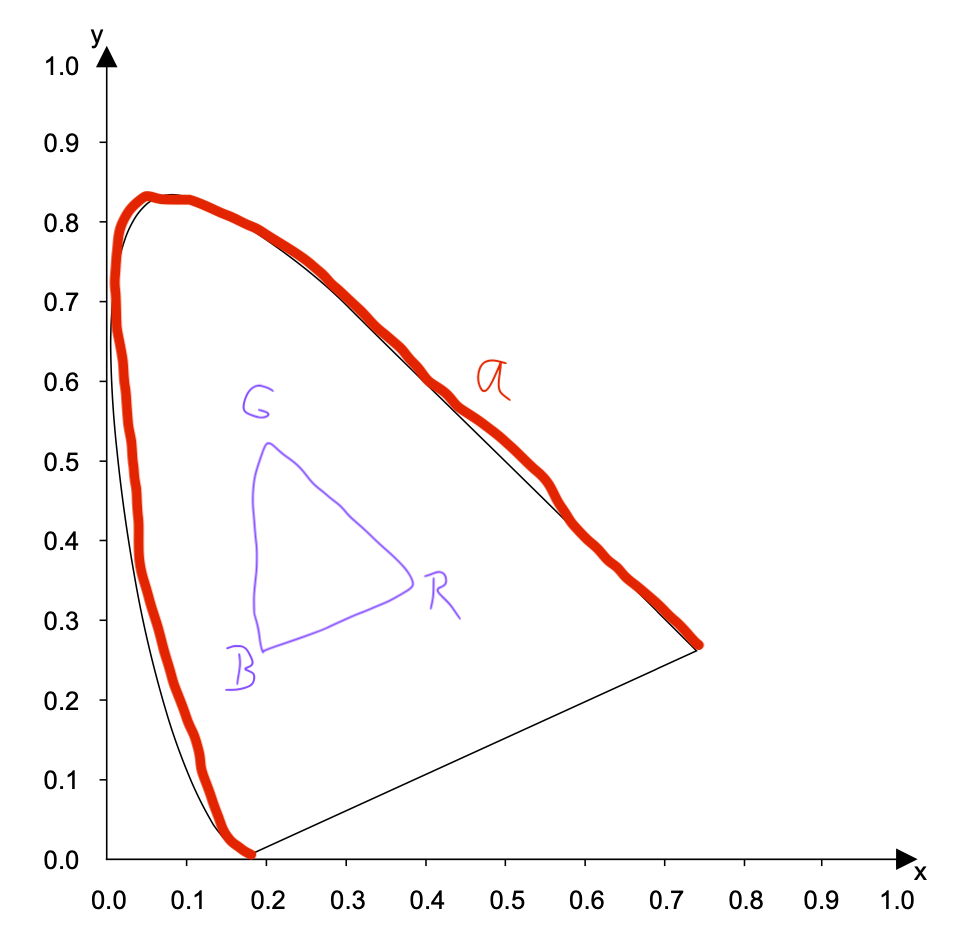
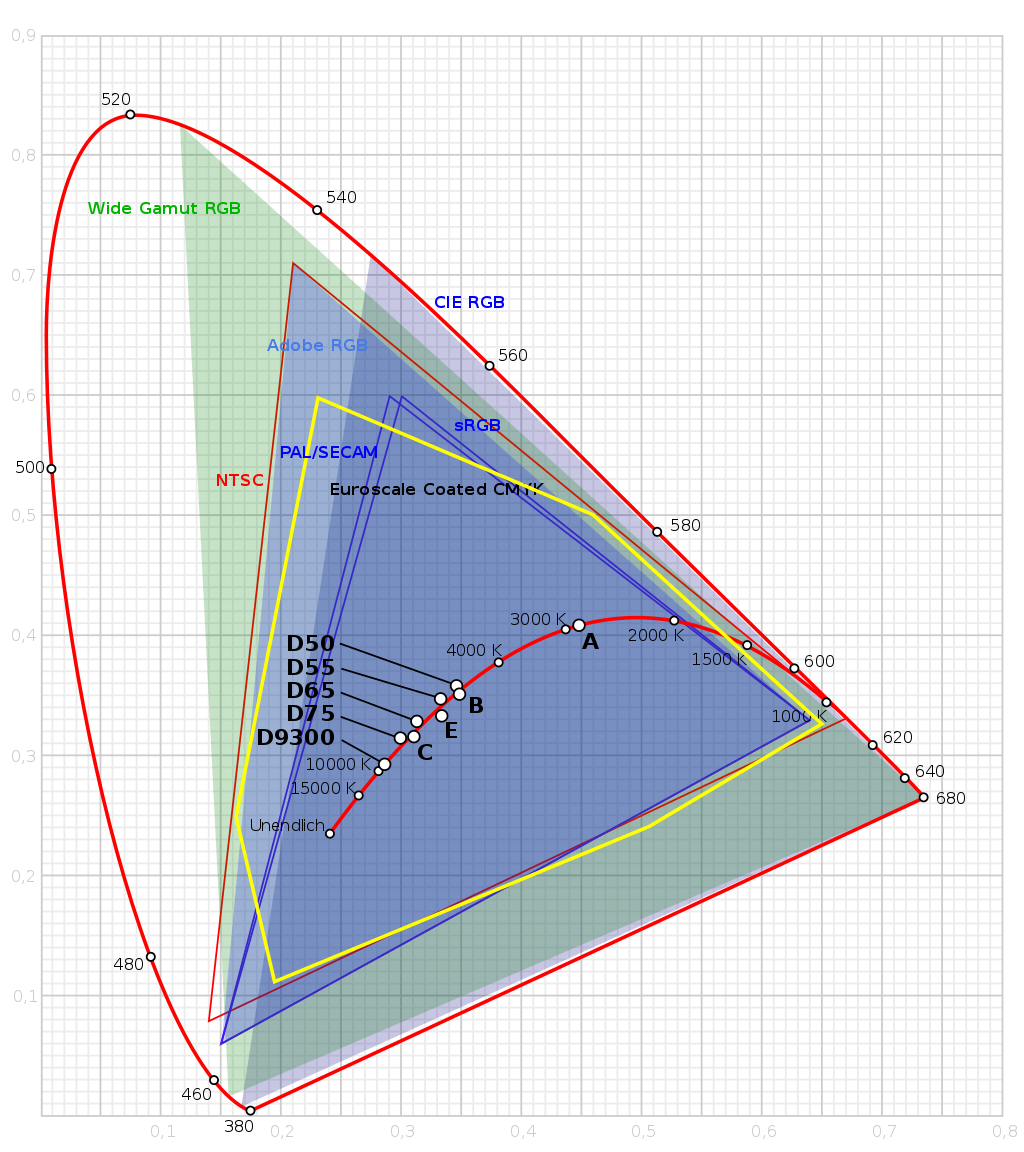
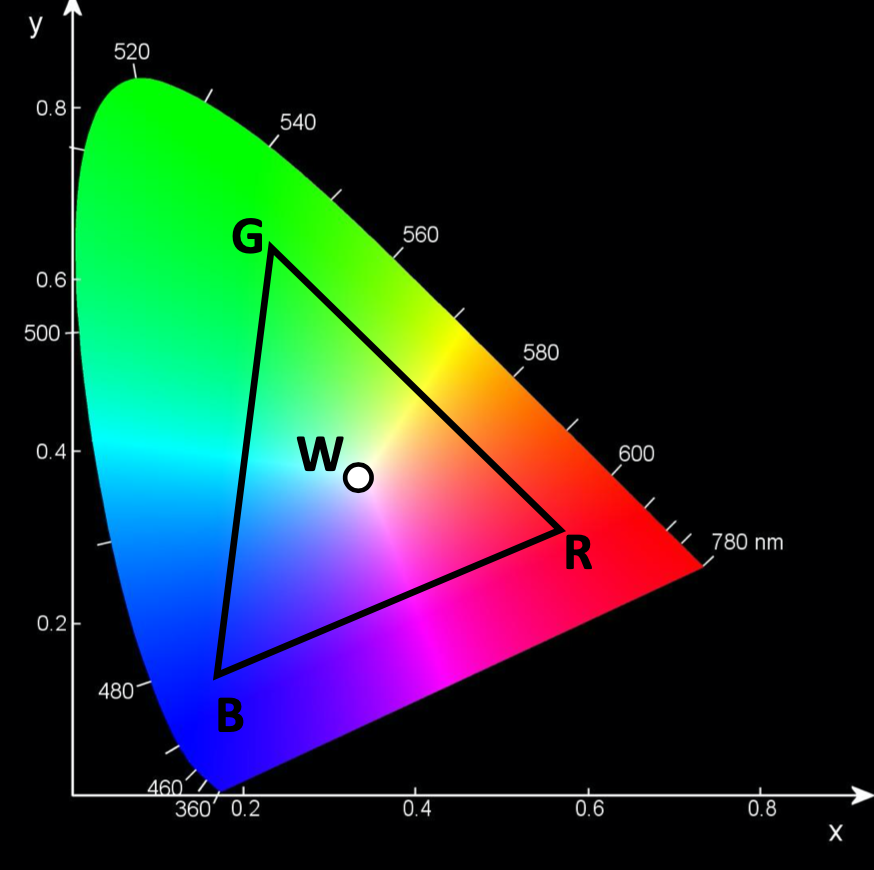
## Aufgabe 1

### + b)







### c)

Durch die Streifen ergibt sich wohl das arithmetische Mittel der Transferfunktionen von weiß und schwarz als Eindruck:

(angenommen die Intensität ist im Intervall [0, 1])

### d)

| Bei der subtraktiven Farbmischung werden Farben durch Multiplikation ihrer Spektren gemischt. | Wahr |
| --- | --- |
| Die subjektiv empfundene Stärke eines Sinneseindrucks ist proportional zur Intensität des physikalischen Reizes. | Falsch, proportional zur logarithmischen Intensität (Weber-Fechner-Gesetz) |
| Zur Beschreibung des Farbeindrucks von additiven Misch- farben sind nur die Tristimuluswerte und der Farbraum der Ausgangsfarben erforderlich. | Wahr, bei RGB besteht das drei-Tupel ja gerade aus den Tristimuluswerten (r, g, b) |
| Von einem Tristimuluswert kann eindeutig auf das zugrunde- liegende Spektrum geschlossen werden. | Falsch |

## Aufgabe 2

### a)

Nein:

### b)

Ja, solange der Winkel zwischen V und N nicht 45 Grad ist, denn dann wäre RL \* V = 0.

Oder Beispiel: . Dann ist und der spekulare Teil ist

### c)

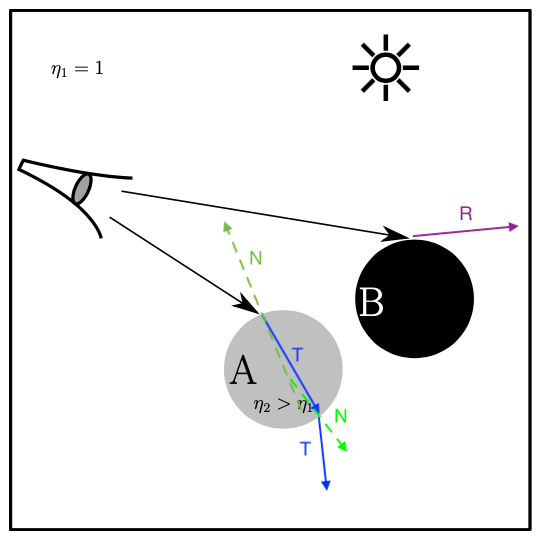
Da man bei Environment Maps auch Spiegelungen abbilden will, die nicht durch perfekte Reflexion entstanden sind, muss man die Environment Map vorfiltern, indem man eine gewichtete Summe über einen Winkelbereich vorberechnet (siehe Folie 04/117). Die Environment Map ist eine Kugel, auf deren Innenseite nur über die Richtung unabhängig von der Position zugegriffen wird.

## Aufgabe 3

### a)

Laut dem Pseudocode auf Folie 02/110 werden die Schattenstrahlen unabhängig von den Phong-Koeffizienten ausgesandt. Eigentlich sind sie aber schon überflüssig, da die Farbe eh 0 sein wird.

Hier eine geupdatete Version, auch mit Normalen zur Anschauung (nicht gefordert)



Diskussion zur Brechung: Es ist nur die **Normale** am Schnittpunkt relevant und der **Brechungsindex**

**Im Diagramm oben fehlen Schattenstrahlen von den beiden diffusen Flächen!**

### b)

Mat constants = …

vec3 trace(Ray ray, float w = 1.0f) {

vec3 color = vec3(0);

Isect i = intersect(ray);

if (!i.found)

return color;

color = w \* shade(i, ray, l);

// deckt auch constants.k\_r == 0 ab

if (w \* constants.k\_r >= epsilon) {

Ray reflection = reflect(ray, i.N, i.P);

color += trace(reflection, w \* constants.k\_r)

}

return color;

}

## Aufgabe 4

### a)

Spatial Median: BVH 1

Object Median: BVH 2

### b)

| **Strahl** | **BVH 1** | **BVH 2** |
| --- | --- | --- |
| **r1** | ACD2, | AB12 |
| **r2** | ACE34  (3 vor 4 da es das linke Kind ist) | AC34 |

### c)

BVH 1: Die Aufteilung dauert nur O(n log n)

BVH 2: Die Traversierung geht im Durchschnitt schneller, da der Baum flacher ist

## Aufgabe 5

### a)

p = c + λA\*(a - c) + λB\*(b - c)

Siehe Folie 04/29

### b)

a = 0.5

b = 1 - (5/8 - 0.25) / 0.5 = 0.25

t12 = (1-a)\*80 + a\*80 = 80

t34 = (1-a)\*0 + a\*0 = 0

CBL(P) = (1-b)\*t12 + b\*t34 = 0.75\*80 + 0.25\*0 = 60

### c)

Texelgröße(n) <= Pixel-Footprint < Texelgröße(n+1)

Siehe Folie 04/57

Stufe n-1 halbieren, Farbwert über 2x2 Texel mitteln (in der Aufgabe wird gefragt, wie die Stufe bestimmt wird, nicht wie darauf zugegriffen wird)

### d)

Texture-Bleeding kann auftreten. Siehe <https://gamedev.stackexchange.com/questions/46963/how-to-avoid-texture-bleeding-in-a-texture-atlas>

### e)

Sphere Mapping (Siehe Folie 04/106)

Problem: ungleichmäßige Abtastung

Cube Mapping (Siehe Folie 04/111)  
Problem: ungleichmäßige Abtastung in den Ecken

## 

## Aufgabe 6

Mat constants = …

vec3 main() {

vec3 winddir\_obj = (N \* vec4(winddir, 1.0)).xyz;

float offset = dot(in\_pos, winddir\_obj);

float amplitude = in\_flex \* sin(freq \* offset);

vec3 new\_pos = in\_pos + amplitude \* normalize(in\_nrm);

gl\_Position = VP \* M \* vec4(new\_pos, 1.0);

}

## Aufgabe 7

a)

b)

Tuv->xy = [u v p; 0 0 1]; (3x3 Matrix)

Su = [-1 0 0; 0 1 0; 0 0 1];

T = Tuv->xy \* Su \* inv(Tuv->xy);

Alternative (schöner):

Matrix die UV-Koordinaten in XY-Koordinaten umwandelt:

(3x3 Matrix)

Matrix die Punkte über die Y-Achse spiegelt

## **Aufgabe** 8

a)

Sortieren ist nicht notwendig;

// nicht notwendig wegen default: glDepthMask(GL\_TRUE);

glEnable(GL\_DEPTH\_TEST);

Von hinten nach vorne;

glDepthMask(GL\_FALSE);

glEnable(GL\_BLEND);

glBlendEquation(GL\_FUNC\_ADD);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

## Aufgabe 9

a)

Vertex-Shader, bevor die Primitive gebaut werden. Einfach aus der Textur auslesen und entsprechende Vertices verschieben.

b)

~~Geometry-Shader, neue Geometrie muss erzeugt werden.~~

Fragment-Shader. Begründung: Interpolation ist linear zwischen den Abschwächungswerten der Vertices, jedoch ist die Abschwächung exponentiell -> Daher kann das nur realistisch im Fragment-Shader berechnet werden.

c)

Normale wird in Vertex-Shader ausgelesen und vor dem Fragment-Shader interpoliert. Die Beleuchtung anhand interpolierten Normale wird im Fragment-Shader berechnet.

d)

mat3 M;

M[0] = T;

M[1] = N;

M[2] = B;

vec3 SNw = normalize(M \* SN);

//TODO: Lichtrichtung der direktionalen Lichtquelle?

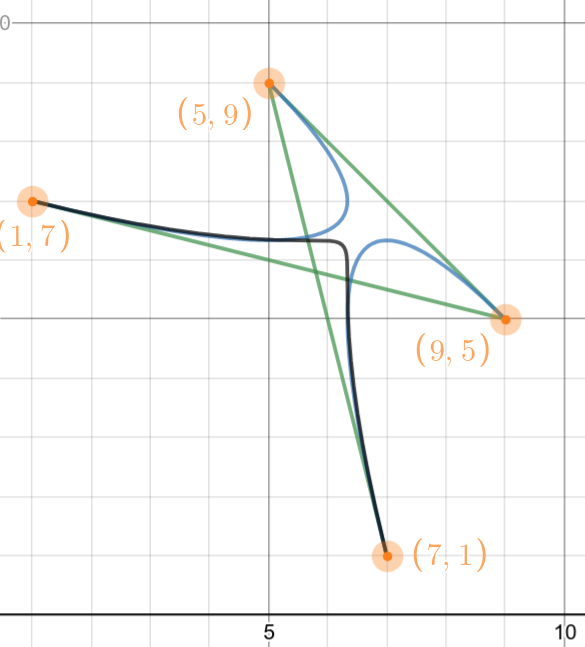
return dot(SNw, -L);  
  
Mein Vorschlag:  
Mat4 transform = mat4(vec4(T,0), vec4(N,0), vec4(B,0), vec4(P,1));  
Vec3 SNw = normalize(transform \* vec4(SN,0));  
return max(0.f, dot(SNw, -L));

//Keine Ahnung warum die einen vec3 wollen

## Aufgabe 10

a)

b)



c)

b3 - b2 = c1 - c0

Alternative?