Aufgabe 1: Farben und Farbwahrnehmung

Teilaufgabe 1a: Chromatizitätsdiagramm

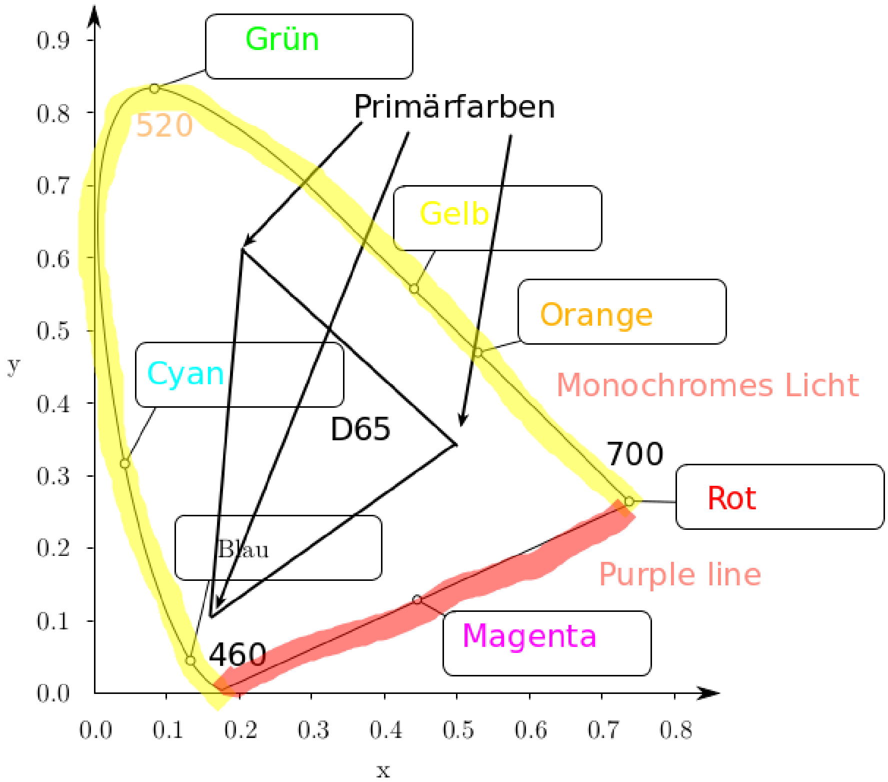


Abbildung 1: Aufgabe 1a

Teilaufgabe 1b

Welcher der Farbeindrücke aus Aufgabe a) lässt sich nicht durch monochromatisches Licht erzeugen?

Alles auf der Purple line. Also insbesondere **Magenta**.

Teilaufgabe 1c

$$x = \frac{X}{X+Y+Z} \tag{1}$$

$$y = \frac{Y}{X+Y+Z} \tag{2}$$

Teilaufgabe 1d

(2) < (3) < (1), also
RGB < Raum aller Farben die durch 100 monochromatische Leuchtdioden darstellbar sind < XYZ

Teilaufgabe 1e

#	Aussage	Wahr	Falsch	Begründung
1	Den Weißpunkt eines Farbraums bezeichnet man auch als Tristimuluswert.	☐	☑	Die RGB-Werte sind die Tristimuluswerte. Der Weißpunkt heißt üblicherweise D[Zahl], wobei die Zahl die Temperatur angibt. D65 hat eine Farbtemperatur von ca. 6504 K.
2	Die subjektiv empfundene Stärke von Sinneseindrücken ist proportional zum Logarithmus ihrer Intensität.	☑	☐	
3	Jeder Farbeindruck für den Menschen kann mit drei Grundgrößen beschrieben werden.	☑	☐	1. Graßmannsches Gesetz

Aufgabe 2: Whitted-Style Raytracing

Teilaufgabe 2a-d

Teilaufgabe 2e

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t \tag{3}$$

$$1 \cdot \frac{4}{10} = 1.5 \sin\theta_t \tag{4}$$

$$\Leftrightarrow \sin\theta_t = \frac{4}{15} = \frac{2}{7.5} \tag{5}$$

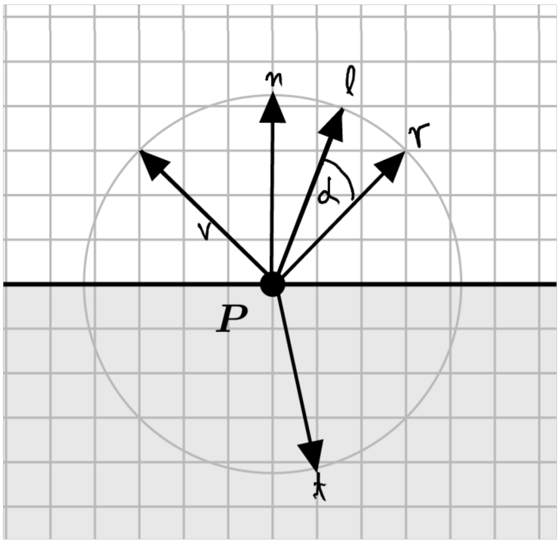


Abbildung 2: Lösung. Siehe Issue auf Github

Teilaufgabe 2f

$$\mathbf{r}_L = 2 \cdot \mathbf{n} \cdot (\mathbf{n} \cdot \mathbf{l}) - \mathbf{l} \tag{6}$$
$$\alpha = \arccos(\mathbf{r}_L \cdot \mathbf{v}) \tag{7}$$
$$I_s = k_s \cdot I_L \cdot \cos^n \alpha \tag{8}$$

wobei k_s ein Materialparameter und I_L die Intensität der Lichtquelle ist. n ist der Phong-Exponent des Materials.

Teilaufgabe 2g

Snellsches Brechungsgesetz

$$\eta_i \sin \theta_i = \eta_t \sin \theta_t$$

Aufgabe 3: Transformationen

$$\begin{pmatrix} s_x & h_x & t_x \\ h_y & s_y & t_y \\ a & b & c \end{pmatrix}$$

- Die Parameter s_x, s_y skalieren in Richtung der x bzw. y Achse.

- Die Parameter h_x, h_y scheren in Richtung der x bzw. y Achse.
- Die Parameter t_x, t_y führen eine Translation in x bzw. y Richtung aus.
- Die Parameter a, b, c skalieren.

Die Matrix

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

rotiert um θ um den Ursprung (gegen den Uhrzeigersinn.)

- Bild 1: Translation um 1 in x und 3 in y-Richtung.
- Bild 2: Scherung im -2 in y-Richtung.
- Bild 3: Rotation um 45° gegen den Uhrzeigersinn.
- Bild 4: In x-Richtung um $1/2$ stauchen, in y-Richtung um 3 strecken und dann um 4 nach rechts verschieben.
- Bild 5: Projektion auf die zur x-Achse parallele Gerade durch $(0, 3)$.

Aufgabe 4

Teilaufgabe 4a

Es müssen nur die Mittelwerte berechnet werden, also:

- Stufe 1: 5, 3, 8, 4
- Stufe 2: 4, 6
- Stufe 3: 5

Teilaufgabe 4b

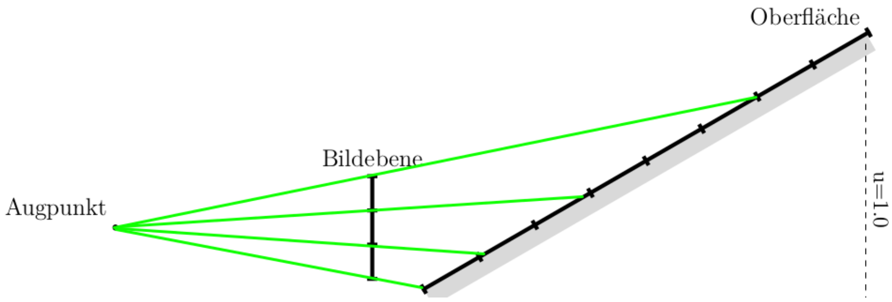


Abbildung 3: Aufgabe 4b; Der Footprint eines Bildpixels in der Textur wird ermittelt, indem man überprüft wie viele Texel diesen Bildpixel beeinflussen.

- oben: 2.8
- mitte: 1.8
- unten: 1.1

Siehe Abbildung 3 (vgl. Kapitel 4, Folie 58)

Teilaufgabe 4c

Teilaufgabe 4c (I)

TODO

Teilaufgabe 4c (II)

TODO

Teilaufgabe 4c (III)

TODO

Teilaufgabe 4d

#	Aussage	Wahr	Falsch	Begründung
1	Texturkoordinaten müssen sich immer im Intervall $[0; 1]$ befinden.	☐	☑	Koordinaten, die außerhalb liegen, werden je nach Einstellung (Clamp, Repeat, ..) auf Texturkoordinaten abgebildet
2	Texturkoordinaten können als Attribute der Eckpunkte (Vertizes) übergeben werden und werden als solche interpoliert.	☑	☐	Nicht eindeutig formuliert, aber siehe Kapitel 4, Folie 28
3	Texturkoordinaten müssen für die Darstellung wie Eckpunktkoordinaten der Model-View-Transformation unterzogen werden.	☐	☑	Interpolation, s. Kapitel 4, Folie 31

Aufgabe 5: Vorgefilterte Environment-Maps

Teilaufgabe 5a

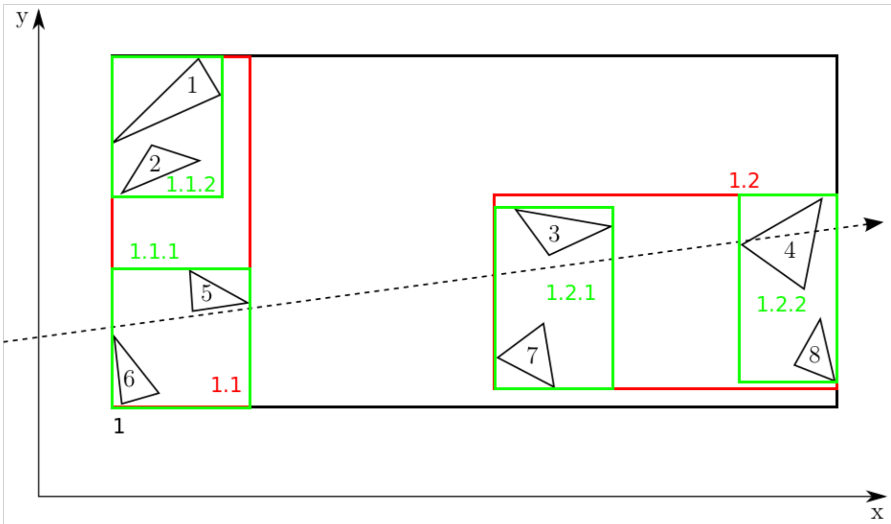
1. Diffuse Beleuchtung: $\mathbf{n}$
2. Imperfekte Spiegelung: $\mathbf{r}$

Teilaufgabe 5b

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

Aufgabe 6: Hierarchische Datenstrukturen

Teilaufgabe 6a



Teilaufgabe 6b

Inklusive Schnitttests der AABB-Hüllkörper:

1. 1
2. 1.1
3. 1.1.1
4. 5, 6
5. 1.1.2
6. 1.2
7. 1.2.1
8. 3, 7
9. 1.2.2
10. 4, 8

Teilaufgabe 6c

#	Aussage	Wahr	Falsch	Begründung
1	Beim Traversieren eines kD-Baums müssen immer beide Kinder in Betracht gezogen werden.	☐	☑	vgl. Folie 103
2	Das Traversieren einer Hüllkörperhierarchie mit achsenparallelen Boxen (Bounding Volume Hierarchy, BVH) erfordert Mailboxing, um mehrfache Schnitttests mit einem Dreieck zu verhindern.	☐	☑	BVHs fügen Objekte nur in einen Kindknoten ein
3	Der Speicheraufwand einer BVH hängt logarithmisch von der Anzahl der Primitive ab.	☐	☑	Linear, siehe github.com/MartinThoma/KIT-Musterloesungen/issues/13
4	kD-Bäume sind eine Verallgemeinerung von BSP-Bäumen.	☐	☑	Es ist genau anders herum. kD-Bäume müssen Achsenparallele Trennebenen haben, BSP-Bäume jedoch nicht.
5	BSP-Bäume sind adaptiv und leiden nicht unter dem „Teapot in a Stadium“-Problem.	☑	☐	

Teilaufgabe 6d

#	Aussage	BVH	Octree	kD-Baum	Gitter
1	Die Datenstruktur partitioniert den Raum.	☐	☑	☑	☑
2	Der Aufwand für den Aufbau der Datenstruktur ist linear in der Anzahl der Primitive.	☐	☐	☐	☑
3	Eine effizientere Traversierung wird erreicht, wenn die Surface Area Heuristic bei der Konstruktion verwendet wird.[1]	☑	☐	☑	☐
4	Die Datenstruktur eignet sich am besten für Szenen, in denen die Geometrie gleichmäßig verteilt ist und kaum leere Zwischenräume vorhanden sind.	☐	☐	☐	☑

[1] vgl. `05_ Raumliche Datenstrukturen.pdf`, Folie 103

Aufgabe 7: Rasterisierung und OpenGL

Teilaufgabe 7a

#	Aussage	Wahr	Falsch	Begründung / Quelle
1	In der OpenGL-Pipeline wird View Frustum Clipping vor der perspektivischen Division durchgeführt.	☑	☐	Kapitel 6, Folie 45
2	Vertex-Shader können auf Texturen zugreifen.	☑	☐	Alle Shader können Texturen verwenden.
3	Bei Gouraud-Shading muss man die Normale im Fragment-Shader erneut normalisieren.	☐	☑	Normale wird nicht interpoliert
4	Gouraud-Shading mit dem Phong-Beleuchtungsmodell kann im Geometry-Shader implementiert werden.	☑	☐	TODO
5	Phong-Shading kann man alleine mit einem Vertex-Shader und einem Geometry-Shader implementieren; letzterer gibt dann die Farbe aus.	☐	☑	TODO
6	Bei beliebig feiner Tessellierung ist kein Unterschied zwischen Gouraud- und Phong-Shading erkennbar.	☑	☐	TODO
7	Selbst wenn der Tiefentest für ein Fragment fehlschlägt, kann der Stencil-Puffer verändert werden.	☑	☐	Kapitel 7, Teil 3, Folie 51
8	Instanziierung von Geometrie kann man sowohl mit dem Vertex- als auch dem Geometry-Shader durchführen.	☐	☑	nur mit dem Geometry-Shader

Teilaufgabe 7b

Warum zieht man das Tiefenpuffer-Verfahren (Z-Buffering) dem Sortieren von Dreiecken vor? Nennen Sie drei Gründe!

- Dreiecke können nicht sortierbar sein (wenn ein Dreieck ein anderes schneidet)
- Sortieren ist aufwendig, muss bei Veränderung der Blickrichtung jedes Mal neu durchgeführt werden
- Unterstützung durch Grafik-Hardware

Aufgabe 8: OpenGL-Primitive

(a) `GL_TRIANGLE_STRIP`: Ganz links ist (1), (2) ist rechts unten davon, (3) ist rechts oben von (1). Dann im Zick-Zack-Muster weiter.
Oder mit minimaler Anzahl Vertices: `GL_TRIANGLE_FAN` mit (1) beim “inneren“ Knick, (2) rechts-unterhalb davon, (3) rechts-oberhalb usw. bis (8) links davon.

(b) `GL_TRIANGLE_FAN`: Der mittlere Knoten ist (1), dann wird von ganz links gegen den Uhrzeigersinn nummeriert.

Aufgabe 9: OpenGL und Blending

Teilaufgabe 9a

Teilaufgabe 9a (I)

Die Reihenfolge ist wegen des Tiefenpuffers egal.	☐
Von hinten nach vorne.	☑
Von vorne nach hinten.	☐

Teilaufgabe 9a (II)

```
glBlendFunc(GL_SRC_ALPHA /*(2)*/, GL_ONE_MINUS_SRC_ALPHA /*(3)*/)
```

Teilaufgabe 9b

```
glBlendFunc(GL_ONE /*(1)*/, GL_ONE /*(1)*/)
```

Teilaufgabe 9c

```
glBlendFunc(GL_ZERO /*(0)*/, GL_ONE_MINUS_SRC_COLOR /*(7)*/)
```

Aufgabe 10: Bézier-Kurven und Bézier-Splines

Teilaufgabe 10a

- **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**

Teilaufgabe 10b

shader.vert

```
uniform mat4 matrixMVP; // Model-View-Projection-Matrix
in vec3 position; // Koordinaten des Eingabe-Vertex
uniform vec3 b[12]; // Array der Kontrollpunkte
uniform float time; // Zeitpunkt für die Animation in [0;3)

// bezier3(..) soll die Bezier-Kurve an der Stelle s
//     auswerten und das Resultat als vec3 zurückgeben.
//     Sie können die Bernstein-Polynome oder den
//     Algorithmus von de Casteljau verwenden.
vec3 bezier3(float s, // Parameter s in [0;1)
             const vec3 b0, const vec3 b1, // Kontrollpunkte b0, b1, b2, b3
             const vec3 b2, const vec3 b3) {
    // Fügen Sie Ihren Code hier ein.

    // Lösung mit Bernstein-Polynomen
    vec3 result = vec3(0.);
    result += b0 * (1. - s) * (1. - s) * (1. - u) * 1.;
    result += b1 * (1. - s) * (1. - s) * u * 3.;
    result += b2 * (1. - s) * s * s * 3.;
    result += b3 * s * s * s * 3.;
    return result;
}

vec3 bezier3(float s, // Parameter s in [0;1)
             const vec3 b0, const vec3 b1, // Kontrollpunkte b0, b1, b2, b3
             const vec3 b2, const vec3 b3) {
    // Algorithmus von de Casteljau
    vec3 b01 = mix(b0, b1, s);
    vec3 b11 = mix(b1, b2, s);
    vec3 b21 = mix(b2, b3, s);
    vec3 b02 = mix(b01, b11, s);
    vec3 b12 = mix(b11, b21, s);
    return mix(b02, b12, s);
}

// bezierspline3(..) soll die Auswertung des Bezier-Splines an der
//     Stelle t als vec3 zurückgeben.
//     Verwenden Sie dazu die Funktion bezier3(..)!
vec3 bezierspline3(float t) {
    // Fügen Sie Ihren Code hier ein.
    int i = int(t);
```

```
    float s = fract(t);
    return bezierspline3(s, b[3*i], b[3*i+1], b[3*i+2], b[3*i+3]);
}

void main() {
    vec3 offset = bezierspline3(time);
    vec3 newpos = position + offset;
    gl_Position = matrixMVP * vec4(newpos,1.0);
}
```

Aufgabe 11: Wasseroberfläche mit GLSL

Teilaufgabe 11a

shader.frag

```
vec3 determineIntersection(in vec3 P, in vec3 r, out int index)
{
    // Ermitteln Sie hier den Schnittpunkt mit der nächsten Gefäßfläche
    // und geben Sie ihn zurück. Zusätzlich muss 'index' auf den Index
    // der entsprechenden Seitenfläche gesetzt werden.

    bool intersected = false;
    float t_min;

    for (int i = 0; i < 5; i++) {
        float t;
        if (intersect(i, P, r, t) && t > 0. && (!intersected || t < t_min)) {
            t_min = t;
            index = i;
            intersected = true;
        }
    }

    return P + t_min * r;
}
```

Teilaufgabe 11b

shader.frag

```
vec2 determineTextureCoordinate(in vec3 S, in int index)
{
    vec2 UV;

    switch(index)
    {
        // Vervollständigen Sie die Fälle entsprechend der Aufgabenstellung
        case 0:
        case 1:
            UV = S.yz;
            break;
        case 2:
        case 3:
            UV = S.xy;
            break;
        case 4:
        case 5:
            UV = S.xz;
            break;
    }

    // Fügen Sie ggf. notwendige weitere Anweisungen hier ein
    UV = (UV + vec(1.0)) / 2.0;

    return UV;
}
```