Ray Trace Präsentation

Programmiersprachen und Softwarentwicklung
Sommersemester 2018

Xiaoni Cai
2.Medieninformatik(118931)

Die Vorstellung des Projekts

Ein Ray Trace ist ein **Programm**, welches versucht, anhand einer Objektbeschreibung(z.B, Position, Farbe, Reflektionsgrad) **die Lichtstrahlen zu verfolgen** und die **Farben** auf einem Bild zu **bestimmen.**

Wir brauchen **alle Lichtstrahlen** der Lichtquellen **zu verfolgen**, die von den Objekten farbig gestreut oder reflektiert werden, bis sie auf der Bildebene ankommen.

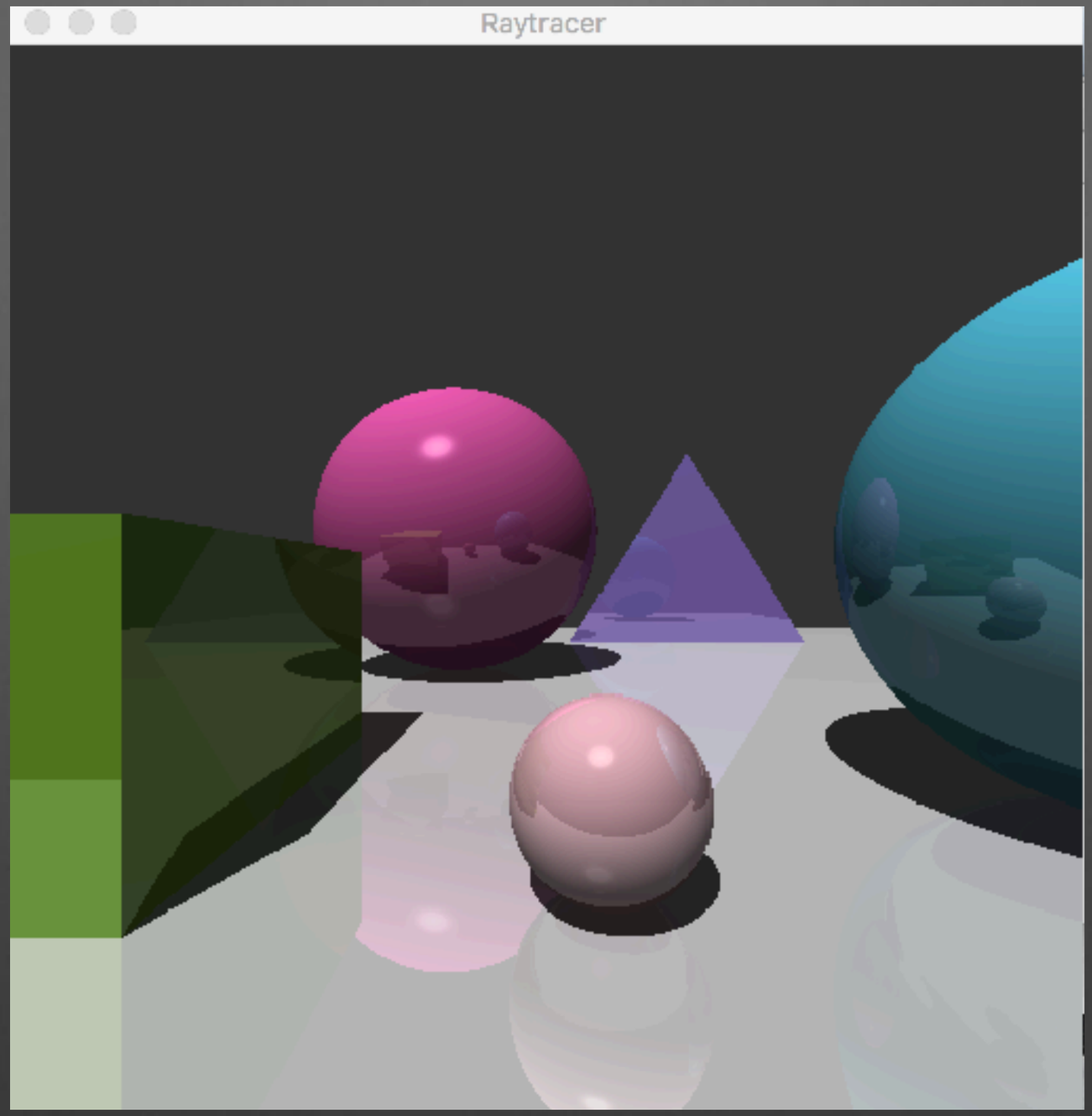
Die Dauer des Projekts: 26.07.2018-08.07.2018(Jeden Tag $\geq$10 Stunden)

Der Arbeitsort: Digital Bauhaus Labor 1. Etage

Die Arbeitsform: Gruppendiskussion, aber allein arbeiten und abgeben

Hilfsmaterial: hilfreiche Vorlesungsfolien von Adrian Kreskowski

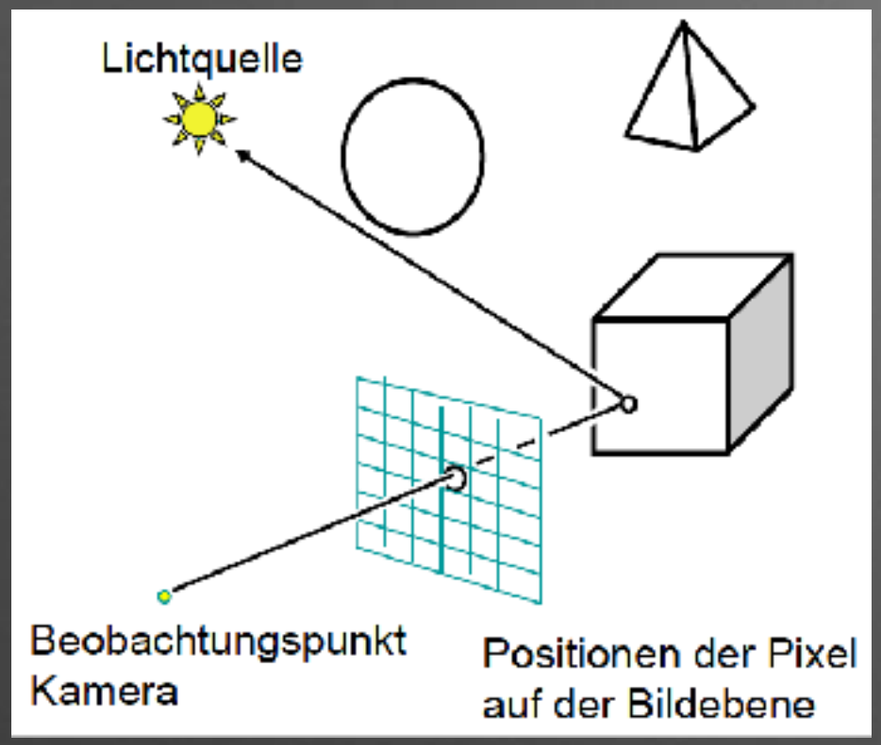
streuen散射

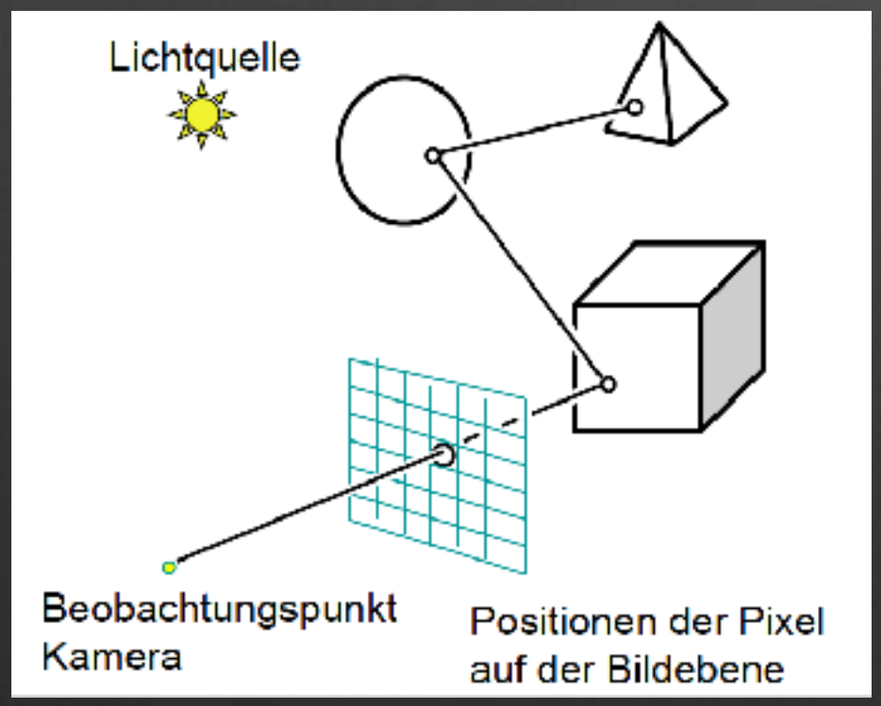


Das Endergebnis des Projekts

Github: https://github.com/caixiaoniweimar/programmiersprachen-raytracer

Die theoretische Grundlage des Projekts





1. Kamera wird mit einer Position festgelegt.
   -> Beobachtungspunkt

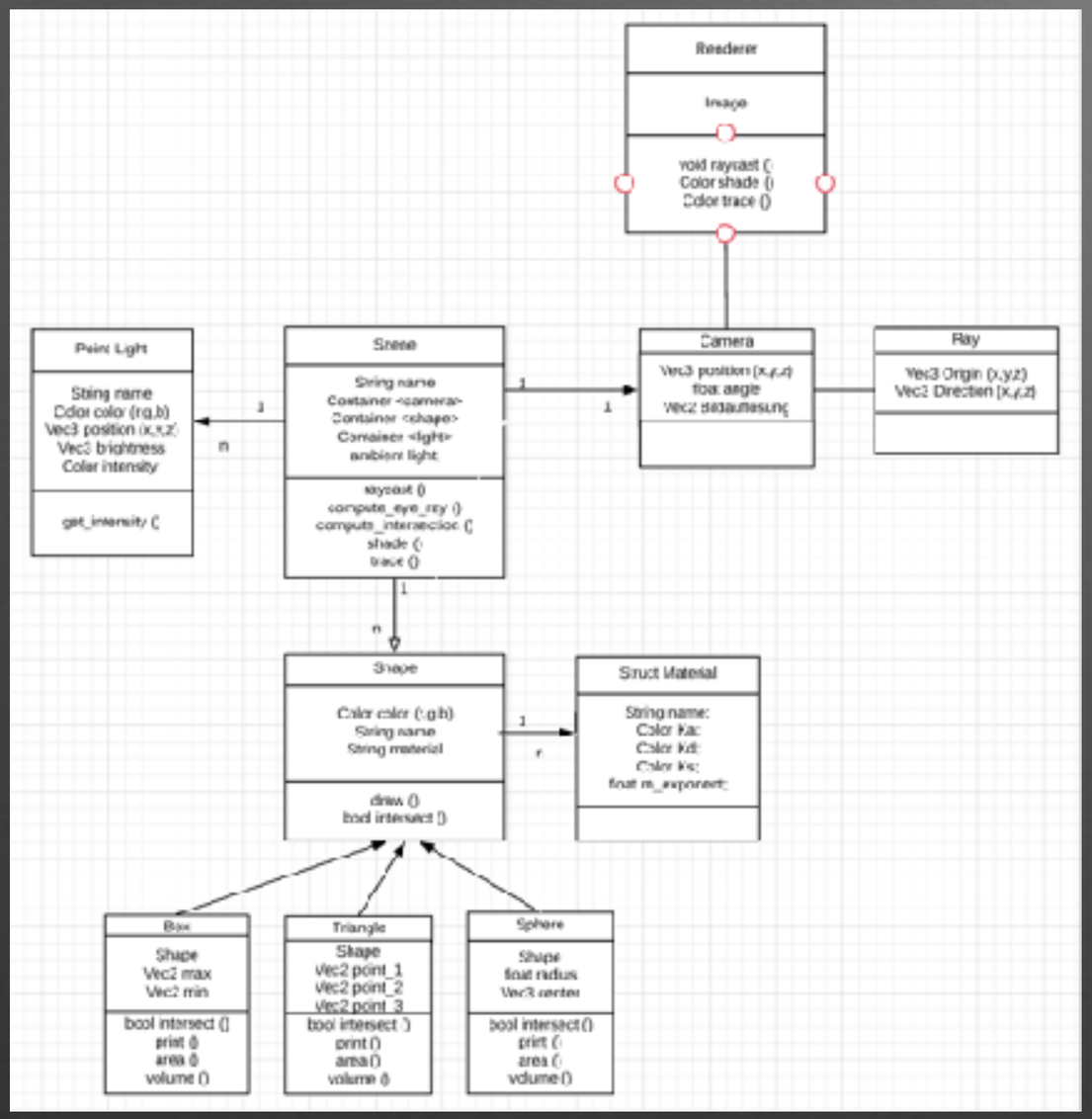
2. Ein Strahl wird vom Beobachtungspunkt durch jedes Pixel verfolgt.

3. Für alle Objekt im Szene rechnen wir Schnittpunkt des Strahls mit Objekt, bestimmen closest Schnittpunkt. (intersect( ))
   -> Oberflächenpunkt

4. Ein Strahl wird in Richtung der Lichtquelle geschossen. Wenn Objekt zwischen Lichtquelle und Oberflächenpunkt liegt, dann liegt der Punkt um Schatten, sonst nicht.
Das heißt, wir können Schattenbereich festlegen, und bekommen die sichtbare Oberfläche.

5. Der Strahl wird an einer Oberfläche gespiegelt. Dann wird ein neuer Strahl in Spiegelsrichtung geschossen.

UML Design



Ein hilfreiche Class: intersectionResult, um Intersect Ergebnisse zu speichern.

float distance; -> t;

bool hit;
-> bool intersection(Ray const& ray, float& t) const;

glm::vec3 position;
-> schnittpunktsposition, getpoint(),origin + direction*t;

glm::vec3 normal;
-> get_Normal()

Shape const* closest_shape=this;

Effizient!
intersection(), get_Normal() override für verschiedene Geometrie

Beleuchtungsberechnung

Für mehrere Lichtquellen, braucht man zu bestimmen, ob es noch Objekt zwischen Schnittpunkt und Lichtquellen gibt. bestimmen Schatten,Color -> raytrace(ray)

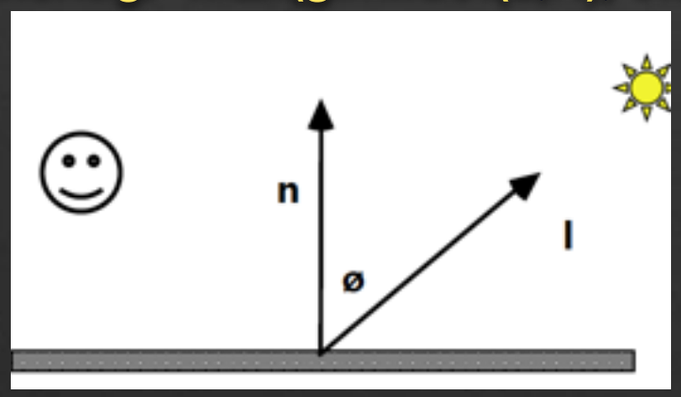
definiert Color ambiente{.2, .2, .2} als ein Attribute im Scene

1. ResultColor = I_a * K_a; I_a -> ambiente, K_a -> objekt.material.ka

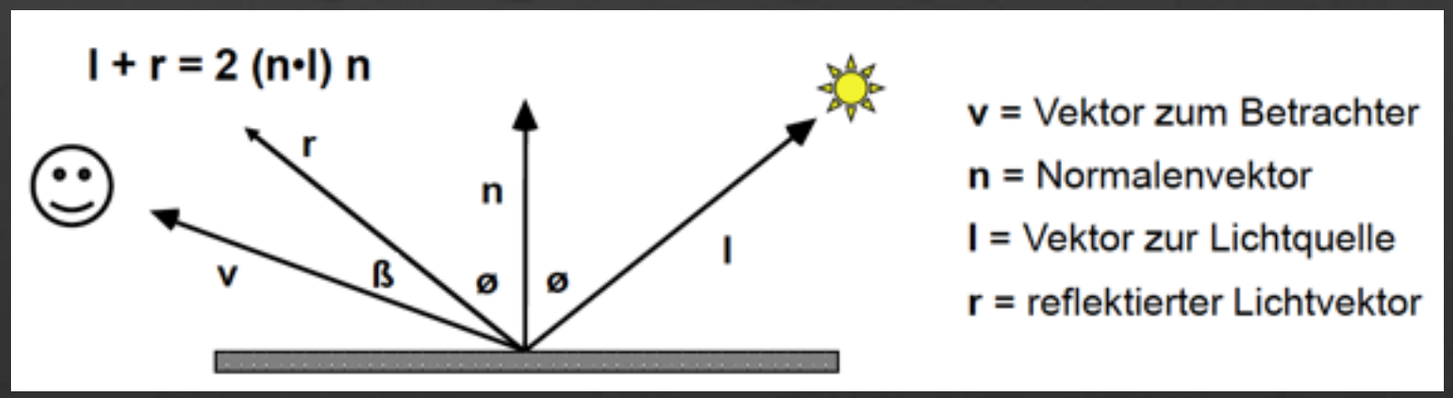
2. Ray light_ray{ schnittpunkt.position, scene.light[i].position - schnittpunkt.position };
glm::normalize(light_ray.direction)!!! Erzeugen Light_Ray
3. intersectionResult andere_objekt = istIntersect(light_ray);
4. lightray_distance = glm::distance(scene.light[i].position, light_ray.origin);
vergleichen mit Distance

5. if (andere_objekt == false && andere_objekt.distance > lightray_distance)
=> Es gibt keine Schatten, Lichtquelle sichtbar

6. ResultColor += I_p*K_d*glm::dot(L,N);
N -> schnittpunkt.normal
L -> light_ray.direction (schon normalisiert)
I_p -> scene.light[i].rechnen_intensitaet()
intensität: light.color*brightness
wichtig: *max(glm::dot(L,N), 0.0f)



7. ResultColor += I_p*K_s*pow(glm::dot(R,V),m);
R -> glm::normalize(2*LNdot*N-L)
V -> -ray.direction
auch glm::normalize(camera.eye-schnittpunkt.position)
wichtig: *max(glm::dot(R,V), 0.0f)



Transformation

1. drei Funktionen für Mat4 Berechnung

translation(glm::vec3 translation_vec):

$$\begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix}$$

```
glm::mat4 result(0.0f);
result[0] = glm::vec4{1,0,0,0};
result[1] = glm::vec4{0,1,0,0};
result[2] = glm::vec4{0,0,1,0};
result[3] = glm::vec4{translation_vec,1};
```

scale(glm::vec3 scale_vec):

$$\begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x s_x \\ p_y s_y \\ p_z s_z \\ 1 \end{bmatrix}$$

```
glm::mat4 result(0.0f);
result[0] = glm::vec4{scale_vec.x,0,0,0};
result[1] = glm::vec4{0,scale_vec.y,0,0};
result[2] = glm::vec4{0,0,scale_vec.z,0};
result[3][3] =1;
```

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
rotation(float winkel, glm::vec3 rotation_vec):
if(rotation_vec.x > 0)
result[0][0] =1;
result[1][1] = cos(winkel*M_PI/180);
result[1][2] = -sin(winkel*M_PI/180);
result[2][1] = sin(winkel*M_PI/180);
result[2][2] = cos(winkel*M_PI/180);
result[3][3] =1;
```

zwei Attribute von Shape:
world_transformation_,world_transformation_inv_
werden bei Transformation verändert.

2. Hilfsfunktion transformRay(glm::mat4 const& mat, Ray const& ray)
glm::vec4 a{ray.origin, 1};
glm::vec4 b{ray.direction, 0};
glm::vec3 origin1{mat*a};
glm::vec3 direction1{mat*b};
Ray new_ray{origin1,direction1};

Kameratransformation

u = **n** x **up** (**u** steht senkrecht auf **n** und **up**)

v = **u** x **n** (um sicherzustellen, dass **up** senkrecht zu **u** und **n** ist)

u, **v** und **n** sind zu normieren und sind damit die x-, y- und negative z-Achse des Kamerakoordinatenssystems. Damit ergibt sich die Kameratransformation als

$$C = \begin{bmatrix} u_x & v_x & -n_x & e_x \\ u_y & v_y & -n_y & e_y \\ u_z & v_z & -n_z & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

erzeugen_ray(x,y,width,height)

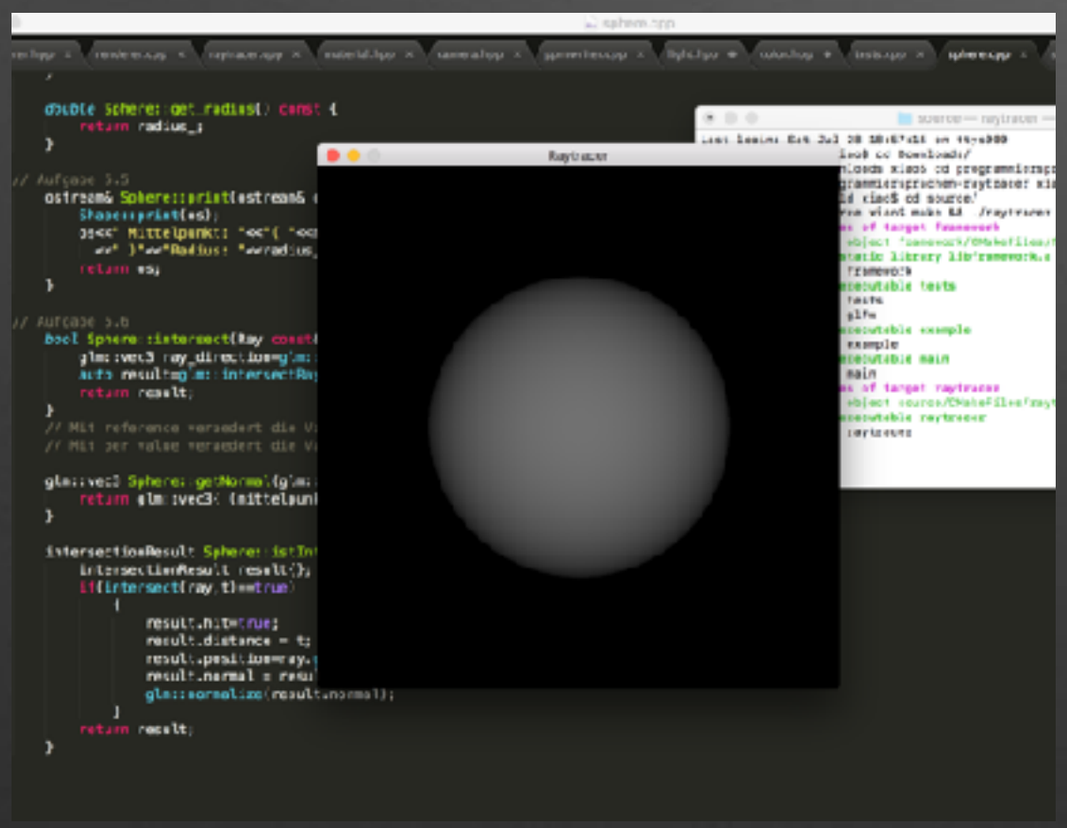
glm::vec3 u = glm::normalize(glm::cross(glm::normalize(dir), up));
glm::vec3 v = glm::normalize(glm::cross(u, glm::normalize(dir)));
transformatrix[0] = glm::vec4(u,0,0);
transformatrix[1] = glm::vec4(v,0,0);
transformatrix[2] = glm::vec4(glm::normalize(dir)*-1.0f, 0.0);
transformatrix[3] = glm::vec4(eye,1);
transformRay(mat,ray);

Projektablauf

26.07.2018-27.07.2018

durchlesen die Folien und die Aufgaben, Am Anfang hatte ich noch keine Idee, wie ich das Projekt anfangen kann. Der Umfang des Problems ist zu groß und es gibt keine spezifischen Probleme. Ich musste einen Plan machen, um das Projekt Schritte für Schritt zu erledigen. Ich hatte einen Termin mit Adrian gemacht und die Hilfe bekommt.

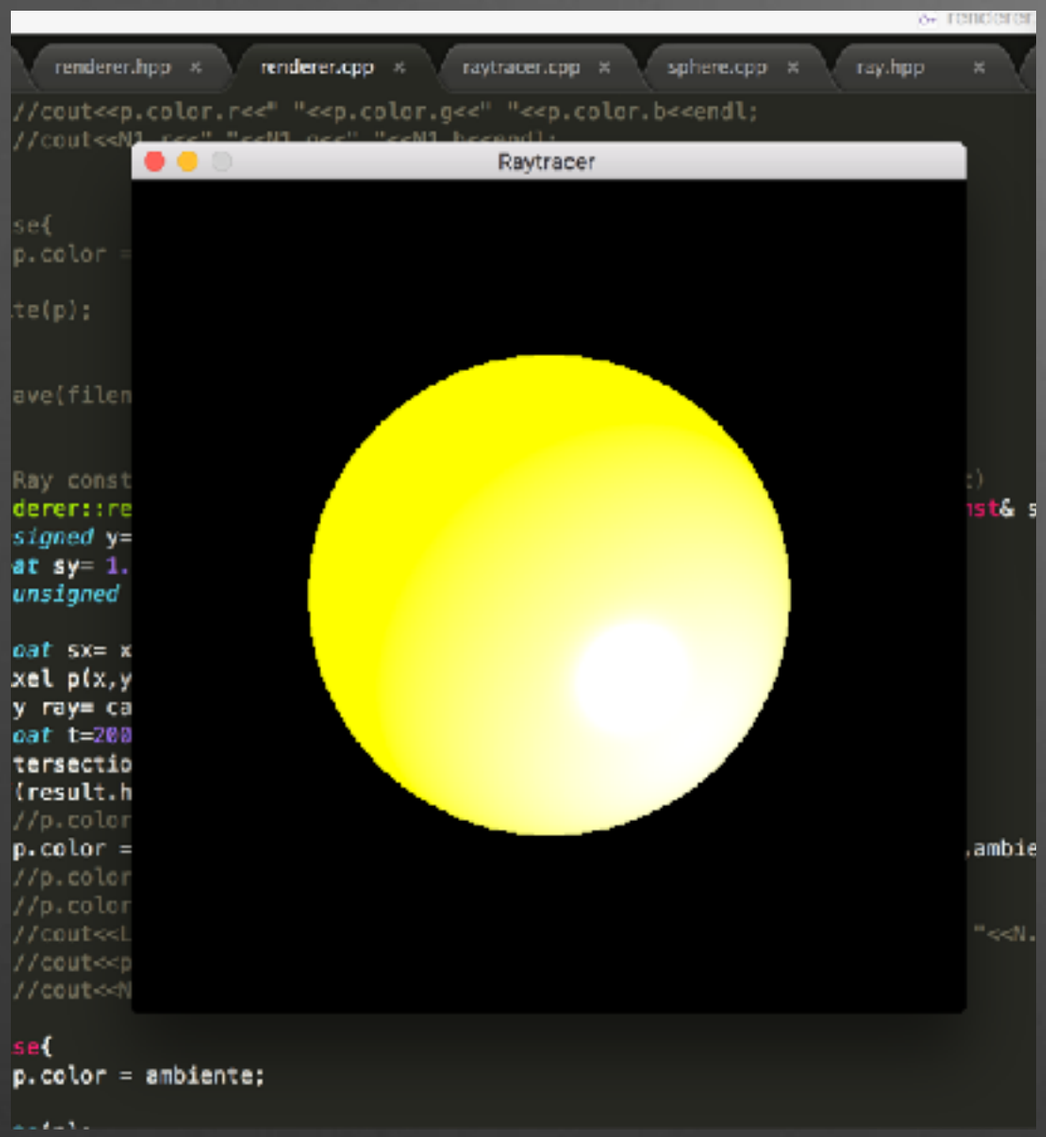
28.07.2018



verstehen die Beispielsmethode render( )
->checkerboard.ppm

benutzten ohne Lichtquelle, ohne Lesen SDF Datei, Alle Objekt wird definiert und dann als Übergabeparameter
Es gibt nur Camera, um Ray zu erzeugen.
Die Methode mit depth wird eine Sphere wie Links erzeugt.

30.07.2018



Hinzufügen Methode um Camera_Ray zu erzeugen und nur ein Light und ein Sphere im Scene. Hinzufügen Methode für Beleuchtungsrechnung. Aber leider waren sie noch nicht so richtig.

Es war noch schlimmer, entspracht Color der Sphere nicht mit dem Definition. (Sphere sollte als Rot wie Definition gezeigt sein, nicht gelb) Leider konnte ich den Grund nicht finden.

31.07.2018-02.08.2018

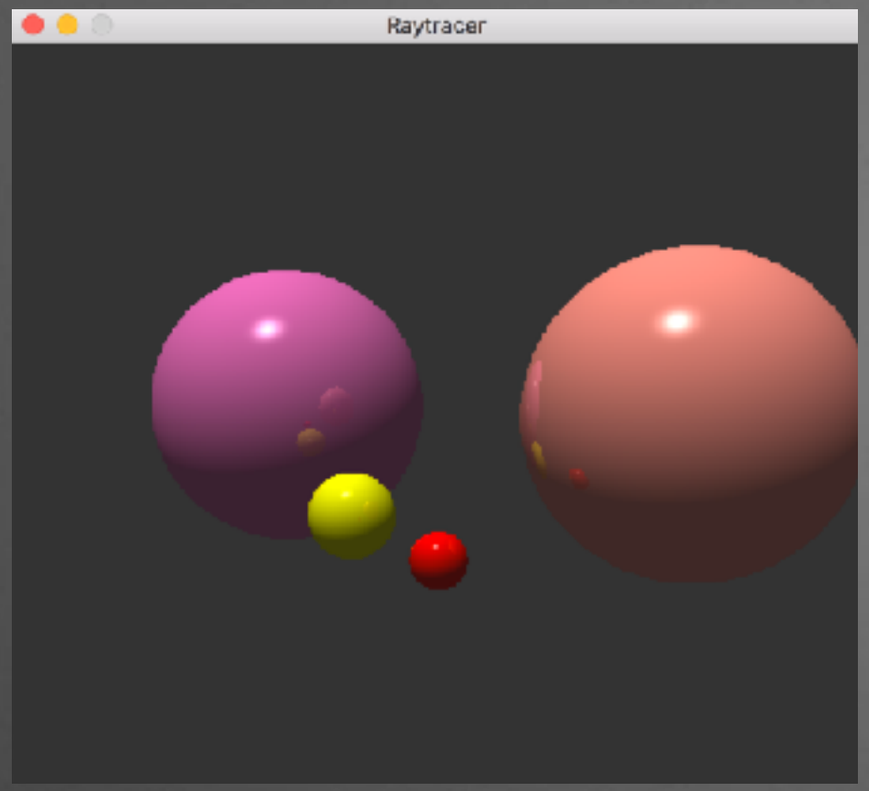
Egal wie ich die Farbe des Objekts definierte, das Ergebnis zeigte nur Gelb an. Ich war sehr niedergeschlagen.
Ich fing an, Ray_Erzeugen Methode und Render Methode zu korrigieren. Ich denke, diese beiden Teile sind die wichtigste Grundlage. Aber am Ende war es schlimmer als in den vorherigen Tagen. Es zeigt nichts.

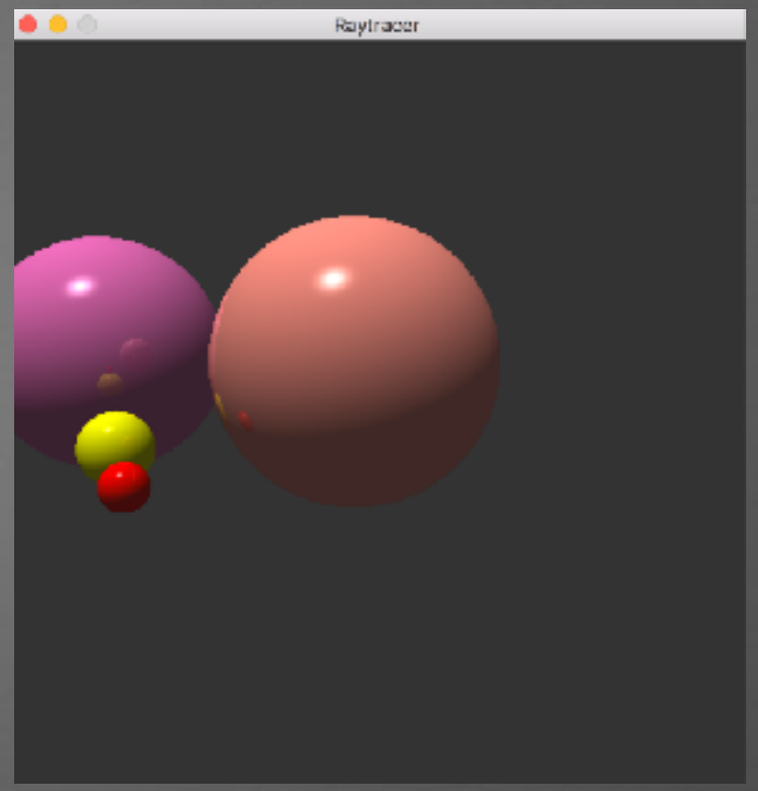
Deshalb habe ich mit anderen Methoden angefangen, die ich leichter ignorieren. Ich fing ganz von vorne an.
Ich wende mich an Theresa um Hilfe. Wir haben den Code verglichen.

Am Ende habe ich endlich einen kleinen Fehler im Color. hpp gefunden. Der Fehler ist so klein, aber dieser Fehler hat einen großen Einfluss auf die Berechnung der Farbe.
Ich habe auch viel Zeit für meine Nachlässigkeit ausgegeben.

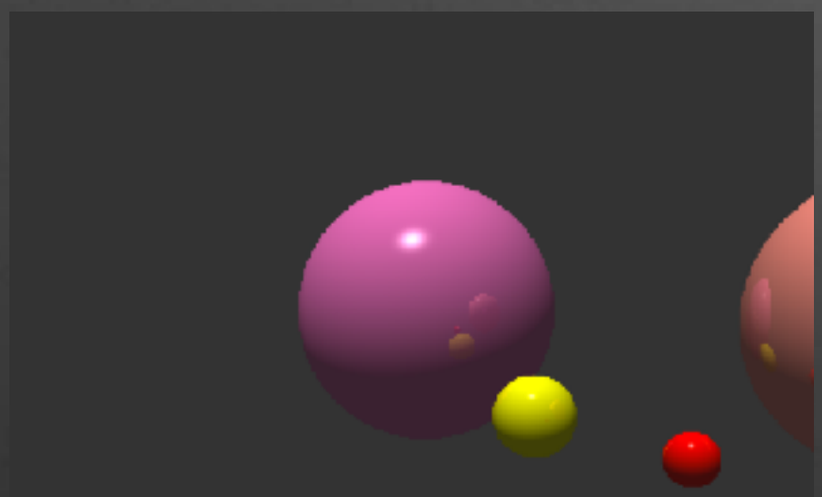
Fehler: Overloading/Überladung operator /= und operator *= haben die Position umgekehrt.

Nachdem ich den Fehler geändert habe, funktionierte es.





03.08.2018

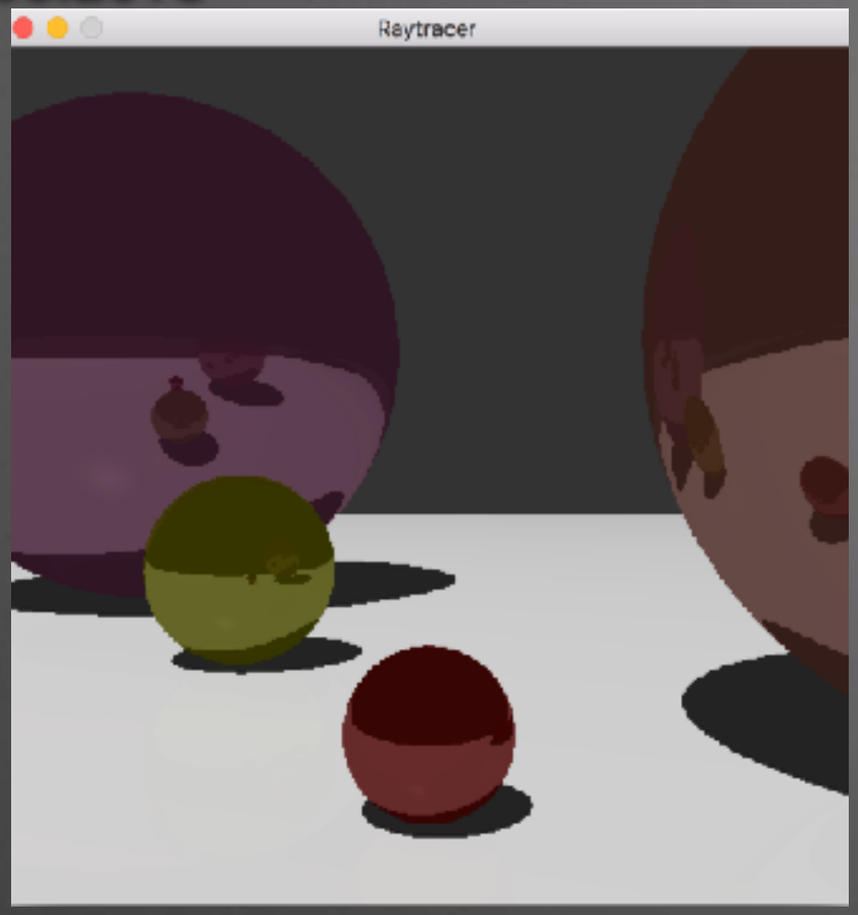


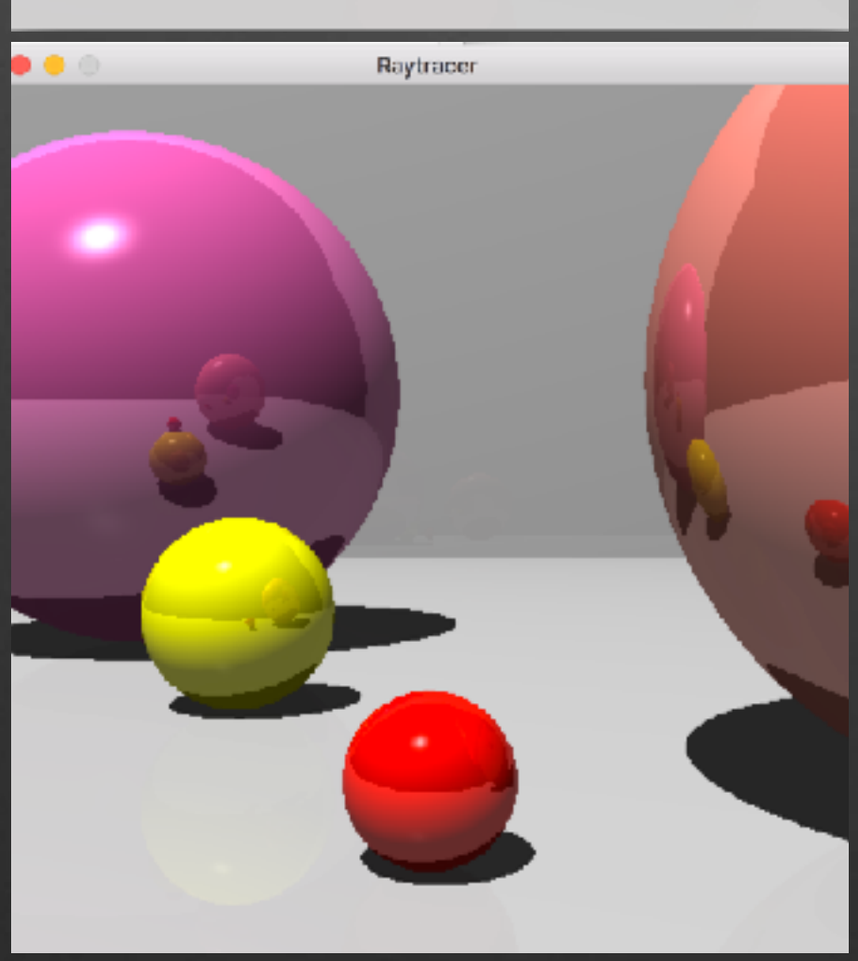
probieren mehrere Objekte im Scene hinzuzufügen, aber nur Sphere. Wenn es viele Objekt im Scene gibt, dann sollen wir eine Methode schreiben, um zu bestimmen, welches Intersectsobjekt am nächsten ist.

hinzufügen mehrere Attribute für Camera Class, name, fov-x, eye, dir{0,0,-1}, up{0,1,0}

beobachten die Unterschiede/Änderungen, wenn die Position des Camera verändert wird. (x)

04.08.2018





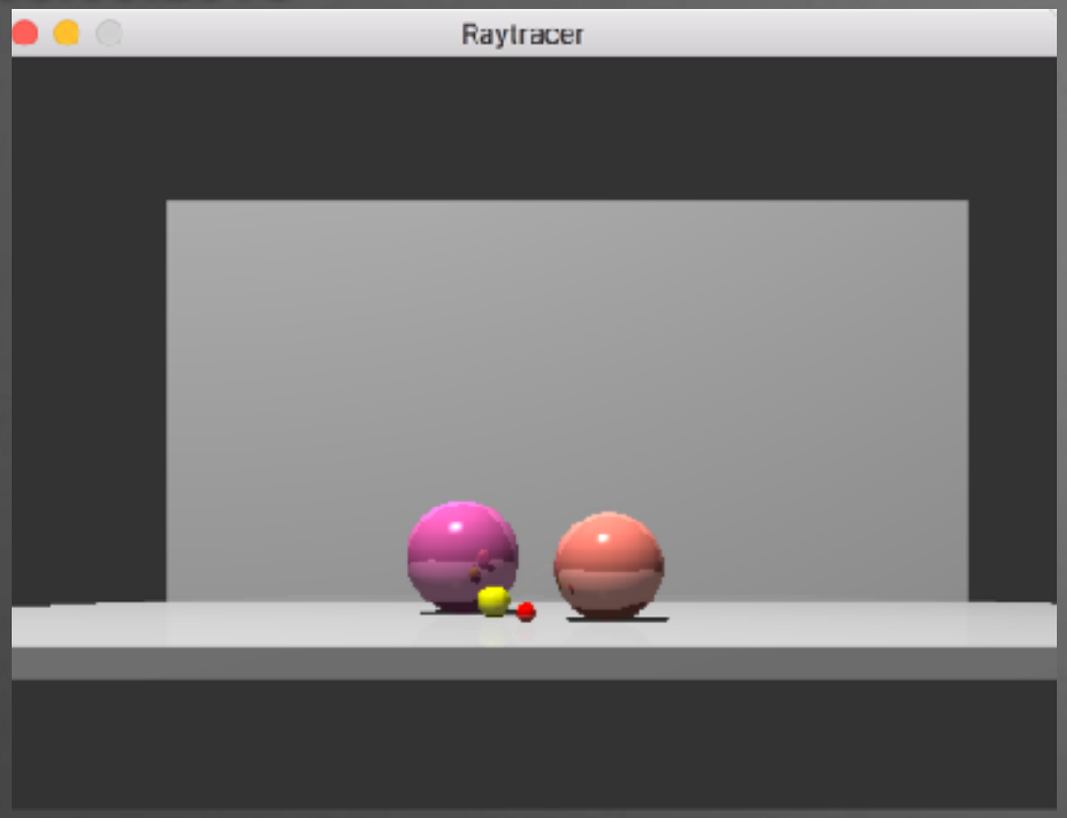
Hinzufügen ein Box Objekt als Boden
Dann werden das Schatten und die Reflektion gezeigt. Das Problem ist auch entstanden.
Die Objekte liegen nicht im Box sondern nur auf dem Box. Aber die Color verändert ganz anders, dunkler.
Die Objekte haben seine Helligkeit verloren.

Dann habe ich meinen Fehler wiedergefunden. Ich habe manchmal vergessen, die Richtung zu normalisieren. Der Unterschied ist gross.

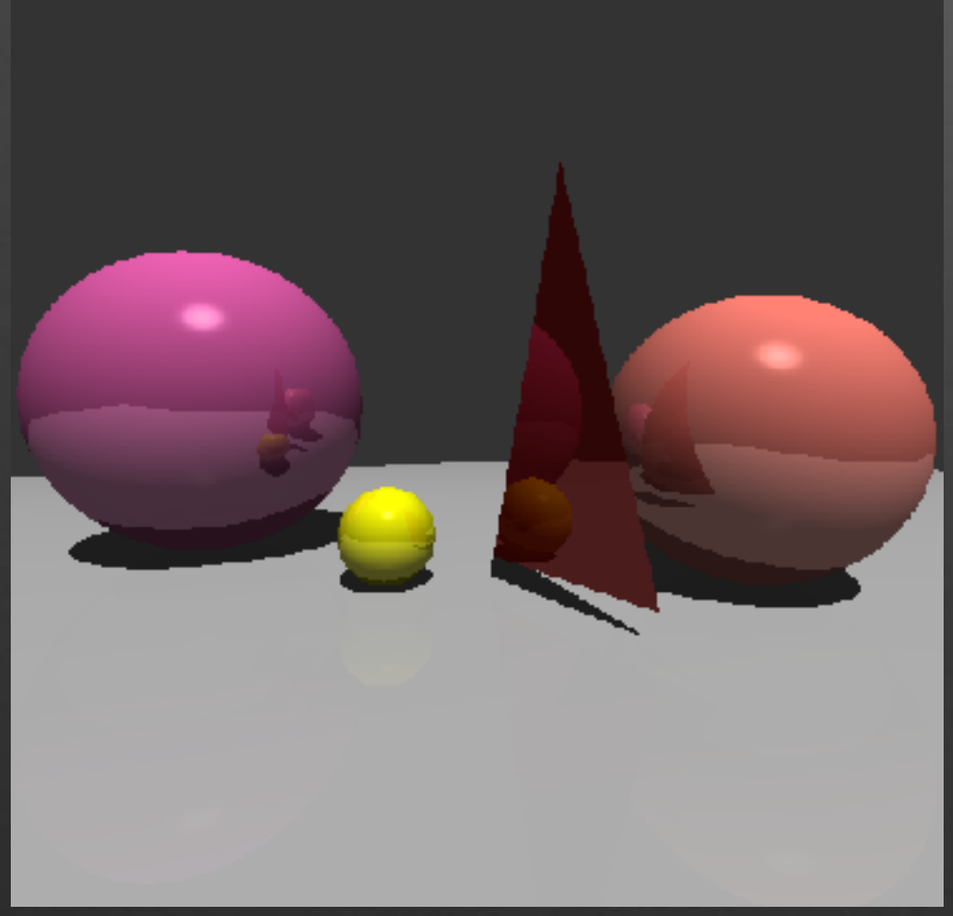
Korrektur:
schreiben ein Struktur für Ray
garantieren, ray.direction wird jedes Mal normalisiert.

```
Ray(glm::vec3 const& origin, glm::vec3 const& direction):
    origin{origin},
    direction{ glm::normalize(direction) }
```

05.08.2018



Add noch ein Box als Wand
verändern die Position des Camera(z)

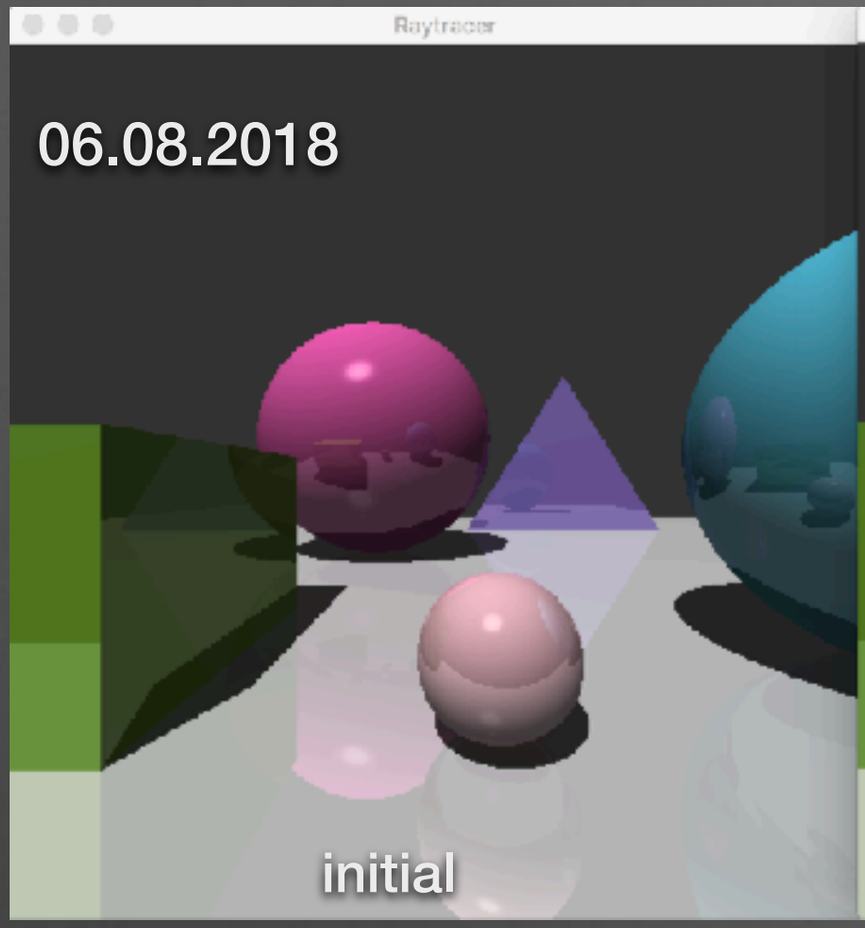


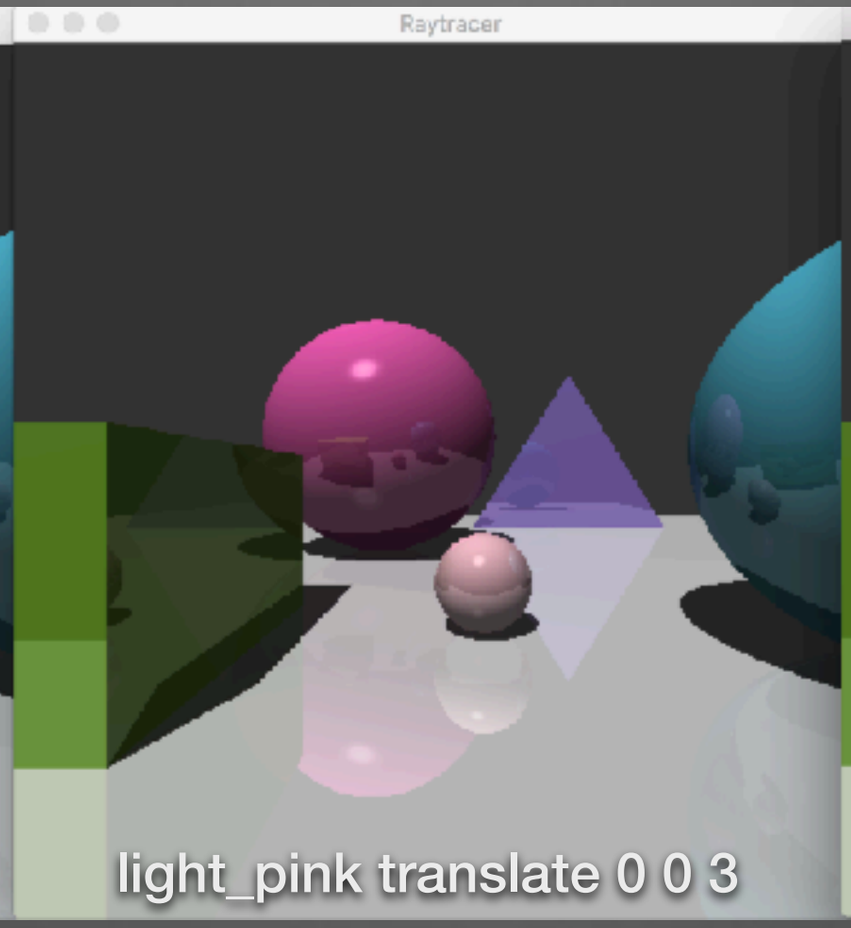
hinzufügen neue Class Dreieck, erweitern die Shape-Hierarchie um Dreieck,
override Methoden: bool intersect( )&get_Normal( )

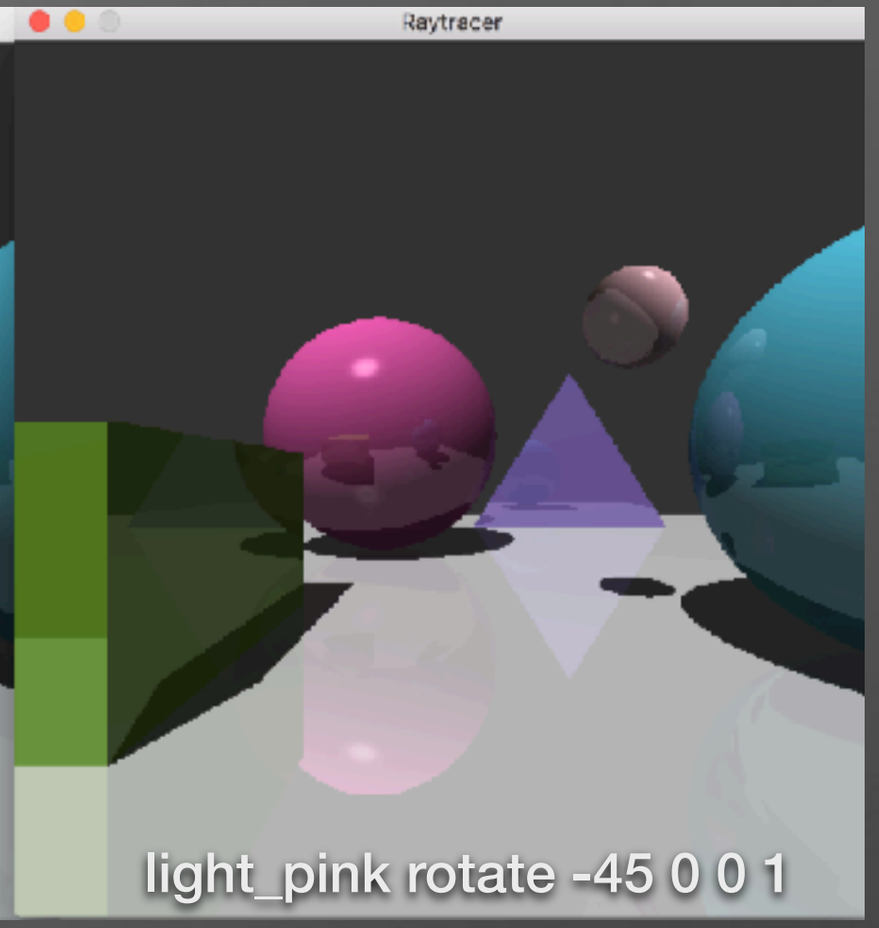
bool intersect( )
-> glm::intersectRayTriangle(ray.origin,
ray.direction, punkt1, punkt2, punkt3, schnittpunkt)

glm::vec3 get_Normal( )
-> normal = glm::cross(punkt1-punkt2, punkt1- punkt3);
glm::normalize(normal)

Add ein Dreieck im Scene







verändern Color, die Positionen und die Größe der Objekte(Sphere, Dreieck, Box)
finden die beste Position(eye_) und Öffnungswinkel(fov_x) des Kamera,
wenn es nötig ist, kann die Richtung(dir_) des Kamera auch verändert werden.

Transformation: Translation, Skalierung, Rotation laut SDF Datei Für Objekt und Kamera

Tone_mapping: schreiben eine Methode, um zu überprüfen, ob color_wert >1.0f ist.

```
float max_wert = max( color.r, max(color.g, color.b) );
if(max_wert > 1.0f){
    return (color/max_wert);
}else return color;
```

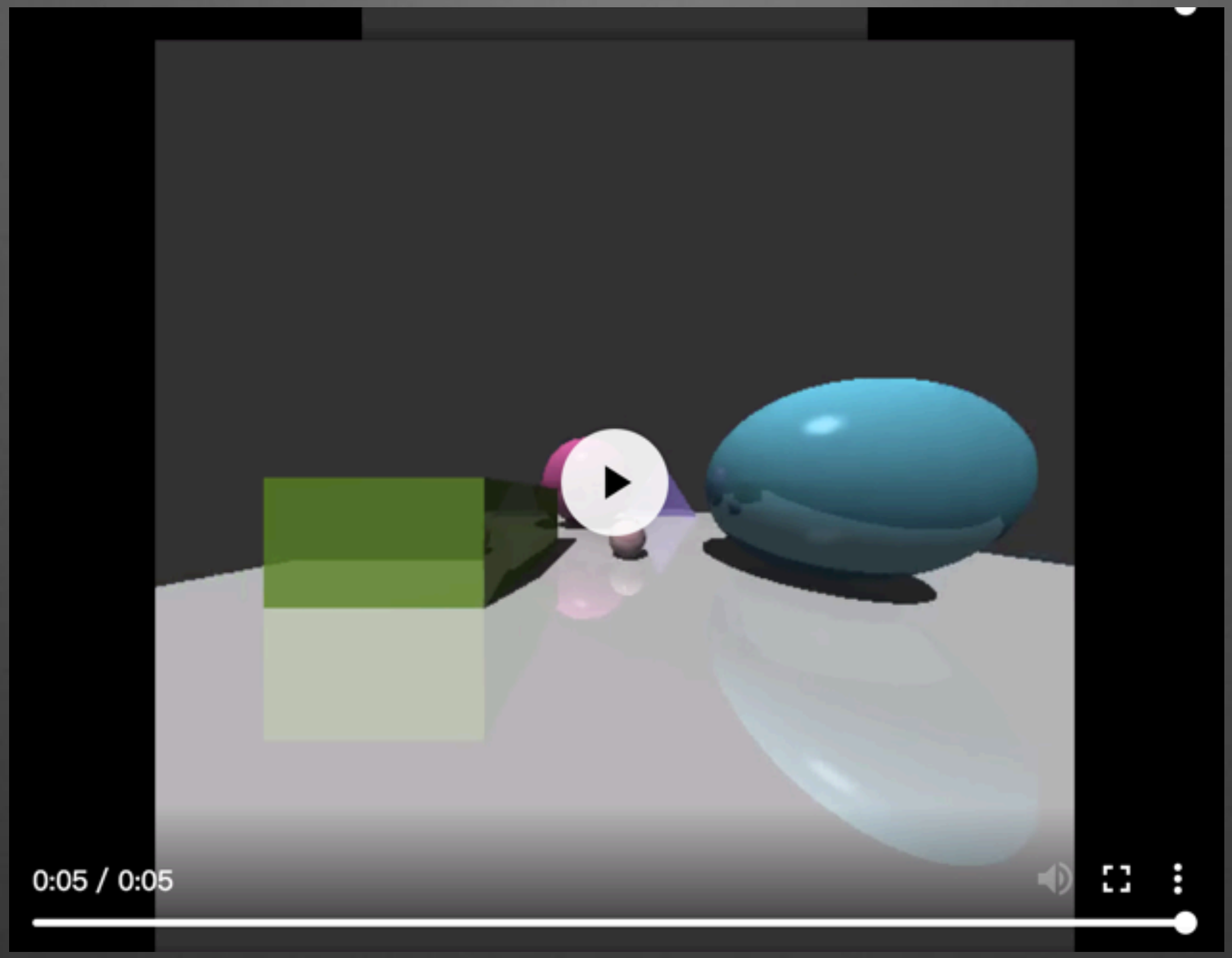
Animation: erzeugen min. 120 Fotos benannt img000.ppm - img120.ppm mit 120 SDF Daten,
verändern die erzeugte Datei Name und fov_x (von 20 bis 140)
ffmpeg -r 24 -i img%03d.ppm animation.mp4

Probleme, die nicht gelöst wurden:

- Anti-Aliasing
- Erweitern die Snape- Hierarchie um die Primitive Kegel und Zylinder
- Verständnis und Durchführung der Refraktion

Diese Probleme werden auf jeden Fall noch weiter versucht und gelöst……

Animation



https://www.uni-weimar.de/~sili4182/programmiersprachen/animation.html

Vielen Dank für Ihre Aufmerksamkeit!

recht x(+) unter z(+) up y(+)