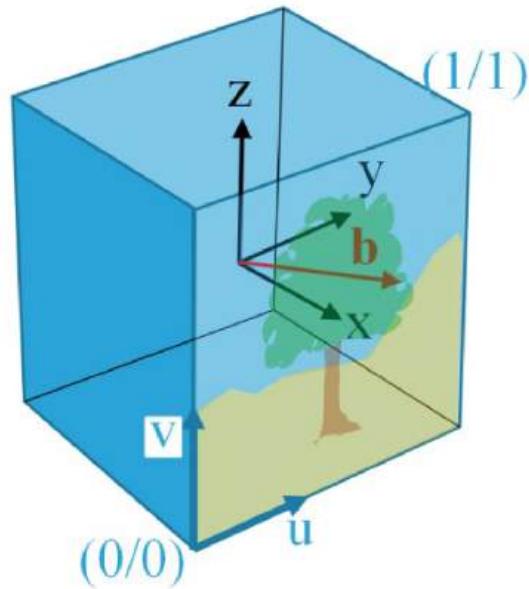
→ trace light rays from light source(s) and store illumination on objects



## Environment Mapping

[EVC\\_Skriptum\\_CG, p.46](#)

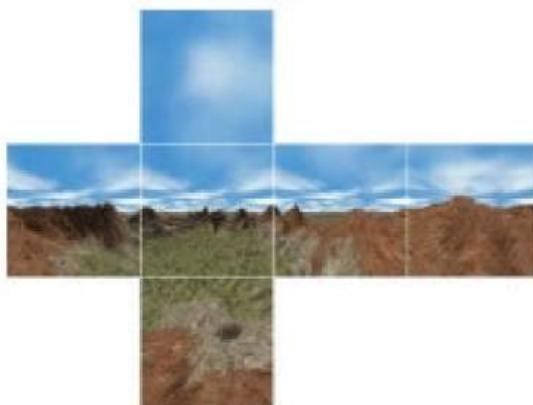
- **Definition:** Environment Mapping (Umgebungsabbildung) bezeichnet die Technik, wie die Umgebung oder Umwelt eines Objekts dessen Aussehen beeinflusst, insbesondere durch Reflexionen.
- **Effekt:** Je nach Oberflächeneigenschaften des Objekts wird die Umgebung auf verschiedene Weisen im Erscheinungsbild des Objekts widergespiegelt.
- **Anwendung bei verschiedenen Oberflächen:**
  - **Perfekt spiegelnde Oberflächen:** Für diese Oberflächen würde Ray-Tracing das exakte Spiegelbild der Umgebung erzeugen. Environment Mapping kann dies annähernd simulieren.
  - **Nicht perfekt diffuse Oberflächen:** Für diese Oberflächen, die einen gewissen Glanz aufweisen, kann man mit dem Phong-Modell (oder ähnlichen Reflexionsmodellen) einen Glanz-Effekt erzeugen, der die Spiegelung von Lichtquellen annähert.
  - **Mehr oder weniger spiegelnde Oberflächen:** Reflektieren ihre ganze Umgebung mehr oder weniger scharf. Die Genauigkeit der Spiegelung wird dabei reduziert, d.h. sie ist nicht exakt.
- **Motivation für Environment Mapping:**
  - Um ein effizientes Rendering von Objekten in einer komplexen Umgebung zu ermöglichen, ohne die gesamte Umgebung vollständig und exakt modellieren zu müssen.
  - **Vorgehen:** Die Umgebung wird in einem Vorverarbeitungsschritt von einem zentralen Punkt aus (z.B. dem Mittelpunkt des Objekts oder der darzustellenden Szene) als Bild (oder Satz von Bildern) produziert.



- **Art der Umgebungsinformation:**

- Es spielt keine Rolle, ob die Umgebungsbilder berechnet oder fotografiert wurden.
- **Wichtig:** Die Umgebung ist im Verhältnis zu den Objekten, die man darstellen will, sehr groß.
- **Annahme:** Bei der Darstellung eines Objekts wird für jeden Oberflächenpunkt näherungsweise angenommen, dass er sich im Zentrum dieser Umgebung befindet. Dies ist eine Vereinfachung, die für Objekte, die klein im Vergleich zur Umgebung sind, gute Ergebnisse liefert.

- **Vorteil:** Man kann allein aus der Richtung des Reflexionsstrahls bestimmen, welcher Umgebungschnitt getroffen wird (z.B. mit Polarkoordinaten), und erspart sich aufwändiges Ray-Tracing für die Umgebungsinformation.

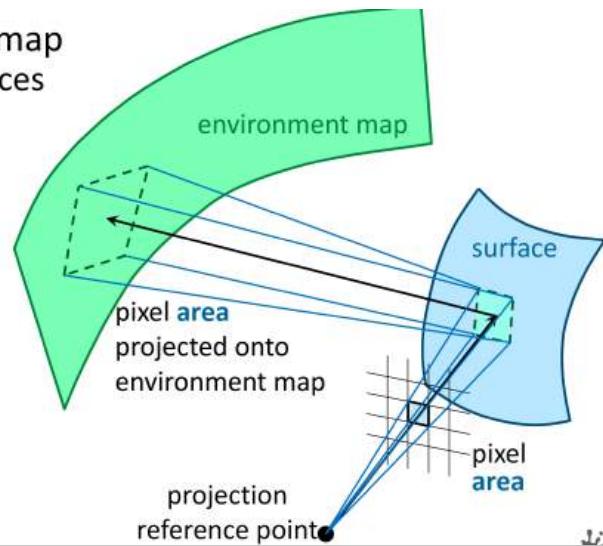


### information in the environment map

- intensity values for light sources
- sky
- background objects

### pixel area

- projected onto surface
- reflected onto environment map



## Cube Map als Environment Map

EVC\_Skriptum\_CG, p.46

- **Häufigste Methode:** Häufig wird ein achsenparalleler Würfel als Environment Map verwendet, der als **Cube Map** bezeichnet wird.
- **Struktur der Cube Map:** Jede Seite dieses Würfels ist ein Bild der Größe  $u \times v = [0, 1] \times [0, 1]$ .
- **Effiziente Bildwertabfrage:** Um den Bildwert für eine bestimmte Reflexionsrichtung  $R_S$  effizient zu bestimmen, kann man die  $x, y, z$  Koordinaten des Reflexionsvektors nutzen.
  - **Beispiel (für die Seite  $+x$ ):** Wenn der dominante Anteil des Reflexionsvektors die  $x$ -Komponente ist, wird die Textur der  $+x$ -Seite verwendet. Die  $u, v$ -Koordinaten werden dann aus den verbleibenden Komponenten  $y$  und  $z$  abgeleitet:
    - $u = (y_R + x_R)/(2x_R)$
    - $v = (z_R + x_R)/(2x_R)$
    - Hierbei ist  $R_S = (x_R, y_R, z_R)$  der Reflexionsvektor.
- **Reflexionsrichtung  $R_S$ :** Die Reflexionsrichtung erhält man nach dem gleichen Prinzip wie schon beim Ray-Tracing und bei den Schattierungsverfahren abgeleitet:
  - $R_S = (2n \cdot v)n - v$
  - $v$ : Vektor vom Betrachter zum Oberflächenpunkt.
  - $n$ : Oberflächennormale am Punkt.
  - **Intuitive Erklärung:** Dies ist die klassische Formel für die Reflexion eines Vektors an einer Oberfläche. Der Term  $(2n \cdot v)n$  ist der Anteil des Vektors  $v$ , der parallel zur Normalen  $n$  ist, gespiegelt an der Oberfläche. Davon wird der ursprüngliche Vektor  $v$  subtrahiert, um die reine Reflexionsrichtung zu erhalten.



(1/1) find  $(u, v)$  from the direction vector  $\mathbf{r}(x_r, y_r, z_r)$ :

if  $x_r > |y_r|$  and  $x_r > |z_r|$  then "front face"

$$u = (y_r + x_r)/2x_r$$

$$v = (z_r + x_r)/2x_r$$

top view

analogous formulas for the other 5 faces  
 $(-x > |y| \wedge -x > |z|, \quad y > |x| \wedge y > |z|, \quad -y > |x| \wedge -y > |z|, \quad z > |x| \wedge z > |y|, \quad -z > |x| \wedge -z > |y|)$

(1/1) calculation of the direction vector  $\mathbf{r}$ :

for a pixel:  
 viewing direction  $\mathbf{v}$   
 and normal vector  $\mathbf{n} \Rightarrow \mathbf{r} + \mathbf{v} = (2\mathbf{n} \cdot \mathbf{v})\mathbf{n}$

$$\mathbf{r} = (2\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v}$$

## Texture Mapping

EVC Skriptum CG, p.47

- Problem:** Viele Oberflächen in der realen Welt sind nicht einfarbig, sondern weisen komplexe Muster, Farben und Details auf (z.B. Holzmaserung, Bilder, Schrift, Verschmutzung, Kleidung, Marmor).
- Lösung:** Um solche Muster auf Oberflächen darzustellen, auch wenn die grundlegende geometrische Modellierung grob ist, wird die Technik des **Texture Mapping** verwendet.
- Was ist eine Textur?** Ein Muster, das auf eine Oberfläche aufgebracht wird. Beispiele: Fenster auf einer Hauswand, Wolken am Himmel, Gesichter, Knöpfe, Reißverschlüsse, Pflastersteine.

- **Zweischrittiger Prozess des Texture Mapping:**

1. **Textur-Objekt Transformation (Modellierungsschritt):**

- **Definition:** Zuerst muss definiert werden, welche Textur (oft ein 2D-Bild) auf welche Oberfläche des 3D-Objekts aufgebracht werden soll.
- **Parameter:** Dabei werden Orientierung, Skalierung, Wiederholung (Tiling) und andere Parameter festgelegt.
- **Raum:** Hier findet eine Transformation von **Textur-Raum (u,v) Array Koordinaten** (die 2D-Koordinaten innerhalb der Textur) in **Objekt-Raum (u',v')** **Flächen-Parameter** (Parameter, die einen Punkt auf der 3D-Oberfläche des Objekts eindeutig identifizieren) statt.
- **Bedeutung:** Dieser Schritt ist eigentlich Teil der Modellierung, bei der die Oberflächen ein visuelles Aussehen erhalten.

2. **Viewing- & Projektions-Transformation (Rendering-Schritt):**

*Ziel: Die Textur muss korrekt auf das Abbild des Objekts im finalen Bild gerendert werden.*

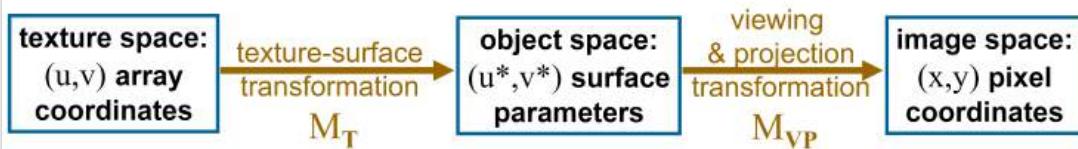
**Raum:** Dies beinhaltet die Transformation von den **Objekt-Raum (u',v')** **Flächen-Parametern** in **Bild-Raum (x,y) Pixel Koordinaten**.

\* **Bedeutung:** Hier wird die Textur tatsächlich auf das gerenderte Objekt projiziert und dargestellt.

### texture mapping

forward: texture scanning  $(u,v) \rightarrow (x,y)$

backward: inverse scanning  $(x,y) \rightarrow (u,v)$

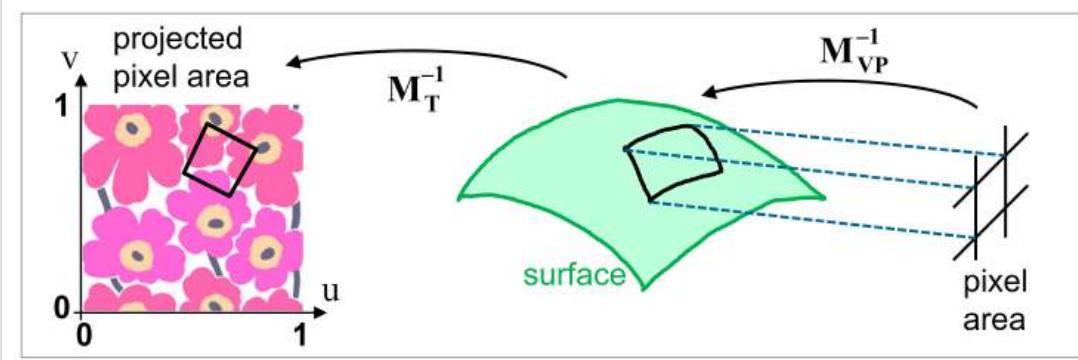


### texture-surface transformation

$$\begin{aligned} u^* &= u^*(u,v) = a_{u^*}u + b_{u^*}v + c_{u^*} \\ v^* &= v^*(u,v) = a_{v^*}u + b_{v^*}v + c_{v^*} \end{aligned}$$

projecting pixel areas to texture space =

= inverse scanning  $(x,y) \rightarrow (u,v)$



## Erzeugung einer Textur

[EVC\\_Skriptum\\_CG, p.47](#)

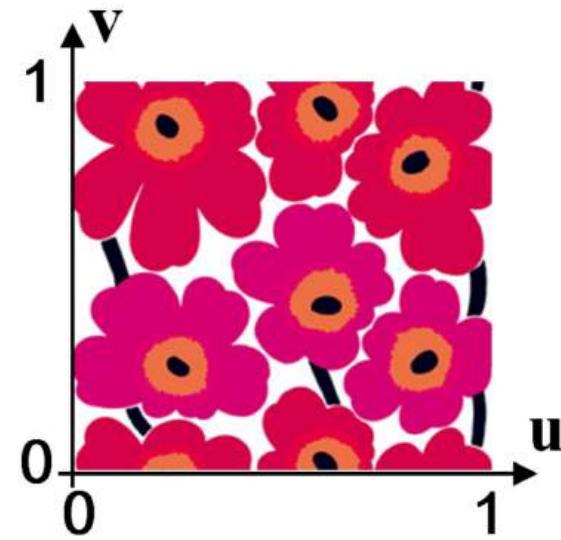
- **Grundsatz:** Die Herkunft einer Textur ist zweitrangig; wichtig ist nur, dass sie an allen benötigten Stellen definiert und abrufbar ist.
- **Standardverfahren:** Meist wird eine Textur in einem Vorverarbeitungsschritt als **Pixel-Array** (also ein Raster von Farbwerten) erstellt und später im Rendering-Prozess darauf zugegriffen.
- **Mögliche Quellen für Texturen:**
  - **Fotografien:** Man kann Fotos von realen Oberflächen verwenden.
  - **Scans:** Gescanntes Material.
  - **Programmgenerierte Texturen:** Texturen, die durch ein Programm erzeugt werden, bis hin zu Zufallswerten (Noise).
  - **Datenbanken:** Für häufig verwendete Texturen (z.B. Holzmaserung, Gras, Sand, Marmor, Kopfsteinpflaster, Stoffstrukturen) können Datenbanken angelegt werden.
- **Prozedurale Texturierung:**
  - **Definition:** Wenn Texturen aus einer **mathematischen Funktion** gewonnen werden, spricht man von "Procedural Texturing".
  - **Vorteile von Prozeduralen Texturen:**
    - **Kein Speicherplatz für Bilder:** Die Textur wird zur Laufzeit berechnet, nicht gespeichert.
    - **Unendliche Detailtiefe:** Können beliebig hoch aufgelöst werden, ohne an Qualität zu verlieren oder Pixelartefakte zu zeigen (im Gegensatz zu Bitmap-Texturen).
    - **Parameterisierbarkeit:** Oft lassen sich Parameter der Funktion ändern, um Variationen der Textur zu erzeugen.



## Textur Objekt Transformation

[EVC\\_Skriptum\\_CG, p.47](#)

- **Ausgangssituation:**
  - Die Textur liegt normalerweise in einem **2D-Koordinatensystem** vor, dessen Achsen mit  $(u, v)$  bezeichnet werden. Diese  $(u, v)$ -Koordinaten sind die Texturkoordinaten.
  - Die Oberfläche des 3D-Objekts, auf die die Textur aufgebracht werden soll, hat ebenfalls eine **parametrische Darstellung**, die mit  $(u^*, v^*)$  bezeichnet wird. Diese  $(u^*, v^*)$ -Koordinaten sind die Oberflächenparameter des Objekts.

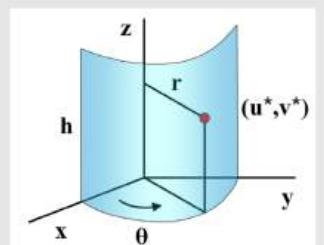


- **Ziel der Transformation:** Für jeden Punkt auf der 3D-Objektoberfläche (identifiziert durch seine  $(u^*, v^*)$ -Parameter) soll die zugehörige Farbe aus der Textur ermittelt werden. Das bedeutet, wir brauchen eine Abbildung von den Objekt-Oberflächenparametern zurück zu den Texturkoordinaten, oder umgekehrt, eine Abbildung von den Texturkoordinaten zu den Objekt-Oberflächenparametern.
- **Biliniere Funktion für die Textur-Objekt-Transformation:**
  - Eine häufig verwendete Methode, um eine Textur auf eine Oberfläche aufzubringen, ist eine bilinare Funktion. Diese Funktion bildet die Texturkoordinaten  $(u, v)$  auf die Oberflächenparameter  $(u^*, v^*)$  ab:
    - $u^* = u^*(u, v) = a_u u + b_u v + c_u$
    - $v^* = v^*(u, v) = a_v u + b_v v + c_v$
  - **Erklärung:** Diese linearen Gleichungen (bilinear, wenn man die unabhängigen Variablen  $u, v$  und die Konstanten  $a_u, b_u, c_u, a_v, b_v, c_v$  betrachtet) definieren, wie ein Punkt in der 2D-Textur  $(u, v)$  auf einen Punkt auf der 3D-Oberfläche  $(u^*, v^*)$  abgebildet wird. Die Koeffizienten  $(a, b, c)$  steuern dabei Skalierung, Rotation, Scherung und Translation der Textur auf der Oberfläche.
  - **Praktische Bedeutung:** Man kann für jeden Punkt der Objektoberfläche die zugehörige Farbe aus der Textur bestimmen.
- **Bezeichnung:** Diese Funktion wird als **Textur-Objekt-Transformation** bezeichnet und mit  $M_T$  denotiert.

**Beispiel:**

Eine Textur  $t(u, v), 0 \leq u, v \leq 1$ , soll auf einen Viertel-Zylinder mit Höhe  $h$  aufgetragen werden, dessen Mantel in  $z$ -Richtung mit  $v^*$  parametrisiert ist und entlang der Krümmung mit  $u^* (= \theta)$ . Um nun für ein Texturpixel  $t(u, v)$  [auch Texel genannt] zu berechnen, an welche Stelle des Zylinders es zu liegen kommt, muss man die Abbildung  $M_T$  definieren, das könnte etwa sein:

$$u^* = u \cdot \pi/2, v^* = v \cdot h \quad (\text{so passt die Textur genau auf das Zylinderviertel}).$$



## Viewing und Projektionstransformation

- Die Abbildung eines 3D-Modells auf eine Bildebene ist grundsätzlich eine einfache *Projektion*  $M_{VP}$ .
- Beim *Rasterscannen* von Flächen (dem Prozess des Füllens von Pixeln), um sicherzustellen, dass jedes Pixel *genau einmal* gefärbt wird (also keine Übermalungen oder Löcher), arbeitet man in umgekehrter Richtung:
  - Man bestimmt für jedes Pixel  $(x, y)$  auf der Bildebene, welcher Oberflächenpunkt dort gezeichnet wird.
  - Dazu werden die  $(u_*, v_*)$ -Koordinaten (Texturkoordinaten) der Fläche und daraus das *Texel* (Textur-Pixel) für dieses Pixel bestimmt.
  - Dafür werden die inversen Operatoren  $M_{VP}^{-1}$  und  $M_T^{-1}$  benötigt.

**Beispiel (Fortsetzung):**

Für eine beliebige **Projektion** reicht es, wenn wir von jedem Punkt die  $(x, y, z)$ -Koordinaten kennen. Im Falle des obigen Zylinders ergibt sich:  $x = r \cdot \cos u_*$ ,  $y = r \cdot \sin u_*$ ,  $z = v_*$ .

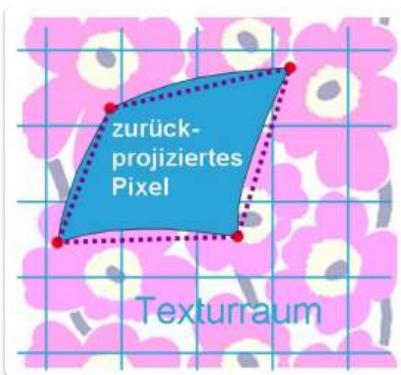
**Nun wird das Problem von hinten angegangen:**

- Für einen Bildpunkt  $P$  bestimmt man zuerst die Position  $(x, y, z)$  am Zylinder, die dort dargestellt wird (z.B. durch Ray-Casting).
- Für diesen Punkt muss man die Parameter der Oberfläche finden:  $u_* = \cos^{-1}(x/r)$ ,  $v_* = z$ . Bis hier ist das also die inverse Transformation  $M_{VP}^{-1}$ .
- Jetzt muss für das Parameterpaar  $(u_*, v_*)$  noch die Textur gefunden werden, indem  $M_T$  invertiert wird:  $u = 2u_* / \pi$ ,  $v = v_* / h$  (das ist  $M_T^{-1}$ )

## Anti-Aliasing für Texturen

[EVC\\_Skriptum\\_CG, p.48](#)

- **Problem:** Texturen sind besonders anfällig für *Aliasing-Effekte*. Dies tritt verstärkt auf, wenn Texturmuster vergrößert (Verpixelung) oder verkleinert (Moiré-Effekte, Flackern) werden.
  - **Aliasing:** Unerwünschte Artefakte (wie Treppeneffekte, Flackern, Moiré-Muster), die entstehen, wenn ein kontinuierliches Signal (die Textur) mit einer zu geringen Abtastrate (den Pixeln) diskretisiert wird.
- **Korrekte, aber langsame Lösung:**
  - Man müsste den **Textur-Durchschnittswert** der Fläche berechnen, die von einer Rückprojektion des zu füllenden Pixels auf die Textur erzeugt wird.
  - **Rückprojektion:** Ein Pixel auf dem Bildschirm entspricht nicht einem einzelnen Punkt, sondern einer Fläche in der Textur. Man muss diese Fläche in der Textur bestimmen.
  - **Näherung:** Oft reicht es näherungsweise aus, das Viereck zu verwenden, das durch die gerade Verbindung der rückprojizierten Eckpunkte des Pixels entsteht.
  - **Nachteil:** Diese Methode ist sehr langsam.



- Optimierungen für Anti-Aliasing bei Texturen:

1. Mip-Mapping:

- **Prinzip:** Die Textur wird in **verschiedenen Größen (Auflösungen)** vorab berechnet und gespeichert (ein sogenannter "Mip-Map-Pyramide").
- **Anwendung:** Je nachdem, wie stark ein Objekt im Bild verkleinert erscheint, wird die passende Texturgröße aus der Mip-Map-Pyramide ausgewählt.
- **Qualitätsverbesserung:** Zwischen den verschiedenen Größenstufen kann dann linear interpoliert werden, um fließende Übergänge und eine bessere Qualität zu erzielen.
- **Vorteil:** Reduziert Aliasing bei Verkleinerung der Textur und verbessert die Cache-Kohärenz.

2. Summed-Area-Table-Methode (SAT):

- **Prinzip:** Man erstellt eine sogenannte **Summen-Textur (Summed-Area-Table)**. In jedem Punkt dieser Tabelle ist die Summe aller Textelwerte (Textur-Pixel) von einem Referenzpunkt bis zu diesem Punkt gespeichert.
- **Anwendung:** Durch einfache Differenzenbildung in dieser Summen-Textur kann man leicht den Durchschnittswert **beliebiger rechteckiger Bereiche** in der Originaltextur ermitteln.
- **Vorteil:** Ermöglicht die schnelle Berechnung von Durchschnittswerten für rechteckige Texturbereiche, was für die Filterung von Texturen beim Anti-Aliasing nützlich ist.
- **Einschränkung:** Bei nicht-rechteckigen rückprojizierten Pixeln kann die Summed-Area-Table nur eine Approximation liefern.

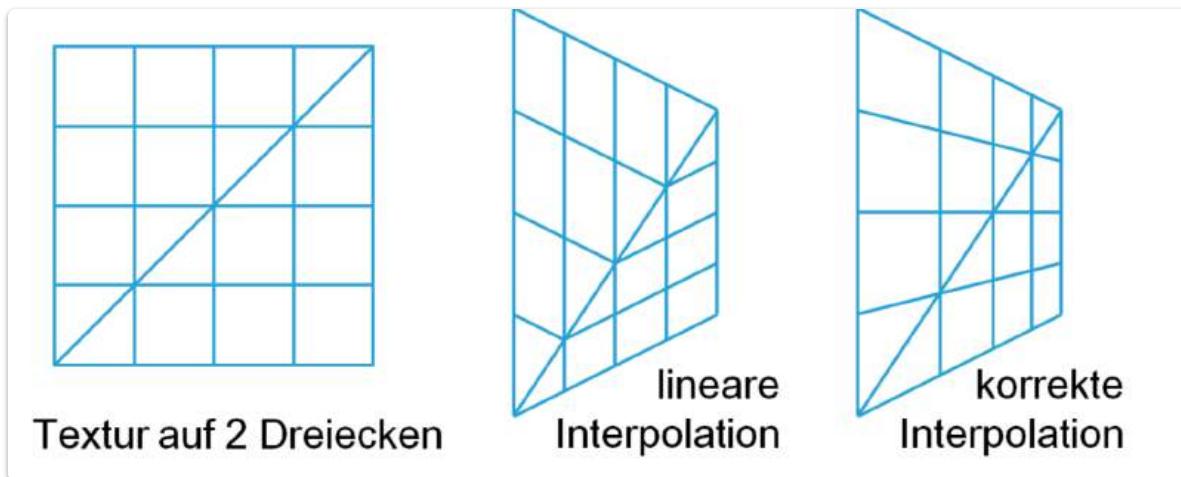
## Texturen auf perspektivisch verzerrten Dreiecken

[EVC\\_Skriptum\\_CG, p.48](#)

Die Verwendung von **baryzentrischen Koordinaten** ermöglicht eine **lineare Interpolation der Eckpunktwerte** über die Dreiecksfläche.

Bei der **perspektivischen Transformation** geht diese Linearität der Oberflächenparameter jedoch verloren. Daher muss die Interpolation vor der Homogenisierung erfolgen, da sie dort nicht linear ist.

Anstatt die Farbe für einen Punkt  $p(\alpha, \beta, \gamma)$  mit  $\text{color}(x, y) = \alpha \cdot t_0 + \beta \cdot t_1 + \gamma \cdot t_2$  zu berechnen, werden die **Texturparameter  $u$  und  $v$**  mithilfe von **baryzentrischen Koordinaten ( $\alpha_w, \beta_w, \gamma_w$ )** vor der Perspektive ermittelt. Diese werden dann zur Berechnung des Texturwertes am Punkt  $p(\alpha, \beta, \gamma) = p(x, y)$  verwendet.



- **Lineare Interpolation (nicht korrekt für perspektivisch verzerrte Dreiecke):** Wenn man die Texturparameter  $u$  und  $v$  direkt nach der perspektivischen Transformation linear interpoliert, entstehen Verzerrungen, wie im mittleren Bild gezeigt. Die Textur "verläuft" nicht gleichmäßig über die Fläche.
- **Korrekte Interpolation:** Um dies zu vermeiden, werden die Texturparameter  $u$  und  $v$  (oder genauer, die baryzentrischen Koordinaten  $\alpha_w, \beta_w, \gamma_w$ , die zu  $u, v$  führen) vor der perspektivischen Transformation berechnet. Dadurch bleibt die Linearität erhalten und die Textur wird korrekt auf das perspektivisch verzerrte Dreieck abgebildet.

### Formeln für die korrekte Texturinterpolation:

Gegeben sind die baryzentrischen Koordinaten  $\alpha, \beta, \gamma$  des Punkts  $p(x, y)$  im perspektivisch transformierten Raum. Um die korrekten baryzentrischen Koordinaten  $\alpha_w, \beta_w, \gamma_w$  im "World-Space" (vor der Projektion) zu finden, benötigen wir die homogenen Koordinaten und die  $w$ -Komponente  $(h_0, h_1, h_2)$ .

Die Formeln lauten:

$$d = h_1 h_2 + h_2 \beta(h_0 - h_1) + h_1 \gamma(h_0 - h_2)$$

$$\beta_w = h_0 h_2 \beta / d$$

$$\gamma_w = h_0 h_1 \gamma / d$$

$$\alpha_w = 1 - \beta_w - \gamma_w$$

- **Erklärung der Variablen:**

- $h_0, h_1, h_2$ : Dies sind wahrscheinlich die  $w$ -Komponenten (homogene Koordinaten) der Eckpunkte des Dreiecks vor der Division durch  $w$  (d.h., nach der Multiplikation mit der Projektionsmatrix, aber bevor die einzelnen Koordinaten durch  $w$  geteilt

werden). In der Praxis werden diese  $w$ -Werte oft als  $w_0, w_1, w_2$  der Eckpunkte  $v_0, v_1, v_2$  bezeichnet.

- $d$ : Ein Hilfswert, der für die Berechnung von  $\beta_w$  und  $\gamma_w$  benötigt wird.
- $\alpha_w, \beta_w, \gamma_w$ : Die korrigierten baryzentrischen Koordinaten im "World-Space".

Sobald wir die korrigierten baryzentrischen Koordinaten  $\alpha_w, \beta_w, \gamma_w$  haben, können wir die Texturparameter  $u$  und  $v$  interpolieren:

$$u = \alpha_w u_0 + \beta_w u_1 + \gamma_w u_2$$

$$v = \alpha_w v_0 + \beta_w v_1 + \gamma_w v_2$$

- **Erklärung der Variablen:**

- $u_0, v_0, u_1, v_1, u_2, v_2$ : Die Texturkoordinaten der drei Eckpunkte des Dreiecks.

Zuletzt wird die Farbe des Punktes mit den berechneten Texturparametern  $u$  und  $v$  bestimmt, indem der Texturwert aus einer Texturfunktion  $t(u, v)$  abgefragt wird:

$$\text{color}(x, y) = t(u, v)$$

## Solid Texturing

---

### EVC\_Skriptum\_CG, p.49

Solid Texturing ist eine Methode, bei der eine Textur nicht als 2D-Bild, sondern als **3D-Volumen** definiert ist.

- **Wie es funktioniert:**
  - In einem **3-dimensionalen Parameterraum** wird für jeden Raumpunkt eine Farbe oder ein Texturwert definiert.
  - Wenn sich eine Oberfläche an einem bestimmten Raumpunkt befindet, wird die dort definierte Farbe/der Texturwert abgerufen und auf die Oberfläche angewendet.
- **Darstellung der Textur:**
  - Die 3D-Textur kann entweder als **mathematische Funktion** gegeben sein (z.B. eine Perlin Noise Funktion für Wolken oder Marmor).
  - Oder sie kann durch **Volumendaten** repräsentiert werden (ähnlich wie ein 3D-Gitter, bei dem jeder Voxel einen Farbwert speichert).
- **Wichtige Voraussetzung:** Man muss in der Lage sein, die Texturwerte für **jeden Raumpunkt** abzufragen.



## Vorteile von Solid Texturing:

1. **Kohärentes Muster über Kanten hinweg:** Der größte Vorteil ist, dass das Texturmuster **nahtlos über alle Kanten** und zwischen verschiedenen Polygonen eines Objekts verläuft.
  - **Keine Zusammenfüngungsprobleme:** Bei traditionellem 2D-UV-Mapping kann es zu sichtbaren Nähten oder Verzerrungen an den Grenzen von UV-Segmenten kommen. Solid Texturing eliminiert dieses Problem, da die Textur "durch" das Objekt geht, als wäre es aus dem Texturvolumen geschnitten.
2. **Einfachere Abbildung der Textur auf das Objekt:** Die Zuordnung der Textur zu den Objektkoordinaten ist wesentlich einfacher zu handhaben, da sie direkt auf den 3D-Positionen des Objekts basiert und nicht erst komplexe 2D-UV-Koordinaten berechnet werden müssen.

## Bump Mapping

[EVC\\_Skriptum\\_CG, p.49](#)

Viele Oberflächen in der realen Welt besitzen eine **geometrische Detailstruktur** (z.B. die Rinde eines Baumes, eine Münzoberfläche, Stoffgewebe, Putz an einer Wand). Solche feinen Details auf der Oberfläche als tatsächliche Geometrie (d.h. durch zusätzliche Polygone) zu modellieren, ist extrem aufwendig und erzeugt riesige Datenmengen.

**Bump Mapping** ist eine Technik, um diesen Aufwand signifikant zu reduzieren.

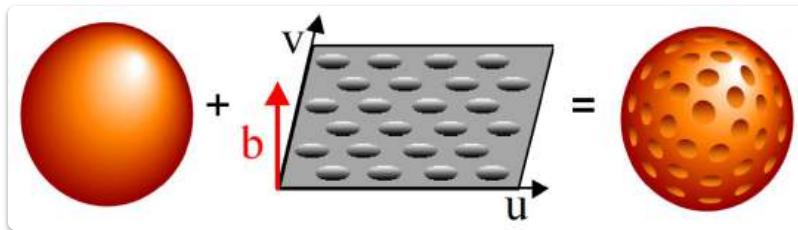


Betrachtet man die graue Leiste, scheint sie sechs Ausbuchtungen und eine Einbuchtung zu haben. Fühlt man die Oberfläche jedoch an, stellt man fest, dass sie vollkommen eben ist.

- **Warum sehen wir die Unebenheiten?**

- Der Eindruck von Unebenheiten entsteht allein durch die **Schattierung**. Unser Gehirn interpretiert die Helligkeits- und Schattenmuster als dreidimensionalen Raum.

## Grundidee des Bump Mappings:



Die Grundidee des Bump Mappings besteht darin, den **Eindruck von Oberflächenunebenheiten (Bumps)** zu erwecken, ohne die tatsächliche Geometrie der Oberfläche zu verändern.

- Wie es funktioniert:**
  - Die geometrische Oberfläche bleibt unverändert (z.B. eine glatte Kugel oder Ebene).
  - Stattdessen wird der **Normalvektor** der Oberfläche (der für die Lichtberechnung verwendet wird) entsprechend der gewünschten Unebenheit **modifiziert**.
  - Diese Modifikation des Normalvektors beeinflusst, wie das Licht von der Oberfläche reflektiert wird, was wiederum die Schattierung verändert.
  - Dadurch entspricht die **Schattierung den "Bumps"**, obwohl geometrisch nichts an der Oberfläche verändert wurde.
- Vorteil:** Man kann mit sehr geringem Aufwand (nur durch die Anpassung der Normalen in der Fragment-Shader-Phase) den visuellen Eindruck von komplexen Details erzeugen, ohne die Anzahl der Polygone erhöhen zu müssen.

## Bump-Mapping-Algorithmus

EVC\_Skriptum\_CG, p.49

Sei die Bump-Textur in Form eines Arrays von Höhenwerten  $b(u, v)$  gegeben, das heißt also, dass die Position der Stelle  $p(u, v)$  der Oberfläche, die durch das Parameterpaar  $(u, v)$  erzeugt wird, um  $b(u, v)$  in Richtung des Normalvektors  $n$  an dieser Stelle verschoben erscheinen soll.  $n$  erhält man indem man das Kreuzprodukt zweier Tangentenvektoren auf die Länge 1 normiert:

$$n^* = p_u \times p_v, \quad n = n^*/|n^*|$$

Der verschobene Punkt  $p'(u, v)$  ergibt sich dann zu:  $p'(u, v) = p(u, v) + b(u, v) \cdot n$ . Wir aber brauchen  $n'$ , also die Normale auf den verschobenen Punkt:

$$n' = p'_u \times p'_v$$

Nun gilt:

$$p'_u = \partial(p + b_n)/\partial u = p_u + b_u n + b n_u$$

und weil  $b$  sehr klein ist:  $p'_u \approx p_u + b_u n$  analog gilt natürlich  $p'_v \approx p_v + b v n$ , sodass sich  $n'$  ergibt:

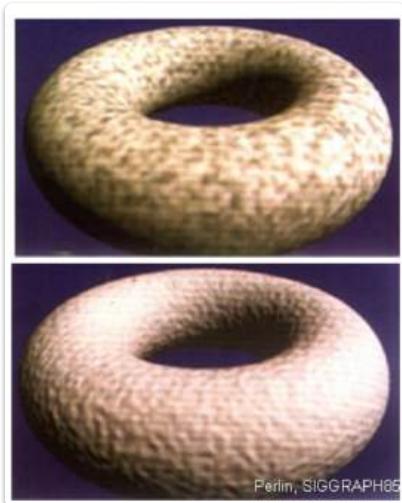
$$n' = p'_u \times p'_v = p_u \times p_v + b_v(p_u \times n) + b_u(n \times p_v) + b_u b_v(n \times n)$$

und aus  $n \times n = 0$  folgt schließlich:  $n' = n + b_v(p_u \times n) + b_u(n \times p_v)$ .

Für die Berechnung des modifizierten Normalvektors benötigt man nicht den Höhenwert  $b(u, v)$  direkt, sondern die **Ableitungen** von  $b(u, v)$  nach  $u$  und  $v$  (also  $\frac{\partial b}{\partial u}$  und  $\frac{\partial b}{\partial v}$ ).

- **Praktische Anwendung:**

- In der Praxis sind die Parametrisierung der Oberfläche (UV-Koordinaten) und die der Bump-Textur häufig identisch.
- Das ermöglicht es, diese Ableitungen **vorzuberechnen** und direkt in der Bump-Map zu speichern, anstatt die eigentlichen Höhenwerte  $b(u, v)$  zu speichern. Eine solche Map, die Ableitungen oder direkt die Normalen speichert, nennt man oft **Normal Map**.



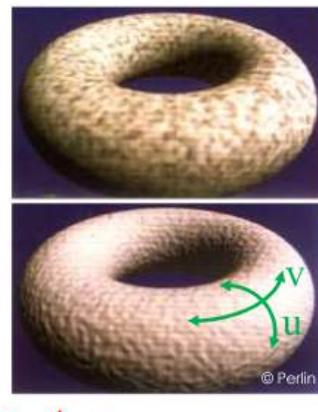
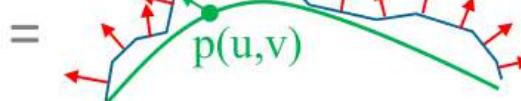
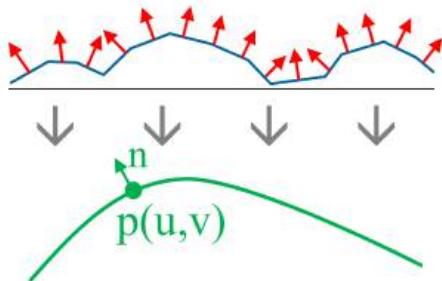
Beachte den Unterschied in der Donut-Abbildung:

- **Texture-Map (oben):** Zeigt die reinen Farb- oder Helligkeitsinformationen der Textur.
- **Bump-Map (unten):** Ist eine Darstellung, die die lokalen Unebenheiten kodiert.

surface roughness is simulated

perturbation function varies surface normal *locally*

bump map  $b(u,v)$  derivative  $b'(u,v)$



Der räumliche Eindruck der Oberfläche entsteht erst, wenn die **Schattierung eine Richtungsabhängigkeit bekommt**, d.h., wenn das Licht aus verschiedenen Richtungen unterschiedlich reflektiert wird, basierend auf den modifizierten Normalen.

## Einschränkungen und Nachteile von Bump Mapping

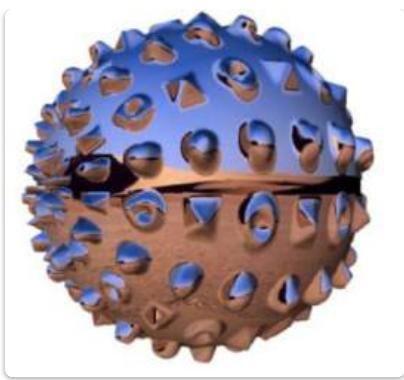
[EVC\\_Skriptum\\_CG](#), p.50

Bump Mapping ist ein **visueller Trick**, der die Schattierung verändert, aber die tatsächliche Geometrie nicht korrigiert. Dies führt zu **sichtbaren Fehlern**, die bei ausgeprägten "Bumps" deutlicher werden:

1. **Verzerrung bei flachen Winkeln:** Bei flachen Blickwinkel erscheint die simulierte Struktur stark verzerrt und unrealistisch.
2. **Glatte Silhouette:** Der **Umriss des Objekts bleibt glatt**, da die Geometrie unverändert ist, was bei Objekten mit echten Unebenheiten falsch aussieht.
3. **Glatte Schattenränder:** Bedingt durch die glatte Silhouette wirft das Objekt **Schatten mit glatten Rändern**, statt unregelmäßigen.
4. **Keine gegenseitigen Schatten der Bumps:** Die simulierten Unebenheiten **werfen keine Schatten aufeinander** oder auf die Oberfläche selbst, da sie keine echte Geometrie besitzen.
5. **Falsche Beleuchtung auf lichtabgewandter Seite:** Modifizierte Normalen können dazu führen, dass Oberflächenbereiche, die geometrisch im Schatten liegen sollten, **fälschlicherweise beleuchtet werden**.

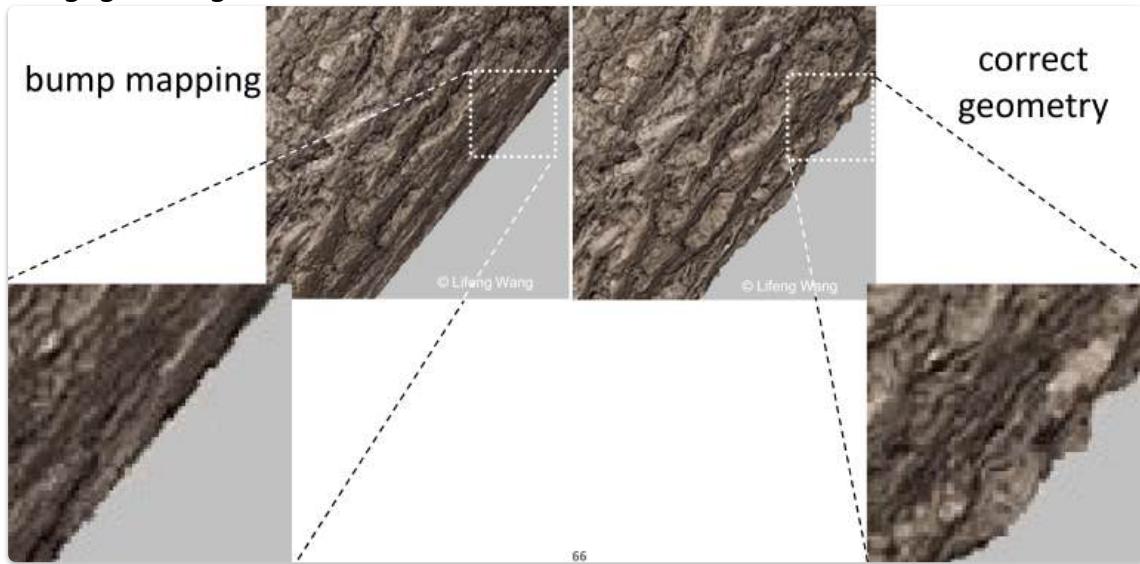
## Displacement Mapping

Während Bump Mapping eine Vielzahl von Fehlern aufweist (wie in den vorherigen Notizen beschrieben), gibt es für jeden dieser Fehler mehr oder weniger aufwendige Hilfslösungen. Die **korrekteste Methode** zur Darstellung von Oberflächenunebenheiten ist jedoch, die **Oberfläche tatsächlich um die "Bump-Höhe" zu verändern**. Diese Methode wird als **Displacement Mapping** bezeichnet.



## Funktionsweise und Vorteile:

- **Tatsächliche Geometrieverziehung:** Bei Displacement Mapping werden die Oberflächenpunkte tatsächlich verschoben. Das bedeutet, dass die geometrischen Koordinaten der Scheitelpunkte (Vertices) des Modells verändert werden, um die Unebenheiten zu repräsentieren.
- **Korrekte Silhouette:** Da die Geometrie physisch verändert wird, entsteht natürlich auch eine korrekte Silhouette des Objekts. Die Ränder des Objekts zeigen nun die tatsächlichen Unebenheiten und nicht mehr die glatte Form der ursprünglichen Geometrie.
- **Korrekte Schatten:** Die versetzte Geometrie wirft auch korrekte Schatten, einschließlich der gegenseitigen Schatten der Unebenheiten aufeinander.



## Implementierung und Hardware-Unterstützung:

EVC\_Skriptum\_CG, p.50

- **Aufwand:** Displacement Mapping ist viel aufwendiger zu implementieren und rechenintensiver als Bump Mapping, da es eine tatsächliche Veränderung der Geometrie erfordert (oft durch Hinzufügen vieler neuer Polygone).
- **Hardware-Unterstützung:** Moderne Grafikkarten unterstützen Displacement Mapping jedoch. Dies geschieht typischerweise mittels einer zusätzlichen programmierbaren Hardware-Stufe in der Rendering-Pipeline, die oft als "Tessellation-Stage" bezeichnet wird.

- **Tessellation-Stage:** Diese Stufe befindet sich **zwischen dem Vertex- und dem Pixel-Stage**. Ihre Aufgabe ist es, die Dreiecke direkt auf der Hardware in **viele kleine Dreiecke zu unterteilen** (Tessellation). Jedes dieser neu erzeugten kleinen Dreiecke kann dann entsprechend einer **Displacement Map** verschoben werden, um die feinen Details zu erzeugen.

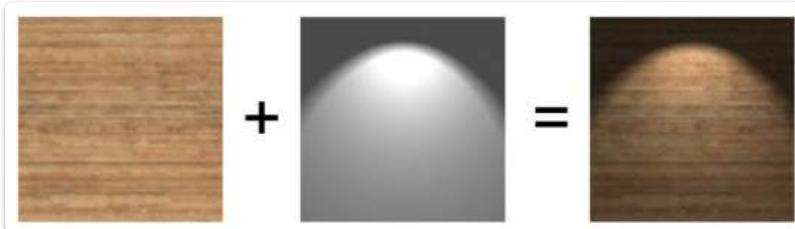
**Zusammenfassend:** Displacement Mapping bietet eine physikalisch korrektere Darstellung von Oberflächenstrukturen als Bump Mapping, indem es die Geometrie tatsächlich modifiziert. Dies ist zwar komplexer, wird aber durch moderne GPU-Hardware effizient unterstützt.

## Kombination mehrerer Mappings

---

EVC\_Skriptum\_CG, p.50

- **Multitexturing** ist eine mächtige Methode, um *mehrere Mappings zu kombinieren*.
- Beispiele für kombinierbare Texturen sind:
  - Grundmuster (z.B. Holzmaserung)
  - Beleuchtung (z.B. Lichtkegel)
  - Verschmutzung
  - Unebenheiten
  - Fotos plus Annotationen (zusätzliche Informationen oder Markierungen auf einem Bild)



# 11. Deep Learning for Computer Vision

Mehr dazu siehe auch [14. Machine Learning für 3D Graphics](#)

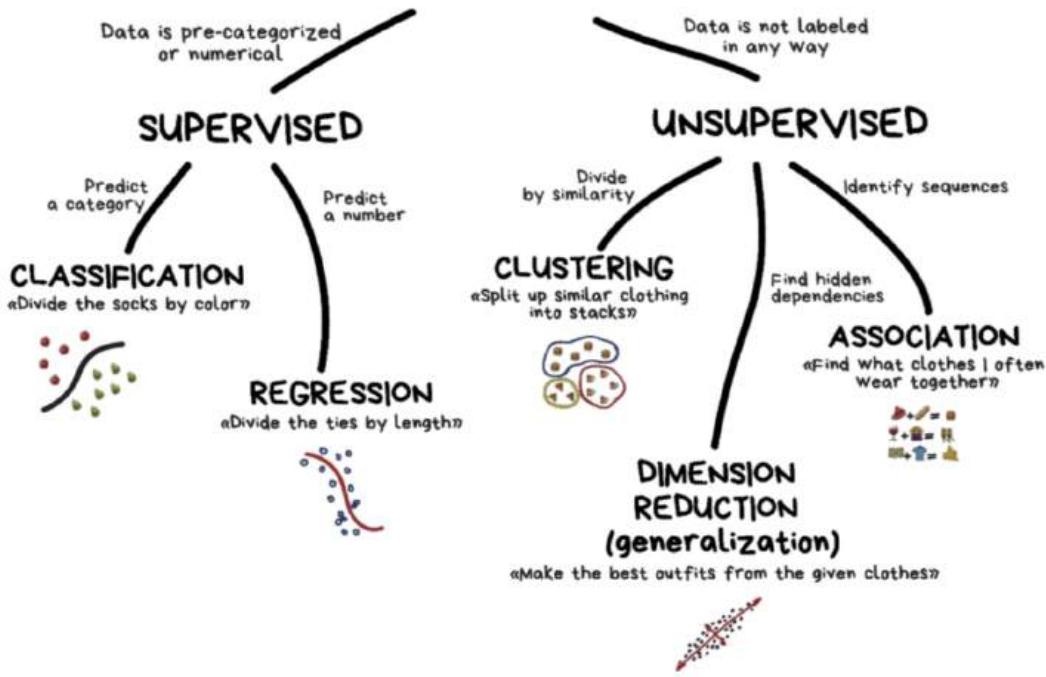
## Maschinelles Lernen

---

[EVC\\_Skriptum\\_CV, p.55](#)

- **Definition:** Maschinelles Lernen (ML) ist ein Teilgebiet der künstlichen Intelligenz (KI).
- **Geschichte:** Etabliertes Forschungsgebiet, geprägt durch rasante technologische Fortschritte seit Jahrzehnten.
- **Begriff der KI:** Erstmals 1955 von Minsky, McCarthy, Newell und Simon definiert: Maschinen, die sich verhalten, als würden sie über eine Art menschliche Intelligenz verfügen.
- **Allgemeines Prinzip des ML:** ML umfasst Lernprozesse, um Zusammenhänge in bestehenden Datensätzen zu erkennen und darauf basierend Vorhersagen zu treffen.
- **Wesentliche Bedeutung:** Modelle ziehen aufgrund von selbstlernenden Algorithmen und existierenden Daten zukunftsrelevante Rückschlüsse, ohne explizit programmiert zu werden.
- **Grundlage:** Lernprozesse dienen als Dateneingabe, die durch eine vordefinierte Reihe an Attributen charakterisiert ist.
- **Kategorien des Maschinellen Lernens:**
  - Überwachtes Lernen (Supervised Learning)
  - Unüberwachtes Lernen (Unsupervised Learning)
  - Bestärkendes Lernen (Reinforcement Learning)

# CLASSICAL MACHINE LEARNING



- **Überwachtes Lernen (Supervised Learning):**

- Daten sind vor-kategorisiert oder numerisch (Data is pre-categorized or numerical).
- Ziel: Vorhersage einer Kategorie (Klassifikation) oder einer Zahl (Regression).
- **Klassifikation:** Teilt die Daten nach Farben (Divide the socks by colors).
  - Beispiel: Vorhersage einer Kategorie.
- **Regression:** Passt eine Linie durch Datenpunkte (Predict a number).
  - Beispiel: Vorhersage eines Preises.

- **Unüberwachtes Lernen (Unsupervised Learning):**

- Daten sind in keiner Weise gelabelt (Data is not labeled in any way).
- Ziel: Struktur in den Daten finden.
- **Clustering:** Gruppert ähnliche Datenpunkte (Divide by similarity; spot similar clothing styles together).
  - Ziel: Gruppen ähnlicher Daten finden.
- **Assoziationsanalyse:** Findet häufige Abhängigkeiten (Find hidden dependencies; find what clothes I often wear together).
  - Ziel: Beziehungen zwischen Datenpunkten identifizieren.
- **Dimensionsreduktion (Generalisierung):** Reduziert die Anzahl der Merkmale (Create the best outfits from the given clothes).
  - Ziel: Wichtige Informationen extrahieren und Rauschen reduzieren.

## Überwachtes Lernen (Supervised Learning)

- **Zielvariable (dedizierte Ausgabewerte):** Wenn eine dedizierte AusgabevARIABLE definiert ist, kann durch Supervised Learning ein Modell darauf trainiert werden, diese für bisher

unbekannte Datensätze (Test Set) vorherzusagen.

- **Generelles Ziel:** Maximierung der Genauigkeit dieser Vorhersagen.
- **Anwendung:** Bei Datensätzen mit präzisen Ausgabewerten.
- **Lernprozess:** Algorithmus lernt die Beziehung zwischen Eingabewerten (Attributen) und den zugehörigen Ausgabewerten anhand des Trainingsdatensatzes.
- **Validierung:**
  - Der gesamte Datensatz wird in ein **Training Set** und ein **Test Set** aufgeteilt.
  - Der eigentliche Lernprozess zur Vorhersage der Zielvariable basiert auf dem **Training Set**.
  - Die Evaluation des trainierten Modells erfolgt mithilfe des **Testdatensatzes**.
  - Dadurch kann sichergestellt werden, dass die Bewertungsgrößen (Genauigkeit, Fehlerrate) anhand bisher unbekannter Daten bestimmt werden.

## Unüberwachtes Lernen (Unsupervised Learning)

- **Anwendung:** Vor allem dann, wenn ein Datensatz keine präzisen Ausgabewerte aufweist.
- **Ziel:** Durch Anwendung von Unsupervised Learning basierend auf den Eingabedaten bisher unbekannte Muster und Zusammenhänge abzuleiten.

## Bestärkendes Lernen (Reinforcement Learning)

- **Lernprozess:** Basiert auf Formen der Belohnung und Bestrafung.
- **Ziel:** Nutzen zu maximieren.
- **Hinweis:** Unsupervised und Reinforcement Learning werden im Folgenden nicht weiter behandelt.

## Ziel des überwachten Lernverfahrens

- Ausgabewerte anhand der vorhandenen Eingabewerte (den Attributen) mit möglichst hoher Genauigkeit vorherzusagen.

### Traditional Programming:



### Machine Learning:



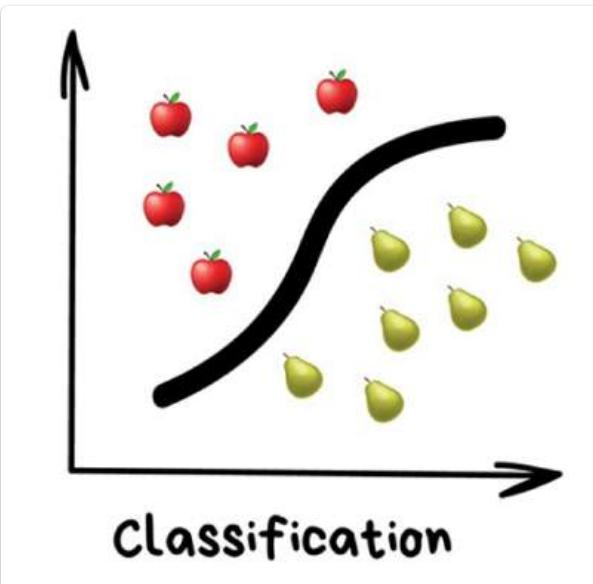
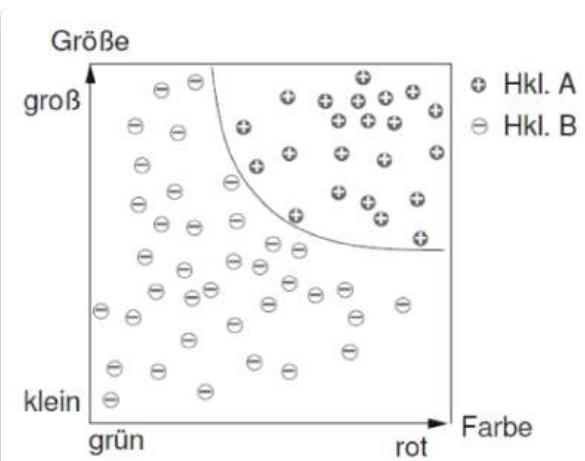
## Klassifikation

- **Häufigste Anwendung in der Computer Vision:** Klassifikation anhand von Merkmalen (Features).
- **Merkmale und Klassenzuordnung:** Im Vorhinein bekannt, die Daten sind "gelabelt".
- **Beispiel (Obstbauer):**
  - Geerntete Äpfel sollen automatisch in die Handelsklassen A und B eingeteilt werden.
  - Sortieranlage misst für jeden Apfel zwei Merkmale (Features): Größe und Farbe.
  - Aufgabe: Entscheidung, zu welcher der beiden Klassen der Apfel gehört.
  - **Typische Klassifikationsaufgabe.**
  -
- **Klassifikatoren (Classifier):** Systeme, welche in der Lage sind, Merkmalsvektoren in eine endliche Anzahl von Klassen einzuteilen.
- **Trainingsdaten-Erstellung:**
  - Zur Einstellung der Maschine werden Äpfel von einem Fachmann händisch verlesen (klassifiziert).
  - Die Messwerte (Größe und Farbe) werden zusammen mit der Klasse in einer Tabelle eingetragen.
  - Größe: Durchmesser in Zentimetern.
  - Farbe: Zahlenwert zwischen 0 (grün) und 1 (rot).
- **Aufgabe beim maschinellen Lernen:** Aus den gesammelten klassifizierten Daten eine Funktion zu generieren, die für neue Äpfel aus den beiden Features (Größe und Farbe) den Wert der Klasse (A oder B) berechnet.

Größe [cm]	8	8	6	3	...
Farbe	0.1	0.3	0.9	0.8	...
Handelsklasse	B	A	A	B	...

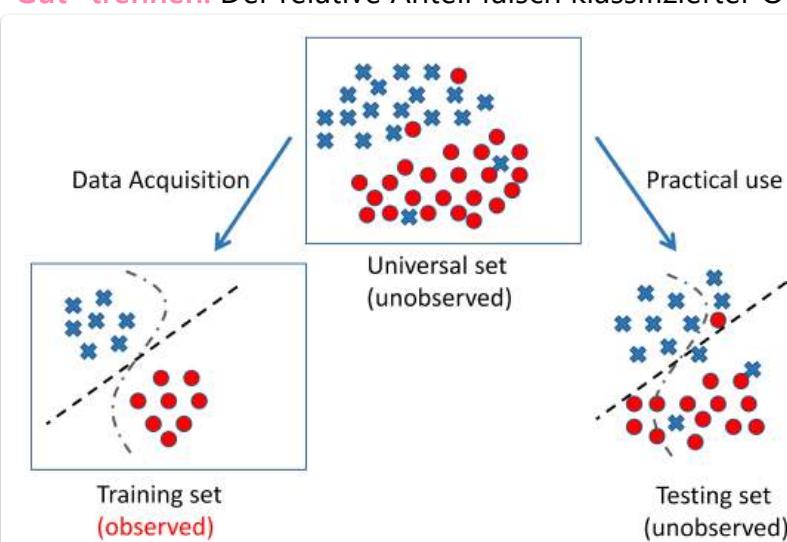
## Funktion zur Klassifikation

- Die in Abbildung 46 eingezeichnete Trennlinie repräsentiert eine Funktion, die die Klassifizierung vornimmt.
- **Klassenzuweisung:**
  - Äpfel mit Merkmalsvektor links unterhalb der Linie: Klasse B.
  - Alle anderen Äpfel: Klasse A.
- **Einfaches Beispiel:** Die Trennlinie ist in diesem Fall leicht zu finden.



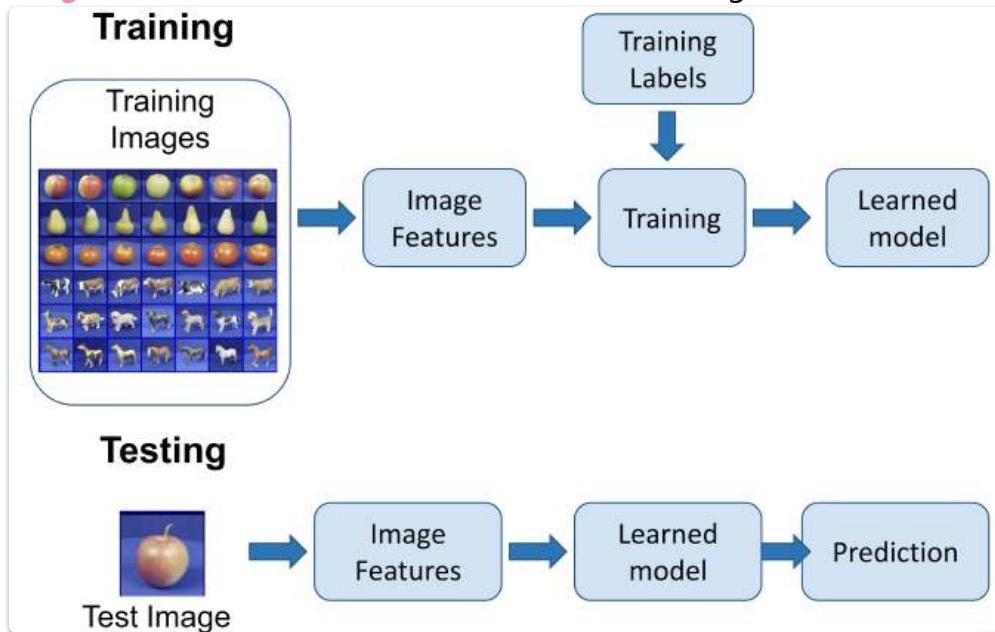
## Herausforderungen bei mehrdimensionalen Daten

- **Schwierigkeit bei vielen Merkmalen:** Die Aufgabe wird deutlich schwieriger und weniger anschaulich, wenn Objekte durch mehr als zwei Merkmale beschrieben werden.
- **Computer Vision:** Nutzt oft viele Merkmale, die aus Bilddaten abgeleitet werden.
- **n Merkmale:** Die Aufgabe besteht darin, im n-dimensionalen Merkmalsraum eine (n-1)-dimensionale Hyperfläche zu finden, welche die Klassen möglichst gut trennt.
- **"Gut" trennen:** Der relative Anteil falsch klassifizierter Objekte soll möglichst klein sein.



## Klassifikator als Abbildung

- Ein Klassifikator bildet einen Merkmalsvektor auf einen Klassenwert ab.
- **Feste Anzahl von Alternativen:** Meist eine kleine Anzahl möglicher Klassen.
- **Zielfunktion:** Diese Abbildung wird auch Zielfunktion genannt.
- **Aufgabe des Lernverfahrens:** Eine solche Abbildung zu lernen.

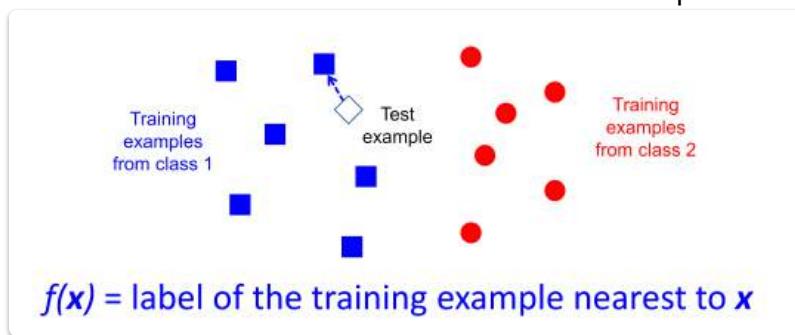


## Klassifikationsalgorithmen im Maschinellen Lernen

- **Vielfalt:** Es existiert eine Vielzahl von Klassifikationsalgorithmen im Bereich des maschinellen Lernens.
- **Überblick über geläufige überwachte Lernverfahren:**

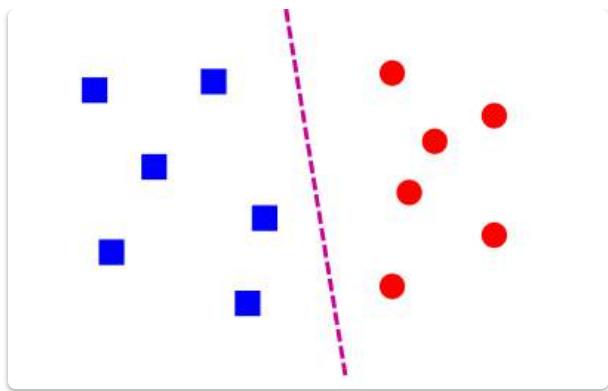
### Nearest Neighbor Algorithmus

- **Grundgedanke:** Ein Datenpunkt wird der Klasse zugeordnet, die durch den Mehrheitsentscheid seiner benachbarten Datenpunkte bestimmt wird.



### Lineare Gleichung

- Finden einer linearen Gleichung die die Klassen trennt



## Entscheidungsbäume (Decision Trees)

- **Klassenzuordnung:** Instanzen werden anhand ihres Pfades durch den Baum (Wurzelknoten und innere Knoten mit Entscheidungsregeln) einer Klasse zugeteilt.

## Random Forest Algorithmus

- **Erweiterung der Entscheidungsbäume.**
- **Klassenzuordnung:** Erfolgt durch den Mehrheitsbeschluss einer Vielzahl von unabhängigen Entscheidungsbäumen.

## Support Vector Machine (SVM)

- **Trennung der Klassen:** Verwendet eine Hyperebene, welche die Klassen mit dem möglichst größten Abstand voneinander trennt (Maximum-Margin-Klassifikator).

## Boosting-Verfahren (z.B. AdaBoost)

- **Ziel:** Steigerung der Gesamtleistung.
- **Funktionsweise:** Kombiniert eine Vielzahl von "schwachen" Klassifikatoren durch einen gewichteten Mehrheitsentscheid zu einem einzigen, stärkeren Klassifikator.

## Künstliche Neuronale Netze (NN)

- **Bedeutung:** Haben sich aufgrund der wachsenden Komplexität und Menge der Datensätze etabliert.
- **Weitere Behandlung:** Werden später noch genauer betrachtet.

## Generalisierung

---

[EVC\\_Skriptum\\_CV, p.56](#)



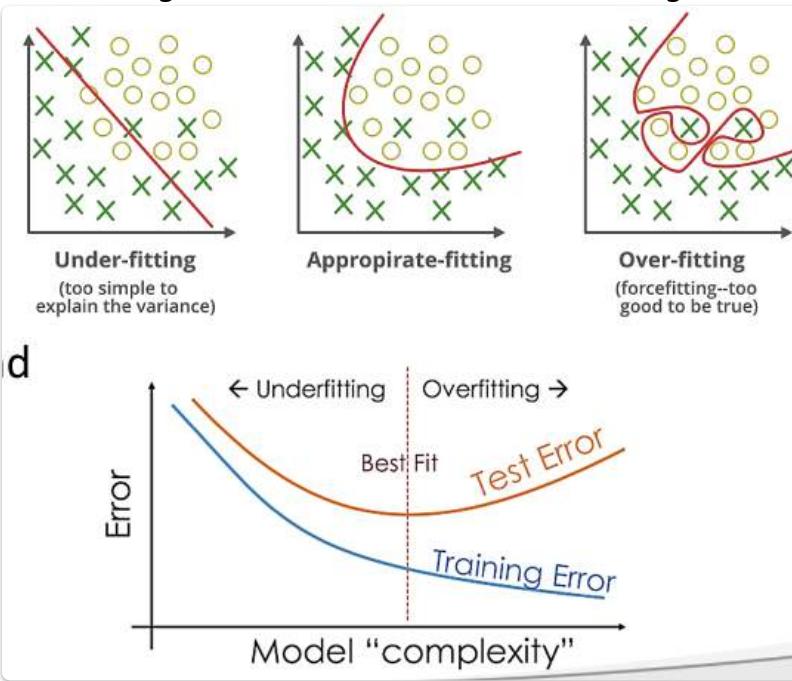
Training set (labels known)



Test set (labels unknown)

- **Überanpassung (Overfitting) - Auswirkungen:**

- Das Modell lernt nicht die generalisierbaren Muster, sondern die spezifischen Details und das Rauschen der Trainingsdaten.
- Führt zu einer hohen Genauigkeit auf den Trainingsdaten, aber einer geringen Genauigkeit auf neuen, ungesiehten Daten (Testdaten).
- Die Leistung des Modells auf realen Anwendungen ist schlecht.

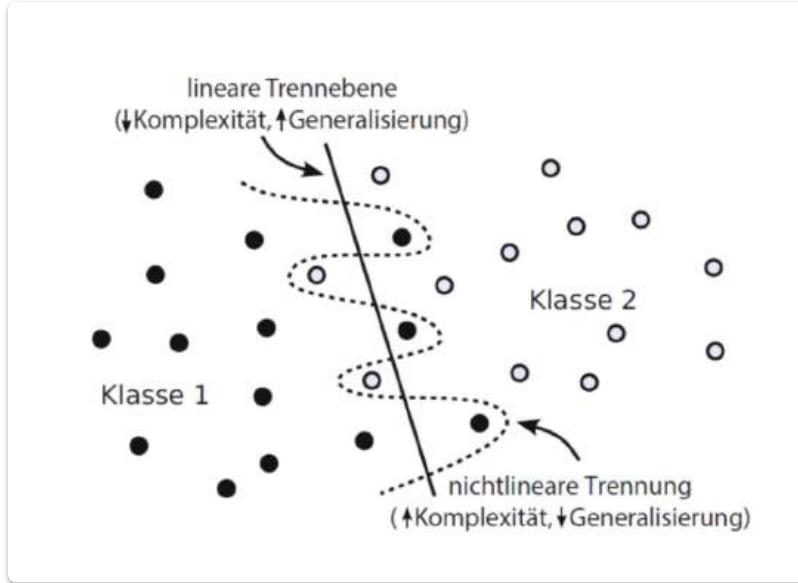


- **Generalisierung - Das Ziel:**

- Ein gutes Modell soll in der Lage sein, Muster zu erkennen, die in den Trainingsdaten vorhanden sind und diese Muster auch auf neue, unbekannte Daten anzuwenden.
- Eine hohe Generalisierungsfähigkeit ist das primäre Ziel beim Entwickeln von Machine-Learning-Modellen.

- **Modellkomplexität - Der Balanceakt:**

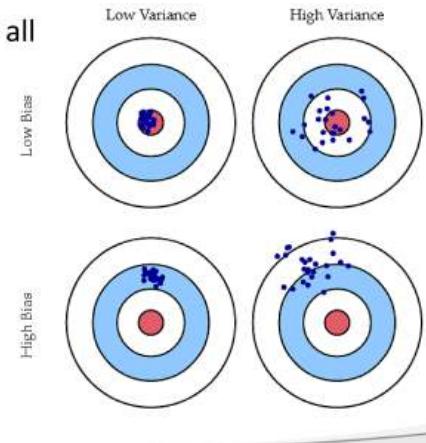
- **Zu einfache Modelle (Underfitting):** Können die zugrundeliegenden Zusammenhänge in den Daten nicht erfassen. Sowohl Trainings- als auch Testdaten werden schlecht vorhergesagt.
- **Zu komplexe Modelle (Overfitting):** Lernen die Trainingsdaten zu gut, inklusive Rauschen. Gute Leistung auf Trainingsdaten, schlechte Leistung auf Testdaten.
- **Die goldene Mitte:** Ein Modell mit der richtigen Komplexität, das die relevanten Muster erfasst, ohne sich zu stark an die Trainingsdaten anzupassen.
- **Praktische Implikationen:**
  - Die Wahl der Modellarchitektur und der Hyperparameter beeinflusst die Modellkomplexität maßgeblich.
  - Techniken wie Kreuzvalidierung (Cross-Validation) werden eingesetzt, um die Generalisierungsfähigkeit eines Modells auf unabhängigen Daten zu schätzen und Überanpassung zu erkennen.
  - Regularisierungsmethoden können verwendet werden, um die Komplexität eines Modells zu reduzieren und die Generalisierung zu verbessern.



## Generalisierung - Bias-Variance Tradeoff

- **Bias (Verzerrung):**
  - Maß für den Fehler aufgrund vereinfachter Annahmen im Modell.
  - Hoher Bias bedeutet, dass das Modell die zugrundeliegenden Muster in den Daten nicht gut erfassst (Underfitting).
  - Einfache Modelle haben tendenziell einen hohen Bias.
  - Ein komplexes Modell hat einen niedrigen Bias, da es flexibler ist und sich besser an die Trainingsdaten anpassen kann.

- **Bias:** how much does the **average model** over all training sets **differ** from the **true model**?
- **Variance:** how much **models** estimated from different training sets **differ from each other**



- **Variance (Varianz):**

- Maß für die Sensitivität der Modellschätzung gegenüber Schwankungen in den Trainingsdaten.
- Hohe Varianz bedeutet, dass das Modell stark auf spezifische Details und Rauschen in den Trainingsdaten reagiert (Overfitting).
- Komplexe Modelle haben tendenziell eine hohe Varianz.
- Ein einfaches Modell hat eine niedrige Varianz, da seine Schätzungen weniger stark durch Änderungen in den Trainingsdaten beeinflusst werden.

- **Bias-Variance Tradeoff:**

- Das Ziel ist es, ein Modell zu finden, das einen niedrigen Bias und eine niedrige Varianz aufweist, um eine gute Generalisierung zu erreichen.
- Es besteht ein Tradeoff, da das Reduzieren des Bias oft zu einer Erhöhung der Varianz führt und umgekehrt.
- **Gesamtfehler = Bias<sup>2</sup> + Varianz + Irreduzierbarer Fehler** (Der irreduzierbare Fehler ist durch die Daten selbst bedingt und kann durch das Modell nicht beeinflusst werden).

- **Komplexität und Bias-Varianz:**

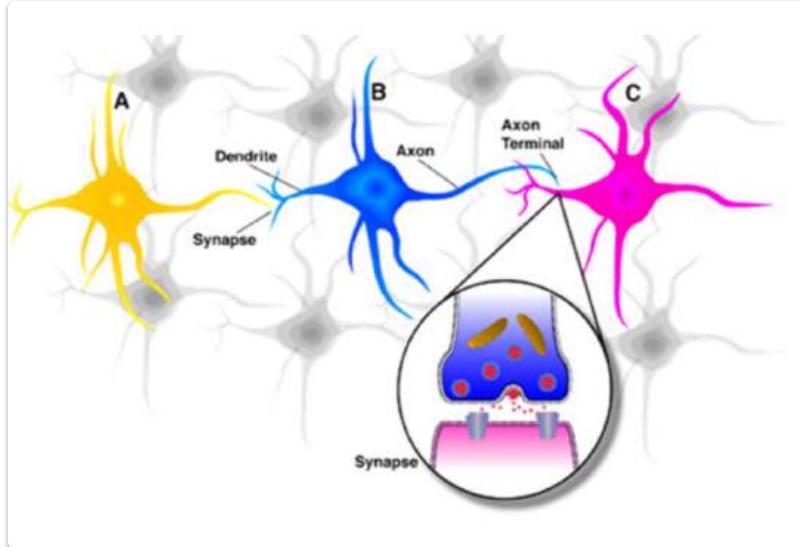
- **Geringe Komplexität:** Hoher Bias, niedrige Varianz (Underfitting).
- **Hohe Komplexität:** Niedriger Bias, hohe Varianz (Overfitting).
- **Optimale Komplexität:** Findet ein Gleichgewicht zwischen Bias und Varianz, was zu einer guten Generalisierung führt.

- **Praktische Konsequenzen:**

- Bei der Modellentwicklung müssen wir versuchen, die optimale Komplexität zu finden.
- Die Leistung des Modells auf unabhängigen Validierungsdaten ist ein guter Indikator für das Vorliegen von Bias oder Varianz Problemen.
- Techniken wie Regularisierung können helfen, die Varianz zu reduzieren, während komplexe Modelle den Bias reduzieren können (bis zu einem gewissen Punkt).

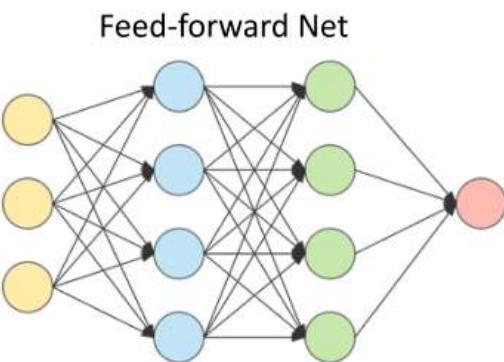
## Künstliche Neuronale Netze (NN) - Biologische Inspiration

- **Biologisches Vorbild:** Neuronale Netze im Gehirn von Menschen und Tieren.
- **Komplexität des Gehirns:** Ca. 10 bis 100 Milliarden Nervenzellen im menschlichen Gehirn.
- **Grundlage für Intelligenz:** Komplexe Verschaltung und Adaptivität der Neuronen ermöglichen Lernen und Anpassung.



- **Struktur und Funktion eines biologischen Neurons (vereinfacht):**
  - **Zellkörper (A, B, C):** Enthält den Zellkern und andere Organellen. Fungiert als Speicher für kleine elektrische Spannungen (ähnlich Kondensator/Akku).
  - **Dendriten:** Verzweigte Fortsätze, die eingehende Signale von anderen Neuronen über Synapsen empfangen.
  - **Axon:** Langer Fortsatz, der Ausgangssignale (Spannungsimpulse) zu anderen Neuronen über Synapsen weiterleitet.
  - **Feuern des Neurons:** Wenn die im Zellkörper gespeicherte Spannung einen bestimmten Schwellwert überschreitet, entlädt sich das Neuron und sendet einen Spannungsimpuls über das Axon.
  - **Synapsen:** Kontaktstellen zwischen dem Axon eines Neurons und den Dendriten eines anderen Neurons. Hier wird das Signal übertragen.
- **Lernen im biologischen neuronalen Netz:**
  - **Adaptivität der Synapsen:** Nicht die Neuronen selbst, sondern die Verbindungen (Synapsen) sind adaptiv.
  - **Veränderung der Verbindungsstärke:** Die Stärke der synaptischen Verbindungen kann sich verändern. Dies ermöglicht Lernen und Anpassung.
- **Mathematisches Modell der NN:**

- In short: Neural Networks (NN)
- Multi-layer fully-connected NN
- Elements:
  - Neurons (nodes)
  - Synapses (weights)
- Consist of:
  - Input layer,
  - Multiple hidden layers,
  - Output layer.
- Every node in one layer is connected to every other node in the next layer.

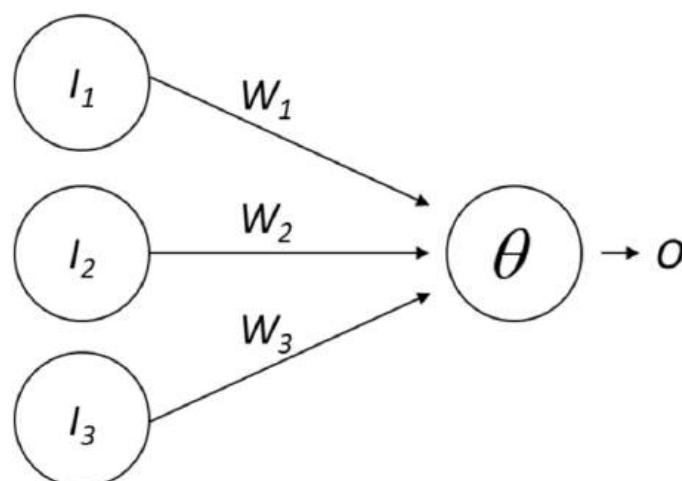


- **Inspiration:** Das biologische Modell diente als Vorbild.
- **McCulloch und Pitts (1943):** Stellten das erste mathematische Modell eines Neurons als grundlegendes Schaltelement vor.
- **Grundlage für NN:** Dieses Modell war die Basis für den Bau künstlicher neuronaler Netze.
- **Bestandteile des mathematischen Modells:**
  - **Neuronen (Knoten):** Entsprechen den Nervenzellen.
  - **Synapsen (Kanten):** Verbindungen zwischen den Neuronen.
  - **Gewichte:** Den Kanten (Synapsen) werden Gewichte zugewiesen.
- **Adaption:** Durch die Anpassung der Gewichte kann das "Verhalten" des künstlichen Neurons verändert werden (entspricht der Adaptivität der biologischen Synapsen).

## Das Perceptron - Ein frühes künstliches neuronales Netz

[EVC\\_Skriptum\\_CV, p.57](#)

- **Entwicklung:** 1958 von Frank Rosenblatt vorgestellt.
- **Struktur:** Besteht aus einem einzelnen künstlichen Neuron und einer Reihe von Eingängen ( $I_1$  bis  $I_k$ ).



- **Analogie zum biologischen Neuron:**

- **Eingänge ( $I_1$  bis  $I_k$ ):** Entsprechen den Dendriten.
- **Gewichte ( $w_1$  bis  $w_k$ ):** Werden mit den Eingangssignalen multipliziert. Entsprechen der Verstärkung oder Abschwächung natürlicher Signale durch Synapsen.
- **Gewichtete Summe der Eingaben:**  $\Phi(x) = \sum_i w_i I_i$
- **Aktivierungsfunktion ( $f$ ):** Wird auf die gewichtete Summe angewendet, um die Ausgabe des Neurons zu bestimmen:  $O = f(\sum_i w_i I_i)$ .
- **Ausgabe ( $O$ ):** Wird über "synaptische Gewichte" an nachgeschaltete Neuronen weitergegeben.

## Aktivierungsfunktionen

- **Vielfalt:** Es existieren verschiedene Möglichkeiten für die Aktivierungsfunktion  $f$ .
- **Einfachste Form: Identität**
  - $f(x) = x$
  - Das Neuron gibt lediglich die gewichtete Summe der Eingabewerte weiter.
  - **Problem:** Führt zu Konvergenzproblemen, da die Funktion nicht beschränkt ist und die Funktionswerte unbegrenzt wachsen können.

## Schwellwertfunktion im Perceptron

- **Motivation:** Um die unbegrenzte Ausgabe der Identitätsfunktion zu vermeiden, wird eine Schwellwertfunktion eingesetzt.
- **Funktionsweise:** Die gewichtete Summe der Eingangssignale  $\Phi(x) = \sum_i w_i I_i$  wird mit einem Schwellwert  $\theta$  verglichen.
- **Ausgabe ( $O$ ):**

$$O = \begin{cases} 1 & \text{falls } \sum_i w_i I_i + \theta \geq 0 \\ 0 & \text{sonst} \end{cases}$$

- **Abhängigkeit der Ausgabe:** Von den Eingangssignalen ( $I$ ), den zu lernenden Gewichten ( $w$ ) und dem Schwellwert ( $\theta$ ).
- **Ziel des Perceptrons (Klassifikation):** Trennung von Daten, die zu zwei unterschiedlichen Klassen gehören.
- **Lernaufgabe:** Anpassung der Gewichte  $w_i$ , sodass das Neuron bei Daten der ersten Klasse 0 und bei Daten der zweiten Klasse 1 ausgibt (oder umgekehrt).

## Perceptron-Lernalgorithmus ( $\delta$ -Regel / Widrow-Hoff-Regel)

- **Ziel:** Anpassung der Gewichte, um die gewünschte Ausgabe zu erzielen.
- **$\delta$ -Regel:** Die Gewichtsänderung  $\Delta w_i(t)$  zum Zeitpunkt  $t$  wird basierend auf der Differenz zwischen der gewünschten Ausgabe ( $T$ ) und der tatsächlichen Ausgabe ( $O$ ) berechnet:

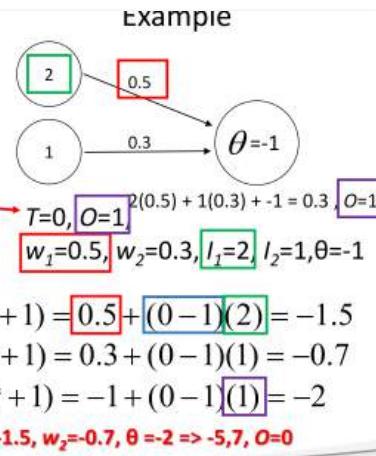
$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = (T - O)I_i$$

- **Zwei Schlüsselkonzepte der  $\delta$ -Regel:**

1. **Fehlerbasierte Anpassung:** Die Gewichtsanpassung basiert auf dem Fehler ( $T - O$ ).
2. **Eingabeabhängigkeit:** Die Änderung hängt auch von der Eingabe  $I_i$  (der Ausgabe des vorherigen "Neurons") ab.

1. Randomly **assign weights** (between 0-1)
2. Present inputs from **training data**
3. **Get output  $O$ , nudge weights to give results toward desired output  $T$**
4. **Repeat;** stop when no errors, or enough epochs completed
  - Weights include Threshold
    - $T$ =Desired,  $w_i(t+1) = w_i(t) + \Delta w_i(t)$
    - $O$ =Actual output.  $\Delta w_i(t) = (T - O)I_i$
  - If we present this input again, output 0 instead  $w_1=-1.5, w_2=-0.7, \theta=-2 \Rightarrow -5.7, O=0$



## Konvergenz des Perceptron-Lernalgorithmus

- **Rosenblatts Theorem:** Der Lernalgorithmus des Perceptrons konvergiert in endlicher Zeit, wenn die Daten linear separierbar sind.
- **Implikation:** Das Perceptron kann alles lernen, was es repräsentieren kann, in endlicher Zeit.

## Beschränkungen des Perceptrons

- **Nicht-lineare Separierbarkeit:** Ein einschichtiges Perceptron kann keine Funktionen lernen, die nicht linear separierbar sind.
- **Lineare Separierbarkeit:** Die Eigenschaft, dass Daten im Raum durch Hyperebenen (in 2D: Geraden, in 3D: Ebenen) voneinander getrennt werden können.

## Mehrschichtige Perceptrons

- **Zweischichtige Perceptrons:** Können konvexe Mengen trennen.
- **Mehrschichtige Perceptron-Netze:** Können beliebige Mengen trennen. Dies ist ein wichtiger Schritt zur Überwindung der Beschränkungen des einfachen Perceptrons.

## Aktivierungsfunktionen für stetige Neuronen

- **Problem der Stufenfunktion:** Für binäre Neuronen (Ausgabe 0 oder 1) sinnvoll, erzeugt aber eine Unstetigkeit bei stetigen Neuronen (Aktivierungen zwischen 0 und 1).
- **Lösung: Sigmoid-Funktion:** Eine Möglichkeit, die Unstetigkeit zu glätten.
- **Beispiel einer Sigmoid-Funktion:**

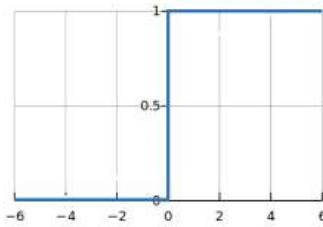
$$O = \frac{1}{1 + e^{-(\Phi(x)+\theta)}}$$

wobei  $\Phi(x) = \sum_i w_i I_i$  die gewichtete Summe der Eingaben und  $\theta$  der Schwellwert ist.

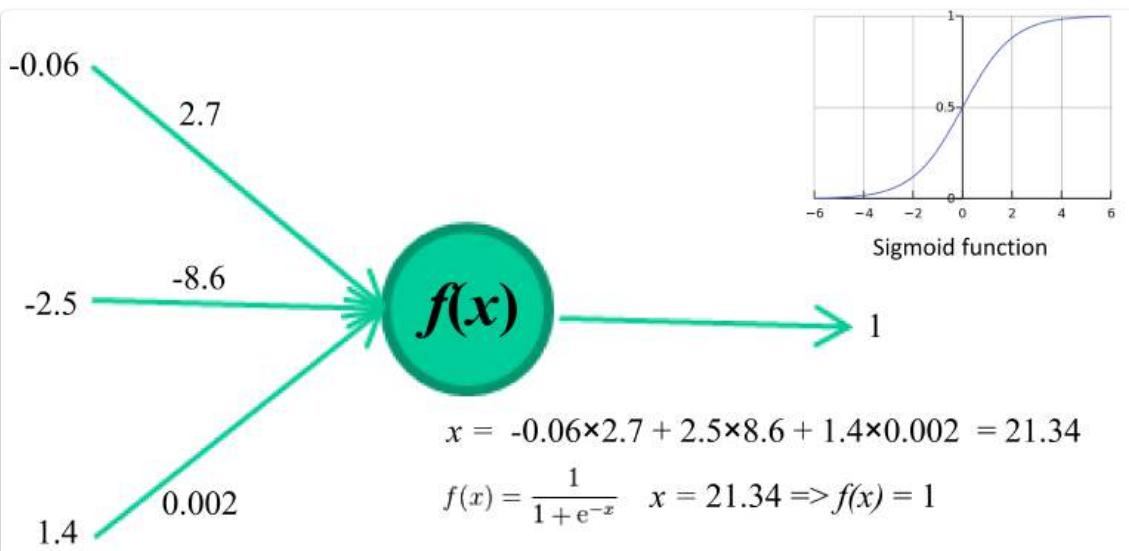
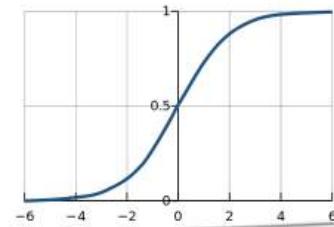
- **Eigenschaften der Sigmoid-Funktion:**
  - **Annähernd linear:** Verhält sich in der Nähe des kritischen Bereichs um den Schwellwert  $\theta$  annähernd linear.
  - **Asymptotisch beschränkt:** Die Ausgabe liegt immer zwischen 0 und 1. Dies verhindert das Problem des unbegrenzten Wachstums der Aktivierungen.
- **Vorteil gegenüber der Stufenfunktion:** Ermöglicht differenzierbare Ausgaben, was für viele Lernalgorithmen in neuronalen Netzen (insbesondere für das Training mit Gradientenabstieg) entscheidend ist.

$$\Delta w_k = cI_k(T_j - O_j)f'(ActivationFunction)$$

Old:  $O = \begin{cases} 1: \sum_i w_i I_i + \theta > 0 \\ 0: otherwise \end{cases}$



New:  $O = \frac{1}{1 + e^{-\sum_i w_i I_i + \Theta}}$



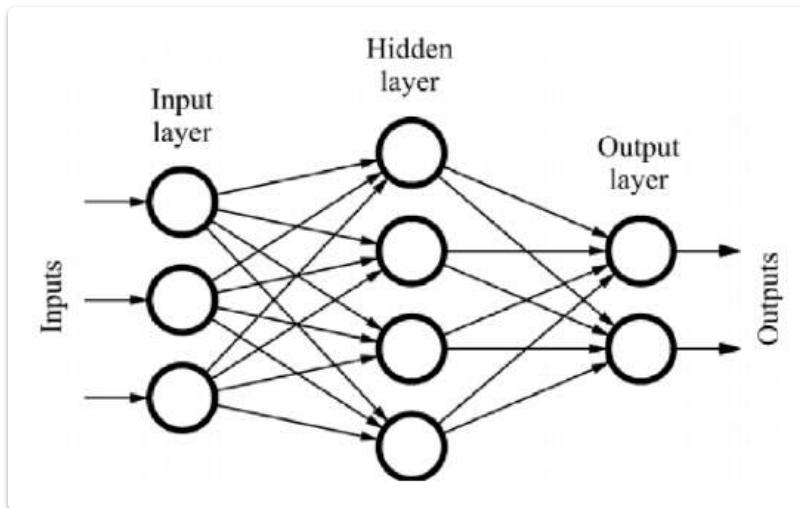
Ein animiertes Beispiel findet man in den Slides: [EVC-CV11-Deep Learning\\_2025S\\_Slides](#), p.42

## Multilayer Perceptron (MLP)

[EVC\\_Skriptum\\_CV](#), p.58

- **Typischer Aufbau:** Besteht aus mehreren Schichten von Verarbeitungseinheiten.
- **Beispiel (Abbildung 49):**
  - **Eingangsschicht (Input Layer):** Empfängt die Eingabedaten.
  - **Verborgene Schicht (Hidden Layer):** Eine oder mehrere Zwischenschichten, die komplexe Muster in den Daten verarbeiten.

- **Ausgabeschicht (Output Layer):** Liefert das Ergebnis der Verarbeitung (z.B. Klassifikation oder Regression).
- **Verbindungen:** Zwischen den Schichten befinden sich Lagen von Verbindungen, die mit Gewichten versehen sind.



- **Variationen der Struktur:**
  - **Shortcut-Verbindungen:** Direkte Verbindungen zwischen der Eingangsschicht und der Ausgabeschicht.
  - **Mehrere verborgene Schichten (Deep Neural Networks):** Ermöglichen die Modellierung noch komplexerer Beziehungen in den Daten.
- **Arbeitsweise: Vorwärtsvermittlung (Feed-forward Network):**
  - Die Aktivierung erfolgt schichtweise von der Eingabe- zur Ausgabeschicht.
  - **Keine Rückkopplungen:** Signale fließen nur in eine Richtung.

## Training von Multilayer Perceptron (MLP)

- **Überwachter Lernvorgang:** Die Gewichtsfaktoren zwischen den Verarbeitungseinheiten werden während des Trainings angepasst.
- **Lernzyklus (Epoche):** Alle Muster des Trainingsdatensatzes werden vom Netz klassifiziert (oder für Regression verwendet).
- **Fehlerberechnung:** Die Ergebnisse des Netzes werden mit den Soll-Werten (Labels) des Trainingsdatensatzes verglichen.
- **Gewichtsanpassung:** Die Gewichtsfaktoren werden so angepasst, dass der Fehler zwischen der Netzwerkausgabe und dem Soll-Wert minimiert wird.
- **Ziel:** Nach einer Reihe von Trainingszyklen soll der Trainingsdatensatz mit einer vorgegebenen Genauigkeit reproduziert werden.

## Backpropagation-Training (Fehler-Rückvermittlung)

- **Bekanntestes Trainingsverfahren für MLPs.**
- **Prinzip:** In jedem Lernschritt werden die Gewichte in Richtung des abnehmenden Fehlers verändert (Gradientenabstieg).
- **Ablauf eines Lernschritts:**

1. **Vorwärtslauf (Forward Pass):** Ein Eingangsmuster wird an das Netz angelegt und die Aktivierungen werden schichtweise bis zur Ausgabeschicht berechnet.
2. **Vergleich mit Soll-Output:** Der Zustand der Ausgabeschicht wird mit dem bekannten Soll-Output für dieses Muster verglichen.
3. **Fehlerberechnung:** Die Abweichung (der Fehler) zwischen der tatsächlichen und der gewünschten Ausgabe wird berechnet.
4. **Rückwärtslauf (Backward Pass):** Der Fehler wird vom Ende des Netzes (Ausgabeschicht) zurück zu den vorherigen Schichten propagiert.
5. **Gewichtsaktualisierung:** Die Gewichte jeder Verbindung im Netz werden proportional zum Beitrag dieser Verbindung zum Fehler angepasst.

## Generalisierte Delta-Regel und Fehlerrückvermittlung

- **Problem bei mehrschichtigen Netzen:** Für die verborgenen Schichten ist kein direkter Soll-Zustand bekannt. Die einfache Delta-Regel ist daher nicht direkt anwendbar.
- **Lösung:** Die Generalisierte Delta-Regel mit Fehlerrückvermittlung (Backpropagation).
- **Kernidee:** Der Fehler der Ausgabeschicht wird verwendet, um einen "virtuellen Fehler" für die Neuronen in den verborgenen Schichten zu berechnen. Dieser virtuelle Fehler gibt an, wie stark die Aktivität eines verborgenen Neurons zum Fehler in der Ausgabeschicht beigetragen hat.
- **Anwendung:** Mithilfe dieses virtuellen Fehlers können dann auch die Gewichte der Verbindungen, die in die verborgenen Neuronen führen, angepasst werden.

## Zielfunktion des Backpropagation-Algorithmus

- **Ziel:** Minimierung der Summe der quadratischen Fehler (Least Mean Squares, LMS) bei der Klassifikation aller Trainingsmuster.
- **Fehlerfunktion (LMS):**

$$Distance(LMS) = \frac{1}{n} \sum_{p=1}^n (T_p - O_p)^2$$

- $T_p$ : Soll-Ausgangswert für das  $p$ -te von  $n$  Trainingsbeispielen.
- $O_p$ : Tatsächlicher Ausgangswert des Netzes für das  $p$ -te Trainingsbeispiel.
- **Gesucht:** Das globale Minimum dieser Gesamtfehlerfunktion.

## Gradientenabstieg

- **Prinzip der Gewichtsanpassung:** Die Gewichte werden stets in Richtung des abnehmenden Fehlers verändert, d.h., entgegen dem Gradienten des Fehlers.
- **Ursprüngliche Delta-Regel:** Eine frühe Form dieser Idee.

## Kernidee des Backpropagation-Verfahrens

- **Fehler als Funktion der Gewichte:** Der Ausgangswert der Ausgabeeinheiten (und damit der Fehler  $F$ ) ist eine Funktion ihrer Eingabewerte. Diese Eingabewerte sind wiederum Funktionen der Ausgänge der vorhergehenden (verdeckten) Schicht und der Gewichte zwischen diesen Schichten.
- **Kettenregel:** Die Ausgänge der Zwischenschicht sind eine Funktion der Gewichte zwischen Eingangs- und Zwischenschicht. Daraus folgt, dass der Fehler  $F$  letztendlich auch eine Funktion dieser Gewichte ist.
- **Partielle Differentiation:** Durch Anwendung der Kettenregel ist es möglich, auch für die Zwischenschichten einen Fehler zu definieren.
- **Gradient für Zwischenschichten:** Aus diesem Fehler kann ein Gradient berechnet werden, um die Gewichte zwischen den Schichten anzupassen.

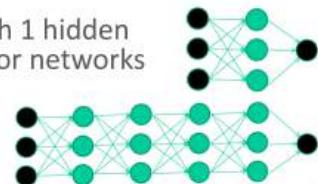
## Fehlerrückvermittlung (Error Backpropagation)

- **Berechnung des Fehlerwerts:** Zuerst für die Ausgabeeinheiten, dann für die verdeckten Einheiten.
- **"Backpropagation":** Die Fehleroptimierung pflanzt sich von hinten (Ausgabeschicht) nach vorne durch das Netz.
- **Implikation:** Die Information, wie die einzelnen Gewichte zum Fehler der Ausgabeschicht beigetragen haben, wird zurück durch das Netzwerk geleitet, um eine sinnvolle Anpassung der Gewichte in allen Schichten zu ermöglichen.

# Deep Learning

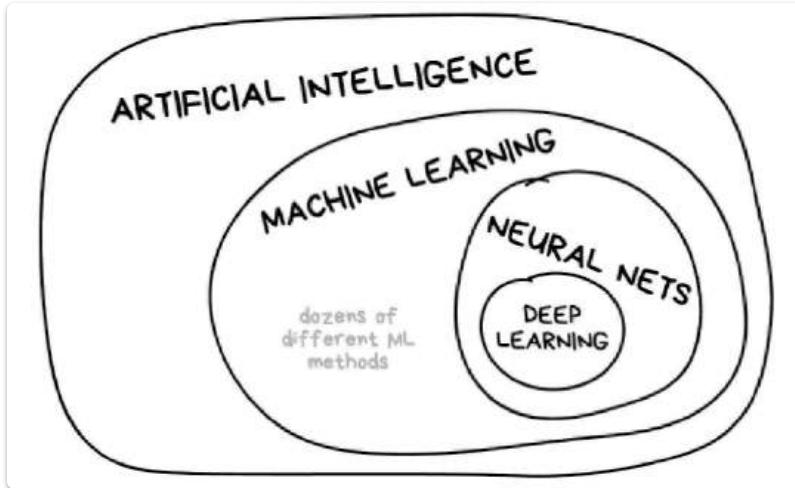
[EVC\\_Skriptum\\_CV, p.58, p.59](#)

1. **What exactly is Deep Learning?**
  - Deep Learning means using a neural network with several layers of nodes between input and output
2. **Why is it generally better than other methods?**
  - The series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.
3. **Multilayer neural networks have been around for 30 years.  
What's actually new?**
  - We had good algorithms for learning weights in networks with 1 hidden layer, but these algorithms are not good at learning weights for networks with more hidden layers.
  - What's new is: algorithms for training many-layer networks



- **Bisherige Ansätze (Machine Learning):** Lernalgorithmen können aus Trainingsdaten komplexe Klassifikationsaufgaben lernen.
- **Manuelle Merkmalsgenerierung:** Die zur Klassifikation verwendeten Merkmale (z.B. Farbe, Kanten, SIFT) mussten bisher manuell von einem "Wissensingenieur" ausgewählt und generiert werden.
  - Ziel: Einen sinnvollen Satz von möglichst wenigen, aussagekräftigen Merkmalen finden.

- Diese Merkmale dienen dann als Eingabe für die Lernalgorithmen.



## Herausforderung: Direkte Verwendung von Sensordaten

- **Idee:** Warum nicht direkt alle verfügbaren Sensordaten verwenden (z.B. Pixel eines Bildes)?
- **Beispiel (RGB-Bild):** Ein Foto mit 10 Millionen Pixeln hätte einen Eingabevektor der Länge 30 Millionen (10 Mio. Pixel x 3 Farbwerte).
- **Problem: Skalierung mit der Dimension der Eingabedaten:**
  - **Trainingszeiten:** Wachsen sehr schnell, oft exponentiell, mit der Dimension der Eingabedaten.
  - **Rechenaufwand:** Um die Rechenzeiten in Grenzen zu halten, müssen die Eingabedaten zuerst auf kurze Merkmalsvektoren abgebildet werden.

## Traditionelle Merkmalsgewinnung

- **Frühe Merkmalsbestimmung:** Merkmale werden frühzeitig anhand manuell definierter Formeln bestimmt.
- **Grundlage:** Meist Expertenwissen, orientiert an der menschlichen Wahrnehmung ("Wie würde der Mensch die Klassifikationsaufgabe lösen?").

## Deep Learning: End-to-End-Learning

- **Herausforderung der manuellen Merkmalsgenerierung:** Für viele komplexe Anwendungen (z.B. Objektklassifikation in Bildern) ist es sehr schwierig bis unmöglich, manuell gute Merkmale zu definieren.
- **Prinzip von Deep Learning:** End-to-End-Learning.
  - **Simultanes Lernen:** Nicht nur die optimale Verarbeitung der Merkmale wird gelernt, sondern auch gleichzeitig die optimale Herleitung dieser Merkmale aus den Rohdaten.
  - **Automatisierte Merkmalsentwicklung:** Das Netzwerk lernt selbstständig, welche Merkmale für die jeweilige Aufgabe relevant sind.

## Deep Learning Architekturen

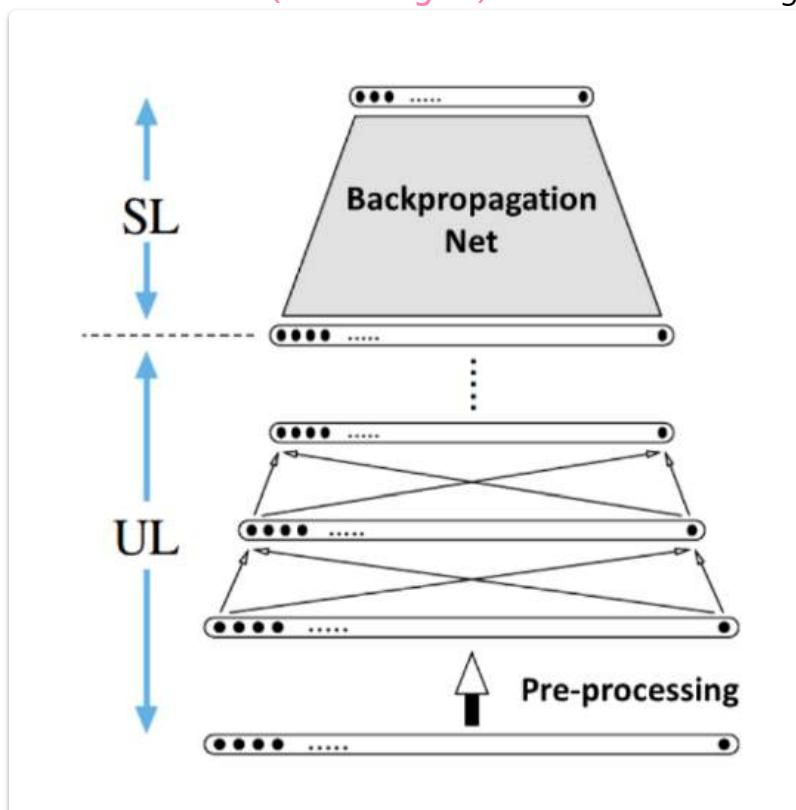
- **Computer Vision:**
  - **Convolutional Neural Networks (CNNs):** Kommen in der Regel zum Einsatz.
  - **Filterkoeffizienten:** Werden gelernt, um die Bilder zu Beginn zu falten (convolved). Diese Faltungsschritte extrahieren automatisch hierarchische Merkmale aus den Bildern (z.B. Kanten, Texturen, Objektteile).
- **Andere Deep Learning Verfahren:**
  - **Recurrent Neural Networks (RNNs):** Gut geeignet für sequenzielle Daten (z.B. Text, Zeitreihen).
  - **Generative Adversarial Networks (GANs):** Werden für generative Aufgaben eingesetzt (z.B. Erzeugung neuer Bilder, Texte).
  - **Variationen dieser Architekturen.**

## Komplexität tiefer neuronaler Netze

- **Anzahl der Schichten:** Architekturen mit bis zu fünfzig oder mehr Schichten sind möglich.
- **Hohe Komplexität:** Die genauen Architekturen sind oft sehr komplex und können hier nicht im Detail dargestellt werden.
- **Ressource für weitere Informationen:** Gute Einführungen finden sich auf [ufldl.stanford.edu/tutorial/](http://ufldl.stanford.edu/tutorial/).

## Deep Learning mit vielen Schichten

- **Aktueller Erfolg:** Erfolgreiche Deep-Learning-Lösungen verwenden viele Schichten von Neuronen.
- **Netzwerkstruktur (Abbildung 50):** Oft in zwei Teile aufgespalten.

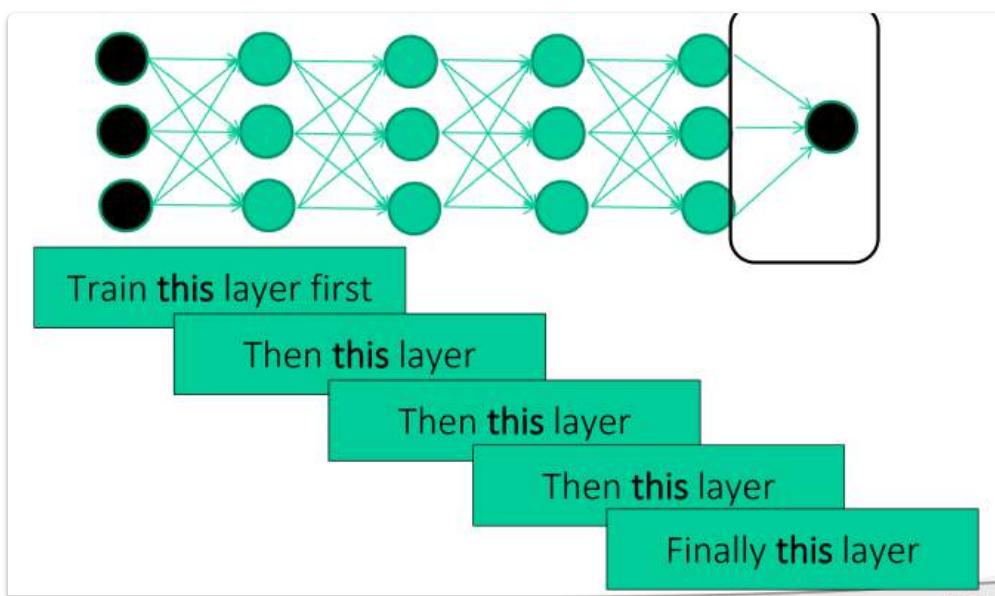


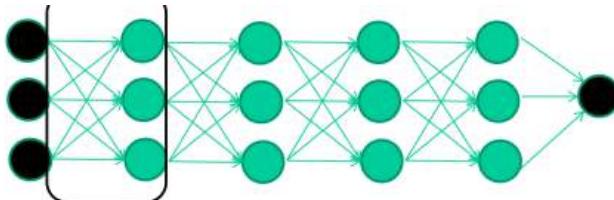
- **Unsupervised Learning (UL) zur Vorverarbeitung:**

- Nach einer Vorverarbeitungsschicht folgen mehrere Lagen, die mit unüberwachtem Lernen vorgenommen werden.
- **Merkmalsrepräsentation:** Jede Lage im UL-Netz repräsentiert Merkmale des Eingabemusters.
- **Hierarchische Merkmale:**
  - **Tiefe Lagen:** Repräsentieren einfache Merkmale (z.B. Kanten oder Linien in verschiedenen Ausrichtungen bei der Objekterkennung in Bildern).
  - **Höhere Lagen:** Können komplexe Merkmale bilden (z.B. das Vorhandensein eines Gesichts).
- **Supervised Learning (SL) zur Klassifikation/Regression:**
  - An das UL-Netz schließt sich ein klassisches überwachtes Lernen an.
  - **Training:** Kann beispielsweise mit Backpropagation trainiert werden.
- **Ablauf des Lernens:**
  1. **UL-Training:** Vortrainieren der Gewichte aller Merkmalschichten (Extraktion der Merkmale).
  2. **SL-Training:** Training des gesamten Netzes (einschließlich der vorgenommenen Schichten) mit überwachten Daten (z.B. Gradientenabstieg).

## Unsupervised Learning der Gewichte und Merkmalsextraktion

- **Ziel des UL-Trainings:** Bildung von Merkmalsgruppen durch das Lernen der Gewichte zu den Merkmalschichten. Hier findet die eigentliche Merkmalsextraktion statt.
- **Eigenschaft der Merkmalsextraktion:** Die Eingabedaten sollen in einen niedrigerdimensionalen Raum abgebildet werden, möglichst ohne (zu viel) Informationsverlust. Dies reduziert die Dimensionalität des Problems für die nachfolgende SL-Phase und kann zu robusteren und effizienteren Modellen führen.



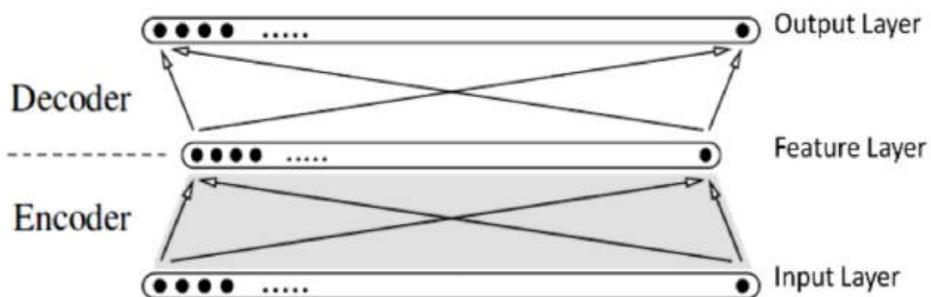


- EACH of the (non-output) layers is trained to be an auto-encoder
- Basically, it is forced to learn good features that describe what comes from the previous layer

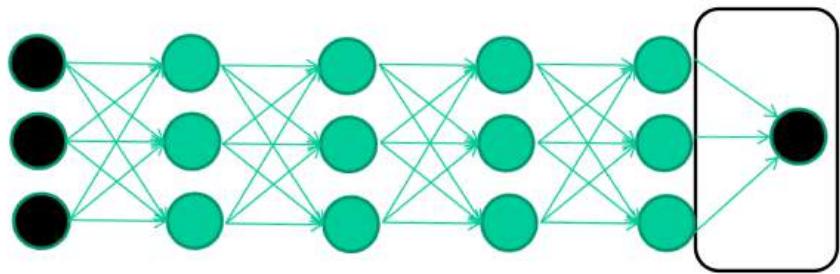
## Autoencoder

EVC\_Skriptum\_CV, p.59

- **Bestimmung der Gewichte der Merkmalschichten (UL):** Erfolgt bei sogenannten Autoencodern nach dem in der Abbildung gezeigten Verfahren.



- **Training der ersten verdeckten Merkmalschicht:**
  - Ein Autoencoder wird mit unüberwachtem Lernen (UL) trainiert.
  - **Ziel des Autoencoders:** Die identische Abbildung aller Eingabevektoren  $x$  auf sich selbst zu lernen (Identitätsfunktion).
  - **Merkmalschicht:** Dient hier als die erste verdeckte Lage (Feature Layer).
- **Nach dem Training des ersten Autoencoders:**
  - Die nicht benötigte Decoderlage (die zweite Lage von Gewichten) wird gelöscht.
  - Die erste Lage der Gewichte (Encoder) wird eingefroren und für die Berechnung der Merkmale in der ersten Lage des UL-Netzes übernommen.
- **Training der zweiten Merkmalschicht:**
  - Mit dieser festen ersten Schicht von Gewichten wird die zweite Merkmalschicht wieder mit dem Autoencoder-Verfahren trainiert.
  - Die Gewichte der Encoder-Seite des zweiten Autoencoders werden eingefroren.
- **Iterativer Prozess:** Dieser Vorgang wird so weitergeführt bis zur letzten Merkmalschicht.
- **Abschluss des unüberwachten Teils des Lernens:** Sobald alle Merkmalschichten auf diese Weise vortrainiert wurden, ist der unüberwachte Teil des Lernens beendet.



- Is trained to predict class based on outputs from previous layers

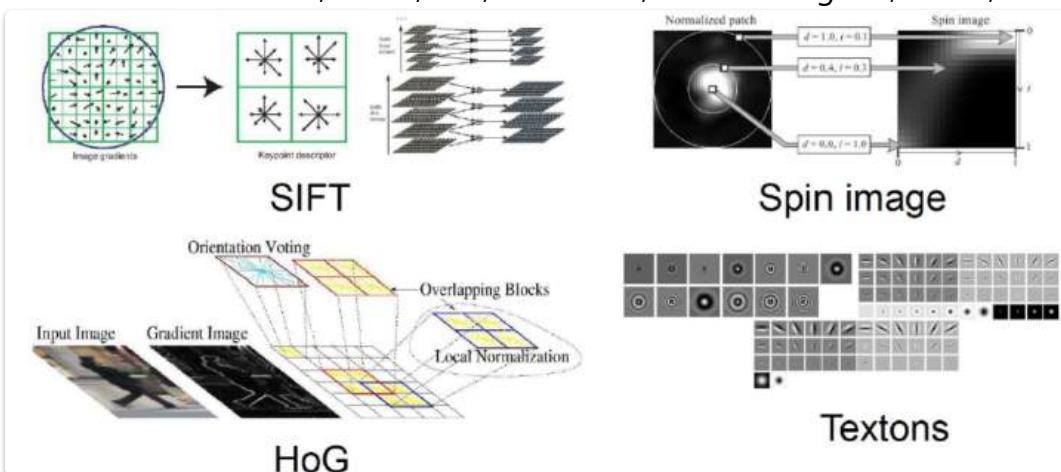
## Pros / Cons von Deeplearning

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• Pros           <ul style="list-style-type: none"> <li>• Not-domain specific</li> <li>• Supervised / Semi-supervised / Unsupervised</li> <li>• Classification / regression in last layer</li> <li>• Simple math</li> <li>• Hip</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• Cons           <ul style="list-style-type: none"> <li>• Lots of meta-parameters</li> <li>• Needs a lot of data</li> <li>• Very computational intensive</li> <li>• Hip</li> </ul> </li> </ul> |
|---|---|

## Weitere VO-Slides:

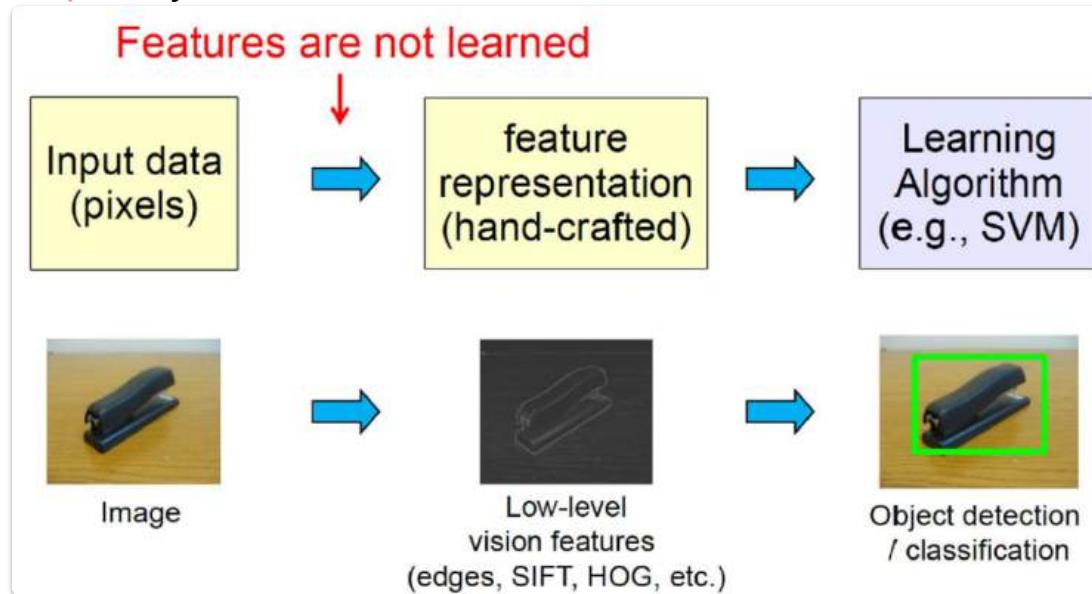
### Traditional Computer Vision Features

- Beispiele für handgefertigte Features:
  - SIFT (Scale-Invariant Feature Transform)
  - Spin image
  - HoG (Histogram of Oriented Gradients)
  - Textons
  - Viele andere wie SURF, MSER, LBP, Color SIFT, Color Histogram, GLOH, ...



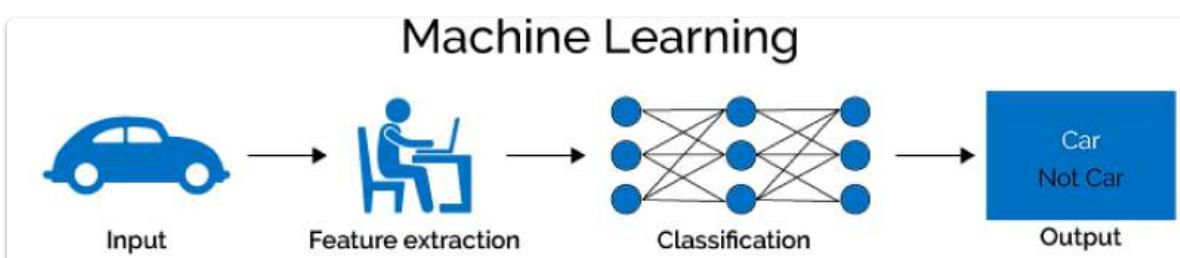
## Traditional Recognition Approach

- Merkmale werden nicht gelernt (Features are not learned).
- Ablauf:
  1. **Input data (pixels)**: Rohdaten, z.B. ein Bild.
  2. **Feature representation (hand-crafted)**: Manuell entworfene Merkmalsrepräsentation (Low-level vision features wie edges, SIFT, HoG, etc.).
  3. **Learning Algorithm (e.g., SVM)**: Ein Lernalgorithmus (z.B. Support Vector Machine) wird auf den extrahierten Merkmalen trainiert.
  4. **Output**: Objekt-Detektion / Klassifikation.



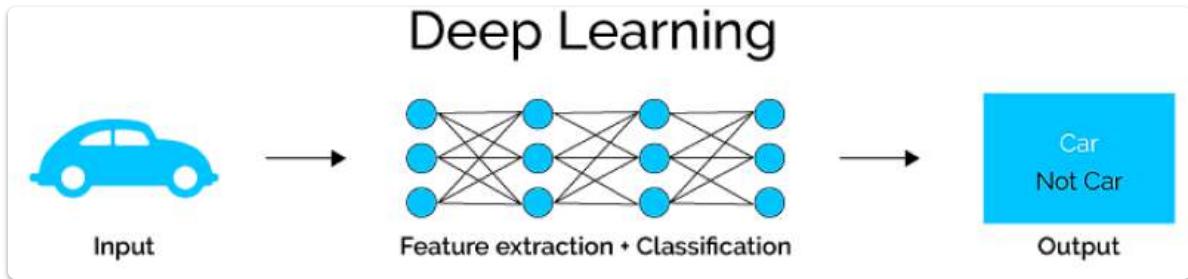
## Difference of Deep Learning to Classic Machine Learning

- **Machine Learning (Klassisch):**
  - Computes features.
  - Has simple and complex features.
  - **Ablauf:**
    1. **Input**: Rohdaten (z.B. ein Auto-Bild).
    2. **Feature extraction**: Manuelle oder algorithmische Extraktion von Merkmalen.
    3. **Classification**: Ein separater Klassifikator lernt, die extrahierten Merkmale den Klassen zuzuordnen.
    4. **Output**: Klassifikation (z.B. "Car" oder "Not Car").



- **Deep Learning:**
  - Does not need human interaction for feature design.
  - **Ablauf:**

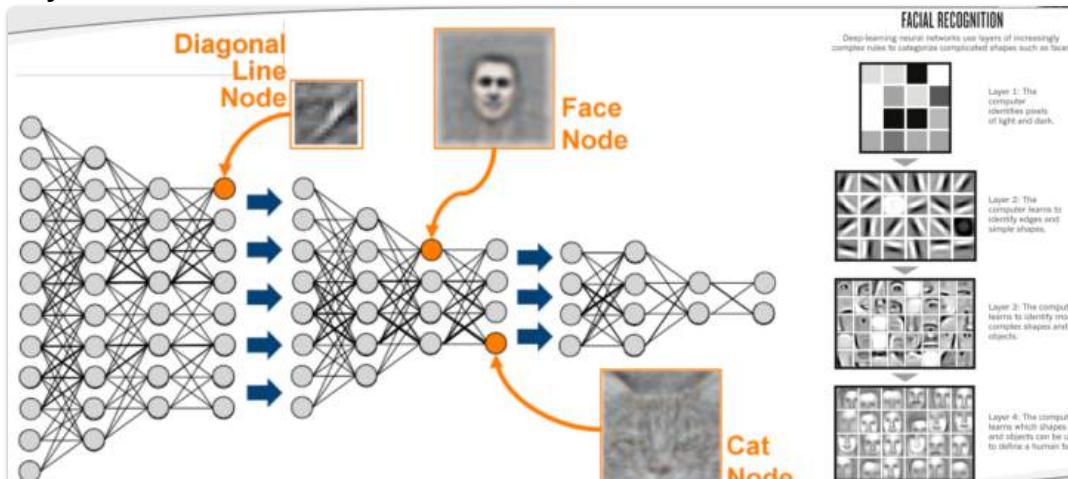
1. **Input:** Rohdaten (z.B. ein Auto-Bild).
2. **Feature extraction + Classification:** Die Merkmalsextraktion und die Klassifikation erfolgen gemeinsam und werden end-to-end gelernt in einem tiefen neuronalen Netzwerk.
3. **Output:** Klassifikation (z.B. "Car" oder "Not Car").



**Zusammenfassend:** Der Hauptunterschied besteht darin, dass traditionelles Machine Learning auf handgefertigten Merkmalen basiert, während Deep Learning die relevanten Merkmale direkt aus den Rohdaten lernt, was die Notwendigkeit menschlicher Expertise im Feature Engineering reduziert und oft zu besseren Ergebnissen bei komplexen Aufgaben führt.

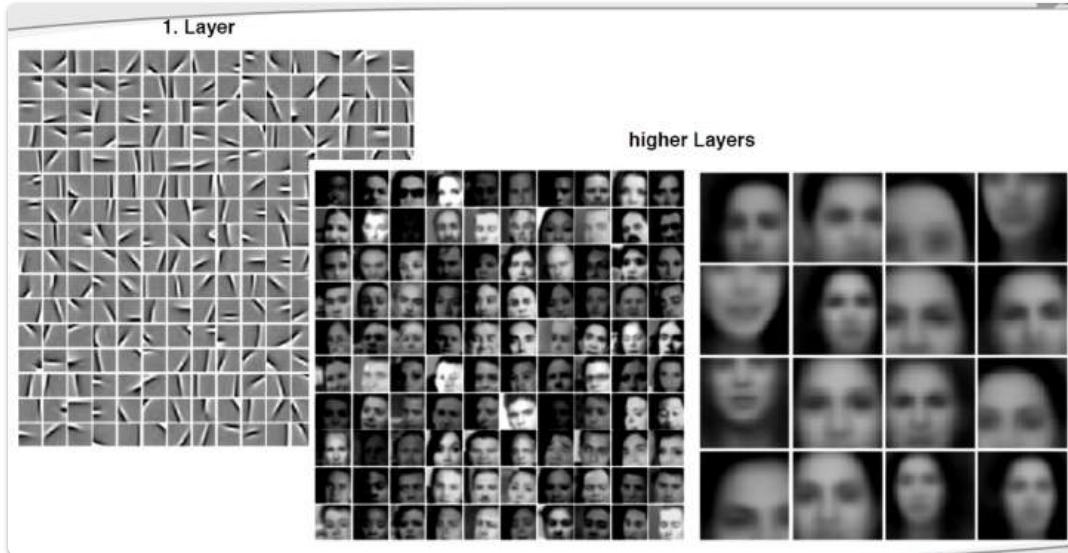
## CNN Features

- **Hierarchische Merkmalsrepräsentation:** Convolutional Neural Networks (CNNs) lernen hierarchische Merkmale.
- **Beispiel (Gesichtserkennung):**
  - **Diagonale Kanten (frühe Schichten):** Neuronen in frühen Schichten können auf einfache Merkmale wie diagonale Kanten reagieren.
  - **Gesichtsteile (mittlere Schichten):** Spätere Schichten kombinieren diese einfachen Merkmale, um komplexere Muster wie Augen, Nasen oder Münden zu erkennen.
  - **Gesicht (späte Schichten):** Noch höhere Schichten können das gesamte Gesicht als abstraktes Merkmal repräsentieren.
  - **Katze (anderes Beispiel):** Ähnliche hierarchische Merkmalsentwicklung für andere Objekte.



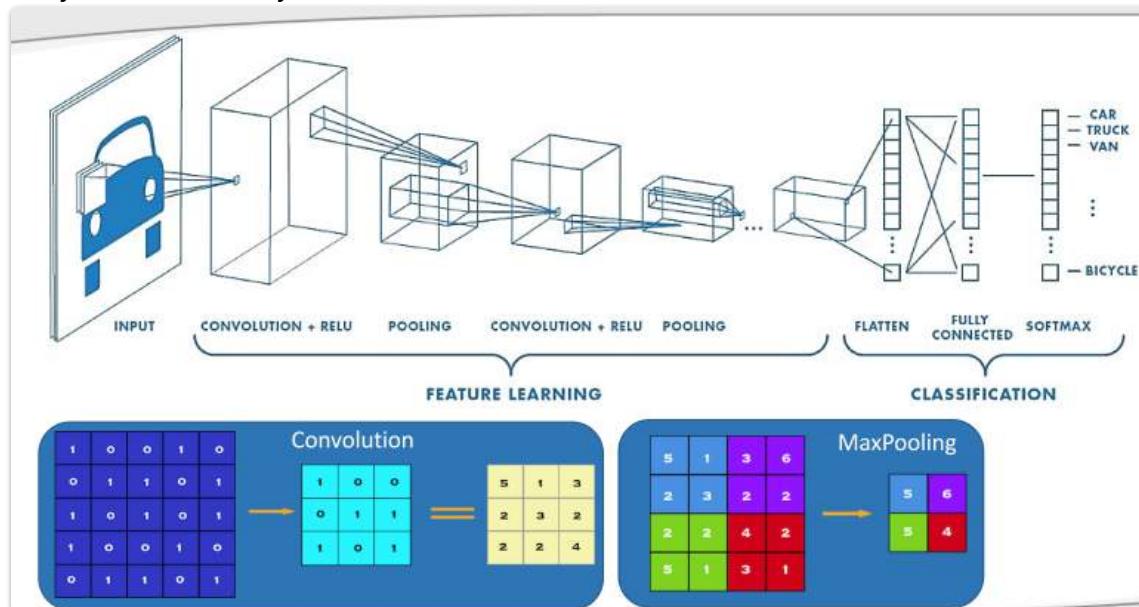
## CNN Visualization

- **Visualisierung gelernter Filter:** Die Abbildung zeigt beispielhafte Filter, die von CNNs in verschiedenen Schichten gelernt wurden.
  - **1. Layer:** Lernt oft einfache, lokale Muster wie Kanten und Orientierungen.
  - **Höhere Layer:** Lernen zunehmend komplexere und globalere Muster, die spezifische Objekte oder Teile davon repräsentieren können (z.B. Gesichtszüge).



## Inception Topologies

- **Beispiel einer komplexeren CNN-Architektur:** Die Inception-Architektur (hier vereinfacht dargestellt) verwendet verschiedene Filtergrößen und Operationen parallel, um Merkmale unterschiedlicher Skalen zu erfassen.
- **Feature Learning Pfad:** Zeigt typische Operationen wie Convolution und Pooling zur Merkmalsgewinnung.
- **Classification Pfad:** Führt die gelernten Merkmale zu einer Klassifikationsschicht (z.B. Fully Connected Layer mit Softmax).



## Deep Learning - Why does it work?

- **Can cope with vast amounts of data:** Tiefe Netze können von großen Datenmengen profitieren, um komplexe Muster zu lernen.
- **Learns small invariances:** Sie lernen, invariant gegenüber kleinen Variationen in den Eingabedaten zu sein (z.B. leichte Verschiebungen, Skalierungen, Rotationen).
- **Over-complete, sparse representations:** Können redundante und gleichzeitig spezialisierte Repräsentationen lernen.
- **Learn embedding:** Lernen, Eingabedaten in einen semantisch sinnvollen, niedrigerdimensionalen Raum einzubetten.
- **Lots of data:** Die Verfügbarkeit großer Trainingsdatensätze ist entscheidend für den Erfolg von Deep Learning.
- **Recent advance: it is actually computable!:** Fortschritte in der Hardware (z.B. GPUs) und in den Algorithmen haben das Training tiefer Netze in praktikabler Zeit ermöglicht.

## Weitere vo-slides zu ImageNet

---

### CNN Success Story: ILSVRC

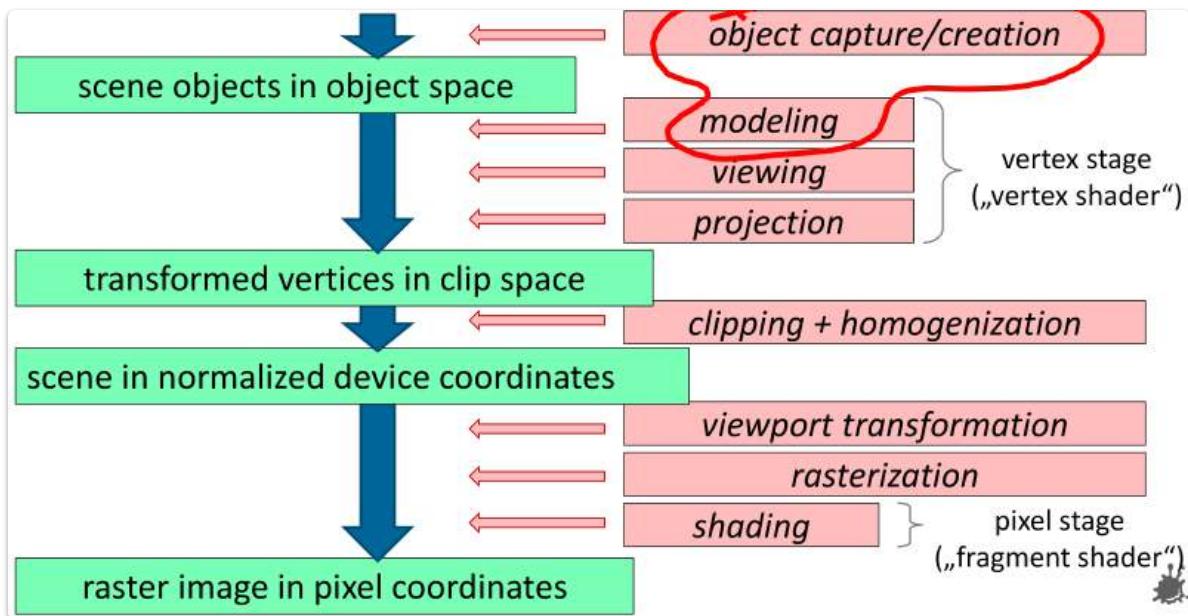
- **ImageNet database:**
  - 14 million labeled images.
  - 20,000 categories.

### ILSVRC: Classification

- **Computer Vision: International Large-Scale Visual Recognition Challenge (ILSVRC).**
- **Ziel:** Bildklassifizierung auf sehr großem Maßstab.
- **Beispielbilder aus dem ILSVRC Datensatz.**



# 12. Kurven und Flächen

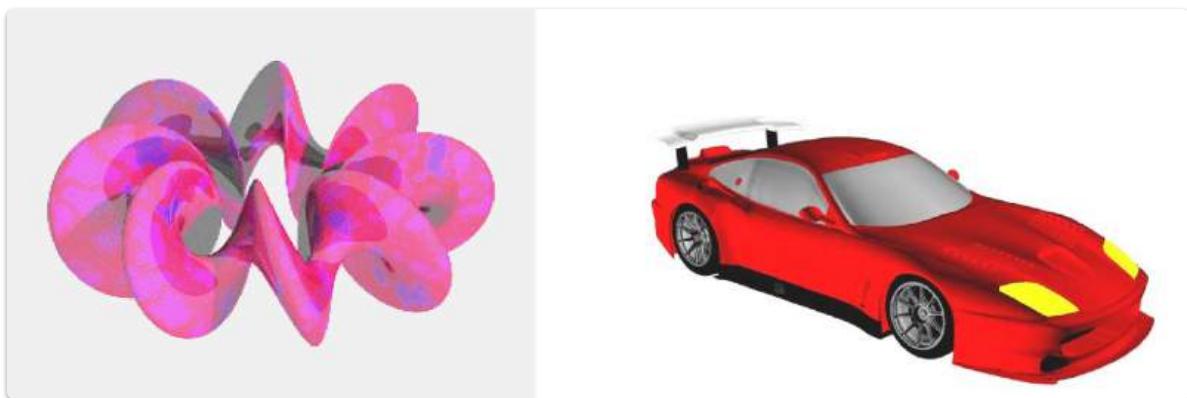


## Freiformflächen (Modellierung und Darstellung)

[EVC\\_Skriptum\\_CG, p.51](#)

Viele Anwendungen in der Computergrafik benötigen nicht nur elementare geometrische Formen, sondern auch die Möglichkeit, **beliebige allgemeine Flächen (Freiformflächen)** zu modellieren und darzustellen.

Die Prinzipien dafür werden oft zuerst anhand von Freiformkurven erläutert, da die zugrunde liegende Mathematik hier etwas einfacher ist. Die daraus entwickelten Verfahren lassen sich jedoch leicht auf Flächen erweitern.



## Quadratische Flächen

[EVC\\_Skriptum\\_CG, p.51](#)

Häufig verwendete Flächen können durch mathematische Formeln definiert werden. Man spricht hier von **analytischer Darstellung**. Es gibt drei Hauptarten der Definition:

## 1. Implizite Repräsentation:

- Bei dieser Darstellung liegen alle Punkte  $(x, y, z)$ , die eine bestimmte Formel erfüllen, auf der Oberfläche der Fläche.
- Beispiel Kugel:**  $x^2 + y^2 + z^2 = r^2$ .
- Intuition:** Man kann sich das vorstellen wie eine "Filterfunktion". Wenn man einen Punkt  $(x, y, z)$  in die Formel einsetzt und die Gleichung erfüllt ist, dann liegt der Punkt auf der Oberfläche. Wenn nicht, dann nicht.

## 2. Explizite Repräsentation:

- Eine Darstellung ist explizit, wenn ein Koordinatenwert (typischerweise  $z$ ) von den anderen Koordinaten  $(x, y)$  abhängig dargestellt wird.
- Beispiel Kugel:**  $z = \sqrt{r^2 - x^2 - y^2}$ .
- Intuition:** Man kann sich das als eine "Funktion" vorstellen: Für jedes gegebene  $x$  und  $y$  berechnet man direkt das zugehörige  $z$  auf der Oberfläche. Dies funktioniert aber nur für Oberflächen, die für jedes  $(x, y)$  genau einen (oder eine begrenzte Anzahl von)  $z$ -Wert(e) haben.

## 3. Parametrische Repräsentation:

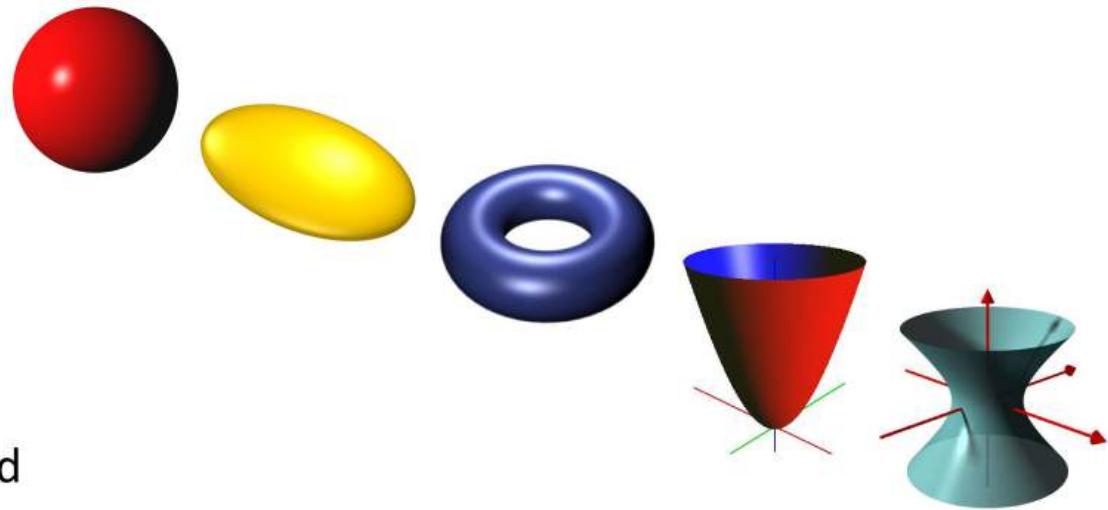
- Bei dieser Darstellung wird für jede Kombination von **Parameterwerten** (z.B.  $u, v$  oder  $\phi, \theta$ ) ein Punkt auf der Oberfläche erzeugt. Die Koordinaten  $(x, y, z)$  sind Funktionen dieser Parameter.
- Beispiel Kugel:**
  - $x = r \cdot \cos \phi \cdot \cos \theta$
  - $y = r \cdot \cos \phi \cdot \sin \theta$
  - $z = r \cdot \sin \phi$
  - mit Parameterbereichen:  $-\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2}$  und  $-\pi \leq \theta \leq \pi$ .
- Intuition:** Hier hat man "Stellschrauben"  $(\phi, \theta)$ , die man verstellt. Jede Einstellung dieser Schrauben liefert einen Punkt auf der Oberfläche. Man "zeichnet" die Oberfläche, indem man diese Parameter über ihren Bereich variiert. Diese Methode ist sehr flexibel, da sie auch komplexe Formen darstellen kann, die bei expliziter Darstellung schwierig wären (z.B. eine geschlossene Kugel, bei der  $z$  nicht eindeutig von  $x, y$  abhängt).

### • Weitere häufig verwendete quadratische Flächen:

**Ellipsoid:** Eine gestreckte oder gestauchte Kugelform.

**Torus:** Eine Donut-Form.

\* **Quadratics:** Dies ist ein allgemeiner Begriff für Flächen, die durch eine quadratische Gleichung (zweiten Grades) in  $x, y, z$  definiert sind. Dazu gehören Kugeln, Ellipsoide, Paraboloiden, Hyperboloiden usw.



$y = f(x)$ <i>axis dependent</i>	$x = f(u)$ $y = g(u)$ <i>axis independent</i>
<i>example:</i> $y = \sqrt{1-x^2}$ 	$x=\cos(u)$ $y=\sin(u)$ 

## Mehr Formen

Hier nochmal mehr Informationen zu den Formen aus den Slides

**Quadric Surfaces: Sphere**

- *implicit:*  $x^2 + y^2 + z^2 = r^2$
- *parametric:*  $x = r \cos \phi \cos \theta, \quad -\pi/2 \leq \phi \leq \pi/2$   
 $y = r \cos \phi \sin \theta, \quad -\pi \leq \theta \leq \pi$   
 $z = r \sin \phi$

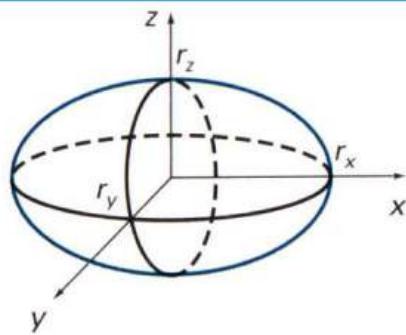
*parametric coordinate position*  
 $(r, \theta, \phi)$  *on the surface of a*  
*sphere with radius r*

Werner Purgathofer

## Quadric Surfaces: Ellipsoid

■ *implicit:*

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$



■ *parametric:*

$$x = r_x \cos \phi \cos \theta, \quad -\pi/2 \leq \phi \leq \pi/2$$

$$y = r_y \cos \phi \sin \theta, \quad -\pi \leq \theta \leq \pi$$

$$z = r_z \sin \phi$$

## Quadric Surfaces: Torus

■ *implicit:*

$$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$$

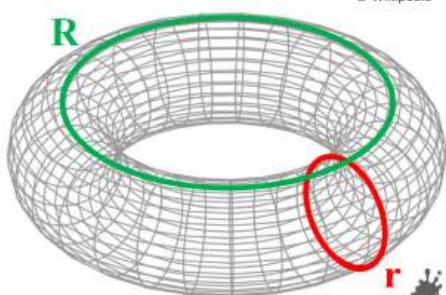


■ *parametric:*

$$x = (R + r \cos \phi) \cos \theta, \quad -\pi \leq \phi \leq \pi$$

$$y = (R + r \cos \phi) \sin \theta, \quad -\pi \leq \theta \leq \pi$$

$$z = r \sin \phi$$



Werner Purgathofer

9

## Free Form Surfaces

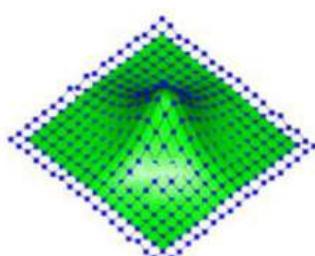
can be represented by

**huge number of points (or polygons)**

- + arbitrary shapes possible
- large memory requirements
- changes cause much work
- corners after scaling!
- modeling?

**mathematical functions**

- only for some shape categories
- + marginal memory requirements
- + changes are rather simple
- + definition arbitrarily exact
- modeling!



$$x = f(u,v)$$

$$y = g(u,v)$$

$$z = h(u,v)$$

# Kurven

[EVC\\_Skriptum\\_CG, p.51](#)

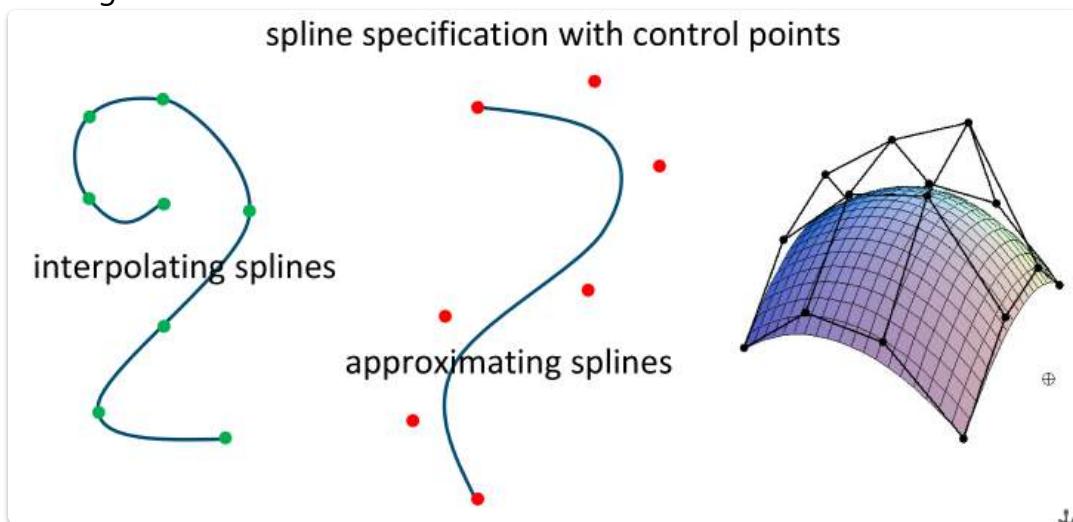
Kurven in der Computergrafik können auf zwei Arten definiert werden:

1. **Analytisch (Formelbasiert):** Die Kurve wird mathematisch durch eine Formel beschrieben.  
Diese Methode ist oft weniger intuitiv für den Benutzer.
2. **Punktbasiert (Stütz-/Kontrollpunkte):** Die Kurve wird durch eine Reihe von Punkten geformt, die ihren Verlauf bestimmen. Dies ist die gängigere und benutzerfreundlichere Methode.

## Unterscheidende Merkmale von Kurventypen:

Verschiedene Kurvenmodelle können anhand ihrer Eigenschaften klassifiziert werden:

- **Interpolation vs. Approximation:**
  - **Interpolierend:** Die Kurve geht direkt durch alle vorgegebenen Stützpunkte.
  - **Approximierend:** Die Kurve nähert sich den Kontrollpunkten an, berührt sie aber in der Regel nicht direkt.



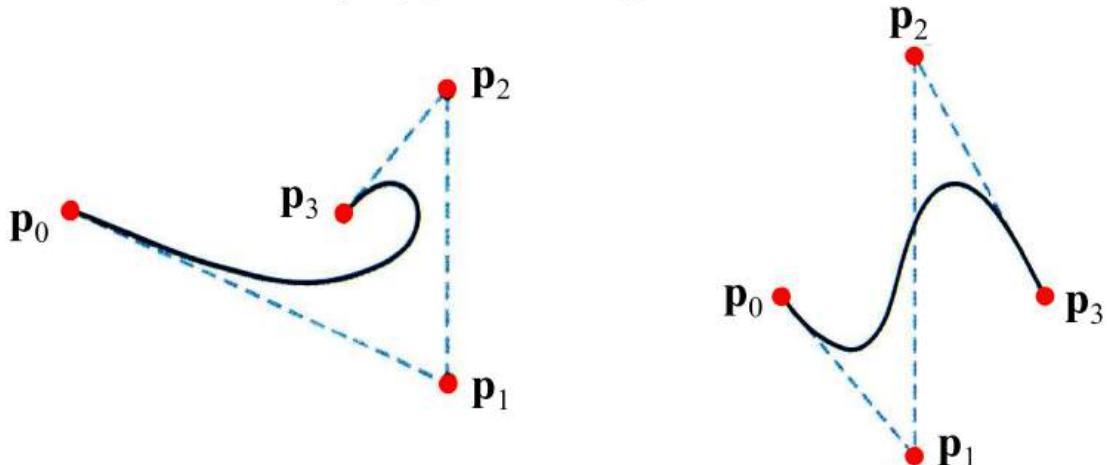
- **Stetigkeit an Verbindungsstellen:** Beschreibt die "Glätte" der Übergänge zwischen einzelnen Kurvensegmenten (z.B. keine Knicke oder abrupte Krümmungswechsel).
- **Einflussbereich von Punkten:**
  - **Globaler Einfluss:** Eine Änderung an einem Punkt wirkt sich auf die gesamte Kurve aus.
  - **Lokaler Einfluss:** Eine Änderung beeinflusst nur einen begrenzten Abschnitt der Kurve.
- **Abhängigkeit vom Koordinatensystem:**
  - **Achsenabhängig:** Die Kurvenform ändert sich, wenn das Koordinatensystem gedreht wird.
  - **Achsenunabhängig:** Die Kurvenform bleibt bei Rotation des Koordinatensystems erhalten.

- **Verhalten bei Richtungswechseln:** Tendenz zu "Dämpfung" (sanfte Übergänge) oder "Überschwingen" (die Kurve schwingt über den Kontrollpunkt hinaus, bevor sie sich annähert).
- **Morphologische Eigenschaften:** Dies umfasst Aspekte wie die möglichen Formen, die eine Kurve annehmen kann, ob sie doppelte Punkte (Schleifen) aufweisen kann oder ob sie sich zu einer geschlossenen Form (z.B. ein Kreis) schließen lässt.

Kurven, die mittels Stütz- oder Kontrollpunkten definiert werden, bezeichnet man allgemein als **Splines**. Im Folgenden werden gängige Spline-Verfahren vorgestellt.

(also called „Characteristic Polygon“)

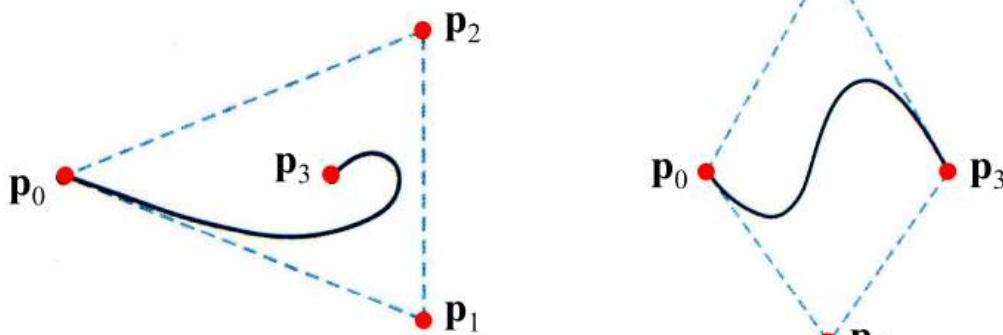
= polygon defining the curve



operations on splines:

- move, insert control points
- spline transformation by transforming all control points

convex hull property

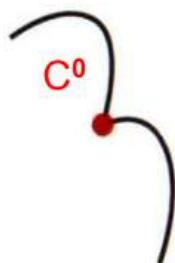


**parametric continuity conditions ( $C^n$ )**

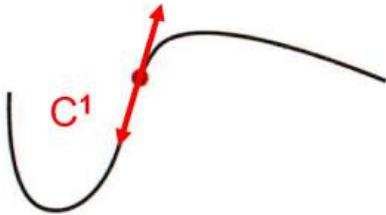
$n$  = number of derivations at section joints that are equal

$$x = x(u) \quad y = y(u) \quad z = z(u) \quad u_{\min} < u < u_{\max}$$

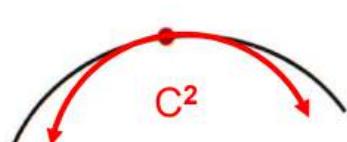
$C^0$  continuity



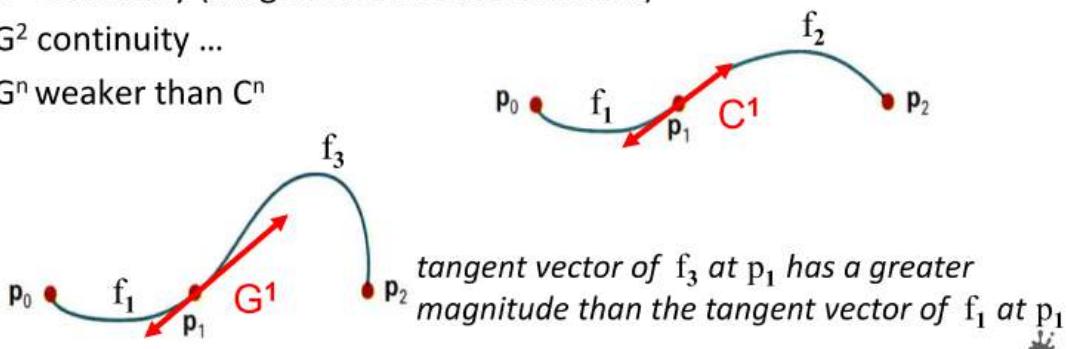
$C^1$  continuity



$C^2$  continuity

**geometric continuity conditions ( $G^n$ )**

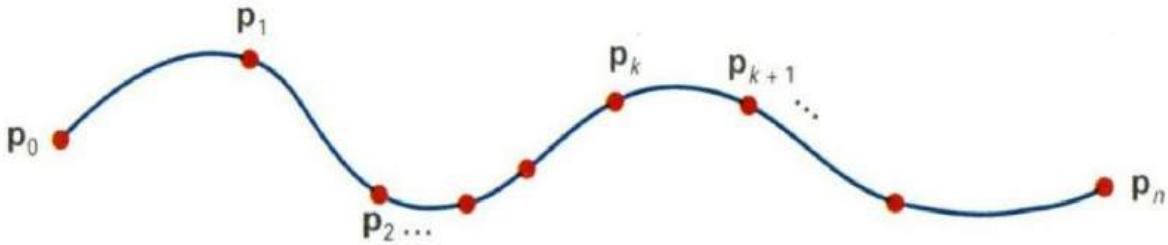
- derivations at joints have different magnitudes
- $G^0 (=C^0)$  continuity
- $G^1$  continuity (tangent vectors are collinear)
- $G^2$  continuity ...
- $G^n$  weaker than  $C^n$



## Kubische Spline-Interpolation

[EVC\\_Skriptum\\_CG, p.52](#)

- Gegeben sind  $n + 1$  Stützpunkte  $p_i = (x_i, y_i(\dots), i)$ , für  $i = 0, \dots, n$ .
- Eine Interpolationskurve, die zwischen je 2 Stützpunkten aus einem kubischen Polynom besteht, nennt man Kubischen Spline.
- Zwischen Stützpunkten  $p_k$  und  $p_{k+1}$  wird die Kurve durch einen Parameter  $u$  beschrieben:
  - $p_k(u) = a_k u^3 + b_k u^2 + c_k u + d_k$
  - Mit  $k = 0, 1, 2, \dots, n - 1$  und  $0 \leq u \leq 1$ .
  - **Achtung:**  $a_k, b_k, c_k, d_k$  sind dabei Vektoren.



- **Berechnung eines Kurvenstücks:**

- Um ein Kurvenstück zwischen 2 Stützpunkten zu berechnen, benötigt man 4 Angabestücke.
- Die Definition an den Stützpunkten ist so gewählt, dass die kubischen Polynome sowohl  $C^1$ -stetig (differenzierbar) als auch  $C^2$ -stetig (zweifach differenzierbar, also gleiche Krümmung) verbunden sind.
- Man spricht dann von *natürlichen kubischen Splines*.
- Diese erhält man, indem man ein Gleichungssystem mit  $4n$  Variablen löst.
- Dabei müssen an den Rändern 2 Nebenbedingungen vorgegeben werden, z.B. Krümmung am Anfang und am Ende ist null.
- **Nachteil kubischer Splines:** Jeder Stützpunkt hat einen Einfluss auf den gesamten Kurvenverlauf (globaler Einfluss).

## Hermite-Interpolation

[EVC\\_Skriptum\\_CG, p.52](#)

- Die Hermite-Interpolation ist eine spezielle Form der kubischen Splines.
- **Besonderheit:** Neben den Stützpunkten  $p_k$  werden auch die *Ableitungen*  $Dp_k$  an den Stützpunkten vorgegeben.
- Das kubische Interpolationspolynom  $p_k(u)$ ,  $0 \leq u \leq 1$ , zwischen den Punkten  $p_k$  und  $p_{k+1}$  lässt sich aus den 4 Bestimmungsstücken eindeutig berechnen:
  - $p_k(0) = p_k$
  - $p_k(1) = p_{k+1}$
  - $p'_k(0) = Dp_k$  (Ableitung am Startpunkt  $p_k$ )
  - $p'_k(1) = Dp_{k+1}$  (Ableitung am Endpunkt  $p_{k+1}$ )
  - Dies gilt für  $k = 0, \dots, n - 1$ .
- Das Polynom  $p_k(u) = a_k u^3 + b_k u^2 + c_k u + d_k$  lässt sich auch in Matrixschreibweise anschreiben.
- Die erste Ableitung dieser Kurve ist  $p'_k(u) = 3a_k u^2 + 2b_k u + c_k$ .
- Daraus kann man die 4 Bestimmungsstücke  $p_k, p_{k+1}, Dp_k, Dp_{k+1}$  formulieren.

$$\mathbf{p}_k(u) = [ u^3 \quad u^2 \quad u \quad 1 ] \cdot \begin{bmatrix} \mathbf{a}_k \\ \mathbf{b}_k \\ \mathbf{c}_k \\ \mathbf{d}_k \end{bmatrix} \quad \mathbf{p}'_k(u) = [ 3u^2 \quad 2u \quad 1 \quad 0 ] \cdot \begin{bmatrix} \mathbf{a}_k \\ \mathbf{b}_k \\ \mathbf{c}_k \\ \mathbf{d}_k \end{bmatrix}$$

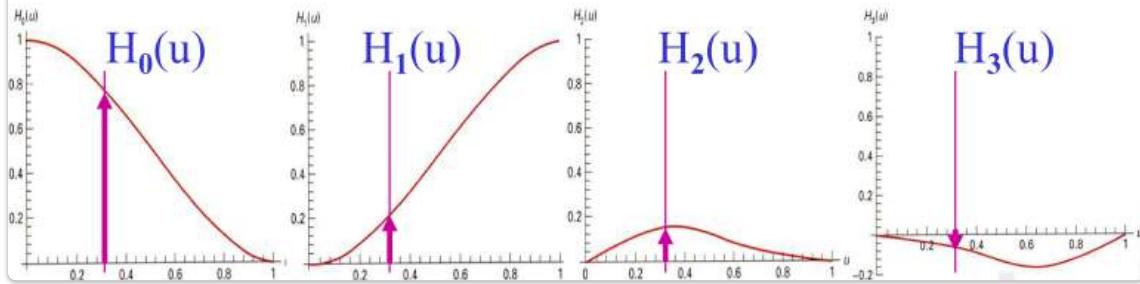
$$\begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{D}\mathbf{p}_k \\ \mathbf{D}\mathbf{p}_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a}_k \\ \mathbf{b}_k \\ \mathbf{c}_k \\ \mathbf{d}_k \end{bmatrix}$$

Um die Koeffizientenvektoren  $\mathbf{a}_k, \mathbf{b}_k, \mathbf{c}_k, \mathbf{d}_k$  von  $\mathbf{a}_k u^3 + \mathbf{b}_k u^2 + \mathbf{c}_k u + \mathbf{d}_k$  zu berechnen, invertiert man diese Matrix. Die resultierende Matrix heißt Hermite-Matrix  $\mathbf{M}_H$ :

$$\begin{bmatrix} \mathbf{a}_k \\ \mathbf{b}_k \\ \mathbf{c}_k \\ \mathbf{d}_k \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{D}\mathbf{p}_k \\ \mathbf{D}\mathbf{p}_{k+1} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{D}\mathbf{p}_k \\ \mathbf{D}\mathbf{p}_{k+1} \end{bmatrix}$$

$$\mathbf{p}_k(u) = [ u^3 \quad u^2 \quad u \quad 1 ] \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{D}\mathbf{p}_k \\ \mathbf{D}\mathbf{p}_{k+1} \end{bmatrix}$$

### $H_k(u)$ blending functions:

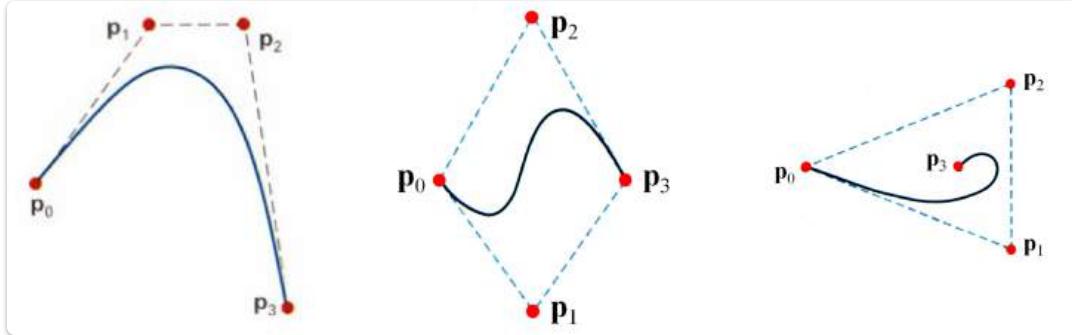


## Bézier-Kurven

[EVC\\_Skriptum\\_CG, p.52, p.53](#)

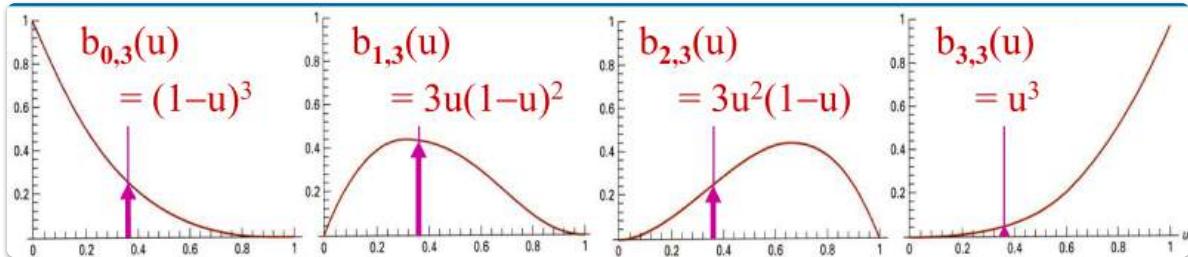
- Von Pierre Bézier um 1960 für die Beschreibung von Autokarosserien bei Renault beschrieben.
- Eine *approximierende* Kurve, die sogenannte *Bernstein-Polynome*  $b_{k,n}$  als Gewichtsfunktionen für die Kontrollpunkte verwendet.
- Jeder Kurvenpunkt ist das gewichtete Mittel aller Kontrollpunkte:
  - $p(u) = \sum_{k=0}^n p_k b_{k,n}(u)$ , mit  $0 \leq u \leq 1$
  - Wobei  $b_{k,n}(u) = \binom{n}{k} u^k (1-u)^{n-k}$  die Bernstein-Polynome sind.
- Die Gewichtsfunktionen  $b_{k,n}(u)$  sind über einen Parameterbereich  $u$  im Bereich von 0 bis 1 definiert.
- Sie hängen von zwei Werten ab:
  - $n$ : Anzahl der Stützpunkte der Kurve (genau genommen gibt es  $n+1$  Stützpunkte).
  - $k$ : gibt an, für welchen Stützpunkt die Gewichtsfunktion verwendet wird.
- Jeder dieser Werte (außer an den Rändern  $u=0$  und  $u=1$ ) ist größer als Null.
- Da die Summe der  $b_{k,n}(u)$  über alle  $k$  überall 1 ist, ist jeder Punkt der Kurve  $p(u)$  somit ein gewichteter Mittelwert aller Stützpunkte.
- **Beispiel für 4 Kontrollpunkte (also  $n = 3$ ):**

- $p(u) = (1-u)^3 \cdot p_0 + 3u(1-u)^2 \cdot p_1 + 3u^2(1-u) \cdot p_2 + u^3 \cdot p_3$



## Eigenschaften von Bézier-Kurven:

- Bei  $n+1$  Kontrollpunkten ist  $p(u)$  vom Grad  $n$ .
- Jeder Kontrollpunkt "zieht" die Kurve wie mit einem Gummiband an.
- Globaler Einfluss:** Gewichtsfunktion fast überall  $> 0$ . (Eine Änderung an einem Kontrollpunkt beeinflusst die gesamte Kurve.)
- $p_0$  und  $p_n$  (Start- und Endpunkt) liegen auf der Kurve.
- Die Tangenten in  $p_0$  und  $p_n$  sind die Verbindung zu den nächsten Punkten  $p_1$  und  $p_{n-1}$ .
- Die Kurve liegt zur Gänze in der *konvexen Hülle* der Kontrollpunkte (die konvexe Hülle ist das kleinste konvexe Polygon, das alle Kontrollpunkte enthält).



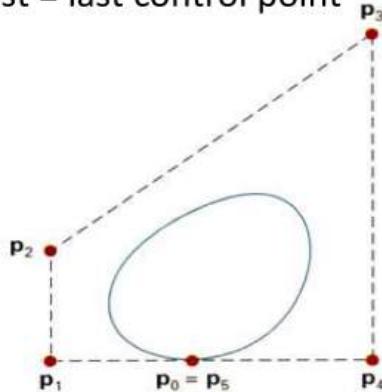
- Einige dieser Eigenschaften erkennt man aus der Form der Bernstein-Polynome  $b_{k,n}$ .

## Weiteres aus den Slides Slides:

## Bézier Curves Design Techniques (1)

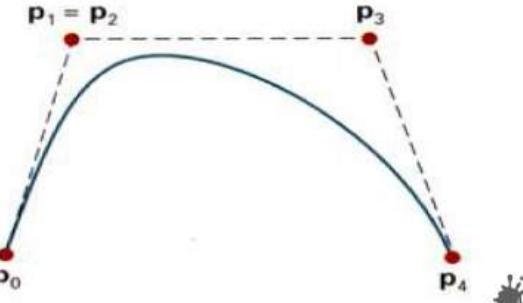
a ***closed Bézier curve***

generated by setting:  
first = last control point



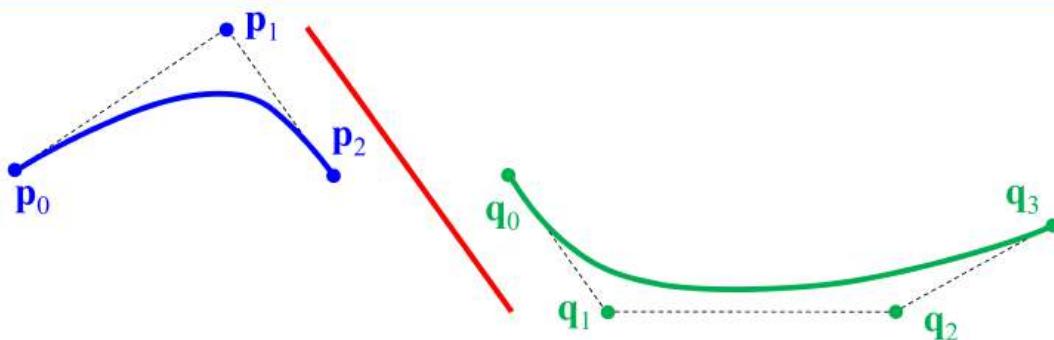
Werner Purgathofer

a Bézier curve can be made to pass closer to a given coordinate position by assigning ***multiple control points*** to that position



30

## Bézier Curves Design Techniques (2)



***piecewise approximation curve*** formed with 2 Bézier sections.  
0-order and 1<sup>st</sup>-order continuity ( $C^0$ ,  $C^1$  or  $G^0$ ,  $G^1$ ) are attained by setting  $q_0 = p_2$  and by making  $p_1$ ,  $p_2$ , and  $q_1$  collinear.

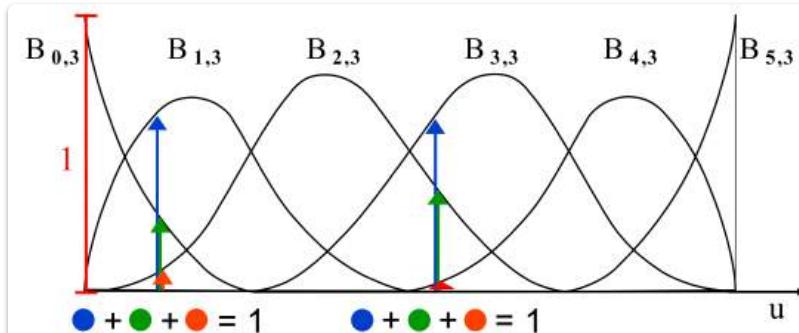
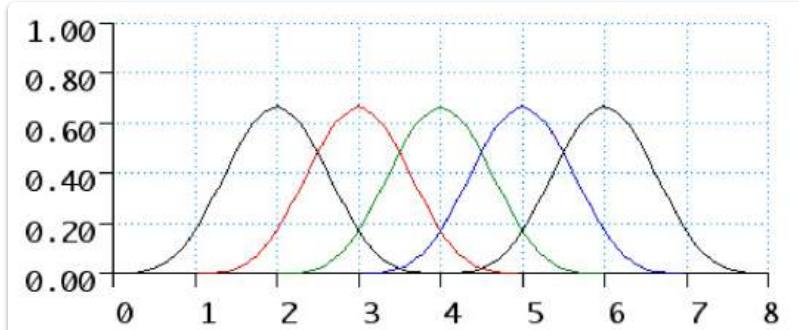
## Cubic Bézier Curve Matrix Notation

$$\mathbf{p}(u) = (1-u)^3 \cdot \mathbf{p}_0 + 3u(1-u)^2 \cdot \mathbf{p}_1 + 3u^2(1-u) \cdot \mathbf{p}_2 + u^3 \cdot \mathbf{p}_3$$

$$\mathbf{p}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot M_{\text{Bez}} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad \text{with} \quad M_{\text{Bez}} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

## B-Spline-Kurven

- Der Hauptnachteil der Bézierkurven ist der globale Einfluss der Kontrollpunkte auf die ganze Kurve. Dies hat zwei Hauptnachteile:
  - Jede Veränderung der Kontrollpunkte (Einfügen, Entfernen, Verschieben) verändert das Aussehen der Kurve an allen Stellen.
  - Die Rechenzeit für große Kontrollpunktmenge ist höher.
- Die Ursache liegt in der Form der Gewichtsfunktionen.
- Die sogenannten **B-Splines** sind ebenso wie die Bézier-Splines approximierende Kurven.
- Jedoch sind die Bernstein-Polynome durch **B-Spline-Polynome  $B_{k,d}$**  ersetzt.
- Diese beschränken die Anzahl der Kontrollpunkte, die einen Kurvenpunkt beeinflussen, auf  $d$ .
- Die Berechnung der  $B_{k,d}$  ist etwas komplexer und erfolgt rekursiv.
- Für das Verständnis reicht es allerdings, die Form der B-Spline-Polynome zu sehen.**
- Man erkennt, dass jede Gewichtskurve nur in einem begrenzten Bereich ungleich null ist, so dass jeder Punkt über weite Bereiche keinen Einfluss auf die Kurve hat (im Gegensatz zu Bézier-Kurven).



- Eine wichtige Eigenschaft der B-Spline-Gewichtsfunktionen ist die Tatsache, dass (wie bei den Bernstein-Polynomen) für jeden Kurvenpunkt ihre Summe genau 1 ist:

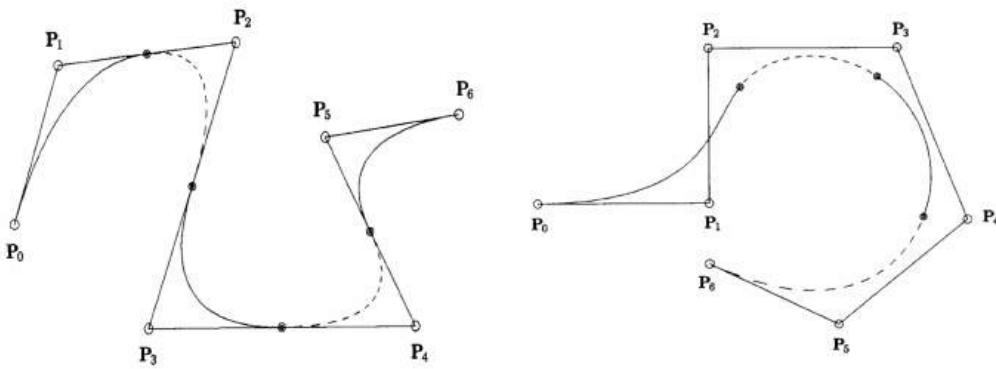
$$\sum_{k=0}^n B_{k,d}(u) = 1$$

- Jeder B-Spline-Kurvenpunkt ist somit ein **gewichteter Mittelwert** aus den Kontrollpunkten.

## Beispiele für B-Spline-Kurven

EVC\_Skriptum\_CG, p.54

- Beispiele für B-Spline-Kurven mit  $d = 3$  (links) und  $d = 4$  (rechts):**



- Wenn man  $d = n + 1$  wählt, erhält man Bézier-Kurven. Diese sind also ein Spezialfall der B-Splines.

### Influence of d



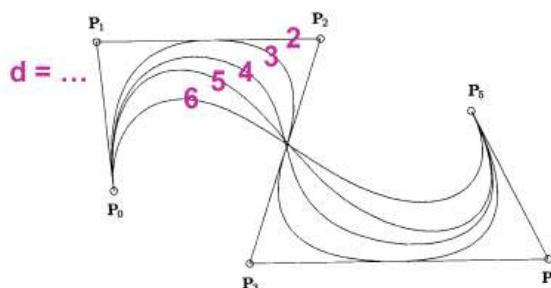
$d$  describes, how many control points influence any point on the curve

$d = 2$  linear

$d = 3$  quadratic

$d = 4$  cubic

...



for  $d=n+1$  you get Bézier curves!

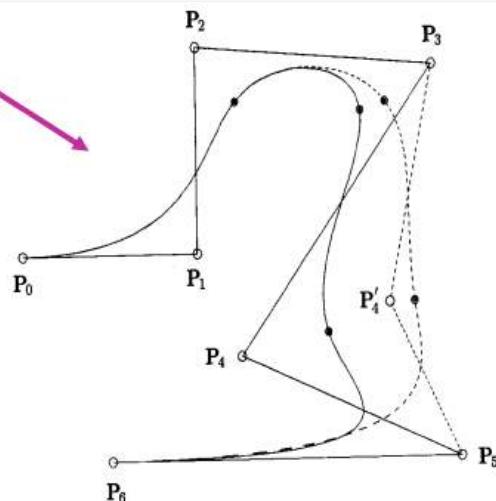
### Hauptunterschiede zu Bézierkurven:

EVC\_Skriptum\_CG, p.54

- **Lokaler Einfluss der Kontrollpunkte** (Änderungen wirken sich nur auf einen begrenzten Bereich der Kurve aus, nicht global).
- **Aufwand linear zur Anzahl der Kontrollpunkte** (statt quadratisch bei Bézierkurven, was die Berechnung bei vielen Kontrollpunkten effizienter macht).

control points have local influence

effort is linearly dependent on  $n$ ,  
therefore splitting of huge point  
sets not necessary

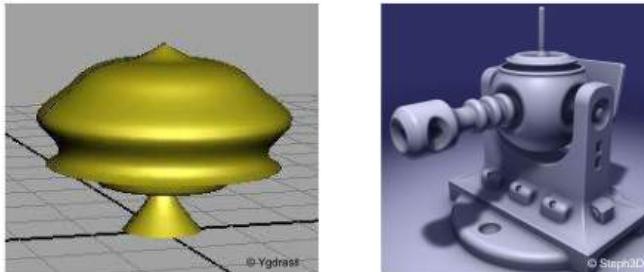


### NURBS (Non-Uniform Rational B-Splines)

EVC\_Skriptum\_CG, p.54

- Die wichtigsten Erweiterungen dieser sogenannten *uniformen* B-Splines führen zu den **Non-Uniform Rational B-Splines**, besser bekannt als **NURBS**.
- Mit NURBS können auch *regelmäßige geometrische Formen* konsistent repräsentiert werden.
- Wichtiger Hinweis:** Alle beschriebenen Methoden (Splines) gelten gleichermaßen für Punkte im zweidimensionalen und im dreidimensionalen Raum.
- Grundsätzlich beschreiben also alle diese Splines **räumliche Kurven im dreidimensionalen Raum**.

further extension: **Non-Uniform Rational B-Splines = "NURBS"**  
allow to combine freeform surfaces with regular surfaces



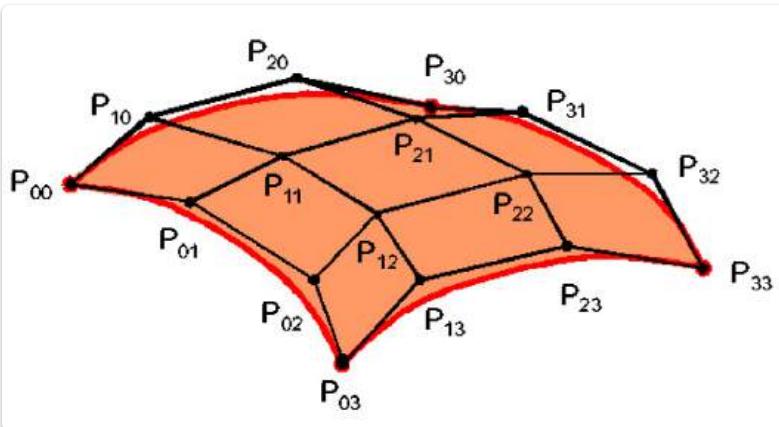
## Freiformflächen -- Bézier- und B-Spline-Flächen

[EVC\\_Skriptum\\_CG, p.54](#)

- Bildet man über einem zweidimensionalen Punkteraster das kartesische Produkt zweier Kurvenscharen, so erhält man auf natürliche Weise **Freiformflächen**.
- Je nach zugrundeliegender Kurvenart erhält man verschiedene Flächen, z.B. Bézierflächen aus Bézierkurven, B-Splineflächen aus B-Splinekurven usw.
- Formel für Freiformflächen:**

$$p(u, v) = \sum_{j=0}^m \sum_{k=0}^n P_{j,k} b_{j,m}(v) b_{k,n}(u)$$

- Diese Formel zeigt, dass ein Punkt auf der Fläche  $p(u, v)$  durch eine doppelte Summe über Kontrollpunkte  $P_{j,k}$  und die Produkte von zwei Bernstein-Polynomen (oder anderen Gewichtsfunktionen) bestimmt wird.

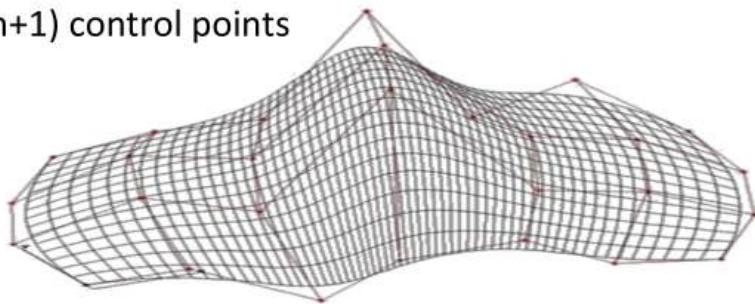


- Jedes Parameterpaar  $(u, v)$  führt zu einem Punkt der entstehenden Fläche.

- Die Randkurven der Fläche sind in diesem Beispiel (der Bézierfläche) wieder Bézierkurven.
- Auch die anderen Eigenschaften der Bézierkurven werden auf die Flächen übertragen.
- Analog zu diesen Bézierflächen erhält man B-Spline-Flächen, wenn man die B-Spline-Gewichtsfunktionen in die Formel einsetzt, oder etwa NURBS-Flächen für NURBS-Kurven.

$\mathbf{p}_{j,k}$  : grid of  $(m+1) \times (n+1)$  control points

just like for  
Bezier surfaces!



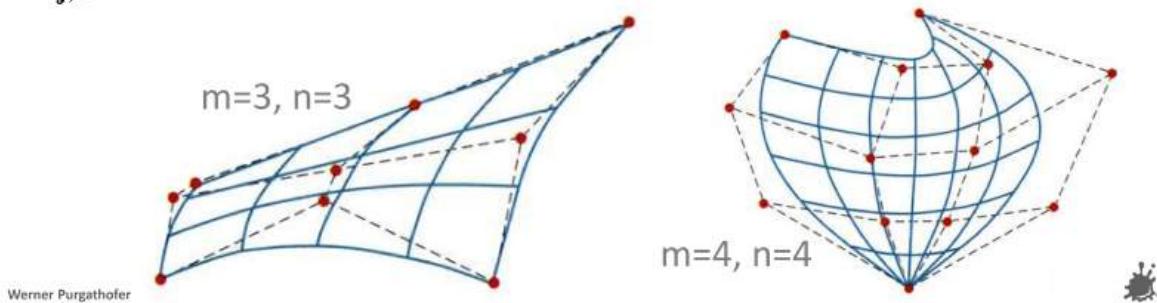
## Bézier Surfaces Definition



Cartesian product of two Bézier curve bundles

$$\mathbf{p}(u, v) = \sum_{j=0}^m \sum_{k=0}^n \mathbf{p}_{j,k} b_{j,m}(v) b_{k,n}(u)$$

$\mathbf{p}_{j,k}$  : grid of  $(m+1) \times (n+1)$  control points

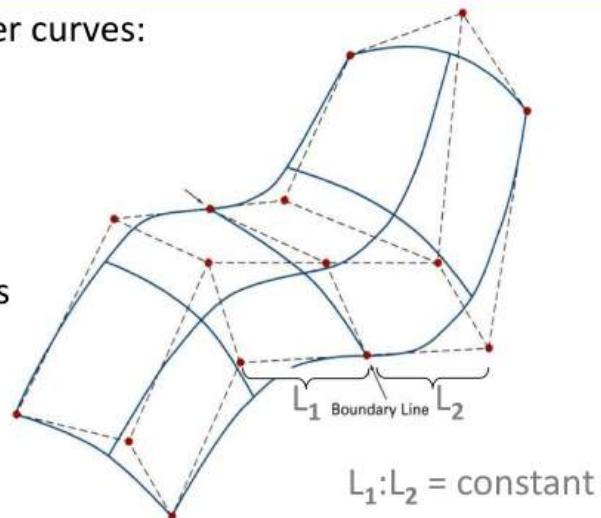


## Bézier Surfaces Properties



have the same properties as Bézier curves:

- global influence
- interpolates corner points
- tangents at corner points
- convex hull property
- 1<sup>st</sup>-order continuity connections



## Zeichnen von Freiformflächen:

- Um Freiformflächen zu zeichnen, können **Dreiecksnetze** aus den Flächen erzeugt werden, die wie B-Reps gerendert werden.
- Alternativ werden **Ray-Casting-Methoden** eingesetzt:
  - Dazu muss man eine Prozedur implementieren, die den Schnittpunkt einer Geraden mit der Fläche möglichst genau berechnet.
  - An dieser Stelle wird auch die Oberflächennormale zurückgeliefert (wichtig für Beleuchtungsberechnungen).

# 12. Computational Photography

[EVC\\_Skriptum\\_CV](#), p.60

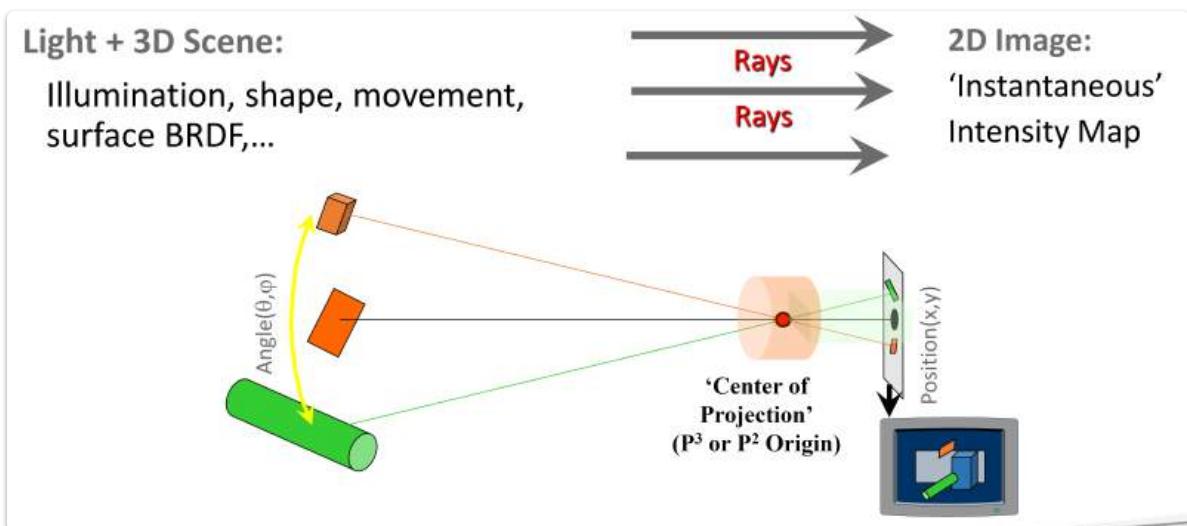
Computational Photography ist ein Forschungsgebiet, das die **Verbreitung von Digitalkameras** nutzt, um die alltägliche Fotografie zu verbessern.

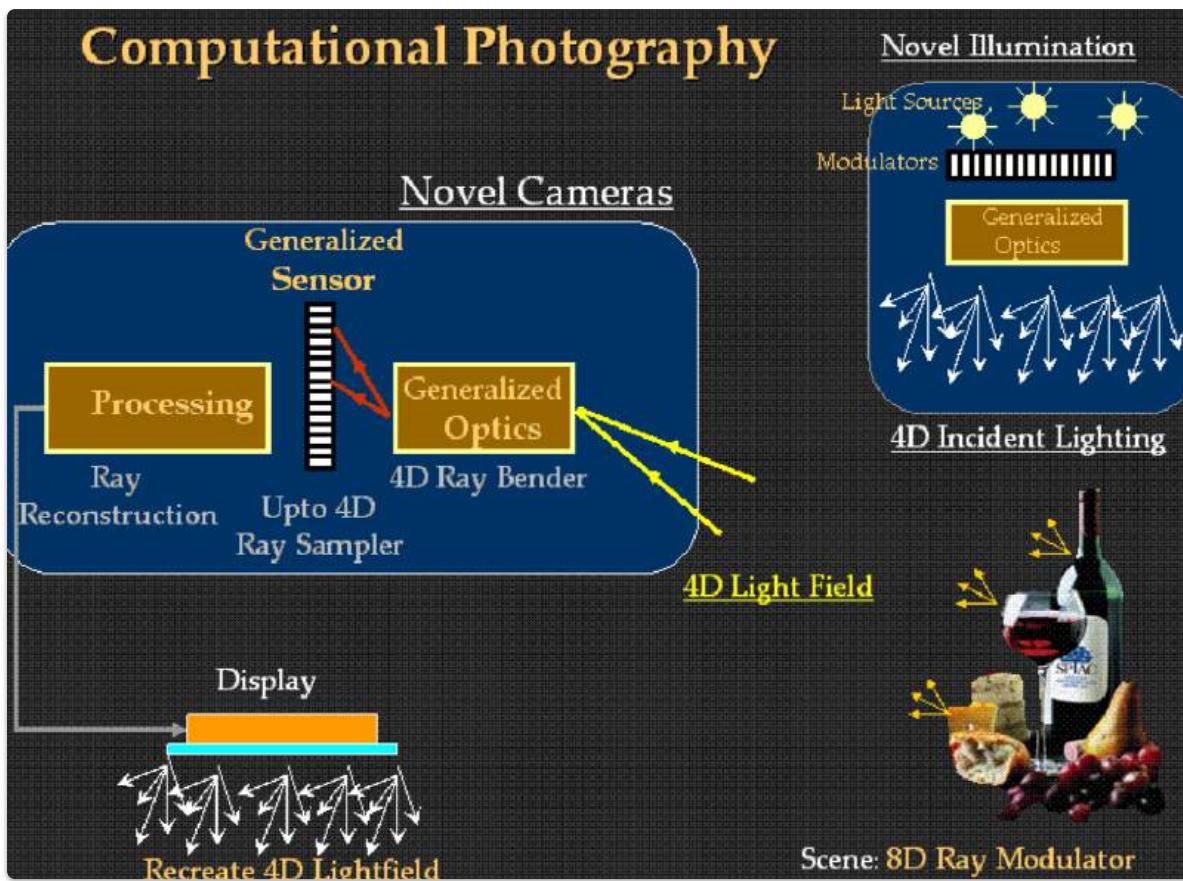
- **Ziele:**

- Neue Aufnahme- und Bearbeitungsmethoden entwickeln, die normalerweise fotografisch nicht möglich wären.
- Beispiele: Aufnahme bei extremem Kontrastumfang, verbesserte verwackelte Bilder, Komposition mehrerer Personen oder Objekte zu einem neuen Gesamtbild, Entfernen störender Personen oder Ähnliches.
- Übergeordnetes Ziel ist es, neue Funktionalitäten und Anwendungsgebiete für die Fotografie zu erschließen.

- **Grundlage:** Mittels algorithmischer ("Computational") Ansätze werden die physikalischen Grenzen traditioneller Projektionskameras überwunden.

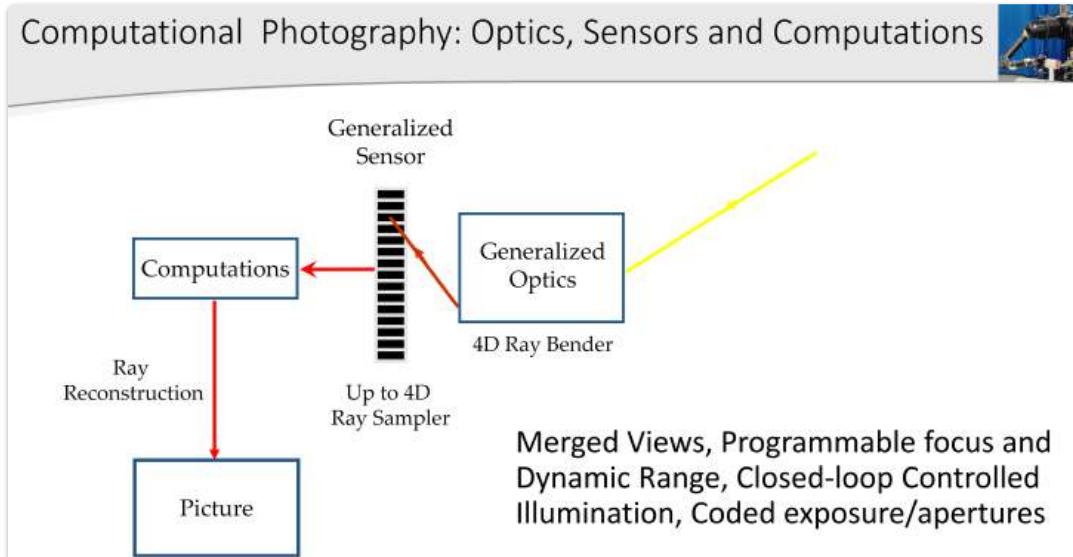
- Ermöglicht Bilderzeugung, die mit herkömmlichen Kameras nicht realisierbar wäre (z.B. große Tiefenschärfe, hohe Auflösung, geringer Verlust der dritten Dimension).
- Durch Kombination von Software, digitalen Sensoren, moderner Optik sowie intelligenter Beleuchtung verschiebt Computational Photography die Grenzen der traditionellen Fotografie.





## Moderne Kameras vs. Traditionelle Kameras

- **Moderne Kameras (Computational Photography):**
  - Komplexe Produkte, die eine über hundertjährige technische Evolution durchlaufen haben.
  - Grundlegender optischer Aufbau aus Linse(n), Blende und Sensor ist praktisch unverändert.
  - Aktuelle Systeme sind in der Lage, Leistungen zu erbringen, die das menschliche Auge übertreffen.



- **Traditionelle Kameras:**

- Nachteile, besonders bei der Bildaufnahme von traditionellen Kameras:  
*Führen dem Verlust einer Dimension der Szenengeometrie zu.*

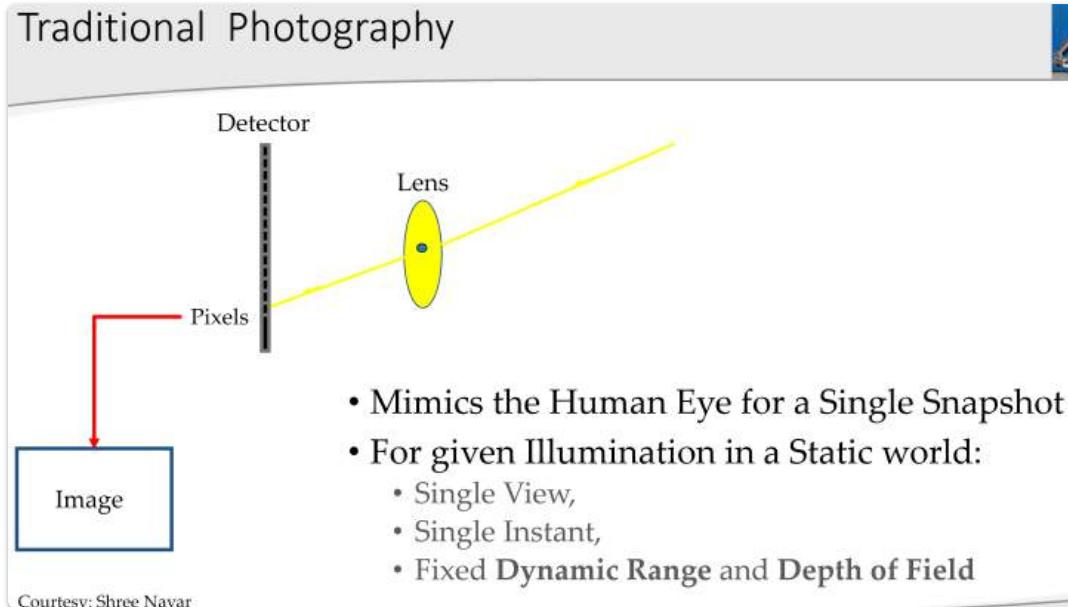
Der Sensor summiert einfallende Lichtstrahlen aus allen Einfallswinkeln zu einer Intensität.

*Informationen über die Reflexionseigenschaften von Szenenoberflächen (charakterisiert durch die Bidirectional Radiance Distribution Function, kurz BRDF) gehen verloren.*

Lassen sich aus der Bestrahlungsstärke auf der Bildebene nicht die strahlungsdichten Objekte bestimmen.

\* Die **Objektreffektivität** und der **unbekannten Bestrahlungsstärke** des Objekts werden zusammengesetzt dargestellt, was die Identifizierung und Klassifizierung von Objekten erschwert ("Helligkeits-Dilemma").

### Traditional Photography



## Zusätzliche optische Komponenten und deren Vorteile

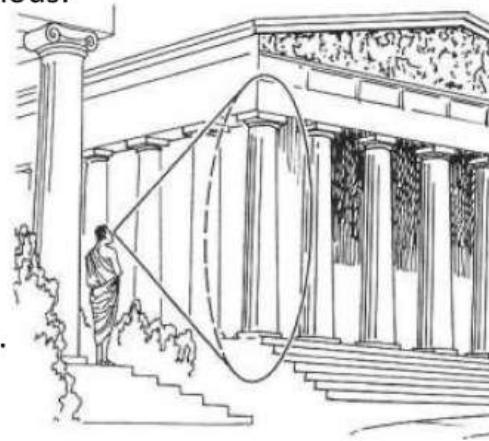
- **Abbildung in Kombination mit einer Blende zur Reduktion der Tiefunschärfe:**
  - Führt zu einem Verlust brauchbarer Informationen für weite Tiefenbereiche der Szene, da nur ein kleiner Bereich scharf abgebildet wird.
  - Verfügbare Sensoren integrieren über ein breites Spektrum der Wellenlängen einfallendes Licht.
  - Messung des spektralen Dimensions von Oberflächen (z.B. Farben) an unterschiedlichen Orten.
  - Zeit (Belichtungszeit) wird ebenfalls in die Dimension des Lichts integriert (zeitliche Auflösung geht verloren).
  - Räumliche Auflösung und die Quantisierung zu einer begrenzten Helligkeitsauflösung sind ebenfalls reduziert.
  - Dies führt zu einer **Reduktion des Signalumfangs (Dynamic Range)**.

- Core ideas are ancient, simple, and seem obvious:

- **Lighting:** ray sources
- **Optics:** ray bending / folding devices
- **Sensor:** measure light
- **Processing:** assess it
- **Display:** reproduce it

- **Ancient Greeks:**

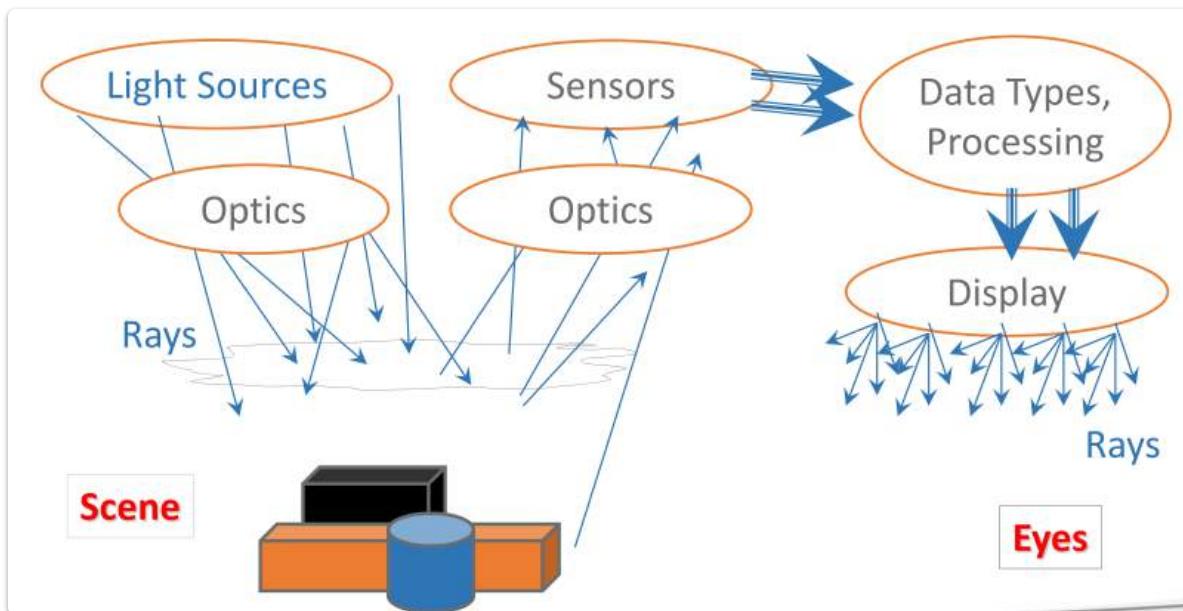
'Eye rays' wipe the world to feel its contents...



## 12.2 Lichtfeld

[EVC\\_Skriptum\\_CV](#), p.60, p.61

Das Lichtfeld ist eine Funktion, die die **Lichtmenge** beschreibt, die an jedem Punkt des dreidimensionalen Raums aus allen Richtungen einfällt.



- In der Computational Photography betrachtet man das Lichtfeld, das auf einen Punkt fällt, d.h., **welche Strahlen an diesen Punkt reflektiert werden**.
- **Objekte** werden als **Volumen** betrachtet.
- Einfallendes Beleuchtungsfeld wird in ein Beleuchtungsfeld umgewandelt, welches vom Objekt reflektiert wird.

### 4D-Lichtfeld-Parametrisierung

[EVC\\_Skriptum\\_CV](#), p.61

Einfallende Beleuchtung kann mittels eines **4D-Lichtfelds** parametrisiert werden.



- Man stellt sich um das Objekt eine Kugel vor, die das Objekt einschließt.
- **Position auf der Kugel:**  $(u_i, v_i)$
- **Richtung des eintreffenden Lichts:**  $(\theta_i, \phi_i)$
- **Einfallendes Lichtfeld:**  $R_i(u_i, v_i, \theta_i, \phi_i)$

Das ausgehende Licht kann ebenfalls mittels eines **4D-Lichtfelds** parametrisiert werden:

- **Was das Licht die umgebende Oberfläche verlässt:**  $R_r(u_r, v_r, \theta_r, \phi_r)$

Hier gabs dann noch ein Recap zur [PLenopischen Funktion: EVC-CV12-Computational Photography\\_S2025\\_Slides, p.11](#)

## Reflexionsfeld (8D-Funktion)

Die **Reflexionseigenschaften eines Objekts** können mittels einer **achtdimensionalen Funktion** - dem **Reflexionsfeld** - charakterisiert werden.

- Das Reflexionsfeld codiert für jeden einfallenden Lichtstrahl die 4D Reflexion des Lichtstrahls.
- Es beinhaltet die nötigen Informationen, um das Objekt unter verschiedenen Beleuchtungsgegebenheiten und von verschiedenen Blickwinkeln aus zu rendern.

## 4D-Lichtfeld-Kameras (Plenoptische Kameras)

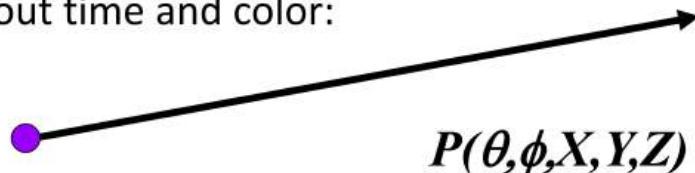
4D-Lichtfeld-Kameras sind nicht nur in der Lage, Ort und Intensität der Projektion der von einer Szene ausgehenden Lichtstrahlen zu detektieren, sondern auch den **Einfallswinkel** auf der Kamera-Ebene zu quantifizieren.



- Die so gewonnene **4D-Abtastung des Lichtfelds** einer Szene ermöglicht einen vielfältigen Informationsgewinn.
- **Aufbau:** 4D-Lichtfeld-Kameras verwenden ein **Mikrolinsen-Array**, das direkt auf dem Sensor aufgebracht wird.
- **Funktionsweise:** Die eintreffenden Lichtstrahlen werden durch die einzelnen Pixel im Sensorbereich unter jeder Mikrolinse nach ihrem Einfallswinkel quantifiziert, um ein 4D-Lichtfeld einer Szene zu rekonstruieren.
- **Nachteil:** Die direkt erreichbare **Ortsauflösung** entspricht der **Anzahl der Mikrolinsen**. Deren Durchmesser (in Pixeln) bestimmt wiederum die **Winkelauflösung**.
  - Je größer die Winkelauflösung, desto geringer ist die Ortsauflösung und umgekehrt.

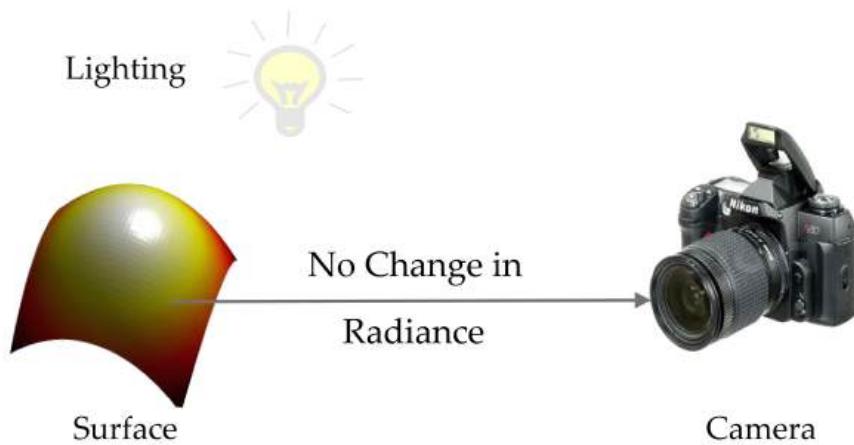
## Ray

- Let's not worry about time and color:



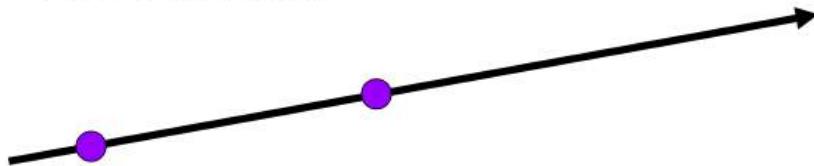
- 5D
  - 3D position
  - 2D direction

## How can we use this?



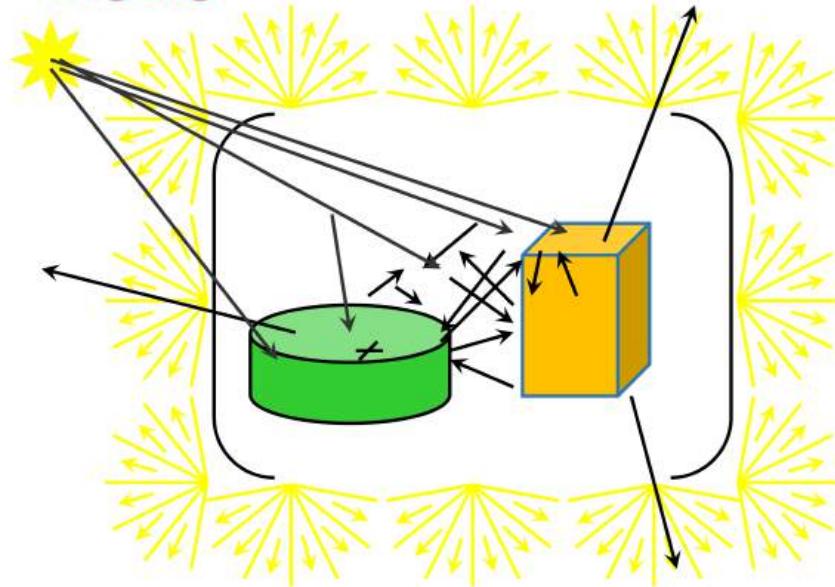
## Ray Reuse

- Infinite line
  - Assume light is constant (vacuum)



- 4D
  - 2D direction
  - 2D position
  - Non-dispersive medium

- Measure all the **outgoing** light rays.



$(u_i, v_i)$  indicate the position on the surface where the light enters,  
 $(\theta_i, \phi_i)$  indicate the direction in which it enters.

$(u_r, v_r)$  indicate the position on the surface where the light leaves,  
 $(\theta_r, \phi_r)$  indicate the direction in which it leaves.



$$(u_i, v_i, \theta_i, \phi_i ; u_r, v_r, \theta_r, \phi_r)$$

8D reflectance field

Since it is linear, we can represent as a matrix

$$R(u_i, v_i, \theta_i, \phi_i; u_r, v_r, \theta_r, \phi_r)$$

$360 \times 180 \times 180 \times 180 \times 360 \times 180 \times 180 \times 180$

= 4.4e18 measurements

x 6 bytes/pixel (in RGB 16-bit)

= 26 exabytes (billion GB)

= 1.3 million 20TB hard drives



(für nur ein Bild)

## High Dynamic Range (HDR)

[EVC\\_Skriptum\\_CV](#), p.61

**High Dynamic Range (HDR)** beschreibt eine Fülle an Techniken, die im Vergleich zu herkömmlichen Techniken für digitale Bildgebung oder fotografischen Methoden einen **größeren Bereich zwischen den hellsten und dunkelsten Regionen eines Bildes** darzustellen ermöglichen.

- **Ziel von HDR-Bildern:** Den Helligkeitsbereich in realen Szenen genauer zu repräsentieren.
- **Möglicher Helligkeitsbereich:** Erstreckt sich von direktem Sonnenlicht bis hin zu schwachem Sternenlicht.
- **Erzeugung:** Erzielt durch die Aufnahme von mehreren Bildern mit unterschiedlicher Belichtung der gleichen Szene.
- **HDR kompensiert Detailverlust**, der bei Nicht-HDR-Kameras mit einem einzigen Belichtungslevel auftritt (dort erhält man ein Bild mit Detailverlust in hellen oder dunklen Bildbereichen).
- **Ergebnis:** Ein Bild, dessen dunkle als auch helle Bildbereiche im Detail darstellbar sind.



## Anzeige von HDR-Bildern auf Geräten mit kleinem dynamischen Bereich

Um HDR-Bilder auf Geräten mit einem kleinen dynamischen Bereich anzuzeigen, werden **Techniken zur Abbildung von Farbtönen (engl. Tone Mapping)** benötigt.

- **Ziel von Tone Mapping:** Den gesamten Kontrast reduzieren.
- **Grund:** Ausdrücke, CRT- oder LCD-Monitore und Projektoren haben einen begrenzten dynamischen Bereich, der für die Darstellung der vollen Bandbreite der Lichtintensitäten ungeeignet ist.
- **Funktionsweise:** Tone Mapping behandelt die **Kontrastreduktion** der Lichtintensitäten der Szene zu den anzeigbaren Wertebereichen.
- **Wichtige Aspekte dabei:** Bilddetails und die Farbwirkung sollen möglichst erhalten bleiben, um den originalen Szeneninhalt wahrnehmen zu können.

Beispiel zu ToneMapping:



## Fotomontage/Komposition

EVC\_Skriptum\_CV, p.61

Fotomontage (auch bekannt als "Komposition" oder "image compositing") bezeichnet das Schneiden und Zusammenfügen von verschiedenen Fotos zu einem zusammengesetzten Bild.

- **Realisierung:** Heutzutage mittels moderner Bildbearbeitungssoftware.
- **Ziel:** Visuelle Elemente von unterschiedlichen Quellen werden in einem einzigen Bild kombiniert, um die **Illusion zu schaffen, dass alle diese Elemente Teil derselben Szene sind.**





- Bei Filmaufnahmen ist die Komposition unter verschiedenen Bezeichnungen bekannt:
  - "chroma key"
  - "blue screen"
  - "green screen" und andere Bezeichnungen.

## Image Inpainting

[EVC\\_Skriptum\\_CV](#), p.62

**Inpainting** ist ein Bildbearbeitungsprozess, der bei der **Rekonstruktion von schlecht erhaltenen oder nicht vorhandenen Bild- oder Videoteilen** Anwendung findet.

- **Analogie zur Malerei:** Im Bereich der Malerei wäre dies die Arbeit eines ausgebildeten Bildrestaurators.
- **In der digitalen Welt (auch bekannt als Bild- oder Videointerpolation):** Bezieht sich auf die Anwendung hochentwickelter Algorithmen, um **verlorene oder fehlerhafte Teile von Bilddaten zu ersetzen**.



- **Anwendungsbeispiele:**
  - **Entfernung eines Objekts aus einer Szene**, um ein beschädigtes Bild zu retuschieren.
  - **Für Fotografie und Film:**
    - Beseitigung von Zerfallsmerkmalen (z.B. Risse in Fotografien, Kratzer und Staubflecken im Film).
    - Hinzufügen oder Entfernen von Bildteilen (z.B. Aufnahmedatum).
- **Generelles Ziel:** Ein verändertes Bild zu produzieren, in dem die Inpaint-Region mit dem restlichen Bild so verschmolzen wird, dass ein gewöhnlicher Betrachter **nicht bemerkt**,

dass das Bild bearbeitet wurde.

## Warping

EVC\_Skriptum\_CV, p.62

**Warping** ist der Prozess der **Bildmanipulation**, bei dem jede im Bild vorkommende Form **räumlich signifikant verändert** wird.

Image filtering: change **range** of image

$$g(x) = T(f(x))$$

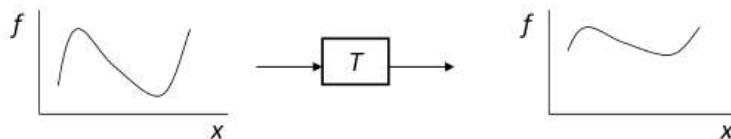


Image warping: change **domain** of image

$$g(x) = f(T(x))$$

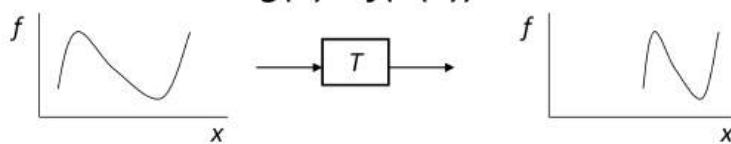


Image warping: change **domain** of image



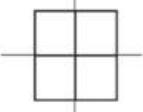
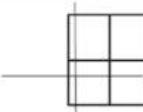
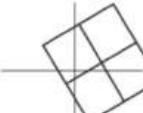
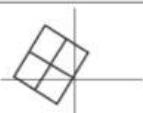
$$g(x) = f(T(x))$$



- **Anwendung:**

- **Verzerrung von Bildern korrigieren.**
- **Kreative Zwecke** (z.B. für Morphing).

- **Funktionsweise:** Beim reinen Warping wird der **Punkt auf einen anderen Punkt abgebildet, ohne die Farbe des Punktes zu ändern**.
- **Grundlage:** Warping kann auf jeder **Funktion** basieren, die (von einem Teil) einer Ebene auf eine Ebene abbildet.
- **Injektive Funktion:** Bei einer injektiven Funktion kann das Original wieder rekonstruiert werden.
- **Bijektive Funktion:** Wenn die Funktion bijektiv ist, kann das Bild invers transformiert werden.

<b>Identity</b>	$W(x) = x$	
<b>Translation</b>	$W(x; t) = x - t$	
<b>Rigid</b>	$W(x; R, t) = Rx - t$	
<b>Similarity</b>	$W(x; R, a, t) = aRx - t$	
<b>Affine</b>	$W(x; A, t) = Ax - t$	

where  $\alpha \in \mathcal{R}$ ,  $A \in \mathcal{R}^{2 \times 2}$ ,  $t \in \mathcal{R}^2$ ,  $R \in \mathcal{R}^{2 \times 2}$ ,  $|R| = 1$

- **Mathematische Beschreibung:**

- Gegeben sei ein gesampeltes Bild  $I(x)$ .
- **Warping** ist die **räumliche (geometrische) Deformation** des Quellbildes  $I_s(x)$ .
- Das Ergebnis  $I_d(x)$  wird **Zielbild** genannt.
- Das Warping wird mittels der **Warpingfunktion**  $W(x; p)$  mit den Parametern  $p$  bestimmt.
- Für jedes Pixel  $x$  in  $I_d$  gibt uns die Warpingfunktion an, woher dieses Pixel im Quellbild  $I_s$  stammt:  $I_d(x) = I_s(W(x; p))$ .

- **Zwei Formen von Transformationen:**

- **Affine Transformationen**
- **Projektive Transformationen** (auch perspektivische Transformation oder Homographie genannt).

## Affine Transformation

Die affine Transformation ist eine Transformation, die **gerade Linien und Distanzverhältnisse zwischen Punkten, die auf einer Linie liegen, erhält**.

- **Beispiel:** Der Mittelpunkt eines Liniensegments bleibt auch nach der Transformation dessen Mittelpunkt.
- **Wird nicht zwingend erhalten:** Längen und Winkel.
- **Typische affine Transformationen:** Translation (Verschiebung), Rotation (Drehung), Scherung und ähnliche Transformationen.
- **Kombinationen** von affinen Transformationen sind ebenfalls affin.

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

## Projektive Transformation

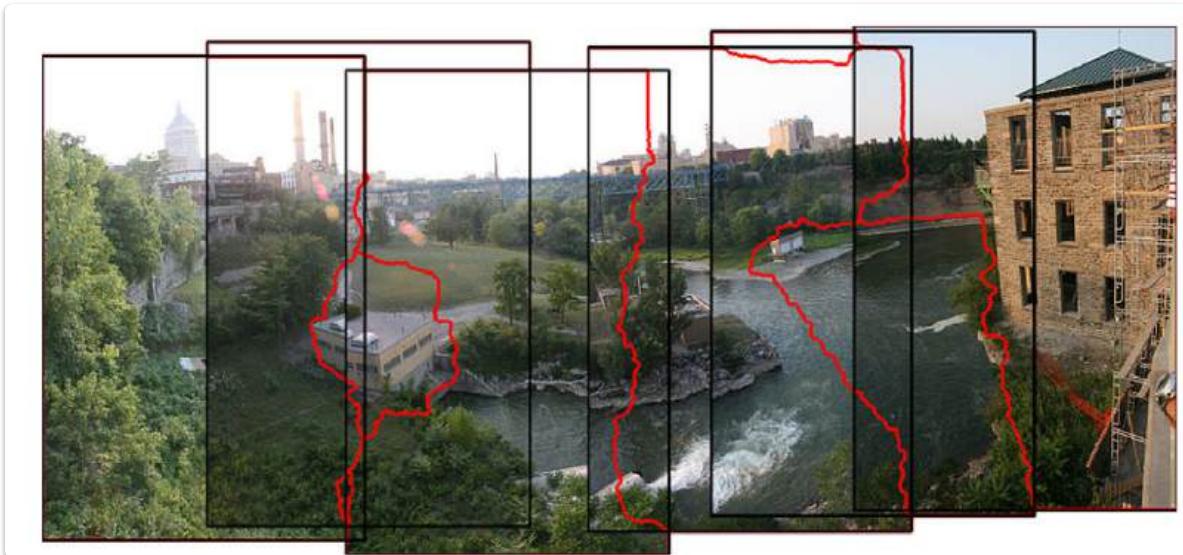
Die projektive Transformation einer Ebene ist eine Transformation, die in der **projektiven Geometrie** verwendet wird.

- **Wird nicht erhalten:** Größen oder Winkel.
- **Anwendung:** Jedes Bildpaar, das dieselbe ebene Fläche im Raum zeigt, steht mittels einer projektiven Transformation zueinander in Beziehung.
- **Praktische Anwendungen:**  
*Bildrektifizierung (Entzerrung von Bildern).*  
Bildregistrierung (Ausrichtung von Bildern zueinander).  
\* Berechnung der Kamerabewegung zwischen zwei Bildern.

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

## Bildmosaik (Image Mosaicing / Stitching)

EVC\_Skriptum\_CV, p.63



Der Prozess zur Erstellung eines **Bildmosaiks** (auch oft als **Stitching** bezeichnet) kombiniert mehrere Bilder mit überlappenden Bereichen zu einem **hochauflösenden Panoramabild**.

- **Voraussetzung für korrekte Überlagerung (ohne Parallaxenfehler):** Die Kamera muss um den **Fokuspunkt** rotiert werden.
- **Der Prozess kann in drei Bestandteile aufgeteilt werden:**
  1. **Bildregistrierung**
  2. **Kalibrierung**

### 3. Blending

## 1. Bildregistrierung

Bei der Bildregistrierung werden **korrespondierende lokale Merkmale** zwischen den Bildern bestimmt, um die Bilder korrekt "übereinander legen" zu können.

## 2. Kalibrierung

Die Kalibrierung versucht, die **Differenzen zwischen einem idealen Linsenmodell und dem realen Linsensystem der Kamera(s)** zu minimieren. Das bedeutet, sie korrigiert:

- Optische Defekte (wie **Verzerrungen**).
- Unterschiedliche **Belichtungszeiten** der Bilder.
- **Vignettierung** (Randabdunkelung).
- **Chromatische Aberrationen** (Farbsäume).

## 3. Blending ("Verschmelzen")

Das Blending korrigiert die im Kalibrierungsschritt festgemachten Abweichungen und **bildet die einzelnen Aufnahmen auf ein gemeinsames Ausgabebild ab**.

- **Farben werden zwischen den Bildern angepasst**, um die Beleuchtungsunterschiede auszugleichen.
- Die Bilder müssen so miteinander verschmolzen werden, dass **keine Übergänge (engl. seams)** von einem Einzelbild zum nächsten sichtbar sind.

## Image Morphing

---

[EVC\\_Skriptum\\_CV](#), p.63

**Image Morphing** ist ein spezieller Bildeffekt, der ein Bild **nahtlos in ein anderes verwandelt**.

- **Anwendung:** Oft in surrealen Sequenzen oder im Fantasyfilm-Genre verwendet, um z.B. eine Person in eine andere zu verwandeln.
- **Traditionelle Methode:** Durch Aus- und Einblendtechniken (engl. *fading*) im Film erzielt.
- **Seit den frühen 1990ern:** Immer häufiger Computertechniken verwendet, die überzeugendere Ergebnisse lieferten.



- **Heutige Methode (Computertechniken):** Beim Überblenden werden die Bilder gleichzeitig anhand von **markierten korrespondierenden Punkten verzerrt**.
- **Beispiel (Gesichts-Morphing):**
  - Umwandlung eines Gesichts in ein anderes.
  - **Keypoints** im ersten Gesicht (z.B. die Kontur der Nase oder Position der Augen) werden markiert. Diese Keypoints müssen auch im zweiten Bild existieren.
  - Anschließend wird das erste Gesicht **verzerrt**, um die Form des zweiten Gesichts zu erhalten.
  - **Gleichzeitig** wird das erste Gesicht **ausgeblendet** und das zweite Gesicht **eingebettet**.
- **Höher entwickelte Überblendungstechniken (später):** Verschiedene Teile eines Bildes werden graduell auf das andere überblendet, anstatt das gesamte Bild auf einmal zu wandeln.

### Idea #1: Cross-Dissolve



- Interpolate whole images:  

$$\text{Image}_{\text{halfway}} = (1-t) * \text{Image}_1 + t * \text{image}_2$$
- This is called **cross-dissolve** in the film industry
- But what if the images are not aligned?

- Align first, then cross-dissolve
  - Alignment using global warp – picture still valid



## Testähnliches Beispiel

- Benennen Sie die dargestellten Verfahren der Computational Photography:

Eingabebild(er)	Ergebnis	Verfahren
		<u>Image Inpainting</u>
		<u>Image Morphing</u>
		<u>Panoramic Images</u>
		<u>HDR – High Dynamic Range Imaging</u>

# 13. Computer Animation

## Arten von Animation

Flipbook



[https://youtu.be/p3q9MM\\_hM](https://youtu.be/p3q9MM_hM)

2D Animation



<https://tenor.com/bV6ph.gif>

Zoetrope



Stop Motion



### ■ Artistic Control vs. Automation



Time consuming, but  
flexible

Visual Effects



3D Animation



## Transformationen

[EVC\\_Skriptum\\_CG](#), p.55

- Eine affine Transformation, die sowohl eine Translation als auch eine Rotation beinhaltet, kann als homogene  $4 \times 4$  Matrix dargestellt werden (siehe [3. Transformationen](#)):
  - $\begin{pmatrix} R & x \\ 0 & 1 \end{pmatrix}$ , mit  $R \in \mathbb{R}^{3 \times 3}$  als Rotationale Komponente und Translation  $x \in \mathbb{R}^3$ .

### Translation

[EVC\\_Skriptum\\_CG, p.55](#)

- Angenommen, wir haben keine Rotation, dann kann ein Objekt mit der initialen Position  $p^{(t_0)} \in \mathbb{R}^3$  zum Zeitpunkt  $t_0$  zu einer Zielposition  $p^{(t_1)}$  zum Zeitpunkt  $t_1$  bewegt werden mittels:

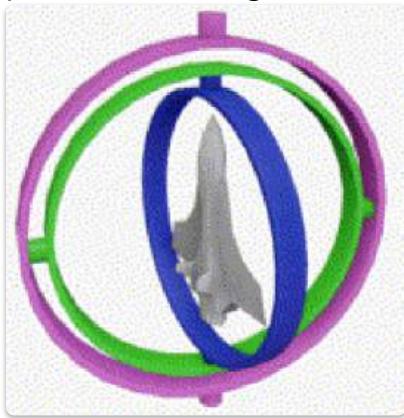
$$\mathbf{p}^{(t_1)} = \begin{bmatrix} 1 & \mathbf{x} \\ 0 & 1 \end{bmatrix} \mathbf{p}^{(t_0)}$$

- Die Position des Objekts zu einem beliebigen Zeitpunkt  $t \in \mathbb{R}$ , mit  $t_0 < t < t_1$ , kann zwischen der Start- und Endposition linear interpoliert werden:

$$p^{(t)} = p^{(t_0)} + (p^{(t_1)} - p^{(t_0)}) \frac{t - t_0}{t_1 - t_0}$$

[Rotation](#)[EVC\\_Skriptum\\_CG, p.55](#)

- Matrixdarstellungen von Rotationen** sind eine instabile Darstellung und bereiten Probleme, wie z.B. das *Gimbal Lock*. Normalerweise hat man drei Freiheitsgrade, wenn man ein Objekt im 3D-Raum rotiert.
- Verwendet man jedoch Matrizen für Rotationen, gibt es Konfigurationen, bei denen ein Objekt so gedreht wird, dass zwei Achsen parallel sind.
- Im Bild rechts zum Beispiel verklemmen sich der pinkfarbene und der grüne Kreis, und es macht keinen Unterschied mehr, ob man den inneren blauen Ring oder den äußeren pinkfarbenen Ring dreht. Somit haben wir einen Freiheitsgrad verloren.



- Eine Lösung besteht darin, **Quaternionen** zur Darstellung von Rotationen und *sphärische Interpolation* zu verwenden.
- Quaternionen sind eine Erweiterung der komplexen Zahlen und bestehen aus einer skalaren Komponente  $s \in \mathbb{R}$  und einer Vektor-Komponente  $\mathbf{v} \in \mathbb{R}^3$ .
- Um eine beliebige Achse  $\mathbf{n} \in \mathbb{R}^3$  um einen Winkel  $\phi$  zu rotieren, kann diese Achsenwinkel-Darstellung einer Rotation in ein Quaternion  $q$  wie folgt umgewandelt werden:

$$q = [s; \mathbf{v}] = [\cos \frac{\phi}{2}; \sin \frac{\phi}{2} \mathbf{n}]$$

(Hierbei ist  $n$  ein Einheitsvektor der Rotationsachse.)

- $q^{(t)} = \text{slerp}(q^{(t_0)}, q^{(t_1)}, t) = q^{(t_0)}(q^{(t_0)})^{-1}q^{(t_1)})^t$

(slerp steht für *spherical linear interpolation* und interpoliert zwischen zwei Quaternionen  $q^{(t_0)}$  und  $q^{(t_1)}$  über die Zeit  $t$ .)

- Um schließlich eine rotierte Position  $P^{(t)}$  zu einem beliebigen Zeitpunkt  $t$  zu berechnen, wenden wir die interpolierte Rotation  $q^{(t)}$  auf die Ausgangsposition  $P^{(t_0)}$  an.
- Dazu bilden wir zuerst ein neues Quaternion, indem wir  $P^{(t_0)}$  als Vektor-Komponente und Null als skalare Komponente nehmen.
- Das Ergebnis des Produkts mit der berechneten Rotation  $q^{(t)}$  ist wiederum ein Quaternion mit Null als skalarer Komponente, und die Vektor-Komponente ist unsere gewünschte, rotierte Position  $P^{(t)}$ :

$$[0; P^{(t)}] = q^{(t)} \cdot [0; P^{(t_0)}] \cdot (q^{(t)})^{-1}$$

- Matrix representation of rotations is often unstable
  - Additional Problem: *Gimbal Lock*
- Best use **Quaternions**



$$\mathbf{q} = [s; \mathbf{v}] = [s \quad \underbrace{\begin{matrix} x & y & z \end{matrix}}_{\text{vector}}]$$

scalar

- Generalization of complex numbers
- Also written as:

$$\mathbf{q} = s + xi + yj + zk$$

with

$$i^2 = j^2 = k^2 = -1$$

$$i \cdot j \cdot k = -1$$

■ **Quaternions:**  $\mathbf{q} = [s; \mathbf{v}] = [s \quad x \quad y \quad z]$



■ Operations:

■ Addition:  $\mathbf{q}_1 + \mathbf{q}_2 = [(s_1 + s_2) \quad (x_1 + x_2) \quad (y_1 + y_2) \quad (z_1 + z_2)]$

■ Multiplication:  $\mathbf{q}_1 \cdot \mathbf{q}_2 = \begin{bmatrix} (s_1 \cdot s_2 - x_1 \cdot x_2 - y_1 \cdot y_2 - z_1 \cdot z_2) \\ (s_1 \cdot s_2 + x_1 \cdot x_2 + y_1 \cdot y_2 - z_1 \cdot z_2) \\ (s_1 \cdot s_2 - x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2) \\ (s_1 \cdot s_2 + x_1 \cdot x_2 - y_1 \cdot y_2 + z_1 \cdot z_2) \end{bmatrix}$

■ Inverse of normalized Quaternion:  $\mathbf{q}^{-1} = \bar{\mathbf{q}} = [s \quad -x \quad -y \quad -z]$

- Can be computed from angle  $\phi$  and normalized vector  $\mathbf{n}$  (*axis-angle representation*):

$$\mathbf{q} = \left[ \cos \frac{\phi}{2}; \sin \frac{\phi}{2} \mathbf{n} \right]$$

■ Spherical Interpolation:

$$\mathbf{q}^{(t)} = \text{slerp}(\mathbf{q}^{(t_0)}, \mathbf{q}^{(t_1)}, t) = \mathbf{q}^{(t_0)} (\mathbf{q}^{(t_0)-1} \mathbf{q}^{(t_1)})^t$$

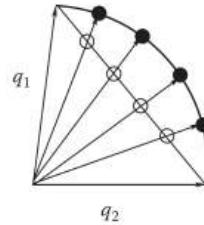


Figure taken from Shirley&Manzner:  
Foundations of Computer Graphics, 5th Edition

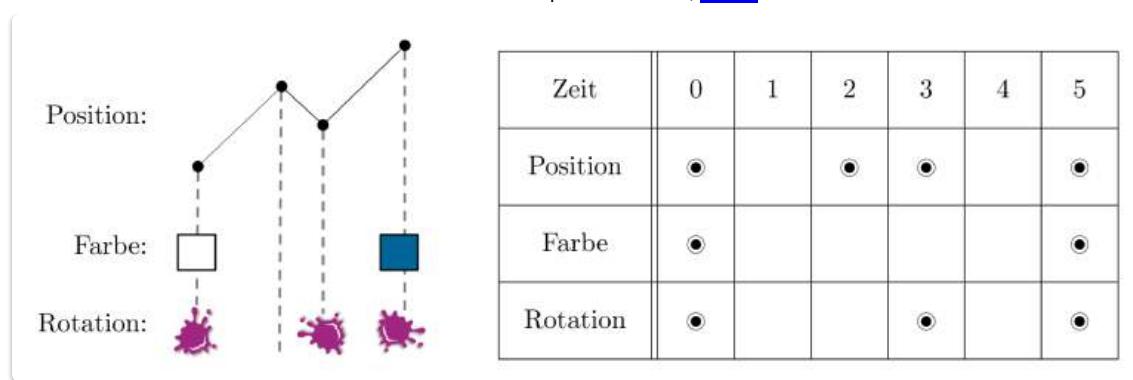
■ Apply a rotation to a point  $\mathbf{p}^{(t)}$ :

$$[0; \mathbf{p}^{(t)}] = \mathbf{q}^{(t)} \cdot [0; \mathbf{p}^{(t_0)}] \cdot \mathbf{q}^{(t)-1}$$

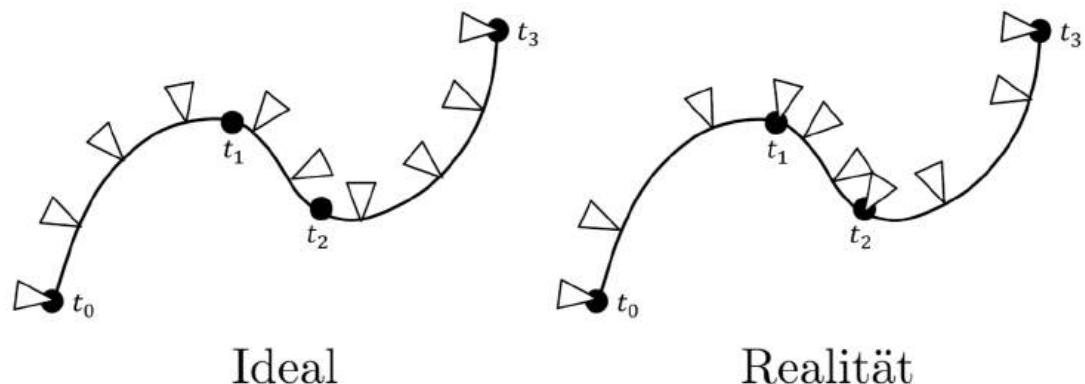
## Keyframing

EVC\_Skriptum\_CG, p.55, p.56

- **Keyframing** ermöglicht komplexe Animationen mit einem Gleichgewicht zwischen Automatisierung und manueller Spezifikation.
- Szenenparameter werden nur zu bestimmten Zeitpunkten (**Keyframes**) festgelegt und sonst interpoliert.
- Parameteränderungen im Laufe der Zeit können in einer Tabelle kodiert werden.
- Die Zeitschritte bezeichnen wir als **Frames**.
- Jede Kombination aus einer **Zeit t** und einer Menge an **Szenenparametern f** zu diesem Zeitpunkt nennen wir einen **Keyframe k**.
- In unserem Fall besteht diese Menge von Szenenparametern f aus der Position p, Farbe c und einer Rotation, die durch ein Quaternion q dargestellt wird.
- Dies ergibt das Keyframe k:  $(t_k, f^{(t_k)}) = (t_k, (p^{(t_k)}, c^{(t_k)}, q^{(t_k)}))$ .
- Es kann auch nur eine **Teilmenge** der Szenenparameter angegeben werden, wie im Beispiel für Frame 2 und 3.



- Das Anpassen einer interpolierenden Kurve durch alle Keyframe-Positionen führt zu einer flüssigeren Bewegung anstelle einer linear Positionsänderung.
  - Je nach verwendeter Methode zur Konstruktion der Kurve können jedoch *plötzliche Sprünge* um die Keyframe-Positionen herum auftreten und die Geschwindigkeit, die das Objekt entlang der Kurve hat, kann *unregelmäßig* verlaufen (schlecht für eine gleichmäßige Kamerabewegung).
  - Idealerweise würden wir die Zeit gleichmäßig abtasten und erwarten, dass die resultierenden Punkte entlang der Kurve ebenfalls gleichmäßig verteilt sind.
  - Die meisten Kurvenberechnungsverfahren gruppieren jedoch die räumlichen Abtastpunkte nicht gleichmäßig.
  - **Problem:** Ungleichmäßige räumliche Verteilung der Abtastpunkte entlang der Kurve.
  - **Lösung für dieses Problem:** Die Zeit nicht gleichmäßig abtasten. Stattdessen bestimmen wir die *Länge der Kurve* (Bogenlänge) näherungsweise und teilen diese dann gleichmäßig ab. Dies bedeutet, dass wir die Kurve in Abschnitte gleicher Länge unterteilen.

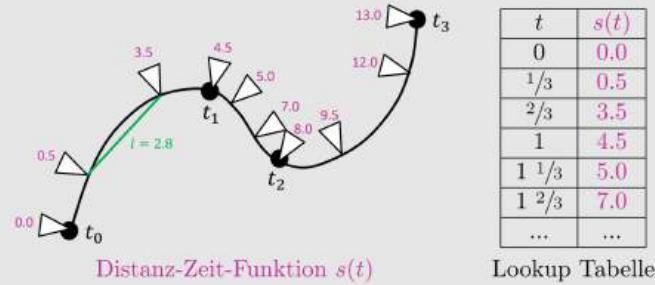


- Um dies zu erreichen, führen wir die **Distanz-Zeit-Funktion**  $s(t) : \mathbb{R} \rightarrow \mathbb{R}$  ein.
  - Sie gibt im Grunde an, wie weit wir entlang einer Kurve bereits gereist sind, d.h. sie ordnet jedem Zeitpunkt die bis dahin zurückgelegte Entfernung zu.
  - Während  $s(t)$  ursprünglich kontinuierlich ist, können wir sie diskretisieren und die Entfernung zwischen zwei Zeitpunkten durch euklidischen Distanz approximieren.
  - Somit approximieren wir die Form der Kurve mittels mehrerer linearer Abschnitte:
    - $s(t_i) \approx s(t_{i-1}) + \|P_i - P_{i-1}\|$  (Die Distanz zum Zeitpunkt  $t_i$  ist die Summe der vorherigen Distanz und der euklidischen Distanz zwischen dem aktuellen und vorherigen Punkt.)

- Wir können diese approximierten Werte unserer Distanz-Zeit-Funktion  $s$  in einer *Lookup-Zeit-Tabelle* speichern.
- Wenn wir dann  $s(t)$  gleichmäßig abtasten, berechnen wir die Werte  $t$ , die wir benötigen, um unsere Kurve unter Verwendung dieser Lookup-Tabelle und linearer Interpolation auszuwerten.
- Durch nicht gleichmäßiges Abtasten der Zeit haben wir somit eine *ungefähr gleichmäßige Raumabtastung* erreicht.

**Beispiel:**

Im Folgenden nehmen wir an, dass wir eine Methode haben, um die **genaue Distanz-Zeit-Funktion** zu berechnen, was zu den **rosa Werten** im untenstehenden Bild führt. Ein Beispiel zur Berechnung einer Approximation von  $s(t)$  zur Zeit  $t = 0.6$  ist rechts, wobei die **euklidische Distanz** zwischen  $\mathbf{p}_{0.3}$  und  $\mathbf{p}_{0.6}$  verwendet wird, von der wir ebenfalls annehmen, dass sie gegeben ist.

Approximiere  $s(t)$ :

$$s(t_{0.6}) \approx s(t_{0.3}) + l = 0.5 + 2.8 = 3.3$$

Indem wir unsere Approximation von 3.3 mit dem richtigen Wert von 3.5 aus der Abbildung rechts vergleichen, stellen wir fest, dass wir einen Fehler 0.2 gemacht haben.

Als nächstes **tasten wir  $s(t)$  gleichmäßig ab, z.B. bei 0.0, 1.0, 2.0, ... und berechnen mittels linear Interpolation die Werte  $t$ , an denen wir unsere Kurve auswerten wollen.** Zum Beispiel, wenn wir die Position eines Punktes entlang der Kurve nach der Distanz  $s = 6.0$  berechnen wollen, benötigen wir dafür:

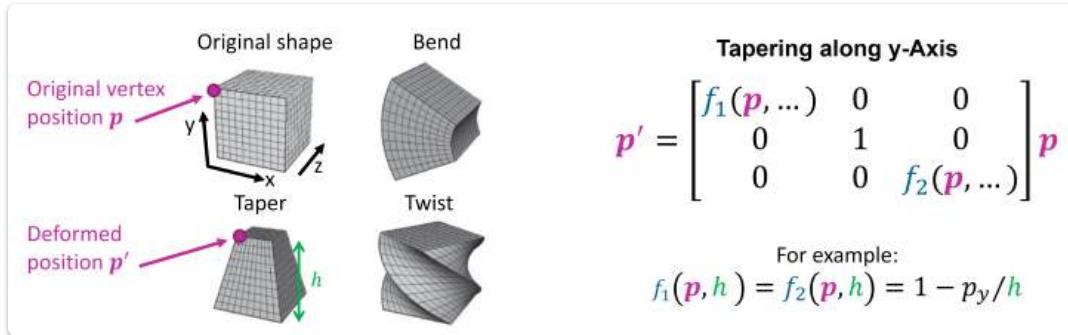
$$t = \frac{1.3 + 1.6}{2} = 2.5, \quad \text{da } 6.0 \text{ in der Mitte von } 5.0 \text{ und } 7.0 \text{ ist (siehe Tabelle).}$$

## Deformationen

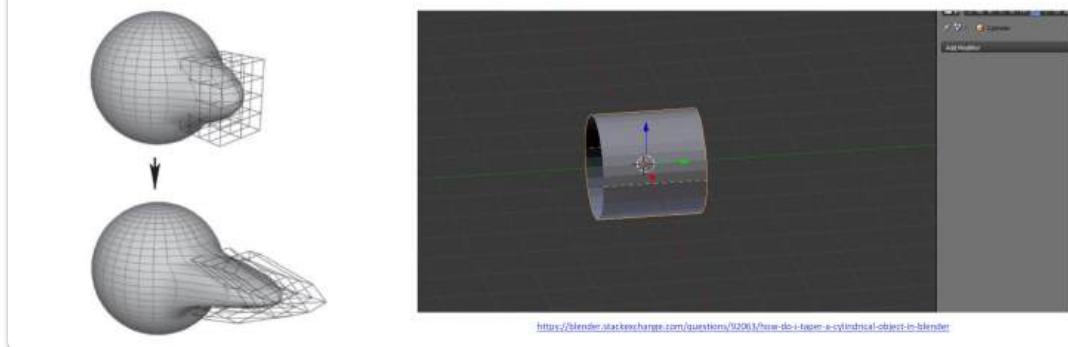
[EVC\\_Skriptum\\_CG, p.57](#)

- **Definition:** Eine Deformation kann als Funktion  $f$  definiert werden, die auf jede Vertex-Position  $p$  angewendet wird.
- Ziel ist es, die neue, deformierte Position  $p' = f(p, \gamma)$  zu erhalten.
- $\gamma$  sind optionale Parameter, die die Deformation steuern.
- **Einfache Deformationen:**
  - Können in Form von **nicht konstanten Matrizen** dargestellt werden.
  - Beispiele:
    - **Verjüngungs-/Taper-Operationen** (Verjüngung eines Objekts)
    - **Biege-/Bend-Operationen** (Biegen eines Objekts)
    - **Verdrehungs-/Twist-Operationen** (Verdrehen eines Objekts)
- **Komplexere Deformationen:**
  - Eine mögliche Lösung ist die Verwendung eines **Deformationskäfigs**.
  - **Vorteil:** Reduziert die Anzahl der manuell festzulegenden Vertex-Positionen erheblich.
  - **Funktionsweise:** Die Vertices (Eckpunkte) des Objekts sind an den Käfig "gebunden".

- **Effekt:** Immer wenn der Käfig bewegt wird, ändert sich auch die Position der Objekt-Vertices.



**Free Form Deformation:** Define  $f$  wrt. a deformation cage



## Physikbasierende Simulation

[EVC\\_Skriptum\\_CG](#), p.57

Hierbei wird ein Objekt anhand physikalischer Gesetze simuliert und durch mehrere Punkte beschrieben, die über Bedingungen (engl. *Constraints*) verbunden sind.

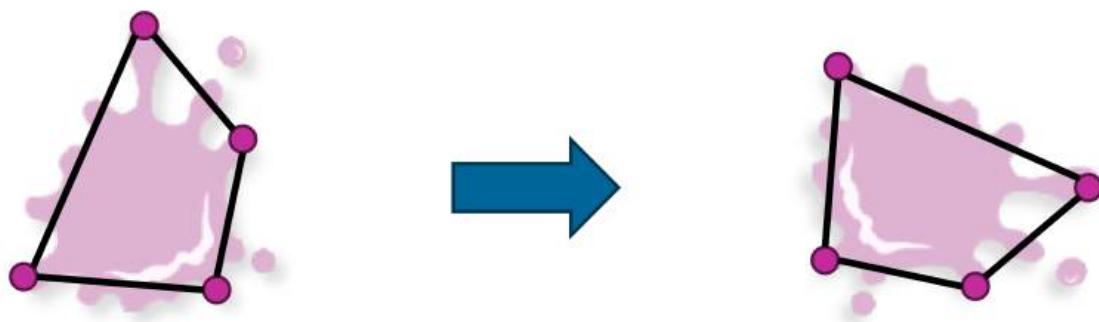
### Punkt:

- **Position**  $p \in \mathbb{R}^3, [m]$
- **Geschwindigkeit**  $v = \frac{dp}{dt} \in \mathbb{R}^3, [m \cdot s^{-1}]$
- **Beschleunigung**  $a = \frac{d^2p}{dt^2} \in \mathbb{R}^3, [m \cdot s^{-2}]$
- **Masse**  $m \in \mathbb{R}, [kg]$

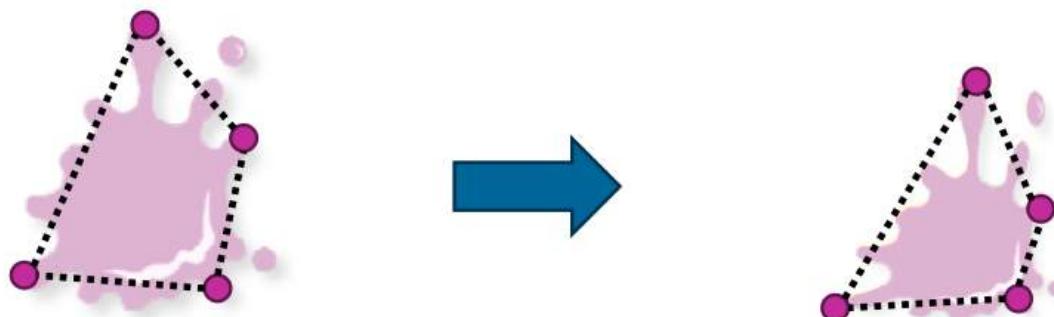
### Einschränkung:

- **Starr:** Distanz/Volumen bleibt gleich, nur Transformationen (Holzwürfel, Stahlschwert) (engl. *rigid*)
- **Weich:** Distanz/Volumen variabel, Transformationen & Deformationen (Kleidung, weiches Gewebe) (engl. *soft*)

Unser Hauptinteresse besteht darin, die **Position**  $p$  zu simulieren. Dazu betrachten wir ebenfalls die Veränderung der Position über die Zeit, also die **Geschwindigkeit** eines Punktes.



- Distance stays the same
- Transformations only



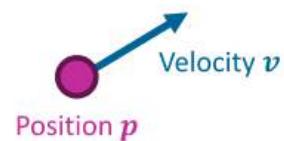
- Distance can change
- Transformations + Deformations

## Bewegung bei konstanter Geschwindigkeit

Wenn die Geschwindigkeit  $v$  konstant ist (d.h. sie hängt nicht von der Zeit ab), können wir die zukünftige Position  $\mathbf{p}^{(t+\Delta t)}$  eines Punktes nach einem beliebigen Zeitschritt  $\Delta t$  basierend auf der aktuellen Position  $\mathbf{p}^{(t)}$  zur Zeit  $t$  wie folgt berechnen:

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}$$

- Represent by multiple points
- Velocity  $\mathbf{v}$  is the change of position over time



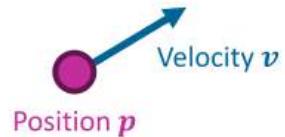
- If there is *constant* velocity, we can thus change the position via:

$$\mathbf{p}^{(t+1)} = \mathbf{p}^{(t)} + \frac{d\mathbf{p}^{(t)}}{dt} = \mathbf{p}^{(t)} + \mathbf{v}$$

Auf diese Weise erfolgt die Bewegung des Punktes mit konstanter Geschwindigkeit entlang einer geraden Linie (**Newtons erstes Gesetz**).

■ With *constant velocity*:

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}$$



Newton's 1<sup>st</sup> law

"A body *remains* at rest,  
or in motion  
at a constant speed  
in a straight line,  
unless acted upon by a **force**."

[https://commons.wikimedia.org/wiki/File:Portrait\\_of\\_Sir\\_Isaac\\_Newton,\\_1689\\_\(brightened\).jpg](https://commons.wikimedia.org/wiki/File:Portrait_of_Sir_Isaac_Newton,_1689_(brightened).jpg)

## Änderung der Geschwindigkeit

Wir können die Geschwindigkeit ändern, indem wir eine Kraft  $\mathbb{F} \in \mathbb{R}^3$ , [ $kg \cdot m \cdot s^{-2}$ ] wie zum Beispiel die Schwerkraft auf unseren Körper ausüben. Die Änderung der Geschwindigkeit wird als **Beschleunigung**  $a$  bezeichnet, die gemäß dem **zweiten Newtonschen Gesetz** berechnet werden kann:

$$\mathbb{F} = m \cdot a \Rightarrow a = \frac{\mathbb{F}}{m}$$

■ If there is *varying* velocity and a *constant* Force:

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t)} + \frac{1}{2}(\Delta t)^2 \mathbf{a}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \mathbf{a}$$

$$\mathbf{a} = \frac{\mathbf{F}}{m}$$

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \int_t^{t+\Delta t} \mathbf{v}^{(t')} dt'$$

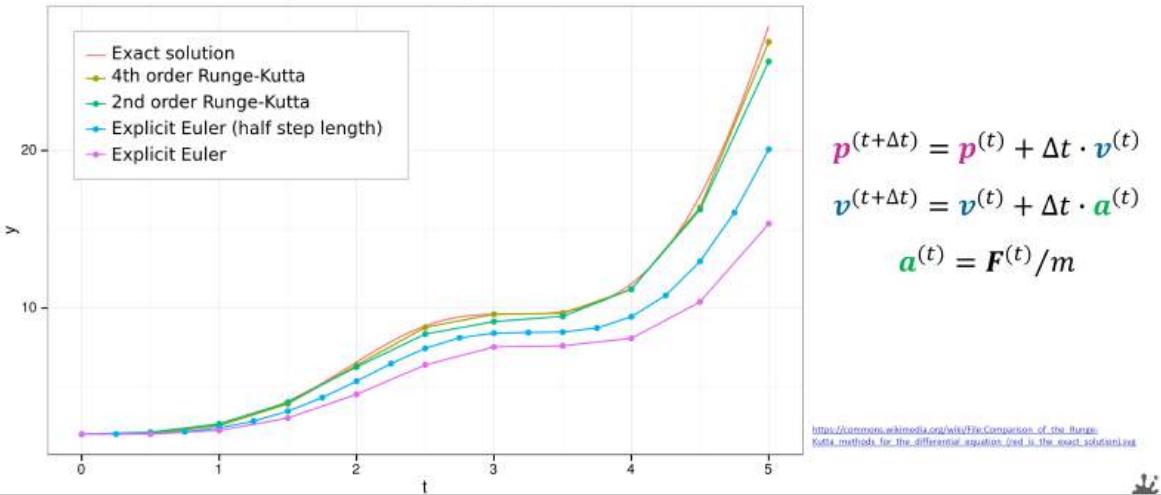
Hard to compute

⇒ use numerical integration methods!

$$\mathbf{a}^{(t)} = \mathbf{F}^{(t)}/m$$

Wenn sowohl eine sich ändernde Geschwindigkeit als auch eine sich ändernde Kraft annehmen, beinhaltet die Berechnung von  $\mathbb{P}^{(t+\Delta t)}$  mehrere Integrale.

## ■ Numerical integration: **Explicit Euler**



- Large timesteps: Unstable  
Small timesteps: more computations
- Better explicit integration scheme:  
**Runge-Kutta**

- More accurate, even with larger timesteps
- But also involves more computations again
- Often used in offline settings  
or when higher accuracy is required

## Numerische Integration

### Explicit Euler:

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t)}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t)}$$

$$\mathbf{a}^{(t)} = \frac{\mathbf{F}^{(t)}}{m}$$

unstabil bei großen Zeitschritten  
(hohe Abweichung/Oszillationen),  
einfache Berechnung,  
real-time

### Implicit Euler:

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t+\Delta t)}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t+\Delta t)}$$

$$\mathbf{a}^{(t+\Delta t)} = \frac{\mathbf{F}^{(t+\Delta t)}}{m}$$

bedingungslos stabil,  
aufwändige Berechnung  
(Lösung eines Gleichungssystems),  
offline

### Semi-Implicit Euler:

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t+\Delta t)}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t)}$$

$$\mathbf{a}^{(t)} = \frac{\mathbf{F}^{(t)}}{m}$$

energiebewahrend,  
einfache Berechnung,  
real-time

```

1 //Expliziter Euler
2 Vec3 F = berechneKraft();
3 Vec3 beschleunigung = F * objekt.inverseMasse;
4 objekt.position += objekt.geschwindigkeit * deltaZeit;
5 objekt.geschwindigkeit += beschleunigung * deltaZeit;

1 //Semi-Impliziter Euler
2 Vec3 F = berechneKraft();
3 Vec3 beschleunigung = F * objekt.inverseMasse;
4 objekt.geschwindigkeit += beschleunigung * deltaZeit;
5 objekt.position += objekt.geschwindigkeit * deltaZeit;

```

## ■ Numerical integration: Explicit Euler

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t)}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t)}$$

$$\mathbf{a}^{(t)} = \mathbf{F}^{(t)}/m$$

```

Vec3 F = computeForce();
Vec3 acceleration = F * object.inverseMass;
object.position += object.velocity * deltaTime;
object.velocity += acceleration * deltaTime;

```

## ■ Numerical integration: Semi-Implicit Euler

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t+\Delta t)}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t)}$$

$$\mathbf{a}^{(t)} = \mathbf{F}^{(t)}/m$$

```

Vec3 F = computeForce();
Vec3 acceleration = F * object.inverseMass;
object.velocity += acceleration * deltaTime;
object.position += object.velocity * deltaTime;

```

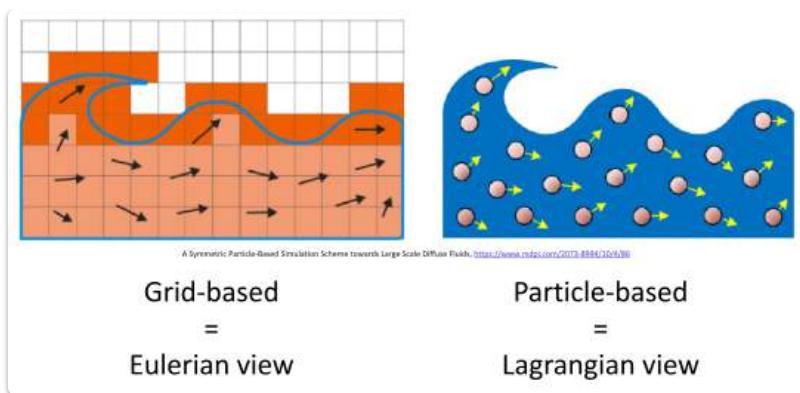
*Simple & more stable  
Often used in practice*

# Differentialgleichungen

[EVC\\_Skriptum\\_CG](#), p.58

Bisher haben wir uns sogenannte **partikelbasierte Simulationen** angesehen, bei denen eine Form durch mehrere Punkte approximiert wird. Dies wird auch als **Lagrange-Sichtweise** der Simulation bezeichnet.

Ein anderer Ansatz besteht darin, den Raum mit einem oft gleichmäßigen Gitter zu unterteilen, was dann als **Euler-Sichtweise** bezeichnet wird.



Letztendlich formulieren wir **Differentialgleichungen**, d.h. Gleichungen, die eine Funktion und ihre Ableitungen enthalten.

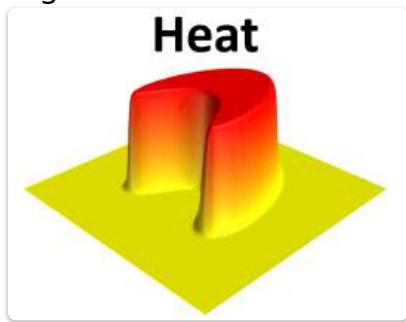
## Arten von Differentialgleichungen

### Gewöhnliche Differentialgleichungen (ODEs)

- Enthalten nur vollständige Ableitungen (geschrieben als  $\frac{dp}{dx}$ ,  $p'(x)$  oder manchmal  $\dot{p}$ ).
- **Beispiel:** Zweites Newtonsches Gesetz  $F(t, \mathbf{p}, \mathbf{v}) = m \frac{d^2\mathbf{p}^{(t)}}{dt^2}$ .

### Partielle Differentialgleichungen (PDEs)

- Komplexer als ODEs.
- Beschreiben mehrdimensionale Funktionen mit mehreren Parametern und deren partiellen Ableitungen (z.B.  $\frac{\partial p}{\partial x}$  oder  $\partial_x p$ ) bezüglich nur eines der Parameter.
- **Beispiel:** Die 2D-Wärmeleitungsgleichung (Bild 2), die beschreibt, wie sich Wärme entlang einer 2D-Oberfläche ausbreitet.
- **Weitere Anwendungen:** Akustische Wellen oder Fluidsimulationen (wobei die zugrunde liegende PDE die Navier-Stokes-Gleichung ist).



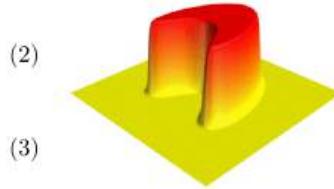
## Numerische Berechnung

Eine Technik zur numerischen Berechnung von Ableitungen sind sogenannte **Finite-Differenzen-Verfahren** (Bild 3).

Die unbekannten Werte können wiederum mithilfe von **expliziten** oder **impliziten Verfahren** berechnet werden.

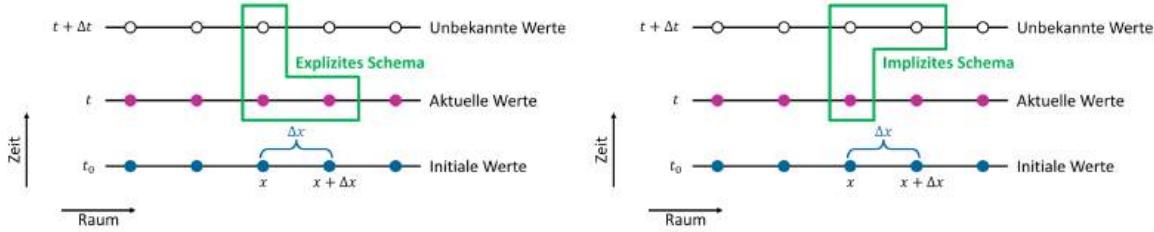
$$\frac{\partial h}{\partial t} = \alpha \left( \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \right), \quad \text{with } h : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}$$

$$\frac{\partial u(x, t)}{\partial x} \approx \frac{u(x + \Delta x, t) - u(x, t)}{\Delta x}$$



(2)

(3)



## Charakter Animation

EVC\_Skriptum\_CG, p.58

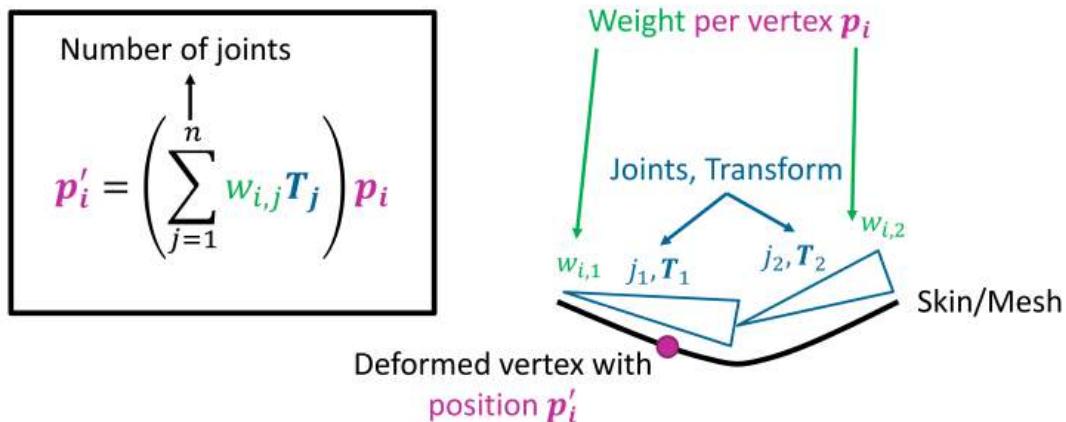
**Linear Blend Skinning (LBS)** ist eine einfache Methode zur Berechnung aktualisierter Vertex-Positionen basierend auf der Deformation eines zugrundeliegenden Skeletts.

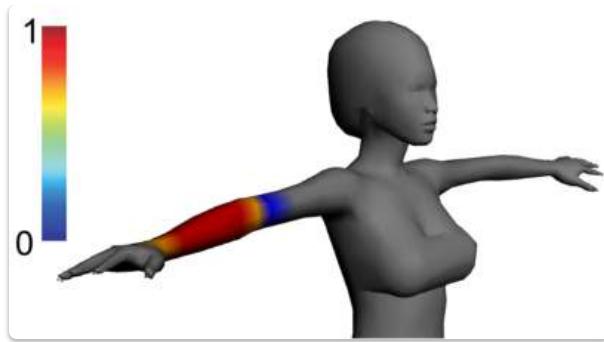
### Funktionsweise von LBS

1. **Gelenk-Transformation:** Jedes Gelenk  $j$  wird eine Transformation  $\mathbb{T}_j$  zugewiesen (die auch von einer Hierarchie von Gelenken abhängen kann und zum Beispiel als Matrix  $\mathbb{T}_j \in \mathbb{R}^{4 \times 4}$  dargestellt werden kann).
2. **Einfluss auf Vertices:** Jedes Gelenk  $j$  hat einen individuellen Einfluss auf jeden Vertex  $i$ , der durch das Gewicht  $w_{i,j}$  erfasst wird.
3. **Deformation:** Um einen Vertex  $i$  von seiner ursprünglichen Position  $\mathbb{p}_i$  zu einer deformierten Position  $\mathbb{p}'_i$  zu bewegen, summiert man über die gewichteten Beiträge aller Gelenke ( $n$  ist die Anzahl der Gelenke):

$$\mathbb{p}'_i = \left( \sum_{j=1}^n w_{i,j} \mathbb{T}_j \right) \mathbb{p}_i$$

#### ■ Linear Blend Skinning:





## Probleme von LBS

LBS hat jedoch mehrere Probleme, wie zum Beispiel:

- Hoher manueller Aufwand.
- Das „Bonbonverpackungs“-Artefakt (engl. *candy wrapper effect*)



## Alternativen zu LBS

Alternativ können **Ragdolls** basierend auf physikalischer Simulation oder **Motion-Capture-Techniken** verwendet werden:

- **Motion-Capture:** Die Bewegungen realer Menschen werden direkt über Kameras und Marker auf den Schauspielern erfasst und dann auf einen virtuellen Charakter übertragen.

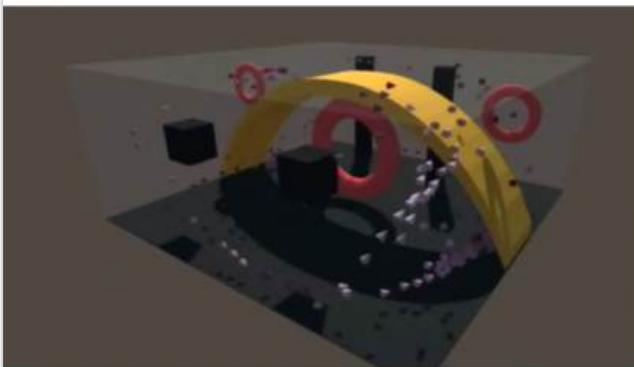
## Prozedurale Techniken

---

[EVC\\_Skriptum\\_CG](#), p.58

Ähnlich wie physikbasierte Simulationen können **prozedurale Techniken** verwendet werden, um automatisch Animationen zu erstellen.

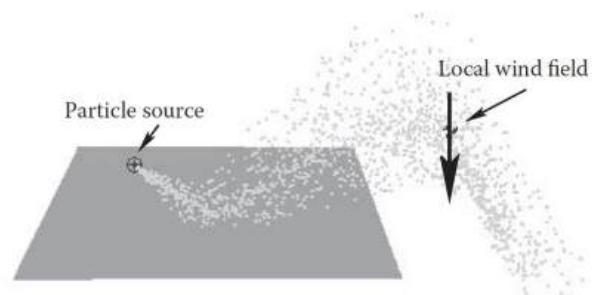
## Swarm/Flock/Boids



## Game of Live (Cellular automata)



## Particle System + Wind Field



## Merkmale

- Folgen einer bestimmten Reihe von Regeln, kombiniert mit Zufallsprinzipien.
- Ziel: plausible Animationen zu erstellen, jedoch nicht mit dem Ziel der physikalischen Korrektheit.

## Beispiele

- **Game of Life:** Ein bekanntes Beispiel für einen zellulären Automaten.
- **Partikelsysteme:** Für Phänomene wie Regen, Schnee, Feuer.
- **Schwarmverhalten:** Erzeugung von plausiblem Schwarmverhalten (z.B. Vogelschwärme oder Fischschwärme).

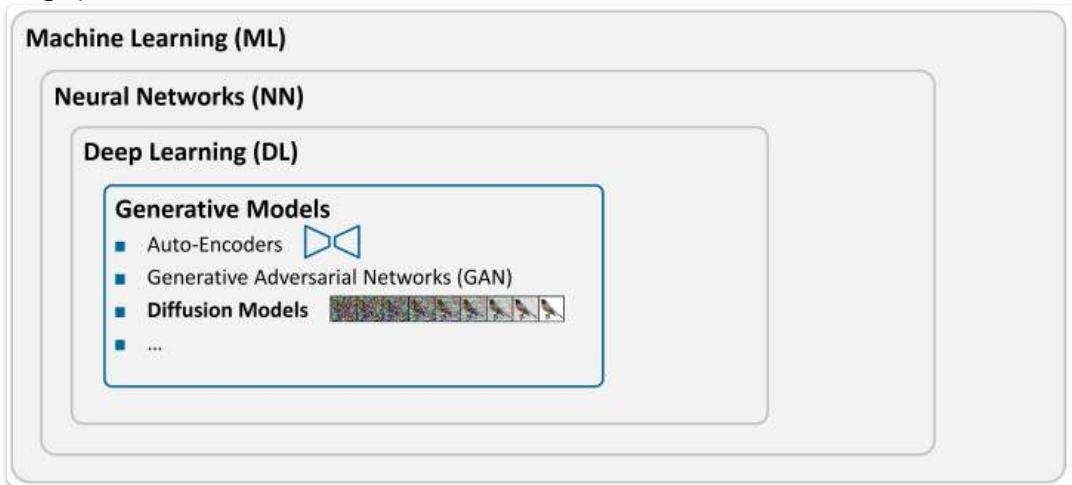
# 14. Machine Learning für 3D Graphics

## Machine Learning und Neural Networks

EVC\_Skriptum\_CG, p.59

- **Maschinelles Lernen (ML):**

- Ein Zweig der *Künstlichen Intelligenz*.
- Konzentriert sich auf die Entwicklung von Algorithmen, die aus Daten lernen, um Vorhersagen oder Entscheidungen zu treffen.
- Dazu gehört die *Optimierung von Parametern* und Beziehungen innerhalb von Daten, um Modelle zu erstellen.
- *Grundlegendes Beispiel:* Lineare Regression, bei der Parameter an Datenpunkte angepasst werden.



- **Neurale Netze (NN):**

- Grundlegend für ML.
- Bestehen aus *miteinander verbundenen Knoten* (Neuronen).
- Verarbeiten Eingaben, um Ausgaben zu erzeugen.
- Funktionieren mittels *Multiplikation von Eingangsvektoren mit Gewichtsmatrizen* und Anwendung von *Aktivierungsfunktionen* (z.B. *ReLU* oder *Sigmoid*).
- Trotz ihrer Einfachheit sind diese Netze leistungsfähig genug für eine breite Palette an Aufgaben.

- **Deep Learning (DL):**

- Eine *Untergruppe von ML*.
- Verwendet Neurale Netze mit *vielen Parametern*, um komplexe Probleme zu lösen.
- Eignet sich hervorragend für Aufgaben wie:
  - *Bildklassifizierung* (Bilder in Kategorien sortieren).
  - *Segmentierung* (verschiedene Teile eines Bildes identifizieren).



- *Stilübertragungen* (Stil von Bildern ändern, Inhalt beibehalten).
- Synthesierung neuartiger Ansichten (neue 3D-Perspektiven aus begrenzten Daten).
- **Klassifizierer:**
  - Eine Anwendung des maschinellen Lernens.
  - Bei *hochdimensionalen Eingabedaten* (z.B. Bilder) werden diese in eine *niedrigdimensionale Ausgabe* (i.d.R. eine *Klassenbezeichnung*) umgewandelt.
  - Klassenbezeichnungen können zum Beispiel Kategorien wie „*Mensch*“, „*Hund*“ oder „*Vogel*“ sein.
  - Das Trapezsymbol in Klassifizierern steht für diese *Dimensionsreduktion* und zeigt, wie Rohdaten durch den Klassifizierungsprozess zu bestimmten Kategorien verdichtet werden.

## Generative Modelle

---

[EVC\\_Skriptum\(CG\).pdf](#), p.59

- **Generative Modelle (GM):**
  - Eine Untergruppe des Deep Learning (DL).
  - Klasse von *statistischen Modellen*.
  - Dienen zur *Erzeugung neuer Dateninstanzen* (z.B. Bilder, Texte), die der ursprünglichen Trainingsverteilung ähneln.
  - Lernen die *Wahrscheinlichkeitsverteilung der Daten*, auf denen sie trainiert wurden.
  - Ermöglichen die Erzeugung neuer Datenpunkte mit *ähnlichen Merkmalen*.

**Wichtige Arten von generativen Modellen:**

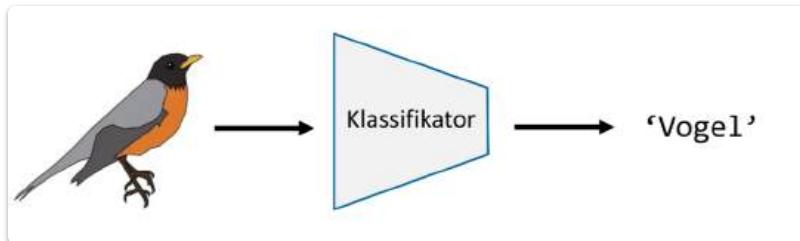
- **Auto-Encoder:**
  - Lernen, Eingabedaten in eine *kleinere Darstellung zu komprimieren*.
  - Rekonstruieren dann die Ausgabe anhand dieser Darstellung (dienen primär der Datenkompression und Merkmalsextraktion, können aber auch generativ eingesetzt werden).
- **Generative Adversarial Networks (GANs):**
  - Bestehen aus einem *Generator*, der Stichproben erzeugt.
  - Bestehen aus einem *Diskriminator*, der diese auf Echtheit auswertet (ein Wettbewerb zwischen Generator und Diskriminatator führt zu immer realistischeren Generierungen).

- **Diffusionsmodelle:**

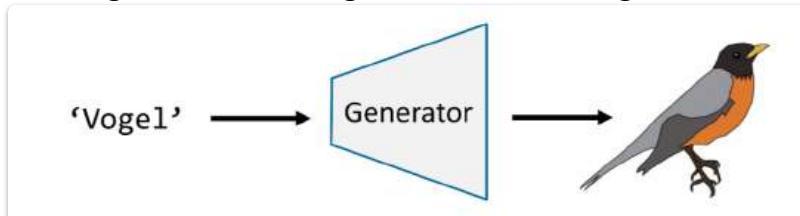
- Nutzen die Idee der *umgekehrten Diffusion* (d.h. Rauschentfernung).
- Erzeugen iterativ eine Ausgabe aus zufälligem Rauschen (beginnen mit Rauschen und entfernen es schrittweise, um ein klares Bild zu erhalten).

### Konzeptionelle Vorstellung:

- **Klassifikator (rückwärts):** Ein Klassifikator würde eine hochdimensionale Eingabe (z.B. RGB-Bild) in eine niedrigdimensionale Ausgabe (eine Klassenbezeichnung wie 'Vogel') umwandeln.



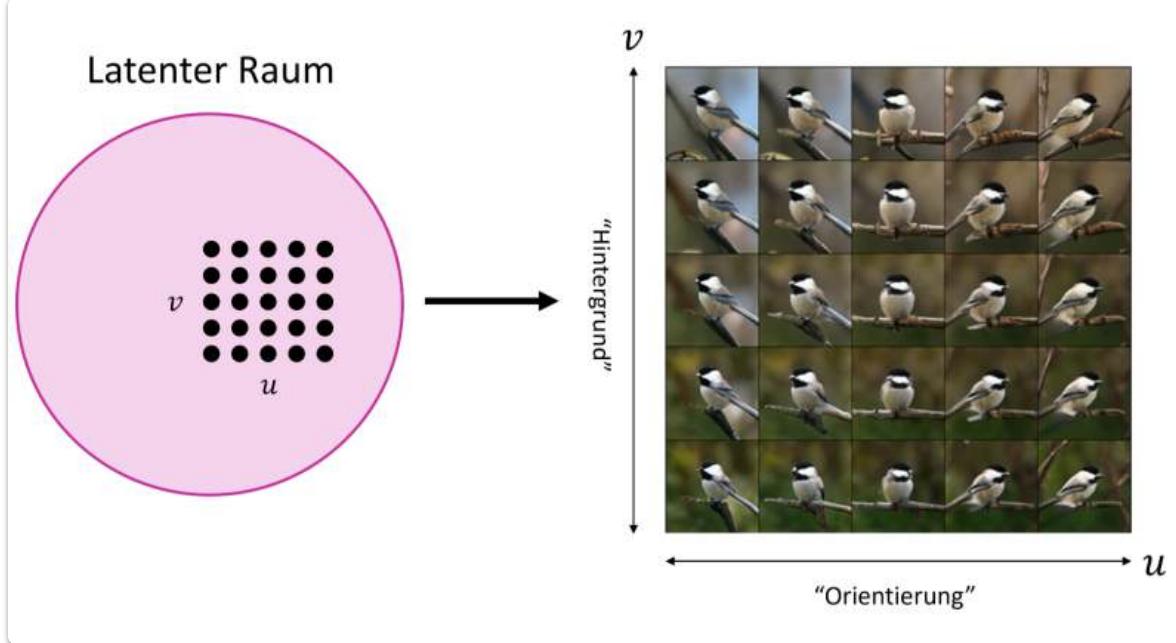
- **Generatives Modell:** Nimmt eine Eingabe mit *geringer Dimensionalität* (z.B. ein Texteingabe oder zufälliges Rauschen) und gibt ein *RGB-Bild* aus.



- Wenn man verschiedene Instanzen von Vögeln generieren und steuern möchte, welche Instanzen erzeugt werden, sind *zusätzliche Eingaben* in den Generator notwendig.
- Um *Variabilität* zu erreichen, wird oft *zufälliges Rauschen* verwendet. Dieses Rauschen dient dazu, eine Verteilung von zufälligen Eingaben auf eine gewünschte Verteilung von Ausgaben abzubilden.

## Latenter Raum

EVC\_Skriptum\_CG, p.60



- Innerhalb des Modells sind die Daten in einem *hochdimensionalen, latenten Raum* enthalten.
- Dieser Raum umfasst Punkte, die durch *latente Variablen* definiert sind.
  - Diese Variablen kodieren die *Merkmale* unseres Vogels wie Farbe und Größe.
- Anfänglich sind diese Variablen *zufällig verteilt*.
- Um bestimmte Attribute der erzeugten Ausgabe besser zu kontrollieren, werden sie *bewusst manipuliert*.
- In einer zweidimensionalen Darstellung dieses Raums könnte beispielsweise:
  - Die Koordinate ' $u$ ' die *Ausrichtung* des Vogels (links, vorne, rechts) vorgeben.
  - Die Koordinate ' $v$ ' den *Hintergrundkontext* (blauer Himmel, grüne Blätter) bestimmen.
- **Variable Verschränkung (variable entanglement):** Latente Variablen sind jedoch nicht immer klar voneinander getrennt. Änderungen in einer Variablen können *unbeabsichtigt auf andere auswirken*.
  - *Beispiel:* Ein nach rechts gerichteter Vogel könnte einen braunen Hintergrund erscheinen lassen.

## Generative Modelle für Bilderstellung

[EVC\\_Skriptum\\_CG, p.60](#)

- **Notwendigkeit zur Bilderstellung mit generativen Modellen:**
  - **Architektur-Auswahl:** Der erste Schritt ist die Auswahl einer Architektur, z.B. *GANs* oder *Diffusionsmodelle*.
  - **Training:**
    - Verwendet einen *großen Datensatz von Bildern*.
    - Manchmal auch mit *Beschriftungen* (Labels).
    - Ziel: Dem Modell beibringen, *neue Bilder akkurat zu erzeugen*.

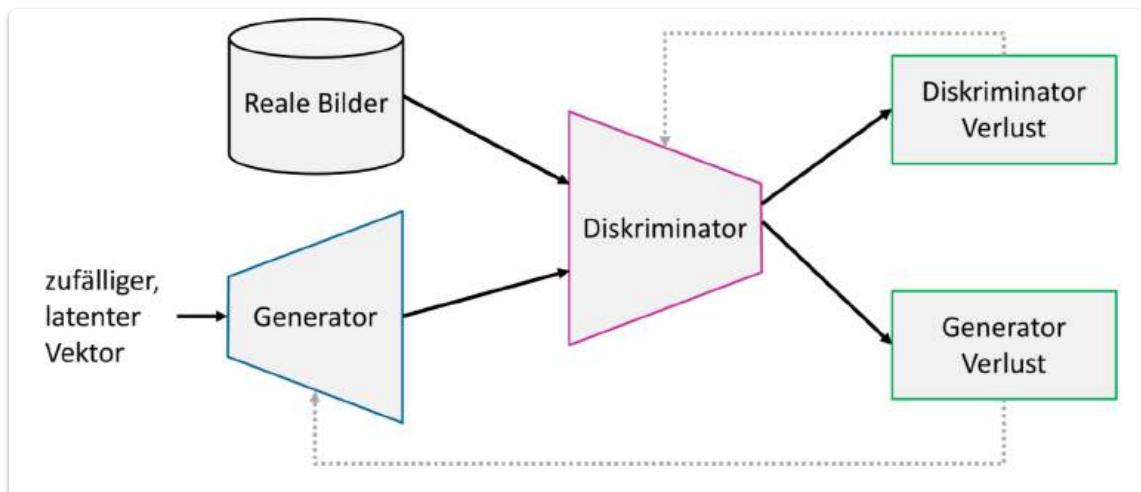
- **Inferenz:**

- Das Modell wendet das *Gelernte an*.
  - Erzeugt *neue Bilder* auf der Grundlage neuer Eingaben.
  - Ermöglicht die *kontrollierte Erzeugung spezifischer Merkmale* in der Ausgabe, wie Farbe und Größe.
- 

## Training eines Generative Adversarial Network (GAN)

EVC\_Skriptum\_CG, p.60

- Bei GANs liefern sich *zwei neuronale Netze*, der **Generator** und der **Diskriminator**, einen **Wettstreit** (Adversarial Process).
- **Aufgabe des Generators:**
  - Erzeugt Bilder aus *zufälligen, latenten Vektoren*.
  - Ziel: Die erzeugten Bilder sollen von realen Bildern *nicht zu unterscheiden* sein, um den Diskriminator zu "täuschen".
- **Aufgabe des Diskriminators:**
  - Unterscheidet zwischen den *erzeugten* Bildern (Fakes) und den *realen* Bildern.
- **Verlustfunktionen:**
  - Der Prozess beinhaltet *zwei Verlustfunktionen* (Loss Functions), die *gleichzeitig optimiert* werden.
  - Ziel:
    - Verbesserung der Fähigkeit des Generators zur Täuschung.
    - Verbesserung der Fähigkeit des Diskriminators zur Erkennung von Fälschungen.
  - **Entscheidend:** Die Verlustfunktionen müssen *differenzierbar* sein, um die *Backpropagation* (Anpassung der Gewichte im neuronalen Netz basierend auf dem Fehler) während des Trainings zu ermöglichen.
  - Dies verbessert die Lern- und Anpassungsfähigkeiten beider Netzwerke.



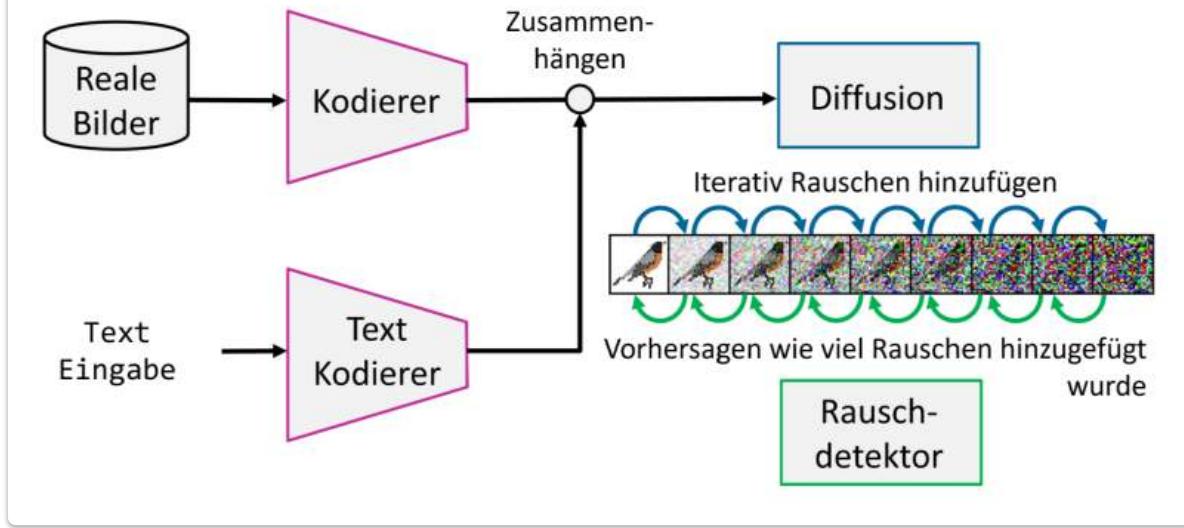
- **Reale Bilder** und **Zufälliger, latenter Vektor** sind die Eingaben.

- Der **Generator** erzeugt Bilder aus dem latenten Vektor.
  - Der **Diskriminator** erhält sowohl die realen Bilder als auch die vom Generator erzeugten Bilder.
  - Basierend auf der Ausgabe des Diskriminators werden der **Diskriminator Verlust** und der **Generator Verlust** berechnet.
- 

## Training eines Diffusionsmodells

EVC\_Skriptum\_CG, p.60

- Diffusionsmodelle werden trainiert, indem Bilder und Textaufforderungen (Prompts) in einem **gemeinsamen latenten Raum** kodiert werden.
- **Rauschhervorhebung:**
  - In den Bildern wird *iterativ Rauschen hinzugefügt*.
  - Typischerweise wird *gaußsches Rauschen* verwendet, aufgrund seiner probabilistischen Eigenschaften.
- **Lernen aus "verrauschten" Daten:**
  - Das hinzugefügte Rauschen ermöglicht es dem Modell, aus diesen verrauschten Daten zu lernen.
  - Dies geschieht mithilfe eines **Rauschdetektors**.
  - Der Rauschdetektor *sagt den Grad des Rauschens bei jedem Schritt vorher*.
- **Aufgabe des Rauschentferrners (Denoising):**
  - Das Rauschen soll anhand des *verrauschten Bildes* und der *zugehörigen Texteingabe* vorhergesagt werden.
  - Anschließend wird das vorhergesagte Rauschen *subtrahiert*.
- **Lernprozess und Ergebnis:**
  - Dieser Lernprozess stellt sicher, dass sich das Bild mit jeder Iteration einer *weniger verrauschten und genaueren Darstellung* annähert.



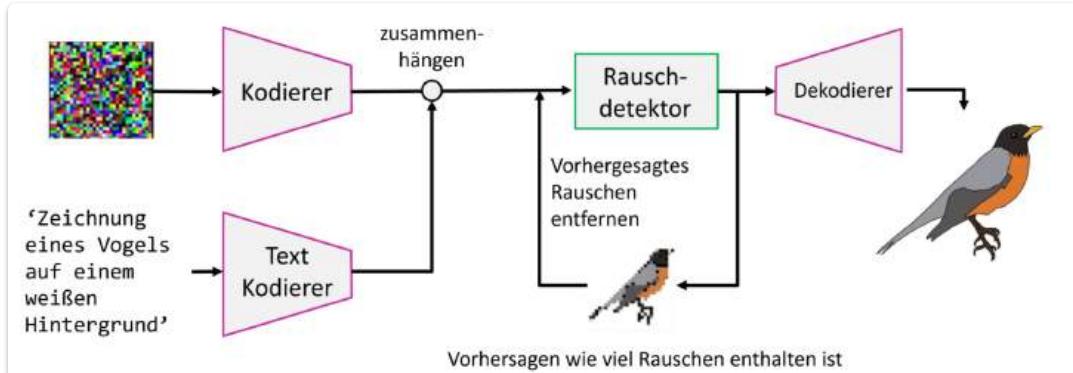
- **Reale Bilder** werden von einem **Kodierer** verarbeitet.
- **Text Eingabe** wird von einem **Text Kodierer** verarbeitet.
- Die Ausgaben beider Kodierer werden zusammengeführt ("Zusammenhängen").
- Dies fließt in die **Diffusion**, wo *iterativ Rauschen hinzugefügt* wird.
- Ein Beispielbild zeigt eine Reihe von immer stärker verrauschten Bildern.
- Der **Rauschdetektor** erhält die verrauschten Bilder und sagt vorher, wie viel Rauschen hinzugefügt wurde.

## Inferenz in einem Diffusionsmodell

EVC\_Skriptum\_CG, p.61

- **Generierungsprozess (Inferenzphase):**
  - Beginnt mit einer **Textaufforderung** und einem Ausschnitt bestehend aus **zufälligem, gaußschem Rauschen**.
- **Kodierung und Verfeinerung:**
  - Beide (Textaufforderung und Rauschen) werden in den **latenten Raum** kodiert.
  - Sie werden durch eine Reihe von **Rauscherkennungs- und Subtraktionsschritten** kontinuierlich verfeinert.
- **Rauschdetektor:**
  - Bei jeder Iteration schätzt der **Rauschdetektor** die **Menge des vorhandenen Rauschens**.
  - Dieses Rauschen wird dann **subtrahiert**, um das Bild allmählich zu klären (denoising).
- **Entrauschungsprozess:**
  - Diese Abfolge wird so lange fortgesetzt, bis das Bild **ausreichend entrauscht** ist.
- **Dekodierung zum visuellen Bild:**

- Die resultierende latente Darstellung wird von einem **Dekodierer** wieder in ein *visuelles Bild dekodiert*.
- Der Dekodierer ist in der Regel als "Variational Auto-Encoder" (VAE) strukturiert.
- **Ergebnis und Fähigkeit:**
  - Die resultierenden 2D-Bilder, die aus verallgemeinerten Rausch- und Texteingaben generiert wurden, demonstrieren die Fähigkeit des Modells, *detaillierte und kontextgerechte Bilder aus einem hochdimensionalen latenten Raum zu erzeugen*.
- **Anwendung:**
  - Nun kann ein Werkzeug zur Generierung von *2D-Bildern anhand von Texteingaben* genutzt werden.
  - Für die Erstellung von *3D-Modellen und -Szenen* wird jedoch mehr benötigt.



- Ein Rauschbild (z.B. 'Zeichnung eines Vogels auf einem weißen Hintergrund') wird von einem **Kodierer** verarbeitet.
- Eine **Texteingabe** wird von einem **Text Kodierer** verarbeitet.
- Beide Ausgaben werden zusammengeführt ("zusammenhängen"). \* Dies fließt in den **Rauschdetektor**, der das "Vorhergesagtes Rauschen entfernen" signalisiert und auch "Vorhersagen wie viel Rauschen enthalten ist".
- Anschließend geht es in den **Dekodierer**, der das endgültige, entrauschte Bild (z.B. einen Vogel) ausgibt.

## Szenendarstellungen für Machine 3D-Learning

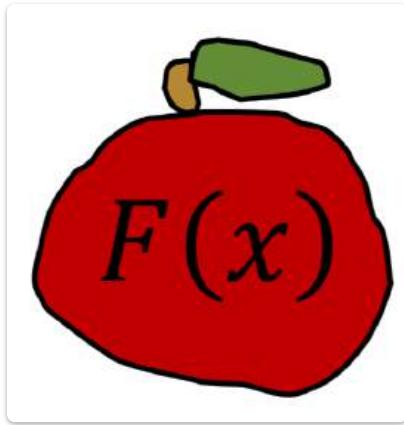
[EVC\\_Skriptum\\_CG, p.61](#)

- Ein reales Objekt kann nicht *detaillgenau* dargestellt werden, um jedes einzelne Detail zu erfassen.
- Es gibt verschiedene Darstellungen, die jeweils *Vor- und Nachteile* haben.
- Sie können in zwei Kategorien unterteilt werden: **implizit** und **explizit**.

### Implizite Darstellungen

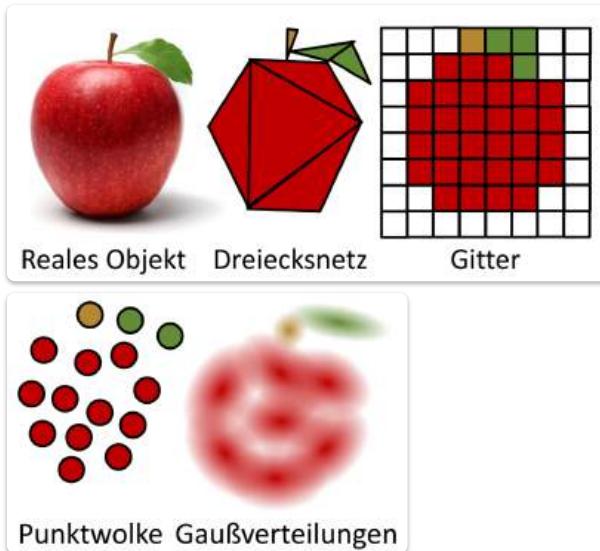
- Sind **Funktionen**, die für jeden Punkt im Raum einen Wert definieren.
- Beispiele:
  - Ob das Objekt vorhanden ist oder nicht.

- Wie weit die nächstgelegene Oberfläche entfernt ist (z.B. **Signed Distance Functions (SDFs)**).



## Explizite Darstellungen

- Sind in der Regel **Mengen von diskreten Elementen**.
- Beispiele:
  - **Dreiecksnetz** (Mesh)
  - **Gitter** (Grid)
  - **Punktwolke** (Point Cloud)



## Herausforderungen und Überlegungen

- **Beispiel Punktwolken:**
  - Beim Scannen eines realen Objekts erhält man einen *endlichen Satz von Punkten*.
  - Es gibt *keinen einfachen Weg*, um sicherzustellen, dass beim Rendern der Punkte *keine Löcher* bleiben, die verdecken, was sich hinter dem Objekt befindet.
  - Man könnte versuchen, die Löcher zu "schließen" und ein Dreiecksnetz zu konstruieren.
- **Training mit Dreiecksnetzen:**
  - Wichtig ist sicherzustellen, dass das Netz *nicht deformiert und ungültig* wird.
  - Dies kann passieren, wenn sich *Vertices (Eckpunkte)* frei bewegen können und sich dadurch *Flächen überschneiden*.

- **Fazit:** Es ist immer wichtig zu überlegen, welche Darstellung für die jeweilige Aufgabenstellung am *besten geeignet* ist.

## Neural Radiance Fields (NeRFs)

[EVC\\_Skriptum\\_CG](#), p.61, p.62

- NeRFs bieten eine *implizite Darstellung räumlicher Daten*.
- Sie nutzen die Gewichte innerhalb eines neuronalen Netzes zur Modellierung von 3D-Umgebungen.
- Dieser Prozess wird dem **Deep Learning** zugeordnet und erfordert *erhebliche Rechenressourcen*.

### Kernmechanismus eines NeRF

- Umfasst die **Abtastung entlang eines Strahls im 3D-Raum** (Ray Casting/Sampling).
- Diese Methode wird von einem **mehrschichtigen Perzepron** (engl. *multi-layer perceptron, MLP*) gesteuert.
- **Prozess:**
  1. Der Strahl durchquert die Szene.
  2. Das Modell bewertet die *Farbe und Dichte* an verschiedenen Punkten entlang des Pfades.
  3. Diese Werte werden *projiziert und gemischt*, um das endgültige Bild zu erzeugen.
  4. Durch diese Projektion werden die **volumetrischen Daten** (3D-Informationen im Raum) zu einem *einzelnen Farbwert pro abgetastetem Strahl* verdichtet.
  5. Dies resultiert in einer *sehr detaillierten Wiedergabe der Szene*.

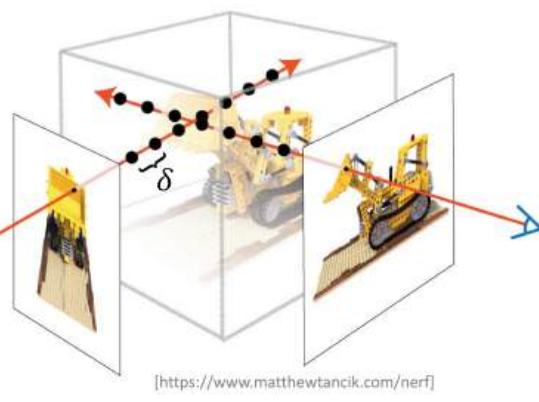
■ Each point in space contains:

- (View dependent) color –  $c$
- Density (transparency) –  $\sigma$

■ Use *raymarching* to render it

$$C(r) = \sum_{i=1}^N T_i (1 - \exp(-\sigma_i \delta_i)) c_i$$

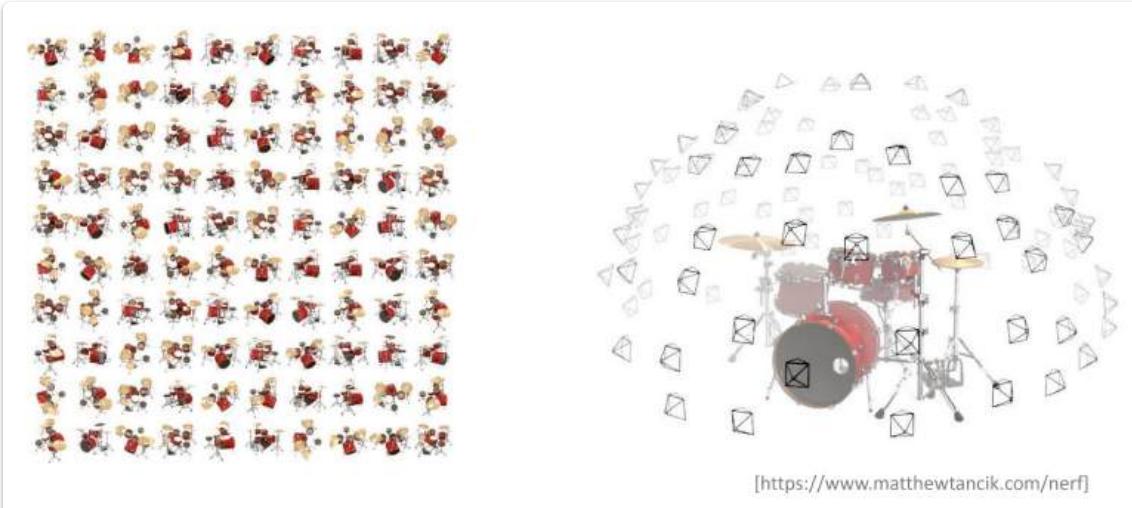
$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$



### Rekonstruktion einer realen Szene mit NeRFs

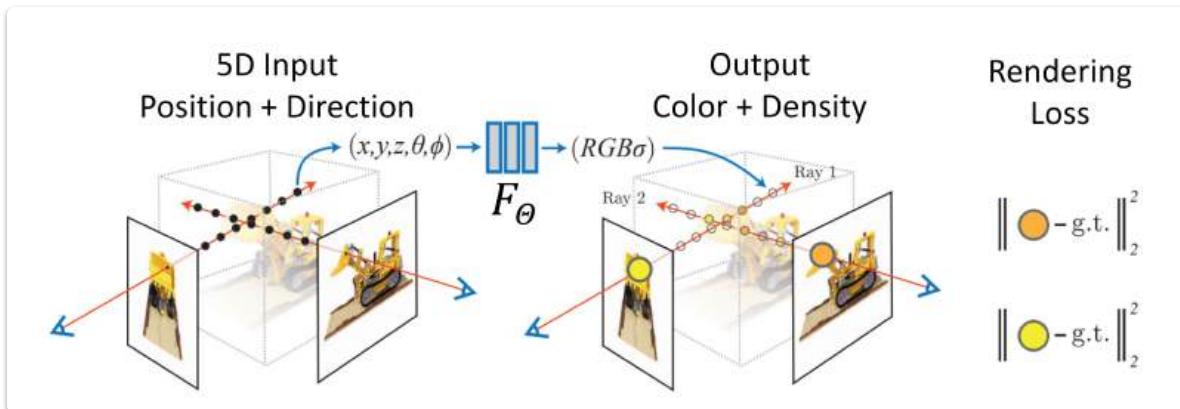
- **Basis:** Es werden *echte Fotografien* als Basis benötigt.
- **Kamerarekonstruktion:** Der *Structure-from-Motion-Algorithmus* (SfM) kann verwendet werden, um die entsprechenden Kamerapositionen zu erhalten, aus denen dann die Szene gerendert wird.

- **Optimierung:** Das gerenderte Bild wird mit dem entsprechenden Basisbild verglichen und das Netzwerk so optimiert, dass es mit diesen Bildern übereinstimmt.

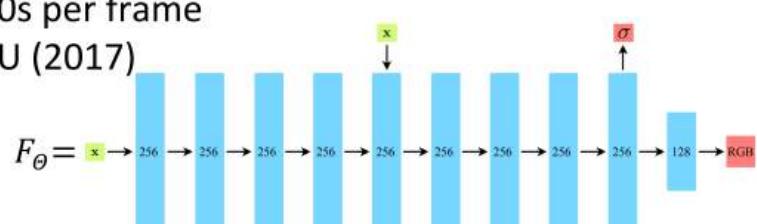


## Herausforderungen und Flexibilität

- NeRFs basieren auf neuronalen Netzen, die **Blackboxen** sind.
- Das bedeutet, es ist sehr schwierig, einen Teil einer Szene **direkt zu ändern** (z.B. ein Objekt zu entfernen).
- Es kann notwendig sein, die Szene vor der Bearbeitung zunächst in eine **explizite Darstellung** (z.B. ein Dreiecksnetz) umzuwandeln.
- Dies kann unter Verwendung des **Marching-Cubes-Algorithmus** geschehen.
- **Fazit:** NeRFs haben sich als **sehr leistungsfähig** erwiesen und werden heutzutage **häufig verwendet**.

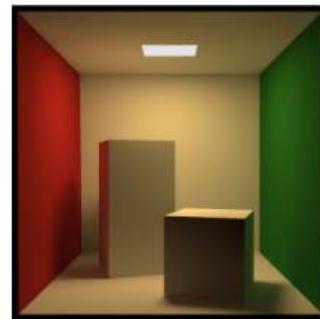


- Approx. 590 000 weights  $\approx 3 \text{ MB}$
- 1-2 days to train and 30s per frame on an NVIDIA V100 GPU (2017)



- Remember that an explicit  $1024^3$  voxel grid requires **512 GB**

- The weights of a network represent its memory
- Given a network  $F_\theta$ , the weights of  $F_\theta$  may be underutilized or overutilized – not a lot of control
- Do we need a large NN to represent a simple scene?



- Is our NN large enough to capture all details?

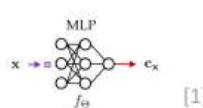


## Gaussian Splatting (GS)

[EVC\\_Skriptum\\_CG, p.62](#)

### Neural Radiance Fields

- Implicit



- Deep Learning



- Ray Marching



### Gaussian Splatting

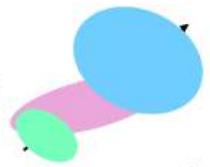
- Explicit



- Simple ML

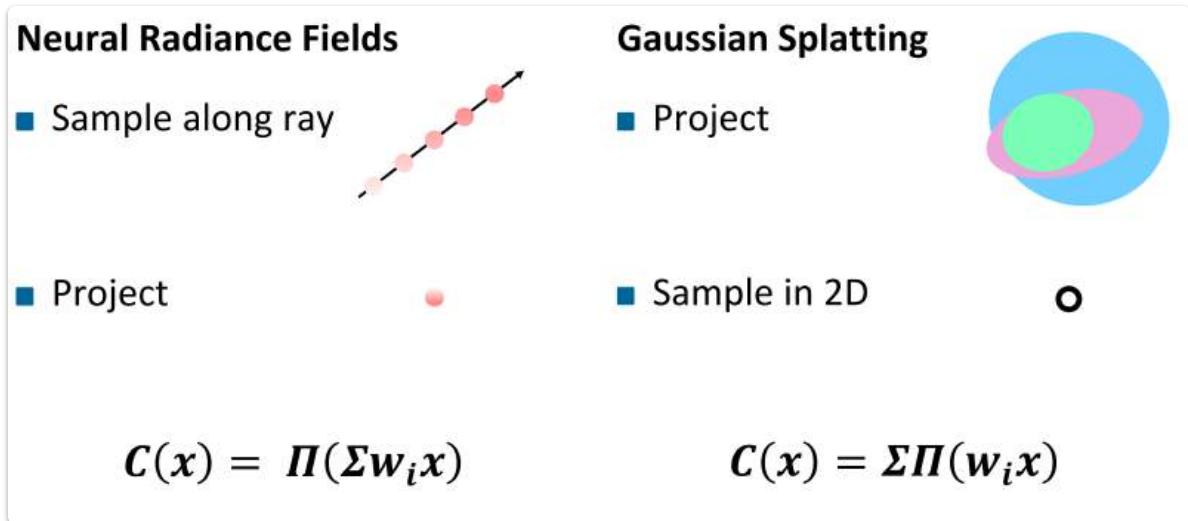


- Blending Gaussians



- Im Gegensatz zu NeRFs ist Gaussian Splatting (GS) eine **explizite Technik** zur Szenendarstellung.
- Es nutzt die Prinzipien des **Volumen Renderings**.

- Anstatt neuronale Netze als Blackboxen zu verwenden, betrachtet GS **Gaußsche Funktionen als primitive Formen**.



## Definition einer Gaußglocke

- Jede Gaußglocke wird durch ihren **Mittelwert (Position)**, ihre **Kovarianzmatrix (Form)**, ihre **Deckkraft** und **Farbe** definiert.
- Dies ermöglicht eine *einfachere Analyse der maschinellen Lernverfahren* (z.B. stochastische Gradientenverfahren), um die *räumlichen Merkmale der Szene zu lernen*.

## Normal distribution in 3D:

$$\mathbf{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \mathbf{X} = X_1, X_2, X_3$$

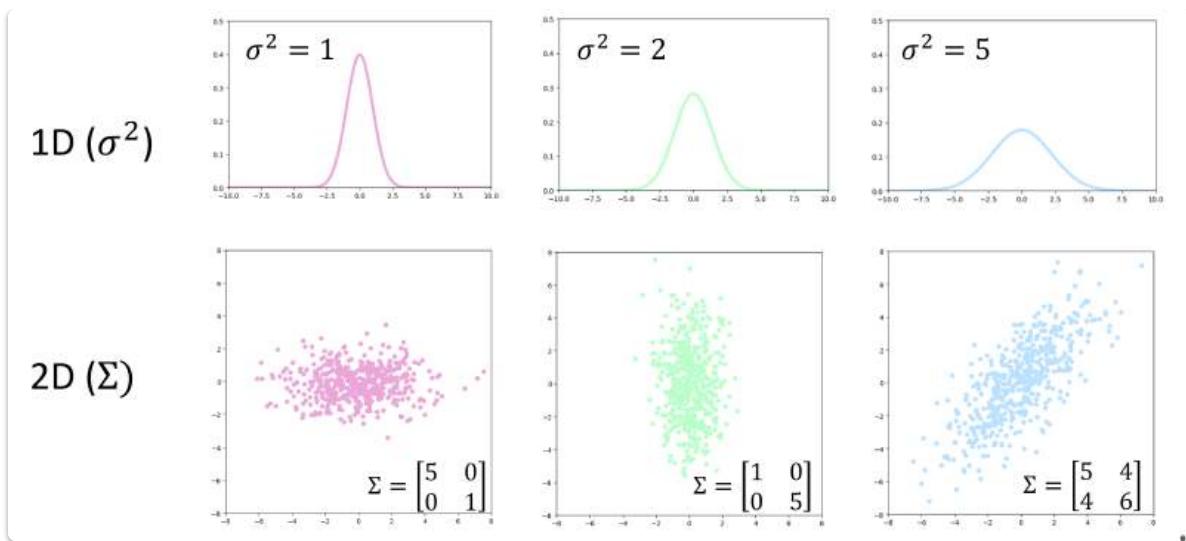
- Position  $\mathbf{X}$ , where  $\mathbf{X}$  is the vector  $\boldsymbol{\mu} - \mathbf{x}$  ( $\mathbf{x}$  ... query point)
- Covariance matrix  $\boldsymbol{\Sigma}$  (stretching/scaling)

$$G(\mathbf{X}) = e^{-\frac{1}{2}(\mathbf{X})^T \boldsymbol{\Sigma}^{-1} (\mathbf{X})}$$

- Additionally: Color (RGB → Spherical Harmonics) + transparency ( $\alpha$ )

## Rendering-Prozess bei GS

- Umfasst die **Projektion der 3D-Gaußglocke auf eine 2D-Ebene**.
- Anschließend erfolgt die **Rasterung** der projizierten Daten, um ein Bild zu generieren.
- Der Beitrag jeder Gaußglocke zum endgültigen Bild wird im 2D-Raum *sortiert und abgetastet*.
- Dies ermöglicht ein *effizientes Rendering* ohne den hohen Auswertungsaufwand von Deep-Learning-Architekturen.



## Training von Gaussian Splatting

- **Initialisierung:** Das Training beginnt mit einer *anfänglichen Menge an Gaußglocken*.
- **Datengewinnung:** Diese Gaußglocken können mit dem **Structure-from-Motion-Algorithmus** (SfM) gewonnen werden.
  - SfM gibt einen Satz von *3D-Punkten* aus, die mit den realen Basisbildern übereinstimmen.
- **Anpassung der Gaußglocken:**
  - Anschließend werden diese Punkte direkt in *Gauß'sche Punkte umgewandelt*.
  - Ihre *Kovarianzmatrizen* werden so eingestellt, dass sie und ihre Nachbarn *Flächen ohne Löcher* bilden.
  - Idealerweise sollten sie eine *erhebliche Überlappung* bilden.
- **Detaillierung und Gradienten:**
  - Falls der anfängliche Satz an Gaußglocken nicht ausreicht, um alle Details in der Szene darzustellen, müssen diese entweder *geklont* oder *aufgeteilt* werden.
  - Dies kann über die *akkumulierte Größe ihrer Positionsgradienten* bestimmt werden.
  - Ein großer Gradient bedeutet, dass sich die Gaußglocke *schnell irgendwohin bewegen muss* oder dass sie an *mehreren verschiedenen Orten gleichzeitig befindlich* sein muss.
  - In diesen Fällen wird die Gaußglocke geklont oder aufgeteilt.
- **Ergebnis:** Auf diese Weise wird eine *qualitativ hochwertige 3D-Darstellung* mit mehr Gaußglocken und somit mehr Details erzielt, wo sie benötigt werden.

1 million Gaussians:



1 million Gaussians:  
(as fully opaque ellipsoids)

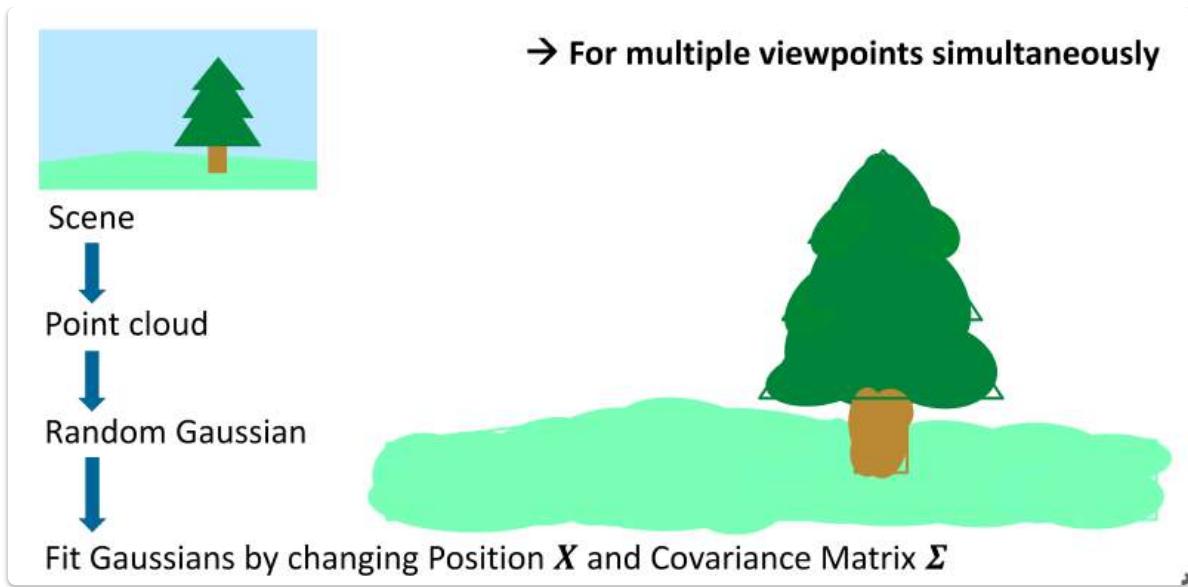


## Generative Modelle für 3D Objects

[EVC\\_Skriptum\\_CG](#), p.62

- Bisher wurde die Erzeugung von 2D-Bildern und die Rekonstruktion realer Szenen behandelt.
- **Kernfrage:** Wie können diese beiden Verfahren kombiniert werden?
- **Ansatz:**
  - Der Prozess des *iterativen Änderns/Entrauschens von Bildern* (wie bei Diffusionsmodellen) kann genutzt werden, um neue Bilder zu erzeugen.
  - Gleichzeitig soll eine **3D-Objektdarstellung iterativ geändert** werden.
- **Ziel:** Es wird beschrieben, wie ein **Diffusionsmodell** und **3D Gaussian Splatting** kombiniert werden, um *neue 3D-Objekte zu synthetisieren*.

### Ablauf der 3D-Objekt-Erstellung



### 1. Texteingabe als Startpunkt:

- Der Prozess beginnt mit einer **Texteingabe** (dem "Prompt"), die beschreibt, welches 3D-Objekt erstellt werden soll.

### 2. Initialisierung der Gaußschen Punkte:

- Ein erster Satz von **Gaußschen Punkten** wird erzeugt. Diese dienen als grundlegende Darstellung des 3D-Objekts.
- Die Erstellung kann **manuell** erfolgen.
- Alternativ kann ein **vortrainiertes Netzwerk** verwendet werden, das basierend auf der Texteingabe eine (anfänglich oft grobe und nicht optimierte) Punktwolke generiert.
- Diese Punkte werden anschließend mit einem Verfahren, das dem der Szenenrekonstruktion ähnelt, in **Gaußglocken** umgewandelt.

### 3. Rendering und Rauschhinzufügung:

- Die Gaußglocken werden aus **zufälligen Blickwinkeln** gerendert. Dieser Rendering-Prozess wird auch als "Splatting Renderer" bezeichnet.
- Die Ergebnisse sind **2D-Bilder** (Renderings).
- Da **Diffusionsmodelle** mit verrauschten Bildern arbeiten, wird diesen 2D-Renderings **künstlich Rauschen hinzugefügt**.

### 4. Anpassung durch Diffusionsmodell:

- Ein **Diffusionsmodell** analysiert die verrauschten 2D-Renderings.
- Es lernt, wie diese Bilder verändert werden müssen, um dem ursprünglichen Text-Prompt zu entsprechen (z.B. indem es Rauschen entfernt und Details hinzufügt, die zum Prompt passen).
- Basierend auf den vom Diffusionsmodell vorgeschlagenen Änderungen der 2D-Bilder wird die **zugrunde liegende 3D-Objektdarstellung** (die Gaußglocken selbst) entsprechend angepasst.

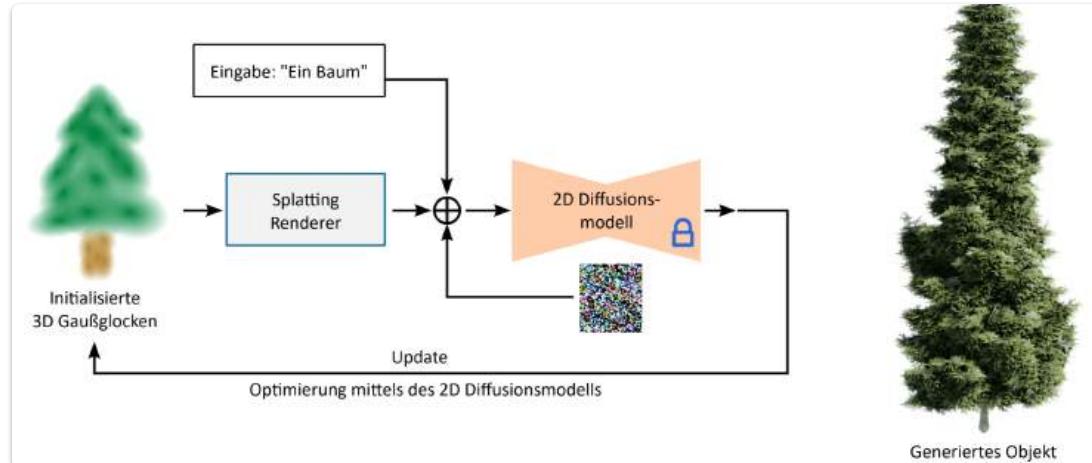
### 5. Verfeinerung und Detailverbesserung:

- Um die Detailtiefe des 3D-Objekts zu erhöhen, werden die Gaußglocken **aufgeteilt und geklont**.

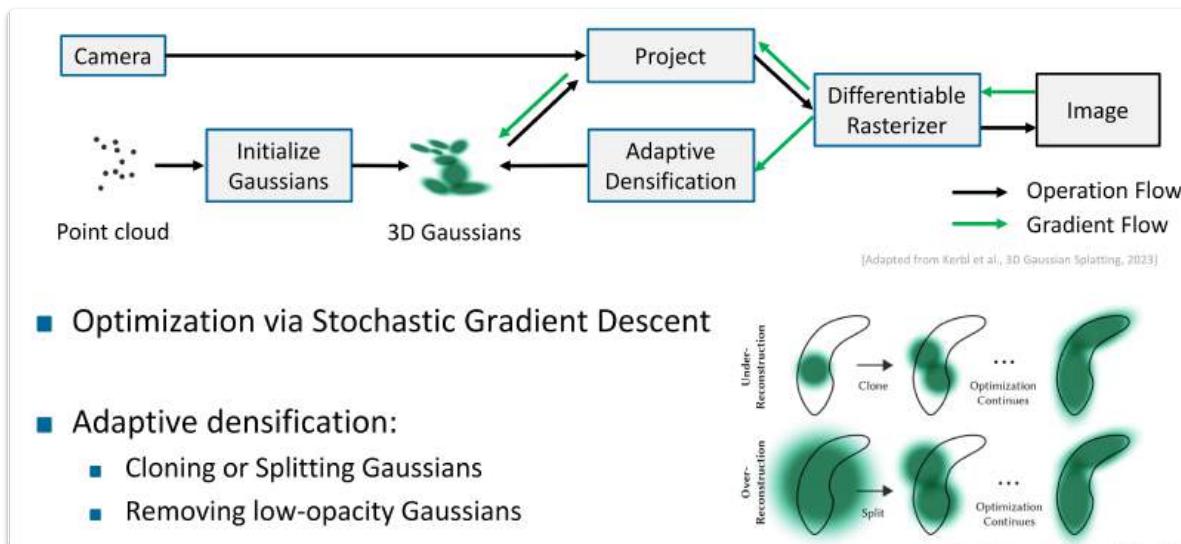
- Dieser Schritt erfolgt unter Berücksichtigung der **Größe ihres Gradienten** (ein Maß für die lokale Veränderung der Eigenschaften der Gaußglocke, was auf Bereiche hinweist, die mehr Details benötigen).

## 6. Automatisierte Erstellung:

- Nach **mehreren Hundert bis Tausend Iterationen** dieses Zyklus aus Rendering, Diffusionsmodell-Anpassung und Verfeinerung wird das neue 3D-Objekt **automatisch erstellt**.



## Training im Detail:



- Optimization via Stochastic Gradient Descent
- Adaptive densification:
  - Cloning or Splitting Gaussians
  - Removing low-opacity Gaussians