## Aufgabe 1

a)

i) ~ Zwischen 380nm und 700nm

ii) eher rot

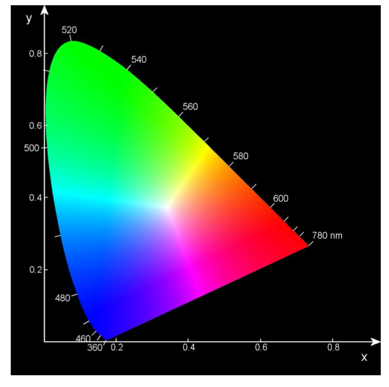
b)

0.5 ^ gamma = 0.25

Gamma = 2

c)

Pink, da keine Spektralfarbe. Das kann man am Chromatizitätsdiagramm sehen, denn die Spektralfarben sind auf der Kurve:



d)

logaritmisch

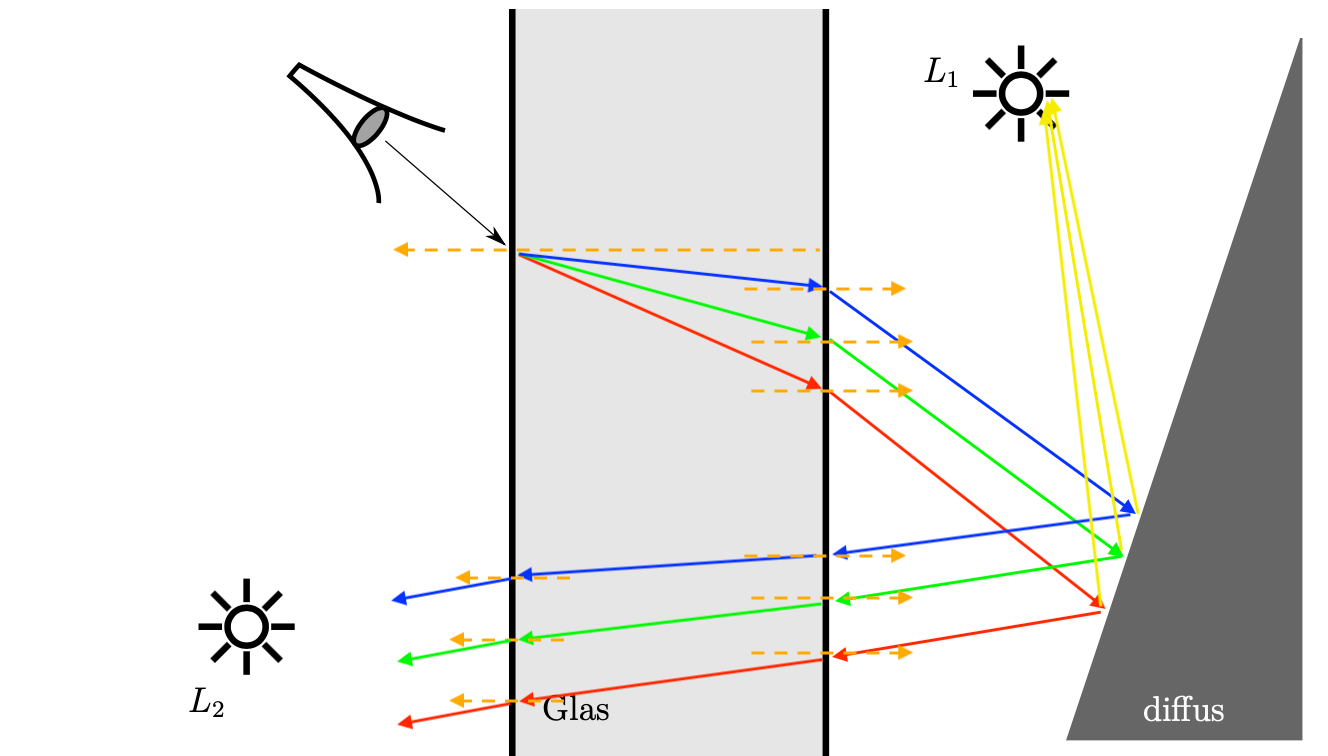
Siehe 01/134

## Aufgabe 2

### a)

i)

Schattenstrahlen in gelb, normalen in orange (nicht gefordert):



### 

ii)

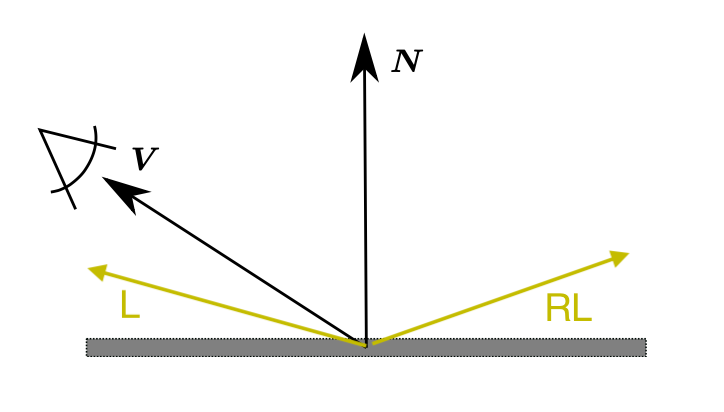
Es würde gleich aussehen, da beim diffusen Term der Winkel zwischen Normale und Lichtstrahl zählt. Dieser wäre auf dem diffusen Körper bei L2 wie bei L1.

Bei L2 bleibt nur der ambiente Term, da der Glaskörper den diffusen Körper verschattet. Wenn ein Schattenstrahl ausgesendet wird, findet bei Whitted keine Transmission statt.

### b)

i)

(der Winkel zwischen RL und V muss größer als 90 Grad sein, damit das Skalarprodukt negativ ist)



ii)

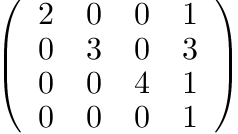
Der Abfall der Reflexion mit zunehmendem Winkel wird stärker.

Das Glanzlichter wird kleiner

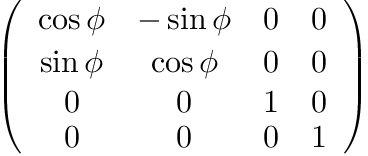
## Aufgabe 3

### a)

i)



ii)



### b)

Mat3 R;

Vec3 z = r;

//Vec3 up = vec3(0, 1, 0); // Das funktioniert nur, wenn der vektor r != (0,y,0) | y beliebig

// Das ist ja aber durch die Aufgabenstellung gegeben

//Vec3 x = normalize(cross(up, z));

Vec3 x = normalize(vec3(0,r.z,-r.y)); // x immer orthonormal zu z.

Vec3 y = normalize(cross(z, x));

R[0] = x;

R[1] = y;

R[2] = z;

return R \* rotate\_z(theta) \*transpose(R) \* p;

mat3 rotate\_z(float theta); // gibt eine 3x3-Matrix zurück,

// die um die Achse (0,0,1) rotiert

vec3 rotate(vec3 p, float theta, vec3 r) {

// siehe Folie 03/31

// bestimme (irgendein) orthonormales KoSys mit r als z-Achse

vec3 e(0, -r.z, r.y);

e = normalize(e);

vec3 f = normalize(cross(r, e));

// Basiswechsel

mat3 MI(e, f, r);

mat3 M = transpose(MI);

mat3 Rz = rotate\_z(theta);

return MI \* Rz \* M \* p;

}

## 

## Aufgabe 4

a)

Raum wird in Gruppen unterteilt sodass ein Strahl ein Primitiv schneiden kann nur wenn er erst den Hüllkörper der Gruppe schneidet.

Hierarchische Datenstrukturen nutzen die räumliche Lokalität zu Objekten um Strahlen, die weit von den Objekten entfernt sind, vor dem Schnitttest auszuschließen.

b)

B C D DE DE

Ob das E oder D eher kommt ist wahrscheinlich nicht bestimmt, außer die Primitive sind alphabetisch sortiert.

c)

i)

i = floor((ex - xmin)/w)

j = floor((ey - ymin)/h)

ii)

ex + tnextx \* dx = (i + 1)\*w + xmin;

Tnextx = ((i + 1)\*w + xmin - ex)/dx;

Tnexty = ((j + 1)\*h + ymin - e)/dy;

iii)

Delta\_x = w/dx;

Delta\_y = h/dy;

If tnextx < tnexty:

i += signx

Tnextx += delta\_x

Else

j += signy

Tnexty += delta\_y

d)

Sphere

Vorteil: einfach zu berechnen

Nachteil: gar nicht genau

AABB

Vorteil: auch einfach zu berechnen

Nachteil: besser als Sphere, aber trotzdem evtl. Platz-ineffizient

OBB:

Vorteil: genauer als AABB

Nachteil: PCA muss berechnet werden

e)

| Die Datenstruktur partitioniert Objekte. | BVH (Octree und kD-Baum nicht, da dort Raum unterteilt wird) |
| --- | --- |
| Der Aufwand für die Traversierung der Datenstruktur ist im Durchschnitt (Average Case) logarithmisch in der Anzahl der Primitive. | BVH, Octree, kD-BaumOctree |
| Die Datenstruktur eignet sich für Szenen mit großer Ausdehnung und ungleichmäßig verteilter Geometrie. | BVH, kD-Baum, Octree |
| Für den Aufbau der Datenstruktur kann die Surface Area Heuristic  (SAH) eingesetzt werden. | BVH, kD-Baum |

## Aufgabe 5

a)

i)

L1 = A(p, p2, p3)/A(p1, p2, p3)

L2 = A(p1, p, p3)/A(p1, p2, p3)

L3 = A(p1, p2, p)/A(p1, p2, p3)

ii) t = l1 \* t1 + l2 \* t2 + l3 \* t3

b)

Magnification: Pixel-Footprint zu klein im Texelraum -> bilineare Filterung

Minification: Pixel-Footprint zu groß im Texelraum -> Mipmap + trilineare Filterung

c)

Überflüssige Filterung in eine Richtung (verwaschene Details)

d)

Normal-Mapping: Texelwert enthält Normale; diffuse Komponente wird geändert (Normale) sowie spekulare Komponente (Reflexion wird auch angepasst)

Gloss-Map: Kontrolle der Stärke und Streuung der spekularen Reflexion, ks und n-Phong aus Textur

Ambient occlusion: Kontrolle der ambienten Beleuchtung (Umgebungslicht), ka und kd aus Textur

Displacement Map: Verschiebung der Oberfläche und Änderung der Normale

Shadow mapping: Berechnen von projektiven Schatten

Bump mapping: Veränderung der Normalen, keine Veränderung der Geometrie (ggü. Displacement)

## Aufgabe 6

a)

VS uniform:

Mat4 M // OKS -> WKS

Mat4 N // Normale OKS -> Normale WKS

Mat4 VP // View Projection

VS in:

vec3 pos // OKS

Vec3 normal // OKS

VS out / FS in

Vec3 wpos // WKS

Vec3 wnormal // WKS

FS uniform

Vec3 light\_pos

// material …

Kameraposition (für v)

FS out

Color

b)

Fragment shader

c)

In b

Die Primitive werden dadurch geändert + Farbauswertung wird für sinnlose Fragmente vermieden

d)

Nein? Im Vertex-Shader sind die interpolierte Normalen pro Primitiv noch nicht bekannt

Nein, im Vertex-Shader kann man keine Vertices entfernen.

## Aufgabe 7

a)

Alpha Blending

Final = src.rgba \* src.a + dst.rgba \* (1 - src.a)

(Alpha wird mit-geblendet, siehe [hier Gleichung 2](https://learnopengl.com/Advanced-OpenGL/Blending))

b)

(0.1 + 0.4 \* 0.1, 0.2 + 0.5 \* 0.6, 0.3 + 0.6 \* 0.4) = (0.14, 0.5, 0.54)

c)

Siehe Ü11

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

glBlendEquation(GL\_FUNC\_ADD);

MethodA

Return vec4(src.rgb \* src.a, src.a)

MethodB

Return vec4(src.rgb, 0.0)

MethodC

Return vec4(src.rgb \* src.a, 0.0)

d)

Ja wenn sie semitransparent sind.

Blending ist im allgemeinen nicht kommutativ

## Aufgabe 8

a)

return length(p - c) - r;

b)

float d1 = length(p - c0) - r0;

float d2 = length(p - c1) - r1;

return max(d1, -d2);

c)

float t = 0;

for (int i = 0; i < 4; ++i) {

t += (1.0 / pow(2.0, i)) \* noise(p \* pow(2.0, i));

}

return t;

d)

vec3 p\_plusx = vec3(p.x + eps, p.y, p.z);

vec3 p\_minusx = vec3(p.x - eps, p.y, p.z);

vec3 p\_plusy = vec3(p.x, p.y + eps, p.z);

vec3 p\_minusy = vec3(p.x, p.y - eps, p.z);

vec3 p\_plusz = vec3(p.x, p.y, p.z + eps);

vec3 p\_minusz = vec3(p.x - eps, p.y, p.z - eps);

float n = normalize(vec3(D(p\_plusx) - D(p\_minusx), D(p\_plusy) - D(p\_minusy), D(p\_plusz) - D(p\_minusz)));

e)

vec3 pos = p;

color = vec4(0.0);

for (int i = 0; i < MAX\_IT; ++i) {

float dist = D(pos);

if (dist < EPS) {

vec3 n = normalize(D\_n(pos));

color = shade(pos, n);

return;

}

pos += dist\*d;

}

discard;

Musterlösung vom Institut:

for (int i = 0; i < MAX\_IT; ++i) {

float t = D(p);

if (t < EPS) {

color = vec4(shade(p, D\_n(p)), 1.0);

return;

}

p += t\*d;

}

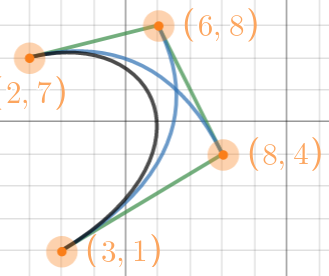
discard;

## Aufgabe 9

a)

Man kann einfach die Kontrollpunkte transformieren. (affine Invarianz)

b)



(8, 6) liegt also außerhalb des Kontrollpolygons

c)

b3 - b2 = c1 - c0

c3 - c2 = d1 - d0

