Aufgabe 1: Raytracing

Teilaufgabe 1a

Phong-Beleuchtungsmodell anwenden:

- Berechnung des ambienten Anteils (indirekte Beleuchtung, Licht von anderen Oberflächen)
- Berechnung der Reflektion
  - Berechnung des spekularen Reflektion. Diese findet nur in Richtung $R_L = 2N \cdot (N \cdot L)$ statt. In alle andren Richtungen fällt sie stark ab:

    $$I_s = k_s \cdot I_L \cdot \cos^n \alpha = k_s \cdot I_L (R_L \cdot V)^n$$

    wobei n der Phong-Exponent ist.
  - Berechnung der diffusen (Lambertschen) Reflektion:

    $$I_d = k_d \cdot I_L \cdot \cos\theta = k_d \cdot I_L \cdot (N \cdot L)$$

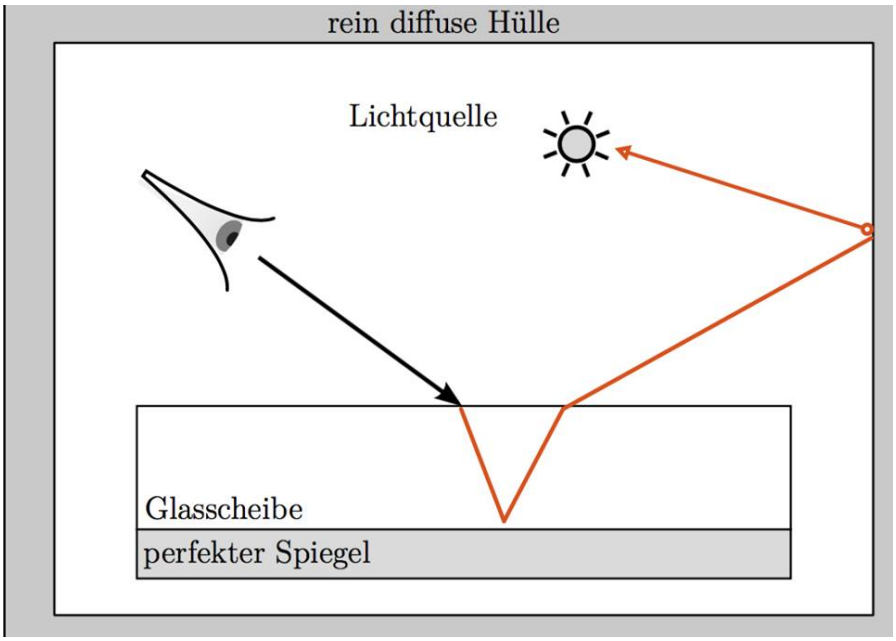
Also:

$$I = \underbrace{k_a \cdot I_L}_{\text{ambient}} + \underbrace{k_d \cdot I_L \cdot (N \cdot L)}_{\text{diffus}} + \underbrace{k_s \cdot I_L (R_L \cdot V)^n}_{\text{spekular}}$$

Auch:

- Berechnung von Schattenstrahlen
- Weitere, Strahlen rekursive verschießen

Teilaufgabe 1b



Aufgabe 2: Farben und Spektren

Teilaufgabe 2a: Farbräume

- RGB: LCD/CRT-Displays
- CMYK: Drucker
- HSV: Comuter Vision
- XYZ Color Space: Farbraum für Konversion zwischen Farbräumen
- Lab-Farbraum: TODO

Teilaufgabe 2b: Gamma-Korrektur

Nennen Sie einen Grund für die Durchführung von Gamma-Korrektur!

Helligkeiten werden vom Menschen nicht linear wahrgenommen und von einem Bildschirm auch nicht linear ausgegeben. Um trotzdem einen linearen Helligkeitseindruck entstehen zu lassen wendet man die Gamakorrektur an.

Teilaufgabe 2c: Metamerismus

Metamerismus ist das Phänomen, dass unterschiedliche Spektren gleich aussehen können. Dies ist wichtig für Monitore, da sie aufgrund dieses Phänomens mit nur 3 Farben den gleichen Farbeindruck erwecken können wie mit einem komplexeren Spektrum.

Aufgabe 3: Transformationen

Teilaufgabe 3a

Transformationen mit homogenen Koordianten laufen Grundsätzlich nach folgendem Schema ab:

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{pmatrix} \leftarrow T \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Die Transformationsmatrix T für die Translation von homogenen Koordinaten ist von der Form

$$T = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

Die Transformationsmatrix R für eine Rotation um den Punkt $c = (c_x, c_y)$ um den Winkel α ist

$$R_{\alpha,c} = \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Die Idee ist nun, zuerst eine Rotation um 90° gegen den Urzeigersinn um $(0, 0)$ zu machen (Matrix R). Dann wird das Rechteck in Richtung der x-Achse um die hälfte gestaucht (Matrix S) und schließlich um 0.5 nach links verschoben (Matrix T):

$$R = \begin{pmatrix} \cos 90 & -\sin 90 & 0 \\ \sin 90 & \cos 90 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

$$S = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

$$T = \begin{pmatrix} 1 & 0 & -0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

$$M = T \cdot S \cdot R \tag{4}$$

$$= \begin{pmatrix} 0 & -0.5 & -0.5 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

Zur Kontrolle:

$$M \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 0 \\ 1 \end{pmatrix} \qquad M \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 1 \\ 1 \end{pmatrix} \tag{6}$$

$$M \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} \qquad M \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \tag{7}$$

Teilaufgabe 3b: Koordinatensysteme

$$z' = E - C \tag{8}$$

$$x' = u \times z' \tag{9}$$

$$y' = z' \times x' \tag{10}$$

Das Koordinaten-System ist dann gegeben durch die normierten Basisvektoren sowie den Ursprung E.

Teilaufgabe 3c

Die Transformation von Welt- in Kamerakoordinaten wird auch *Kameratransformation* genannt. Die virtuelle Kamera ist durch die Position **e** und die negative Blickrichtung **w** definiert. Mithilfe des Up-Vektors **up** ergibt sich

$$\mathbf{u} = \mathbf{up} \times \mathbf{w} \qquad \mathbf{v} = \mathbf{w} \times \mathbf{u}$$

Es wird zuerst eine Translation um $-\mathbf{e}$ durchgeführt und dann eine Transformation in das Kamera-Koordiantensystem durchgeführt. Die Basis des Kamera-Koordinatensystems ist $\mathbf{u}, \mathbf{v}, \mathbf{w}$.

Das Verschieben ist einfach die Matrix

$$T = \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

nun muss noch rotiert werden:

$$R = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die Gesamttrasformationsmatrix ist also $V = R \cdot T$.

(vgl. `03_ Transformationen und homogene Koordinaten.pdf`, Folie 50)

Aufgabe 4: Phong-Beleuchtungsmodell

Teilaufgabe 4a

$$\mathbf{r}_l = 2 \cdot \mathbf{n} \cdot (\mathbf{n} \cdot \mathbf{l}) - \mathbf{l}$$

Kaptiel 2, Folie 96.

Teilaufgabe 4b

Die spekulare Komponente im Phong Beleuchtungsmodell lautet

$$I_s = k_s \cdot I_L \cdot \cos^n \alpha = k_s \cdot I_L (\mathbf{r}_l \cdot \mathbf{v})^n$$

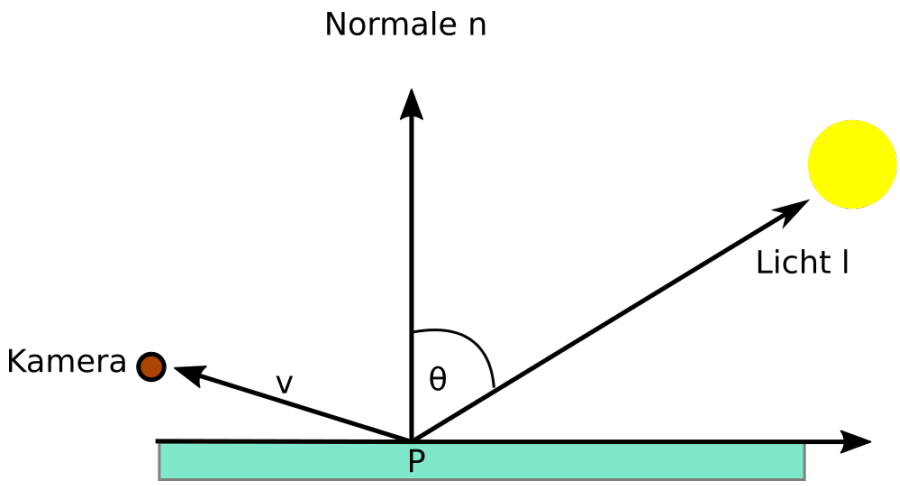


Abbildung 1: Situationsskizze. r_l ist senkrecht auf L in P.

Teilaufgabe 4c

(i) Wie verändert sich das Glanzlicht, wenn e größer wird?
⇒ Das Glanzlicht wird kleiner, aber intensiver.
(ii) Wie verändert sich das Glanzlicht, wenn die Kugel um eine beliebige Achse rotiert?
⇒ Da $N, L, R_L, n, I_L, k_a, k_d, k_s, V$ gleich bleiben ändert sich das Glanzlicht nicht.

Aufgabe 5: Dreiecke und Schattierung

Teilaufgabe 5a

Lösung 1: $n = \text{normalize}(\text{cross}(P_3 - P_1, P_2 - P_1))$

Lösung 2: Drei Punkte P_1, P_2, P_3 definieren eine Ebene. Diese Ebene ist eine Menge von Punkten

$$\{x \in \mathbb{R}^3 | x \text{ liegt auf der Ebene von } P_1, P_2, P_3\} = \{x \in \mathbb{R}^3 = n \cdot x = d\}$$

Es gilt also folgendes Gleichungssystem zu lösen:

$$\|n\| = 1 \tag{11}$$
$$n_x \cdot P_{1,x} + n_y \cdot P_{1,y} + n_z \cdot P_{1,z} = d \tag{12}$$
$$n_x \cdot P_{2,x} + n_y \cdot P_{2,y} + n_z \cdot P_{2,z} = d \tag{13}$$
$$n_x \cdot P_{3,x} + n_y \cdot P_{3,y} + n_z \cdot P_{3,z} = d \tag{14}$$

Teilaufgabe 5b

Die Normalen der Vertices eines Dreiecksnetzes können wie folgt berechnet werden:

1. Berechne Normale jedes Dreiecks
2. Für jeden Vertex wird nun die Summe der Normalen der angrenzenden Dreiecke gebildet.
3. Die Vertex-Normalen werden normalisiert, indem sie durch ihre Länge geteilt werden.

Teilaufgabe 5c

Gouraud-Shading berechnet die Werte (z.B. Farbe) an den Eckpunkten der Dreiecke und interpoliert die Werte im inneren der Dreieck mithilfe der Eckpunktwerte.

Phong-Shading nutzt die normalen an den Eckpunkten zur interpolation von Normalen innerhalb des Dreiecks. Diese werden für weitere Beleuchtungsberechnungen verwendet.

Insbesondere bei Glanzlichtern kann dies einen Unterschied machen. Beim Gouraud-Shading können Glanzlichter verloren gehen.

Teilaufgabe 5d

Gouraud-Shading im Vergleich zu Phong-Shading

- Vorteil: Schnellere Berechnung
- Nachteil: Schlechtere Ergebnisse

Aufgabe 6: Texturen und Texturfilterung

Teilaufgabe 6a

vgl. geometrische Interpretation (Kaptiel 2, Folie 43)

$$\lambda_1 = \frac{A_\Delta(S, P_2, P_3)}{A_\Delta(P_1, P_2, P_3)} \tag{15}$$

$$\lambda_2 = \frac{A_\Delta(P_1, S, P_3)}{A_\Delta(P_1, P_2, P_3)} \tag{16}$$

$$\lambda_3 = \frac{A_\Delta(P1, P_2, S)}{A_\Delta(P_1, P_2, P_3)} \tag{17}$$

Teilaufgabe 6b

$$T_S = \sum_{i=1}^{3} \lambda_i T_i$$

Teilaufgabe 6c

- Magnification; Beipsiele: Unschärfe, harte Kanten, Blockbildung
- Minification; Beispiel: Aliasing-Artefakte durch Unterabtastung

Teilaufgabe 6d

Welche texturbezogenen Lösungen bietet OpenGL an, um die Artefakte aus Aufgabe c) zu reduzieren? Erklären Sie stichpunktartig, wie sie funktionieren!

Mipmaping: speichere orignale Textur in 0. Stufe, dann halbiere pro Stufe die Textur in jeder Dimension Anisotrope Texturfilterung (RIPs): Vorfilterungen unabhängig jeder Achse RIPs bzw. Anisotrope Filterung ist anscheinend keine Kernfunktionalität von OpenGL, aber durch Extensions unterstützt.

Teilaufgabe 6e

Unter trilinearer Texturfilterung versteht man eine bilineare Interpolation der Stufe n, eine bilineare Interpolation der Stufe $n + 1$ und dann eine lineare Interpolation dieser beiden Farben.

Teilaufgabe 6f

- `GL_TEXTURE_MAG_FILTER`: The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texture element. It sets the texture magnification function to either `GL_NEAREST` or `GL_LINEAR`. `GL_NEAREST` is generally faster than `GL_LINEAR`, but it can produce textured images with sharper edges because the transition between texture elements is not as smooth. The initial value of `GL_TEXTURE_MAG_FILTER` is `GL_LINEAR`.
  Quelle: www.talisman.org/opengl-1.1/Reference/glTexParameter.html
- `GL_TEXTURE_MIN_FILTER`: The texture minifying function is used whenever the pixel being textured maps to an area greater than one texture element. There are six defined minifying functions. Two of them use the nearest one or nearest four texture elements to compute the texture value. The other four use mipmaps. [...]
  Quelle: www.talisman.org/opengl-1.1/Reference/glTexParameter.html

Bilineare Filterung (Kaptiel 4, Folie 38):

$$a = \frac{\frac{3}{8} - \frac{1}{4}}{1/2} = \frac{1}{4} \tag{18}$$

$$b = \frac{\frac{1}{2} - \frac{1}{4}}{1/2} = \frac{1}{2} \tag{19}$$

$$t_{12} = t_1 + a(t_2 - t_1) = 16 + \frac{1}{4} \cdot (-16) = 12 \tag{20}$$

$$t_{34} = (1 - a)t_3 + at_4 = \frac{7}{8} \cdot 0 + \frac{1}{4} \cdot 32 = 8 \tag{21}$$

$$t = (1 - b)t_{12} + bt_{34} = \frac{1}{2} \cdot 12 + \frac{1}{2} \cdot 8 = 10 \tag{22}$$

Aufgabe 7: Projektionen

Wir müssen eine Matrix $M \in \mathbb{R}^{3\times 3}$ finden, welche die Projektion von Punkten $P \in [0,\infty)^2$ auf den Unterraum $\{P \in [0,\infty)^2 | P_x + P_y = 1\}$ transformiert.

Die Homogenen Koordinaten (x, y, z) entsprechen den 2D-Koordinaten $(\frac{x}{z}, \frac{y}{z})$.

Es gilt, dass alle Punkte auf der x-Achse auf (1, 0) abgebildet werden:

$$m_{1,1} \cdot x_1 + m_{1,2} \cdot 0 + m_{1,3} \cdot 1 = m_{3,1} \cdot x_1 + m_{3,2} \cdot 0 + m_{3,3} \cdot 1 \quad \forall x_1 \in [0,\infty) \tag{23}$$
$$m_{2,1} \cdot x_1 + m_{2,2} \cdot 0 + m_{2,3} \cdot 1 = 0 \quad \forall x_1 \in [0,\infty) \tag{24}$$

Daraus folgt: $m_{2,1} = 0, m_{2,3} = 0$. Außerdem lässt sich Gleichung (23) vereinfachen:

$$m_{1,1} \cdot x_1 + m_{1,3} = m_{3,1} \cdot x_1 + m_{3,3} \quad \forall x_1 \in [0,\infty) \tag{25}$$
$$(m_{1,1} - m_{3,1}) \cdot x_1 + (m_{1,3} - m_{3,3}) = 0 \quad \forall x_1 \in [0,\infty) \tag{26}$$

Daher gilt:

- $m_{1,1} = m_{3,1}$
- $m_{1,3} = m_{3,3}$

Bisher wissen wir:

$$M = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{3,3} \\ 0 & m_{2,2} & 0 \\ m_{1,1} & m_{3,2} & m_{3,3} \end{pmatrix}$$

Des weiteren gilt, dass alle Punkte auf der y-Achse auf (0, 1) abgebildet werden:

$$m_{1,1} \cdot 0 + m_{1,2} \cdot x_2 + m_{1,3} \cdot 1 = 0 \quad \forall x_2 \in [0,\infty) \tag{27}$$
$$m_{2,1} \cdot 0 + m_{2,2} \cdot x_2 + m_{2,3} \cdot 1 = m_{3,1} \cdot 0 + m_{3,2} \cdot x_2 + m_{3,3} \cdot 1 \quad \forall x_2 \in [0,\infty) \tag{28}$$

Wie zuvor, folgt: $m_{1,2} = m_{1,3} = 0$. Es verbleiben 3 Parameter.

Bisher wissen wir:

$$M = \begin{pmatrix} m_{1,1} & 0 & 0 \\ 0 & m_{2,2} & 0 \\ m_{1,1} & m_{3,2} & 0 \end{pmatrix}$$

Es gilt, dass alle Punkte auf der Diagonalen auf $(1/2, 1/2)$ abgebildet werden:

$$m_{1,1} \cdot x = 1/2 \cdot (m_{1,1}x + m_{3,2}x) \quad \forall x \in [0,\infty) \tag{29}$$
$$m_{2,2} \cdot x = 1/2 \cdot (m_{1,1}x + m_{3,2}x) \quad \forall x \in [0,\infty) \tag{30}$$

Für $x \neq 0$ kann man hier beide Gleichungen jeweils durch x teilen und erhält:

$$m_{1,1} = m_{3,2} \quad \forall x \in (0, \infty) \tag{31}$$
$$m_{2,2} = 1/2 \cdot (m_{1,1} + m_{3,2}) \quad \forall x \in (0, \infty) \tag{32}$$
$$\overset{32}{\Rightarrow} m_{2,2} = m_{1,1} \tag{33}$$

Bisher wissen wir:

$$M = \begin{pmatrix} m_{1,1} & 0 & 0 \\ 0 & m_{1,1} & 0 \\ m_{1,1} & m_{1,1} & 0 \end{pmatrix}$$

Nun gilt für die Transformation:

$$T(\begin{pmatrix} x \\ y \end{pmatrix}) = M \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{34}$$
$$= \begin{pmatrix} m_{1,1} \cdot x \\ m_{1,1} \cdot y \\ m_{1,1} \cdot (x + y) \end{pmatrix} \tag{35}$$
$$= \begin{pmatrix} \frac{x}{x+y} \\ \frac{y}{x+y} \end{pmatrix} \tag{36}$$

Die Konkrete Wahl von $m_{1,1}$ ist also egal, solange $m_{1,1} \neq 0$. Wähle oBdA $m_{1,1} = 1$ und daher:

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

Für den Punkt (2,1) gilt also:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}_H = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}_H = \begin{pmatrix} 2/3 \\ 1/3 \end{pmatrix}$$

Aufgabe 8: Beschleunigungsstrukturen

Teilaufgabe 8a

Um Schnitttests zu beschleunigen, kann man den Raum durch ein Gitter in Zellen unterteilen. Für jede Zelle nimmt man die AABB des Objekts um zu bestimmen, ob ein Objekt in der Zelle ist. Wenn man dann den Schnittest macht, schaut man zuerst welche Zellen der Strahl traversiert. Für jede Zelle schaut man ob dort Objekte sind und macht den Schnittest mit den Objekten. Nun kann ein Objekt in mehreren Zellen sein, aber den Schnitttest macht man nur das erste mal. Dannach speichert man sich, dass der Schnitttest des Strahls mit dem Objekt bereits gemacht wurde. Dieses Speichern nennt man *mailboxing* (vgl. Kapitel 5, Folie 56).

Die Schnitttests sind in Abbildung 2 dargestellt.

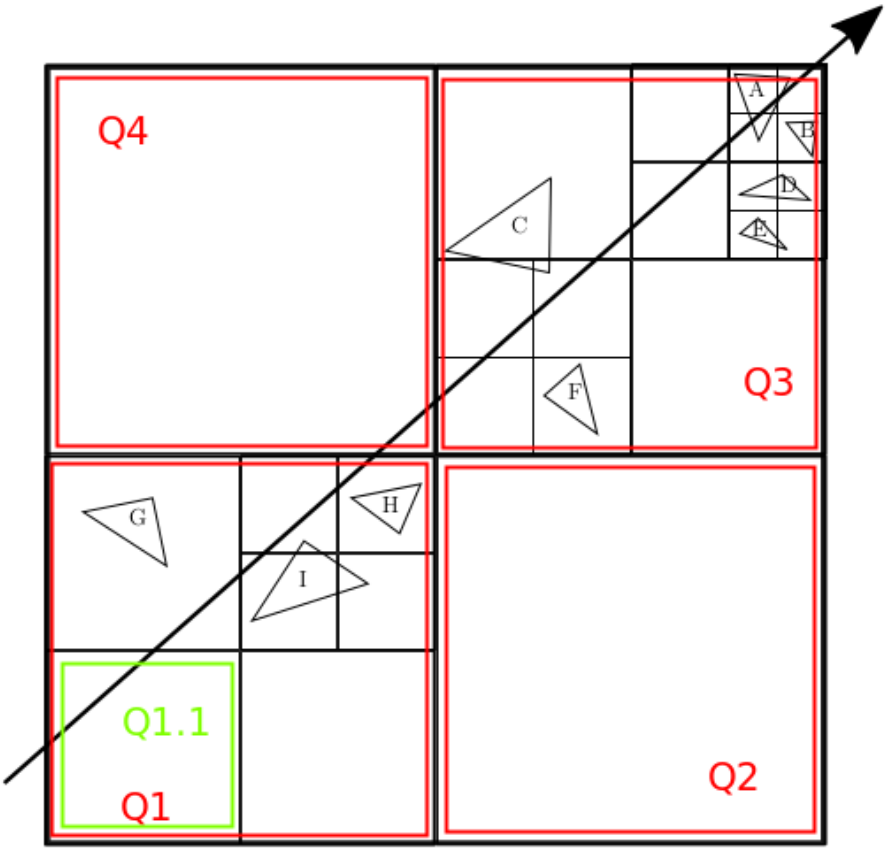


Abbildung 2: Alle durchgeführten Schnitttests.

Die Anzahl der Schnittests ist:

- Q1 mit Strahl: Schnitt

  - Q1.1 mit Strahl: Schnitt, aber hat kein Kind
  - Q1.2 mit Strahl: kein Schnitt
  - Q1.3 mit Strahl: Schnitt
    - Q1.3.1 mit Strahl: Schnitt. Schnitttest mit I.
    - Q1.3.2 mit Strahl: kein Schnitt.
    - Q1.3.3 mit Strahl: Schnitt. Schnitttest mit H.
    - Q1.3.4 mit Strahl: Schnitt. Schnitttest mit I.
  - Q1.4 mit Strahl: Schnitt. Schnittest mit G.
- Q2 mit Strahl: kein Schnitt
- Q3 mit Strahl: Schnitt
  - Q3.1 mit Strahl: Schnitt
    - Q3.1.1: Schnitt.
    - Q3.1.2 mit Strahl: Kein Schnitt.
    - Q3.1.3 mit Strahl: Schnitt. Schnitttest mit C.
    - Q3.1.4 mit Strahl: Schnitt. Schnitttest mit C.
  - Q3.2 mit Strahl: kein Schnitt.
  - Q3.3 mit Strahl: Schnitt.
    - Q3.3.1 mit Strahl: Schnitt.
    - Q3.3.2 mit Strahl: Kein Schnitt
    - Q3.3.3 mit Strahl: Schnitt
      - Q3.3.3.1 mit Strahl: Schnitt. Schnitttest mit A.
      - Q3.3.3.2 mit Strahl: kein Schnitt
      - Q3.3.3.3 mit Strahl: Schnitt. Schnitttest mit A.
      - Q3.3.3.4 mit Strahl: Schnitt. Schnitttest mit A.
    - Q3.3.4 mit Strahl: Schnitt.
  - Q3.4 mit Strahl: Schnitt. Schnitttest mit C.
- Q4 mit Strahl: Schnitt.

Teilaufgabe 8b

1 Der Baum einer Hüllkörperhierarchie ist immer balanciert.
⇒ Falsch.
2 Der Speicherbedarf für ein reguläres Gitter ist unabhängig von der Anzahl der Primitive.
⇒ Falsch. Man muss pro Zelle speichern welche Objekte diese Zelle enthält.
3 Ein kD-Baum hat immer achsenparallele Split-Ebenen.
⇒ Richtig. Siehe Folie 85.
4 Ein kD-Baum braucht spezielle Vorkehrungen, um redundante Schnitttests mit demselben Dreieck auszuschliessen.
⇒ TODO
5 Ein Verfahren zur Erzeugung eines kD-Baumes erzeugt auch gültige BSP-Bäume.
⇒ Richtig. kD-Bäume sind spezialfälle von BSP-Bäumen.
6 Reguläre uniforme Gitter leiden nicht unter dem Teapot-in-a-Stadium Problem.

⇒ Falsch. Das „Teapot-in-a-Stadium“ Problem bezeichnet das Problem, dass Resourcen bei regulären Gittern verschwendet werden weil die meisten Zellen leer sind und einige wenige die gesamte Komplexität beinhalten. Ein riesiges Objekt (das Stadium) und ein deutlich kleineres, aber komplexes Objekt (Teapot) rufen es hervor.

7 Die Komplexität der Bestimmung eines Schnittpunktes in einem BSP-Baum mit n Primitiven liegt im Optimalfall in $\mathcal{O}(\log n)$.

⇒ Richtig. Im Optimalfall ist der BSP-Baum balanciert.

8 Das Traversieren einer Hüllkörperhierarchie kann abgebrochen werden sobald ein Schnittpunkt gefunden wurde.

⇒ Falsch. Es könnte einen Schnitt geben, der näher zur Kamera ist (TODO: Beispiel in Folien raussuchen.)

Aufgabe 9: Instancing (GLSL)

shader.vert

```
uniform mat4 VP; // View-Projection Matrix
uniform sampler2D heightMap; // Höhenkarte (Rot-Komponente speichert Höhe)
uniform vec2 rcpHMS; // Kehrwerte der heightMap-Auflösung in s- und t-Richtung

// (r,g,b) == (x-Position, z-Position, Skalierung)
uniform sampler1D treeInstanceData;
in vec3 POS; // Vertex-Position in Objektkoordinaten
in vec3 NRM; // Vertex-Normale in Objektkoordinaten
out vec3 wpos; // Vertex-Position in Weltkoordinaten
out vec3 wnrm; // Vertex-Normale in Weltkoordinaten
void main(void)
{
    vec3 data = texelFetch(treeInstanceData, gl_InstanceID, 0);

    // Aufgabe beginnt hier:

    // Höhe aus Höhenkarte auslesen
    float height = texture(heightMap, data.xy * rcpHMS).r;

    // Baum wird um data.xy in xz-Richtung und um height in y-Richtung
    // verschoben
    vec3 translation = vec3(data.x, height, data.y);

    // Weltposition durch Skalierung und Verschiebung
    wpos = data.z * POS + translation;

    // Die Normale bleibt gleich
    wnrm = NRM;

    // Transformation der Position von Welt- in Clipkoordinaten
    gl_Position = VP * vec4(wpos, 1.);
}
```

Aufgabe 10: Normal Mapping in Objektkoordinaten (GLSL)

shader.frag

```
uniform samplerCube eMap; // Environment Map mit diffuser Beleuchtung
uniform sampler2D nMap; // Normal Map mit Normalen in Objektkoordinaten
uniform mat3 matO2W; // Transformationsmatrix von Objekt- zu Weltkoordinaten
uniform mat3 matW2O; // Transformationsmatrix von Welt- zu Objektkoordinaten
uniform vec3 kd; // diffuser Reflexionskoeffizient
in vec2 tc; // Texturkoordinate des Fragments
out vec3 color; // Ausgabefarbe des Fragments
void main(void)
{
    // Die Normale wird aus der Normal Map ausgelesen...
    vec3 normal = texture(nMap, tc).xyz;

    // ... und mithilfe der gegebenen Matrizen in Weltkoordinaten transformiert.
    vec3 normal_wc = matO2W * normal;

    // Die Ausgabefarbe wird durch diffuse Reflexion mit dem
    // Reflexionskoeffizienten kd und der Environment Map bestimmt.
    color = kd * texture(eMap, normal_wc).rgb;
}
```

Aufgabe 11: Bézier-Kurven und Bézier-Splines

Teilaufgabe 11a

Damit ein Bézier-Spline, welcher aus C^2-stetig ist, muss er C^0 und C^1 stetig sein. Für C^0-Stetigkeit muss nur $\mathbf{b}_3 = \mathbf{c}_0$ gelten.

Für C^1-Stetigkeit muss $\mathbf{b}_3 - \mathbf{b}_2 = \mathbf{c}_1 - \mathbf{c}_0$ gelten. Das heißt:

$$\mathbf{b}_3 - \mathbf{b}_2 \overset{!}{=} \mathbf{c}_1 - \mathbf{c}_0 \tag{37}$$

$$\begin{pmatrix}7\\8\end{pmatrix} - \begin{pmatrix}5\\9\end{pmatrix} \overset{!}{=} \mathbf{c}_1 - \begin{pmatrix}7\\8\end{pmatrix} \tag{38}$$

$$\Leftrightarrow \mathbf{c}_1 \overset{!}{=} \begin{pmatrix}9\\7\end{pmatrix} \tag{39}$$

Für C^2-Stetigkeit muss zusätzlich noch folgende Bedingung gelten:

$$\mathbf{b}_2 + (\mathbf{b}_2 - \mathbf{b}_1) \overset{!}{=} \mathbf{c}_1 + (\mathbf{c}_1 - \mathbf{c}_2) \tag{40}$$

$$\begin{pmatrix}5\\9\end{pmatrix} + \left(\begin{pmatrix}5\\9\end{pmatrix} - \begin{pmatrix}3\\9\end{pmatrix}\right) \overset{!}{=} \begin{pmatrix}9\\7\end{pmatrix} + \left(\begin{pmatrix}9\\7\end{pmatrix} - \mathbf{c}_2\right) \tag{41}$$

$$\Leftrightarrow \begin{pmatrix}7\\9\end{pmatrix} \overset{!}{=} \begin{pmatrix}18\\14\end{pmatrix} - \mathbf{c}_2 \tag{42}$$

$$\Leftrightarrow \mathbf{c}_2 \overset{!}{=} \begin{pmatrix}11\\5\end{pmatrix} \tag{43}$$

Das sieht dann wie folgt aus:

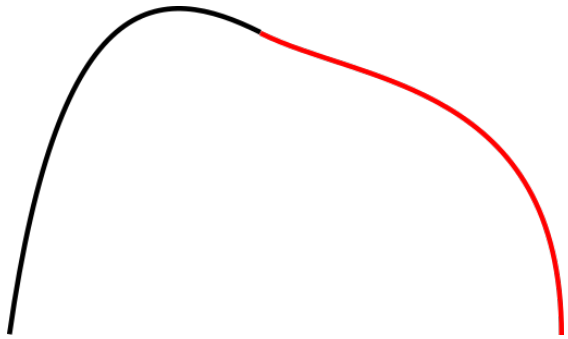


Abbildung 3: Ergebnisspline. TODO: Ich glaube c_2 ist falsch.

Teilaufgabe 11b

Bézierkurven sind immer:

- **Tangentenbedingung**: c_0c_1 ist Tangential an die Bezierkurve am Anfang, c_2c_3 ist Tangential an die Bezierkurve am Ende.
- **Wertebereich**: Bézierkurven liegen innerhalb der konvexen Hülle, die durch die 4 Kontrollpunkte gebildet werden.
- **Endpunktinterpolation**: Bézierkurven beginnen immer beim ersten Kontrollpunkt und enden beim letzten Kontrollpunkt.
- **Variationsredukion**: Eine Bézierkurve F wackelt nicht stärker als ihr Kontrollpolygon B ($\sharp(H \cap F) \leq \sharp(H \cap B)$).
- **Affine Invarianz**

Die ersten zwei Eigenschaften gelten für alle 3 dargestellten Kurven, die dritte ist bei der dritten Kurve jedoch verletzt. Also ist die dritte Kurve mit den dargestellten Kontrollpunkten keine Bézierkurve.

Begründung: Legt man eine Hyperebene H (Gerade) so, dass sie kurz über dem zweiten Kontrollpunkt und leicht unter dem zweiten Kontrollpunkt liegt, so schneidet H das Kontrollpolygon genau 1 mal, aber 3 mal die Bézierkurve.