

2. Test AI-Zusammenfassung

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 auf der des Skriptums.

Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das vollständige Kapitel der Stoffsammlung mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Inhalt

Computergrafik

- 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

Computervision

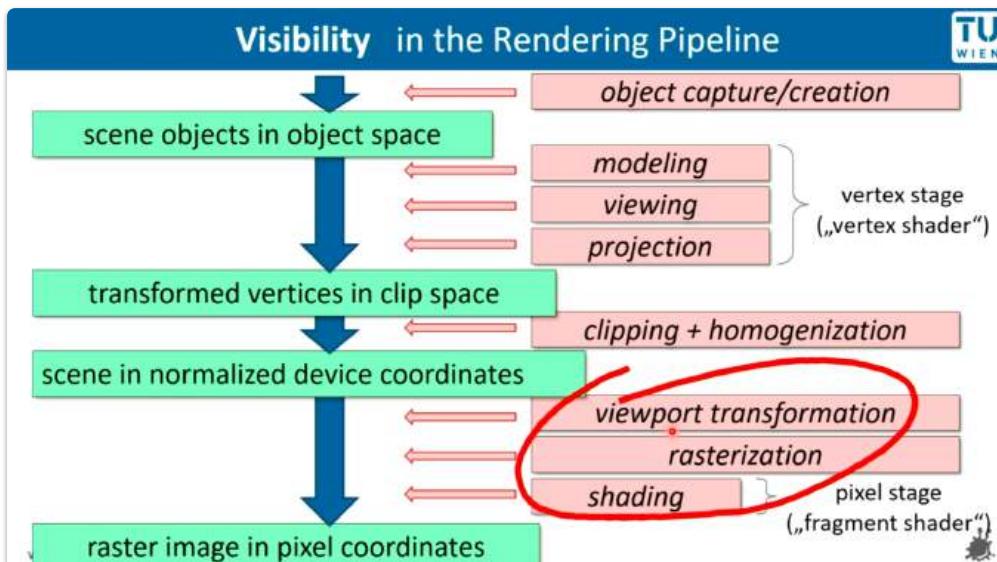
- 9. Multiskalenrepräsentation_AI
- 10. Stereo und Motion_AI
- 11. Deep Learning for Computer Vision_AI
- 12. Computational Photography_AI

8. Sichtbarkeitsverfahren

⚠ Disclaimer

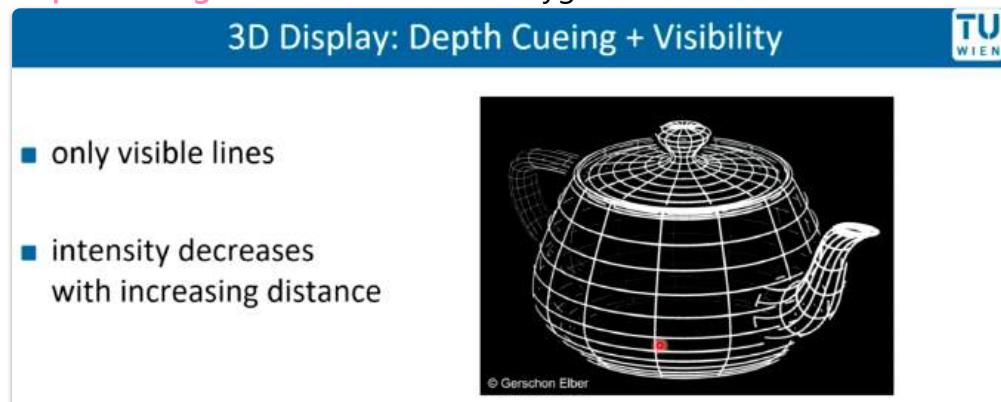
Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Sichtbarkeitsverfahren sind essenziell in der Rendering Pipeline, um Szenen korrekt darzustellen, indem unsichtbare Teile von Objekten (Rückseiten, verdeckte Bereiche) entfernt werden. Dies wird auch als Hidden-Line- oder Hidden-Surface Eliminierung bezeichnet.



Ansätze

- **Objektraum-Methoden**: Vergleichen Objekte direkt miteinander, um nur sichtbare Teile zu zeichnen.
- **Bildraum-Methoden**: Berechnen pixelweise, was an jeder Stelle sichtbar ist.
- **Anmerkung**: Die meisten Verfahren berücksichtigen keine transparenten Objekte.
- **Depth Cueing**: Macht unsichtbare Polygone anders sichtbar.



Backface Detection (Backface Culling)

Backface Culling ist eine Optimierung zur Reduzierung des Rechenaufwands, indem Polygone eliminiert werden, die sicher nicht sichtbar sind (deren **Oberflächennormale vom Betrachter weg zeigt**). Es ist kein vollständiges Sichtbarkeitsverfahren, reduziert aber im Schnitt etwa 50% der Polygone.

Berechnungsverfahren

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

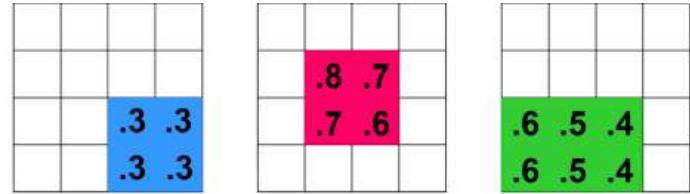
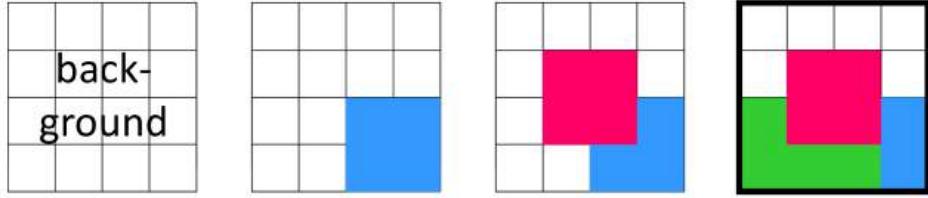
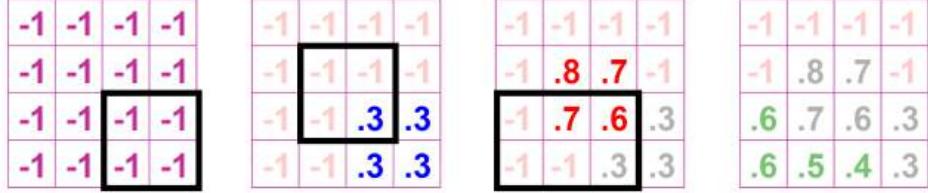
Depth Buffer Verfahren (Z-Puffer Verfahren)

Löst das Sichtbarkeitsproblem pixelweise.

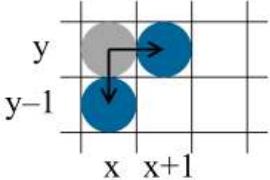
- **Kernidee:** Ein **Z-Puffer** speichert für jedes Pixel die Tiefeninformation (z-Koordinate) des bisher gezeichneten Objekts, zusätzlich zum Framebuffer (Farbinformation).
- **Ablauf:**

```

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 nichts! (bereits verdecktes Objekt bleibt verdeckt)
  
```

polygons with corresponding z-values 
image 
depth-buffer 

- **Vorteil:** Keine Sortierung der Objekte notwendig.
- **Effizienz:** Z-Werte können für ebene Polygone inkrementell berechnet werden.



depth at (x,y) :

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

$$z = \frac{-Ax-By-D}{C}$$

constants !

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

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

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

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

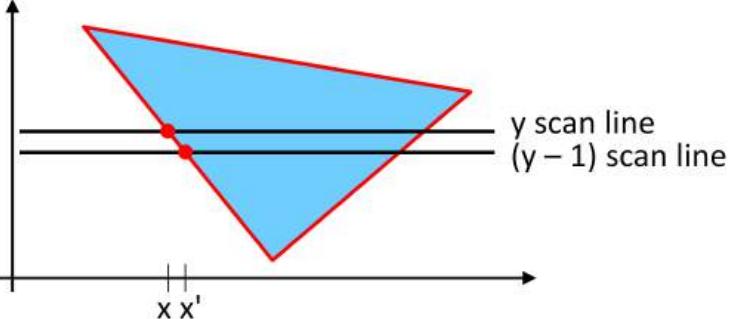
$$y' = y - 1$$

$$x' = x - 1/m$$

$$\Rightarrow z' = \frac{-A(x-1/m)-B(y-1)-D}{C}$$

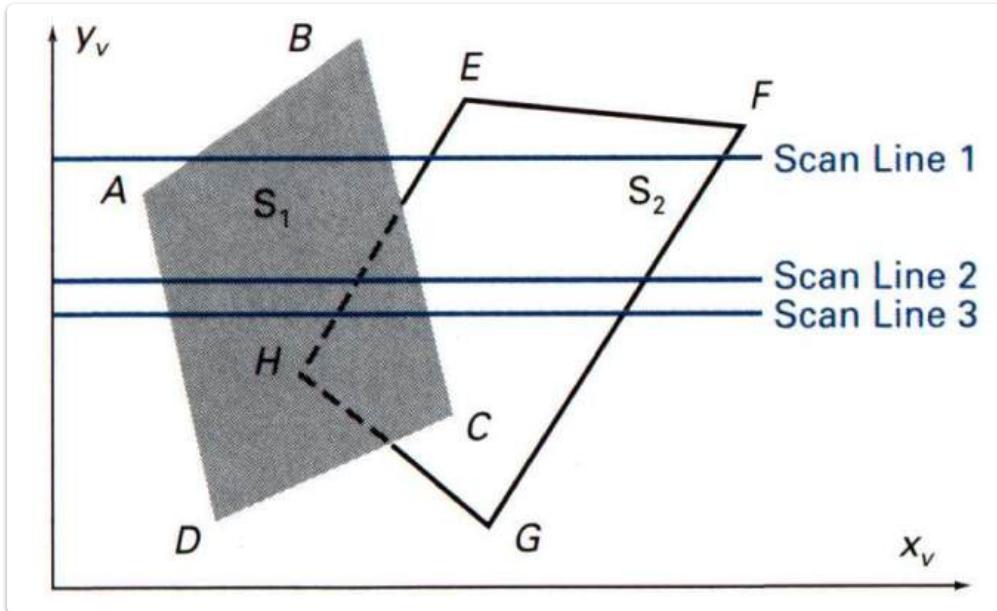
$$= z + \frac{A/m + B}{C}$$

constant !



Scanline-Methode

Berechnet die Sichtbarkeit **pixelzeilenweise** (z.B. von oben nach unten). Nutzt die Kohärenz zwischen aufeinanderfolgenden Scanlines, um effizient zu sein.



Depth-Sorting-Methode (Painter's Algorithm)

- Prinzip:** Polygone werden nach ihrer **Tiefenlage von hinten nach vorne sortiert** und dann in dieser Reihenfolge gezeichnet. Spätere (vordere) Polygone übermalen frühere (hintere), was die korrekte Sichtbarkeit erzeugt.
- Herausforderung:** Sicherstellen einer korrekten Sortierung, besonders bei überlappenden oder zirkulären Verdeckungen.

surfaces sorted in order of decreasing depth (viewing in $-z$ -direction)

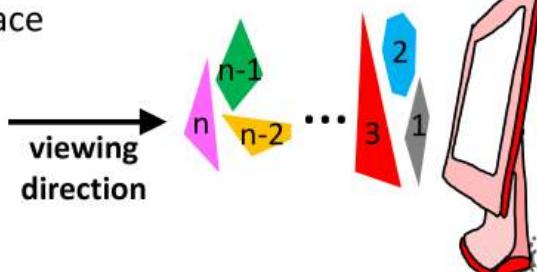
- (1) "approximate"-sorting using smallest z -value (greatest depth)
- (2) fine-tuning to get correct depth order

surfaces scan converted in order

sorting both in image and object space

scan conversion in image space

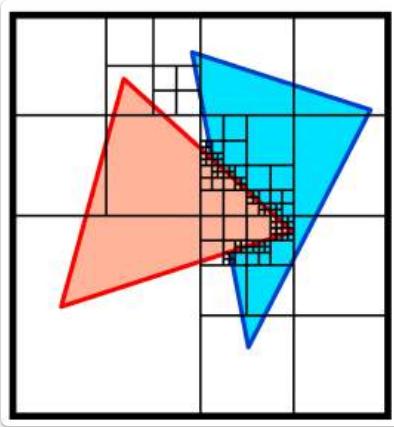
also called "painter's algorithm"



Area-Subdivision Methode

Nutzt einen **Divide-and-Conquer-Ansatz**.

- Vorgehen:** Das Problem wird zunächst in grober Auflösung gelöst. Bei komplexen Bereichen wird die Bildfläche rekursiv in vier Viertel unterteilt, bis eine pixelgenaue Lösung erreicht ist (ähnlich Quadtree).



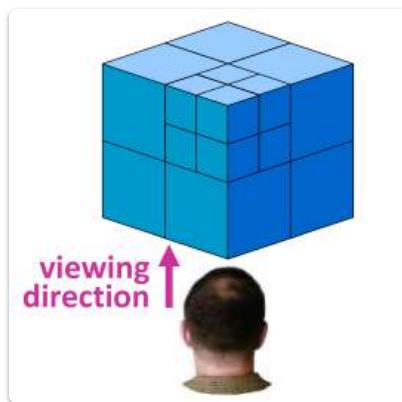
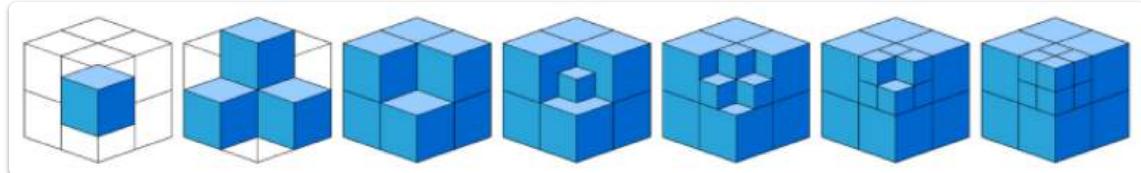
Octree-Methode

Repräsentiert die Szene als **Octree** (raumorientierter Baum).

- **Vorteil:** Die Datenstruktur enthält implizites Wissen über die Tiefenordnung.
- **Rendering-Strategien:**

Von hinten nach vorne: Rekursives Durchlaufen und Rendern der Teilwürfel vom entferntesten zum vordersten.

Von vorne nach hinten: Rekursives Durchlaufen und Zeichnen nur der sichtbaren Bereiche (muss Overdraw vermeiden).

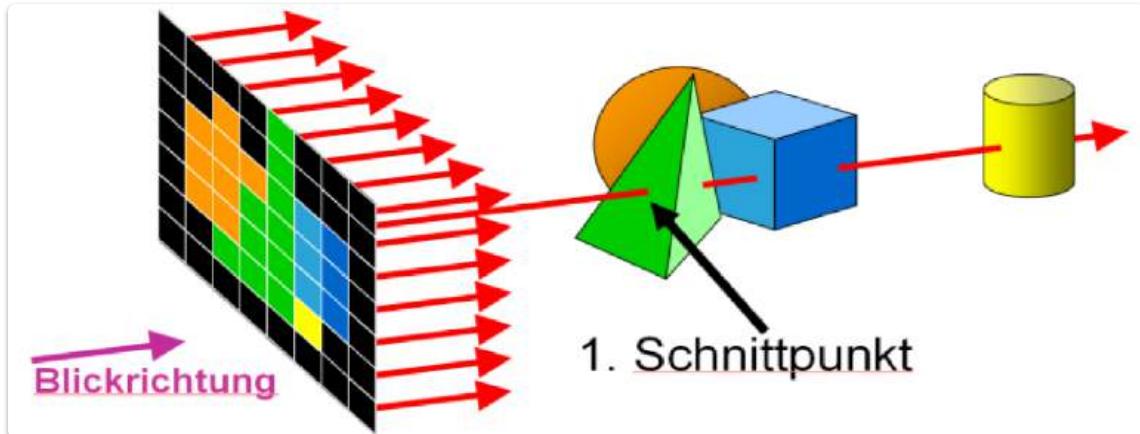


Ray-Casting

Berechnet die Sichtbarkeit für jedes Pixel einzeln.

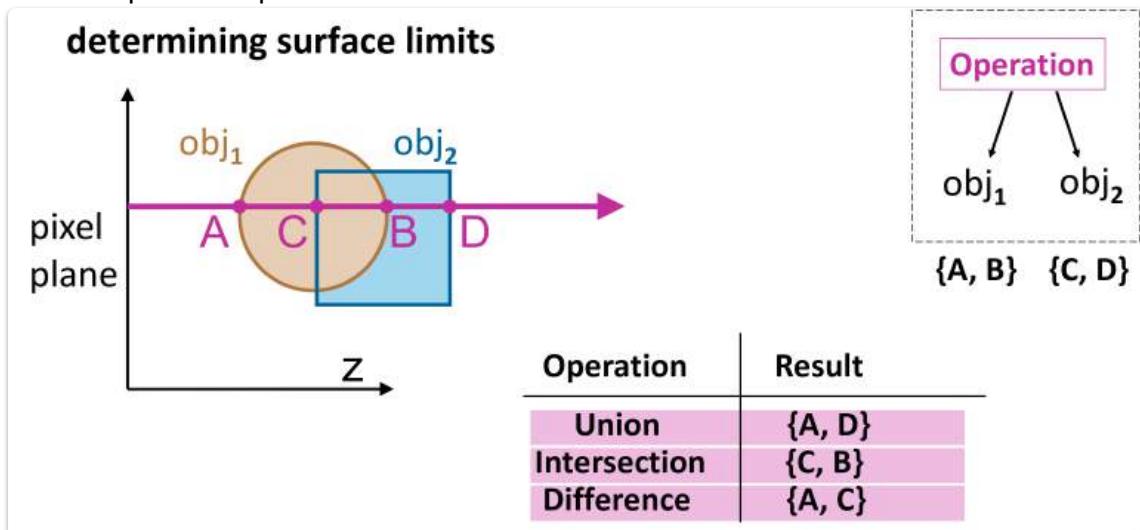
- **Ablauf pro Pixel:**
 1. Ein **Blickstrahl (Ray)** wird vom Pixel in die Szene projiziert.
 2. Alle Schnittpunkte des Strahls mit Objekten/Polygonen werden berechnet.
 3. Der **nächste Schnittpunkt** zum Betrachter wird ausgewählt.
 4. Die Farbe der Oberfläche am nächsten Schnittpunkt bestimmt die Pixelfarbe.

- **Vorteile:** Ermöglicht das Rendern verschiedenster Oberflächen (nicht nur Polygone).
- **Nachteile:** Hoher Rechenaufwand (viele Schnittberechnungen). Effiziente Implementierung ist entscheidend.



Ray-Casting von CSG-Objekten

- **Ablauf:** Für jedes Pixel wird ein Ray erzeugt und mit allen Objekten geschnitten. Der vorderste Schnittpunkt bestimmt das sichtbare Objekt und dessen Farbe.
- **Rekursive Berechnung bei CSG-Bäumen:**
 - **Endknoten (Primitive):** Einfache Schnittpunktberechnung.
 - **Zwischenknoten (Boolesche Operationen):** Schnittpunktlisten der Kindknoten werden verknüpft (Union, Intersection, Subtraction).
 - **Wurzelknoten:** Der dem Betrachter nächste Punkt der resultierenden Liste wird ausgewählt.
- **Beziehung zu Ray-Tracing:** Ray-Casting ist eine vereinfachte Version von **Ray-Tracing**, das komplexere optische Effekte simulieren kann.



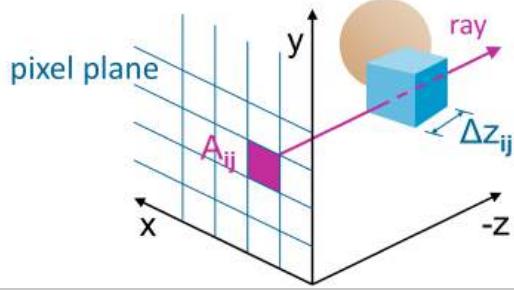
Ray-Casting = für jedes Pixel der Darstellungsfläche:

- 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

volume determination

$$V_{ij} \approx A_{ij} \cdot \Delta z_{ij}$$

$$V \approx \sum V_{ij}$$



Klassifizierung der Verfahren

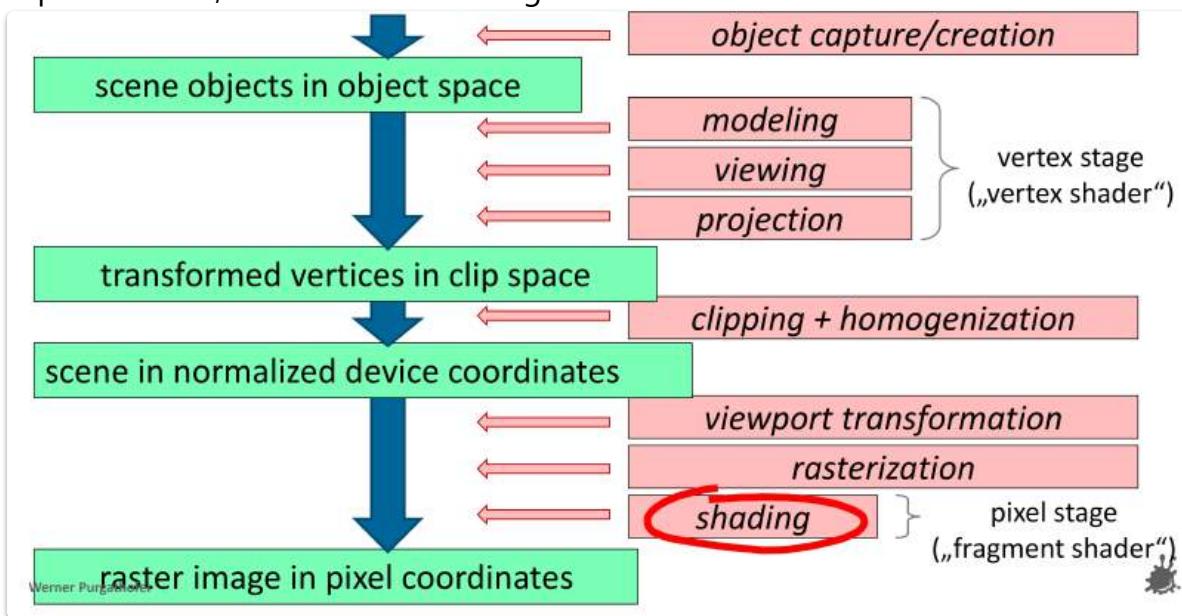
- **Objektraum-Verfahren:**
 - Backface Detection
 - Depth Sorting
 - Octree-Methode
- **Bildraum-Verfahren:**
 - Z-Puffer
 - Scanline-Methode
 - Area Subdivision
 - Ray Casting

9. Beleuchtung und Schattierung

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

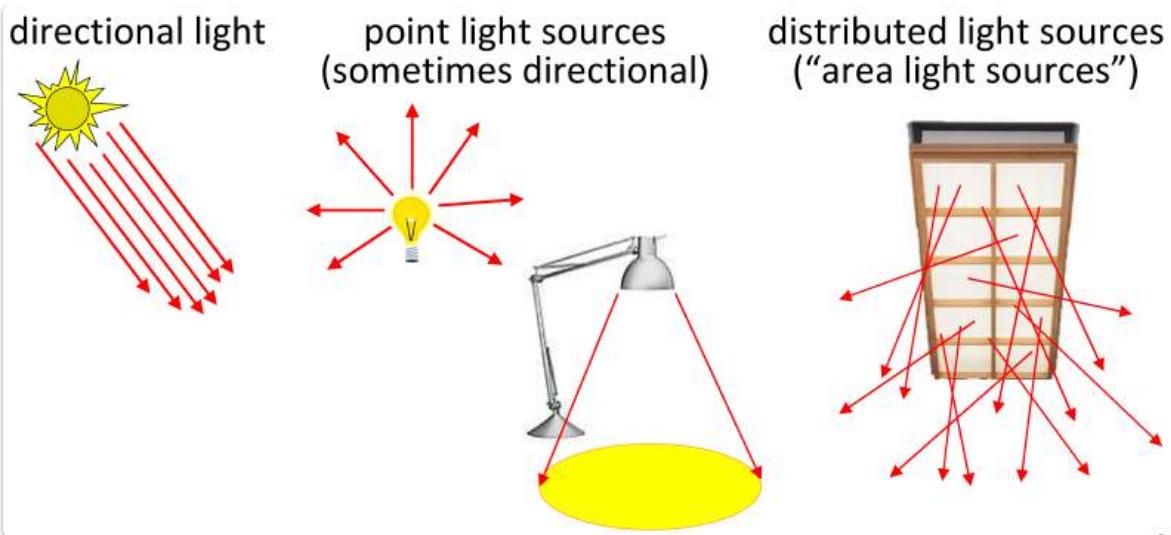
Ein **Beleuchtungsmodell** berechnet die wahrgenommene Farbe/Helligkeit eines Objekts für den Betrachter. Es berücksichtigt dabei die **Lichtverhältnisse in der Szene** und die **Oberflächeneigenschaften des Objekts**. Das Ziel ist, die Farbe zu bestimmen, die ein Pixel im Bild erhalten soll, was für realistisch aussehende Computergrafiken entscheidend ist. Zunächst konzentriert man sich auf die Helligkeit, für Farben werden die Berechnungen separat für Rot, Grün und Blau durchgeführt.



Lichtquellen und Objektoberflächen

Lichtquellen

Lichtquellen sind für Beleuchtungseffekte unerlässlich und können verschiedene **Formen** haben (z.B. paralleles Licht, Punktlichtquellen, gerichtete Lichtquellen, flächige Lichtquellen) und **Eigenschaften** (Helligkeit, Farbe, Entfernung).

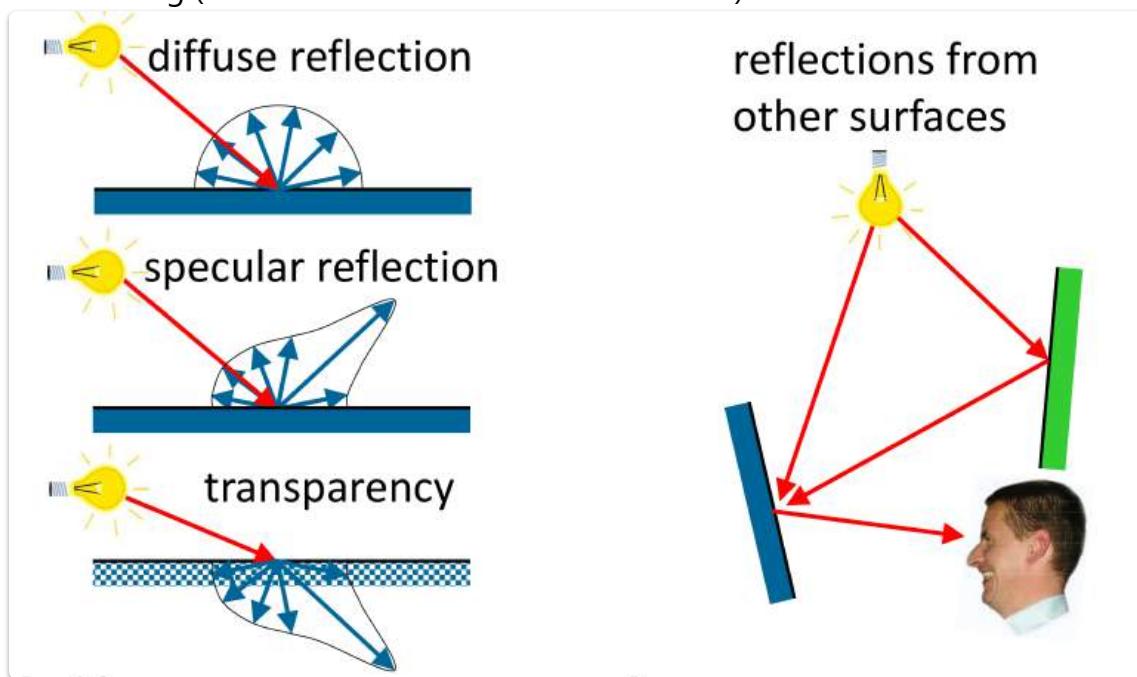


Objektoberflächen

Oberflächen interagieren mit Licht durch:

- **Diffuse Reflexion:** Licht wird gleichmäßig in alle Richtungen reflektiert (z.B. Papier).
- **Spiegelnde Reflexion:** Licht wird bevorzugt in eine Spiegelungsrichtung reflektiert (z.B. Lack).
- **Transparenz:** Licht durchdringt die Oberfläche (z.B. Glas).

Die meisten Oberflächen weisen eine Kombination dieser Eigenschaften auf. Indirekte Beleuchtung (Reflexionen von anderen Oberflächen) ist ebenfalls relevant.



Ein einfaches Beleuchtungsmodell

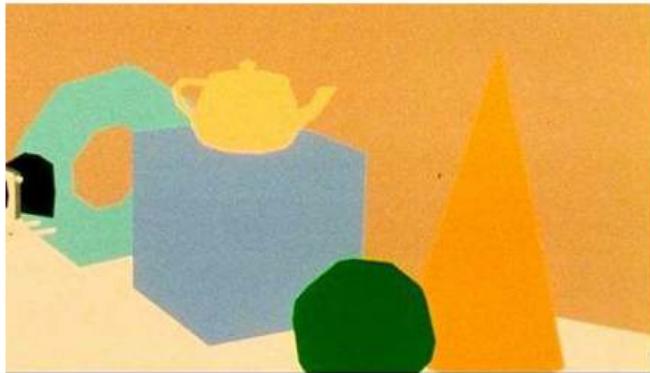
Die physikalisch genaue Simulation von Licht ist aufwendig. Daher werden vereinfachte, empirische Modelle verwendet, um visuell plausible Beleuchtung mit überschaubarem Aufwand zu erzielen.

Hintergrundlicht (Ambientes Licht)

Ambientes Licht ist ein konstanter Helligkeitswert (I_a), der zu jeder Beleuchtungsberechnung addiert wird. Es approximiert das überall vorhandene, indirekte Basislicht, das verhindert, dass Bereiche ohne direkte Beleuchtung vollständig dunkel sind.

- ambient light (background light) \mathbf{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)

Die Helligkeit einer diffus reflektierenden Oberfläche ist proportional zum Kosinus des Winkels θ zwischen der Oberflächennormale \mathbf{n} und der Richtung zur Lichtquelle \mathbf{l} . Dies erzeugt den Eindruck räumlicher Form.

- **Formel:** $L = k_d \cdot I \cdot \cos \theta$ oder $L = k_d \cdot I \cdot (\mathbf{n} \cdot \mathbf{l})$

I: Helligkeit der Lichtquelle

k_d : Diffuser Reflexionskoeffizient ($0 \leq k_d \leq 1$)

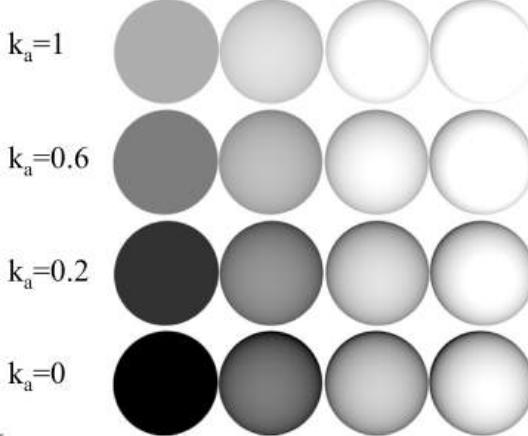
* $\mathbf{n} \cdot \mathbf{l}$: Skalarprodukt der Normalen \mathbf{n} und des Lichtrichtungsvektors \mathbf{l} .

Kombiniert man dies mit ambientem Licht (I_a), ergibt sich eine bereits ansprechende Darstellung.

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) / Phong-Beleuchtungsmodell

Um den Glanz von Oberflächen zu modellieren, wird oft das **Phong-Beleuchtungsmodell** verwendet, das einen Glanzpunkt (Specular Highlight) hinzufügt.

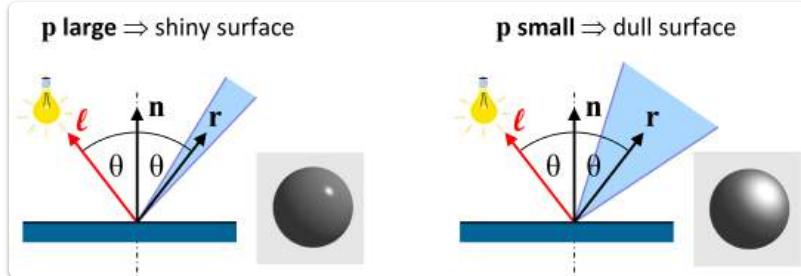
- **Formel für den Glanzpunkt:** $I_{spec} = k_s \cdot I_L \cdot \cos^p(\alpha)$

I_L : Intensität des Lichts

k_s : Spekularer Reflexionskoeffizient

α : Winkel zwischen dem exakten Reflexionsstrahl r und der Blickrichtung v .

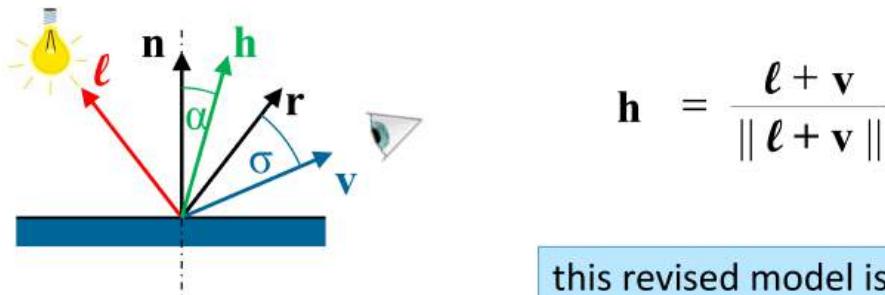
p : **Glanz-Exponent (Poliertheit)**. Ein größerer p erzeugt einen kleineren, glatter wirkenden Glanzpunkt.



- Der Reflexionsvektor r berechnet sich als $r = (2\mathbf{n} \cdot \mathbf{l})\mathbf{n} - \mathbf{l}$.
- **Blinn-Phong-Beleuchtungsmodell:** Eine vereinfachte Form, bei der $\cos^p(\alpha)$ durch $(\mathbf{n} \cdot \mathbf{h})^p$ ersetzt wird, wobei \mathbf{h} der Halbierungsvektor zwischen \mathbf{l} und \mathbf{v} ist. Der Winkel zwischen \mathbf{n} und \mathbf{h} ist oft ähnlich dem zwischen \mathbf{r} und \mathbf{v} .

simplified Phong model with halfway vector \mathbf{h}

$$L_{spec} = k_s \cdot I \cdot (\mathbf{v} \cdot \mathbf{r})^p \quad \rightarrow \quad L_{spec} = k_s \cdot I \cdot (\mathbf{n} \cdot \mathbf{h})^p$$



- **Komplettes Beleuchtungsmodell (Blinn-Phong):**

$$L = k_a \cdot I_a + \sum_{i=1,\dots,N} (k_d \cdot I_i \cdot (\mathbf{n} \cdot \mathbf{l}_i) + k_s \cdot I_i \cdot (\mathbf{n} \cdot \mathbf{h}_i)^p)$$

Dabei sind k_a der ambiente Reflexionskoeffizient und N die Anzahl der Lichtquellen.



Schattierung von Polygonen

Beim Schattieren von Polygonen kann es durch die einzelnen Flächen zu sichtbaren Kanten kommen (**Mach-Band-Effekt**). Um dies zu vermeiden, werden die Schattierungen zwischen den Polygonen interpoliert.

Flat-Shading

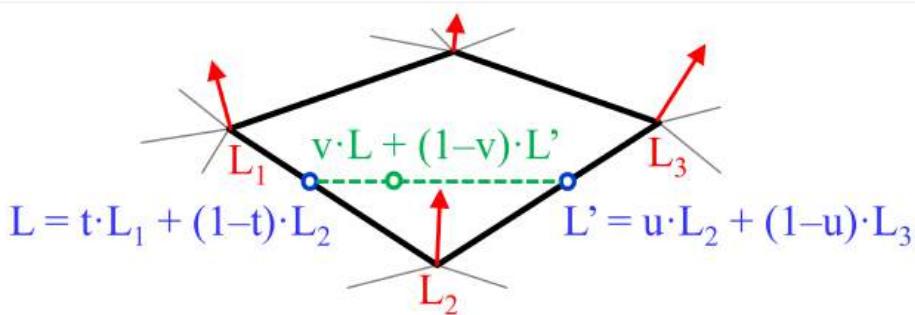
Jeder Punkt eines Polygons erhält die gleiche Farbe, basierend auf einer einzigen Beleuchtungsberechnung für das gesamte Polygon. Dies führt zu deutlich sichtbaren Kanten zwischen Polygonen.



Gouraud-Schattierung

Interpoliert die **berechneten Helligkeitswerte** über die Polygonflächen.

1. **Eckennormalen:** Mittelwert der Normalen angrenzender Polygone an jedem Eckpunkt.
2. **Eckpunktintensitäten:** Helligkeitswerte an jedem Eckpunkt basierend auf den Eckennormalen und Lichtrichtung.
3. **Interpolation entlang Kanten:** Lineare Interpolation der Helligkeitswerte entlang der Polygonkanten.
4. **Interpolation entlang Scanlines:** Lineare Interpolation der Helligkeitswerte entlang jeder Scanline.



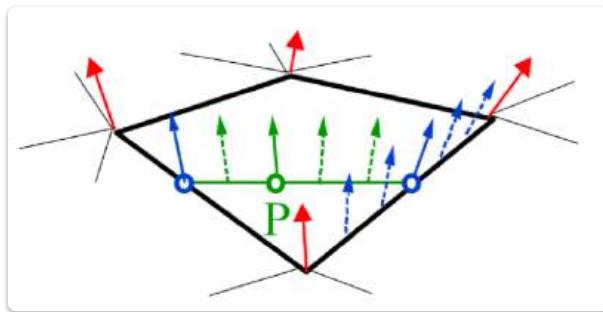
1. find normal vectors at corners and calculate shading (intensities) there: L_i
2. interpolate intensities along edges linearly: L, L'
3. interpolate intensities along scanlines linearly: L_p

- **Problem bei Glanzpunkten:** Kann zu zufälligen Interpolationsergebnissen führen ("wandernde" Glanzpunkte bei Bewegung).

Phong-Schattierung

Interpoliert die **Normalenvektoren** und berechnet die Helligkeit pro Pixel.

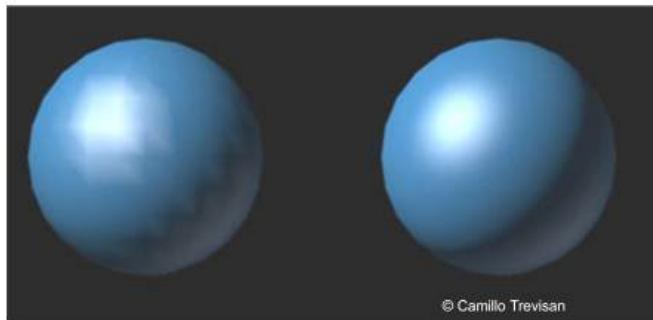
1. Normalen an Polygon-Eckpunkten berechnen.
 2. Normalen entlang Polygonkanten interpolieren.
 3. Interpolierte Normalen entlang Scanlines weiter interpolieren (pro Pixel).
 4. Pro Pixel mit der interpolierten Normalen die Helligkeit nach dem Beleuchtungsmodell berechnen.
- **Vorteil:** Konsistenter Glanzpunkte.
 - **Nachteil:** Höherer Rechenaufwand, da die Beleuchtung pro Pixel berechnet wird.



Wichtig: Phong-Beleuchtungsmodell und Phong-Schattierung sind unterschiedliche Konzepte!

comparison to Gouraud shading

- better highlights
- less Mach banding
- more costly
- wrong silhouette stays!



10. Ray-Tracing

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Ray-Tracing ("Verfolgen von Strahlspuren") ist eine mächtige Methode zur Simulation optischer Effekte in der Computergrafik. Sie baut auf dem **Ray-Casting** auf und ermöglicht die Darstellung von **Schatten, Spiegelbildern und Lichtbrechung**. Das Verfahren zeichnet sich durch seine Einfachheit aus, die die Darstellung komplexer Objekte wie Freiformflächen oder fraktaler Oberflächen erlaubt. Lichtstrahlen werden dabei in **verkehrter Richtung** verfolgt, also vom Auge (Kamera) zur Lichtquelle.

Das Ray-Tracing Prinzip

Die Kernidee ist, für jeden Bildpunkt (Pixel) den dorthin treffenden Lichtstrahl in umgekehrter Richtung zu verfolgen, um zu untersuchen, woher das Licht kommt. Daraus wird dann das Aussehen des Pixels bestimmt.

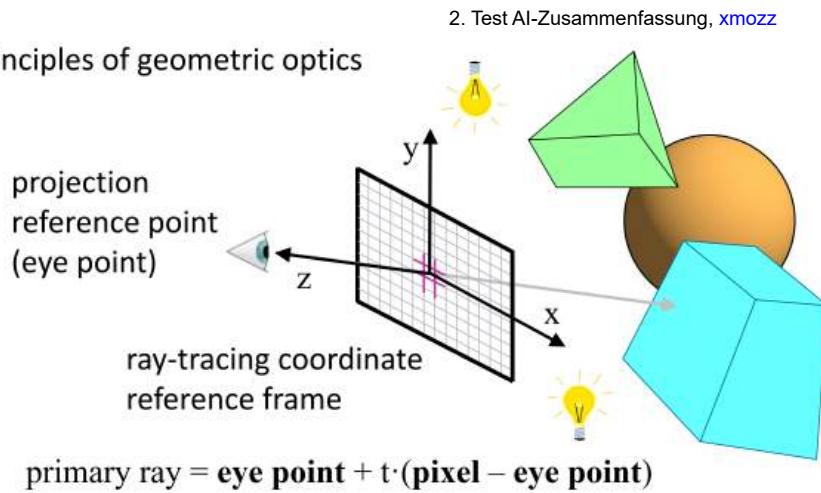
Korrekte Sichtbarkeit und Schattierung

1. Ein **Blickstrahl (Primärstrahl)** wird durch jeden Bildpunkt gelegt.
2. Dieser Strahl wird mit allen Oberflächen der Szene geschnitten.
3. Der **nächste Schnittpunkt** zum Bild hin wird ausgewählt.
4. Die Schattierung dieses Objektpunktes (aus Blickrichtung) wird als Pixelwert verwendet.
Dies wird für alle Pixel wiederholt, um eine Szene mit korrekter Sichtbarkeit zu erhalten.
Ein beliebiges Schattierungsmodell (z.B. Phong-Modell) kann verwendet werden.

Schatten

Um Schatten zu berechnen, wird für jeden zu schattierenden Punkt ein **Schattenfühler (Sekundärstrahl)** zur Lichtquelle gelegt. Wenn dieser Strahl auf ein anderes Objekt trifft, bevor er die Lichtquelle erreicht, liegt der Punkt im Schatten, und der Lichteinfluss dieser Quelle wird reduziert oder eliminiert.

principles of geometric optics

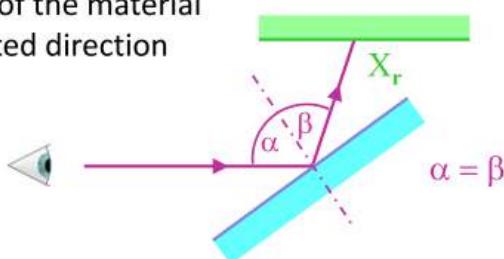


Spiegelbilder

Trifft ein Blickstrahl auf ein spiegelndes Objekt, wird die Sichtbarkeit nicht vom Objekt selbst bestimmt, sondern von dem, was in **Spiegelungsrichtung** liegt. Ein **Reflexionsstrahl (Sekundärstrahl)** wird gemäß dem Reflexionsgesetz (Einfallsinkel = Ausfallsinkel) erzeugt und verfolgt. Die Farbe des Pixels wird dann durch die Schattierung des Objekts bestimmt, das dieser Reflexionsstrahl trifft. Dies ermöglicht die 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

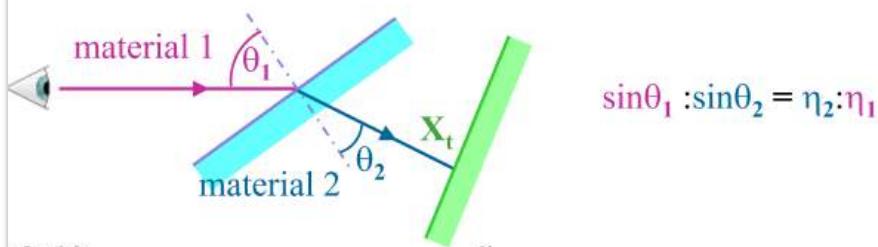


Transparenz

Bei transparenten Objekten wird ein **Transparenzstrahl (Sekundärstrahl)** vom Auftreffpunkt durch das Objekt verfolgt, wobei das **Brechungsgesetz** berücksichtigt wird. Die Farbe des Pixels ergibt sich aus der Schattierung des Objekts, das dieser Transparenzstrahl trifft.

$$I_t = k_t \cdot X_t$$

I_t ... illumination caused by transparency
 k_t ... transparency coefficient of the material
 X_t ... shading in the transparency direction

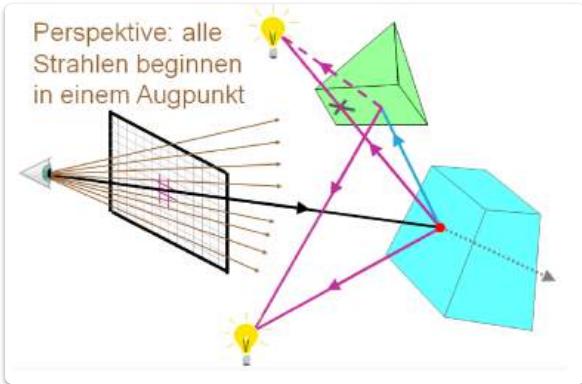


Rekursion

Ray-Tracing nutzt **Rekursion**: Jeder Strahl (außer Schattenfühler) ist gleichwertig, unabhängig davon, ob es sich um einen Primär- oder Sekundärstrahl handelt. Dies ermöglicht die einfache Simulation von **Mehrfachspiegelungen** und Spiegelungen hinter transparentem Material.

Perspektive

Die Erzeugung der primären Blickstrahlen bestimmt die Projektionsart. Strahlen, die von einem fiktiven **Augpunkt** ausgehen, erzeugen eine **perspektivische Projektion** ohne zusätzlichen Aufwand, was ein natürliches Bild erzeugt.



Ray-Tracing Implementierung

Ein Ray-Tracer erfordert 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 Schattenfühler  $p \rightarrow s$  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:** Die xy-Ebene ist die Bildebene, die Hauptblickrichtung ist die negative z-Achse.
- **Strahlen (parametrisierte Form):** $p(t) = p_0 + t \cdot d$
 - p_0 : Startpunkt
 - t : Parameter
 - d : Richtungsvektor

- **Primärstrahlen:** $e + t \cdot (p_0 - e)$ (e ist der Augpunkt, p_0 die Pixelkoordinate).
- **Schattenfühler:** $p + t \cdot (s - p)$ (p ist der Oberflächenpunkt, s die Lichtquellenposition).
- **Reflexionsstrahlen:** $p + t \cdot r$, wobei $r = (2n \cdot v)n - v$ (n ist die Normale, v die Richtung des einfallenden Strahls).

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 (η_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

Objekte, die per Ray-Tracing dargestellt werden sollen, müssen bestimmte Bedingungen erfüllen: Der **Schnittpunkt mit einer Geraden muss berechenbar sein**, die **Oberflächennormale** am Schnittpunkt muss bekannt sein und die **Materialeigenschaften** am Schnittpunkt müssen verfügbar sein. Diese Bedingungen werden von BReps (Boundary Representations), CSG-Bäumen und vielen anderen Datenformaten erfüllt. Für jede Primitivart (Kugel, Polygon etc.) werden spezielle Schnittberechnungsfunktionen benötigt.

Schnitt Strahl-Kugel

Die Schnittpunkte eines Strahls $p(t) = e + td$ mit einer Kugel (Mittelpunkt c , Radius R) werden durch das Einsetzen des Strahls in die Kugelgleichung $|p - c|^2 - R^2 = 0$ ermittelt. Dies führt zu einer quadratischen Gleichung in t :

$$t^2 - 2(\mathbf{d} \cdot \Delta \mathbf{p})t + (|\Delta \mathbf{p}|^2 - R^2) = 0$$

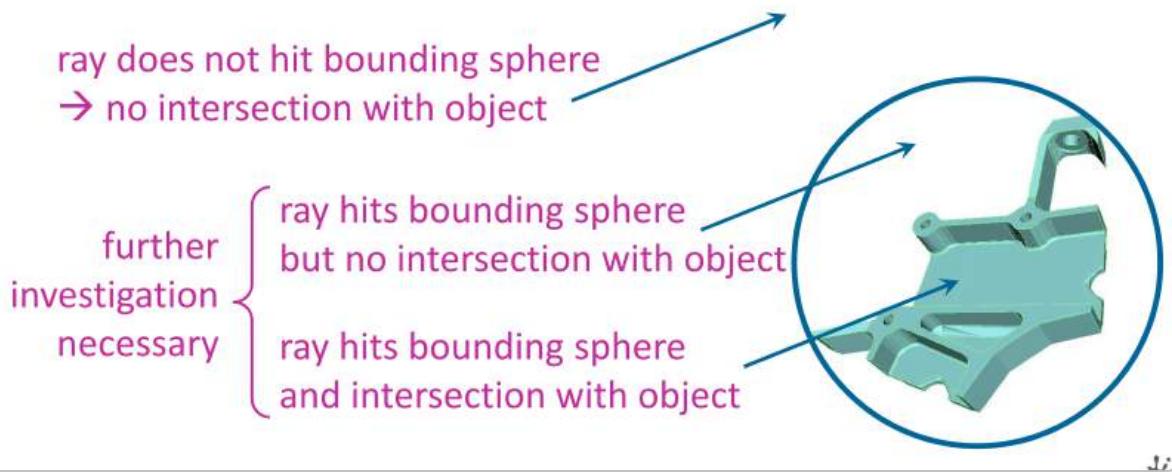
wobei $\Delta \mathbf{p} = \mathbf{c} - \mathbf{e}$. Die Lösungen für t sind:

$$t = \mathbf{d} \cdot \Delta \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \Delta \mathbf{p})^2 - |\Delta \mathbf{p}|^2 + R^2}$$

Alternativ, zur Vermeidung von Rundungsfehlern:

$$t = \mathbf{d} \cdot \Delta \mathbf{p} \pm \sqrt{|\Delta \mathbf{p}|^2 - (\mathbf{d} \cdot \Delta \mathbf{p})^2 - R^2}$$

use **bounding sphere** to eliminate easy cases



Schnitt Strahl-Polygon

- Backface Detection:** Zuerst prüfen, ob das Polygon zur Kamera ausgerichtet ist.
- Strahlengleichung:** $\mathbf{p} = \mathbf{p}_0 + t \cdot \mathbf{d}$
- Ebenengleichung des Polygons:** $\mathbf{n} \cdot \mathbf{p} = -D$
- Schnittpunkt mit der Ebene:** Setze die Strahlengleichung in die Ebenengleichung ein und löse nach t auf:

$$t = -\frac{D + \mathbf{n} \cdot \mathbf{p}_0}{\mathbf{n} \cdot \mathbf{d}}$$

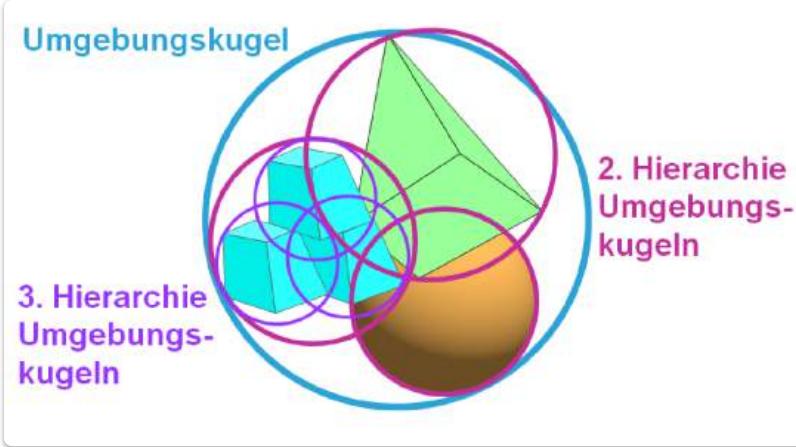
- Überprüfung:** Prüfen, ob der Schnittpunkt tatsächlich innerhalb der Polygonbegrenzung liegt (z.B. durch 2D-Projektion).

Ray-Tracing Beschleunigung

Ray-Tracing ist sehr rechenintensiv (z.B. 10^9 Schnittberechnungen für 1000 Polygone auf 1000x1000 Pixeln). Daher ist eine **signifikante Beschleunigung** durch Reduktion der notwendigen Schnittberechnungen entscheidend.

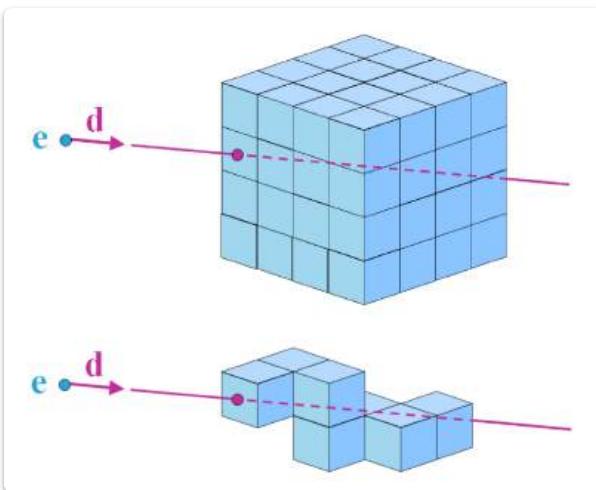
Objektumgebungen (Bounding Volumes)

- Idee:** Komplexe Objekte werden von einfacheren Formen (z.B. **Umgebungskugeln** oder **Umgebungsquader**) umschlossen.
- Funktionsweise:** Zuerst wird geprüft, ob der Strahl die Umgebungskugel trifft. Nur wenn ja, wird eine detaillierte Schnittberechnung mit dem eigentlichen Objekt durchgeführt.
- Hierarchische Umgebungskugeln:** Komplexe Objekte können hierarchisch in Teilobjekte mit eigenen Umgebungskugeln unterteilt werden, um die Suche effizienter zu gestalten (reduziert die Komplexität von $O(n)$ auf $O(\log n)$).



Raumteilungs-Methoden

- **Idee:** Der gesamte Raum der Szene wird in Teilräume unterteilt, unabhängig von den Objekten.
- **Methoden:** **Regelmäßiges Raster** (Array von Würfeln/Voxeln) oder **Octree** (hierarchische Raumauftteilung).
- **Vorteile:** Man muss nur die Objekte in den Teilräumen betrachten, durch die der Strahl geht. Die Berechnung des nächsten Teilraums auf dem Strahlpfad ist schnell (ähnlich 3D-Bresenham-Verfahren). Sobald ein Schnittpunkt gefunden wurde, kann die Suche in diesem Teilwürfel beendet werden.

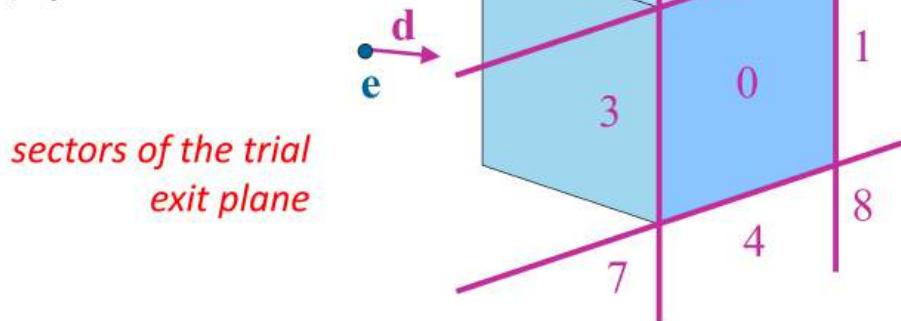


- **Berechnung:** Der Eintritts- und Austrittspunkt des Strahls in/aus einem Teilwürfel sowie die zugehörigen t -Werte können durch die Schnittpunkte des Strahls mit den Würfelflächen bestimmt werden.

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.

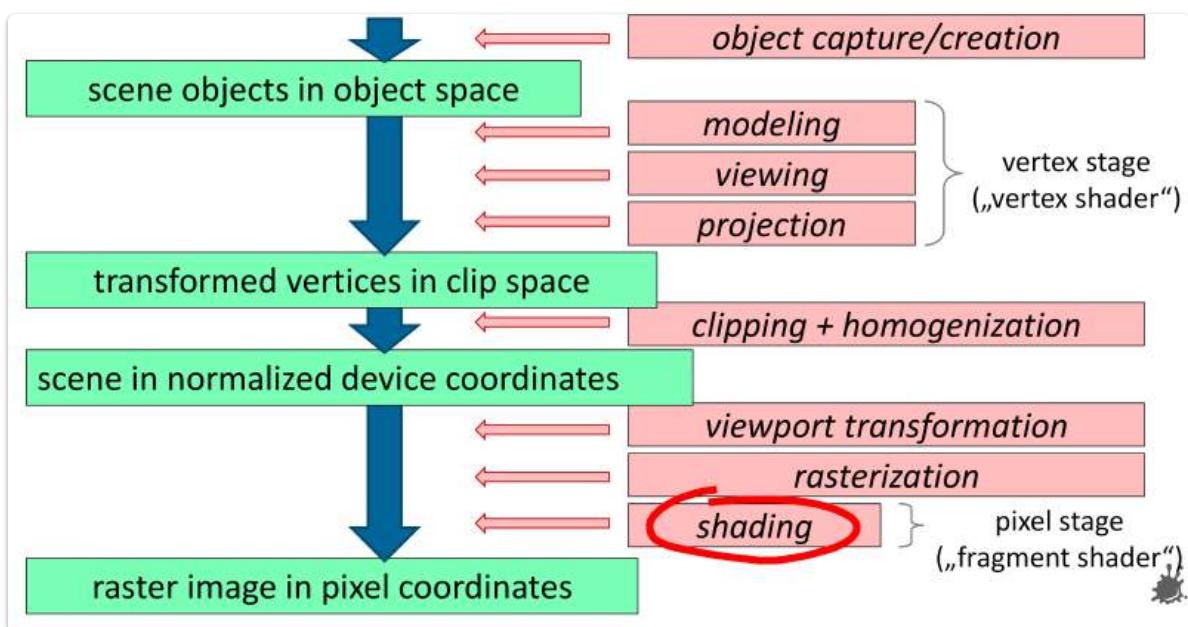


11. Globale Beleuchtung und Texturen

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Rendering Pipeline



Radiosity

Radiosity ist ein physikalisches Beleuchtungsmodell, das die **Lichtausbreitung in einer diffus reflektierenden Umgebung** basierend auf dem Energiegleichgewicht modelliert. Es stammt ursprünglich aus der Wärmeübertragung. Ziel ist die Berechnung der Helligkeit aller Flächen einer Szene unter Berücksichtigung ihrer gegenseitigen Beeinflussung. Das bedeutet, auch indirekt beleuchtete Flächen erhalten Helligkeit, da jeder beleuchtete Gegenstand als sekundäre Lichtquelle wirkt.

Ein wesentlicher Vorteil ist, dass die Lichtausbreitung im Raum **unabhängig von der Kameraposition** berechnet wird. Die Szene kann anschließend aus verschiedenen Blickrichtungen dargestellt werden, ohne die Beleuchtung neu berechnen zu müssen.

Die Radiosity Gleichung

Die Szene wird in n **Patches** (ebene Polygone mit homogener, perfekt diffuser Oberfläche) unterteilt. Lichtquellen sind ebenfalls Patches. Die **Radiosity B_i** eines Patches P_i ist die

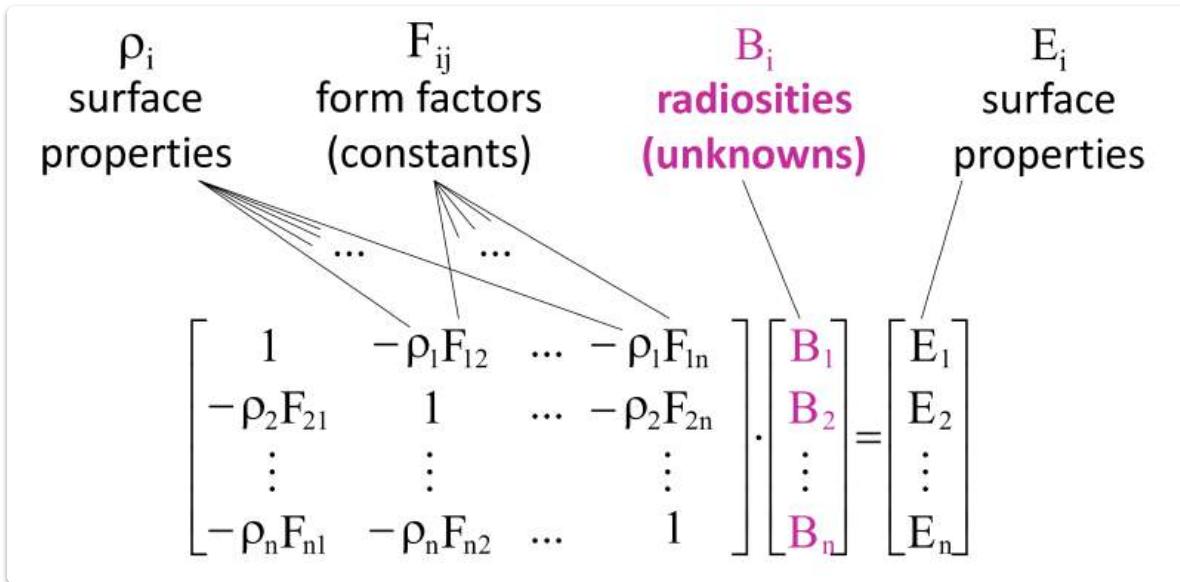
gesamte abgestrahlte Energie pro Flächeneinheit und setzt sich aus der Eigenemission E_i und der reflektierten Leistung zusammen:

$$B_i = E_i + \rho_i \sum_{j=1}^n B_j F_{ij}$$

- B_i : Radiosity des Patches i
- E_i : Eigenemission des Patches i
- ρ_i : Diffuser Reflexionskoeffizient (Albedo) von Patch i
- F_{ij} : **Formfaktor** zwischen Patch i und Patch j . Er gibt den Anteil der Radiosity von Patch j an, der auf Patch i trifft (und umgekehrt, aufgrund des Reziprozitätsgesetzes). Formfaktoren sind rein geometrisch.

Diese Formeln bilden ein lineares Gleichungssystem, das oft iterativ gelöst wird:

$$\begin{pmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$



Berechnung der Formfaktoren

Formfaktoren F_{ij} sind ein Maß für den geometrischen Austausch von Energie zwischen zwei Patches. Sie hängen von den Flächengrößen, Abständen und Orientierungen der Normalen ab. Unter der Annahme, dass Patches klein im Verhältnis zum Abstand sind und keine Hindernisse dazwischen liegen, gilt:

$$F_{ij} = \frac{\cos \phi_i \cos \phi_j A_j}{\pi r^2}$$

- ϕ_i, ϕ_j : Winkel zwischen den Patch-Normalen und der Verbindungsgeraden.
- A_j : Fläche von Patch j .
- r : Abstand zwischen den Patches.

Das **Reziprozitätsprinzip** besagt: $A_i F_{ij} = A_j F_{ji}$.

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

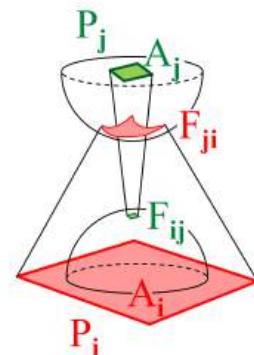
$$\sum_{j=1}^n F_{ij} = 1$$

- uniform light reflection

$$A_i F_{ij} = A_j F_{ji}$$

- no self-incidence

$$F_{ii} = 0$$



Praktische Berechnungsmethoden umfassen die **Hemicube-Methode** (Projektion der Szene auf einen Halbwürfel) oder Ray-Tracing-Verfahren, um die gegenseitige Sichtbarkeit zu berücksichtigen.

Fortschreitende Verfeinerung (Progressive Refinement)

Dieses iterative Verfahren löst das Radiosity-Gleichungssystem. Es wählt in jedem Schritt das **hellste Patch** aus und "schießt" (verteilt) dessen noch nicht abgestrahlte Energie auf alle anderen Patches.

- **Vorteil:** Die Szene wird in jedem Schritt besser beleuchtet, und das Verfahren konvergiert schnell.
- Jedes Patch speichert seine **gesammelte Radiosity (B_i)** und die **noch nicht verschossene Radiosity (ΔB_i)**. Initial sind beide gleich der Eigenemission E_i .

Ein Iterationsschritt umfasst:

1. Wähle das Patch P_s mit der größten ΔB_s .
2. Berechne für alle anderen Patches P_j den Energieanteil, der von P_s auf P_j übertragen wird: $\Delta B_j = \Delta B_j + \rho_j \Delta B_s F_{sj}$.
3. Setze $\Delta B_s = 0$.

$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

"shooting" → select brightest patch P_i and distribute its radiosity B_i

$$B_{j \text{ due to } B_i} = \rho_j B_i F_{ij} \frac{A_i}{A_j}$$

Aspekte von Radiosity

- **Blickpunktunabhängigkeit:** Einmal berechnet, kann die Beleuchtung aus beliebigen Kamerablickpunkten verwendet werden.
- **Kombination mit Ray-Tracing:** Radiosity modelliert nur diffuse Beleuchtung. Für Effekte wie Spiegelungen oder scharfe Schatten wird oft ein zusätzlicher Ray-Tracing-Schritt angewendet, der die Radiosity-Ergebnisse als diffuse Grundbeleuchtung nutzt.

radiosity is viewpoint-independent
needs a rendering step to display

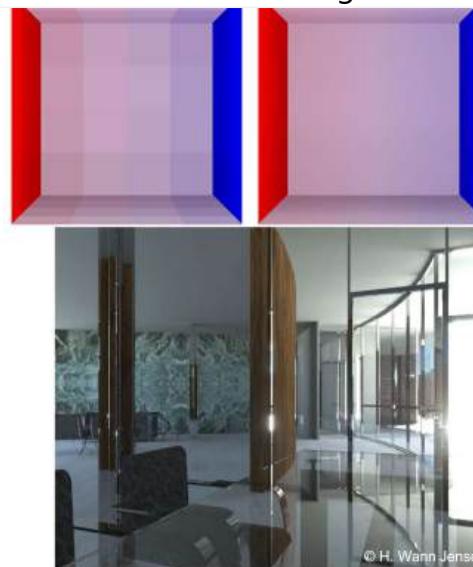
- polygon rendering
- Gouraud shading
- ray-tracing
- ...

combination with ray-tracing enables

- reflections
- shadows
- ...

Werner Purgathofer

23



© H. Wann Jense

Erweiterungen des Radiosity-Prinzips

- **Hierarchische Strukturierung von Patches:** Reduziert den Rechenaufwand, indem entfernte Patches als Gruppen zusammengefasst werden.
- **Discontinuity-Meshing:** Passt die Patch-Größe an Helligkeitsunterschiede an (kleinere Patches an Schattenkanten, größere in gleichmäßigen Bereichen), um Artefakte zu vermeiden und die Effizienz zu steigern.

Path Tracing

Path Tracing, auch bekannt als **Monte Carlo Ray-Tracing**, ist eine Erweiterung des Ray-Tracing-Verfahrens zur realistischen Lichtsimulation.

- **Kernprinzip:** Im Gegensatz zum klassischen Ray-Tracing, das an jedem Auftreffpunkt mehrere Sekundärstrahlen erzeugt (für Reflexion, Brechung, Schatten), wird beim Path Tracing **zufällig nur eine Richtung** ausgewählt, entsprechend einer gültigen Verteilungsfunktion. Es wird ein "Lichtpfad" von der Kamera bis zu einer Lichtquelle oder ins Unendliche verfolgt.
- **Vorteile:** Ermöglicht die realistische Simulation komplexer Lichtverhältnisse, modelliert diffuse Reflexionen sehr gut und kommt mit ausgedehnten Lichtquellen besser zurecht.
- **Herausforderung:** Da pro Pixel nur eine zufällige Richtung verfolgt wird, führt dies zu **Rauschen** im Ergebnis. Um dieses Rauschen zu reduzieren, müssen **viele Strahlen pro Pixel** berechnet und gemittelt werden, was rechenintensiv ist.
- **Mathematischer Hintergrund:** Das Vorgehen entspricht der **Monte-Carlo-Integration** der Rendering Equation, einer komplexen Gleichung, die die vollständige

Lichtausbreitung beschreibt. Die Qualität kann durch die Verwendung von **Quasi-Zufallszahlen** (gleichmäßiger verteilt als Pseudo-Zufallszahlen) verbessert werden.

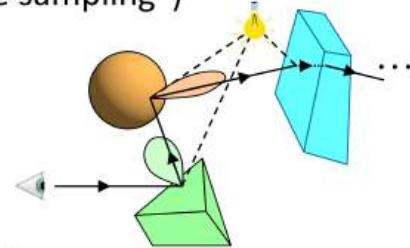
also called Monte Carlo ray tracing

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 **ρ** ... reflection coefficient

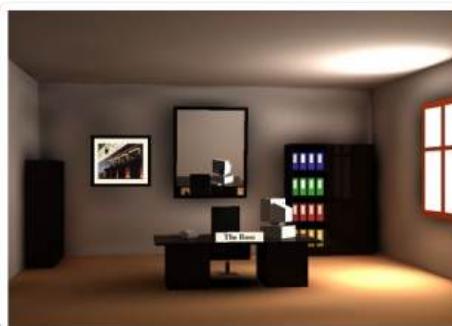
Photon Mapping

Photon Mapping ist eine globale Beleuchtungsmethode, die besonders gut mit komplexen Lichtinteraktionen wie **Kaustiken** (Lichtbündel durch Brechung/Reflexion) und Spiegelungen umgehen kann.

- **Vorgehensweise:**

1. **“Vorwärts”-Verfolgung:** Photonen (Lichtstrahlen) werden von den Lichtquellen ausgesendet und in der Szene verfolgt.
2. **Speichern der Lichtwirkung:** Wenn ein Photon auf eine diffuse oder teilweise spiegelnde Oberfläche trifft, wird seine Wirkung (Energie, Farbe) an diesem Punkt in einer **Photon Map** gespeichert.
3. **Interpolation:** Während des Renderings werden die gespeicherten Photonen nahe eines Oberflächenpunktes interpoliert, um dessen Beleuchtung zu bestimmen.

- **Vorteile:** Berechnet korrekt Kaustiken und Spiegelungen von Lichtquellen.
- **Kombination:** Photon Mapping kann mit Path Tracing kombiniert werden, um nahezu alle Lichteffekte für sehr realistische Renderings zu integrieren.

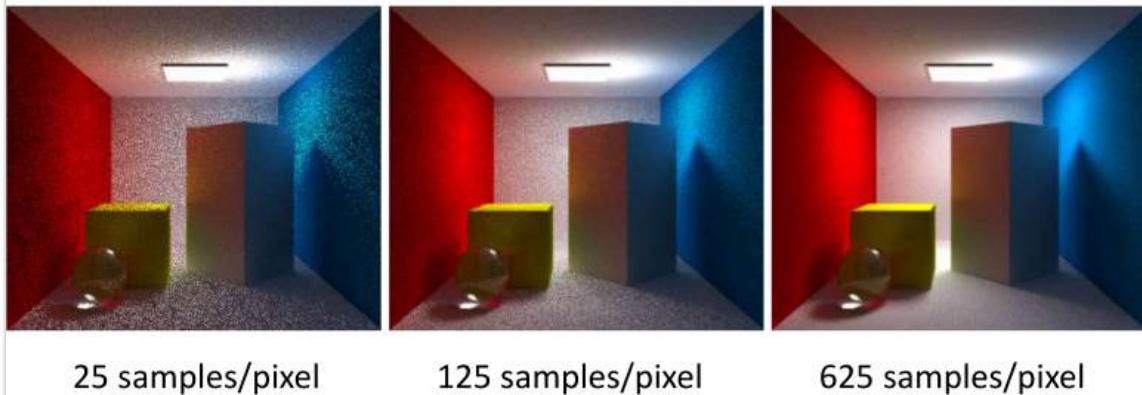


Optische Eigenschaften natürlicher Oberflächen: Mapping-Methoden

Natürliche Oberflächen weisen Unregelmäßigkeiten auf, die mit speziellen "Mapping"-Methoden simuliert werden können:

- **Environment Mapping:** Simuliert die Reflexion der Umgebung auf einer Oberfläche. Eine vorab generierte Umgebung (z.B. ein Bild) wird auf das Objekt "gespiegelt".
- **Texture Mapping:** Bringt ein 2D-Bild (Textur) auf eine 3D-Oberfläche auf, um deren Farbe und Muster zu definieren (z.B. Ziegel auf einer Wand).
- **Bump Mapping:** Simuliert Oberflächenunebenheiten, indem es die Normalenvektoren modifiziert, was die Schattierung beeinflusst und den Eindruck von Unebenheiten erzeugt, ohne die Geometrie zu ändern.

→ trace light rays from light source(s) and store illumination on objects



Environment Mapping

Environment Mapping (Umgebungsabbildung) ist eine Technik, bei der die Umgebung eines Objekts dessen Aussehen durch Reflexionen beeinflusst. Anstatt die gesamte Umgebung explizit zu modellieren und Ray-Tracing durchzuführen, wird die Umgebung in einem Vorverarbeitungsschritt von einem zentralen Punkt aus als Bild (oder Satz von Bildern) erfasst. Die Annahme ist, dass sich jeder Oberflächenpunkt des Objekts im Zentrum dieser Umgebung befindet.

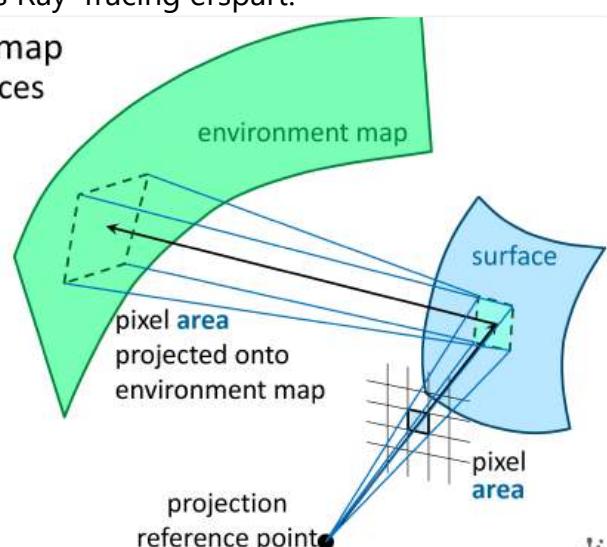
- **Vorteil:** Man kann aus der Reflexionsrichtung direkt den entsprechenden Bildteil der Umgebung abrufen, was aufwendiges Ray-Tracing erspart.

information in the environment map

- intensity values for light sources
- sky
- background objects

pixel area

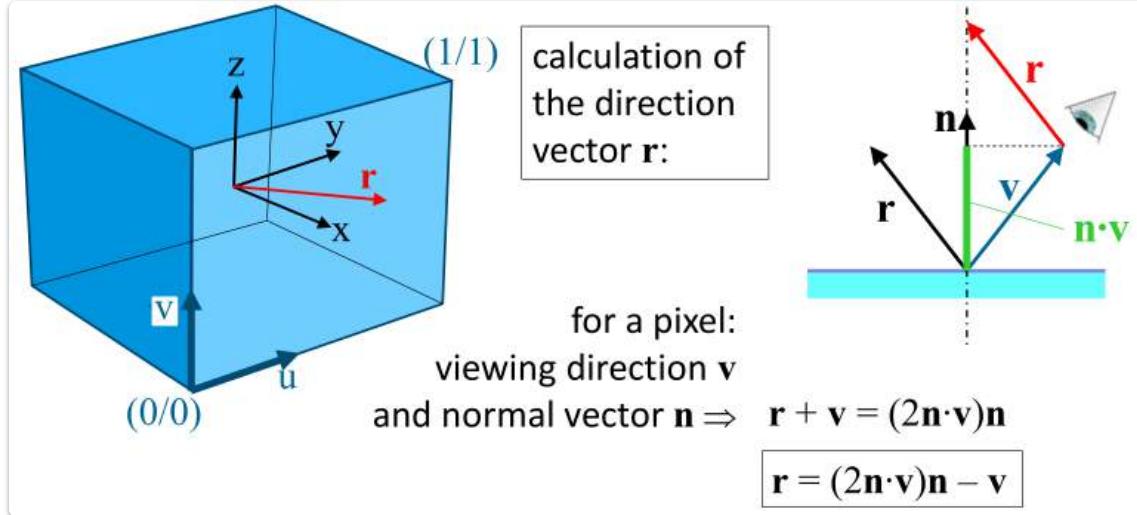
- projected onto surface
- reflected onto environment map



Cube Map als Environment Map

Die häufigste Methode ist die **Cube Map**, ein achsenparalleler Würfel, dessen sechs Seiten jeweils ein Bild der Umgebung darstellen.

- Um den Bildwert für eine Reflexionsrichtung $\mathbf{R}_S = (x_R, y_R, z_R)$ zu bestimmen, wird die Seite mit der dominantesten Komponente gewählt. Zum Beispiel für die $+x$ -Seite:
 - $u = (y_R + x_R)/(2x_R)$
 - $v = (z_R + x_R)/(2x_R)$
- Die Reflexionsrichtung \mathbf{R}_S wird dabei wie gewohnt berechnet: $\mathbf{R}_S = (2\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v}$, wobei \mathbf{v} der Vektor vom Betrachter zum Oberflächenpunkt und \mathbf{n} die Oberflächennormale ist.



Texture Mapping

Texture Mapping ist eine Technik, um komplexe Muster, Farben und Details auf Oberflächen darzustellen, auch wenn die zugrundeliegende Geometrie grob ist. Eine **Textur** ist ein Muster, das auf eine Oberfläche aufgebracht wird, oft ein Pixel-Array (Bild) oder prozedural generiert (mathematische Funktion).

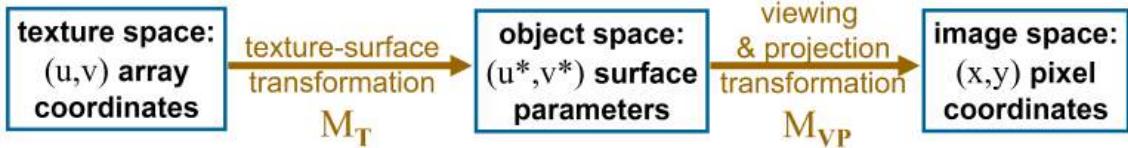
Der Prozess besteht aus zwei Schritten:

- Textur-Objekt Transformation (Modellierungsschritt):** Definiert, wie eine 2D-Textur (Koordinaten u, v) auf die 3D-Objektoberfläche (Oberflächenparameter u^*, v^*) abgebildet wird. Dies beinhaltet Skalierung, Rotation und Wiederholung der Textur. Eine bilineare Funktion kann dabei verwendet werden:
 - $u^* = u^*(u, v) = a_u u + b_u v + c_u$
 - $v^* = v^*(u, v) = a_v u + b_v v + c_v$
 Diese Funktion wird als M_T bezeichnet.
- Viewing- & Projektions-Transformation (Rendering-Schritt):** Die Textur wird auf das Abbild des Objekts im finalen Bild gerendert. Dies beinhaltet die Transformation von den Objekt-Raum-Parametern (u^*, v^*) zu den Bild-Raum-Pixelkoordinaten (x, y) .

texture mapping

forward: texture scanning $(u,v) \rightarrow (x,y)$

backward: inverse scanning $(x,y) \rightarrow (u,v)$



texture-surface transformation

$$\begin{aligned} u^* &= u^*(u,v) = a_u u + b_{u^*} v + c_{u^*} \\ v^* &= v^*(u,v) = a_v u + b_{v^*} v + c_{v^*} \end{aligned}$$

Erzeugung einer Textur

Texturen können aus verschiedenen Quellen stammen: Fotografien, Scans, programmgenerierte Texturen (inkl. Zufallswerten/Noise) oder Datenbanken.

- **Prozedurale Texturierung:** Texturen, die aus einer mathematischen Funktion gewonnen werden.
 - **Vorteile:** Kein Speicherplatz für Bilder, unendliche Detailtiefe und Parameterisierbarkeit.

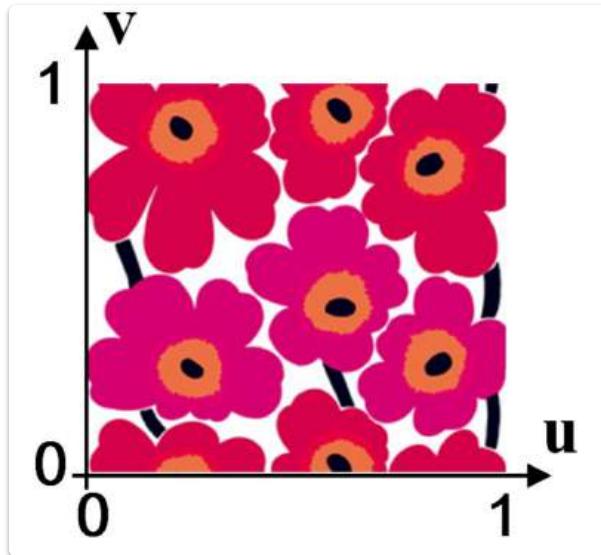
Anti-Aliasing für Texturen

Texturen sind anfällig für **Aliasing-Effekte** (Verpixelung bei Vergrößerung, Moiré-Effekte/Flackern bei Verkleinerung). Die korrekte, aber rechenintensive Lösung besteht darin, den **Textur-Durchschnittswert** der Fläche zu berechnen, die von einer Rückprojektion des zu füllenden Pixels auf die Textur erzeugt wird. Die Rückprojektion eines Pixels auf den Texturraum erzeugt eine Fläche, deren Durchschnittswert benötigt wird. Eine einfachere Annäherung ist die Verwendung des Vierecks, das durch die Verbindung der rückprojizierten Pixel-Eckpunkte entsteht.

Textur-Objekt-Transformation

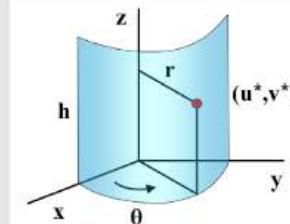
- **Ausgangssituation:** Texturen liegen in 2D- (u, v) -Koordinaten vor. 3D-Objektoberflächen haben parametrische Darstellungen (u^*, v^*) .
- **Ziel:** Abbildung von Objekt-Oberflächenparametern zu Texturkoordinaten, um für jeden Punkt auf der 3D-Oberfläche die zugehörige Texturfarbe zu finden.
- **Biliniere Funktion (M_T):** Bildet Texturkoordinaten (u, v) auf Oberflächenparameter (u^*, v^*) ab:
 - $u^* = a_u u + b_{u^*} v + c_{u^*}$
 - $v^* = a_v u + b_{v^*} v + c_{v^*}$

- Definiert Skalierung, Rotation, Scherung und Translation der Textur auf der Oberfläche.

**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:

$u^* = u \cdot \pi/2, v^* = v \cdot h$ (so passt die Textur genau auf das Zylinderviertel).



Viewing- und Projektionstransformation

- Abbildung eines 3D-Modells auf eine Bildebene ist eine Projektion M_{VP} .
- Beim Rasterscannen arbeitet man umgekehrt: Für jedes Pixel (x, y) wird der zugehörige Oberflächenpunkt und dessen Texturkoordinaten (u_*, v_*) bestimmt.
- Hierfür sind die inversen Operatoren M_{VP}^{-1} und M_T^{-1} notwendig.

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: $x = r \cdot \cos u^*, y = r \cdot \sin u^*, z = v^*$.

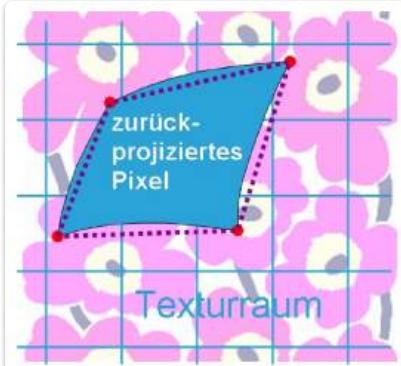
Nun wird das **Problem von hinten angegangen**:

- 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

- Problem:** Texturen sind anfällig für **Aliasing-Effekte** (Verpixelung bei Vergrößerung, Moiré-Effekte/Flackern bei Verkleinerung).

- **Korrekte, aber langsame Lösung:** Berechnung des Textur-Durchschnittswerts der Fläche, die durch die Rückprojektion eines Pixels auf die Textur entsteht.



- **Optimierungen:**

1. **Mip-Mapping:**

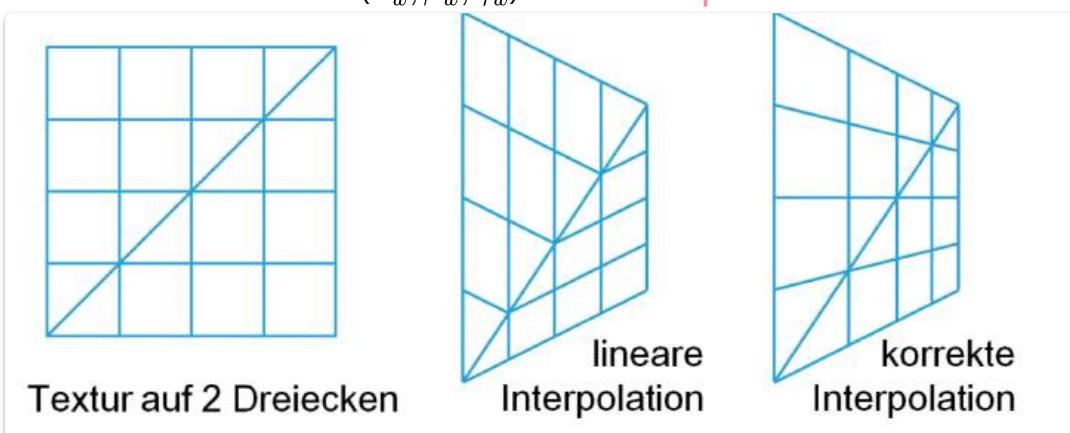
- Textur wird in **verschiedenen Auflösungen** (Mip-Map-Pyramide) vorab berechnet.
- Die passende Texturgröße wird je nach Objektverkleinerung gewählt; Interpolation zwischen Stufen für Qualität.
- Vorteil: Reduziert Aliasing bei Verkleinerung, verbessert Cache-Kohärenz.

2. **Summed-Area-Table-Methode (SAT):**

- Erstellung einer **Summen-Textur**, in der die Summe aller Texelwerte bis zu einem Punkt gespeichert ist.
- Schnelle Berechnung des Durchschnittswerts **beliebiger rechteckiger Bereiche** durch Differenzenbildung.
- Einschränkung: Nur Approximation für nicht-rechteckige rückprojizierte Pixel.

Texturen auf perspektivisch verzerrten Dreiecken

- **Problem:** Perspektivische Transformation zerstört die Linearität der Oberflächenparameter, wodurch direkte lineare Interpolation von u, v nach der Projektion zu Verzerrungen führt.
- **Lösung:** Korrekte Interpolation der Texturparameter u, v erfolgt mithilfe von **baryzentrischen Koordinaten** ($\alpha_w, \beta_w, \gamma_w$) **vor der Perspektive**.



- **Formeln für korrigierte baryzentrische Koordinaten im World-Space:**

- Gegeben baryzentrische Koordinaten α, β, γ im transformierten Raum und w -Komponenten (h_0, h_1, h_2) der Eckpunkte vor Division durch w .

$$d = h_1 h_2 + h_2 \beta(h_0 - h_1) + h_1 \gamma(h_0 - h_2)$$

$$\beta_w = h_0 h_2 \beta / d$$

$$\gamma_w = h_0 h_1 \gamma / d$$

$$\alpha_w = 1 - \beta_w - \gamma_w$$

- **Interpolation der Texturparameter:**

$$u = \alpha_w u_0 + \beta_w u_1 + \gamma_w u_2$$

$$v = \alpha_w v_0 + \beta_w v_1 + \gamma_w v_2$$

- Die finale Farbe ist $color(x, y) = t(u, v)$.

Solid Texturing

- Definiert eine Textur als **3D-Volumen** im 3-dimensionalen Parameterraum.
- Oberflächenpunkte rufen den Texturwert direkt an ihrer 3D-Position ab.
- Kann als **mathematische Funktion** (z.B. Perlin Noise) oder **Volumendaten** (Voxel) dargestellt werden.

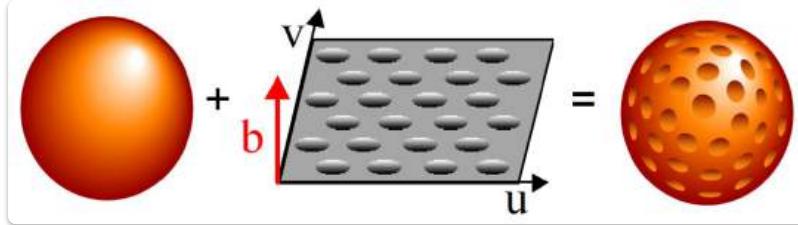


- **Vorteile:**

1. **Kohärentes Muster über Kanten hinweg:** Keine sichtbaren Nähte oder Verzerrungen, da die Textur das Objekt durchdringt.
2. **Einfachere Abbildung:** Direkte Abbildung basierend auf 3D-Positionen, keine komplexen 2D-UV-Koordinaten nötig.

Bump Mapping

- **Ziel:** Erzeugt den Eindruck von Oberflächenunebenheiten ohne tatsächliche Geometrieveränderung.
- **Funktionsweise:** Nur der **Normalvektor** der Oberfläche wird modifiziert. Dies beeinflusst die Lichtreflexion und Schattierung, wodurch Unebenheiten visuell simuliert werden.



- **Implementierung:** Benötigt Ableitungen der Höhenfunktion $b(u, v)$ nach u und v ($\frac{\partial b}{\partial u}, \frac{\partial b}{\partial v}$). Diese werden oft direkt in einer **Normal Map** gespeichert.

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:

$$n' = p'_u \times p'_v$$

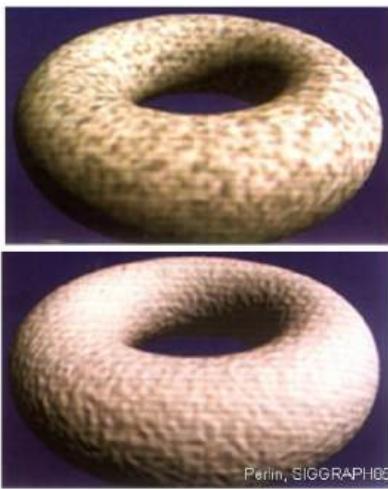
Nun gilt:

$$p'_u = \partial(p + b_n)/\partial u = p_u + b_u n + b n_u$$

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:

$$n' = p'_u \times p'_v = p_u \times p_v + b_v(p_u \times n) + b_u(n \times p_v) + b_u b_v(n \times n)$$

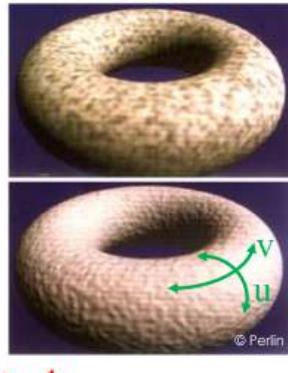
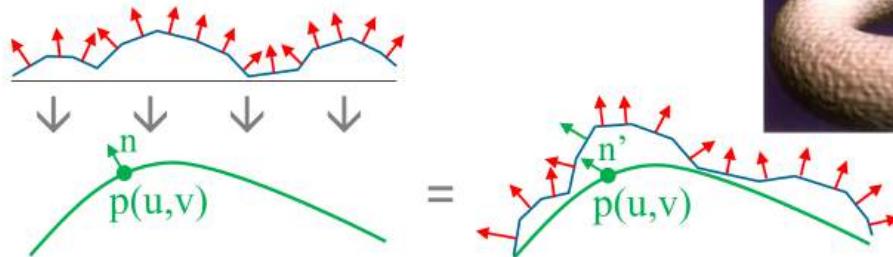
und aus $n \times n = 0$ folgt schließlich: $n' = n + b_v(p_u \times n) + b_u(n \times p_v)$.



- **Schattierung:** Der räumliche Eindruck entsteht durch richtungsabhängige Schattierung mit den modifizierten Normalen.

surface roughness is simulated
perturbation function varies surface normal *locally*

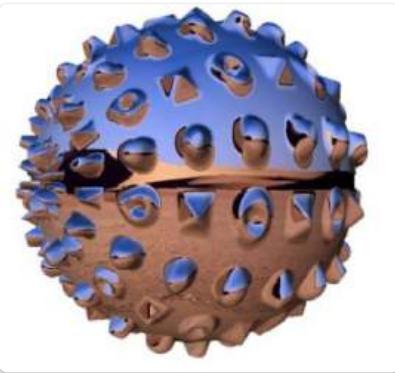
bump map $b(u,v)$ derivative $b'(u,v)$

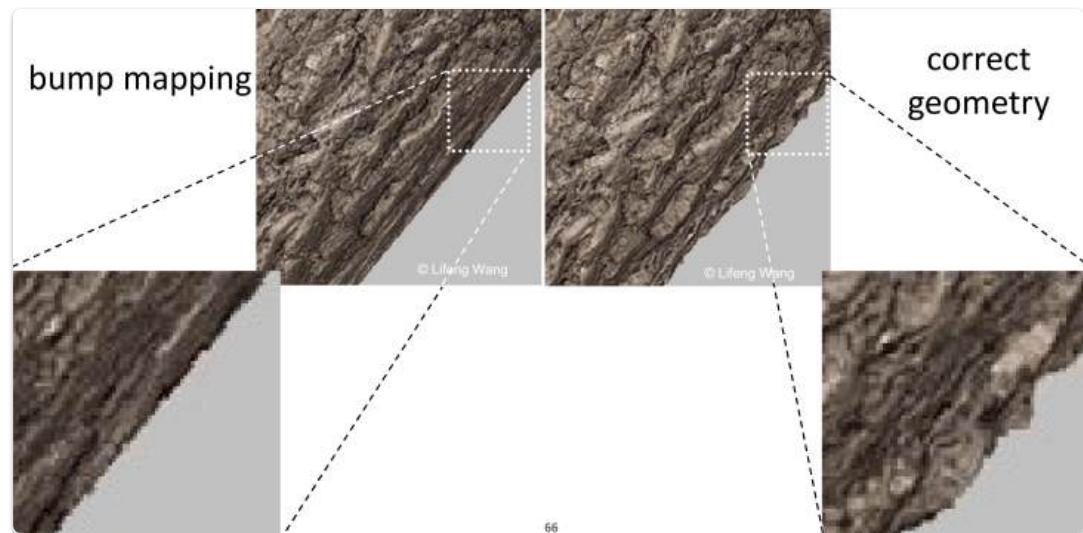


- Einschränkungen/Nachteile:
 1. Verzerrung bei flachen Winkeln.
 2. Glatte Silhouette und glatte Schattenränder.
 3. Keine gegenseitigen Schatten der Bumps.
 4. Falsche Beleuchtung auf geometrisch schattierten Bereichen.

Displacement Mapping

- Korrekteste Methode für Oberflächenunebenheiten.
- Funktionsweise: Die Oberflächenpunkte werden tatsächlich verschoben, die Geometrie wird also physisch verändert.
- Vorteile:
 - Korrekte Silhouette und korrekte Schatten (inkl. gegenseitiger Schatten der Unebenheiten).

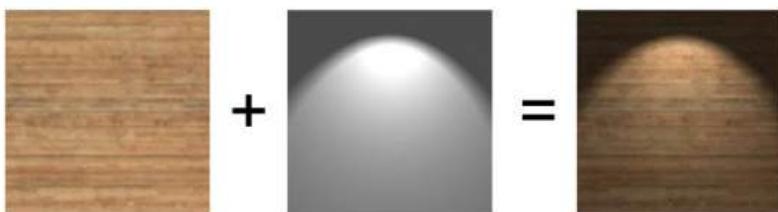




- **Implementierung:** Rechenintensiver als Bump Mapping. Moderne GPUs unterstützen dies via **Tessellation-Stage** (zwischen Vertex- und Pixel-Stage), die Dreiecke unterteilt und dann entsprechend einer Displacement Map verschiebt.

Kombination mehrerer Mappings

- **Multitexturing** ermöglicht die Kombination mehrerer Mappings.
- Beispiele: Grundmuster, Beleuchtung, Verschmutzung, Unebenheiten, Fotos + Annotationen.



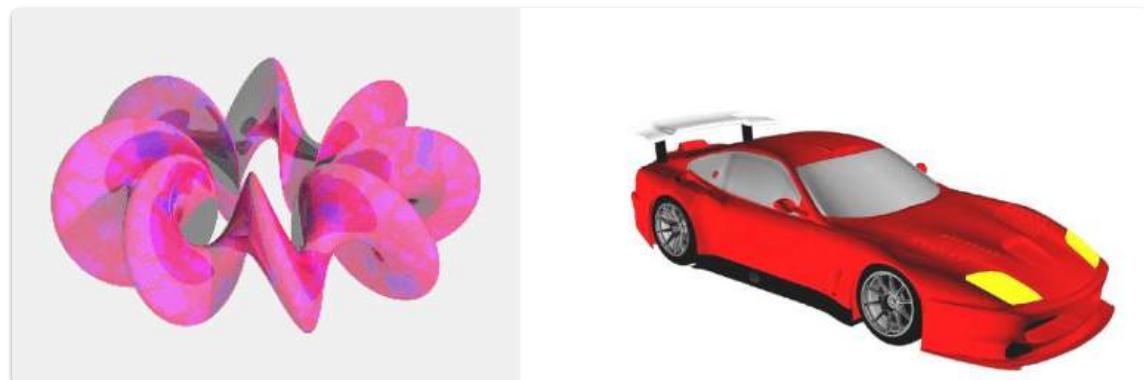
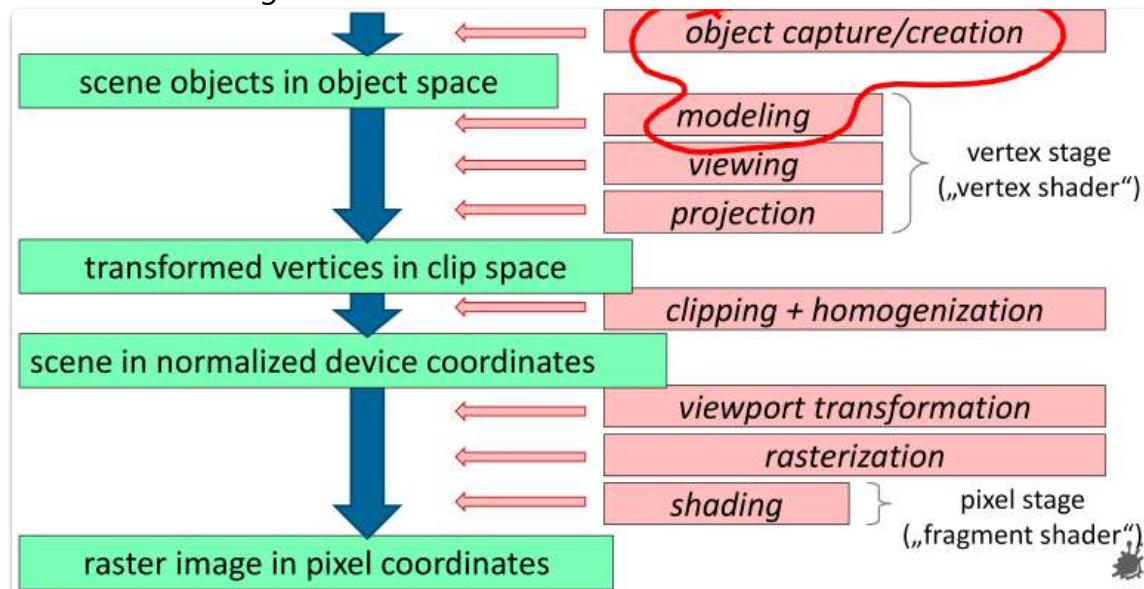
12. Kurven und Flächen

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Freiformflächen (Modellierung und Darstellung)

- **Definition:** Beliebige, allgemeine Flächen, die über elementare geometrische Formen hinausgehen.
- **Prinzip:** Oft von Freiformkurven abgeleitet, da die Mathematik einfacher ist und sich Verfahren übertragen lassen.



Quadratische Flächen (Analytische Darstellung)

Flächen definiert durch mathematische Formeln:

1. Implizite Repräsentation:

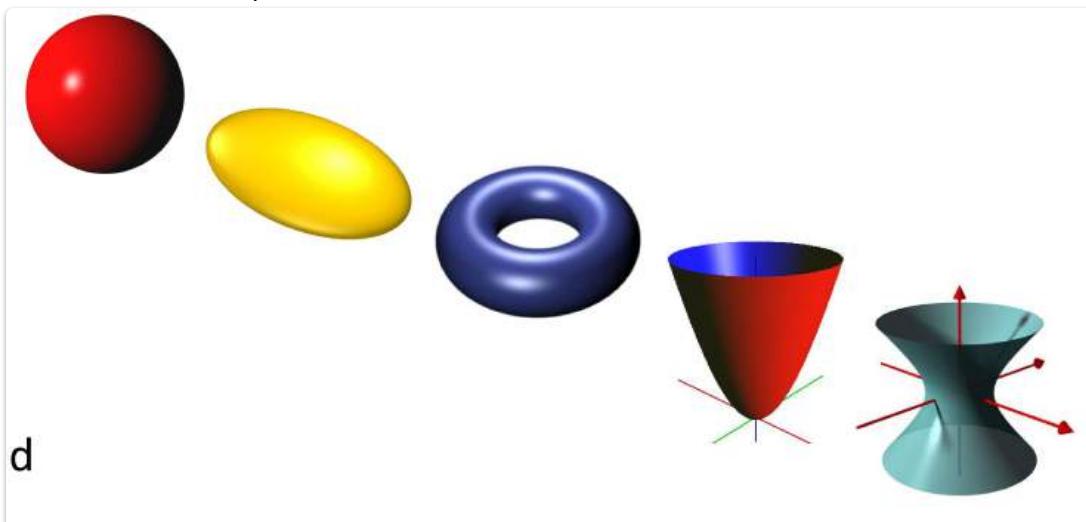
- Alle Punkte (x, y, z) auf der Oberfläche erfüllen eine Formel (z.B. Kugel):
 $x^2 + y^2 + z^2 = r^2$.
- "Filterfunktion": Prüft, ob ein Punkt auf der Oberfläche liegt.

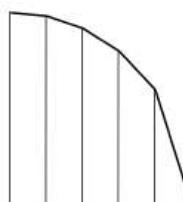
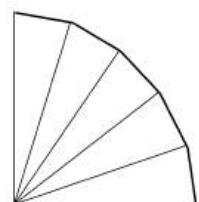
2. Explizite Repräsentation:

- Ein Koordinatenwert (z.B. z) ist eine Funktion der anderen (z.B. x, y).
- Kugel: $z = \sqrt{r^2 - x^2 - y^2}$.
- Funktioniert nur für Oberflächen, die für (x, y) genau einen z -Wert haben.

3. Parametrische Repräsentation:

- Koordinaten (x, y, z) sind Funktionen von **Parameterwerten** (z.B. u, v).
- Kugel:
 - $x = r \cdot \cos \phi \cdot \cos \theta$
 - $y = r \cdot \cos \phi \cdot \sin \theta$
 - $z = r \cdot \sin \phi$
- Sehr flexibel, kann komplexe Formen darstellen.
- Weitere quadratische Flächen: **Ellipsoid, Torus, Quadrics** (allgemeine Bezeichnung für Flächen 2. Grades).

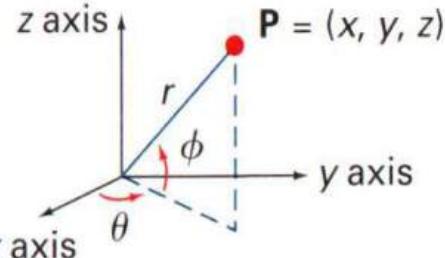
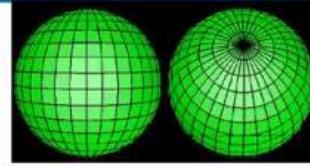


$y = f(x)$ <i>axis dependent</i>	$x = f(u)$ $y = g(u)$ <i>axis independent</i>
<i>example:</i> $y = \sqrt{1-x^2}$ 	$x = \cos(u)$ $y = \sin(u)$ 

Quadric Surfaces: Sphere

- **implicit:** $x^2 + y^2 + z^2 = r^2$

- **parametric:** $x = r \cos \phi \cos \theta, -\pi/2 \leq \phi \leq \pi/2$
 $y = r \cos \phi \sin \theta, -\pi \leq \theta \leq \pi$
 $z = r \sin \phi$



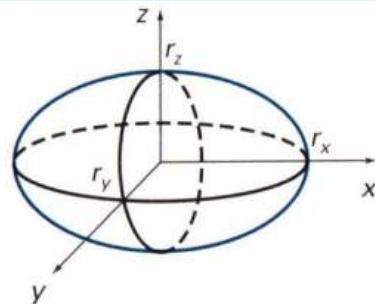
Werner Purgathofer



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:**

$$\begin{aligned} x &= r_x \cos \phi \cos \theta, & -\pi/2 \leq \phi \leq \pi/2 \\ y &= r_y \cos \phi \sin \theta, & -\pi \leq \theta \leq \pi \\ z &= r_z \sin \phi \end{aligned}$$

Quadric Surfaces: Torus

- **implicit:**

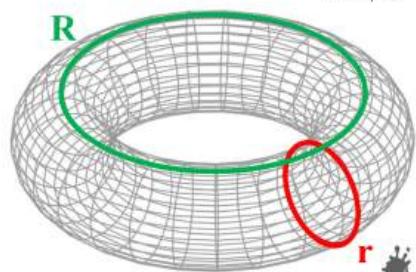
$$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$$



© Wikipedia

- **parametric:**

$$\begin{aligned} x &= (R + r \cos \phi) \cos \theta, & -\pi \leq \phi \leq \pi \\ y &= (R + r \cos \phi) \sin \theta, & -\pi \leq \theta \leq \pi \\ z &= r \sin \phi \end{aligned}$$



Werner Purgathofer

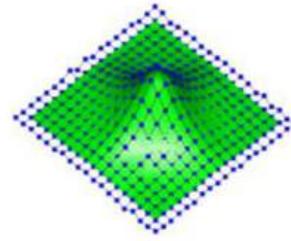
9

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!

$$x = f(u,v)$$

$$y = g(u,v)$$

$$z = h(u,v)$$

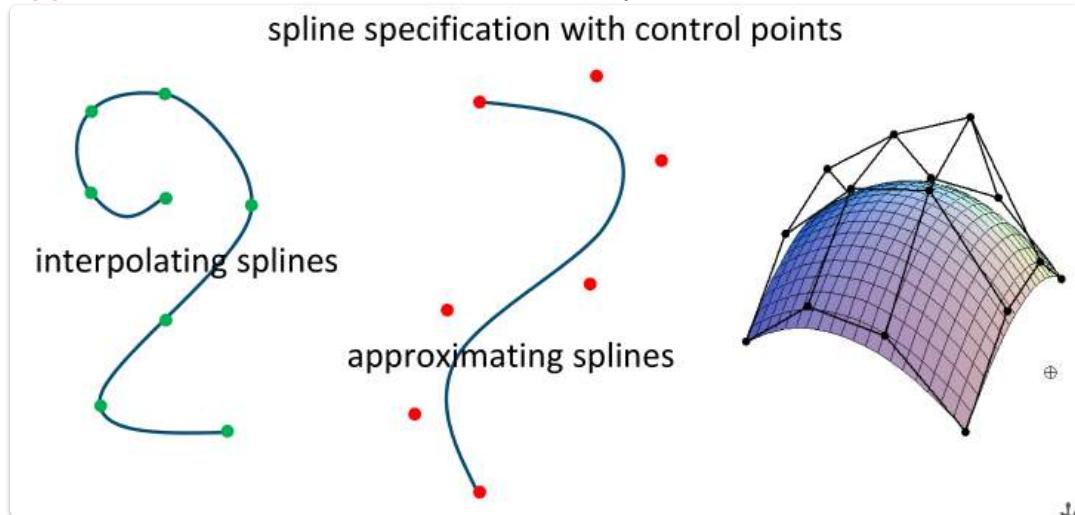
Kurven

Definition durch:

1. **Analytisch (Formelbasiert):** Weniger intuitiv.
2. **Punktbasiert (Stütz-/Kontrollpunkte):** Gängiger und benutzerfreundlicher (-> **Splines**).

Unterscheidende Merkmale von Kurventypen:

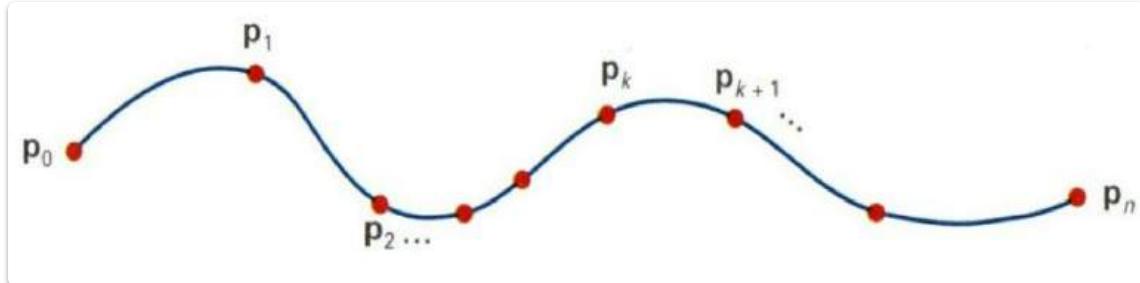
- **Interpolation vs. Approximation:**
 - **Interpolierend:** Kurve geht *durch* Stützpunkte.
 - **Approximierend:** Kurve *nähert sich* Kontrollpunkten.



- **Stetigkeit an Verbindungsstellen:** Glätte der Übergänge (C^1, C^2).
- **Einflussbereich von Punkten:**
 - **Global:** Änderung eines Punktes beeinflusst gesamte Kurve.
 - **Lokal:** Änderung beeinflusst nur begrenzten Abschnitt.
- **Abhängigkeit vom Koordinatensystem:** Achsenabhängig vs. Achsenunabhängig.
- **Verhalten bei Richtungswechseln:** Dämpfung vs. Überschwingen.
- **Morphologische Eigenschaften:** Mögliche Formen, Schleifen, Geschlossenheit.

Kubische Spline-Interpolation

- **Basis:** Interpolationskurve aus kubischen Polynomen zwischen $n + 1$ Stützpunkten p_i .
- **Formel:** $p_k(u) = a_k u^3 + b_k u^2 + c_k u + d_k$ für $0 \leq u \leq 1$. (a_k, \dots, d_k sind Vektoren).



- **Eigenschaften:** C^1 - und C^2 -stetige Verbindung ("natürliche kubische Splines").
- **Nachteil:** **Globaler Einfluss** – jeder Stützpunkt beeinflusst den gesamten Kurvenverlauf.

Hermite-Interpolation

- **Spezialfall der kubischen Splines.**
- **Bestimmung:** Neben den Stützpunkten p_k werden auch die **Ableitungen** (Tangenten) Dp_k an den Stützpunkten vorgegeben.
- **Bedingungen für ein Kurvenstück $p_k(u)$ zwischen p_k und p_{k+1} :**
 - $p_k(0) = p_k$
 - $p_k(1) = p_{k+1}$
 - $p'_k(0) = Dp_k$
 - $p'_{k+1}(1) = Dp_{k+1}$

$$p_k(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix} \quad p'_k(u) = [3u^2 \quad 2u \quad 1 \quad 0] \cdot \begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix}$$

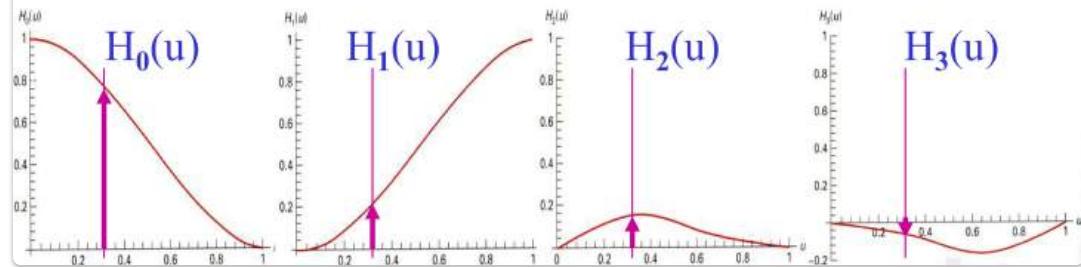
$$\begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix}$$

Um die Koeffizientenvektoren a_k, b_k, c_k, d_k von $a_k u^3 + b_k u^2 + c_k u + d_k$ zu berechnen, invertiert man diese Matrix. Die resultierende Matrix heißt Hermite-Matrix M_H :

$$\begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

$$p_k(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot M_H \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

$H_k(u)$ blending functions:



Bézier-Kurven

- Approximierende Kurven (geht nicht durch alle Kontrollpunkte).
- Verwendet Bernstein-Polynome $b_{k,n}(u)$ als Gewichtsfunktionen.
- Formel:

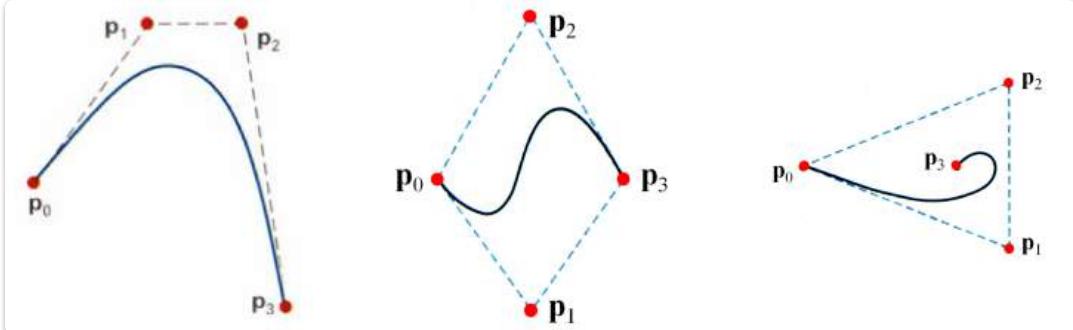
$$p(u) = \sum_{k=0}^n p_k b_{k,n}(u)$$

mit

$$b_{k,n}(u) = \binom{n}{k} u^k (1-u)^{n-k}$$

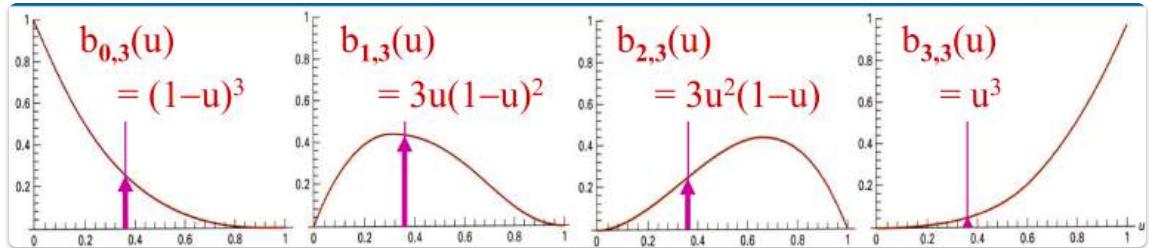
- $p(u)$ ist gewichtetes Mittel aller Kontrollpunkte.
- Beispiel $n = 3$ (4 Kontrollpunkte):

$$p(u) = (1-u)^3 p_0 + 3u(1-u)^2 p_1 + 3u^2(1-u)p_2 + u^3 p_3.$$



Eigenschaften:

- Grad n bei $n+1$ Kontrollpunkten.
- **Globaler Einfluss:** Änderung eines Kontrollpunkts beeinflusst die gesamte Kurve.
- **Endpunkte:** p_0 und p_n liegen auf der Kurve.
- **Tangenten:** In p_0 und p_n durch Verbindung zu p_1 und p_{n-1} .
- **Konvexe Hülle:** Kurve liegt vollständig in der konvexen Hülle ihrer Kontrollpunkte.



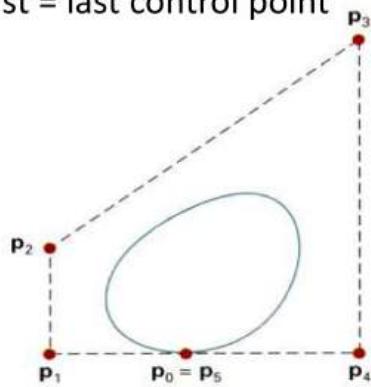
Bézier Curves Design Techniques (1)



a *closed Bézier curve*

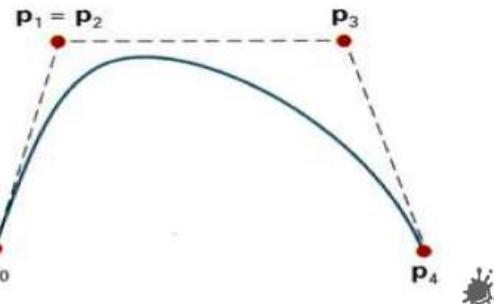
generated by setting:

first = last control point



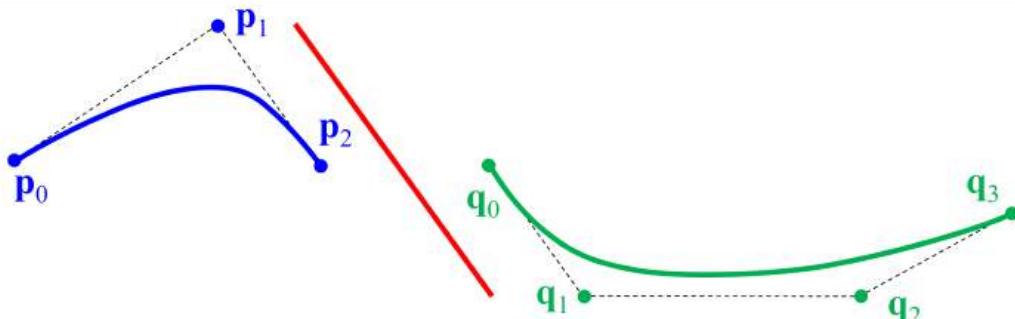
Werner Purgathofer

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 1st-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

$$\mathbf{p}(u) = (1-u)^3 \cdot \mathbf{p}_0 + 3u(1-u)^2 \cdot \mathbf{p}_1 + 3u^2(1-u) \cdot \mathbf{p}_2 + u^3 \cdot \mathbf{p}_3$$

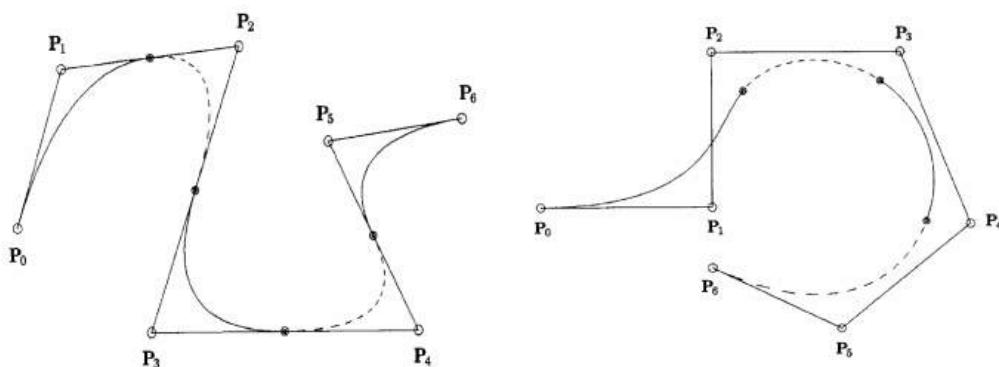
$$\mathbf{p}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot M_{\text{Bez}} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad \text{with} \quad M_{\text{Bez}} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

B-Spline-Kurven

- **Vorteil gegenüber Bézierkurven:** Überwinden den globalen Einfluss.
- Ebenfalls **approximierende Kurven**.
- Verwenden **B-Spline-Polynome $B_{k,d}$** anstelle von Bernstein-Polynomen.
- **Eigenschaft:** Beschränken die Anzahl der Kontrollpunkte, die einen Kurvenpunkt beeinflussen, auf den Grad d .
 - **Lokaler Einfluss** der Kontrollpunkte.
 - Berechnung ist linear zur Anzahl der Kontrollpunkte.
- **Summe der Gewichtsfunktionen:**

$$\sum_{k=0}^n B_{k,d}(u) = 1$$

- **Spezialfall:** Wenn $d = n + 1$, sind B-Splines identisch mit Bézier-Kurven.



Influence of d

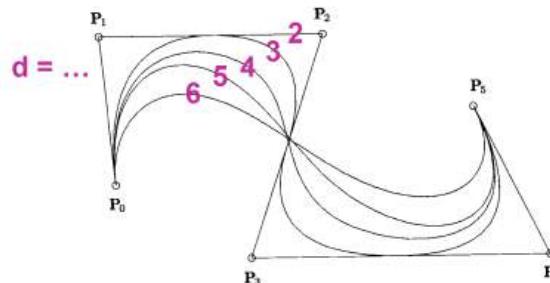
d describes, how many control points influence any point on the curve

$d = 2$ linear

$d = 3$ quadratic

$d = 4$ cubic

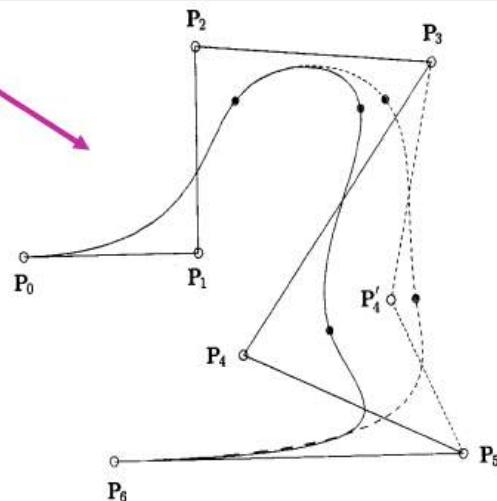
...



for $d=n+1$ you get Bézier curves!

control points have local influence

effort is linearly dependent on n ,
therefore splitting of huge point
sets not necessary

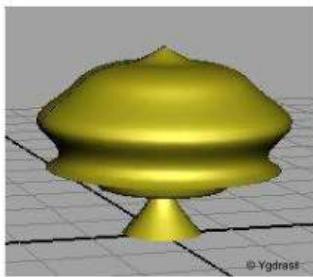


NURBS (Non-Uniform Rational B-Splines)

- Wichtige Erweiterung der B-Splines.
- Können auch **regelmäßige geometrische Formen** konsistent repräsentieren.
- Alle Splines (Bézier, B-Splines, NURBS) beschreiben **räumliche Kurven im dreidimensionalen Raum**.

further extension: **Non-Uniform Rational B-Splines = "NURBS"**

allow to combine freeform surfaces with regular surfaces



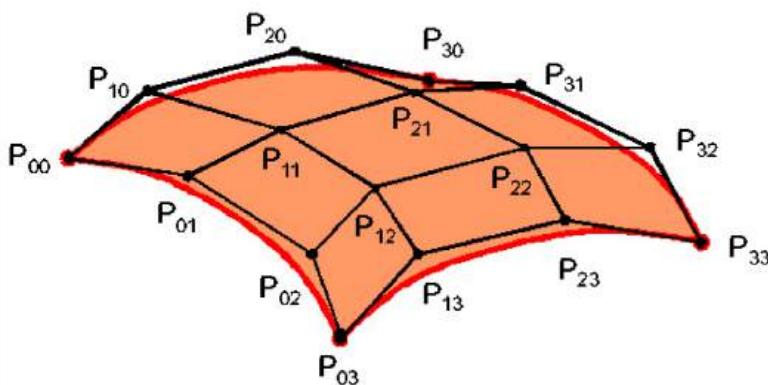
Freiformflächen -- Bézier- und B-Spline-Flächen

- Entstehen durch kartesisches Produkt zweier Kurvenscharen über einem zweidimensionalen Punkteraster.

- **Formel:**

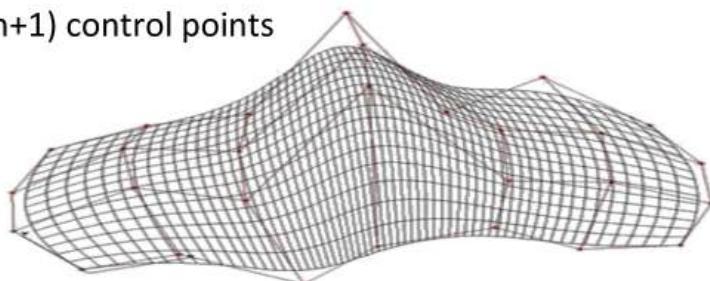
$$p(u, v) = \sum_{j=0}^m \sum_{k=0}^n P_{j,k} b_{j,m}(v) b_{k,n}(u)$$

- $P_{j,k}$ sind Kontrollpunkte im Raster.
- Randkurven sind die jeweiligen Kurven (z.B. Bézierkurven für Bézierflächen).
- Eigenschaften der Kurven übertragen sich auf die Flächen.



$P_{j,k}$: grid of $(m+1) \times (n+1)$ control points

just like for
Bezier surfaces!

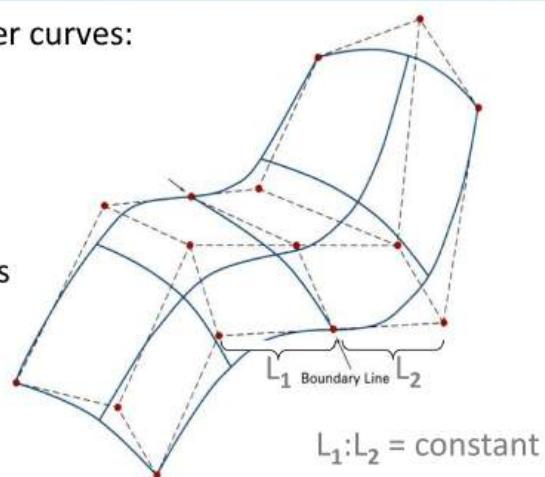


Bézier Surfaces Properties



have the same properties as Bézier curves:

- global influence
- interpolates corner points
- tangents at corner points
- convex hull property
- 1st-order continuity connections



$$L_1:L_2 = \text{constant}$$

Zeichnen von Freiformflächen:

- **Dreiecksnetze** aus den Flächen erzeugen und rendern.
- **Ray-Casting-Methoden:** Schnittpunkt einer Geraden mit der Fläche berechnen und Oberflächennormale zurückliefern.

13. Computer Animation

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Transformationen

Affine Transformationen (Translation & Rotation) können als [homogene \$4 \times 4\$ Matrizen](#) dargestellt werden:

$$\begin{pmatrix} R & x \\ 0 & 1 \end{pmatrix}$$

mit $R \in \mathbb{R}^{3 \times 3}$ (Rotation) und $x \in \mathbb{R}^3$ (Translation).

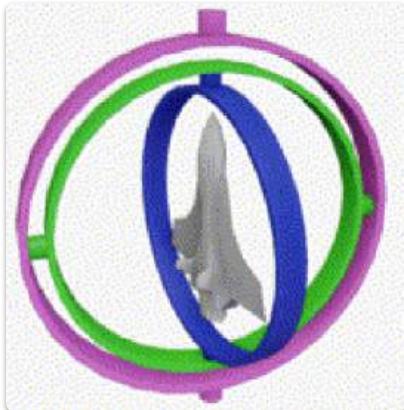
Translation

- [Lineare Interpolation](#) der Position $p^{(t)}$ zwischen Start ($p^{(t_0)}$) und Ziel ($p^{(t_1)}$) über die Zeit t :

$$p^{(t)} = p^{(t_0)} + (p^{(t_1)} - p^{(t_0)}) \frac{t - t_0}{t_1 - t_0}$$

Rotation

- [Problem mit Rotationsmatrizen](#): Instabil, führen zu [Gimbal Lock](#) (Verlust eines Freiheitsgrades).



- [Lösung: Quaternionen und sphärische lineare Interpolation \(slerp\)](#).
- [Quaternion](#): $q = [s; v] = [\cos \frac{\phi}{2}; \sin \frac{\phi}{2} n]$ (Achsenwinkel-Darstellung: Winkel ϕ um Achse n).
- [Interpolation](#): $q^{(t)} = \text{slerp}(q^{(t_0)}, q^{(t_1)}, t)$.
- [Anwendung auf Punkt \$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**



$$\mathbf{q} = [s; \mathbf{v}] = [\textcolor{purple}{s} \quad \begin{matrix} x & y & z \end{matrix}]$$

scalar
vector

- Generalization of complex numbers
- Also written as:

$$\mathbf{q} = \textcolor{purple}{s} + \textcolor{teal}{x}i + \textcolor{teal}{y}j + \textcolor{teal}{z}k$$

with

$$i^2 = j^2 = k^2 = -1$$

$$i \cdot j \cdot k = -1$$

- **Quaternions:** $\mathbf{q} = [s; \mathbf{v}] = [\textcolor{purple}{s} \quad \textcolor{teal}{x} \quad \textcolor{teal}{y} \quad \textcolor{teal}{z}]$



- Operations:

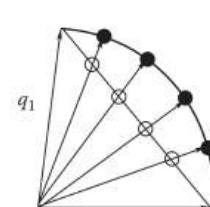
- Addition: $\mathbf{q}_1 + \mathbf{q}_2 = [(\textcolor{purple}{s}_1 + s_2) \quad (x_1 + x_2) \quad (y_1 + y_2) \quad (z_1 + z_2)]$

- Multiplication: $\mathbf{q}_1 \cdot \mathbf{q}_2 = \begin{bmatrix} (\textcolor{purple}{s}_1 \cdot s_2 - x_1 \cdot x_2 - y_1 \cdot y_2 - z_1 \cdot z_2) \\ (\textcolor{purple}{s}_1 \cdot s_2 + x_1 \cdot x_2 + y_1 \cdot y_2 - z_1 \cdot z_2) \\ (\textcolor{purple}{s}_1 \cdot s_2 - x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2) \\ (\textcolor{purple}{s}_1 \cdot s_2 + x_1 \cdot x_2 - y_1 \cdot y_2 + z_1 \cdot z_2) \end{bmatrix}$

- Inverse of normalized Quaternion: $\mathbf{q}^{-1} = \bar{\mathbf{q}} = [\textcolor{purple}{s} \quad -\textcolor{teal}{x} \quad -\textcolor{teal}{y} \quad -\textcolor{teal}{z}]$

- Can be computed from angle ϕ and normalized vector \mathbf{n} (*axis-angle representation*):

$$\mathbf{q} = \left[\cos \frac{\phi}{2}; \sin \frac{\phi}{2} \mathbf{n} \right]$$



- Spherical Interpolation:

$$\mathbf{q}^{(t)} = \textit{slerp}(\mathbf{q}^{(t_0)}, \mathbf{q}^{(t_1)}, t) = \mathbf{q}^{(t_0)}(\mathbf{q}^{(t_0)^{-1}} \mathbf{q}^{(t_1)})^t$$

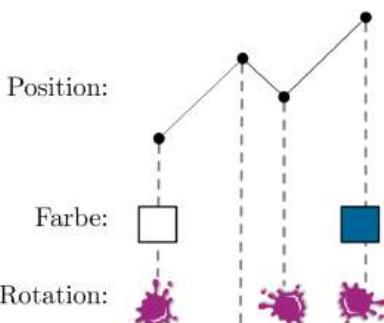
Figure taken from Shirley&Manzner:
Foundations of Computer Graphics, 5th Edition

■ Apply a rotation to a point $p^{(t)}$:

$$[0; p^{(t)}] = q^{(t)} \cdot [0; p^{(t_0)}] \cdot q^{(t)^{-1}}$$

Keyframing

- **Konzept:** Szenenparameter werden nur zu bestimmten Zeitpunkten (**Keyframes**) festgelegt und dazwischen interpoliert.
- **Keyframe k :** $(t_k, f^{(t_k)}) = (t_k, (p^{(t_k)}, c^{(t_k)}, q^{(t_k)}))$ (Zeit, Position, Farbe, Rotation).

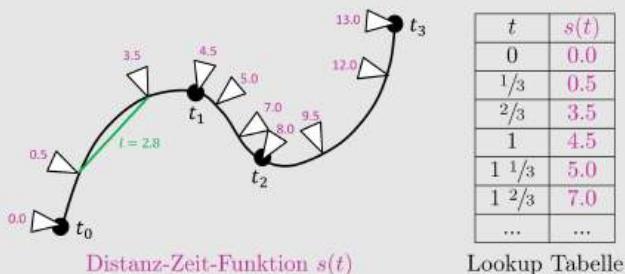


Zeit	0	1	2	3	4	5
Position	●		●	●		●
Farbe	●					●
Rotation	●			●		●

- **Ziel:** Flüssige Bewegung.
- **Problem:** Ungleichmäßige räumliche Verteilung der Punkte bei gleichmäßiger Zeitabtastung.
- **Lösung: Distanz-Zeit-Funktion** $s(t)$, die die zurückgelegte Entfernung zuordnet.
 - $s(t_i) \approx s(t_{i-1}) + \|P_i - P_{i-1}\|$
- Durch ungleichmäßiges Abtasten der Zeit über eine **Lookup-Zeit-Tabelle** und lineare Interpolation wird 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 $t = 0.\bar{6}$ ist rechts, wobei die **euklidische Distanz** zwischen $P_{0,\bar{3}}$ und $P_{0,\bar{6}}$ verwendet wird, von der wir ebenfalls annehmen, dass sie gegeben ist.



t	$s(t)$
0	0.0
$1/3$	0.5
$2/3$	3.5
1	4.5
$1\frac{1}{3}$	5.0
$1\frac{2}{3}$	7.0
...	...

Lookup Tabelle

Approximiere $s(t)$:

$$s(t_{0,\bar{6}}) \approx s(t_{0,\bar{3}}) + l = 0.5 + 2.8 = 3.3$$

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.

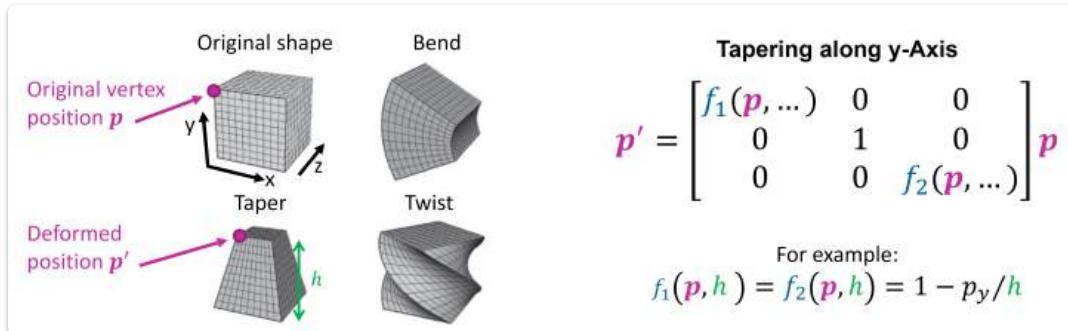
Als nächstes **tasten wir $s(t)$ gleichmäßig ab**, z.B. bei $0.0, 1.0, 2.0, \dots$ und berechnen mittels linearer Interpolation die Werte t , an denen wir unsere Kurve auswerten wollen. Zum Beispiel, wenn wir die Position eines Punktes entlang der Kurve nach der Distanz $s = 6.0$ berechnen wollen, benötigen wir dafür:

$$t = \frac{1.\bar{3} + 1.\bar{6}}{2} = 2.5, \quad \text{da } 6.0 \text{ in der Mitte von } 5.0 \text{ und } 7.0 \text{ ist (siehe Tabelle).}$$

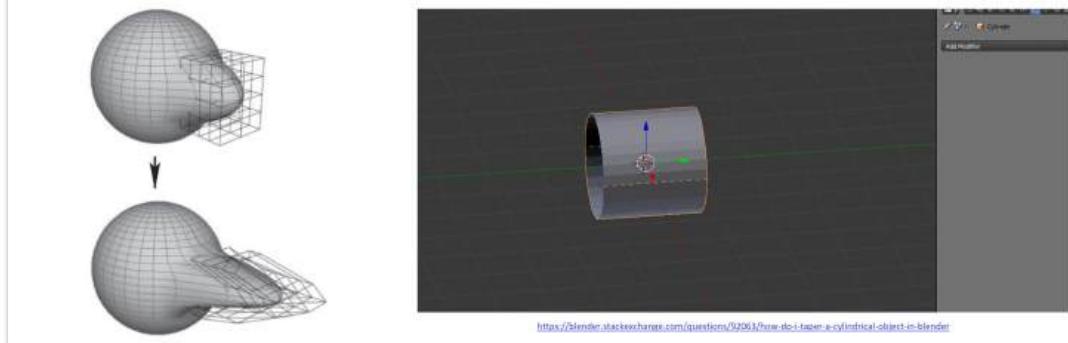
Deformationen

- **Definition:** Funktion f , die Vertex-Positionen p in neue, deformierte Positionen $p' = f(p, \gamma)$ überführt.

- **Einfache Deformationen:** Nicht-konstante Matrizen (z.B. Verjüngung, Biegung, Verdrehung).
- **Komplexere Deformationen: Deformationskäfige.**
 - Objekt-Vertices sind an den Käfig gebunden.
 - Verschieben des Käfigs deformaert das Objekt.
 - Reduziert manuellen Aufwand.



Free Form Deformation: Define f wrt. a deformation cage



Physikbasierende Simulation

- Simulation von Objekten basierend auf physikalischen Gesetzen.
- Objekte werden durch Punkte beschrieben, die über **Constraints** verbunden sind.

Punkt-Parameter:

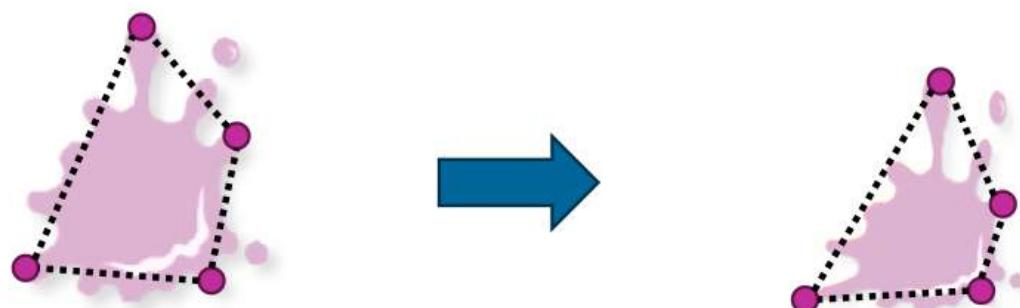
- **Position** p
- **Geschwindigkeit** $v = \frac{dp}{dt}$
- **Beschleunigung** $a = \frac{d^2p}{dt^2}$
- **Masse** m

Einschränkungen (Constraints):

- **Starr (rigid):** Distanz/Volumen bleibt gleich (z.B. Holzwürfel).
- **Weich (soft):** Distanz/Volumen variabel (z.B. Kleidung).



- Distance stays the same
- Transformations only

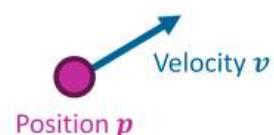


- Distance can change
- Transformations + Deformations

Bewegung bei konstanter Geschwindigkeit:

- Newtons erstes Gesetz: $p^{(t+\Delta t)} = p^{(t)} + \Delta t \cdot v$.
- Represent by multiple points
- Velocity v is the change of position over time

$$v = \frac{dp}{dt}$$



- If there is *constant* velocity, we can thus change the position via:

$$p^{(t+1)} = p^{(t)} + \frac{dp^{(t)}}{dt} = p^{(t)} + v$$

Änderung der Geschwindigkeit:

- Newtons zweites Gesetz: Kraft F führt zu Beschleunigung $a = \frac{F}{m}$.

- If there is *varying* velocity and a *constant* Force:

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t)} + \frac{1}{2} (\Delta t)^2 \mathbf{a}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \mathbf{a}$$

$$\mathbf{a} = \frac{\mathbf{F}}{m}$$

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \int_t^{t+\Delta t} \mathbf{v}^{(t')} dt'$$

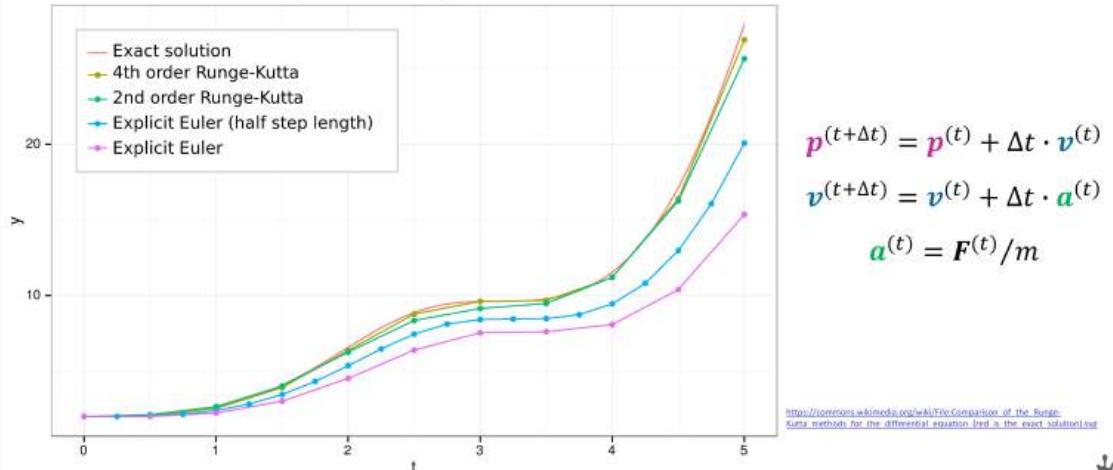
$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \int_t^{t+\Delta t} \mathbf{a}^{(t')} dt'$$

Hard to compute
⇒ use numerical integration methods!

$$\mathbf{a}^{(t)} = \mathbf{F}^{(t)}/m$$

- Berechnung von $\mathbf{p}^{(t+\Delta t)}$ bei variabler Kraft/Geschwindigkeit erfordert Integration.

- 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:

- Verfahren zur annähernden Berechnung von Integralen in der Simulation.

<u>Explicit Euler:</u>	<u>Implicit Euler:</u>	<u>Semi-Implicit Euler:</u>
$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t)}$	$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t+\Delta t)}$	$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t+\Delta t)}$
$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t)}$	$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t+\Delta t)}$	$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t)}$
$\mathbf{a}^{(t)} = \frac{\mathbf{F}^{(t)}}{m}$	$\mathbf{a}^{(t+\Delta t)} = \frac{\mathbf{F}^{(t+\Delta t)}}{m}$	$\mathbf{a}^{(t)} = \frac{\mathbf{F}^{(t)}}{m}$
unstabil bei großen Zeitschritten (hohe Abweichung/Oszillationen), einfache Berechnung, real-time	bedingungslos stabil, aufwändige Berechnung (Lösung eines Gleichungssystems), offline	energiebewahrend, einfache Berechnung, real-time

■ 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**

$$\mathbf{p}^{(t+\Delta t)} = \mathbf{p}^{(t)} + \Delta t \cdot \mathbf{v}^{(t+\Delta t)}$$

$$\mathbf{v}^{(t+\Delta t)} = \mathbf{v}^{(t)} + \Delta t \cdot \mathbf{a}^{(t)}$$

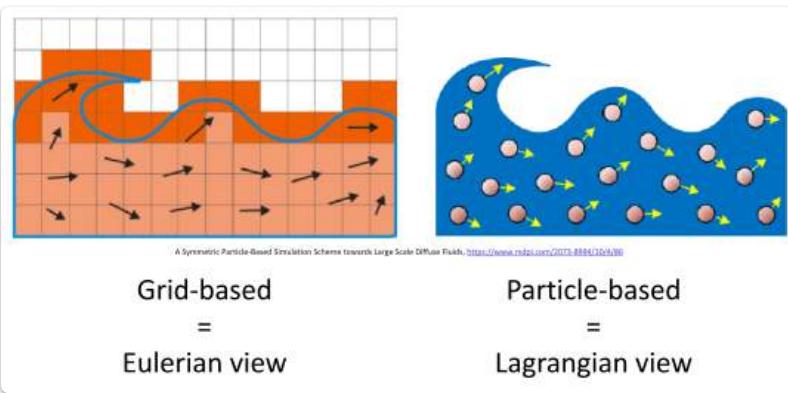
$$\mathbf{a}^{(t)} = \mathbf{F}^{(t)} / m$$

```
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

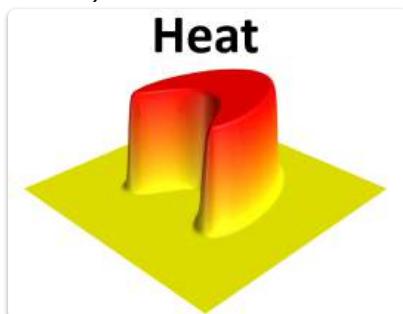
- **Partikelbasierte Simulationen (Lagrange-Sichtweise):** Form wird durch Punkte angenähert.
- **Gitterbasierte Simulationen (Euler-Sichtweise):** Raum wird durch Gitter unterteilt.



- **Differentialgleichungen:** Gleichungen, die eine Funktion und ihre Ableitungen enthalten.

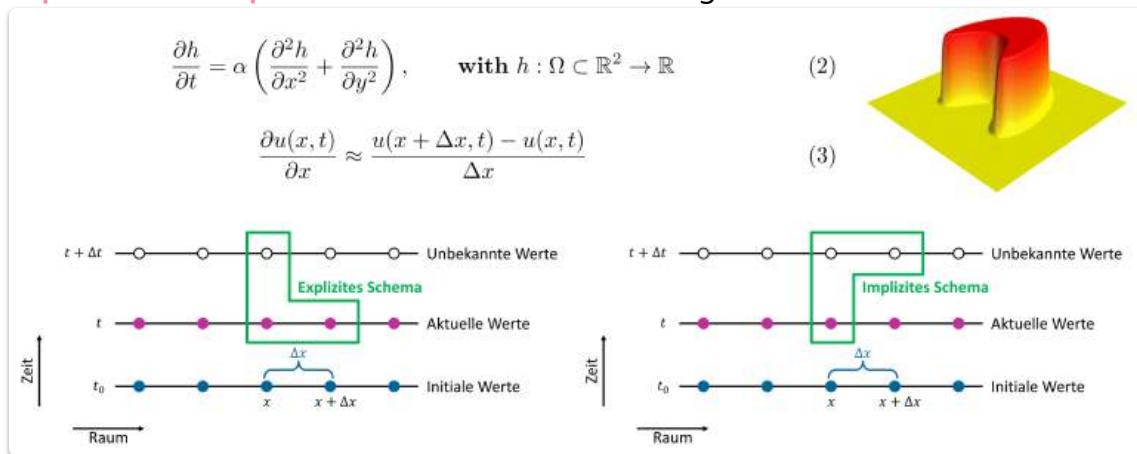
Arten von Differentialgleichungen:

- **Gewöhnliche Differentialgleichungen (ODEs):** Enthalten nur vollständige Ableitungen (z.B. Newtons zweites Gesetz).
- **Partielle Differentialgleichungen (PDEs):** Beschreiben mehrdimensionale Funktionen mit partiellen Ableitungen (z.B. Wärmeleitungsgleichung, Navier-Stokes-Gleichung für Fluide).



Numerische Berechnung:

- **Finite-Differenzen-Verfahren** zur Berechnung von Ableitungen.
- **Explizite** oder **implizite Verfahren** zur Berechnung unbekannter Werte.



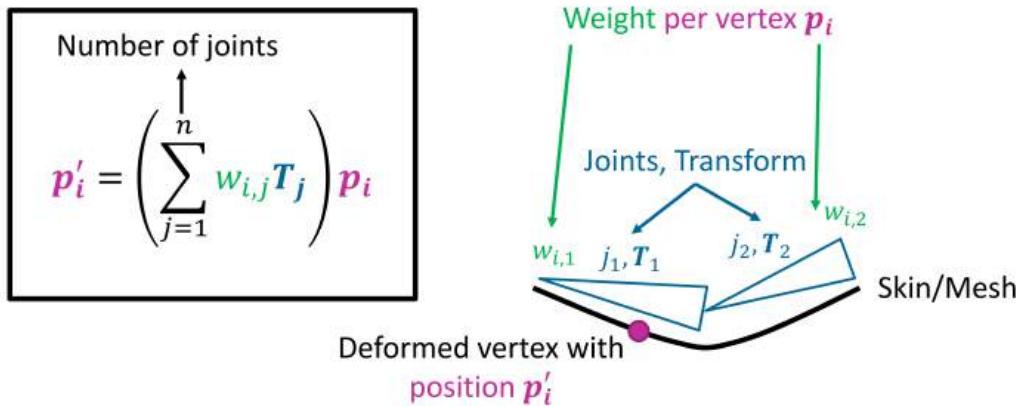
Charakter Animation

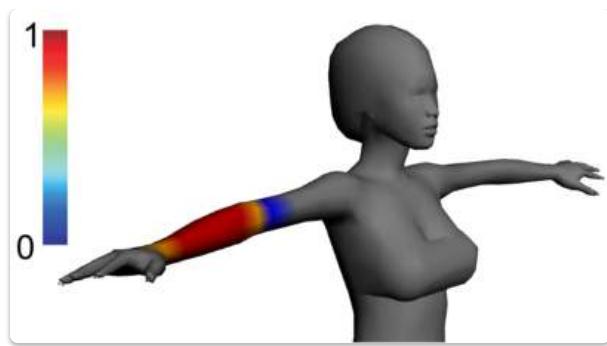
Linear Blend Skinning (LBS)

- **Methode:** Einfache Methode zur Deformation von Charakteren basierend auf einem Skelett.
- **Funktionsweise:**
 1. Jedem Gelenk j wird eine Transformation \mathbb{T}_j (Matrix) zugewiesen.
 2. Jedes Gelenk j hat einen Einfluss $w_{i,j}$ auf jeden Vertex i .
 3. Deformierte Position p'_i :

$$\mathbb{p}'_i = \left(\sum_{j=1}^n w_{i,j} \mathbb{T}_j \right) \mathbb{p}_i$$

■ Linear Blend Skinning:





- **Probleme:** Hoher manueller Aufwand, „Bonbonverpackungs“-Artefakt (candy wrapper effect) bei Gelenken.

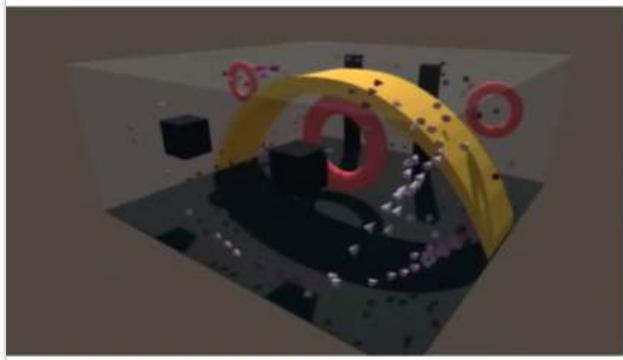


Alternativen zu LBS:

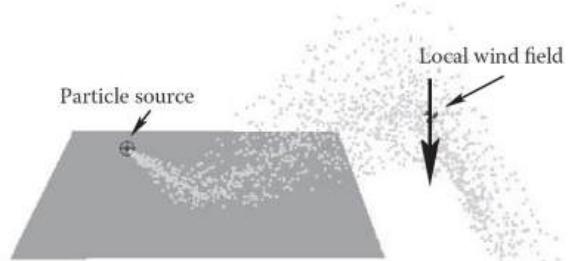
- **Ragdolls:** Physikbasierte Simulation für fallende/bewusstlose Charaktere.
- **Motion-Capture:** Erfassung realer Bewegungen (Kameras, Marker) und Übertragung auf virtuelle Charaktere.

Prozedurale Techniken

- Erstellen Animationen automatisch basierend auf Regeln und Zufallsprinzipien.
- Ziel: plausible, aber nicht unbedingt physikalisch korrekte Animationen.

Swarm/Flock/Boids	Game of Live (Cellular automata)
	

Particle System + Wind Field



- **Beispiele:** Game of Life (zelluläre Automaten), Partikelsysteme (Regen, Feuer), Schwarmverhalten.

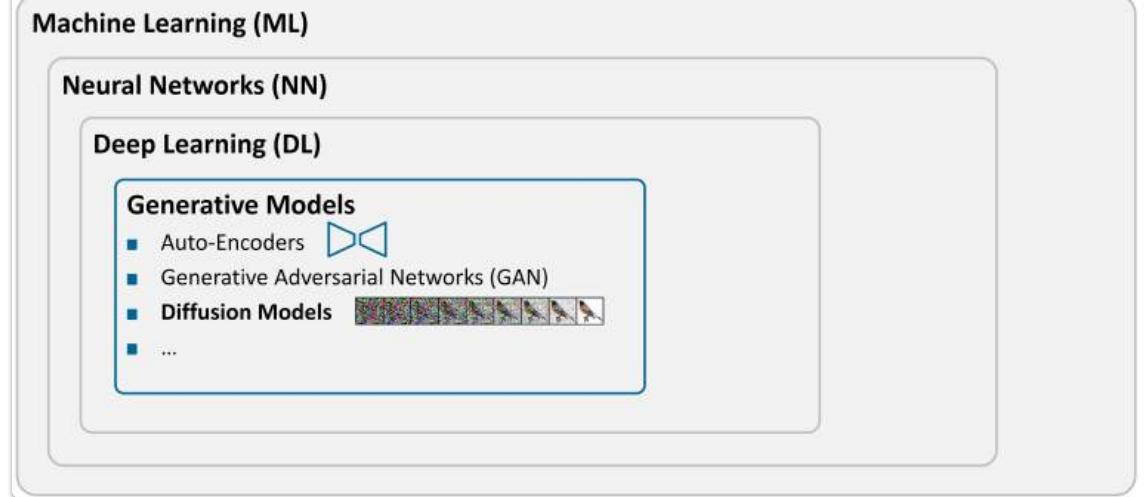
14. Machine Learning für 3D Graphics

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Machine Learning und Neuronale Netze

- **Maschinelles Lernen (ML)**: Entwickelt Algorithmen, die aus Daten lernen, um Vorhersagen oder Entscheidungen zu treffen (z.B. lineare Regression).



- **Neuronale Netze (NN)**: Grundlegend für ML. Bestehen aus verbundenen Knoten (Neuronen), die Eingaben mit Gewichtsmatrizen multiplizieren und Aktivierungsfunktionen (z.B. ReLU, Sigmoid) anwenden.
- **Deep Learning (DL)**: Untergruppe von ML. Nutzt NN mit vielen Parametern für komplexe Aufgaben wie:
 - **Bildklassifizierung & Segmentierung**: Kategorisieren und Identifizieren von Bildteilen.



- **Stilübertragungen**: Ändern des Bildstils bei Beibehaltung des Inhalts.
- **Synthetisierung neuartiger Ansichten**: Erzeugen neuer 3D-Perspektiven.

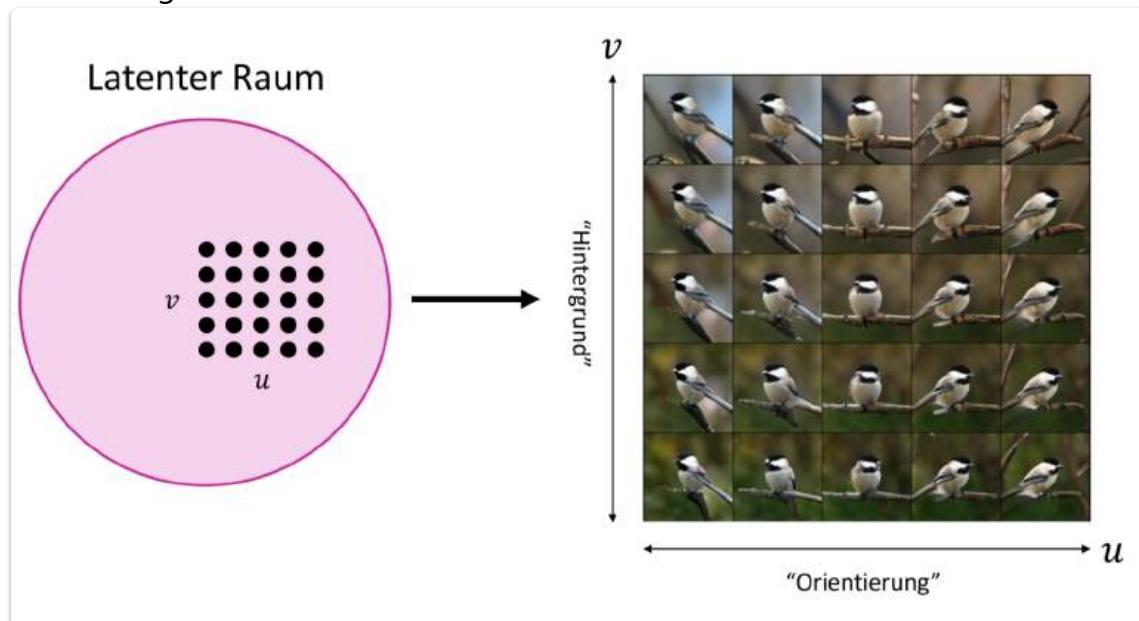
- **Klassifizierer:** ML-Anwendung, die hochdimensionale Daten (z.B. Bilder) in niedrigdimensionale Ausgaben (Klassenbezeichnungen wie "Mensch", "Hund") umwandelt. Das Trapezsymbol visualisiert diese Dimensionsreduktion.

Generative Modelle

- **Generative Modelle (GM):** Untergruppe des DL. Lernen die **Wahrscheinlichkeitsverteilung von Daten**, um **neue Dateninstanzen** (z.B. Bilder, Texte) zu erzeugen, die der Trainingsverteilung ähneln.
 - **Auto-Encoder:** Komprimieren Daten in eine kleinere Darstellung und rekonstruieren sie (primär für Kompression/Merkalsextraktion).
 - **Generative Adversarial Networks (GANs):** Generator erzeugt Daten, Diskriminator bewertet Echtheit; Wettbewerb führt zu realistischeren Generierungen.
 - **Diffusionsmodelle:** Erzeugen iterativ Ausgaben aus zufälligem Rauschen durch schrittweises Entrauschen.
- **Konzeptionelle Vorstellung:** Klassifikatoren reduzieren Dimensionen (Bild → Klasse), Generative Modelle erweitern sie (Text/Rauschen → Bild). Für Variabilität wird oft **zufälliges Rauschen** genutzt.

Latenter Raum

- Hochdimensionaler Raum im Modell, definiert durch **latente Variablen**, die Merkmale (z.B. Farbe, Größe) kodieren.
- Manipulation dieser Variablen ermöglicht **Kontrolle über Ausgabeattribute**.
- **Variable Verschränkung:** Änderungen in einer latenten Variablen können unbeabsichtigte Auswirkungen auf andere haben.



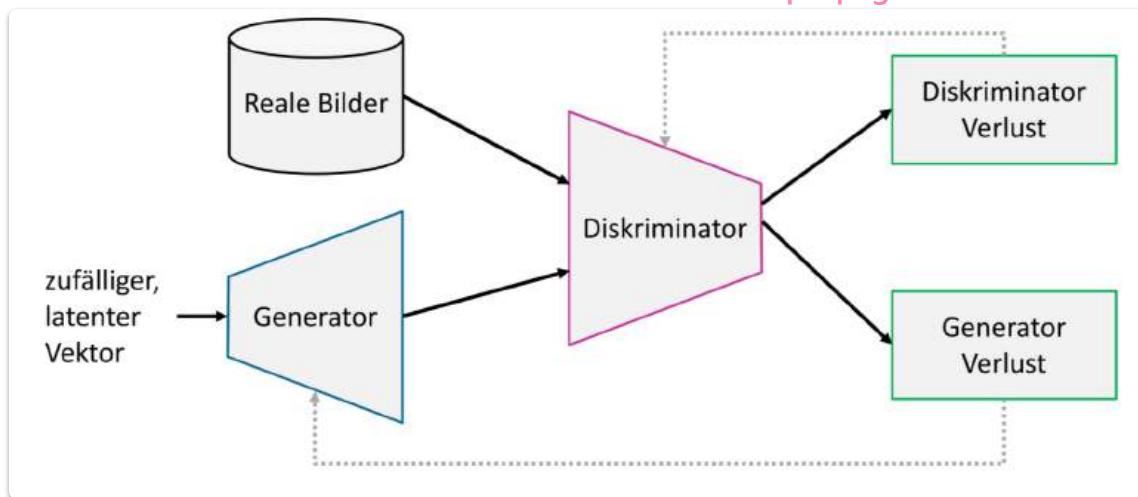
Generative Modelle für Bilderstellung

- **Prozess:**

1. **Architektur-Auswahl:** z.B. GANs, Diffusionsmodelle.
2. **Training:** Anhand großer Datensätze (oft mit Labels) zur Akkurate Generierung neuer Bilder.
3. **Inferenz:** Anwendung des Gelernten zur kontrollierten Erzeugung neuer Bilder mit spezifischen Merkmalen.

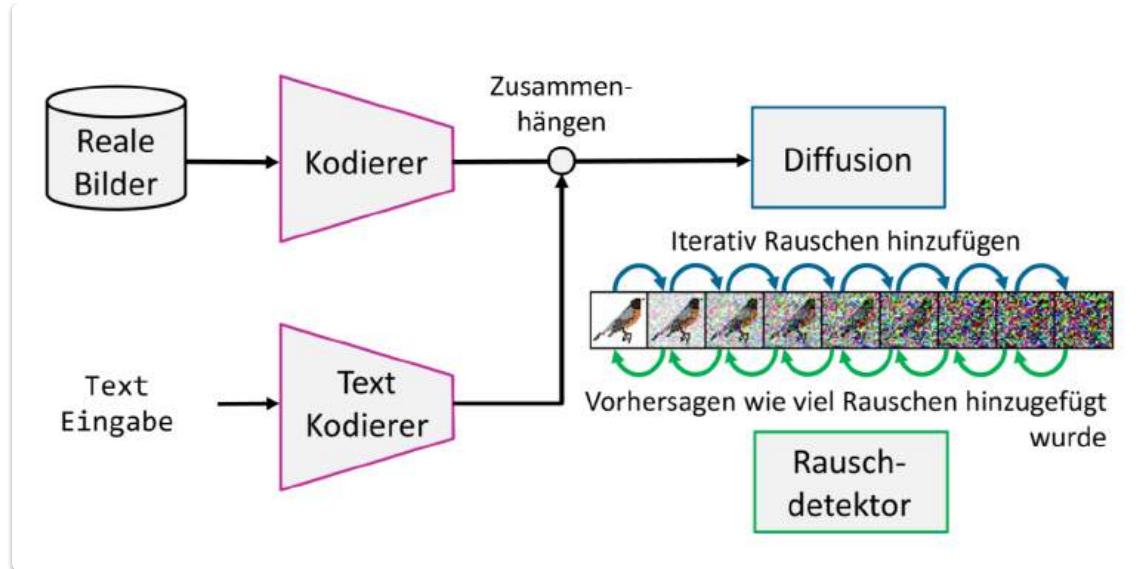
Training eines Generative Adversarial Network (GAN)

- **Wettstreit** zwischen **Generator** (erzeugt Bilder aus latenten Vektoren, versucht zu täuschen) und **Diskriminator** (unterscheidet echt/gefälscht).
- **Zwei Verlustfunktionen** werden gleichzeitig optimiert (Generator für Täuschung, Diskriminator für Erkennung).
- Verlustfunktionen müssen **differenzierbar** sein für **Backpropagation**.



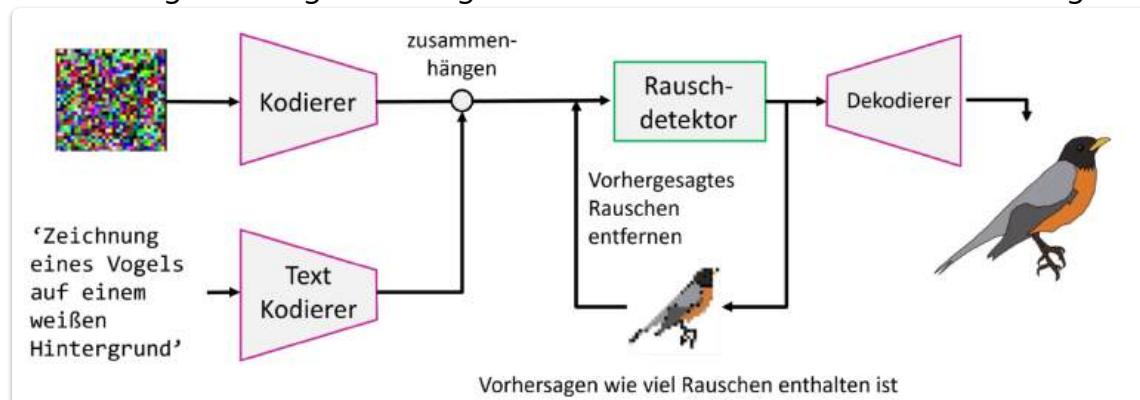
Training eines Diffusionsmodells

- Bilder und Text-Prompts werden in einem **gemeinsamen latenten Raum** kodiert.
- **Rauschhervorhebung:** Iteratives Hinzufügen von **gaußschem Rauschen** zu Bildern.
- **Rauschdetektor:** Lernt, den Grad des Rauschens vorherzusagen.
- **Rauschentferner:** Subtrahiert vorhergesagtes Rauschen vom verrauschten Bild und der Texteingabe.
- Ergebnis: Das Bild wird iterativ **weniger verrauscht und genauer**.



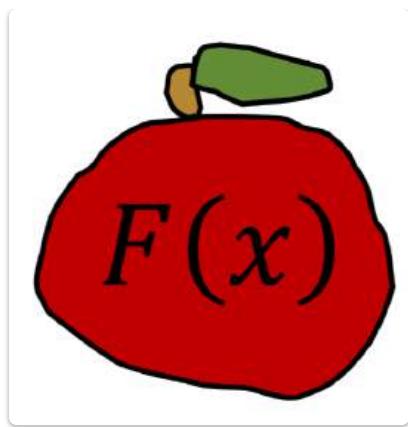
Inferenz in einem Diffusionsmodell

- Beginnt mit **Textaufforderung** und **zufälligem Rauschen**.
- Beides wird in den **latenten Raum** kodiert und iterativ durch **Rauscherkennungs-** und **Subtraktionsschritte** verfeinert.
- Der **Rauschdetektor** schätzt das Rauschen, das subtrahiert wird, um das Bild zu klären.
- Die entrauschte latente Darstellung wird von einem **Dekodierer** (oft VAE) in ein **visuelles Bild** umgewandelt.
- Fähigkeit: Erzeugung **detaillierter, kontextgerechter 2D-Bilder** aus Rausch- und Texteingaben.
- Anwendung: 2D-Bildgenerierung; für 3D-Modelle sind weitere Schritte nötig.

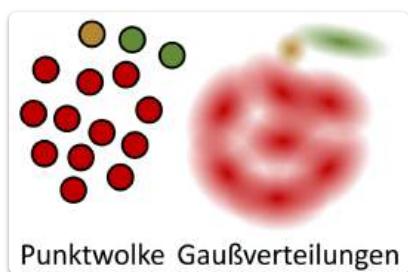
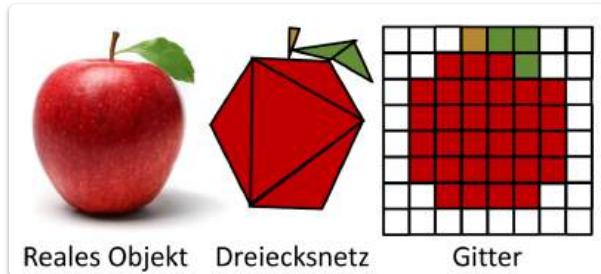


Szenendarstellungen für Machine 3D-Learning

- Realobjekte können nicht perfekt detailliert dargestellt werden; verschiedene Repräsentationen haben Vor- und Nachteile.
- **Implizite Darstellungen:** Funktionen, die für jeden Punkt im Raum einen Wert definieren (z.B. **Signed Distance Functions (SDFs)**).



- **Explizite Darstellungen:** Mengen diskreter Elemente (z.B. **Dreiecksnetz (Mesh)**, **Gitter (Grid)**, **Punktwolke**).



- **Herausforderungen:** Bei Punktwolken können Löcher entstehen; Dreiecksnetze können sich deformieren/überschneiden. Wahl der Darstellung ist aufgabenabhängig.

Neural Radiance Fields (NeRFs)

- **Implizite 3D-Szenendarstellung** mittels Gewichten eines neuronalen Netzes (Deep Learning). Erfordert hohe Rechenressourcen.
- **Kernmechanismus: Ray Casting/Sampling** durch die 3D-Szene, wobei ein **mehrschichtiges Perzepron (MLP)** Farbe und Dichte an Punkten entlang des Strahls bewertet.
- Volumetrische Daten werden zu einem Farbwert pro Strahl verdichtet, was zu **sehr detaillierter Szenenwiedergabe** führt.

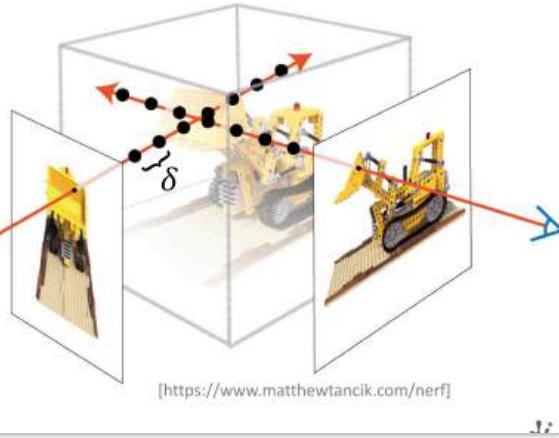
- Each point in space contains:

- (View dependent) color – c
- Density (transparency) – σ

- Use *raymarching* to render it

$$C(r) = \sum_{i=1}^N T_i (1 - \exp(-\sigma_i \delta_i)) c_i$$

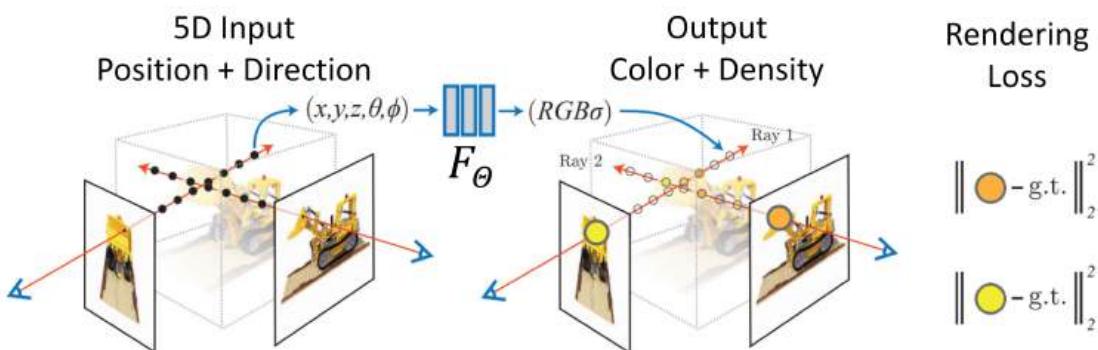
$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$



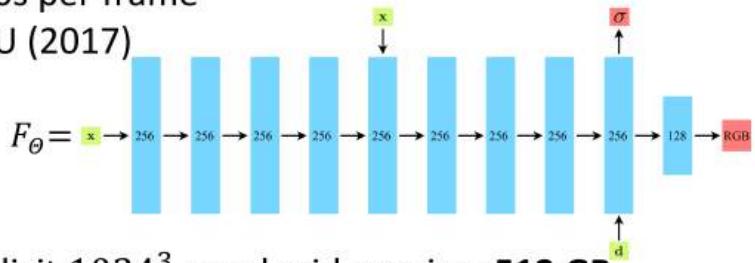
- Rekonstruktion: Basierend auf **realen Fotografien**. Der **Structure-from-Motion-Algorithmus (SfM)** rekonstruiert Kamerapositionen. Das Netzwerk wird optimiert, um gerenderte Bilder an die Basisbilder anzupassen.



- Herausforderungen: NeRFs sind **Blackboxes**, direkte Änderungen an Szenenteilen sind schwierig. Umwandlung in explizite Darstellung (z.B. Mesh via **Marching Cubes**) kann nötig sein.



- Approx. 590 000 weights $\approx 3 \text{ MB}$
- 1-2 days to train and 30s per frame on an NVIDIA V100 GPU (2017)



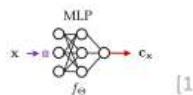
- Remember that an explicit 1024^3 voxel grid requires **512 GB**

Gaussian Splatting (GS)

- Explizite Technik zur Szenendarstellung, nutzt Volumen Rendering.
- Betrachtet Gaußsche Funktionen als primitive Formen statt neuronaler Netze als Blackboxen.

Neural Radiance Fields

- Implicit



- Deep Learning



- Ray Marching



Gaussian Splatting

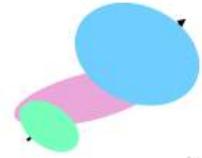
- Explicit



- Simple ML



- Blending Gaussians



Neural Radiance Fields

- Sample along ray



- Project



Gaussian Splatting

- Project



- Sample in 2D



$$\mathcal{C}(x) = \Pi(\Sigma w_i x)$$

$$\mathcal{C}(x) = \Sigma \Pi(w_i x)$$

- Jede Gaußglocke ist definiert durch: Mittelwert (Position), Kovarianzmatrix (Form), Deckkraft, Farbe. Ermöglicht einfachere Analyse von ML-Verfahren.

Normal distribution in 3D:

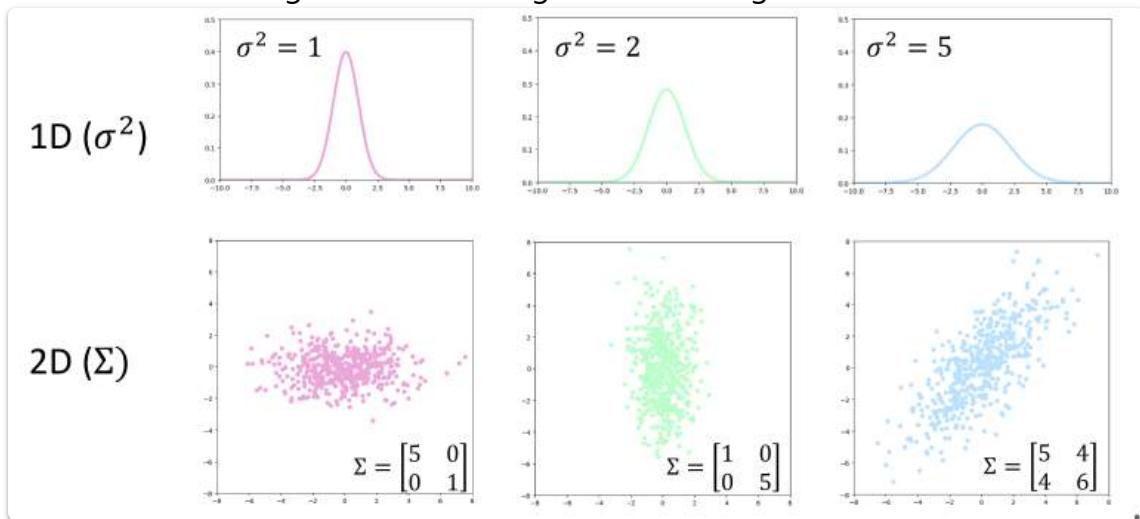
$$\mathbf{X} \sim N(\mu, \Sigma), \quad \mathbf{X} = X_1, X_2, X_3$$

- Position \mathbf{X} , where \mathbf{X} is the vector $\mu - x$ (x ... query point)
- Covariance matrix Σ (stretching/scaling)

$$G(\mathbf{X}) = e^{-\frac{1}{2}(\mathbf{X})^T \Sigma^{-1}(\mathbf{X})}$$

- Additionally: Color (RGB → Spherical Harmonics) + transparency (α)

- Rendering-Prozess: Projektion der 3D-Gaußglocken auf 2D, gefolgt von Rasterung. Effizientes Rendering durch Sortierung und Abtastung.



- **Training:**
 1. Beginnt mit initialen Gaußglocken, oft gewonnen durch **SfM** (3D-Punkte, die mit realen Bildern übereinstimmen).
 2. Punkte werden in Gaußsche Punkte umgewandelt; Kovarianzmatrizen so angepasst, dass sie **Flächen ohne Löcher** bilden und überlappen.
 3. Bei unzureichender Detaildarstellung werden Gaußglocken **geklont oder aufgeteilt** basierend auf der **Größe ihrer Positionsgradienten**.
- Ergebnis: Hochwertige, detailreiche 3D-Darstellung.

1 million Gaussians:

J. Eschner, A. Kovacs, E. Gröller



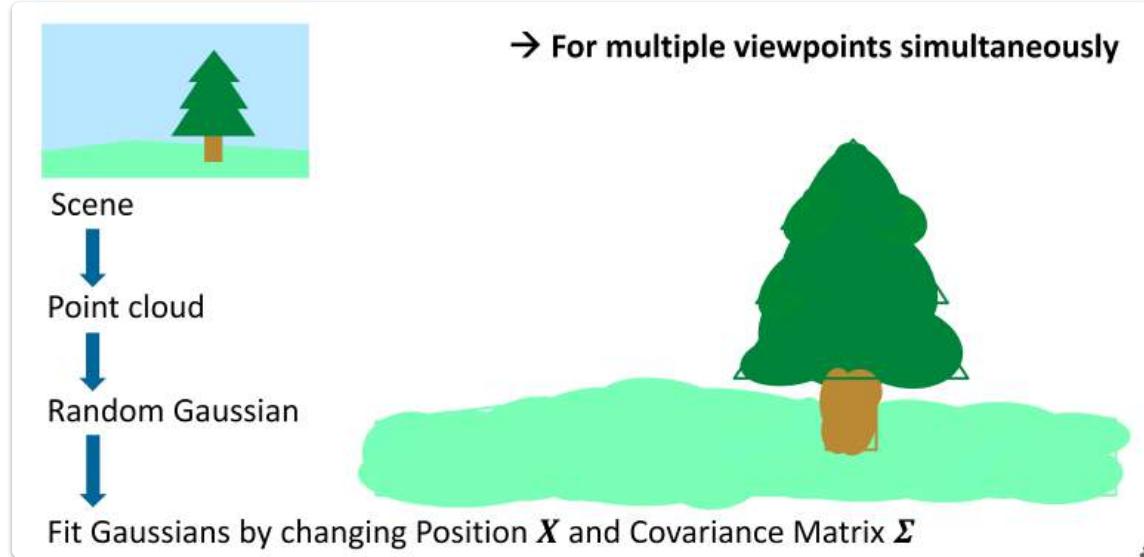
1 million Gaussians:
(as fully opaque ellipsoids)

J. Eschner, A. Kovacs, E. Gröller



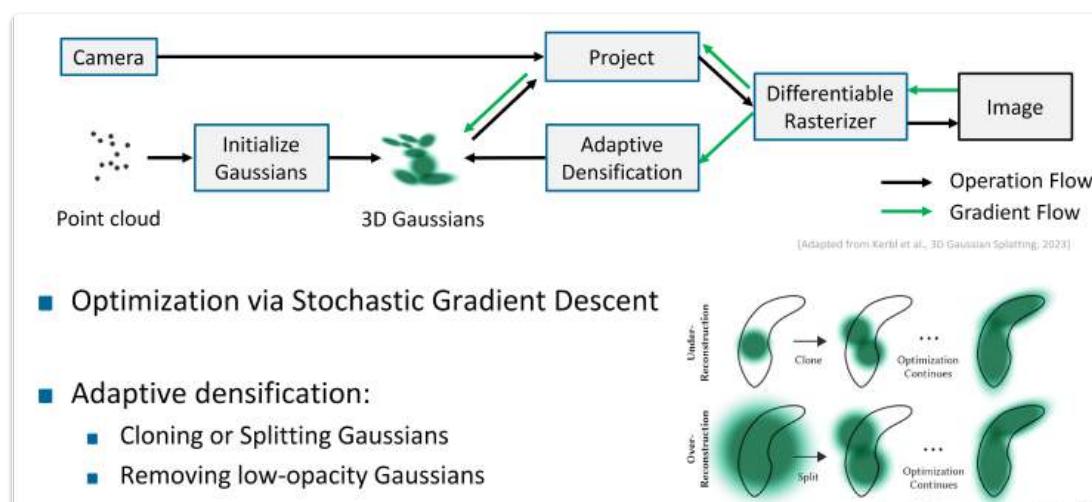
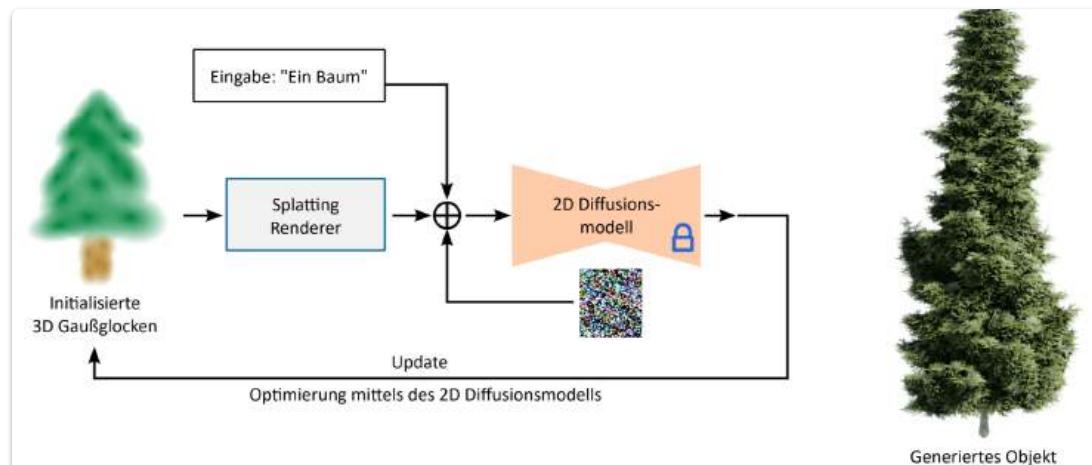
Generative Modelle für 3D-Objekte

- Kombination von **Diffusionsmodellen** und **3D Gaussian Splatting** zur Synthese neuer 3D-Objekte.



- Ablauf:**

1. Startpunkt: **Texteingabe** (Prompt).
2. Initialisierung: Erster Satz von **Gaußschen Punkten** (manuell oder vortrainiertes Netzwerk), umgewandelt in Gaußglocken.
3. Rendering & Rauschen: Gaußglocken werden aus **zufälligen Blickwinkeln gerendert** (Splatting Renderer) zu 2D-Bildern, denen **künstlich Rauschen hinzugefügt** wird.
4. Anpassung: Ein **Diffusionsmodell** analysiert die verrauschten 2D-Renderings, um sie an den Text-Prompt anzupassen. Die **zugrunde liegende 3D-Darstellung (Gaußglocken)** wird entsprechend angepasst.
5. Verfeinerung: Gaußglocken werden **aufgeteilt und geklont** basierend auf der **Größe ihres Gradienten** zur Detailverbesserung.
6. Automatisierte Erstellung: Nach vielen Iterationen ist das neue 3D-Objekt automatisch erstellt.



9. Multiskalenrepräsentation_AI

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

- **Nachbarschaftsoperationen:** Extrahieren lokale Merkmale (einige Pixel).
- **Großkalige Information:** Benötigt größere Filtermasken, was den Rechenaufwand erhöht.
- **Skalenabhängigkeit:** Merkmalsdetektion hängt von der richtigen Skala ab, die von den charakteristischen Objektgrößen bestimmt wird (z.B. Gesichtsgröße bei Gesichtserkennung).
- **Multiskalenanalyse:** Verarbeitet Bilder in unterschiedlichen Skalen/Auflösungsstufen, um feine Details und grobe Strukturen effizient zu analysieren.

Abtastung

- **Definition:** Umwandlung einer kontinuierlichen Funktion in eine diskrete Funktion durch Entnahme von Abtastwerten.
- **Modellierung:** Formal mit der Impulsfunktion (Dirac-Impuls) $\delta(x)$ beschrieben:

$$\int_{-\infty}^{\infty} \delta(x) dx = 1$$

Eine kontinuierliche Funktion $g(x)$ abgetastet an N Positionen:

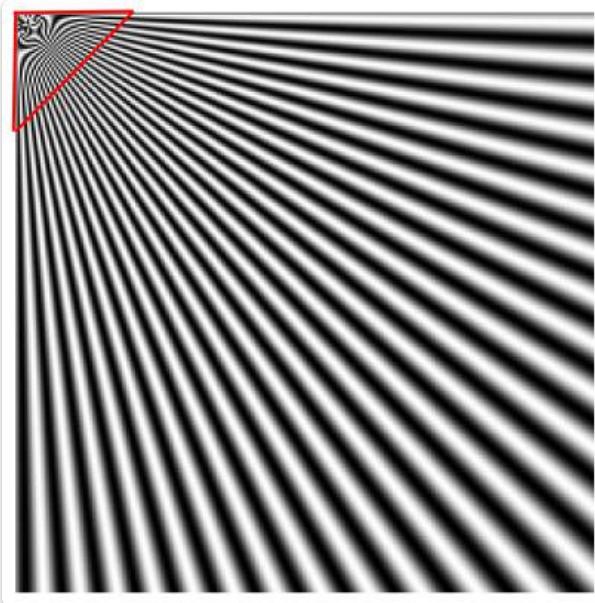
$$g_s(x) = g(x) \cdot \sum_{i=1}^N \delta(x - i)$$

- **Kammfunktion (Shah-Funktion):** Unendlich erweiterte Pulsfolge:

$$S(x) = \sum_{i=-\infty}^{\infty} \delta(x - i)$$

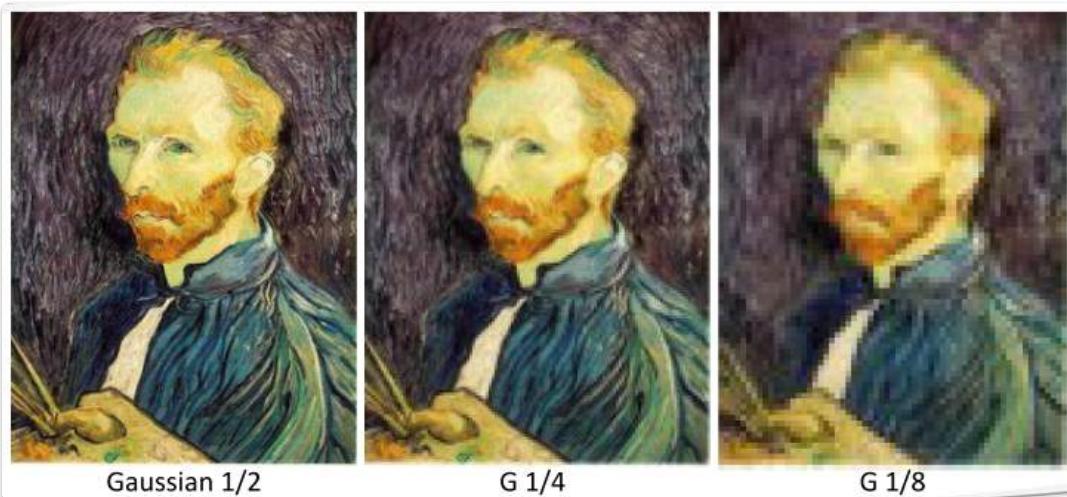
- **Frequenzspektrum:** Die Fouriertransformierte der Kammfunktion ist ebenfalls eine Kammfunktion. Das Spektrum des abgetasteten Signals ist eine periodische Replikation des Originalspektrums, mit Periodenlänge $1/x_0$.
- **Nyquist-Shannon-Abtasttheorem:** Um Aliasing zu vermeiden und das Originalsignal verlustfrei zu rekonstruieren, muss die Abtastfrequenz u_s mindestens doppelt so hoch sein wie die maximale Signalfrequenz u_{max} ($u_s > 2u_{max}$).

- **Aliasing:** Tritt auf, wenn die Abtastbedingung nicht erfüllt ist und sich Spektralkomponenten überlappen, wodurch das Signal verfälscht wird.
- Beispiel für Aliasing:



Bildskalierung

- **Subsampling (Verkleinerung):** Naives Löschen von Pixeln führt zu Aliasing.
- **Korrekte Vorgehensweise:** Zuerst **Gauß-Filterung** (Tiefpass) des Bildes, dann Verkleinerung. Der Filter muss das Abtasttheorem berücksichtigen.
- Korrekte Verkleinerung:



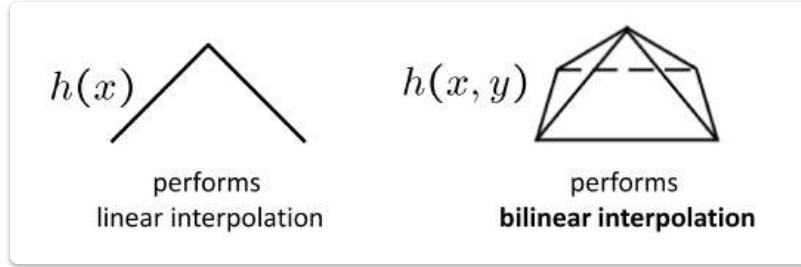
- **Resampling/Upsampling (Vergrößerung):** Neue Datenpunkte werden durch Interpolation aus benachbarten Pixeln erzeugt.
- **Bilineare Interpolation:** Berechnet neue Werte aus den vier nächstgelegenen Pixeln:

$$\begin{aligned} f(x, y) \approx & (1-a)(1-b) \cdot f(x_1, y_1) \\ & + a(1-b) \cdot f(x_2, y_1) \\ & + ab \cdot f(x_2, y_2) \\ & + (1-a)b \cdot f(x_1, y_2) \end{aligned}$$

Dabei sind $a = \frac{x-x_1}{x_2-x_1}$ und $b = \frac{y-y_1}{y_2-y_1}$.

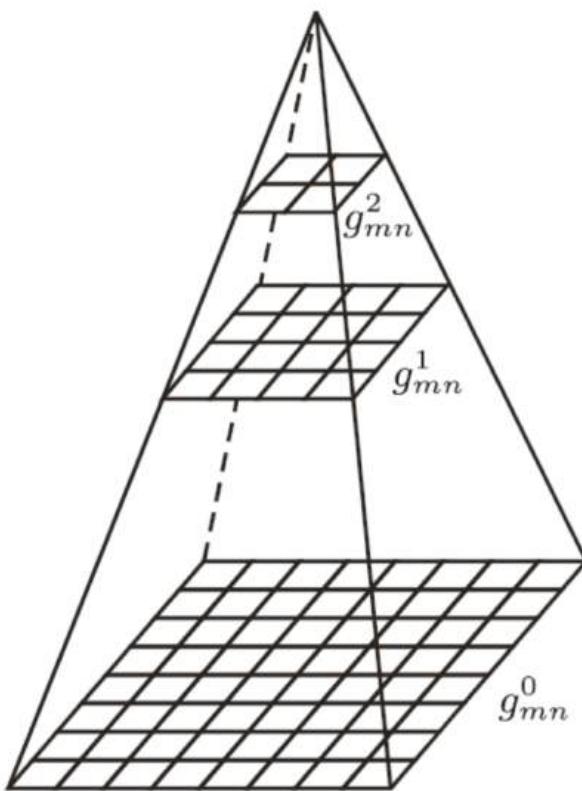
- **Bikubische Interpolation:** Nutzt mehr Nachbarpixel und komplexere Filterkerne für glattere Ergebnisse.

- Schema:



Bildpyramiden zur effizienten Multiskalenanalyse

- **Motivation:** Effiziente Analyse von Strukturen auf verschiedenen Skalen; Invarianz gegenüber Objektgröße.
- **Problem naive Multiskalenanalyse:** Hoher Rechenaufwand ($O(N^2 \cdot L^2)$ für Faltung).
- **Lösung: Bildpyramiden:** Mehrgitterdarstellung, die Bilder in unterschiedlichen Skalen bereitstellt.
 - **Aufbau:** Iterative Filterung (Tiefpass) und Unterabtastung (Faktor 2) des Bildes.
 - Start mit g_{mn} der Größe $N \times N$ (Ebene $v = 0$).
 - Berechnung der nächsten Ebene g'_{mn} ($N/2 \times N/2$) durch Tiefpassfilterung und Unterabtastung.
 - **Vorteile:**
 - Reduzierter Rechenaufwand für größere Strukturen (z.B. $O((\frac{N}{2^v})^2 \cdot (\frac{L}{2^v})^2)$).
 - Gesamtspeicherbedarf nur ca. $\frac{4}{3}$ des Originalbildes.
 - Gesamtberechnungsaufwand nur ca. $\frac{4}{3}$ der Operationen für ein 2D-Bild.
 - Pyramidenstruktur:



Gaußpyramide

- **Definition:** Bildpyramide, die mit einem Tiefpassfilter (Gaußfilter) konstruiert wird.
- **Erzeugung:** Ebene $G^{(v+1)}$ aus $G^{(v)}$ durch Glättung und Unterabtastung:

$$G^{(v+1)} = B \downarrow_2 G^{(v)}$$

wobei \downarrow_2 die Unterabtastung und B den Glättungsfilter darstellt.

- **Eigenschaften:**
 - Serie tiefpassgefilterter Bilder.
 - Grenzfrequenz halbiert sich von Ebene zu Ebene (eine Oktave).
 - Enthält zunehmend gröbere Details in oberen Ebenen.
 - Maximal $\log_2(N) + 1$ Ebenen für ein $N \times N$ -Bild.

Laplacepyramide (Difference of Gaussians)

- **Konstruktion:** Durch Differenzbildung aufeinanderfolgender Ebenen einer Gaußpyramide.

$$L^{(v)} = G^{(v)} - \uparrow_2 G^{(v+1)}, \quad L^{(n)} = G^{(n)}$$

wobei \uparrow_2 den Expansionsoperator bezeichnet.

- **Eigenschaften:**
 - Approximation der zweiten Ableitung des Originalbildes.
 - Zerlegt das Bildsignal in Bandpassbereiche mit logarithmischer Frequenzstaffelung.
 - Hervorhebung feiner (untere Ebenen) bis grober (obere Ebenen) Kantenstrukturen.

- Originalbild kann exakt aus Laplacepyramide und oberster Gaußpyramiden-Ebene rekonstruiert werden.

Anwendungen von Bildpyramiden

- **Coarse-to-Fine Strategien:** Effiziente Suche nach Korrespondenzen (z.B. im Template Matching), Kantenverfolgung.
- **Schnelles Template Matching:** Reduziert die Komplexität von $O(m \cdot n \cdot p \cdot q)$ durch Suche auf groben Skalen und Verfeinerung.
- **Image Fusion / Multi-Sensor Fusion:** Zusammenführung von Bildinformationen aus verschiedenen Quellen.
- **Image Compression:** Effiziente Kompression von Bilddaten (insbesondere mit der Laplace-Pyramide).
- **Image Blending und Mosaicking:** Nahtloses Zusammenfügen mehrerer Bilder.
 - **Feathering:** Glättung der Übergänge zwischen Bildern.
 - **Pyramid Blending:** Kombinieren von Laplace-Pyramiden der Quellbilder unter Verwendung einer Gauß-Pyramide einer Regionsmaske als Gewichte:

$$LS(i, j) = GR(i, j) \cdot LA(i, j) + (1 - GR(i, j)) \cdot LB(i, j)$$

Anschließende Rekonstruktion zum gemischten Bild.

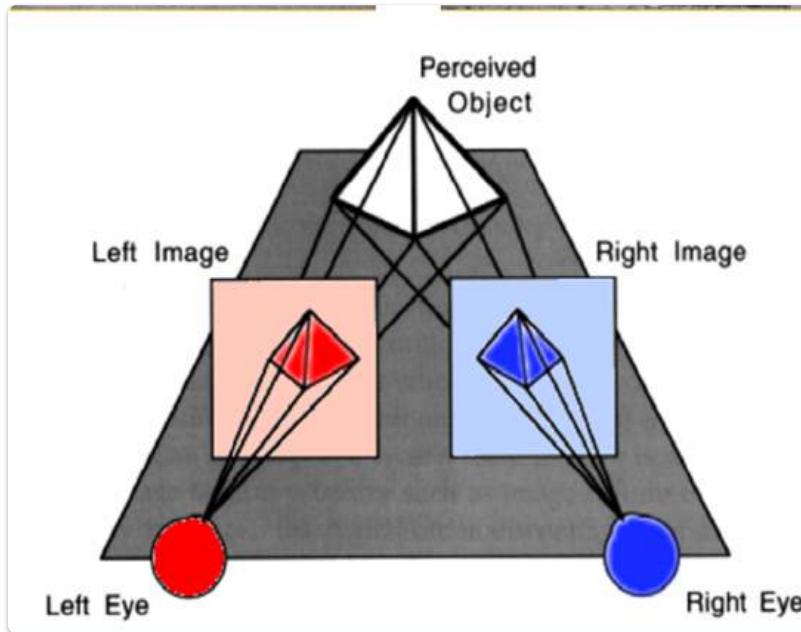
10. Stereo und Motion_AI

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Stereo Vision

- **Definition:** Räumliches Sehen zur Erstellung eines Tiefenbildes aus zwei 2D-Projektionen einer 3D-Szene.
- **Prinzip:** Nutzt zwei Kameras mit bekanntem Abstand.
 - Anwendung von **Triangulation** und **Epipolare Geometrie** zur Berechnung korrespondierender Bildpunkte.
 - Rekonstruktion von Tiefenwerten.
- **Disparität:** Der Versatz eines Objektpunktes zwischen den beiden Kamerabildern.
 - Größer für nahe Objekte, geringer für entfernte Objekte (gegen 0 im Unendlichen).
 - Ist **umgekehrt proportional zur Tiefe**.
 - Disparitätsmatrix: Zuweisung der Disparität zu jedem Bildpunkt zur Ableitung der Tiefe.
- Schema:



Stereoskopie

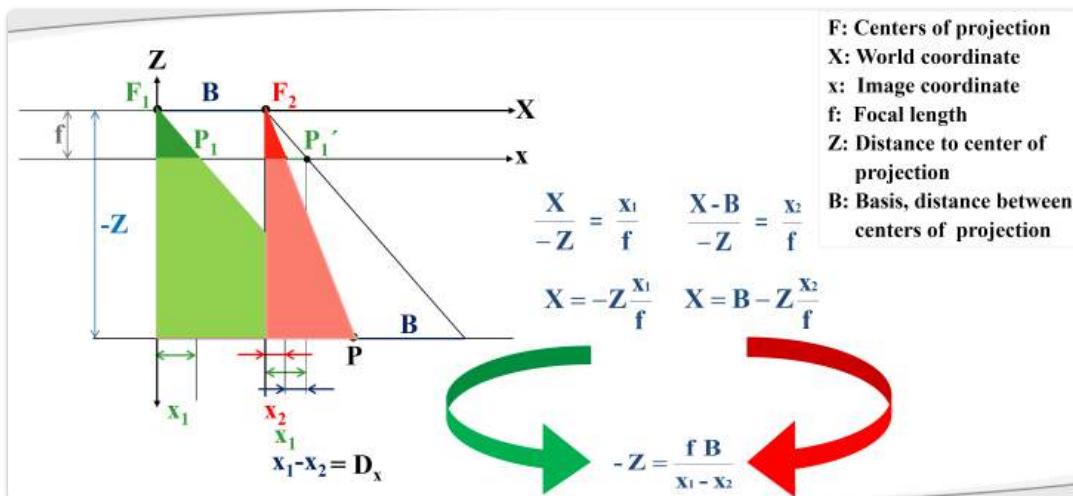
- **Definition:** Wiedergabe von Bildern, die einen räumlichen Eindruck von Tiefe vermitteln, obwohl die Tiefe physikalisch nicht vorhanden ist ("3D"-Effekt).
- **Geschichte:**
 - Euklid (3. Jh. v. Chr.) beschrieb das Prinzip.
 - Charles Wheatstone (19. Jh.) entdeckte die Stereoskopie und konstruierte das **Stereoskop**.
 - David Brewster (1849) stellte die erste Stereokamera vor.

Normalfall (Achsparalleles Stereosystem)

- **Definition:** Zwei horizontal verschobene Kameras, deren Koordinatensysteme nicht verdreht sind.
 - **Basislinie (B):** Abstand zwischen den optischen Zentren.
 - **Brennweite (f):** Identisch für beide Kameras.
 - Führt nur zu **horizontaler Disparität ($D = |x_l - x_r|$)** in Pixeln.
- **Tiefenberechnung:** Der Abstand Z eines Punktes von der Kamera:

$$Z = \frac{B \cdot f}{D}$$

- Disparität ist umgekehrt proportional zur Tiefe.
- Geometrie:



Epipolargeometrie

- **Definition:** Allgemeine Stereogeometrie, bei der Kameras sowohl verschoben als auch gedreht sind.
- **Wichtige Begriffe:**
 - **Epipole (e, e'):** Schnittpunkte der Verbindungsgeraden der Kamerazentren mit den Bildebenen.
 - **Epipolarebene:** Ebene, die durch den 3D-Punkt X und die beiden Brennpunkte aufgespannt wird.

- **Epipolarlinien (l, l'):** Schnittlinien der Epipolarebene mit den Bildebenen. Der korrespondierende Punkt liegt immer auf dieser Linie.

- Schema:

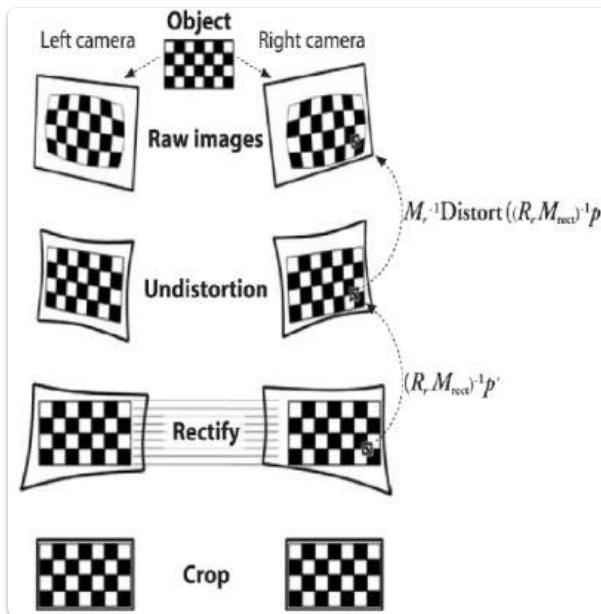


- **Eigenschaft:** Epipolargeometrie hängt nur von der **relativen Pose und den internen Kameraparametern** ab, nicht von der Szenenstruktur.

Image Rectification

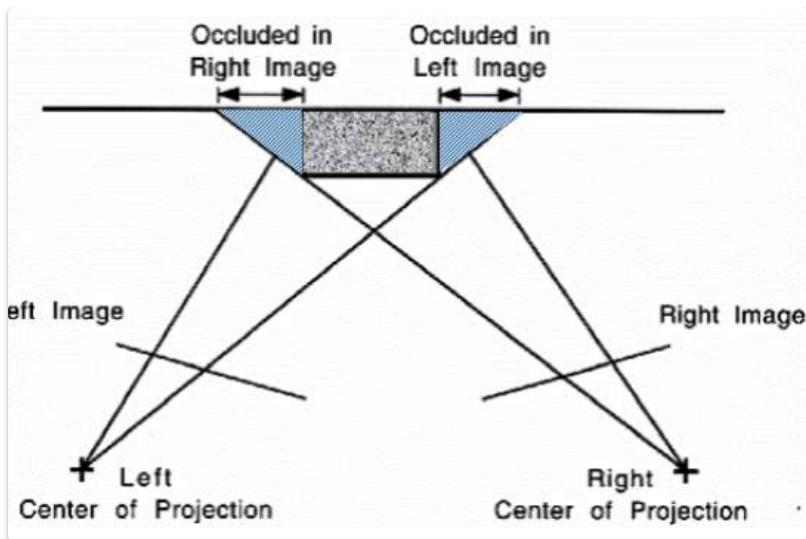
- **Ziel:** Vereinfachung der Stereo-Geometrie, sodass korrespondierende Punkte auf der gleichen horizontalen Zeile liegen (analog zum Normalfall).
- **Prozess:**
 1. **Undistortion:** Korrektur von Linsenverzerrung.
 2. **Rectify:** Transformation der Bilder, sodass Epipolarlinien horizontal werden.
 3. **Crop (optional):** Entfernen von Rändern ohne gültige Informationen.
- **Vorteil:** Vereinfacht die Suche nach Korrespondenzen erheblich.

- Schema:



Okklusionen

- **Problem:** Bereiche, die in einem Bild verdeckt sind, können im anderen Bild nicht abgeglichen werden.
- **Folge:** Fehlende Tiefeninformationen in okkludierten Bereichen.
 - Illustration:



Korrespondenzproblem

- **Definition:** Zentrales Problem der Stereo Vision: Finde für jeden Punkt im linken Bild den entsprechenden Punkt im rechten Bild.
- **Vereinfachung:** Durch **Bildrektifikation** wird die Suche auf eine einzelne horizontale Scanline reduziert.

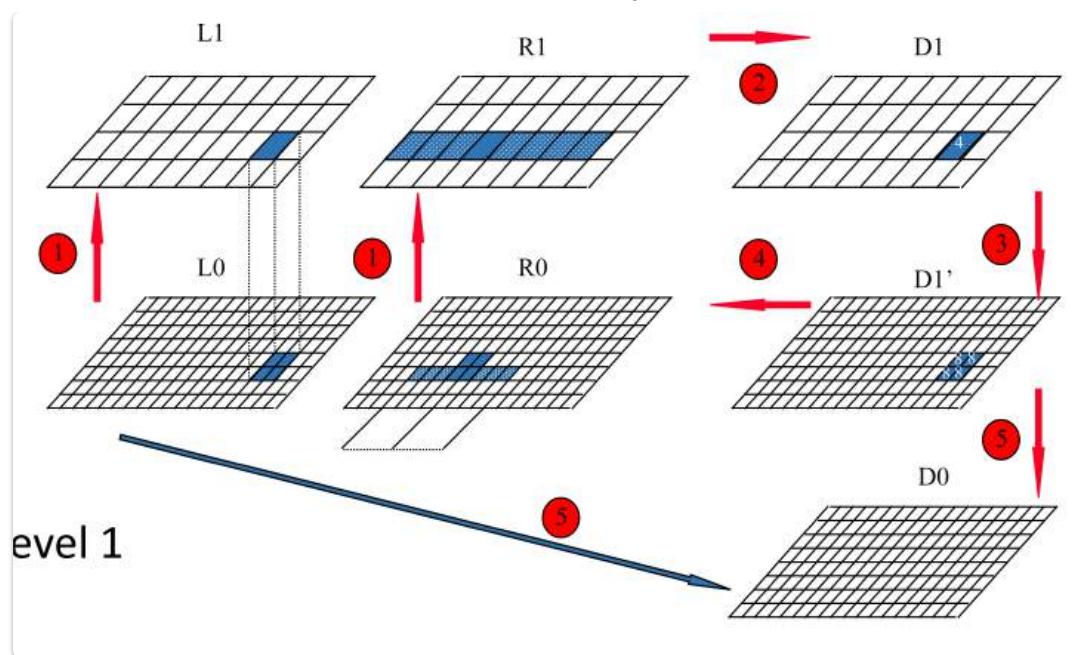
Regionenbasiertes Matching (Area-Based Matching - ABM)

- **Prinzip:** Vergleicht kleine Bildausschnitte (Beobachtungsfenster) anhand ihrer Grauwerte.
- **Ähnlichkeitsmaße:**
 - **SSD (Sum of Squared Differences):**

$$SSD(\Delta m, \Delta n) = \sum_{i,j \in R} [I_1(i, j) - I_2(i - \Delta m, j - \Delta n)]^2$$
 (Minimum bei Übereinstimmung)
 - **CC (Cross-Correlation):**
$$CC(\Delta m, \Delta n) = \sum_{i,j \in R} [I_1(i, j) \cdot I_2(i - \Delta m, j - \Delta n)]$$
 (Maximum bei Übereinstimmung)
- **Vorteil:** Erzeugt ein **dichtes Tiefenbild**.
- **Nachteil:** Hoher Rechenaufwand, Probleme bei homogenen Bildbereichen oder Verdeckungen.

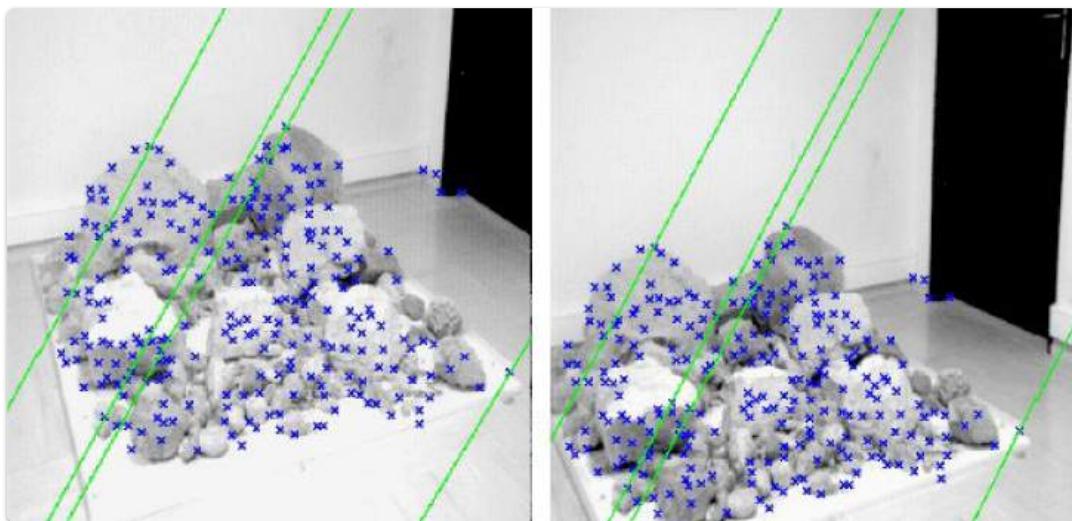
Hierarchical Matching

- **Ansatz:** Nutzt **Bildpyramiden** (z.B. Gaußpyramiden) zur Effizienzsteigerung.
- **Prozess:**
 1. Disparitäten auf der obersten (niedrigsten Auflösung) Ebene berechnen.
 2. Ergebnisse auf die nächstniedrigere Ebene projizieren.
 3. Disparitäten auf der niedrigeren Ebene verfeinern.
 4. Iteration bis zur Originalauflösung.
- **Vorteile:** Schnellere Berechnung, bewältigt große Disparitätsbereiche.
 - Schema:



Merkmalsbasiertes Matching

- **Ansatz:** Verwendet einzelne, gut zuordbare Pixel (Merkmale wie Ecken, Kanten, Interest Points).
 - Extraktion mittels lokaler Operatoren (z.B. Moravec, SIFT).
- **Vorteil:** Schnellerer Korrespondenzvergleich durch Datenreduktion.
- **Nachteil:** Erzeugt nur **spärliche Tiefeninformationen** für die Merkmale; Interpolation für ein dichtes Tiefenbild erforderlich.
- Prozess:



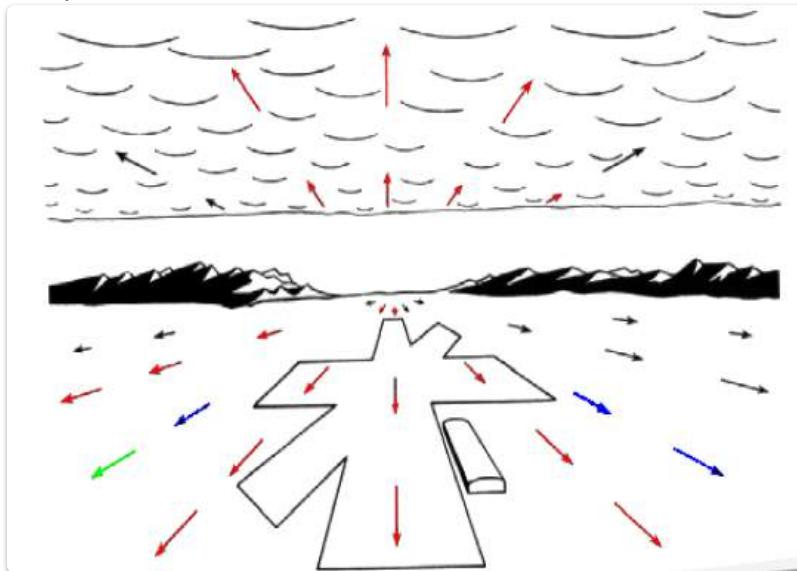
Structure-from-Motion (SfM)

- **Definition:** Gewinnung von 3D-Informationen aus einer **zeitlichen Folge von Bildern** (bewegte Kamera).
- **Zentrales Problem:** Bestimmung der Kamerabewegung (Richtung und Ausmaß).
- **Unterschied zu Stereo Vision:** Kamerageometrie ist zunächst **unbekannt**.
- **Vorgehensweise:** Schätzung von Bewegungsfeldern aus Korrespondenzen, Bestimmung der Kamerabewegung, Rekonstruktion der 3D-Struktur.

- **Motion Parallax (Motion Disparity):** Nahe Objekte bewegen sich bei Betrachterbewegung schneller als entfernte.

Bewegungsfeld

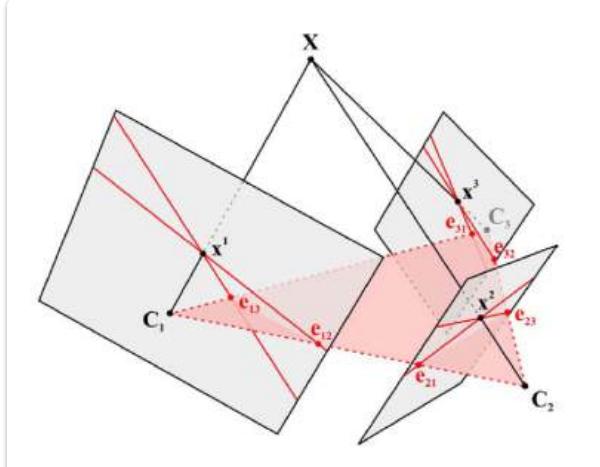
- **Definition:** Einzelne Bewegungsvektoren aller Punkte im Bild.
- **Kamera-Translation:** Führt zu einem "nach außen" (FoE - Focus of Expansion) oder "nach innen" (FoC - Focus of Contraction) Zeigen aller Vektoren zu einem Punkt (Epipol).
- Beispiel:



- **Bildbewegung:** Umgekehrt proportional zur Entfernung eines Punktes zur Kamera.
- **Herausforderungen:** Spärlich besetztes Vektorfeld, unbekannte Kamerabewegung (und damit Epipolarlinien) zu Beginn.

Multi View Geometry

- **Definition:** 3D-Rekonstruktion unter Verwendung von **mehr als zwei beliebigen Bildern**.
- **Vorteil:** Stabilere 3D-Punktwolke durch Hinzunahme mehrerer Ansichten. Ein Objektpunkt in zwei Bildern kann seine Position im dritten Bild voraussagen.
- Beispiel:



- **Bündelausgleich (Bundle Adjustment):**

- Gleichzeitige Bestimmung interner/externer Kameraparameter und 3D-Szenenstruktur.
- Ziel: Optimale Übereinstimmung zwischen erwarteten und gemessenen Bildpunkten.
- Annahmen: Starrheit der Szene, Kollinearitätsgleichung (3D-Punkt, Bildpunkt, Projektionszentrum liegen auf einer Geraden).

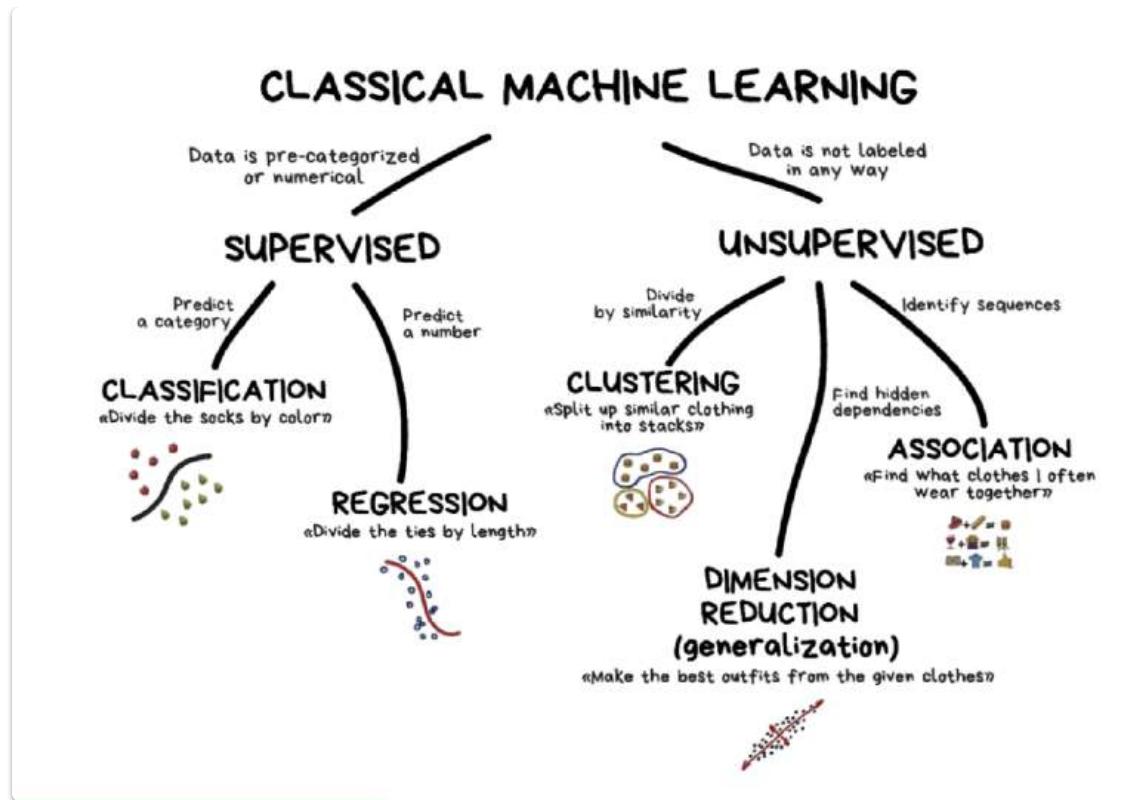
11. Deep Learning for Computer Vision_AI

⚠ Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Maschinelles Lernen

- **Definition:** Maschinelles Lernen (ML) ist ein Teilgebiet der künstlichen Intelligenz (KI).
- **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.
- **Kategorien des Maschinellen Lernens:**
Überwachtes Lernen (Supervised Learning): Vorhersage einer Kategorie (Klassifikation) oder einer Zahl (Regression). Daten sind vor-kategorisiert oder numerisch.
Unüberwachtes Lernen (Unsupervised Learning): Struktur in den Daten finden (Clustering, Assoziationsanalyse, Dimensionsreduktion). Daten sind nicht gelabelt.
* **Bestärkendes Lernen (Reinforcement Learning):** Nutzen maximieren durch Belohnung und Bestrafung.



Überwachtes Lernen (Supervised Learning)

- Zielvariable:** Wenn eine dedizierte Ausgabevariable definiert ist, kann ein Modell darauf trainiert werden, diese für unbekannte Datensätze (Test Set) vorherzusagen.
- Generelles Ziel:** Maximierung der Genauigkeit dieser Vorhersagen.
- Lernprozess:** Algorithmus lernt die Beziehung zwischen Eingabewerten (Attributen) und zugehörigen Ausgabewerten anhand des **Training Set**.
- Validierung:** Der gesamte Datensatz wird in ein **Training Set** und ein **Test Set** aufgeteilt. Die Evaluation erfolgt mit dem Test Set, um die Genauigkeit und Fehlerrate anhand unbekannter Daten zu bestimmen.

Traditional Programming:



Machine Learning:



Unüberwachtes Lernen (Unsupervised Learning)

- Anwendung:** Wenn ein Datensatz keine präzisen Ausgabewerte aufweist.
- Ziel:** Bisher unbekannte Muster und Zusammenhänge aus den Eingabedaten ableiten.

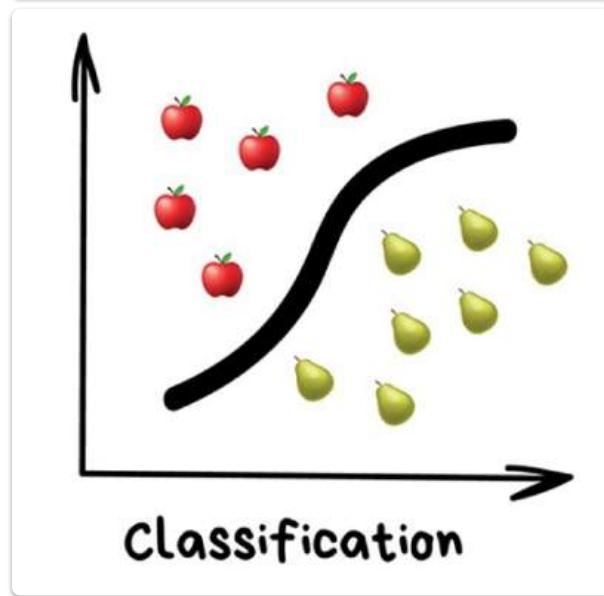
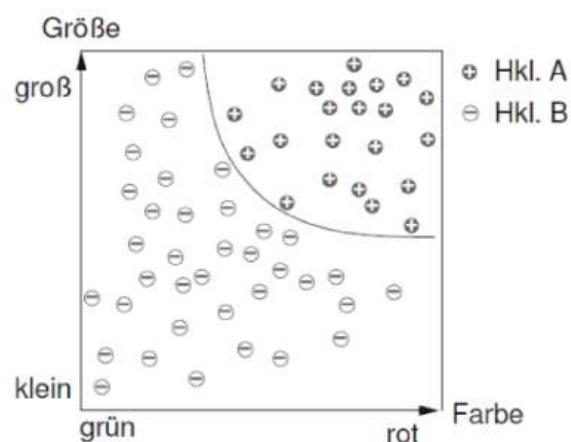
Klassifikation

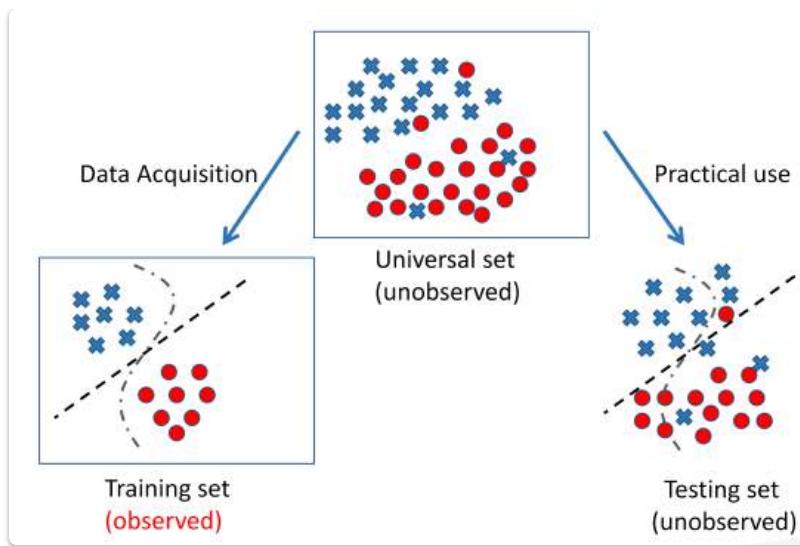
- **Häufigste Anwendung in der Computer Vision:** Klassifikation anhand von Merkmalen (Features).
- **Klassifikatoren (Classifier):** Systeme, die Merkmalsvektoren in eine endliche Anzahl von Klassen einteilen.
- **Aufgabe beim maschinellen Lernen:** Aus gesammelten klassifizierten Daten eine Funktion generieren, die für neue Objekte aus Features den Wert der Klasse berechnet.

Größe [cm]	8	8	6	3	...
Farbe	0.1	0.3	0.9	0.8	...
Handelsklasse	B	A	A	B	...

Funktion zur Klassifikation

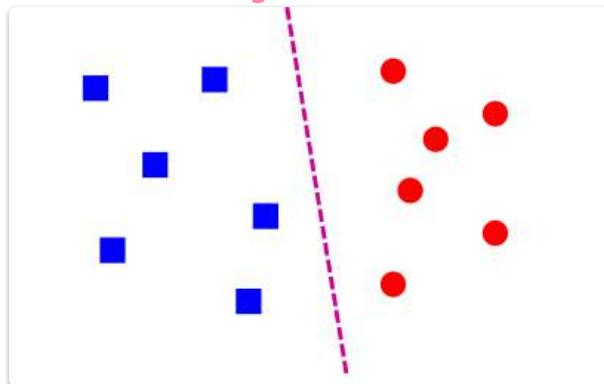
- Eine **Trennlinie** oder **Hyperfläche** repräsentiert eine Funktion, die die Klassifizierung vornimmt.
- **Klassenzuweisung:** Objekte werden basierend auf ihrer Position relativ zur Trennlinie/Hyperfläche einer Klasse zugewiesen.
- **n Merkmale:** Im n -dimensionalen Merkmalsraum wird eine $(n - 1)$ -dimensionale Hyperfläche gesucht, die die Klassen möglichst gut trennt (wenig falsch klassifizierte Objekte).





Klassifikationsalgorithmen im Maschinellen Lernen

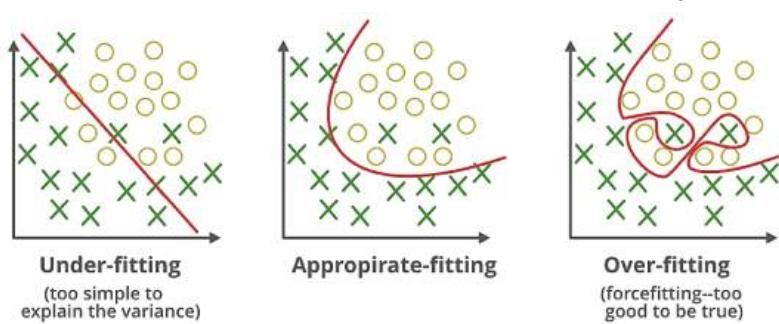
- **Nearest Neighbor Algorithmus:** Datenpunkt wird der Klasse der benachbarten Datenpunkte zugeordnet.
- **Lineare Gleichung:** Finden einer linearen Gleichung zur Klassentrennung.



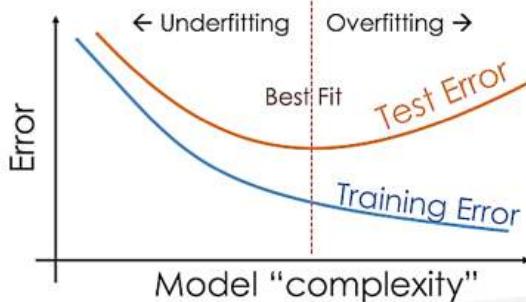
- **Entscheidungsbäume (Decision Trees):** Klassenzuordnung anhand eines Pfades durch den Baum.
- **Random Forest Algorithmus:** Klassenzuordnung durch Mehrheitsbeschluss mehrerer Entscheidungsbäume.
- **Support Vector Machine (SVM):** Trennung der Klassen durch eine Hyperebene mit dem größtmöglichen Abstand (Maximum-Margin-Klassifikator).
- **Boosting-Verfahren (z.B. AdaBoost):** Kombination "schwacher" Klassifikatoren zu einem stärkeren.
- **Künstliche Neuronale Netze (NN):** Werden später genauer betrachtet.

Generalisierung

- **Überanpassung (Overfitting):** Modell lernt spezifische Details und Rauschen der Trainingsdaten, schlecht auf neuen Daten.



d



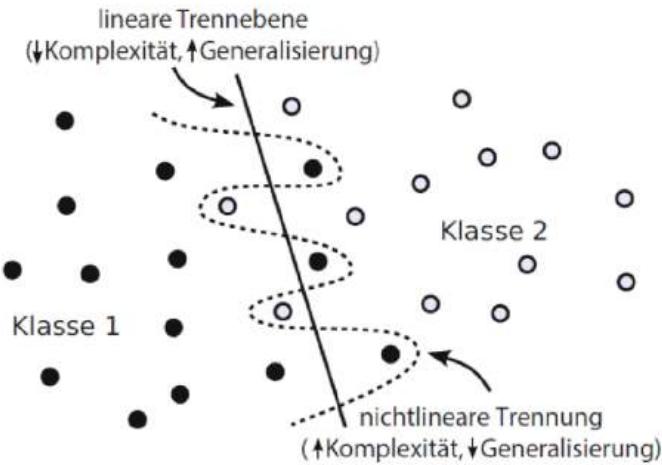
- **Generalisierung - Das Ziel:** Modell soll Muster in Trainingsdaten erkennen und auf neue, unbekannte Daten anwenden können.
- **Modellkomplexität:** Balanceakt zwischen **Underfitting** (zu einfach, kann Zusammenhänge nicht erfassen) und **Overfitting** (zu komplex, lernt Rauschen).



Training set (labels known)



Test set (labels unknown)



Generalisierung - Bias-Variance Tradeoff

- **Bias (Verzerrung):** Fehler aufgrund vereinfachter Annahmen im Modell (Underfitting). Hoher Bias bei einfachen Modellen.
- **Variance (Varianz):** Sensitivität der Modellschätzung gegenüber Schwankungen in den Trainingsdaten (Overfitting). Hohe Varianz bei komplexen Modellen.
- **Bias-Variance Tradeoff:** Das Ziel ist es, ein Modell mit niedrigem Bias und niedriger Varianz für gute Generalisierung zu finden.

$$\text{Gesamtfehler} = \text{Bias}^2 + \text{Varianz} + \text{Irreduzierbarer Fehler}$$

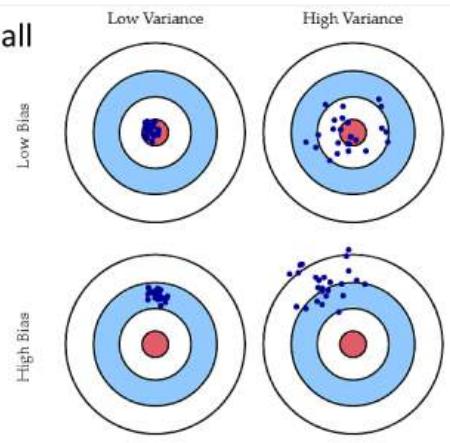
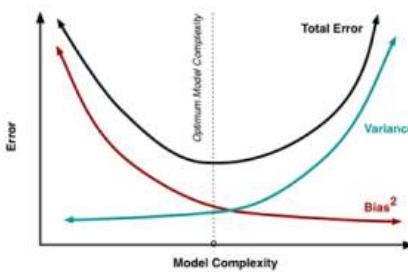
- **Komplexität und Bias-Varianz:**

Geringe Komplexität: Hoher Bias, niedrige Varianz (Underfitting).

Hohe Komplexität: Niedriger Bias, hohe Varianz (Overfitting).

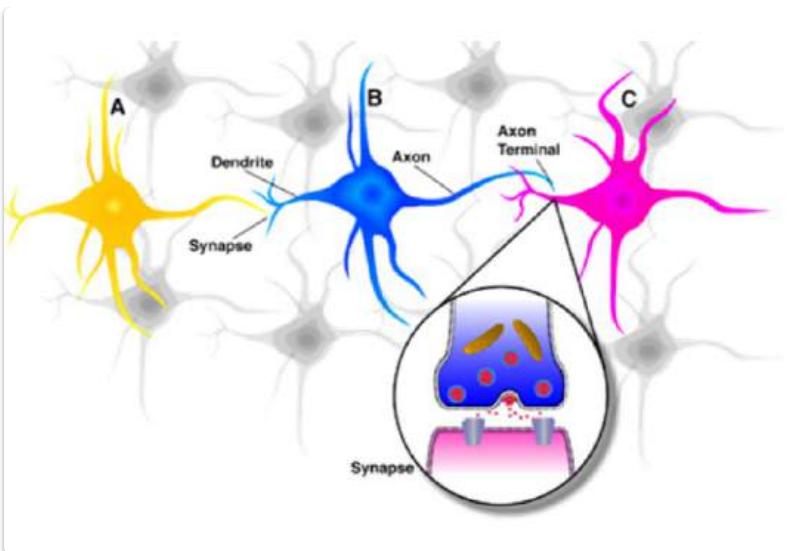
* Optimale Komplexität: Gleichgewicht zwischen Bias und Varianz.

- **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**



Künstliche Neuronale Netze (NN) - Biologische Inspiration

- **Biologisches Vorbild:** Neuronale Netze im Gehirn.
- **Struktur und Funktion eines biologischen Neurons (vereinfacht):** Zellkörper, Dendriten, Axon, Synapsen. Lernen durch Adaptivität der Synapsen.



- **Mathematisches Modell der NN:**

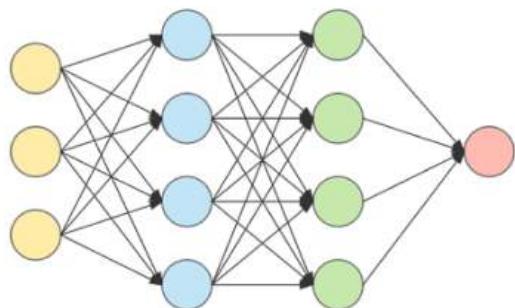
McCulloch und Pitts (1943): Erstes mathematisches Modell eines Neurons.

Bestandteile: Neuronen (Knoten), Synapsen (Kanten), Gewichte.

* **Adaption:** Anpassung der Gewichte verändert das Verhalten des künstlichen Neurons.

- 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.

Feed-forward Net



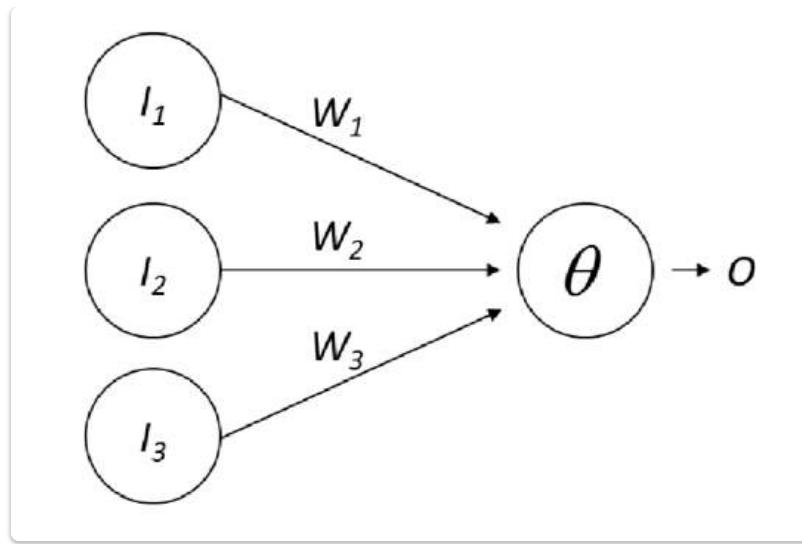
Das Perceptron - Ein frühes künstliches neuronales Netz

- **Entwicklung:** 1958 von Frank Rosenblatt.
- **Struktur:** Einzelnes künstliches Neuron mit Eingängen (I_1 bis I_k) und Gewichten (w_1 bis w_k).
- **Gewichtete Summe der Eingaben:**

$$\Phi(x) = \sum_i w_i I_i$$

- **Aktivierungsfunktion (f):**

$$O = f(\sum_i w_i I_i)$$



Schwellwertfunktion im Perceptron

- **Funktionsweise:** Gewichtete Summe wird mit einem Schwellwert θ verglichen.
- **Ausgabe (O):**

$$O = \begin{cases} 1 & \text{falls } \sum_i w_i I_i + \theta \geq 0 \\ 0 & \text{sonst} \end{cases}$$

- **Ziel:** Trennung von Daten zweier Klassen.

Perceptron-Lernalgorithmus (δ -Regel / Widrow-Hoff-Regel)

- **Ziel:** Anpassung der Gewichte.
- **Gewichtsänderung:**

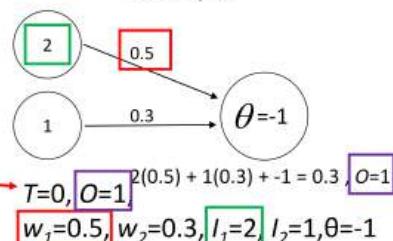
$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = (T - O) I_i$$

(T : gewünschte Ausgabe, O : tatsächliche Ausgabe)

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 $w_1=-1.5, w_2=-0.7, \theta=-2 \Rightarrow -5.7, O=0$

Example



$$w_1(t+1) = 0.5 + (0-1)(2) = -1.5$$

$$w_2(t+1) = 0.3 + (0-1)(1) = -0.7$$

$$w_\theta(t+1) = -1 + (0-1)(1) = -2$$

Konvergenz des Perceptron-Lernalgorithmus

- **Rosenblatts Theorem:** Konvergiert in endlicher Zeit, wenn Daten linear separierbar sind.

Beschränkungen des Perceptrons

- Kann keine nicht-linear separierbaren Funktionen lernen.

Mehrschichtige Perceptrons

- Zweischichtig:** Können konvexe Mengen trennen.
- Mehrschichtig:** Können beliebige Mengen trennen.

Aktivierungsfunktionen für stetige Neuronen

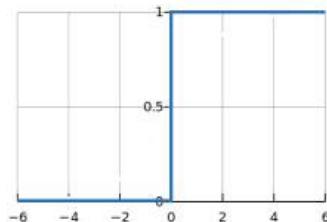
- Sigmoid-Funktion:** Glättet Unstetigkeit der Stufenfunktion.

$$O = \frac{1}{1 + e^{-(\Phi(x) + \theta)}}$$

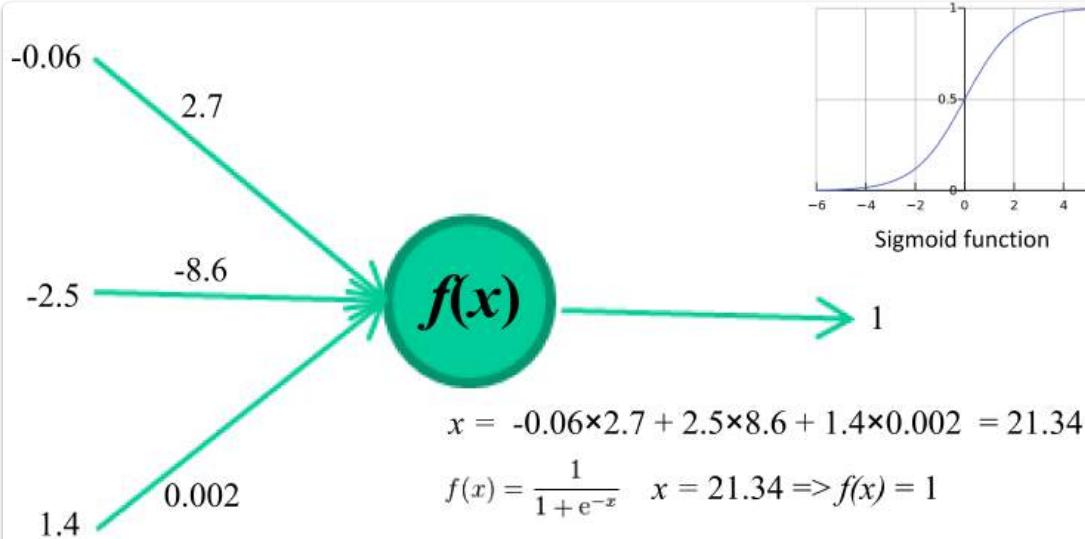
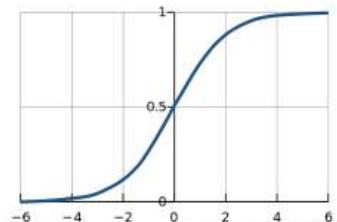
- Eigenschaften:** Annähernd linear im kritischen Bereich, asymptotisch beschränkt (Ausgabe zwischen 0 und 1).
- Vorteil:** Ermöglicht differenzierbare Ausgaben, wichtig für Gradientenabstieg.

$$\Delta w_k = c I_k (T_j - O_j) f'(\text{ActivationFunction})$$

Old: $O = \begin{cases} 1 : \sum_i w_i I_i + \theta > 0 \\ 0 : \text{otherwise} \end{cases}$

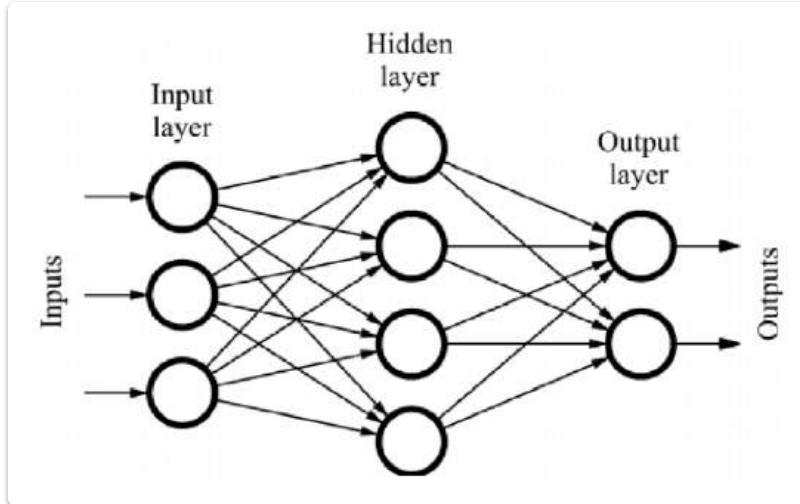


New: $O = \frac{1}{1 + e^{-\sum_i w_i I_i + \Theta}}$



Multilayer Perceptron (MLP)

- **Typischer Aufbau:** Mehrere Schichten von Verarbeitungseinheiten (Input, Hidden, Output Layer).
- **Verbindungen:** Zwischen den Schichten mit Gewichten versehen.



- **Arbeitsweise:** Vorwärtsvermittlung (Feed-forward), keine Rückkopplungen.

Training von Multilayer Perceptron (MLP)

- **Überwachter Lernvorgang:** Gewichtsfaktoren werden angepasst.
- **Fehlerberechnung:** Vergleich der Netzwerkausgabe mit Soll-Werten.
- **Gewichtsanpassung:** Minimierung des Fehlers.

Backpropagation-Training (Fehler-Rückvermittlung)

- **Prinzip:** Gewichte werden in Richtung des abnehmenden Fehlers verändert (Gradientenabstieg).
- **Ablauf:**
 1. **Vorwärtslauf:** Eingangsmuster wird verarbeitet.
 2. **Vergleich mit Soll-Output:** Fehler wird berechnet.
 3. **Rückwärtslauf:** Fehler wird zurück durch das Netz propagiert.
 4. **Gewichtsaktualisierung:** Gewichte proportional zum Fehler angepasst.

Zielfunktion des Backpropagation-Algorithmus

- **Ziel:** Minimierung der Summe der quadratischen Fehler (Least Mean Squares, LMS).

$$\text{Distance(LMS)} = \frac{1}{n} \sum_{p=1}^n (T_p - O_p)^2$$

(T_p : Soll-Wert, O_p : Ist-Wert)

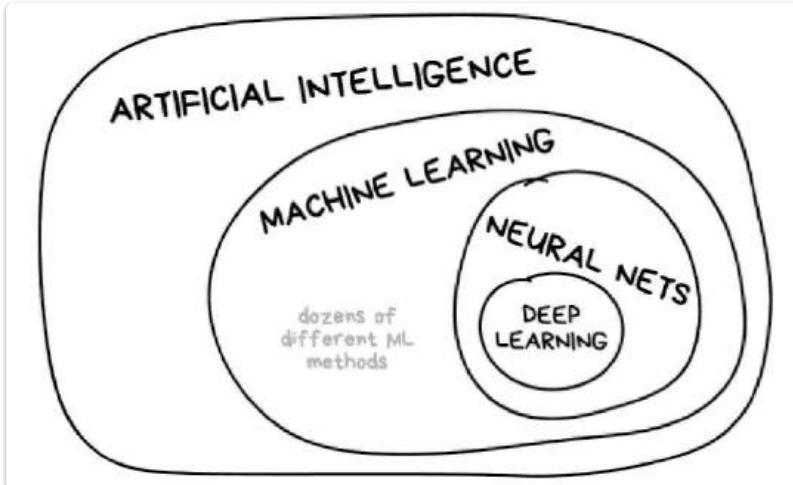
Kernidee des Backpropagation-Verfahrens

- Fehler F ist eine Funktion der Gewichte. Mittels Kettenregel kann ein Gradient auch für Zwischenschichten berechnet werden, um Gewichte anzupassen.

- **Fehlerrückvermittlung:** Fehleroptimierung pflanzt sich von der Ausgabeschicht zurück durch das Netz.

Deep Learning

- **Herausforderung der manuellen Merkmalsgenerierung:** Traditionelle ML-Algorithmen benötigten manuell ausgewählte Merkmale.
- **Deep Learning: End-to-End-Learning:** Das Netzwerk lernt nicht nur die optimale Verarbeitung, sondern auch die optimale Herleitung der Merkmale aus Rohdaten.



1. What exactly is Deep Learning?

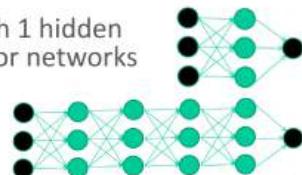
- Deep Learning means using a neural network with several layers of nodes between input and output

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



Deep Learning Architekturen

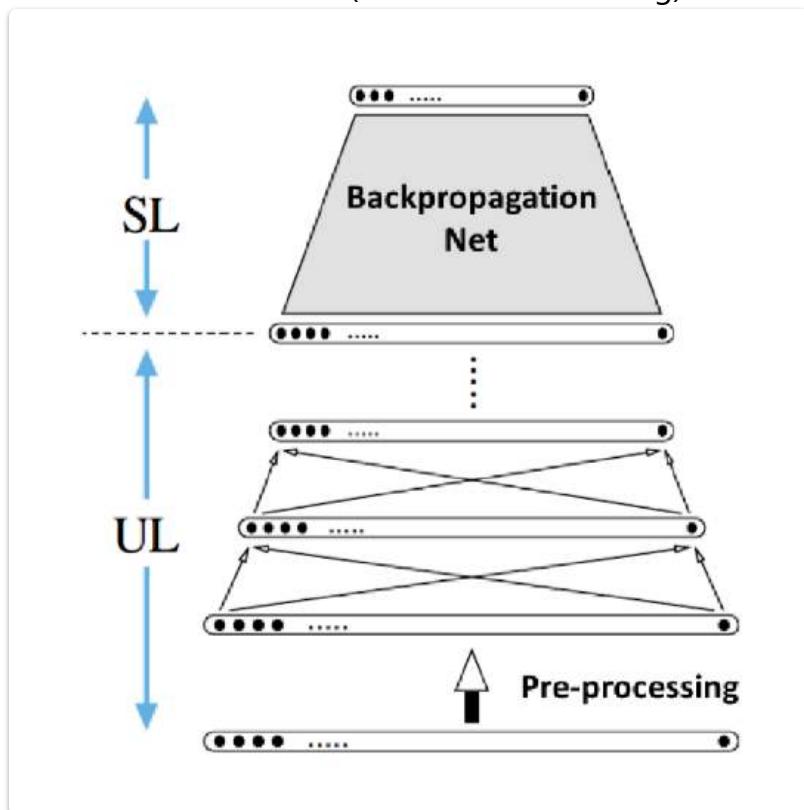
- **Convolutional Neural Networks (CNNs):** Für Computer Vision; lernen Filterkoeffizienten zum Extrahieren hierarchischer Merkmale.
- **Recurrent Neural Networks (RNNs):** Für sequentielle Daten.
- **Generative Adversarial Networks (GANs):** Für generative Aufgaben.

Deep Learning mit vielen Schichten

- **Erfolg:** Viele Schichten von Neuronen.
- **Netzwerkstruktur:** Oft aufgeteilt in:

Unsupervised Learning (UL) zur Vorverarbeitung: Vortraining der Gewichte zur Merkmalsrepräsentation (hierarchische Merkmale).

Supervised Learning (SL) zur Klassifikation/Regression: Training des gesamten Netzes mit überwachten Daten (z.B. Gradientenabstieg).



12. Computational Photography_AI

Disclaimer

Diese komplette Datei, wurde zu 100% AI-Generiert. Als Input und Prompt für die AI war das [vollständige Kapitel der Stoffsammlung](#) mit der Anweisung eine kürzere Version zu erstellen, welche nur für das schnelle Wiederholen geeignet ist. Daher rate ich, diese Version auch nur zum schnellen Wiederholen und nicht zum vom Grund auf Lernen zu verwenden.

Computational Photography

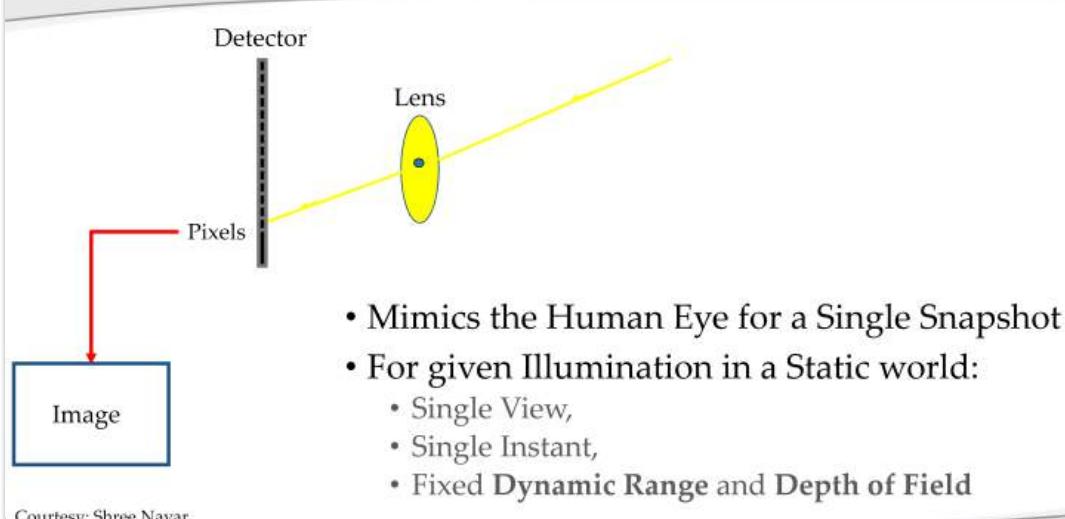
Computational Photography nutzt digitale Kameras und algorithmische Ansätze, um die Möglichkeiten der Fotografie zu erweitern.

- **Ziele:** Neue Aufnahme- und Bearbeitungsmethoden (z.B. extrem hoher Kontrast, verbesserte verwackelte Bilder, Entfernung störender Objekte) entwickeln und physikalische Grenzen traditioneller Kameras überwinden.
- **Grundlage:** Kombination von Software, digitalen Sensoren, moderner Optik und intelligenter Beleuchtung zur Ermöglichung von Bildern, die traditionell nicht realisierbar wären (z.B. große Tiefenschärfe, hohe Auflösung).

Moderne Kameras vs. Traditionelle Kameras

- **Moderne Kameras:** Komplexe Systeme, die über die Grundkonzepte traditioneller Kameras hinausgehen und Leistungen erbringen, die das menschliche Auge übertreffen können.
- **Traditionelle Kameras:** Haben Nachteile bei der Bildaufnahme:
 - Verlust einer Dimension der Szenengeometrie.
 - Informationsverlust über **Reflexionseigenschaften von Szenoberflächen (BRDF)**.
 - "**Helligkeits-Dilemma**": Objektreffektivität und unbekannte Bestrahlungsstärke sind zusammengesetzt.
 - Verlust an brauchbaren Informationen durch Blende (kleiner Schärfebereich).
 - Integration von Licht über Wellenlängen und Zeit (Verlust spektraler/zeitlicher Auflösung).
 - Reduktion von räumlicher Auflösung und Helligkeitsauflösung, was zu einer **Reduktion des Signalumfangs (Dynamic Range)** führt.

Traditional Photography

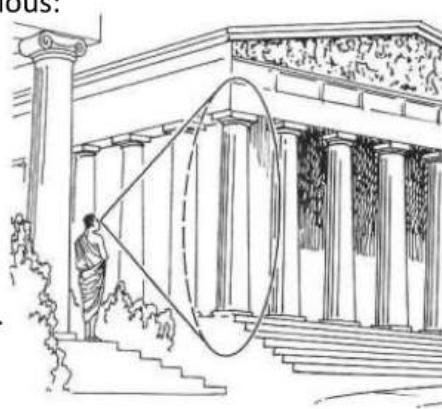


- Core ideas are ancient, simple, and seem obvious:

- **Lighting:** ray sources
- **Optics:** ray bending / folding devices
- **Sensor:** measure light
- **Processing:** assess it
- **Display:** reproduce it

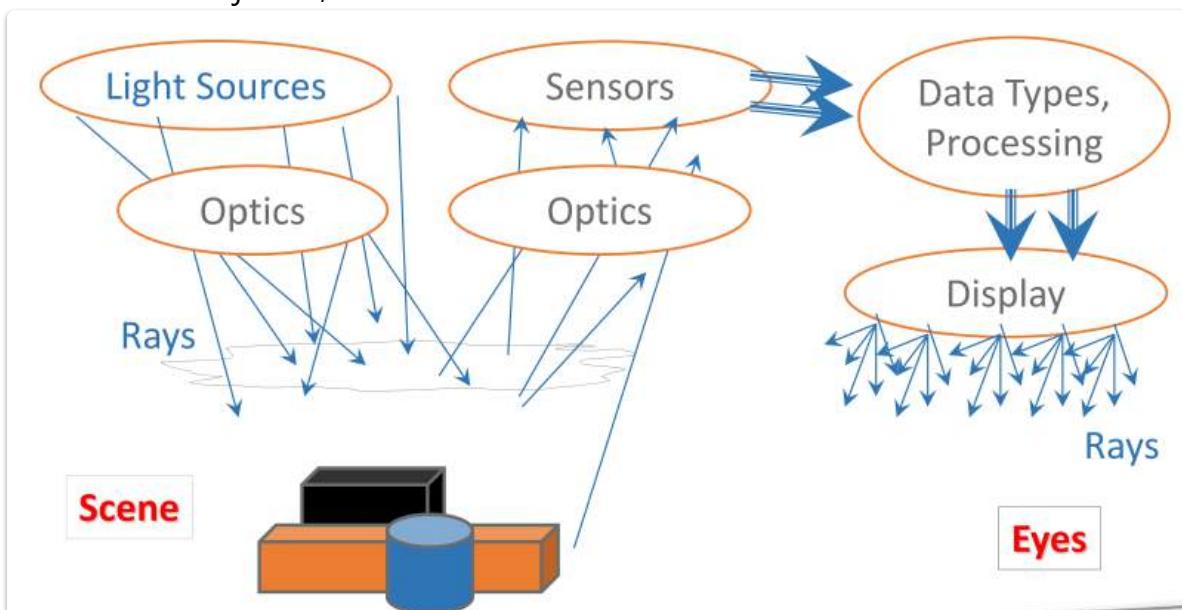
- **Ancient Greeks:**

'Eye rays' wipe the world to feel its contents...



Lichtfeld

Das **Lichtfeld** beschreibt die **Lichtmenge**, die an jedem Punkt im 3D-Raum aus allen Richtungen einfällt. In der Computational Photography fokussiert es auf die reflektierten Strahlen von Objekten, die als Volumina betrachtet werden.



4D-Lichtfeld-Parametrisierung

- **Einfallendes Lichtfeld** auf ein Objekt: $R_i(u_i, v_i, \theta_i, \phi_i)$, parametrisiert durch Position (u_i, v_i) auf einer gedachten Kugel um das Objekt und Einfallsinkel (θ_i, ϕ_i) .
- **Ausgehendes Lichtfeld** vom Objekt: $R_r(u_r, v_r, \theta_r, \phi_r)$.
- **Reflexionsfeld (8D-Funktion)**: Charakterisiert die Reflexionseigenschaften eines Objekts, indem es für jeden einfallenden Lichtstrahl die 4D-Reflexion kodiert. Enthält Informationen zum Rendern des Objekts unter verschiedenen Beleuchtungen und Blickwinkeln.

4D-Lichtfeld-Kameras (Plenoptische Kameras)

Diese Kameras detektieren nicht nur Ort und Intensität, sondern auch den **Einfallsinkel** von Lichtstrahlen auf der Kameraebene.



- **Aufbau:** Nutzen ein **Mikrolinsen-Array** direkt auf dem Sensor.
- **Funktionsweise:** Lichtstrahlen werden unter jeder Mikrolinse nach Einfallsinkel quantifiziert, um ein 4D-Lichtfeld zu rekonstruieren.
- **Nachteil:** Abnehmende Ortsauflösung mit zunehmender Winkelauflösung und umgekehrt, da die Ortsauflösung der Anzahl der Mikrolinsen entspricht.

Ray

- Let's not worry about time and color:



- 5D

- 3D position
- 2D direction

$$R(u_i, v_i, \theta_i, \phi_i ; u_r, v_r, \theta_r, \phi_r)$$

$360 \times 180 \times 180 \times 180 \times 360 \times 180 \times 180 \times 180$

= 4.4e18 measurements

x 6 bytes/pixel (in RGB 16-bit)

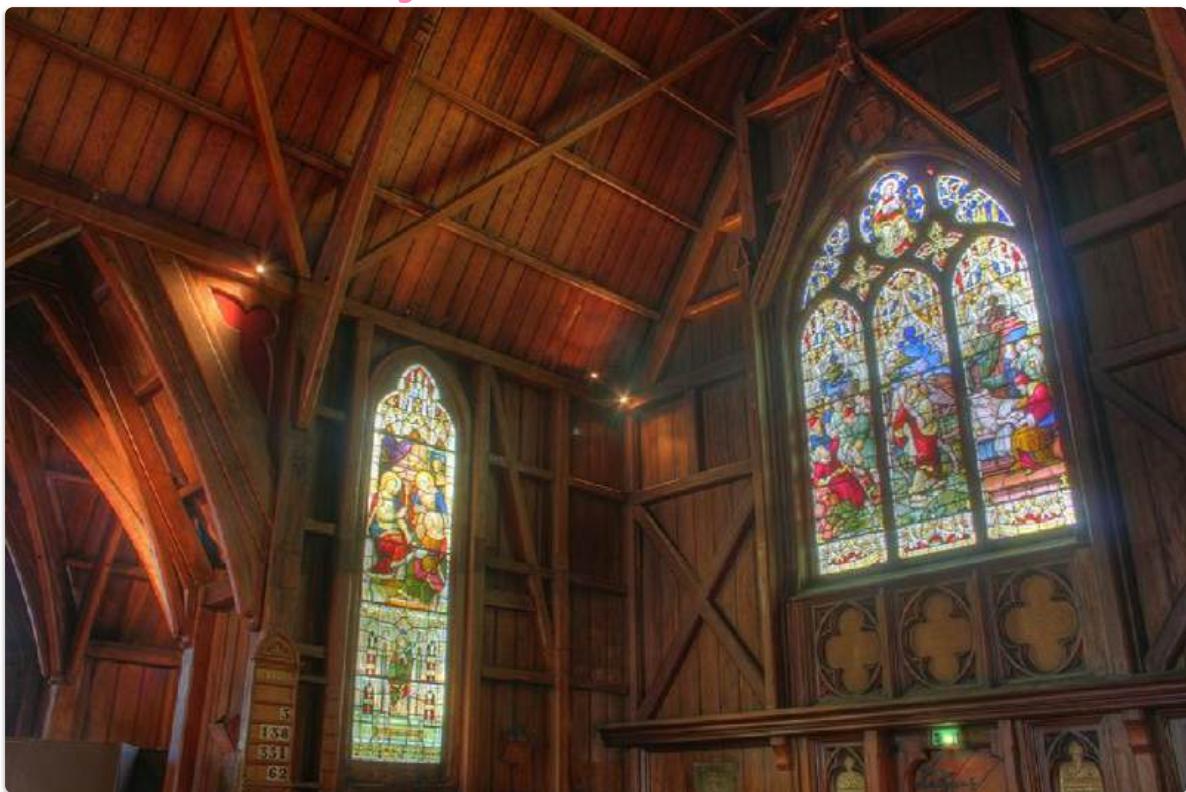
= 26 exabytes (billion GB)

= 1.3 million 20TB hard drives



High Dynamic Range (HDR)

HDR ist ein Bereich von Techniken zur Darstellung eines **größeren Bereichs zwischen hellsten und dunkelsten Regionen eines Bildes** als herkömmliche Methoden.



- **Ziel:** Helligkeitsbereiche in realen Szenen genauer repräsentieren (z.B. von Sonnenlicht bis Sternenlicht).
- **Erzeugung:** Mehrere Aufnahmen der gleichen Szene mit unterschiedlicher Belichtung werden kombiniert.
- **Nutzen:** Kompensiert Detailverlust in hellen oder dunklen Bereichen, die bei Einzelbelichtungen auftreten würden.

Anzeige von HDR-Bildern auf Geräten mit kleinem dynamischen Bereich

- **Tone Mapping:** Techniken zur Abbildung von Farbtönen, die den gesamten Kontrast reduzieren, um HDR-Bilder auf Geräten mit begrenztem Dynamikbereich (z.B. Monitore)

darzustellen.

- **Ziel:** Kontrastreduktion bei gleichzeitigem Erhalt von Bilddetails und Farbwirkung.



Fotomontage/Komposition

Fotomontage (Image Compositing) ist das **Schneiden und Zusammenfügen verschiedener Fotos zu einem neuen Bild**, um die Illusion zu erzeugen, dass alle Elemente Teil derselben Szene sind.



- Realisierung heutzutage mit Bildbearbeitungssoftware.
- Im Film bekannt als "chroma key", "blue screen" oder "green screen".

Image Inpainting

Inpainting ist ein Bildbearbeitungsprozess zur **Rekonstruktion von schlecht erhaltenen oder fehlenden Bild- oder Videoteilen**.



- **Analogie:** Arbeit eines Bildrestaurators in der Malerei.
- **Digitale Welt:** Anwendung von Algorithmen zum Ersetzen verlorener/fehlerhafter Bilddaten.
- **Anwendungen:** Objektentfernung, Retuschierung von beschädigten Bildern (Risse, Kratzer), Hinzufügen/Entfernen von Bildteilen (z.B. Aufnahmedatum).
- **Ziel:** Produzieren eines veränderten Bildes, in dem die bearbeitete Region nahtlos in das restliche Bild übergeht, sodass die Bearbeitung nicht auffällt.

Warping

Warping ist die **Manipulation eines Bildes**, bei der jede Form **räumlich verändert** wird.

Image filtering: change *range* of image

$$g(x) = T(f(x))$$

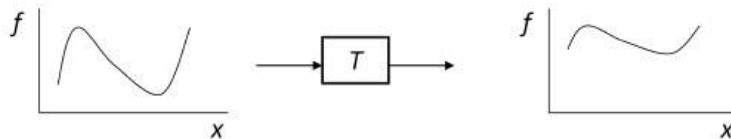
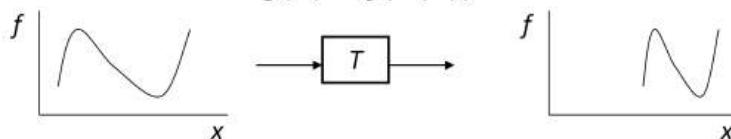


Image warping: change *domain* of image

$$g(x) = f(T(x))$$



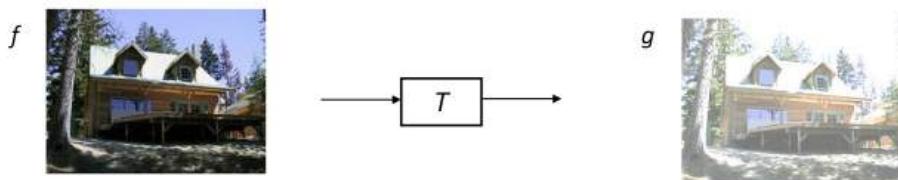
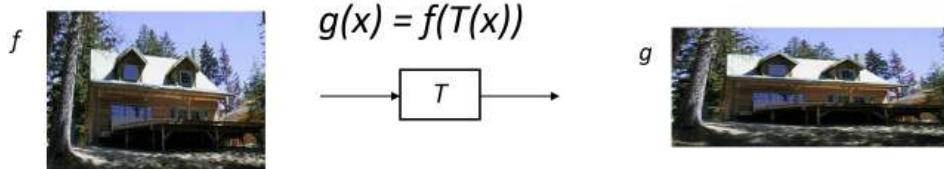


Image warping: change **domain** of image



- **Anwendung:** Korrektur von Bildverzerrungen oder für kreative Zwecke (z.B. Morphing).
- **Funktionsweise:** Punkte werden abgebildet, ohne deren Farbe zu ändern.
- **Mathematische Beschreibung:** Das Zielbild $I_d(x)$ wird aus dem Quellbild $I_s(x)$ mittels der Warpingfunktion $W(x; p)$ und Parametern p bestimmt: $I_d(x) = I_s(W(x; p))$.
- **Zwei Formen von Transformationen:**
 - **Affine Transformation:** Erhält gerade Linien und Distanzverhältnisse auf einer Linie. Längen und Winkel werden nicht zwingend erhalten. Beispiele: Translation, Rotation, Scherung.

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

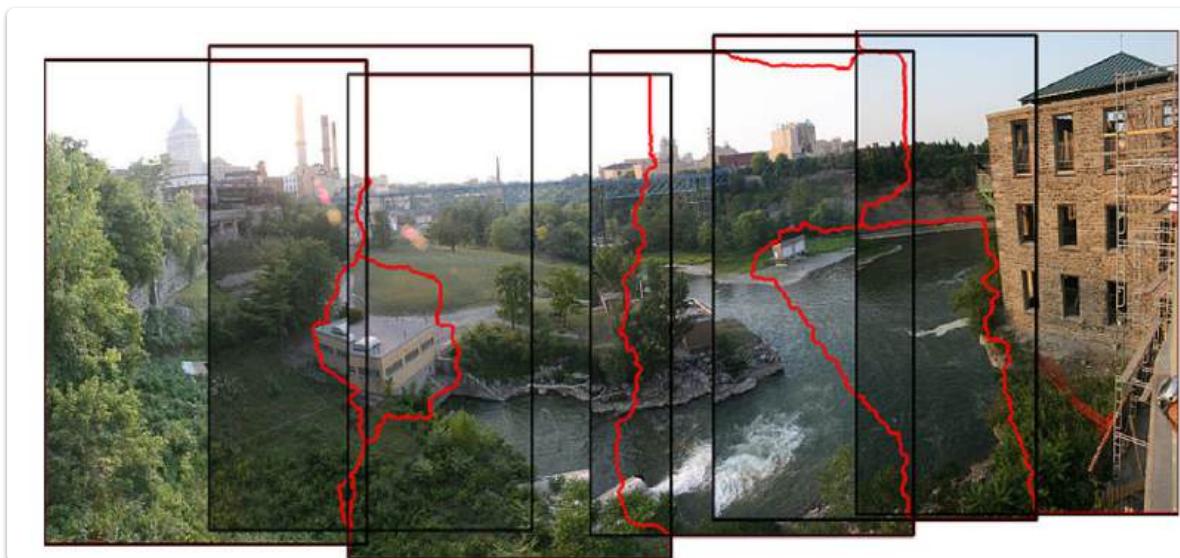
- **Projektive Transformation (Homographie):** Wird in der projektiven Geometrie verwendet; erhält keine Größen oder Winkel. Jedes Bildpaar einer ebenen Fläche im Raum steht durch eine projektive Transformation zueinander in Beziehung.

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Identity	$W(x) = x$	
Translation	$W(x; t) = x - t$	
Rigid	$W(x; R, t) = Rx - t$	
Similarity	$W(x; R, a, t) = aRx - t$	
Affine	$W(x; A, t) = Ax - t$	
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$		

Bildmosaik (Image Mosaicing / Stitching)

Bildmosaik/Stitching ist der Prozess, bei dem mehrere Bilder mit überlappenden Bereichen zu einem **hochauflösenden Panoramabild** kombiniert werden.



- **Voraussetzung:** Kamera muss um den **Fokuspunkt** rotiert werden, um Parallaxenfehler zu vermeiden.
- **Drei Bestandteile:**
 1. **Bildregistrierung:** Bestimmen korrespondierender Merkmale, um Bilder korrekt zu überlagern.
 2. **Kalibrierung:** Minimiert Differenzen zwischen idealem Linsenmodell und realem Linsensystem (korrigiert Verzerrungen, Belichtungszeiten, Vignettierung, chromatische Aberrationen).

3. **Blending:** Korrigiert Abweichungen aus der Kalibrierung und bildet Aufnahmen auf ein gemeinsames Ausgabebild ab. Farben werden angepasst und Übergänge sollen nahtlos sein ("no seams").

Image Morphing

Image Morphing ist ein Bildeffekt, der ein Bild **nahtlos in ein anderes verwandelt**.



- **Anwendung:** Surrealistische Sequenzen, z.B. Verwandlung einer Person in eine andere.
- **Methode:** Heutzutage durch gleichzeitiges **Verzerren und Überblenden** von Bildern basierend auf **markierten korrespondierenden Punkten (Keypoints)**.
- **Beispiel (Gesichts-Morphing):** Keypoints (Nase, Augen) werden in beiden Gesichtern markiert. Das erste Gesicht wird verzerrt, um die Form des zweiten anzunehmen, während es ausgeblendet und das zweite eingeblendet wird.
- **Höher entwickelte Techniken:** Graduelles Überblenden verschiedener Bildteile.