Computergraphik I

Zusammenfassung wichtiger Elemente der Einführung in die Computergraphik

von Alexander Köster
Student der Universität Siegen, CG1 2019/20
Letzte Aktualisierung: 4. Juni 2020

Dieser Kurs der Computergraphik befasst sich mit den Grundlagen der generativen 3D-Computergraphik und damit verbundenen Theorien wie Farbtheorie und Vektorrechnung.

Einzig für Lernzwecke erstellt.

Nicht geeignet als Klausurhilfe, sollte eine solche erlaubt sein.
Das Erstellen einer eigenen Klausurhilfe führt zu einem besonders guten Lerneffekt.
Dieses Dokument sollte nur als Orientierung oder Vergleich dienen.
Jeder sollte seine Klausurhilfe individuell auf seinen Lernstand und seine eigenen Probleme anpassen, gut bekannte Dinge auslassen und schlecht merkbare Dinge hinzufügen.
Dieses Dokument beinhaltet viel mehr, als für eine tatsächliche Klausur durchschnittlich benötigt wird.
Für einen guten Ansatz, welche Inhalte man braucht, sollte sich an der Probeklausur orientiert werden.

1 Grundlagen

1.1 Lineare Algebra

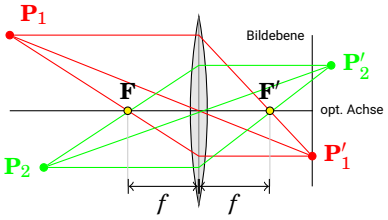
- Vektoren $\vec{v_1},\dots,\vec{v_n} \in K^n$ **linear unabhängig** falls $\sum_{i=1}^k a_i\vec{v_i} = \sum_{i=1}^k b_i\vec{v_i} \Leftrightarrow a_i = b_i$, d. h. falls $\sum_{i=1}^k a_i\vec{v_i} = 0 \Leftrightarrow a_i = 0$.
- **Inneres Produkt**: $(\vec{u}\cdot\vec{v}) = \sum_{i=1}^n u_i v_i$
- **Norm**: $\|\vec{u}\| = \sqrt{(\vec{u}\cdot\vec{u})}$
- **Normierter Vektor**: $\hat{v} := \frac{\vec{v}}{\|\vec{v}\|}$
- **Einschl. Winkel**: $\sphericalangle(\vec{u},\vec{v}) = \arccos\left(\frac{(\vec{u}\cdot\vec{v})}{\|\vec{u}\|\|\vec{v}\|}\right)$
- **Kreuzprodukt** (nur 3D): $\vec{u}\times\vec{v} = \begin{pmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{pmatrix}$
  - ▷ bilinear
  - ▷ orthogonal: $\vec{u}\times\vec{v} \perp \vec{u},\vec{v}$
  - ▷ *nicht* kommutativ, aber $\vec{u}\times\vec{v} = -\vec{v}\times\vec{u}$
  - ▷ i. A. *nicht* assoziativ, Ausnahme: $(\vec{u}\times\vec{v})\times\vec{w} = \vec{u}\times(\vec{v}\times\vec{w}) \Leftrightarrow \vec{u} = \vec{w}$
- **Parallelogrammfläche** aufgespannt durch $\vec{u},\vec{v}$: $A_{\square} = \underbrace{\sin\alpha\,\|\vec{u}\|}_{h}\|\vec{v}\| = \|\vec{u}\times\vec{v}\|$
- **Projektion** von $\vec{v}$ entlang $\vec{n}$: $\vec{v_n} = (\vec{v}\cdot\hat{n})\hat{n}$
- **Spiegelung** von $\vec{v}$ an $\vec{n}$: $\vec{v'} = 2(\vec{v}\cdot\hat{n}) - \vec{v}$
- Ebenengleichungen
  - ▷ **Parameterform**: $E : \vec{x} = \vec{p} + \lambda_1\vec{r_1} + \lambda_2\vec{r_2}$
  - ▷ **Normalenform**: $E : \left(\vec{n}\cdot(\vec{x}-\vec{p})\right) = 0$ mit $\vec{n} = \vec{r_1}\times\vec{r_2}$
  - ▷ **Hesse-NF** (in 3D): $E : (\vec{x}\cdot\hat{n}) = d$ mit $d = (\vec{p}\cdot\hat{n})$
- **Basiswechselmatrix** von B nach B' in K^n: ${}^{B'}\mathrm{id}_{K^n}{}^{B} = \left({}^{B'}B_1 \mid {}^{B'}B_2 \mid \dots \mid {}^{B'}B_n \right)$
  - ▷ ${}^{B}\mathrm{id}_{K^n}{}^{B'} = \left({}^{B'}\mathrm{id}_{K^n}{}^{B}\right)^{-1}$
  - ▷ Sonderfall: Für Standardbasis E ist ${}^{E}\mathrm{id}_{K^n}{}^{B} = \left(B_1 \mid B_2 \mid \dots \mid B_n \right)$

1.2 Farbrepräsentation

- Sichtbares Licht: ca. 380 bis 780nm
- **Farbvalenz**: Menge von Farbreizen, die ein- und dieselbe Farbempfindung erzeugen
- **Metamerie**: Versch. Farbreizfunktionen (Verteilung der Wellenlängen einer Lichtquelle) können dieselbe Farbempfindung erzeugen.
- **Tri-Stimulus**: Farbraum ist 3-dimensionaler Vektorraum, Basis sind Primärvalenzen.
- Verschiedene Farbmodelle
  - ▷ **RGB**: Additives Farbmodell, Spektralvalenzen: Rot 700nm, Grün 546nm, Blau 436nm. *Achtung:* Manche Spektralvalenzen erfordern negative Gewichte!
  - ▷ **CIE-XYZ**: $\mathbf{Y}$ = Helligkeit, $\mathbf{X},\mathbf{Z}$ so, dass alle Spektralvalenzen mit pos. Gewichten erzeugbar.
    $$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 2.365 & -0.875 & -0.468 \\ -0.503 & 1.392 & 0.086 \\ 0.005 & -0.014 & 1.009 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$
  - ▷ **CMY**: Subtraktives Farbmodell, Primärvalenzen sind zu RGB komplementäre Val.: $\mathbf{C} = \mathbf{G}+\mathbf{B}$, $\mathbf{M} = \mathbf{R}+\mathbf{B}$, $\mathbf{Y} = \mathbf{R}+\mathbf{G}$,
    $$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$
  - ▷ **HLS**: Hue, Luminance, Saturation; $H \in [0°, 360°]$, $S, L \in [0,1]$; $H = 0° \Rightarrow$ rot, $H = 120° \Rightarrow$ grün, $S = 0 \Rightarrow$ Graustufe, $S = 1 \Rightarrow$ reine Farbe, $L = 0 \Rightarrow$ schwarz, $L = 1 \Rightarrow$ weiß
    Umrechnung zu RGB nicht linear.
  - ▷ Weitere: HSV (Hue, Saturation, Value), HSI (Hue, Saturation, Intensity), ähnlich zu HLS.
- **Gamut**: Bereich der erzeugbaren Farbvalenzen eines Gerätes

2 Einführung in die Rastergraphik

- **Lichtbrechung**:
  - ▷ Brechungsindex $n = \frac{c_{\text{Vakuum}}}{v_{\text{Licht im Material}}}$
  - ▷ Brechungsgesetz: $\sin(\Theta_1)n_1 = \sin(\Theta_2)n_2$ mit Θ_1 = Einfallswinkel zur Normale, Θ_2 = Brechungswinkel zur Normale.
  - ▷ **Bsp.:** $n_{\text{Luft}} \approx 1$, $n_{\text{Wasser}} \approx 1.333$, $n_{\text{Flintglas}} \approx 1.613$.
  - ▷ Einfaches Modell einer dünnen **Linse**:



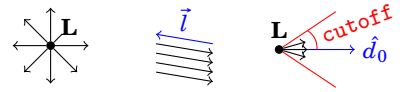
  Brennpunkt $\mathbf{F},\mathbf{F}'$: Schnittpunkt bei parallelen Strahlen auf Objekt- bzw. Bildseite
  Unschärfe, falls $\mathbf{P}'$ nicht auf Bildebene
- **Graphik-Pipeline**:
  - ▷ **Vertex Shader** (VS):
    - Vertex-Transformationen, per-Vertex-Beleuchtung
    - Normalen-Transformation und Normalisierung der Normalen
    - Generierung von Textur-Koordinaten
    - *Stream Input:* (VAO) Position, Farbe, Normale, Textur-Koordinaten
    - *Stream Output:* Farbe, Position, Textur-Koordinaten
  - ▷ **Fragment Shader** (FS):
    - Verarbeitung der vom Rasterisierer interpolierten per-Vertex-Daten auf Fragment-Ebene
    - Auslesen von Daten (Farbe etc.) aus Texturen
    - Textur-Funktion, d. h. Ermittlung der finalen Farbe aus Textur und (interpolierter) Fragment-Intensität
    - per-Fragment-Beleuchtung
    - *Stream Input:* Farbe, Position (nur z-Tiefe), Textur-Koordinaten
    - *Stream Output:* Farbe, Tiefe
  - ▷ **Framebuffer**: Spezieller Speicher, der Informationen pro Pixel verwaltet:
    - Farbe (Color Buffer)
    - Tiefeninfos für Verdeckungsrechnung (z-Buffer/Depth Buffer)
    - Weitere Buffer für Effekte wie Spiegeln, Schatten oder Motion Blur

    Bitplanes: Je nach Anwendung unterschiedliche Anzahl Bits/Pixel (**Bsp.:** Color (8, 16, 24, 32), Tiefe (24, 32)).
    - **Double Buffering**: Front- und Backbuffer für Verhinderung von Flickern bei Verwendung des gleichen Buffers für Anzeige & Bildgenerierung
- **Vertex Array Objects** (VAO): Speichert den Status der gebundenen **Vertex Buffer Objects** (VBOs), welche die Vertex-Stream-Daten speichern (**Bsp.:** VBO0: Vertex-Array (`vec3`), VBO1: Color-Array (`vec3`), VBO2: Normal-Array (`vec3`), VBO3: TexCoord-Array (`vec2`)).
- **Swizzling** in OpenGL-GSM/GLSL: Für Vektor `vec4 a` werden die Komponenten bezeichnet als `x,y,z,w` oder `r,g,b,a` oder `s,t,p,q`.
  Anzahl und Reihenfolge beliebig (**Bsp.:** `a.x`, `a.zy`, `a.xxxx`), Mischen verboten (**Bsp.:** ~~`a.xr`~~).
- Visibilitätstest: ***z*-Buffer-Algorithmus**:

```
setColor(backgroundColor);
setZBuffer(1.0);
foreach (polygon P) {
    foreach (pixel (x,y) in P) {
        z = computeDepth(P, x, y);
        if (zBuffer(x,y) > z) {
            setColor(x, y,
                computeColor(P, x, y));
            zBuffer(x,y) = z;
        }
    }
}
```

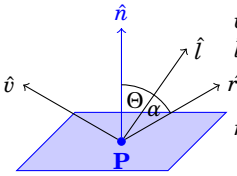
3 Beleuchtungsberechnung

- **Exaktes Modell** (zu kompliziert): Illuminationsfunktion $L(\mathbf{P},\hat{d},\lambda)$, die die Lichtintensität für Wellenlänge λ im Punkt $\mathbf{P}$ der Lichtquelle in Richtung $\hat{d}$ beschreibt
- Einfache Lichtquellen:



Punktlicht Richtungslicht Spotlicht

  - ▷ Spotlicht-Abschwächung: Lichtintensität
    $$L\left(\vec{d}\right) = \begin{cases} \left(\hat{d_0}\cdot\hat{d}\right)^n L_0 & \arccos\left(\hat{d_0}\cdot\hat{d}\right) \\ & < \texttt{cutoff} \\ 0 & \text{sonst} \end{cases}$$
- **Phong-Modell**: Die Intensität am Oberflächenpunkt $\mathbf{P}$ ist: $I_{\text{gesamt}} = I_a + I_d + I_s$



$\hat{v}$: Viewing-Vektor
$\hat{l}$: Richtung Lichtquelle
$\hat{r_v} = 2(\hat{n}\cdot\hat{v})\hat{n} - \hat{v}$ (refl. Viewing-Vekt.)

  - ▷ $I_a = k_a * L_a$: ambienter Anteil
    - k_a: ambienter Reflexionsfaktor (Materialkonstante)
    - L_a: ambiente Lichtintensität

    $I_a, k_a \in [0,1]^3$, $*$ komponentenw. Mult.
    Hängt weder von der Beobachter-, noch von der Lichtquellenposition ab.
  - ▷ I_d: diffuser Anteil
    $I_d = \max\{\cos\Theta, 0\} \cdot k_d * L_d$
    $= \max\{(\hat{n}\cdot\hat{l}), 0\} \cdot k_d * L_d$
    - k_d: diffuser Reflexionsfaktor (Materialkonstante)
    - L_d: diffuse Lichtintensität

    Hängt nur von der Lichtquellenpos. ab.
  - ▷ I_s: spiegelnder Anteil
    $I_s = \max\{\cos\alpha, 0\}^n k_s * L_s$
    $= \max\{(\vec{l}\cdot\vec{r_v}), 0\}^n k_s * L_s$
    - k_s: spiegelnder Reflexionsfaktor (Materialkonstante)
    - L_s: spiegelnde Lichtintensität
    - n: Reflexionsexponent

    Hängt von Lichtquellen- und Beobachterposition ab.

  ↑ Klausuraufgabe (händische Berechnung)!
- Ermittlung von Normalenvektoren:
  - ▷ Ebenes, konvexes Polygon: Kreuzprodukt zweier Kanten
  - ▷ Polyederecke: Mittelung der umliegenden Normalen
  - ▷ Parametrische Fläche $F : \mathbb{R}^2 \to \mathbb{R}^3$, Normale im Punkt $\mathbf{P} = F(u_0, v_0)$:
    $\vec{n} = \frac{\partial F}{\partial u}(u_0,v_0) \times \frac{\partial F}{\partial v}(u_0,v_0)$
    (komponentenweise partielle Ableitung)

4 Transformationen, Hierarchien

- **Affine Kombination**: Für $\mathbf{P_1},\dots,\mathbf{P_k} \in \mathscr{A}$ und Skalare $s_1+\dots+s_k=1$ ist $\sum_{i=1}^k s_i\mathbf{P_i} \in \mathscr{A}$. Ansonsten ist $\mathbf{P}+\mathbf{Q}$ für Punkte i. A. *nicht* def.!
- **Konvexe Hülle** der Punkte $\mathbf{P_1},\dots,\mathbf{P_k} \in \mathscr{A}$: Die kleinste, sie umfassende konvexe Menge, d. h. $\{\sum_{i=1}^k s_i\mathbf{P_i} \mid \sum_{i=1}^k s_i = 1,\ s_i \geq 0\ \forall i \in \underline{k}\}$
- **Affine Transformation**: Abb. $T : \mathscr{A} \to \mathscr{A}$, wenn sie affine Kombinationen invariant lässt, d. h. $\forall \mathbf{P_1},\dots,\mathbf{P_k} \in \mathscr{A}$ und s_i wie oben:
  $T(s_1\mathbf{P_1}+\dots+s_k\mathbf{P_k}) = s_1T(\mathbf{P_1})+\dots+s_kT(\mathbf{P_k})$
- T affin $\Leftrightarrow \exists M \in \mathbb{R}^{n\times n}, \vec{t} \in \mathscr{A} : T(\mathbf{P}) = M\mathbf{P}+\vec{t}$
- **Homogene Koordinaten**:

$$\mathbf{P} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \rightsquigarrow \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}, \quad \vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \rightsquigarrow \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix},$$

$$M\mathbf{P}+\vec{t} \rightsquigarrow \begin{bmatrix} & M & & t_x \\ & & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

  Ist die w-Koordinate weder 0 noch 1, so muss der Vektor auf 1 normiert werden.
- **Translation** um $\vec{t}$:

$$\mathbf{P} \mapsto \mathbf{P} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \rightsquigarrow T_{\vec{t}} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  $(T_{\vec{t}})^{-1} = T_{-\vec{t}}$
- **Skalierung**: $\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \mapsto \begin{pmatrix} s_x \cdot p_x \\ s_y \cdot p_y \\ s_z \cdot p_z \end{pmatrix}$

$$\rightsquigarrow S_{s_x,s_y,s_z} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  $(S_{s_x,s_y,s_z})^{-1} = S_{\frac{1}{s_x},\frac{1}{s_y},\frac{1}{s_z}}$
- **Rotation** um z-Achse mit Winkel ϕ:

$$R_{z,\phi} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  $(R_{z,\phi})^{-1} = R_{z,-\phi}$
- Kumulierte Transformationen:
  - ▷ Transformation bzgl. eines festen Weltkoordinatensystem (WC) – direkte Anwendung der aff. Transf. auf Punkte.
    Nachteil: immer in Relation zum WC, daher: Translation immer zum Schluss.
  - ▷ Transformation bzgl. des lokalen Modell- oder Master-Koordinatensystems (MC).
    Nachteil: Längeneinheit 1 skaliert mit, daher: Skalierung immer zum Schluss.

  Letztere für Modellierung wesentlich intuitiver.
- **Transf. MC→WC** des Objektpunktes $\mathbf{P}^{\text{MC}}$:
  $\mathbf{P}^{\text{WC}} = (T(R(S\,\mathbf{P}^{\text{MC}}))) = (TRS)\mathbf{P}^{\text{MC}}$
- Werden Punkte mit der Matrix T transformiert, müssen Normalen mittels $(T^{-1})^\top = (T^\top)^{-1}$ transformiert werden, damit Orthogonalitätsbeziehung erhalten bleibt.
- Beobachterkoordinatensystem: rechtshändiges, orth. Koord.sys. $\{\mathbf{V},\vec{v_x},\vec{v_y},\vec{v_z}\}$ in WC
  - ▷ $\vec{v_x}$: Beobachter-Horizont
  - ▷ $\vec{v_y}$: Vertikale, d. h. $\vec{v_x}$-$\vec{v_y}$-Ebene ist Projektionsebene für den Beobachter
  - ▷ $\vec{v_z}$: Blickrichtung (entlang $-\vec{v_z}$)

  Viewing-Transformation (Wechsel WC→VC):

$$T_{\text{V}} = \left[\begin{array}{ccc|c} & A^\top & & -A^\top\mathbf{V} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

  mit $A = (\ \vec{v_x} \mid \vec{v_y} \mid \vec{v_z}\ )$,

$$T_{\text{V}}^{-1} = \left[\begin{array}{ccc|c} & A & & \mathbf{V} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

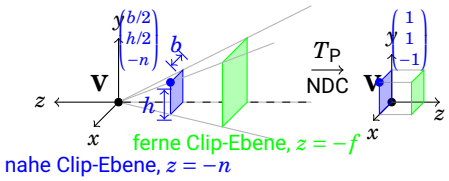
- **Sichtbereich**: Teilmenge des 3D-Raumes in VC, die im 2D-Bild dargestellt wird. In **Normalized Device Coordinates** (**NDC**) $[-1,1]^3$, linkshändiges Koord.-sys., d. h. $+z \propto$ „Entfernung zum Beobachter".
  2D-Bild entsteht durch anschließende Parallelprojektion in NDC entlang z-Achse.
  Projektionsart: Wie wird die 3D-Geometrie später auf das 2D-Bild projiziert?
  - ▷ **Orthographisch**: Sichtbereich ist ein Quader (Parallelprojektion).

$$T_{\text{O}} = \begin{bmatrix} \frac{2}{\Delta x} & 0 & 0 & -2\frac{x_{\min}}{\Delta x}-1 \\ 0 & \frac{2}{\Delta y} & 0 & -2\frac{y_{\min}}{\Delta y}-1 \\ 0 & 0 & \frac{2}{\Delta z} & -2\frac{z_{\min}}{\Delta z}+1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

    d. h. $T_{\text{O}}\mathbf{P} = \begin{pmatrix} \frac{2(p_x-x_{\min})}{\Delta x}-1 \\ \frac{2(p_y-y_{\min})}{\Delta y}-1 \\ \frac{2(p_z-z_{\min})}{\Delta z}+1 \end{pmatrix} \in [-1,1]^3$
  - ▷ **Perspektivisch**: Sichtbereich ist ein Pyramidenstumpf (Viewing Frustrum) mit Spitze im Beobachterpunkt $\mathbf{V}$.

$$T_{\text{P}} = \begin{bmatrix} \frac{2}{b} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n(n-f)} & \frac{2f}{n-f} \\ 0 & 0 & -\frac{1}{n} & 0 \end{bmatrix}$$

    Keine affine Transf.! D. h. nicht winkelerhaltend ⇒ nicht geeignet für Raytracing oder Beleuchtungsberechnung.



    - – Geraden werden auf Geraden abgebildet.
    - – Ordnung von Punkten auf Geraden bleibt erhalten.
    - – Geraden durch Beobachterpkt. werden zu z-achsenparallelen Geraden.
    - – Tiefenabstände werden verzerrt; Abstände nah am Beobachter werden größer, ferne werden verkleinert.
- Gerichteter azykl. Graph (**DAG**)/Szenengraph
  - ▷ Wurzel: Viewing-Transformation V
  - ▷ Knoten: Transformationen und Geometrien
  - ▷ Blätter: Geometriebeschreibungen der Primitive
  - ▷ Pfad von Wurzel zu Blatt (Geometrie) entspricht einem gezeichneten Primitiv.
- **Matrix-Stack**: Speicherstruktur zur Umsetzung eines DAG
  - ▷ CTM: oberstes Element des Stack
  - ▷ Multiplikation von rechts = Transformation in lokalen Koordinaten

5 Bildspeicherung

- **Vektorgrafiken**:
  - ▷ Bildschärfe unabhängig der Skalierung
  - ▷ weniger speicherintensiv bei einfachen Bildern
  - ▷ Speicherung von Details aufwendig bis unmöglich (ungeeignet für Fotografien)
  - ▷ **Bsp.:** SVG-Format, 3D-CAD (DXF, IGES, ...), Meta-Formate (PS, CGM, WMF, ...)
- **Rastergrafiken**:
  - ▷ „punktgenaue" Darstellung bei Details
  - ▷ nur bedingt skalierbar, speicherintensiv
  - ▷ **Bsp.:** JPEG, TIFF, GIF, PNG, PNM (PBM, PGM, PPM), XPM, BMP, ...
  - ▷ Komprimierung
    - – Verlustfreie Komprimierung (d. h. ursprüngliche Daten vollständig wiederherstellbar)
      - · Runlength-Encoding
        Bsp.: $HAAAALLO \to H\backslash 4ALLO$
      - · Huffman-Codierung
      - · Difference Coding
        (z. B. in Videos mit Keyframes)
    - – Verlustbehaftete Komprimierung
      - · **Quantisierung**: Diskretisierung der Farbräume auf z.B. *HighColor* (16 Bit, 65536 Farben), *TrueColor* (24 Bit (bzw. 32 Bit mit Transparenz), 16777216 Farben)
      - · **Frequenzraumverfahren**: Transf. des Bildes in anderen Darstellungsraum durch z. B. Fourier-Transformation oder Diskrete Cosinustransformation (JPEG): Arbeite auf 8×8-Pixelblöcken, speichere Frequenzwerte, Sortierung in aufsteigender Folge, Quantisierung der Freq., Difference Coding
- **Meta-Formate**: Kombination der beiden

6 Texturen

- Textur ist diskr. Bild aus $m \times n$ Bildpkt. (**Texel**)
- auflösungsunabh. **Texturkoord.** $s,t \in [0,1]$
- **Lineares Mapping**: Texturkoordinaten entsprechen Punkt-Ebenen-Abstand:
  Definiere Ebenen (Koord.-f.) anhand $\vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$:
  $E_s : a_sx+b_sy+c_sz+d_s = 0$
  $E_t : a_tx+b_ty+c_t+d_t = 0$
  Texturkoordinaten an Punkt $\mathbf{P} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$:
  $s = a_sp_x+b_sp_y+c_sp_z+d_s$
  $t = a_tp_x+b_tp_y+c_sp_z+d_t$
  (Einsetzen in Koordinatenform für Abstand)
  Nur $(a_s,b_s,c_s,d_s)^\top$, $(a_t,b_t,c_t,d_t)^\top$ als `uniform`-Parameter speichern reicht zur Berechnung von s,t als Skalarprodukt.
- **Cube Maps**: Umgebung in alle 6 Raumrichtungen, Sampling anhand einer Richtung $\vec{r_v} = (x,y,z)^\top$ und deren maximaler Komponente $m = \max\{\pm x,\pm y,\pm z\}$:
  - ▷ $-x = m$: „left"; $(s',t') = \left(\frac{-z}{x},\frac{y}{x}\right)$
  - ▷ $x = m$: „right"; $(s',t') = \left(\frac{z}{x},\frac{y}{x}\right)$
  - ▷ $-y = m$: „down"; $(s',t') = \left(\frac{x}{y},\frac{-z}{y}\right)$
  - ▷ $y = m$: „up"; $(s',t') = \left(\frac{x}{y},\frac{z}{y}\right)$
  - ▷ $-z = m$: „front"; $(s',t') = \left(\frac{x}{z},\frac{y}{z}\right)$
  - ▷ $z = m$: „back"; $(s',t') = \left(\frac{-x}{z},\frac{y}{z}\right)$

  Anpassung von $[-1,1]^2$ auf $[0,1]^2$:
  $(s,t) = \frac{1}{2}(s',t') + \left(\frac{1}{2},\frac{1}{2}\right)$
  - ▷ Umgebung erscheint als unendlich weit entfernt (keine Abhängigkeit von Pos.)
  - ▷ Reflexion durch reflektierten View-Vektor zum Sampling der Cube Map
- Textur-Interpolation (**Magnification**):
  - ▷ **Nearest Neighbor**: Wähle einfach Intensität des nächstliegenden Texel
    - – Pixelstruktur ggf. sichtbar
  - ▷ **Bilineare Interpolation**: aff. Kombination der vier umliegenden Texel-Intensitäten $\mathbf{B}_{i,j},\mathbf{B}_{i+1,j},\mathbf{B}_{i,j+1},\mathbf{B}_{i+1,j+1}$:
    $\alpha,\beta \in [0,1]$ sind x- bzw. y-Abst. zu $\mathbf{B}_{i,j}$
    $I = (1-\alpha)(1-\beta)\mathbf{B}_{i,j} + \alpha(1-\beta)\mathbf{B}_{i+1,j}$
    $\quad +(1-\alpha)\beta\mathbf{B}_{i,j+1} + \alpha\beta\mathbf{B}_{i+1,j+1}$
    - – rechenaufwändiger
    - – Unschärfe statt Pixelstruktur

- **Ungültige Texturkoordinaten** ($s,t \notin [0,1]^2$):
  - ▷ Periodisch (**repeat**):
    $(s,t) \mapsto (s - \lfloor s \rfloor,\ t - \lfloor t \rfloor)$
  - ▷ Abschneiden (**clamp**):
    $$s \mapsto \begin{cases} s & s \in [0,1] \\ 0 & s < 0 \\ 1 & s > 1 \end{cases}$$
  - ▷ Kombination der Anpassungsarten möglich (**Bsp.:** repeat in s, clamp in t).
- **Minification**: mehrere Texel werden auf ein Bildpixel abgebildet ⇒ Aliasing (**Moire-Effekt**)
  - ▷ **Mip-Mapping**: Verwendung von vorgefertigten Verkleinerungen der Textur.
    Bsp.: Durchschnittsberechnung, halbiere Texturauflösung in jeder Dimension (d. h. $\frac{1}{4}$ der Originalgröße).
    Kann automatisch berechnet werden.
  - ▷ Auswahl des MipMap-Levels, sodass Pixelgröße ≈ Texelgröße.
    Umschalten kann sichtbar sein! Daher: lineare Interpolation zwischen Mip-Map-Leveln.
- **Textur-Funktionen**: Kombination der Textur- und Beleuchtungs-Intensität.
  - ▷ **Replace**: Verwende nur den Texturwert.
  - ▷ **Modulate**: Verwende als Objektmaterial weiß, dann beinhaltet die Beleuchtung nur Helligkeitsschwankungen durch Geometrie. Dann lege die Texturintensität darauf: $I(x,y) = I_B(x,y) * I_T(x,y)$ (komponentenweise Multiplikation)

7 Algorithmen der Rastergraphik

- **Erinnerung:** Window-Koord. in NDC: $[-1,1]^2$
- Window Edge Coordinates (**WEC**):
  Berschreibung eines Punktes $\mathbf{P} = (p_x, p_y)^\top$ anhand seiner Lage zu den Fensterkanten:
  - ▷ $\mathbf{P}$ rechts von linker Fensterkante (innen) $\Leftrightarrow p_x \geq -1$
  - ▷ $\mathbf{P}$ links von rechter Fensterkante (innen) $\Leftrightarrow p_x \leq 1$
  - ▷ $\mathbf{P}$ oberhalb von unterer Fensterkante (innen) $\Leftrightarrow p_y \geq -1$
  - ▷ $\mathbf{P}$ unterhalb von oberer Fensterkante (innen) $\Leftrightarrow p_y \leq 1$

  $$\begin{aligned}\mathrm{WEC}(\mathbf{P}) &= (\mathrm{wec}_L(\mathbf{P}), \mathrm{wec}_R(\mathbf{P}), \mathrm{wec}_B(\mathbf{P}), \mathrm{wec}_T(\mathbf{P})) \\ &= (1+p_x,\ 1-p_x,\ 1+p_y,\ 1-p_y)\end{aligned}$$
  - ▷ $\mathrm{sign}(\mathrm{wec}_i(\mathbf{P}))$ gibt Innen- (+) bzw. Außenlage (−) bzgl. Fensterkante i an
  - ▷ $|\mathrm{wec}_i(\mathbf{P})|$ ist Abstand zur Fensterkante.
- **outcode-Schema**: Beschr. von $\mathbf{P} = (p_x, p_y)^\top$ bzgl. *aller* Fensterkanten:
  $\mathrm{outcode}(\mathbf{P}) = (b_L, b_R, b_B, b_T)$,
  wobei $b_i = \begin{cases} 1 & \mathrm{wec}_i(\mathbf{P}) < 0 \text{ (außen)} \\ 0 & \mathrm{wec}_i(\mathbf{P}) \geq 0 \text{ (innen)} \end{cases}$
  - ▷ **Trivial Accept**:
    $\overline{\mathbf{P_1P_2}}$ liegt innerhalb des Fensters
    $\Leftrightarrow \mathrm{outcode}(\mathbf{P_1}) \vee \mathrm{outcode}(\mathbf{P_2}) = 0$
  - ▷ **Trivial reject**:
    $\overline{\mathbf{P_1P_2}}$ liegt außerhalb *einer* Fensterkante
    $\Leftrightarrow \mathrm{outcode}(\mathbf{P_1}) \wedge \mathrm{outcode}(\mathbf{P_2}) \neq 0$

  Ansonsten: **Clipping** gegen die Fensterkanten mit $b_i(\mathbf{P_1}) \neq b_i(\mathbf{P_2})$ durchführen.
- **α-Clipping**: Ermittlung des Schnittpunktes der Strecke $\overline{\mathbf{P_1P_2}}$ mit einer erweiterten Fensterkante. $\mathbf{P}(\alpha_i) = \mathbf{P_1} + \alpha_i(\mathbf{P_2} - \mathbf{P_1})$
  $$\alpha_i = \frac{\mathrm{wec}_i(\mathbf{P_1})}{\mathrm{wec}_i(\mathbf{P_1}) - \mathrm{wec}_i(\mathbf{P_2})}$$
- **Liang-Barsky-Algorithmus**: Berechnet den sichtbaren Anteil $\overline{\mathbf{Q_1Q_2}}$ zu $\overline{\mathbf{P_1P_2}}$ (oder `false` falls Strecke außerhalb) mit outcode/WEC: Für je L,R,B,T: TR? $\xrightarrow{\text{nein}}$ TA? $\xrightarrow{\text{nein}}$ α-Clipping
- Polygon-Clipping: **Sutherland-Hodgman-Alg.**:
  INPUT: Sequenz der Eckpunkte des Subjekt-Polygons (subjectPolygon), Kanten des Clip-Polygon (clipPolygon), hier: $\{L,R,B,T\}$.
  ALGORITHMUS (Pseudocode):

```
List outputList ← subjectPolygon
FOR Edge clipEdge in {L,R,B,T} DO
  List inputList ← outputList
  outputList.clear()
  Point S ← inputList.last()
  FOR Point E in inputList DO
    IF E inside clipEdge THEN
      IF S not inside clipEdge THEN
        outputList.add(ComputeIntersection(
          S,E,clipEdge))
      ENDIF
      outputList.add(E)
    ELSEIF S inside clipEdge THEN
      outputList.add(ComputeIntersection(
        S,E,clipEdge))
    ENDIF
    S ← E
  ENDFOR
ENDFOR
```

- Clipping gegen beliebiges *konvexes* Polygon $\mathcal{Q} = (\mathbf{Q_1}, \ldots, \mathbf{Q_m})$:
  - ▷ Orientierter Abstand von $\mathbf{P}$ zur Kante $\mathbf{Q_iQ_{i+1}}$:
    $$(\vec{n} \cdot (\mathbf{P} - \mathbf{Q_i})) \begin{cases} = 0 & \mathbf{P} \text{ liegt auf Gerade} \\ > 0 & \mathbf{P} \text{ im Äußeren} \\ < 0 & \mathbf{P} \text{ im Inneren} \end{cases}$$
  - ▷ verallgemeinerte WEC:
    $\mathrm{wec}_i(\mathbf{P}) = -(\vec{n} \cdot (\mathbf{P} - \mathbf{Q_i}))$
- Clipping in 3D:
  - ▷ Clipping gegen 6 Seiten*flächen* (statt 4 Geraden oder Polygon)
  - ▷ Nach perspektiv. Tranf. ist i. A. $w \neq 1$.

  Drücke WEC in homogenen Koordinaten ohne Division durch w aus:
  $$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \begin{cases} \text{links innerh.} \\ \text{rechts innerh.} \\ \text{unten innerh.} \\ \text{oben innerh.} \\ \text{vorne innerh.} \\ \text{hinten innerh.} \end{cases} \overset{w>0}{\Leftrightarrow} \begin{cases} w+x \geq 0 \\ w-x \geq 0 \\ w+y \geq 0 \\ w-y \geq 0 \\ w+z \geq 0 \\ w-z \geq 0 \end{cases}$$
  aus Beobachtersicht.
  $\mathrm{WEC}(\mathbf{P}) := (w+x,\ w-x,\ w+y,\ w-y,\ w+z,\ w-z)$
- **Window-to-Viewport-Mapping**: Transformation NDC→RC (Rasterkoordinaten).
  - ▷ Längenverhältnisse müssen erhalten bleiben.
  - ▷ $\mathbf{P} = \begin{pmatrix} p_x \\ p_y \end{pmatrix} \in \left[x^{\mathrm{W}}_{\min}, x^{\mathrm{W}}_{\max}\right] \times \left[y^{\mathrm{W}}_{\min}, y^{\mathrm{W}}_{\max}\right]$
    $T_{\mathrm{W}\to\mathrm{V}}(\mathbf{P}) = \begin{pmatrix} q_x \\ q_y \end{pmatrix} \in \left[x^{\mathrm{V}}_{\min}, x^{\mathrm{V}}_{\max}\right] \times \left[y^{\mathrm{V}}_{\min}, y^{\mathrm{V}}_{\max}\right]$
    mit $q_x = x^{\mathrm{V}}_{\min} + \frac{x^{\mathrm{V}}_{\max} - x^{\mathrm{V}}_{\min}}{x^{\mathrm{W}}_{\max} - x^{\mathrm{W}}_{\min}} \left(p_x - x^{\mathrm{W}}_{\min}\right)$
    und $q_y = y^{\mathrm{V}}_{\min} + \frac{y^{\mathrm{V}}_{\max} - y^{\mathrm{V}}_{\min}}{y^{\mathrm{W}}_{\max} - y^{\mathrm{W}}_{\min}} \left(p_y - y^{\mathrm{W}}_{\min}\right)$.
- **Rasterisierung einer Gerade**: Approximation der Strecke $\overline{\mathbf{P_1P_2}}$ durch Pixel $\mathbf{P_i} = (x_i, y_i)$.
  - ▷ Betrachte Gerade mit Steigung $\in [0,1]$
    - Setze ein Pixel pro Spalte.
    - Zuletzt gesetztes Pixel $(p_x,\ p_y)^\top$, Kriterium f. nächstes Pixel: Lage von $\mathbf{M} = \begin{pmatrix} m_x \\ m_y \end{pmatrix} := \begin{pmatrix} p_x + 1 \\ p_y + \frac{1}{2} \end{pmatrix}$ bzgl. $\overline{\mathbf{P_1P_2}}$, bzw. das Vorzeichen von $\delta := m_y - \left(y\text{-Wert v. } \overline{\mathbf{P_1P_2}} \text{ bei } m_x\right)$:
      - $\overline{\mathbf{P_1P_2}}$ verläuft unterhalb von $\mathbf{M}$ bzw. $\delta < 0 \Rightarrow (p_x + 1,\ p_y)^\top$
      - $\overline{\mathbf{P_1P_2}}$ verläuft oberhalb von $\mathbf{M}$ bzw. $\delta > 0 \Rightarrow (p_x + 1,\ p_y + 1)^\top$

      Grenzfall $M \in \overline{\mathbf{P_1P_2}}$ kann beliebig zugeordnet werden.
    - ▷ $\delta = \left(\frac{\Delta y}{\Delta x} x + b'\right) - m_y = \frac{\Delta y \cdot m_x - \Delta x \cdot m_y + b}{\Delta x}$
      mit $y(x) = \frac{\Delta y}{\Delta x} x + b'$ explizite Geradengleichung, $\Delta x := x_2 - x_1, \ldots, b := \Delta x \cdot b'$.
    - ▷ Entscheidungsvariable:
      $\overline{d} := \Delta y \cdot m_x - \Delta x \cdot m_y + b$
      $\rightarrow\ \overline{d} < 0 \Rightarrow (p_x + 1,\ p_y)^\top$
      $\rightarrow\ \overline{d} > 0 \Rightarrow (p_x + 1,\ p_y + 1)^\top$
      Kann inkrementell berechnet werden.
    - ▷ **Bresenham-Algorithmus**:
      INPUT: $\mathbf{P_1}, \mathbf{P_2}$, ganzzahlig in RC;
      Steigung von $\overline{\mathbf{P_1P_2}}$ zwischen 0 und 1

```
Dx = P_2.x - P_1.x;
Dy = P_2.y - P_1.y;
d = 2 * Dy - Dx;
inc1 = 2 * Dy;
inc2 = 2 * (Dy - Dx);
y = P_1.y;
for (x = P_1.x; x <= P_2.x; x++) {
    plot(x, y);
    if (d < 0) {
        d += inc1;
    } else {
        d += inc2;
        y++;
    }
}
```

- **Scanline-Algorithmus**: Approximation des Polygons $(\mathbf{P_1}, \ldots, \mathbf{P_k})$ durch *innere* Pixel.
  Kantenstruktur:

ID	x	k	$y_{\min}$	$y_{\max}$	dx	rx	Δy

  INITIALISIERUNG:
  - ▷ Edge-Table (ET):
    - enthält alle Kanten
    - x auf x-Wert des untersten Punktes (bei $y_{\min}$), k auf 0
    - Berechne $\Delta x \in \mathbb{Z}$, $\Delta y \in \mathbb{N}$ der Kante, daraus dx, dy mit $\frac{\Delta x}{\Delta y} = dx + \frac{rx}{\Delta y}$.
    - Kanten aufsteigend lexikografisch sortiert nach $(y_{\min}, x, dx)$
  - ▷ Active Edge Table (AET):

ID	x	k	s_x	p_x

    - leer
    - Sortierung nach aufsteigendem s_x

  ALGORITHMUS:
  - ▷ Aktuelle Scanline: $y = y_{\min}$ des ersten ET-Eintrags

  WHILE (ET $\neq \emptyset$ || AET $\neq \emptyset$) Do
  - ▷ AET: Neusortierung nach $\left(x, \frac{k}{\Delta y}\right)$ lex.
  - ▷ AET: Übernahme von $e \in$ ET mit $y_{\min} = y$ (Merge Sort)
  - ▷ Zeichnen der Spans
    - Reihenfolge der Kanten legt fest, ob Schnittpunkt mit linker oder rechter Kante vorliegt.
    - Berechne Schnittpunkt der Kante mit Scanline: $s_x = x + \frac{k}{\Delta y}$
    - Berechne zu zeichnenden Pixel:
      $p_x = \begin{cases} \lfloor s_x \rfloor & \text{rechte Kante} \\ \lceil s_x \rceil & \text{linke Kante} \end{cases}$
  - ▷ AET: Entfernen aller Kanten mit $y_{\max} = y$
  - ▷ $y = y + 1$
  - ▷ $\forall e \in$ AET : Update Schnittparameter x, k:
    - falls $k + rx < \Delta y$:
      $x \leftarrow x + dx; \quad k \leftarrow k + rx$
    - falls $k + rx \geq \Delta y$:
      $x \leftarrow x + dx + 1; \quad k \leftarrow k + rx - \Delta y$

  ENDWHILE

- **Schattierungsmodelle**:
  - ▷ **Flat-Shading**: eine Farbintensität pro Polygon, konstante Farbe
  - ▷ **Gouraud-Shading**: Farbintensität pro Polygonecke, lin. Interpolation im Inneren.

    Effekt: visuell glatte Flächen, **Machband-Effekt** (heller und dunkler „Streifen“).

    Bei n-gon mit $n > 3$ hängt Ergebnis von der Lage des Polygons ab (für Dreiecke eindeutig).
- Problem bei per-Vertex-Beleuchtung (Phong): kann keine spiegelnden Anteile erzeugen, wenn diese nicht nicht an den Eckpunkten vorhanden ist. Lösungsmöglichkeiten:
  - ▷ Verwende hoch aufgelöste Geometrie (hoher Aufwand).
  - ▷ Verwende **per-Fragment-Beleuchtung** (übergebe Phong-Parameter zum FS).
- Lineare Interpolation von Texturkoordinaten von Polygoneckpunkten ⇒ *falsches Ergebnis*, da interpolierte Texturkoordinate unabhängig von der Ausrichtung im Raum ist. z-Werte müssen mitberücksichtigt werden!

  Für $\mathbf{P}(\alpha) = (1-\alpha)\mathbf{P_1} + \alpha\mathbf{P_2}$, $\mathbf{P_i} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$ gilt:

  $$T_\mathsf{P}(\mathbf{P}(\alpha)) = (1-\beta)\,T_\mathsf{P}(\mathbf{P_1}) + \beta T_\mathsf{P}(\mathbf{P_2})$$
  $$\text{mit } \beta(\alpha) = \frac{\alpha z_2}{(1-\alpha)z_1 + \alpha z_2}.$$

  Gegeben Punkte auf Bildebene mit konstanter Schrittweite Δ_O:

  $\mathbf{Q}_\mathbf{i}^\mathsf{B} = (1-\beta_i)\mathbf{P}_\mathbf{1}^\mathsf{B} + \beta_i \mathbf{P}_\mathbf{2}^\mathsf{B}$, $\beta = i \cdot \Delta_\mathsf{O} \in [0,1]$

  sowie Texturkoordinaten $\mathbf{S_i}$ der Eckpunkte $\mathbf{P_i}$.

  Gesucht: Zu $\mathbf{Q}_\mathbf{i}^\mathsf{T}$ geh. Texturkoord. $\mathbf{S_i} = \begin{pmatrix} s_i \\ t_i \end{pmatrix}$.

  Lösung: Umkehrung:

  $$\alpha(\beta) = \frac{\beta z_1}{(1-\beta)z_2 + \beta z_1} \text{ ist urspr. } z\text{-W. von } \mathbf{P_i},$$
  $$\mathbf{S_i} = (1-\alpha(\beta))\,\mathbf{S_1} + \alpha(\beta)\,\mathbf{S_2}.$$

8 Strahlverfolgungstechniken

- **Raycasting**: Strahlverfolgung vom Beobachter durch jedes Pixel, Schnittpunktermittlung jedes Strahls mit jeweils nächstliegendem Objekt, Berechnung der Pixelfarbe anhand des Objektpunktes.
- **Raytracing**: Rekursive Strahlverfolgung ermöglicht indirekte Beleuchtungseffekte, Spiegelungen, Schatten, …
- Implizite Geometriebeschr. $F : \mathbb{R}^3 \to \mathbb{R}^3$ und $\{\mathbf{P} \in \mathbb{R}^3 \,|\, F(\mathbf{P}) = 0\}$ (**Bsp.:** Kreis als $\{(x,y) \,|\, x^2 + y^2 - 1 = 0\}$) ermöglicht effiziente Schnittpunktberechnung: $\mathbf{R}(\alpha) = \mathbf{V} + \alpha\vec{r}$ mit $\alpha \in \{\alpha > 0 \,|\, F(\mathbf{R}(\alpha) = 0)\}$.
  - ▷ Schnitt mit Ebene: $F(\mathbf{P}) = ((\mathbf{P} - \mathbf{Q}) \cdot \vec{n})$
  - ▷ Schnitt mit Polygon:
    1. ber. Schnittpkt. $\mathbf{S}$ mit Polygonebene
    2. Point-in-Polygon-Test (s. u.)
- **Point-in-Polygon-Test**:
  1. Projektion in 2D entlang treibender Achse (betragsgrößte Komponente) von $\vec{n}$ (simples Weglassen der Komponente in allen Eckpunkten des Polygons), dann verschieben, bis $\mathbf{S}$ im Ursprung: $\mathbf{P}_\mathbf{i}''$.
  2. Zähle Schnittpunkte eines beliebigen Strahls von $\mathbf{S}$ mit Polygonkanten:
     - ▷ **TA**: Sicherer Schnitt, wenn $x_i'' > 0 \wedge x_{i+1}'' > 0 \wedge y_i'' \cdot y_{i+1}'' < 0$
     - ▷ **TR**: Kein Schnitt möglich bei
       - $x_i'' < 0 \wedge x_{i+1}'' < 0$
       - $y_i'' < 0 \wedge y_{i+1}'' < 0$
       - $y_i'' > 0 \wedge y_{i+1}'' > 0$
     - ▷ Schnitt v. $\overline{\mathbf{P_i P_{i+1}}}$ mit pos. x-Achse:

       Suche $\alpha_i \in [0,1]$ mit

       $(1-\alpha_i)\mathbf{P}_\mathbf{i}'' y_i'' + \alpha_i y_{i+1}'' = 0$

       $\Rightarrow \alpha_i = \frac{-y_i''}{y_{i+1}'' - y_i''}$.

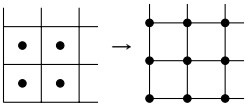
       Dann prüfe $(1-\alpha_i)\mathbf{P}_\mathbf{i}'' + \alpha_i \mathbf{P}_\mathbf{i+1}''$ auf positiven x-Wert.

       Sonderfall: $\mathbf{P}_\mathbf{i}''$ *auf* pos. x-Achse ⇒ $\mathbf{P}_\mathbf{i}''$ ist Schnitt für $\overline{\mathbf{P}_\mathbf{i-1}''\mathbf{P}_\mathbf{i}''}$, $\overline{\mathbf{P}_\mathbf{i}''\mathbf{P}_\mathbf{i+1}''}$ bei echtem y-Vorzeichenwechsel, d. h. nur wenn $y_{i-1}'' \cdot y_{i+1}'' < 0$.

       Liegen k aufeinanderfolgende Punkte $\mathbf{P}_\mathbf{i}'', \ldots, \mathbf{P}_\mathbf{i+k}''$ auf pos. x-Achse, muss $y_{i-1} \cdot y_{i+k+1}$ betr. werden.
- Intensität rekursiver Strahlverfolgung: Betrachte Strahlenbaum, Phong-Modell:
  - ▷ $I_d^{(i)}$: diffuser Anteil bei $\mathbf{P_i}$
  - ▷ $I_s^{(i)}$: spiegelnder Anteil bei $\mathbf{P_i}$
  - ▷ $I_t^{(i)}$: transmissiver Anteil bei $\mathbf{P_i}$
  - ▷ I_back: Hintergrundfarbe
  - ▷ $k_d^{(i)}, k_s^{(i)}, k_t^{(i)}$: diff., spieg., transm. Reflexionskoeffizienten (Materialkonstanten)
  - ▷ Visibilitätskoeffizient:

    $$\varrho(\mathbf{P}, \mathbf{L}) = \begin{cases} 0 & \mathbf{L} \text{ von } \mathbf{P} \text{ verdeckt} \\ 1 & \mathbf{L} \text{ von } \mathbf{P} \text{ sichtbar} \end{cases}$$

  Abbruchkriterium: feste Tiefe n (meist 2 bis 3).
- Strahlen ohne Ausdehnung erzeugen **Aliasing** (Treppenstufen) an Kanten.
  - ▷ Lösung: mehrere Strahlen pro Pixel.

    Kostet praktisch nichts, erzeugt Blur.
- Testen jedes Strahls gegen jedes Primitiv ist teuer ⇒ Einhüllen komplizierter Objekte in einfache Objekte, 2-stufiger Schnittest.
  - ▷ **Kay-Kajya-Algorithmus**: Schnitt $\mathbf{R}(\alpha)$ mit achsenparalleler Bounding Box $[x_\text{min}, x_\text{max}] \times [y_\text{min}, y_\text{max}] \times [z_\text{min}, z_\text{max}]$ für Strahl $\mathbf{R}(\alpha) = \mathbf{V} + \alpha\vec{r}$:

    Nahezu identisch mit Clipping gegen Fensterkanten.

    Algorithmus:

    Setze $\alpha_\text{min} \leftarrow -\infty$, $\alpha_\text{max} \leftarrow \infty$,
    `intersect` ← `true`
    For $i \in \{x, y, z\}$ Do
      $\alpha_\text{min}^{(i)} \leftarrow \min\{\alpha \,|\, \mathbf{R}(\alpha) \in \{i_\text{min}, i_\text{max}\}\}$
      $\alpha_\text{max}^{(i)} \leftarrow \max\{\alpha \,|\, \mathbf{R}(\alpha) \in \{i_\text{min}, i_\text{max}\}\}$
      If $\alpha_\text{max}^{(i)} < 0$ Then
        `intersect` ← `false`
        Break
      EndIf
      $\alpha_\text{min} \leftarrow \max\left\{\alpha_\text{min}, \alpha_\text{min}^{(i)}\right\}$
      $\alpha_\text{max} \leftarrow \max\left\{\alpha_\text{max}, \alpha_\text{max}^{(i)}\right\}$
      If $\alpha_\text{min} > \alpha_\text{max}$ Then
        `intersect` ← `false`
        Break
      EndIf
    EndFor

    Wir wollen nur wissen, *ob* wir einen Schnitt haben. Wenn ja, berechne Schnitt mit eigentlicher Geometrie.
  - ▷ Raumunterteilung mit **Octree**:

    Unterteilung der Szene in disjunkte, achsenparallele Quader; Anzahl Objekte pro Quader etwa gleich.

    Speichere pro Quader:
    - $\mathbf{P}_\text{min}, \mathbf{P}_\text{max}$
    - Nachbar-Quader in alle 6 Richtungen
    - Referenz auf Objekte, die (teilw.) im Quader liegen

    Schnittpunktermittlung:
    - ∀ Strahl: Ermittle Quader-Sequenz, die der Strahl durchläuft.
    - Schnitt-Test mit Objekten in Quaderfolge. Sobald etwas getroffen ist, kann abgebrochen werden.

    $\rightsquigarrow \log n$ Schnitttests (sonst n).
- **Beam tracing**: Verfolge kleine Strahlenbündel (i. A. 2×2) durch Octree.
  - ▷ Strahlen innerhalb der **Safety zone** (Zylinder) haben denselben Strahlenbaum.
  - ▷ parallele Schnitttests durch SIMD- (*single instruction, multiple data*)-Hardware, solange kohärent (meist 1. & 2. Rekursion).

9 Fortgeschrittene Rastergraphik

- *Triangle strips* & *triangle fans* zur Mehrfachverwendung eines Vertex & Sparen von Speicher.
- **Indizierter Vertex-Buffer**: Reihenfolge der Vertex-Attribute durch Index-Buffer, jeder Index kann mehrfach verwendet werden.
- **Transparenz mit α-Kanal**: zusätzlicher Kanal für Transparenz-Info: $(R, G, B, \alpha) \in [0,1]^4$
  - ▷ $\alpha = 0$: komplett transparent
  - ▷ $\alpha = 1$: komplett opaque

  α-Blending der aktuellen Pixelintensität im Framebuffer (*Destination*) mit der des zu zeichnenden Pixels (*Source*):

  $$\text{pixel}_\text{dest}(x,y) \leftarrow k_\text{src}(x,y) * \text{pixel}_\text{src}(x,y) + k_\text{dest}(x,y) * \text{pixel}_\text{dest}(x,y)$$

  Berechnung der Blendfaktoren $k_\text{src}, k_\text{dest}$:
  - ▷ `GL_ZERO`, `GL_ONE`
  - ▷ `GL_DEST_COLOR`, `GL_SRC_COLOR`
  - ▷ …

  Zeichenreihenfolge muss back-to-front sein!
- **Billboarding**: damit Objekttexturen (z. B. Grass, Bäume) sich nicht auffällig mitdrehen, verwende mehrere zueinander verdrehte Billboards, die sich nicht mitdrehen.
- **Framebuffer Objects** (**FBO**): Nutzung von erzeugten Bilddaten als Textur.
  - ▷ z. B. **Masken** (*stencils*)
  - ▷ Spiegeleffekt

  Pass 1: Zeichne Spiegelpolygon als Positiv-Maske in FBO

  Pass 2: Zeichne gespiegelte Szene (negative y-Werte) in Framebuffer

  Pass 3: Zeichne Szene ohne Maske mit Blending, hierbei hat Spiegel $0 < \alpha < 1$.
  - ▷ Bewegungsunschärfe (*motion blur*): Nutze zwei FBOs als Ping-Pong-Buffer und blende mehrere Instanzen des Objekt mit leicht verschobener Position zu einem Bild zusammen.
  - ▷ **Antialiasing**: Mehrfaches Rendern mit zufällig verschobener Bildebene (**Jittern**), Blending.