



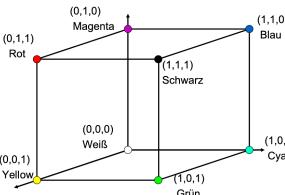
## 7.2.2. CMY-Modell (subtraktiv)

Beim **Farbdruck** empfängt das Auge nur solche Anteile des weißen Lichtes, die **reflektiert** werden. Ein CMY-Tripel beschreibt, wie viel von den Grundfarben Cyan, Magenta und Yellow reflektiert bzw. von den Grundfarben Rot, Grün und Blau absorbiert wird (Weiss reflektiert alles, wenn ich rot malen möchte, muss ich das wegnnehmen).

### Beispiele:

- $(0, 0, 0)$  absorbiert nichts, ergibt Weiss
- $(0, 0, 1)$  absorbiert Blau, ergibt Gelb

**Beispiel Farb-Subtraktion:**  $(0, 1, 0) - (0, 0, 1) = (0, 1, 1)$



### Umrechnung

Umrechnung zwischen dem CMY-Modell und dem RGB-Modell erfolgt in **Vektorschreibweise** über die Subtraktion.

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} C \\ M \\ Y \end{pmatrix}, \quad \begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

## 7.2.3. CMY-Modell

Verwendet zusätzlich **schwarze Farbe**, den **Key**. Wird beim Drucken verwendet.

### Umrechnungsnäherung:

$$\begin{aligned} K &:= \min(C, M, Y) & K &= \min(10, 15, 55) = 10 \\ C' &:= C - K & C' &= 10 - 10 = 0 \\ M' &:= M - K & M' &= 15 - 10 = 5 \\ Y' &:= Y - K & Y' &= 55 - 10 = 45 \end{aligned}$$

Einer der Werte wird so immer 0.

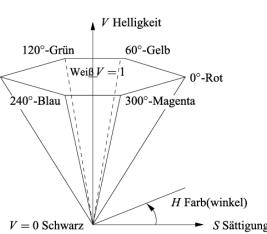
## 7.2.4. YUV-Modell

Wurde verwendet, um Fernsehbilder gerätespezifisch entweder Farbig oder S/W anzuzeigen. Ein Farbwert wird durch ein YUV-Tripel beschrieben, wobei Y die Helligkeit (Luminanz) bezeichnet, und U, V die Farbdifferenzen (Chromianz).

## 7.2.5. HSV-Modell

Beschreibt jede Farbe durch das Tripel HSV.

**Hue:** Farbtön, **Saturation:** Sättigung, **Value:** Helligkeit



### Umrechnung RGB nach HSV

$$\text{RGB-Einheitswürfel, } \frac{256}{64} = 4 \text{ etc.}$$

$$\text{rgb}(64, 128, 32) \equiv (1/4, 1/2, 1/8)$$

$$v = \max(r, g, b) = 1/2 = 50\%$$

$$\min := \min(r, g, b) = 1/8$$

$$s = \frac{v - \min}{v} = \frac{4/8 - 1/8}{1/2} = 3/8 = \frac{3}{4} = 75\%$$

$$h = \left(1 + \frac{v - r}{v - \min}\right) \cdot 60^\circ = \left(1 + \frac{2}{3}\right) \cdot 60^\circ = 100^\circ$$

$$\text{hsv} = (100, 75, 50) \blacksquare$$

**Dominante Grundfarbe:** Grün, weil  $v = g$ . **Schwächste Farbe:** Blau, weil  $\min = b$  = Farbe ist gelb-grünlich.

### Umrechnung HSV nach RGB

$$\max(r, g, b) = v$$

$$\min(r, g, b) = v - s \cdot v$$

$$\text{mitte}(r, g, b) = v - \left(\frac{h}{60^\circ} - 1\right) \cdot v - \min$$

## 8. 3D-TRANSFORMATIONEN

In 3D-Transformationen wird eine  $4 \times 4$ -Matrix mit homogenen Koordinaten verwendet.

### 8.1. TRANSLATION

Mit homogenen Koordinaten lässt sich der um den Translationsvektor  $\vec{t} = (t_x \ t_y \ t_z)^T$  verschobene Punkt  $P = (x, y, z)$

$$(x', y', z') := (x + t_x, y + t_y, z + t_z)$$

in der folgenden Form darstellen:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} T & \vec{t} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

### 8.2. SKALIERUNG

Gegeben sind drei Skalierungsfaktoren  $s_x, s_y, s_z \neq 0$ . Der Fixpunkt der Skalierung liegt im Ursprung:

$$(x', y', z') := (s_x \cdot x, s_y \cdot y, s_z \cdot z)$$

Die daraus resultierende Transformationsmatrix lautet:

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

Fixpunkt liegt nicht im Ursprung, sondern bei  $(Z_x, Z_y, Z_z)$ : Zuerst Translation um  $(-Z_x, -Z_y, -Z_z)$ , dann Skalierung um  $(s_x, s_y, s_z)$ , dann Rücktranslation um  $(Z_x, Z_y, Z_z)$ :

$$T(Z_x, Z_y, Z_z) \cdot S(s_x, s_y, s_z) \cdot T(-Z_x, -Z_y, -Z_z)$$

### 8.2.1. Rotation im Gegenuhrzeigersinn

(Für Uhrzeigersinn müssen Vorzeichen vom Sinus vertauscht werden)

#### Z-Achse

Rotation	Transformationsmatrix
$x' := x \cdot \cos(\delta) - y \cdot \sin(\delta)$	$R_z(\delta) = \begin{pmatrix} \cos(\delta) & -\sin(\delta) & 0 & 0 \\ \sin(\delta) & \cos(\delta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$y' := x \cdot \sin(\delta) + y \cdot \cos(\delta)$	
$z' := z$	

#### X-Achse

Rotation	Transformationsmatrix
$x' := y \cdot \cos(\delta) - z \cdot \sin(\delta)$	$R_x(\delta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\delta) & -\sin(\delta) & 0 \\ 0 & \sin(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$y' := z \cdot \cos(\delta) + y \cdot \sin(\delta)$	

#### Y-Achse

Rotation	Transformationsmatrix
$x' := z \cdot \sin(\delta) + x \cdot \cos(\delta)$	$R_y(\delta) = \begin{pmatrix} \cos(\delta) & 0 & \sin(\delta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\delta) & 0 & \cos(\delta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$y' := z \cdot \cos(\delta) - x \cdot \sin(\delta)$	

#### Rotation um beliebige Achse

Rotationsachse stimmt nicht mit einer der Koordinatenachsen überein

### 0. Einheitsvektor $\vec{u}$ der Rotationsachse $\vec{v}$ (verläuft durch die Punkte $P_1, P_2$ ) berechnen:

$$\vec{v} = P_2 - P_1 = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix}$$

### 1. Translation von Rotationsachse und Objekt, sodass **Rotationsachse durch den Ursprung** läuft:

$$\vec{v}' = (P_1', P_2')$$

$$P_1' = (0, 0, 0), P_2' = (x_2 - x_1, y_2 - x_1, z_2 - z_1)$$

### 2. **Rotation der Rotationsachse** um die x-Achse in die $xy$ -Ebene:

$$d = \sqrt{b^2 + c^2}, \cos(\alpha) = \frac{c}{d}, \sin(\alpha) = \frac{b}{d}$$

### 3. **Rotation der Rotationsachse** um die y-Achse in die $xy$ -Ebene

$$\cos(\beta) = \cos(360^\circ - \beta) = d, \sin(\beta) = -\sin(360^\circ - \beta) = -a$$

### 4. **Rotation des Objekts** um die z-Achse mit Winkel $\delta$ :

$$R_z(\delta) = \begin{pmatrix} \cos(\delta) & -\sin(\delta) & 0 \\ \sin(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### 5. **Rücktransformation** des gedrehten Objekts durch Anwendung der inversen Transformationen der Schritte 3, 2 und 1.

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} x'' \\ y'' \\ z'' \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} x''' \\ y''' \\ z''' \\ 1 \end{pmatrix}$$

## 8.3. TRANSFORMATION DER NORMALENVEKTOREN

Die Normalenvektoren müssen bei der Transformation von Objektpunkten ebenfalls abgebildet werden. Wenn diese Transformation eine **nicht-uniforme Skalierung** (Verzerrung) ist, bleiben die **Winkel** zwischen einzelnen Flächen **nicht erhalten**. Damit der Normalenvektor  $\vec{n}$  trotzdem weiterhin senkrecht zur Fläche steht, muss dieser mit der **transponierten Inversen** der Transformationsmatrix  $M$  transformiert werden.

$$(M^{-1})^T \cdot \vec{n} = \vec{n}'$$

## 8.4. FRAKTALE

Fraktale sind geometrische Formen, die sich durch **Selbstähnlichkeit** auszeichnen: wenn man hineinzoomt, sieht das **Teilstück ähnlich** oder **gleich** wie das **gesamte Gebilde** aus.

### 8.1. FRAKTALE DIMENSIONEN

Ein selbstähnliches Objekt hat **Dimension D**, falls es in **N identische Kopien** unterteilt werden kann, die jeweils **skaliert** sind mit dem Faktor  $r = 1/N^{1/D}$ .

### 9.1. MORPHOLOGIE

Morphologische Operationen **verändern die Form** von Bildobjekten (Distanz von Linien ändern, Vereinen/Trennen von Elementen, Unreinheiten eliminieren).

- **Operator:** Vereinigung, Differenz, Schnitt
- $f_s$ : Menge von Pixeln
- $s$ : Nachbarschaft, definiert durch Strukturelement

**Voraussetzungen für Basisoperationen:**

- Die Mengen  $f_s$  entsprechen Objekten oder Mustern im Bild
- Das Bild ist ein Binär- oder Grauwertbild
- Das Strukturelement (Das Element, das über das Bild gelegt wird) besteht aus 0 und 1, wobei die 1 die Pixel der Nachbarschaft definiert.

### 10.1. DILATATION

Das Strukturelement wird über jeden Pixel des Originalbildes gelegt und eine logische **und-Verknüpfung** zwischen dem Binärbild  $f$  und dem Strukturelement  $s$  wird durchgeführt:

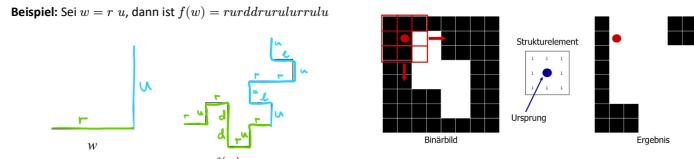
$$g = f \oplus s$$

### Wirkung

Objekte werden **vergrößert** und können **verschmelzen**. Löcher können **geschlossen** werden, unregelmäßige Objekte werden von der **Form** her **ausgeglichener**.

### Beispiel

Das Strukturelement wird auf jeden Pixel angewendet. Da im Strukturelement alle Nachbarn 1 sind, wird der Ursprung weiß eingefärbt, wenn einer der 8 Nachbarn bereits weiß ist.



### 10.2. EROSION

Das Strukturelement wird über jeden Pixel des Originalbildes gelegt und eine logische **und-Verknüpfung** zwischen dem Binärbild  $f$  und dem Strukturelement  $s$  wird durchgeführt:

$$g = f \ominus s$$

### Wirkung

Objekte werden **verkleinert** und können **auseinanderfallen**. Löcher werden **größer**, Objekte werden **gelöscht**, wenn sie **kleiner** als das Strukturelement sind.

### Beispiel

Alle Pixel des Strukturelements müssen auf 1 gesetzt sein, damit der Ursprung weiß bleibt.



### 9.5. MANDELBROT-MENGE

Besteht aus komplexen Zahlen  $c$ , die beim Startwert  $z = 0$  zu einer beschränkten Folge führen. Dann definiert man:

$$\text{farbe}(c) = \begin{cases} \text{schwarz,} & \text{wenn Folge beschränkt bleibt} \\ \text{weiss,} & \text{wenn Folge nicht beschränkt bleibt} \end{cases}$$

### 9.6. ITERIERTE FUNKTIONENSYSTEME

Besteht in der Lage, mit **wenigen Regeln komplexe, natürlich ausscheinende Bilder** zu erzeugen.

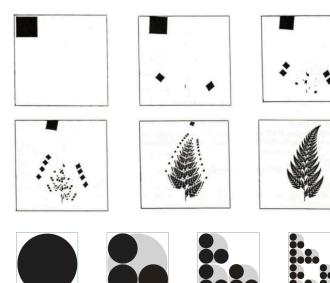
Jede Transformation wird definiert durch eine  $2 \times 2$ -**Deformationsmatrix A**, einen **Translationsvektor b** und eine **Anwendungswahrscheinlichkeit w**, wodurch ein Punkt  $x$  abgebildet wird auf  $Ax + b$ .

### 10.4. CLOSING

**Closing** mit anschließender **Dilation**:  $A \circ B = (A \ominus B) \oplus B$

Wirkung: **Rundet Konturen, verbindet nahe Objekte, füllt Löcher** und Einbuchtungen, die **kleiner** als das Strukturelement sind.

## Beispiele hierfür sind der **Farn** und das **Sierpinsky-Dreieck**:

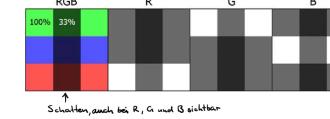


## 10.5. OBJEKTE FREISTELLEN

**Setzung:** Objekte durch Erkennung eines Hintergrundes **freistellen**, **Hintergrund** durch andere Bildquelle **ersetzen**.

### 10.5.1. HINTERGRUNDSERSETZUNG

**Probleme:** Schatten verfälschen Farbtöne. Störungen wie Rauschen oder Kompressionsartefakte sorgen für Ungleichmäigkeiten. Farbtöne und ihre Schattierungen sind in allen **RGB** kodiert.



**Umkehrung:** Umwandlung in **HSV**, um Farbtönen und Sättigung zu trennen. Ermöglicht einfache Auswahl eines Farbtönenbereiches **unbeeinflusst** von Schattierungen.

### Bereinigung der Selektionsmaske

Durch **Störungen** und schlechte **Farbonverteilung** können Fehler, Löcher oder unschöne Kanten entstehen. Durch **Anwendung** diverser **Filter** oder **morphologischen Operationen** können diese Effekte **minimiert** werden.

### Anwendungsschritte

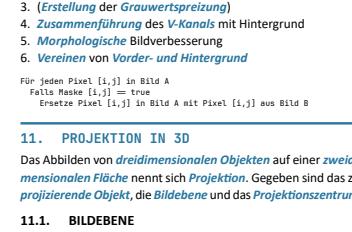
1. Konvertieren des Vordergrundes in **HSV**
2. Erstellen der **Alpha-Maske**
3. (Erstellung der **Grauwertspiegelung**)
4. Zusammenführen des **Kanals** mit Hintergrund
5. **Morphologische** Bildverbesserung
6. **Vereinen** von **Vorder- und Hintergrund**

Für jeden Pixel  $[i, j]$  in Bild A  
Falls Maske  $[i, j] = 1$  true  
Setze Pixel  $[i, j]$  in Bild A mit Pixel  $[i, j]$  aus Bild B

## 11. PROJEKTION IN 3D

Das Abbilden von **dreidimensionalen Objekten** auf einer **zweidimensionalen Fläche** nennt sich **Projektion**. Gegeben sind das zu **projizierende Objekt**, die **Bildecke** und das **Projektionszentrum**.

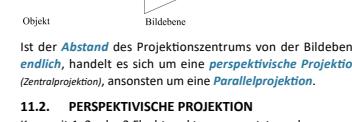
### 11.1. BILDEBENE



Ist der Abstand des Projektionszentrums von der Bildecke **endlich**, handelt es sich um eine **perspektivische Projektion** (Zentralprojektion), ansonsten um eine **Parallelprojektion**.

### 11.2. PERSPEKTIVISCHE PROJEKTION

Kann mit 1, 2 oder 3 Fluchtpunkten umgesetzt werden.



### 11.2.1. Anwendung der Strahlensätze

- **Bildecke:** xy-Ebene
- **Projektionszentrum:** neg. z-Achse im Punkt Z = (0, 0, -a)
- **Gegeben:** P = (x, y, z)
- **Gesucht:** projizierter Bildpunkt P' = (x', y', 0)

Die **homogenen Koordinaten des projizierten Punktes** lautet:

$$P' = \left( \frac{x}{w}, \frac{y}{w}, 0, 1 \right) = \begin{pmatrix} w \\ x \\ y \\ 1 \end{pmatrix} \quad \text{mit } w = 1 + z/a$$

Die **homogenen Koordinaten des projizierten Punktes** lautet:

$$P_{\text{persp},xy}(-a) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{a} & 1 \end{pmatrix}$$

## 1. Modeling (MC → WC):

Beschreibe Polygone in Weltkoordinaten

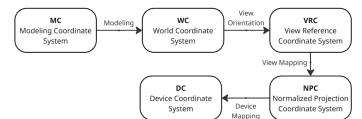
## 2. View Orientation (WC → VR):

Überföhre Szene in Kameraperspektive

## 3. View Mapping (VRC → NPC):

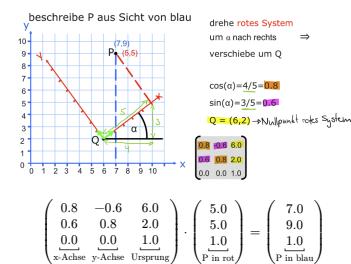
Transformiere Szene in Einheitswürfel

## 4. Device Mapping (NPC → DC): Projiziere Szene auf Bildschirm



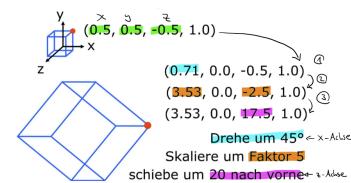
## 12.3. KOORDINATENSYSTEMWECHSEL

Konvertierung von lokalem zu globalen Koordinatensystem.



## 12.3.1. Modeling

Das **Aordnen von Objekten** aus dem **Modellkoordinatensystem** zu einer Szene im **Weltkoordinatensystem**. Die Objekte erhalten aus der **Grundform** ihre individuelle Größe, Orientierung und Position durch **Translation, Rotation und Skalierung**.



## 12.3.2. View Orientation

Zur **Affidlung** einer dreidimensionalen Szene aus den Weltkoordinaten auf dem Bildschirm muss eine **Betrachtersicht** (View Orientation) definiert werden. Diese besteht aus den Parametern:

- **PRP**: Betrachterstandpunkt (Projection Reference Point)
- **VRP**: Blickrichtung (View Reference Point)
- **VUV/VUP**: vertikale Orientierung (View Up Vector/View Up Point)
- **Blckwinkel**: Brennweite

Wenn die Achsen der Kamera mit  $U - V - N$  und der Nullpunkt der Kamera mit  $VRP$  bezeichnet wird (Siehe Kameraklub weiter oben), ist die **Transformation von lokale in globale Koordinaten** wie folgt definiert:

$$M = \begin{pmatrix} U_x & V_x & N_x & VRP_x \\ U_y & V_y & N_y & VRP_y \\ U_z & V_z & N_z & VRP_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Damit die global definierte Szene aus **Kamerasischt** angesehen wird (Wechsel von globalen zu lokalen Koordinaten – also umgekehrt zur obigen Rechnung), muss auf die Modellgeometrie die **inverse Transformation**  $T = M^{-1}$  angewendet werden.

$$M^{-1} = \begin{pmatrix} U_x & U_y & U_z & -(U \cdot VRP) \\ V_x & V_y & V_z & -(V \cdot VRP) \\ N_x & N_y & N_z & -(N \cdot VRP) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Beispiel Kameratransformation

Kamera schaut von  $p_1 = (2, 2, 0)$  auf  $p_2 = (1, 1, 0)$ . Berechne Transformationsmatrix der Kamera und Transformationsmatrix, mit der globale Koordinaten in das Kamerakoordinatensystem transformiert werden. Die Z-Achse der Kamera zeigt in Richtung Blickrichtung  $p_1 \rightarrow p_2$ . Die x-Achse sei parallel zur globalen XY Ebene.

Es gilt:  $N = Z$ -Achse der Kamera,  $V = Y$ -Achse der Kamera,  $U = X$ -Achse der Kamera.  $PRP = \text{Kamerapunkt}$ ,  $VRP = \text{Angeschauter Punkt}$ .

Folgendes kann von der Zeichnung abgelesen werden: Zeigt Veränderung der globalen Koordinaten (+1 zeigt in globale +Richtung, -1 zeigt in globale -Richtung):

$$N = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \quad V = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \quad U = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \quad PRP = VRP$$

Normalisieren mit  $\sqrt{x^2 + y^2 + z^2}; U$  zu  $\hat{U}$

$$\sqrt{(-1)^2 + 1^2 + 0^2} = \sqrt{2} \Rightarrow \hat{U} = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

Normalisierte Werte Eintragen in  $M$  und  $M^{-1}$ :

$$M = \begin{pmatrix} -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 1 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M^{-1} = \begin{pmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & -\frac{2}{\sqrt{2}} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

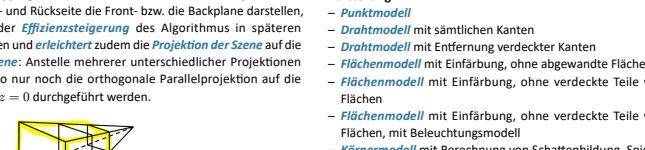
Beispieldatum:

$$-(\hat{U} \cdot VRP) = -\left(\begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}\right) = \left(-\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} + 0\right) = 0$$

## 12.3.3. View Mapping

View Volume: Bildraum, der durch das View Window und die gewählte Projektion definiert wird.

Das **Mapping des View Volume** auf den **Einheitswürfel**, dessen Vorder- und Rückseite die Front- bzw. die Backplane darstellen, dient der **Effizienzsteigerung** des Algorithmus in späteren Schritten und erleichtert zudem die **Projektion der Szene** auf die **Bildebene**: Anstelle mehrerer unterschiedlicher Projektionen muss so nur noch die orthogonale Parallelprojektion auf die Ebene  $z = 0$  durchgeführt werden.

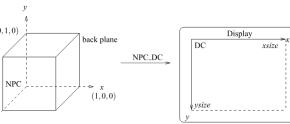


Dieses Mapping wird in einigen komplizierten mathematischen Schritten durchgeführt:

- **Verschiebung** des PRPs in den Ursprung:  $z := z - d$
- **Spiegelung** an der xy-Ebene:  $z := -z$
- **Überführung** in Pyramidenstumpf ( $90^\circ$  Winkel von PRP zu  $V_{Front}$  und  $V_{Back}$ )
- **Die regelmäßige Pyramide** ( $45^\circ, 90^\circ, 45^\circ$ ) kann nun in den **Einheitswürfel** transformiert werden.

## 12.3.4. Device Mapping

Die Abbildung muss nur die  $x$ - und  $y$ -Koordinaten aus dem NPC so in die **Bildschirmkoordinaten DC** (Device Coordinate System) transformieren, dass eine anschließende Rundung die **ganzzahligen Koordinaten der Pixel** ergibt.



Auch ist **DC** oft ein **linkshändiges Koordinatensystem** ( $y$ -Achse zeigt nach unten, der Ursprung ist oben links), dafür muss die Abbildung ebenfalls noch angepasst werden.

Die Transformationsmatrix entspricht einer Skalierung um den Vektor ( $\text{ksize}, -\text{ysize}, 1$ ) konkateniert mit einer Translation des Ursprungs in die linke untere Ecke des Bildschirms ( $0, \text{ysize}, 0$ ).

$$T_{\text{NPC,DC}} = \begin{pmatrix} \text{ksize} & 0 & 0 & 0 \\ 0 & -\text{ysize} & 0 & \text{ysize} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



## 12.3.5. Clipping

Jedes Polygon, dass die Viewing Pipeline durchläuft, kostet **Rechenaufwand**. Deshalb macht es Sinn, den **unsichtbaren Teil** einer Szene so früh wie möglich **loszuwerden**. Es gibt verschiedene Optionen dafür.

**Clipping im WC / NPC**  
Im WC muss an den sechs Flächen des Frustums (Kegelstumpf) geklippt werden, während im NPC an den sechs **einfachen** Flächen des Einheitswürfels mit Cohen-Sutherland (Kapitel 4.1. «Cohen-Sutherland Algorithmus») geklippt werden kann. Dafür ist das eigentliche Clipping im zweiten Fall etwas aufwändiger.

## Umgebungsclipping

Bietet **Effizienzsteigerung** des Clippings, indem ein **Cluster** von mehreren, komplexen Objekten mit einem **grossen Quader** umgeben wird. Ergibt ein erster **Clipping-Test**, dass dieser Quader **ausserhalb** des Frustums liegt, erübrigen sich die Clipping-Abfragen seiner inneren Objekte.

## 13.0. OBJEKTE IN 3D

Für 3-dimensionale Objekte gibt es mehrere Möglichkeiten der **Repräsentation** (d.h. Definition des Objekts) und der **Darstellung** (d.h. Projektion des Objekts auf den Bildschirm).

## Repräsentation:

- **Elementarobjekt mit Definitionspunkten** (veraltet)
- **Drahtmodell** (veraltet)

- **Flächenmodell** mit Punkt- und Flächenliste und Normalen  
- **CSC** (constructive solid geometry) mit mengentheoretischer Verknüpfung von Elementarobjekten (Ein Objekt besteht aus Addition/Subtraktion mehrerer Elementen, z.B. Rohr = Grosser Zylinder - kleiner Zylinder).

## Darstellung:

- **Punktmödell**
- **Drahtmodell** mit sämtlichen Kanten
- **Grenzflächenmodell** mit Entfernung verdeckter Kanten
- **Flächenmodell** mit Einfärbung, ohne abgewandte Flächen
- **Flächenmodell** mit Einfärbung, ohne verdeckte Teile von Flächen
- **Flächenmodell** mit Einfärbung, ohne verdeckte Teile von Flächen, mit Beleuchtungsmodell
- **Körpermodell** mit Berechnung von Schattenbildung, Spiegelungen und Brechungen.

## 13.1. FLÄCHENMODELL

Objekte werden durch approximierte oder analytische Flächen, z.B. durch eine **Liste** von konvexen Polygonen, repräsentiert. Beim **Würfel** verbinden die Kanten die Eckpunkte, bei einer **Kugel** werden die Längen- und Breitenkreise durch  $n$ -Ecke angenähert, wobei man  $n$  die Qualität der Approximation steigt.

## Beispiel Tetraeder:

Punktliste	Flächenliste
$P_1 : (x_1, y_1, z_1)$	$F_1 : P_1, P_2, P_4$
$P_2 : (x_2, y_2, z_2)$	$F_2 : P_1, P_4, P_3$
$P_3 : (x_3, y_3, z_3)$	$F_3 : P_1, P_2, P_3$
$P_4 : (x_4, y_4, z_4)$	$F_4 : P_4, P_2, P_3$

## 13.2. POLYEDER

Eignet sich zur Verwaltung der räumlichen Anordnung von Objekten im dreidimensionalen Raum. Wenn ein Gebiet nicht **komplett** gefüllt ist, wird es gevierielt. Die Viertel werden im Uhrzeigersinn abgearbeitet, start oben links. (Zumindest in diesem Beispiel).

**Grauer Knoten:** Enthält teilweise Objekte.

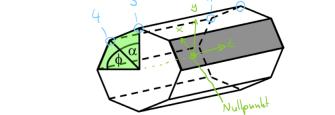
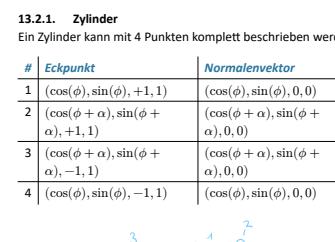
## 13.3.2. Octree

Eignet sich zur Verwaltung der räumlichen Anordnung von Objekten im dreidimensionalen Raum. Wenn ein Gebiet nicht **komplett** gefüllt ist, wird es gevierielt. Zusätzlich zu der Euler-Gleichung müssen BREPs folgende Voraussetzungen erfüllen:

- Jede **Kante** trennt **genau zwei Flächen** (evtl. auch gewölbte Fläche)
- Um jede **Ecke** existiert ein **geschlossener Ring** von **Flächen**
- **Flächen** können sich nur an einer **gemeinsamen Ecke oder Kante** schneiden

## Beispiel Zylinder:

Ein Zylinder kann wie folgt beschrieben werden:



## 13.2.1. Zylinder

Ein Zylinder kann mit 4 Punkten komplett beschrieben werden.

#	Eckpunkt	Normalenvektor
1	$(\cos(\phi), \sin(\phi), +1, 1)$	$(\cos(\phi), \sin(\phi), 0, 0)$
2	$(\cos(\phi + \alpha), \sin(\phi + \alpha), +1, 1)$	$(\cos(\phi + \alpha), \sin(\phi + \alpha), 0, 0)$
3	$(\cos(\phi + \alpha), \sin(\phi + \alpha), -1, 1)$	$(\cos(\phi + \alpha), \sin(\phi + \alpha), 0, 0)$
4	$(\cos(\phi), \sin(\phi), -1, 1)$	$(\cos(\phi), \sin(\phi), 0, 0)$

Das Konzept wird **Octree** genannt, weil ein Würfel jeweils in 8 **Teilstücke** geteilt wird, und Nodes im Baum somit jeweils **8 Child Nodes** haben.

Objekt:

1. Zeile: 2, 3, 7, 6  
2. Zeile: 0, 1, 5, 4  
3. Zeile: 4, 5, 6, 7

Oktree-Darstellung:

Gesamtwürfel  
1. Zerlegungsebene  
2. Zerlegungsebene

■: Vollmaterial  
□: Material fehlt teilweise  
□: Material nicht vorhanden

Nullpunkt

## 14.2. HIDDEN SURFACE REMOVAL (HSR)

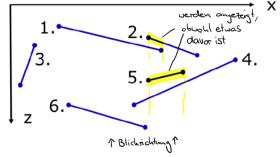
Das Entfernen von **nicht** sichtbaren Flächen.

### 14.2.1. Painter-Algorithmus

Ordnet die zu visualisierenden Flächen und zeichnet sie dann von **hinten nach vorne**. Die weiter entfernten Flächen werden immer wieder **übermalt**. Ablauf:

1. **Ordne** alle Polygone nach kleinsten z-Wert
2. Polygone mit überlappendem z-Ausdehnung ggf. **umordnen**
3. **Ausgabe** von hinten nach vorne.

Problem: Überlappende Flächen werden nicht korrekt angezeigt, müssen zerschnitten werden.



### 14.2.2. z-Buffer Algorithmus

Löst das Problem des Painter-Algorithmus. Benötigt **zwei 2-dimensionale Arrays** als Buffer, einmal den **z-Buffer**, der für jedes Pixel den z-Wert des in diesem Punkt dem Betrachter am nächsten liegenden Objekt enthält (**größerer z-Buffer → näher am Betrachter**), und einmal den **Frame Buffer**, welcher den Farbwert jedes Pixels speichert.

Für each Pixel  $(x, y)$  on this area:  
calculate Color c and Depth z  
if  $z > \text{depth}[x, y]$ :  
Add color c at  $(x, y)$  in the Frame Buffer and set  $\text{depth}[x, y] = z$

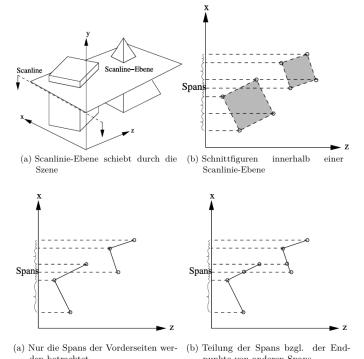
Wird vor diesem Algorithmus Back Face Culling durchgeführt, wird er **effizienter**. Das Ergebnis bleibt aber das Gleiche.

Problem: Nicht sehr effizient, weil er für jedes Pixel einer Scanline die z-Koordinate berechnen muss. Es kann auch sein, dass ein bereits gesetztes Pixel später überschrieben wird.

### 14.2.3. Span-Buffer Algorithmus

Löst das **Effizienz-Problem** des z-Buffer Algorithmus. Die Scanline durchläuft das Bild und zerfällt in Abschnitte (Spans). Pro Span ist genau ein Polygon zuständig.

Spans: Nur Vorderflächen nach x sortieren, ggf. zerschneiden und vereinigen.



Ist **effizienter**, weil **kein Pixel mehrfach eingefärbt** und die **Tiefe nicht überprüft** werden muss, dafür muss **zusätzliche Arbeit** für die Ermittlung der **Spans** aufgewendet werden.

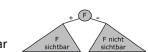
Problem: Gegenseitiges Durchdringen/Kreuzen wird nicht erkannt. Bei hoher Polygonanzahl weniger effizient als Z-Buffer.

### 14.2.4. Binary Space Partition Tree

Analysiert die **Logik** der Objekte untereinander mithilfe einer **Baumstruktur**: Ist unabhängig von der Kamera, die Root-Kante kann **beliebig** gewählt werden. Sehr teuer ( $O(n^2)$ ), nur für statische Szenen (z.B. Kamerafahrt in Cutscene)

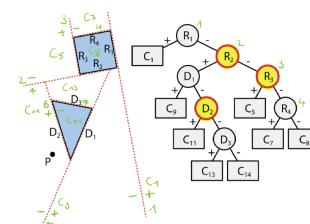
Jeder innere Knoten repräsentiert eine **Polygonfläche F**, welche die Szene aufteilt in

- **vorderer Teil:** F ist hier sichtbar
- **hinterer Teil:** F ist hier nicht sichtbar



### Ablauf:

1. Alle Kanten aller Objekte im Raum nummerieren ( $R_1-R_4, D_1-D_3$ )
2. Der ersten Kante des ersten Objektes entlang **eine Linie ziehen**. (Linie 1 entlang  $R_1$ )
3. Auf der äußeren Seite des Objektes ist der **sichtbare Teilbaum**, auf der anderen der **nicht sichtbare**. Wenn eine Seite gar keinen Inhalt hat, wird dies im Baum mit einem rechteckigen Feld markiert.
4. Mit der **zweiten Seite** des ersten Objektes **fahren** ( $R_2$ ). 5. So lange wiederholen, ob alle Kanten im Raum angegliedert sind und alle Blätter ein **eckiges Feld** sind.



## 15.2. REFLEXION

Das Reflexionsverhalten eines Körpers wird durch folgende Eigenschaften bestimmt:

- $k_a$ : ambienter Reflexionskoeffizient (wie stark reflektiert amb. Licht)
- $k_d$ : diffuser Reflexionskoeffizient (wie stark reflektiert diffuses Licht)
- $k_s$ : spekularer Reflexionskoeffizient (wie stark reflektiert Punktlicht)
- $\overline{C}_d$ : diffuse Objektfarbe (durch Spiegelung erzeugt)
- $\overline{C}_s$ : spekulare Objektfarbe (durch Spiegelung erzeugt)
- $\alpha_s$ : spekularer Exponent

### 15.2.1. Ambiente Reflexion

**Grundhelligkeit** eines Objekts.

$$\overline{C}_a = k_a \cdot \text{intensität des ambiente Lichts} + \overline{C}_d \cdot \text{diffuse Objektfarbe}$$

### 15.2.2. Diffuse Reflexion

Vom Objekt diffus reflektiertes Licht (**Lichtstrahlung**), d.h. überall **gleichmäßig sichtbar**.

$$\overline{C}_d = k_d \cdot \overline{I}_a \cdot \overline{C}_d \cdot \cos(\theta_d)$$

### 15.2.3. Spekulare Reflexion

Vom Objekt gespiegeltes Licht  $R_s$ , **nur in bestimmte Richtung sichtbar**.

$$\overline{C}_s = k_s \cdot \overline{I}_s \cdot \overline{C}_s \cdot \cos(\theta_s)^{\alpha_s}$$

### 15.2.4. Materialeigenschaften

- **Objekte sollten nicht mehr abstrahlen als empfangen:**  $0 \leq k_a, k_d, k_s \leq 1$  und  $k_a + k_d + k_s \leq 1$
- **Kontastraum:** Ambiente Teil ist am stärksten.  $k_a \gg k_d, k_s$
- **matt:** Diffuser Teil ist viel größer als Spekularteil.  $k_d \gg k_s$
- **spiegelnd:** Spekularteil ist größer als Diffuse Teil.  $k_s \gg k_d$

### 15.3. SCHATTIERUNG (SHADING)



### 15.3.1. Flat Shading

Pro Dreieck eine Farbe. Eckpunkte im WC beleuchten, Farbmittelwert  $C_i$  der Eckpunkte berechnen, diese Farbe für alle Pixel im Dreieck verwenden. Kanten zwischen Dreiecken sind mit dieser Variante sichtbar.

$$\overline{C}_i = \frac{\overline{C}_A + \overline{C}_B + \overline{C}_C}{3}$$

$P_A [200, 160, 10]$	$P_B [200, 30, 10]$	$P_D [200, 10, 10]$
$P_A [200, 70, 10]$	$P_C [200, 20, 10]$	
$P_D [200, 10, 10]$		

**Gouraud Shading**  
Interpolation der Farbwerte  $C_i$ , **Eckpunkte**  $\overline{C}_A, \overline{C}_B, \overline{C}_C$  werden fix gesetzt und der Rest wird mit Interpolation gefüllt.  $y = v$  von Scanline.

$$\begin{aligned} \overline{C} &= \overline{C}_A + \frac{C_2 - C_A}{y - Y_C} \cdot (Y_A - Y_C) + \overline{C}_B + \frac{C_3 - C_B}{y - Y_C} \cdot (Y_A - Y_C) \\ &\quad + \overline{C}_D + \frac{C_4 - C_D}{y - Y_B} \cdot (Y_A - Y_B) + \overline{C}_E + \frac{C_5 - C_E}{y - Y_B} \cdot (Y_A - Y_B) \\ &\quad + \dots + \overline{C}_F + \frac{C_n - C_F}{y - Y_n} \cdot (Y_A - Y_n) \end{aligned}$$

Doppelte Interpolation

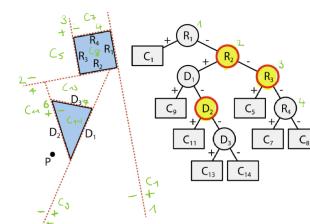
$$\overline{C} = \overline{C}_A + \sum_{i=1}^n \overline{C}_{di} + \sum_{i=1}^n \overline{C}_{si}$$

Interpolation für  $P_i$  aus  $P_A$  und  $P_B$ , gibt Farbwert an Stelle  $P_i$

## 14.3. TRANSPARENZ

### Ablauf:

1. Alle Kanten aller Objekte im Raum nummerieren ( $R_1-R_4, D_1-D_3$ )
2. Der ersten Kante des ersten Objektes entlang **eine Linie ziehen**. (Linie 1 entlang  $R_1$ )
3. Auf der äußeren Seite des Objektes ist der **sichtbare Teilbaum**, auf der anderen der **nicht sichtbare**. Wenn eine Seite gar keinen Inhalt hat, wird dies im Baum mit einem rechteckigen Feld markiert.
4. Mit der **zweiten Seite** des ersten Objektes **fahren** ( $R_2$ ). 5. So lange wiederholen, ob alle Kanten im Raum angegliedert sind und alle Blätter ein **eckiges Feld** sind.



## 15.3. PHONG SHADING

Aufwändige Version. Interpoliert pro Scanline die Anfangs- und die Endnormale. Pro Pixel wird ebenfalls die Normale interpoliert und daraus der Farbwert berechnet.

Im Vergleich zu Gouraud Shading ist bei Phong wirklich **jeder Pixel unterschiedlich**.

### 15.4. SCHATTEN

Von der **Lightquelle nicht sichtbare Pixel**. Um zu berechnen, ob ein Pixel im Schatten liegt oder nicht, kann ein beliebiger **Hidden-Surface-Removal-Algorithmus** verwendet werden.

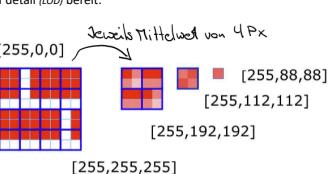
- **Phase 1:** Rendere Bild aus Position der Lichtquelle  $L$  in einem Schattentiefenpuffer  $S_{\text{depth}}$
- **Phase 2:** Rendere Bild aus Position des Betrachters mit modifiziertem Z-Buffer-Algorithmus: falls Pixel  $P(x, y, z)$  sichtbar ist, transformiere  $P$  in den Koordinatenraum von Phase 1 zu  $P'(x', y', z')$ .

Falls  $z' < s_{\text{depth}}[x', y']$ , dann ist  $P$  im Schatten von  $L$ .

Falls  $z' \geq s_{\text{depth}}[x', y']$ , dann ist  $P$  nicht im Schatten von  $L$ .

## 16.2. TEXTUR-ARTEFAKTE / MIP MAPPING

**Multum in parvo mapping:** Viel in wenig. Halte verschiedene Level bereit vorberechnete Textur-Auflösungen für verschiedene Level of detail (LOD) bereit.



$$x = (1-t) \cdot x_0 + t \cdot x_1, \text{ gleiche Formel auch für } y \text{ und } z$$

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$$

Einsetzen liefert quadratische Gleichung in  $t$  ggf. 0, 1 (tangential auf Kugel) oder 2 Schnittpunkte  $(x, y, z)$

## 19. PHYSIKALISCHE SIMULATIONEN

Starre Körper (*rigid body*): Verformt sich nicht, hat Masse, Trägheitsmoment, momentaner Bewegungszustand und Kollisionsgeometrie.

Kollisionen: Einzelne Körper dürfen sich **nicht durchdringen**. Beim Kontakt werden die **Kontaktkräfte berechnet** und auf die Körper **angewendet**.

Physikalische Simulation: aus dem momentanen **Bewegungszustand** aller Körper wird mittels Simulation der **nächste Zustand** abgeleitet. Es gibt zwei Arten der Simulation: **Lagrange** (mit und ohne reduzierte Koordinaten) und **Impulsbasiert**.

### 19.1. CONSTRAINTS

Über Constraints (*Joint*) können mehrere Körper miteinander verbunden werden. Constraints sind **physikalische Zwangsbedingungen** (Schlern einer Türe, Kugelager,...).

- **Point-To-Point Constraint:** Joint
- **Hinge-Constraint:** Scharnier
- **Slider-Constraint:** Führung

Kraftimpulse müssen so berechnet werden, dass Constraints gleichzeitig gültig bleiben.

## 20. SHADERS

Shaders sind **Programme** zur Beeinflussung der Darstellung von Objekten und Effekten in der **Grafikpipeline**. Werden in **HL** geschrieben. Sie nutzen **parallele Berechnungen** für

**Bump Mapping:** Durch Modifizierung der Normalenvektoren in Kombination mit Texturen können Unebenheiten simuliert werden (z.B. Backsteinmauer, bei Fugen weniger hoch). **Implementation:** Benötigt Height Mapping (1 Wert, Grauwertmatrix enthält Höhenänderungen) und Normal Mapping (3 Werte, Farbmatrixt enthält Normalenrichtungsänderung). **Achtung:** Suggestive Höhenänderungen sind von der Seite nicht sichtbar und haben keine Auswirkung auf Physik.

**Displacement Mapping:** Textur enthält Angaben zur **Veränderung der Geometrie**. Vorteile: Displacement Map + grobe Geometrie braucht weniger Platz als feine Geometrie. Eine Geometrie ist mit mehreren Displacements (*Skins*) nutzbar. **Funktionsweise:** Jede Fläche des Körpers wird anhand der Displacement Map in mehrere kleine Flächen unterteilt. Die Punkte können nur entlang der Flächennormalen verschoben werden. Sie werden also entweder aus der Fläche herausgehoben oder hineingeschoben. Die Stärke der Verschiebung ist in der Displacement Map als Grauwert hinterlegt. Eignet sich besonders gut, um Landschaften kompakt zu beschreiben.

**Netzvereinfachung:** Mit Displacement Map ist es auch möglich, die **Zahl der Polygone** für ein Körper deutlich zu **reduzieren** und mittels einer Map wiederherzustellen. Einfache Generierung von LOD-Modellen.

### 17. AUGMENTED REALITY

Unter erweiterten Realität / Augmented Reality versteht man die **computergestützte Erweiterung der Wahrnehmung**. Kann alle menschlichen Sinne ansprechen, häufig wird jedoch nur die **visuelle Darstellung von Informationen** verstanden, also die **Ergebnisse von Bildern und Videos mit computergenerierten Zusatzinformationen oder virtuellen Objekten** mittels **Einblendung/Überlagerung** (XBOX Kinect, Hud).

### 18. RAYTRACING

Strahlverfolgung eignet sich zur **Modellierung von spiegelnden Reflexionen und von Transparenz** mit Brechung. Für jeden Pixel wird ausgehend von der Kamera ein Strahl gelegt und der Schnittpunkt mit dem ersten Objekt bestimmt.

Bei spiegelnden Objekten wird der Reflexionsstrahl bestimmt und rekursiv weiter behandelt, bei Transparenz wird zusätzlich der gebrochene Strahl weiter behandelt.

**Effizienzsteigerung:** Berechnung ist sehr aufwändig, durch Z-Buffer, Begrenzungsvolumen und Tree-Erstellung kann die Effizienz etwas gesteigert werden.

### 20.1. ARTEN VON SHADERN

Neben **Fragment-Shader** und **Vertex-Shader** gibt es auch noch **Geometry-Shader** (Erzeugen zusätzliche Geometrie) und **Tessellation-Shader** (Verfeinern Oberflächen).

#### 20.1.1. Fragment-Shader / Pixel Shader

Berechnen Farbe und Eigenschaften **einzelner Pixel** für realistische Darstellung in 3D Rendering. **Normierte UV-Koordinaten** zwischen 0 und 1 ermöglichen flexible Texturzuordnung auf unterschiedlichen Auflösungen. Steuern Licht, Farbe und Transparenz. Fragment-Shader sind **zustandslos** (weil sonst Probleme mit Parallelisierung). Hohen Rechenleistung nötig.

**Anwendungsbereiche:** Visuelle Effekte, Komplexe Rendering-Techniken (HDR, Bloom, Lensflare), Verzerrungs- und Umwelteffekte (Wasser- & Hitzewellen).

#### 20.1.2. Vertex-Shader

Verändert die **Position von Punkten** im dreidimensionalen Raum anhand ihrer Koordinaten zur Simulation von Bewegung und Tiefe. Verwenden Funktionen wie **Skalierung**, **Rotation** und **Verschiebung**. Sind die **erste Stufe** der Rendering-Pipeline und beeinflussen direkt die Objektgeometrie. Sie verarbeiten **große Datenmengen parallel** und sind deshalb für **Echtzeit-Grafikanwendungen** ideal. Sind **zustandslos**.

**Anwendungsbereiche:** Animation von Wasserwellen oder bewegtem Gras, Terrain-Generierung (Höhen von Vertices mit Noise-Texturen)

**Besonderheiten:** Komplexe Bewegungen müssen in anderen Teilen der Grafik-Engine umgesetzt werden

### 20.1.3. Gegenüberstellung

Material	Compute Shader	Vertex Shader	Fragment Shader
Zweck	Allgemeine Berechnungen auf der GPU	Verteilung von 3D-Vertices	Rechnung des Farb- und Farbglanz-Farbe
Teil der Pipeline	Nein (unabhängig)	Ja (aller Schritte)	Ja (aller Schritte)
Benötigte Daten	Keine	Verteile Daten: Position, Normale, Farbe	Pixel-Fragment-Daten (UV, Position, Farbe)
Ausgabe	Belebige Daten (z.B. Buffer)	Transformierte Vertices: Positionen	Farbwerte (RGBA)
Parallelität	Extrem hoch (Workgroups, Threads)	Pro Vertex	Pro Pixel
Typische Anwendungen	Physik, KI, Belebungsrate, Noise-Generierung	Animationen, Terrain-Verformung	Schattierung, Spiegelungen, Post-Processing
Steiner-Spanne	GLSL, HLSL, Metal	GLSL, HLSL	GLSL, HLSL