DHBW Stuttgart
Duale Hochschule
Baden-Württemberg

Springende Sphären - Raytracer

Lea Soffel: 4962704
Matthias Hartmann: 6835407
Kevin Gavagan: 2240332
Lukas Benner: 3277496

Inhaltsverzeichnis

1 Einführung 2
1.1 Vorab 2
1.1.1 Shadertoy 2
1.1.2 Features 3
1.2 Pixel-Shader 3
1.3 Raytracing im Shader 4
2 Szenenzusammenstellung 5
3 Strahlenwurf / Ray-Casting 7
3.1 Projektionsebene 7
3.2 Mathematische Beschreibung eines Lichtstrahls 8
3.3 Field of view (FOV) 8
3.4 Drehung 11
3.5 Zusammenspiel 12
4 Sphärenkollision 13
5 Multiple Kollisionen 18
6 Berechnung des Lichts 19
6.1 Diffuse Lichtintensität 21
6.2 Spiegelnde Lichtintensität 22
7 Schattenwurf 23
8 Reflexion 25
9 Antialiasing 28
10 Sprunganimation 31
11 Literaturverzeichnis 33
12 Abbildungsverzeichnis 34
13 Formelverzeichnis 35

1 Einführung

Für das Abschlussprojekt im Modul Computergrafik wurde neben einem Raytracer in Python auch ein Raytracer als Shader geschrieben. Die Idee hierzu folgte aus der Beobachtung, dass die Perfomance des Python Raytracers weit entfernt von einer Echtzeitanwendung liegt. Je nach Auflösung und Menge der ausgesendeten Strahlen, sowie der Anzahl an Reflektionen, dauerte das Rendern eines einzelnen Bildes mehrere Minuten oder sogar Stunden.

Dadurch kam die Frage auf, ob und wie ein Raytracer als Shader umgesetzt werden kann, wie viel performanter dieser sein kann und was mit der gewonnenen Leistung getan werden kann.

Diese Arbeit befasst sich mit der Dokumentation des Raytracers als Shader, welcher als Fragment- bzw. Pixel-Shader umgesetzt ist. Dazu werden in den folgenden Abschnitten die einzelnen Bestandteile des Codes mathematisch eingeführt, näher erläutert und erklärt.

1.1 Vorab

1.1.1 Shadertoy

Der Pixel-Shader, so wie er in der index.html zu sehen ist, ist auch auf Shadertoy[1] zu sehen. Shadertoy bietet eine Plattform für die Veröffentlichung von Pixel-Shader an. Dieser wurde dort von Grund auf selbst programmiert.

Um ihn ebenfalls abseits Shadertoy ausführen zu können, wurde eine kleine HTML5 Anwendung geschrieben[2]. So kann durch das Öffnen der Index.html, der Shader lokal ausgeführt werden. Diese HTML5-Anwendung ist aber kein Bestandteil des Shaders selbst und erhält deswegen keine weitere Erläuterung in dieser Dokumentation.

[1] https://www.shadertoy.com/view/ddXGzB

[2] https://github.com/DHBW-INF2020/CG-Shader-LS-LB-KG-MH

1.1.2 Features

Die Kameraführung vom Shader funktioniert voll automatisch, kann aber durch das ziehen mit der Maus auf dem Canvas von der eigenen HTML5 Seite oder auf dem Canvas von Shadertoy, selber gesteuert werden.

Ganz zu Anfang des Shader Codes sind verschiedene Parameter abänderbar. So zum Beispiel die Variable `ANTIALIASING_ENABLED`. Ist diese auf `true` gesetzt, so ist Antialiasing aktiv, andernfalls nicht. Um mit den weiteren Attributen gewisse Einstellungen auszutesten, bietet es sich an diese auf Shadertoy abzuändern. Eine Änderung in der index.html ist aber auch möglich.

1.2 Pixel-Shader

Ein Pixel-Shader ist ein Programm, welches die Farbe eines einzelnen Pixels innerhalb einer Gruppe von Pixeln, also einem Bild, kalkuliert. Wenn also ein Bild berechnet werden soll, dann wird für jeden Pixel der Pixel-Shader ausgeführt. In Summe entsteht so ein Bild.

Das Programm ist dabei aber für jeden Pixel gleich. Ist das Endergebnis des Programms immer die Farbe Rot, dann wird jeder Pixel und damit auch das ganze Bild rot sein, wie im folgenden Programmcode zu sehen ist:

```
void main() {
  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Wenn also komplexere Bilder entstehen sollen, muss das Programm abhängig von verschiedenen Parametern die Farbe bestimmen. Solche Parameter können sein: die Koordinate des zu berechnenden Pixels, die aktuelle Zeit oder zum Beispiel die Position der Maus. Abhängig von diesen Parametern, die auch Uniforms genannt werden, kann der Programmierer eine Mathematische Funktion entwickeln, um die finale Farbe des Pixels dynamisch zu bestimmen. Solch eine dynamische Farbbestimmung wird beispielsweise durch den folgenden Pixel-Shader umgesetzt.

```
precision mediump float;
uniform vec2 u_resolution;
uniform vec2 u_mouse;
uniform float u_time;
```

```
void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    st.x *= u_resolution.x/u_resolution.y;
    vec3 color = vec3(0.);
    color = vec3(st.x,st.y,abs(sin(u_time)));
    gl_FragColor = vec4(color,1.0);
}
```

Da der auszuführende Shader für jeden Pixel gleich ist und das Endergebnis nur abhängig von den Eingabeparametern ist, kann das Programm hoch parallel, für hunderte Pixel gleichzeitig auf der Grafikkarte berechnet werden. Das macht die Berechnung eines Shaders sehr effizient und damit auch interessant als Basis für einen Raytracer.

1.3 Raytracing im Shader

Akkurates Raytracing, auch Path Tracing genannt, ist sehr aufwendig. Deshalb wird im Shader die weniger aufwendige Variante des iterativen Raytracings – einer Abwandlung des rekursiven Raytracings – verwendet. Dadurch können nur harte Schatten kalkuliert und Reflexionen und Beleuchtung nur für spiegelnde Objekte berechnet werden. Genaue diffuse Beleuchtung kann durch den hier gezeigten Shader nicht erzielt werden. Diffuses Licht muss durch das Phong-Beleuchtungsmodell simuliert werden. Dieses Modell wird in Kapitel 6 näher erläutert.

Für den Shader muss iteratives Raytracing verwendet werden, da Rekursion in keiner Art von GLSL unterstützt wird [1]. Und wie die Chruch-Turing-These beweist, ist die Umwandlung jedes rekursiven Programms in ein iteratives möglich, was auch die Implementierung eines Raytracers im Shader ermöglicht.

2 Szenenzusammenstellung

Im Shader ist ein Raum zu sehen, in dem sich springende Sphären befinden. Des Weiteren gibt es eine Punkt-Lichtquelle an der Decke.

Die Wände, der Boden und die Decke des Raumes scheinen gerade zu sein, dies ist jedoch nicht der Fall. In diesem Shader existieren nämlich nur Sphären.

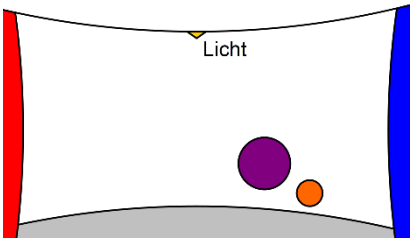

Abbildung 1: Abstraktes Beispiel der Szenenzusammenstellung des Shaders

In Abbildung 1 ist dies etwas überspitzt dargestellt. Um trotzdem den Raum rechtwinklig und die Kanten gerade erscheinen zu lassen, müssen die Wand-, Decken- und Boden-Sphären, im Vergleich zu den springenden Kugeln, entsprechend groß sein.

Um diese Illusion zu erschaffen, müssen die Sphären als Wände um ca. den Faktor 333 größer sein als die kleinen Sphären in der Mitte der Szene. Im folgenden Beispiel wird dies nochmals verdeutlicht. Hier haben die eingrenzenden Kreise einen Radius von 10000 LE und die Sphären in der Mitte lediglich einen Radius von 30 LE.

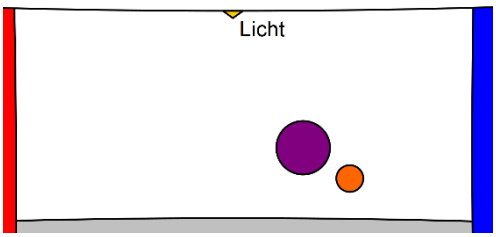

Abbildung 2: Nahezu gerade bzw. flache Oberflächen durch entsprechende Größe der Kreise

Die finale Szene setzt sich aus sechs Sphären für Wände, Decken und Boden zusammen. Eine Wand ist blau und eine weitere rot. Boden, Decke und restliche

Wände sind weiß-grau. In dem Raum springen vier Sphären – 2 kleine Sphären, wovon eine lila und die andere grün ist – und 2 große Sphären von denen eine blau und leicht matt ist sowie eine fast perfekt spiegelnde weiße Kugel.

3 Strahlenwurf / Ray-Casting

3.1 Projektionsebene

Entgegen normaler, wellenartiger Lichtausbreitung wird in diesem Modell das Licht durch einzelne Strahlen simuliert. Für dieses Projekt wird jeder Strahl durch einen Vektor ausgedrückt. Diese Vektoren zeigen, ausgehend von der aktuellen Kameraposition, in die virtuelle Szene. Um die Farbe jedes einzelnen Pixels zu bestimmen, spannt man direkt vor der Kamera eine Projektionsebene auf. Diese Projektionsebene kann man sich wie ein Raster vorstellen. Jedes Viereck in diesem Raster repräsentiert ein Pixel.

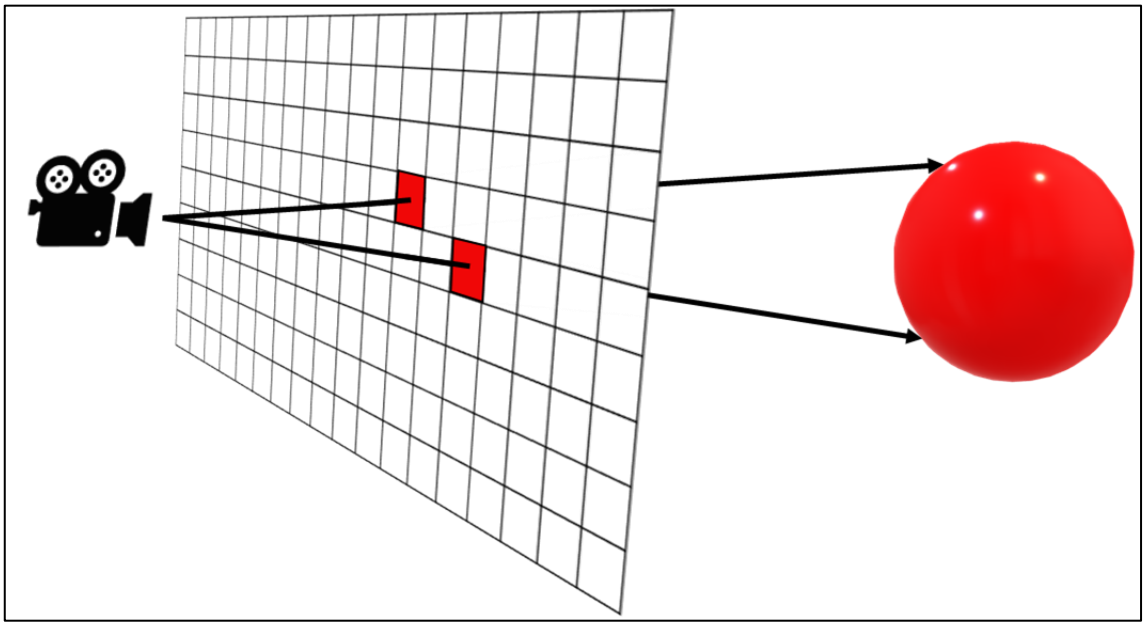

Abbildung 3: Projektionsebene als Raster

In Abbildung 3 ist so ein Raster beispielhaft aufgezeichnet. Hierbei sind zwei Strahlen zu sehen. Diese starten von der Position der Kamera und durchqueren zwei verschiedene Pixel auf der Projektionsebene. Anhand der Kollision der Strahlen mit dem Objekt in der Szene, kann später die Farbe des Pixels berechnet werden.

3.2 Mathematische Beschreibung eines Lichtstrahls

In Formel 1 ist die Funktion eines Strahls abgebildet.

$$Strahl(t) = KameraPosition + \frac{Pixel - KameraPosition}{\|Pixel - KameraPosition\|} * t$$

Formel 1: Parametergleichung für einen Strahl. Dieser ist abhängig vom Zeitparameter t

Die Punkte werden im dreidimensionalen Koordinatensystem als Vektoren dargestellt. Der Operator $||X||$ beschreibt den Betrag und damit die Länge des Ergebnisvektors. Teilt man einen Vektor durch seinen Betrag, so erhält man einen Einheitsvektor. Die Länge eines Einheitsvektors beträgt immer eins.

Der Einheitsvektor stellt in der Funktion die Richtung des Strahls dar. Mit dem Faktor t in der Formel kann man die Länge des Vektors einstellen. Dadurch dass t mit dem Einheitsvektor multipliziert wird, beträgt die Länge des Vektors immer genau den Wert in t. Hierbei ist mit Länge des Vektors der Abstand zur Kameraposition gemeint.

Die eben aufgestellte Gleichung wird für die weitere Verwendung wie folgt vereinfacht:

$$KameraPosition = \vec{K}$$

$$\frac{Pixel - KameraPosition}{\|Pixel - KameraPosition\|} = \vec{D}$$

$$Strahl(t) = \vec{S}(t) = \vec{K} + \vec{D} * t$$

Formel 2: Vereinfachung der Parametergleichung für den Strahl

3.3 Field of view (FOV)

FOV beschreibt das Sichtfeld der Kamera. In Abbildung 4 ist dieser Sachverhalt anhand eines Vergleichs näher erläutert. Auf der linken Seite ist die Kamera mit einem Sichtwinkel von 90° eingestellt und auf der rechten Seite mit einem Sichtwinkel von 45°. Die Skizzen stellen den Sachverhalt aus der Sicht von oben dar. Das bedeutet die gelbe Linie bildet die Breite der Projektionsfläche ab.

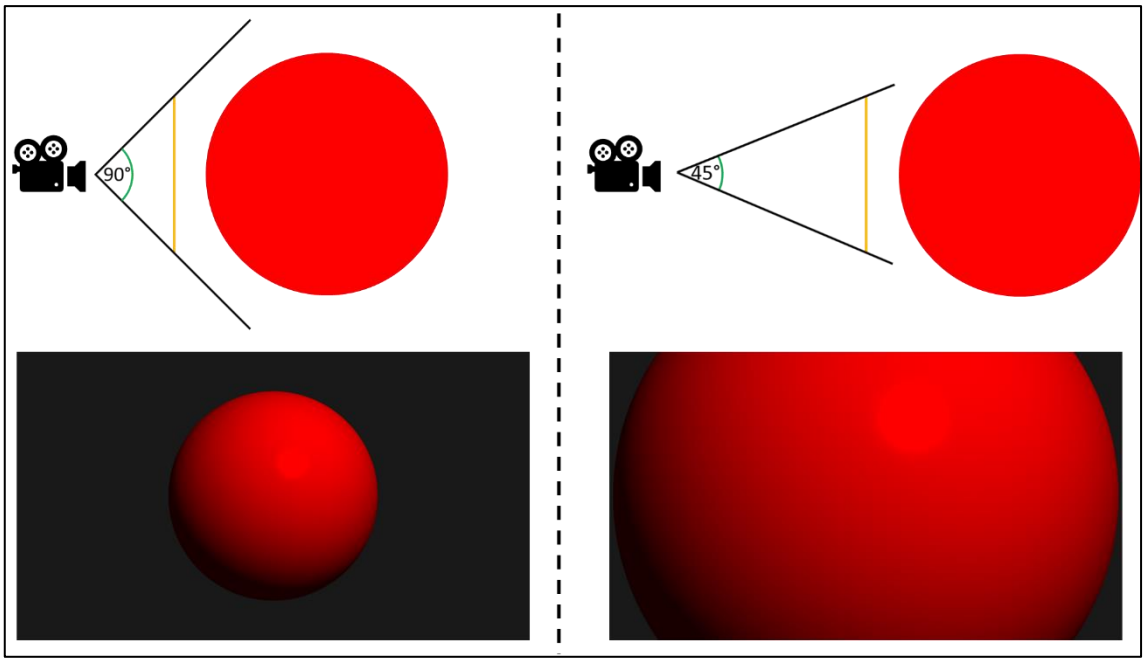

Abbildung 4: Zwei unterschiedliche Sichtwinkel der Kamera

Zu sehen ist, dass die rote Kugel im linken Bild weiter weg erscheint als im rechten Bild. Dies ist jedoch nicht der Fall. In beiden Fällen ist die Distanz zwischen Sphäre und der Projektionsfläche identisch, nur die Kamera im rechten Bild ist nicht in der Lage mit einem Sichtwinkel von 45° die ganze Kugel zu sehen. Daher erscheint das rechte Bild so, als hätte man in die Kugel gezoomt.

In beiden Fällen ist nicht nur die Distanz der Kugel zur Projektionsfläche dieselbe, sondern auch die Größe der Projektionsfläche. Der einzige Unterschied zwischen den beiden Beispielen liegt in der Entfernung von der Kamera zur Projektionsebene. Je weiter weg die Kamera von der Projektionsebene ist, desto kleiner wird der Sichtwinkel. Je näher die Kamera an der Projektionsfläche ist desto höher ist der Sichtwinkel.

Die Umsetzung im Shader ließ sich aus dieser Erkenntnis ableiten. Die Funktion `calculateDistanceFromProjectionPlane` berechnet den nötigen Abstand zwischen Kamera und der Projektionsfläche.

Die Distanz zur Projektionsebene lässt sich durch den Tangens beschreiben, in dem das Sichtfeld in der Mitte geteilt wird und daraus zwei kleinere rechtwinklige Dreiecke entstehen.

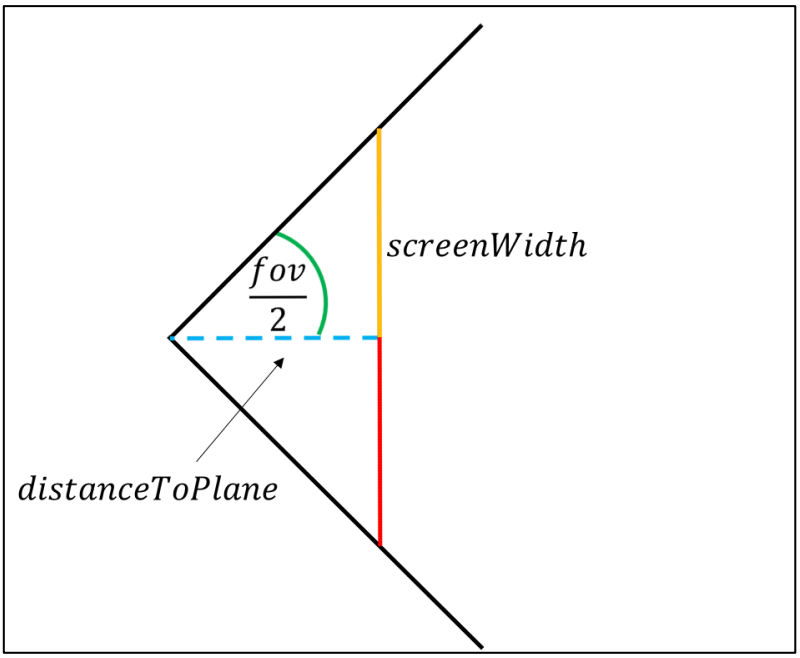

Abbildung 5: Skizze zur Berechnung vom Abstand zur Projektionsfläche

In Abbildung 5 ist dies anhand einer Skizze nochmals verdeutlicht. Dabei stellt die gestrichelte blaue Linie die Distanz zur Projektionsfläche dar. Die Skizze hier ist wieder aus der Sicht von oben dargestellt.

Aus diesem Bild lässt sich folgende Formel für den Abstand zur Projektionsfläche herleiten:

$$\tan\left(\frac{fov}{2}\right) = \frac{\frac{normalisedScreenWidth}{2}}{\boldsymbol{distanceToPlane}}$$

$$\boldsymbol{distanceToPlane} = \frac{\frac{normalisedScreenWidth}{2}}{\tan\left(\frac{fov}{2}\right)}$$

Formel 3: Abstand zur Projektionsebene

Die normalisierte Breite der Projektionsfläche „$normalisedScreenWidth$“ lässt sich durch das Seitenverhältnis beschreiben.

$$normalisedScreenWidth = windowRatio = \frac{screenHeight}{screenWidth}$$

Formel 4: Berechnung der normalisierten Projektionsflächenbreite

Bei $normalisedScreenWidth$ handelt es sich nicht um die Anzahl an Pixel in der Breite, sondern wie groß die Breite der Projektionsebene ist, wenn die Höhe der Projektionsebene genau eins beträgt. So ist die Rechnung unabhängig von den Pixeln und beachtet nur das Seitenverhältnis des Bildschirmes.

3.4 Drehung

Bevor die Strahlen von der Kamera in die Szene ausgesendet werden können, muss noch eine Komponente beachtet werden. Hiermit ist die Drehung der Kamera gemeint.

Das funktioniert so, dass für ein Pixel ganz normal ein Strahl generiert wird (siehe Abbildung 3) und dieser nach der Erstellung mit einer einzigen Drehmatrix gedreht wird. Diese Drehmatrix wird von der Methode `getRotationMatrix` generiert. Hierbei bedient es sich den drei verschiedenen Rotationsmöglichkeiten: Rotation um die x-Achse (Pitch), Rotation um die y-Achse (Yaw) und die Rotation um die z-Achse (Roll).

Die Formeln hierfür sind: [2]

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

Formel 5: Berechnung der Drehung in x-Richtung

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

Formel 6: Berechnung der Drehung in y-Richtung

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

Formel 7: Berechnung der Drehung in z-Richtung

Die Methode `getRotationMatrix` rechnet alle Matrizen mit der jeweils angegeben Rotation aus. Alle drei werden danach noch miteinander zusammenmultipliziert, damit schlussendlich nur eine vollständige Drehmatrix zurückgegeben wird.

3.5 Zusammenspiel

Im Shader bewegt sich die Kamera in einer Kreisbewegung, um die vier Kugeln herum. Dies wird in der Funktion `main` umgesetzt.

Das funktioniert so, dass die x-Position von einer Sinus-Funktion bestimmt wird und die z-Position von einer Kosinus-Funktion. Wendet man diese mit gleicher Phase und Amplitude an, so erhält man eine perfekte Kreisbewegung der Kamera.

Damit die Kamera auch immer in Richtung der Mitte des Raumes schaut, dreht sich die Kamera mit dem genau selben Winkel um die y-Achse. Zudem wurde für die Drehung in x-Richtung und z-Richtung jeweils eine Sinus-Funktion verwendet, damit die Kamera leicht schwankt.

Um den Effekt nun zu vollenden, wird das Sichtfeld mit einer Sinus-Funktion ständig geändert. Das sorgt für einen filmischen Effekt.

4 Sphärenkollision

Um zu verstehen, wie die Kollision eines Strahls mit einer Sphäre mathematisch funktioniert, muss zuerst verdeutlicht werden, wie eine Sphäre mathematisch beschrieben wird.

Im dreidimensionalen Raum ist dies relativ simpel. Dort wird eine Sphäre durch ihren Mittelpunkt A sowie ihrem Radius r beschrieben. Zu sehen ist dies in der folgenden Grafik:

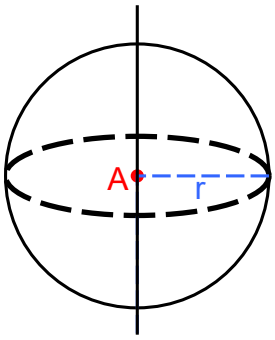

Abbildung 6: Abbildung einer Sphäre mit Mittelpunkt A und Radius r

Ein Punkt P befindet sich also auf der Oberfläche der Sphäre, wenn der Abstand zwischen A und P gleich r ist. Also wenn:

$$\left\|\vec{P} - \vec{A}\right\| = r$$

Formel 8: Sphärengleichung

Die aufgezeigte Sphärengleichung kann jetzt dazu verwendet werden, um zu prüfen, ob ein Strahl, der in die Szene geworfen wurde mit einer gegebenen Sphäre kollidiert. Dazu muss überprüft werden, ob sich ein Strahl $S(t)$ für einen bestimmten Faktor t_n in, oder zumindest auf der Sphäre befindet (t_n bezieht sich

hierbei auf das t aus Formel 2). Diese Fälle lassen sich wie folgt grafisch darstellen.

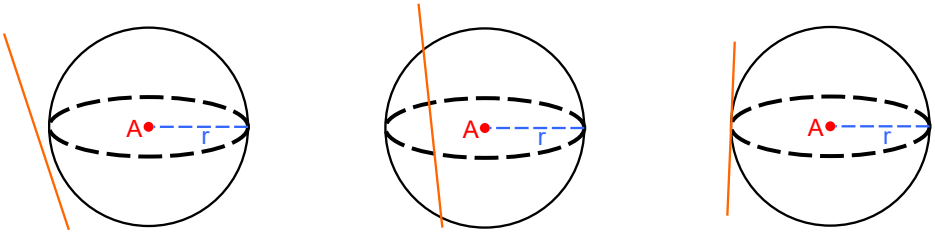

Abbildung 7: Mögliche Fälle für die Sphärenkollision

Die Kollision zwischen Sphäre und Strahl kann mathematisch beschrieben werden, indem der Strahl $\vec{S}(t)$ als Punkt $\vec{P}$ in die Sphärengleichung (siehe Formel 8) eingefügt wird.

$$\left\|\vec{S}(t) - \vec{A}\right\| = \left\|\vec{K} + \vec{D} * t - \vec{A}\right\| = r$$

Formel 9: eingesetzter Strahl in die Sphärengleichung

Zum auflösen der Gleichung werden beide Seiten quadriert, da das Betragsquadrat eines Vektors dann in dessen Skalarprodukt mit sich selbst umgewandelt werden kann. Also $\|X\|^2 = (X) \circ (X)$

$$\left(\left\|\vec{K} + \vec{D} * t - \vec{A}\right\|\right)^2 = r^2$$

$$\left(\vec{K} + \vec{D} * t - \vec{A}\right) \circ \left(\vec{K} + \vec{D} * t - \vec{A}\right)$$

Über das Distibutivgesetzt kann diese Gleichung aufgelöst werden zu:

$$\vec{K} \circ \left(\vec{K} + \vec{D} * t - \vec{A}\right) + \left(\vec{D} * t\right) \circ \left(\vec{K} + \vec{D} * t - \vec{A}\right) - \vec{A} \circ \left(\vec{K} + \vec{D} * t - \vec{A}\right)$$

Wird das Distibutivgesetzt erneut angewandt und die Elemente nach dem skalaren Faktor t sortiert, dann folgt daraus die folgende Gleichung:

$$\left(\vec{D} \circ \vec{D}\right) * t^2 + \left(\left(\vec{D} \circ \vec{K}\right) + \left(\vec{D} \circ \vec{K}\right) - \left(\vec{D} \circ \vec{A}\right) - \left(\vec{D} \circ \vec{A}\right)\right) * t + \left(\left(\vec{K} \circ \vec{K}\right) - \left(\vec{K} \circ \vec{A}\right) - \left(\vec{K} \circ \vec{A}\right) + \left(\vec{A} \circ \vec{A}\right)\right) = r^2$$

Die einzelen Komponenten werden einzeln vereinfacht.

1. $(\vec{D} \circ \vec{D}) = (\|\vec{D}\|)^2$ das Skalarprodukt eines Vektors mit sich selbst entspricht dem Betragsquadrat.
2. $((\vec{D} \circ \vec{K}) + (\vec{D} \circ \vec{K}) - (\vec{D} \circ \vec{A}) - (\vec{D} \circ \vec{A})) = 2 * ((\vec{D} \circ \vec{K}) - (\vec{D} \circ \vec{A}))$
   Diese Gleichung kann weiter durch das Distributivgesetzt vereinfacht werden zu: $2 * (\vec{D} \circ (\vec{K} - \vec{A}))$
3. $((\vec{K} \circ \vec{K}) - (\vec{K} \circ \vec{A}) - (\vec{K} \circ \vec{A}) + (\vec{A} \circ \vec{A}))$ entspricht dem Ergebnis des zweiten Binoms. Deshalb kann dieser Term umgewandelt werden in die Grundform: $((\vec{K} - \vec{A}) \circ (\vec{K} - \vec{A}))$
   Auch hier gilt wieder das Skalarprodukt eines Vektors mit sich selbst entspricht dem Betragsquadrat. Deshalb folgt als Ergebnis: $(\|\vec{K} - \vec{A}\|)^2$

Werden diese Terme wieder in der Gleichung zusammengesetzt und r^2 auf die linke Seite gebracht stellt sich eine normale quadratische Gleichung für den Faktor t dar. Zu sehen ist dies in der folgenden Gleichung:

$$\|\vec{D}\|^2 * t^2 + 2 * (\vec{D} \circ (\vec{K} - \vec{A})) * t + \|\vec{K} - \vec{A}\|^2 - r^2 = 0$$

Formel 10: quadratische Gleichung zur Berechnung des Faktors t

Diese Gleichung lässt sich nun durch die abc-Formel lösen. Dabei sind jedoch nicht nur die Ergebnisse für den Faktor t interessant, sondern auch das Ergebnis der Diskriminanten, also dem Ergebnis unter der Wurzel. Denn letzteres bestimmt, ob eine Kollision vorliegt. Dabei gibt es, wie in Abbildung 7 zu sehen ist, folgende drei Fälle:

1. Keine Kollision: Diskriminante kleiner 0
2. Kollision: Diskriminante größer 0
3. Beinahe Kollision (tangiert): Diskriminante gleich 0

Von diesen Fällen muss nur beim Zweiten, der jeweillige Faktor t berechnet werden. Für alle weiteren Fälle ist t irrelevant, da keine Kollision vorliegt.

Wenn also eine Kollision wie in der oben gezeigten Grafik vorliegt, muss die

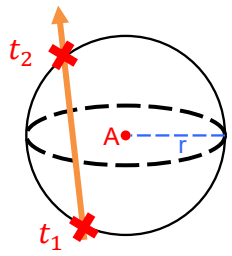

Abbildung 8: Sphäre wird von einem Strahl getroffen

quadratische Gleichung gelöst werden. Zu beachten ist dabei die Richtung des Strahls, die hier von t_1 nach t_2 zeigt. Das bedeutet, dass der Punkt bei t_1 näher zum Ausgangspunkt liegt.

$$t_{1/2} = -\frac{2 * \left(\vec{D} \circ \left(\vec{K} - \vec{A}\right)\right) \pm \sqrt{\left(2 * \left(\vec{D} \circ \left(\vec{K} - \vec{A}\right)\right)\right)^2 - 4 * \left\|\vec{D}\right\|^2 * \left(\left\|\vec{K} - \vec{A}\right\|^2 - r^2\right)}}{2}$$

Wird diese Gleichung unabhäng von dem oben aufgezeigten Beispiel gelöst, muss aus $t_{1/2}$ der kleinere Wert ermittelt werden. Diesen kann man jedoch vorab schon bestimmen. Zuerst beschreibt man t_1 und t_2 getrennt:

$$t_1 = -\frac{2 * \left(\vec{D} \circ \left(\vec{K} - \vec{A}\right)\right) - \sqrt{\left(2 * \left(\vec{D} \circ \left(\vec{K} - \vec{A}\right)\right)\right)^2 - 4 * \left\|\vec{D}\right\|^2 * \left(\left\|\vec{K} - \vec{A}\right\|^2 - r^2\right)}}{2}$$

$$t_2 = -\frac{2 * \left(\vec{D} \circ \left(\vec{K} - \vec{A}\right)\right) + \sqrt{\left(2 * \left(\vec{D} \circ \left(\vec{K} - \vec{A}\right)\right)\right)^2 - 4 * \left\|\vec{D}\right\|^2 * \left(\left\|\vec{K} - \vec{A}\right\|^2 - r^2\right)}}{2}$$

Formel 11: finale Formeln zur Berechnung der Kollisionsdistanzen t

t_1 wird das Ergebnis der Wurzel immer negativ zum restlichen Term berechnen. Das Ergebnis der Wurzel für zwei Kollisonen ist immer größer als Null, also positiv. Daraus lässt sich ableiten, das t_1 das positive Ergebnis der Wurzel immer

vom restlichen Term subtrahieren wird. Daher wird das Ergebnis durch die Subtraktion immer kleiner. Bei t_2 hingegen wird das Ergebnis der Wurzel immer addiert und t_2 wird größer. So beschreibt t_1 immer die kleinere Distanz.

Der in diesem Kapitel abgebildeten Sachverhalt ist genauso in der Methode `collideWith` umgesetzt.

t_1 kann zu guter Letzt in die Strahlengleichung $S(t)$ eingesetzt werden, um den nächstgelegenste Kollsionspunkt zu berechnen. [3]

5 Multiple Kollisionen

Im vorherigen Abschnitt wurde beschrieben, wie eine Kollision mit einer Sphäre abläuft. Jedoch besteht eine Szene selten aus einer einzelnen Kugel. Deshalb muss auch betrachtet werden, wie die Kollision eines Strahls mit mehreren Sphären abläuft. Dieser Fall ist in der folgenden Grafik zu sehen.

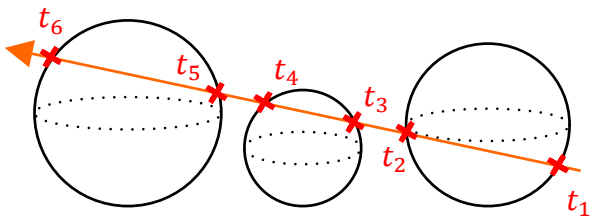

Abbildung 9: Ein Strahl trifft mehrere Sphären zu bestimmten Distanz-/Zeitpunkten

Für eine korrekte Licht-, Schatten- und Reflexions-Berechnung ist nur der erste Kollisionspunkt des Strahls relevant. Im gezeigten Beispiel werden deshalb alle Punkte außer t_1 verworfen, da t_1 den kleinsten Wert von allen t_n darstellt.

Die Funktion `collideWithClosest` setzt dieses Prinzip um. Der generierte Strahl wird versucht mit allen Sphären im Bild zu kollidieren. Durch ein einfaches iteratives Verfahren wird am Ende nur der kleinste Faktor t verwendet, um durch die Strahlengleichung $S(t)$ den Kollisionspunkt zu berechnen und zurückzugeben.

6 Berechnung des Lichts

Für die Simulation der Lichtintensität im Shader wird das Phong-Beleuchtungsmodell verwendet. Dieses setzt sich aus den drei Komponenten Umgebungsbeleuchtung, Diffuse-Beleuchtung und Spiegelnde-Beleuchtung zusammen. Diese drei Beleuchtungsarten werden aufaddiert und ergeben für einen Punkt in der Szene die Lichtintensität. Dieser Zusammenhang wird durch die folgende Grafik[3] visualisiert:

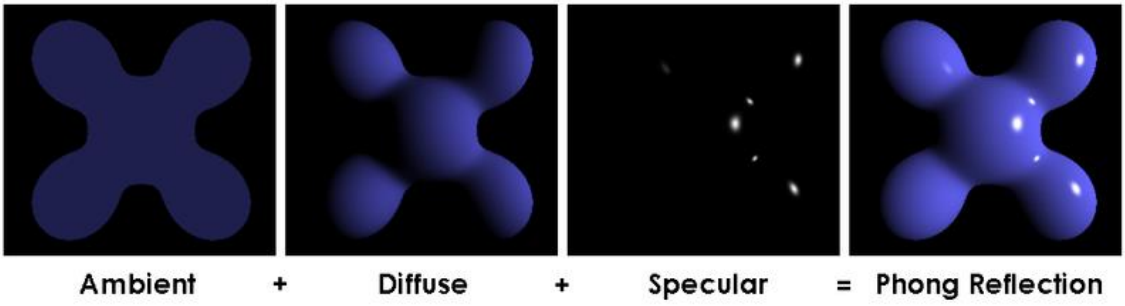

Abbildung 10: Die einzelnen Komponenten des Phong Beleuchtungsmodells

Die Stärke jener Komponenten wird dann kalkuliert, wenn ein Strahl in die Szene geworfen wird und dieser an einem Punkt mit einer Sphäre kollidiert. Dieser Fall ist in der folgenden Grafik modelliert.

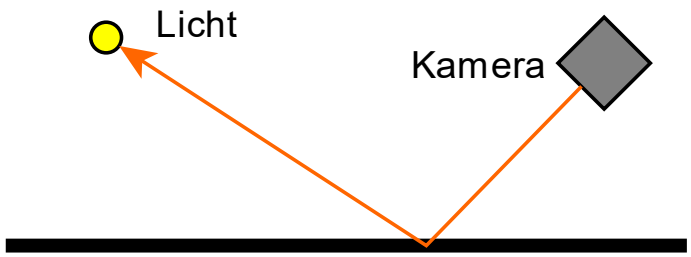

Abbildung 11: Einfallender Strahl - vom Kollisionspunkt wird der Strahl direkt an die Lichtquelle weitergeleitet

[3] https://de.wikipedia.org/wiki/Datei:Phong_components_version_4.png

Dazu wird zum einen der normalisierte Normalenvektor des getroffenen Objekts $\vec{N}_O$ bestimmt. Zum anderen wird der normalisierte Vektor $\vec{R}$ vom getroffenen Objekt zum Licht aufgestellt. Diese werden durch folgende Formeln beschrieben:

$$\vec{N}_O = \frac{(Kollisionspunkt\ -\ Sphärenposition)}{\|Kollisionspunkt\ -\ Sphärenposition\|}$$

$$\vec{R}_{zumLicht} = \frac{(\vec{P}_L - \vec{P}_K)}{\|\vec{P}_L - \vec{P}_K\|} = \frac{(LichtPosition\ - Kollisionspunkt)}{\|LichtPosition\ - Kollisionspunkt\|}$$

Formel 12: Formel für den Normalenvektor der Oberfläche und den Vektor vom Objekt zum Licht

Mit diesen beiden Vektoren können dann die Stärke der diffusen und spiegelnden Beleuchtung bestimmt werden.

Die Intensität der jeweiligen Lichttypen wird nach ihrer Berechnung verwendet, um die Farbe eines Punkts zu berechnen. Die dafür verwendete Formel wird sollte der Punkt aber im Schatten wird ein leicht verändertes Modell angewandt, welches in einem späteren Kapitel genauer beschrieben wird. [4]

$$Farbe_{\ Sphäre} = Sphärenfarbe\ *\ Lichtfarbe\ *\ (Lichtintensität\ nach\ Phong)$$

$$Farbe_{\ Sphäre} = Sphärenfarbe\ *\ Lichtfarbe * (I_U + I_D\ + I_S)$$

$$I_U = Intensität\ des\ Umgebungslichts$$

$$I_D = Intensität\ des\ diffusen\ Lichts$$

$$I_S = Intensität\ des\ spiegelnden\ Lichts$$

Formel 13: Farbkalkulation für Objektkollision außerhalb des Schattens

6.1 Diffuse Lichtintensität

Die diffuse Stärke, die abhängig vom Einfallswinkel des Lichts ist, wird durch das Skalarprodukt aus $\vec{N}_O$ und $\vec{R}$ bestimmt. Da beide Vektoren normiert sind, kann dadurch bestimmt werden, in welcher Relation die Vektoren zueinanderstehen. Steht das Licht beispielsweise direkt über der Fläche ist das Ergebnis des Skalarprodukts gleich eins und damit die Intensität des Lichts maximal. Je flacher das Licht jedoch auf die Oberfläche fällt, desto schwächer wird die Intensität des Lichtes. Dies wird in der folgenden Grafik modelliert.

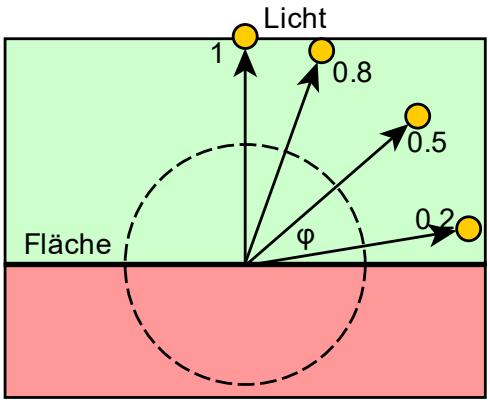

Abbildung 12: Ergebnis des Skalarprodukts je nach Winkel zwischen Normalenvektor und Vektor zum Licht

Da sich die Intensität des Lichts zwischen eins und null bewegt muss das Ergebnis des Skalarprodukts der Vektoren gefiltert werden. Denn wird der Winkel φ zwischen $\vec{N}_O$ und R größer 90° ist das Ergebnis des Skalarprodukts zweier normierter Vektoren negativ. Dies wird durch den roten Bereich in der Grafik dargestellt. Falls dieser Fall eintritt, wird das Ergebnis durch Null ersetzt. Mathematisch lässt sich der im letzten Abschnitt beschriebene Zusammenhang also wie folgt modellieren:

$$[diffuse\ Intensität]\ I_D = \begin{cases} \vec{N}_O \circ \overrightarrow{R} \text{ für } \varphi < 90^\circ \\ 0 \text{ für } \varphi > 90^\circ \end{cases}$$

Formel 14: Formel für die Intensität des diffusen Lichts

6.2 Spiegelnde Lichtintensität

Das spiegelnde oder spekulare Licht simuliert die hellen Punkte auf einem glänzenden Objekt. Um dieses Licht zu simulieren, muss zusätzlich zum Normalenvektor des getroffenen Objekts $\vec{N}_O$ und normalisierte Vektor $\vec{R}$ vom getroffenen Objekt zum Licht, auch der normalisierte Vektor $\vec{V}_{Kamera}$ zwischen Kollisionspunkt und Kamera berechnet werden.

$$\vec{V}_{Kamera} = \frac{(KameraPosition - Kollisionspunkt)}{\|KameraPosition - Kollisionspunkt\|}$$

Formel 15: Formel für den Kameravektor

Zwischen diesem Vektor und dem normalisierten Vektor $\vec{L}_{Reflexion}$ der Reflexion des Lichtstrahls muss nun das Verhältnis Θ bestimmt werden. $\vec{L}_{Reflexion}$ ist wie folgt definiert.

$$\vec{L}_{Reflexion} = \frac{\vec{R} - 2 * (\vec{N}_O \circ \vec{R}) * \vec{N}_O}{\left\|\vec{R} - 2 * (\vec{N}_O \circ \vec{R}) * \vec{N}_O\right\|}$$

Formel 16: Berechnung der Reflexion des Lichtstrahls

Das Verhältnis Θ wird durch das Skalarprodukt aus $\vec{L}_{Reflexion}$ und $\vec{V}_{Kamera}$ bestimmt.

$$\Theta = \vec{L}_{Reflexion} \circ \vec{V}_{Kamera}$$

Formel 17: Berechnung des Verhältnisses zwischen Kameravektor und dem reflektierten Lichtstrahl

Θ wird abschließend um die Spiegelungsfaktor `shininess` der getroffenen Oberfläche potenziert. [5]

$$[spiegelnde\ Intensität]\ I_S = \begin{cases} \Theta^{\text{shininess}}\ für\ \Theta \geq 0 \\ 0\ für\ \Theta < 0 \end{cases}$$

Formel 18: Finale Formel für die Intensität des spiegelnden Lichtanteils

7 Schattenwurf

Die Schattendarstellung in der Szene basiert, wie auch die Reflexionsberechnung auf der Sphärenkollision. Ein Objekt in der Szene wird von einem Strahl der Kamera getroffen. Nun wird geprüft, ob dieser Kollisionspunkt im Schatten liegt. Dies geschieht, indem ein Strahl vom getroffenen Punkt zum Licht geworfen wird. Dies ist analog zur Berechnung des Lichts. Dieser Strahl kann mathematisch durch den normalisierten Vektor R wie folgt beschrieben werden.

$$R_{zumLicht} = \frac{(\vec{P}_L - \vec{P}_K)}{\|\vec{P}_L - \vec{P}_K\|} = \frac{(LichtPosition - Kollisionspunkt)}{\|LichtPosition - Kollisionspunkt\|}$$

Für diesen Vektor werden die Kollisionspunkte mittels Sphärenkollision ermittelt (siehe Abschnitt 4). Da die Szene, einschließlich der Wände aus Sphären besteht wird eine Kollision definitiv stattfinden. Jedoch stellt sich die Frage, ob die Kollision vor der Lichtquelle oder hinter der Lichtquelle liegt. Liegt der neue Kollisionspunkt vor der Lichtquelle dann liegt auch der originale Punkt im Schatten. Liegt hingegen der neue Kollisionspunkt hinter der Lichtquelle, dann liegt auch der originale Punkt im Schatten. Dieses Beispiel wird in der folgenden Grafik genauer beschrieben.

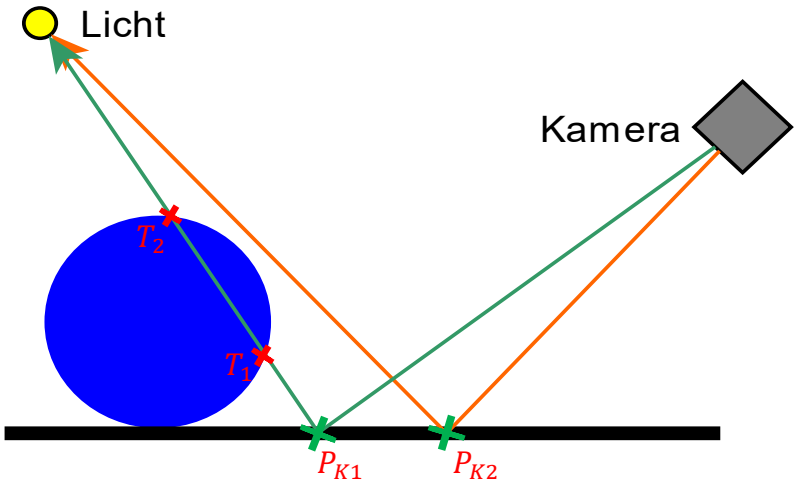

Abbildung 13: Beispiel zweier Strahlen bei der Schattenberechnung

In der Grafik sind zwei Strahlen zu sehen, die mit dem Boden der Szene kollidieren. Von den Kollisionspunkten P_{K1} und P_{K2} werden jeweils Strahlen zur Lichtquelle geworfen. Der grüne Strahl trifft dabei die Kugel, während der orangene Strahl direkt zum Licht zeigt und erst weit hinter dem Licht, mit der Position P_L mit einer weiteren Sphäre kollidiert. P_{K1} liegt im Schatten, da die Distanz D_L zwischen Licht und P_{K1} größer ist als die Distanz D_K zwischen P_{K1} und dem nächsten Schatten-Kollisionspunkt T_1. Mathematisch lässt sich dieser Zusammenhang wie folgt darstellen.

$$D_L = \|P_L - P_{K1}\|$$

$$D_T = \|T_1 - P_{K1}\|$$

$$P_{K1}\ ist\ im\ Schatten\ für\ D_T\ < D_L$$

Der Punkt P_{K2} ist wiederrum nicht im Schatten da hier $D_T\ > D_L$ ist.

Wenn ein Punkt nicht im Schatten liegt, wird die Lichtberechnung für diesen komplett ausgeführt. Liegt es jedoch im Schatten wird nur die Stärke der Umgebungsbeleuchtung des Phong-Models berechnet. Die Stärke des diffusen und spiegelnden Lichts wird gleich Null gesetzt.

Für die Farbkalkulation bedeutet dies:

$$Farbe_{\ Sphäre} = Sphärenfarbe\ *\ Lichtfarbe\ *\ (Lichtintensität\ nach\ Phong)$$

$$Farbe_{\ Sphäre} = Sphärenfarbe\ *\ Lichtfarbe * (Umgebungslichtstärke + 0\ + 0)$$

Formel 19: Berechnung der Farbe im Schatten

Diese „finale“ Schattenfarbe wird dann an zur Weiterverarbeitung in das Reflexionsmodell weitergebeben. Dies wird im folgenden Kapitel beschrieben.

8 Reflexion

Damit die Performance des Shaders nicht zu stark beeinträchtigt wird, werden Reflexionen in der Szene nur bis zu einem bestimmten Grad berechnet. Dieser Grad bestimmt, wie häufig ein Strahl reflektiert werden darf. In der Praxis ist für diesen Grad ein Wert zwischen 2, für die beste Performance, aber schlechtestes Aussehen und ca. 8 für das beste visuelle Ergebnis, aber eine schlechtere Performance, zu wählen.

Reflexionen bauen mathematisch auf der Sphärenkollision aus Abschnitt 4 auf. Hier wird vom berechneten Kollisionspunkt ein neuer Strahl in die Szene geworfen. Dieser neue Strahl verlässt die Sphäre nach dem Reflexionsgesetz mit demselben Winkel wie der einfallende Strahl. Nach dem Reflexionsgesetzt – Eintrittswinkel gleich Austrittswinkel – verlässt der reflektierte Strahl die Sphäre mit demselben Winkel wie der einfallende Strahl. Dazu muss zuerst der Normalenvektor $N_{Sphäre}$ der Sphäre am Kollisionspunkt $B_{Kollision}$ berechnet werden. Dazu wird zuerst der Sphärenmittelpunkt $P_{Sphäre}$ vom Kollisionspunkt subtrahiert und dann das Normalisierungsverfahren angewandt. Dieser Sachverhalt ist in der folgenden Formel dargestellt:

$$N_{Sphäre} = \frac{1}{\left|P_{Sphäre} - B_{Kollision}\right|} * \left(P_{Sphäre} - B_{Kollision}\right)$$

Formel 20: Normalenvektor der Sphäre

Aus dem Normalenvektor und dem Richtungsvektor D_{Strahl} des Strahls kann nun der Richtungsvektor R der Reflexion mit folgender Formel berechnet werden:

$$R = D_{Strahl} - 2 * \left(N_{Sphäre} \circ D_{Strahl}\right) * N_{Sphäre}$$

Formel 21: Formel für den Richtungsvektor der Reflexion

Abschließend wird ein Strahl mit der berechneten Richtung in die Szene geworfen. Auch dieser kollidiert wieder mit einem Objekt und liefert als Ergebnis wieder die Farbe des getroffenen Objektes. So sammeln sich für jeden Strahl, der in die Szene geworfen wird, mehrere Farben an. Diese werden zuerst mit dem Reflexionsfaktor der jeweils getroffenen Sphäre multipliziert. Dieser Faktor bewegt sich zwischen null, falls die Sphäre nicht reflektiert, und eins, falls die Sphäre einen perfekten Spiegel darstellt. Danach werden alle Farben aufaddiert, um die finale Pixelfarbe der Reflexion zu erzeugen. Dieser Sachverhalt wird durch das folgende Beispiel verdeutlicht. Die blaue Sphäre hat dabei einen Reflexionsfaktor $f_b = 0{,}7$ und die rote $f_r = 0{,}2$. In dem Beispiel trifft ein Strahl von der Kamera auf die blaue Sphäre. Der Strahl wird zur roten Kugel reflektiert und von dort ebenfalls wieder weiter in die Szene reflektiert.

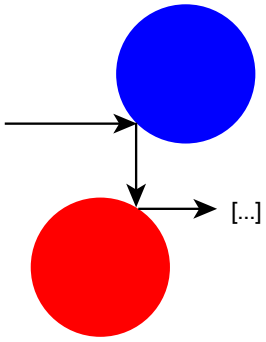

Abbildung 14: Abstrakte Darstellung der Reflexion eines Strahls

Die Farbe eines reflektierten Strahls wird immer mit den Reflexionsfaktoren der vorangegangenen Kugeln multipliziert.

Für die Farbkalkulation des Beispiels bedeutet dies, dass der Strahl, der die Blaue Kugel trifft als Farbe Blau zurückgibt, da er nicht reflektiert wurde. Der zweite Strahl gibt rot als Farbe zurück. Die Farbe muss aber durch den Reflexionsfaktor der Blauen Kugel modifiziert werden. Die Nächste Farbe muss dann mit den Reflexionsfaktoren der roten und der blauen Kugel multipliziert werden. Dies wird durch die folgende Gleichung dargestellt.

$$Farbe_{Reflexion} = blau\ + rot\ *\ f_b + weitereFrabe\ *\ f_b * f_r$$

Formel 22: Farbbestimmung der Reflexion durch aufaddieren der modifizierten Sphärenfarben

Die Reflexionskalkulation wird so lange durchgeführt bis die maximale Anzahl bzw. die Tiefe der Reflexion erreicht ist.

9 Antialiasing

Antialiasing kann optional im Shader aktiviert und deaktiviert werden. Wenn es aktiviert ist, kommt ein Multisampling-Verfahren zum Einsatz. Dabei werden neben dem eigentlichen Bildpunkt auch weitere Punkte, die dem Punkt nahestehen berechnet. Alle Ergebniswerte werden abschließend gemittelt.

Beim Multisampling gibt es verschiedene „Muster“ gebräuchlich. Für diese Arbeit werden 8 weitere Punkte oder Samples um den eigentlichen Bildpunkt berechnet. Ein abstraktes Beispiel der Verteilung der Sample-Punkte ist in der folgenden Grafik zu sehen.

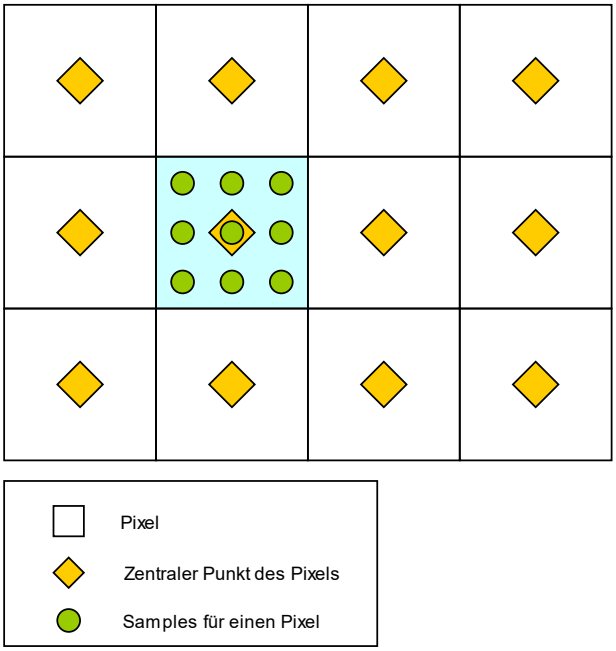

Abbildung 15: Abstrakte Darstellung von Antialiasing durch Mulisampling im Vergleich zum normalen Pixelraster

Jedes Quadrat in der Grafik repräsentiert einen Pixel, für den ohne Antialiasing jeweils ein Strahl ins Zentrum des Pixels, zur orangenen Raute, geworfen wird. Wir Antialiasing für den Shader aktiviert werden 8 weitere Strahlen um den Mittelpunkt geworfen. Die genaue Distanz zwischen den zusätzlichen Sample-Punkten um den zentralen Punkt des Pixels (Punkt in der Mitte) beträgt minimal 0,3 LE und maximal 0,42 LE. Dieser Sachverhalt lässt sich durch folgende Gleichung ausdrücken:

$$PixelFarbe = \frac{\sum_{i=1}^{n} SubPixelFarbe}{n} \text{ für } n = 9$$

Formel 23: Addieren aller Subpixelfarben für die Antialiasing Funktion

Die Summe aller Subpixel-Farben geteilt durch die Anzahl der Subpixel ergibt dann die finale Pixelfarbe. Das Ergebnis dieser Art der Kantenglättung ist in den folgenden zwei Grafiken im Vergleich zu sehen. Dabei zeigt das erste Bild den Shader ohne Antialiasing. In der Grafik ist klare Stufenbildung an Kanten zu erkennen. Des Weiteren erscheinen die Sphären sich perfekt Rund. Das zweite Bild hingegen ist mit aktiviertem Antialiasing aufgenommen. Hier fallen die Abgestuften kannten deutlich weniger ins Auge. Und auch die Sphären und Schatten, sowie die Reflexion erscheinen glatter.

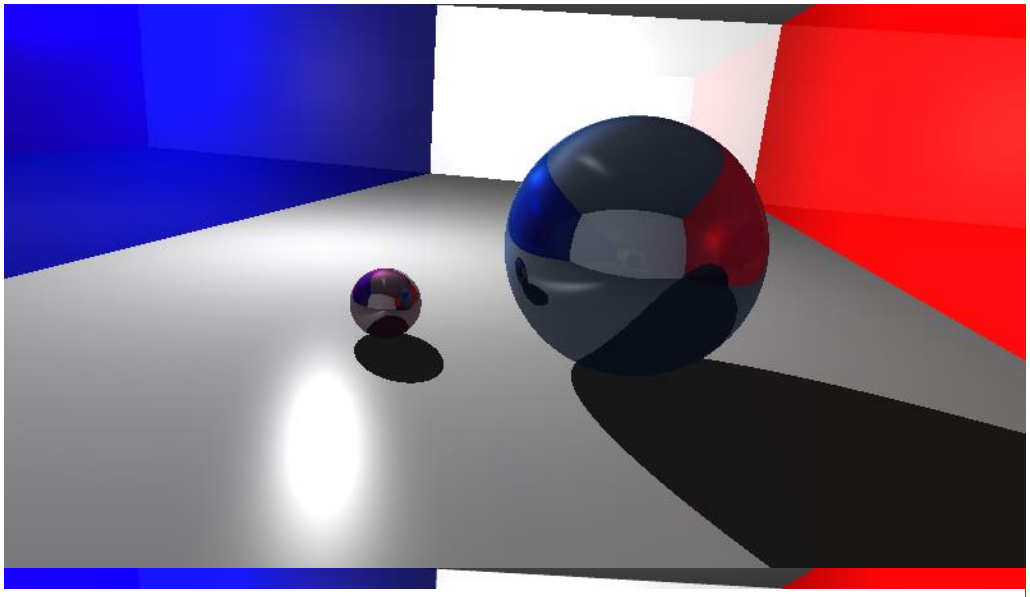

Abbildung 17: Shader ohne Antialiasing

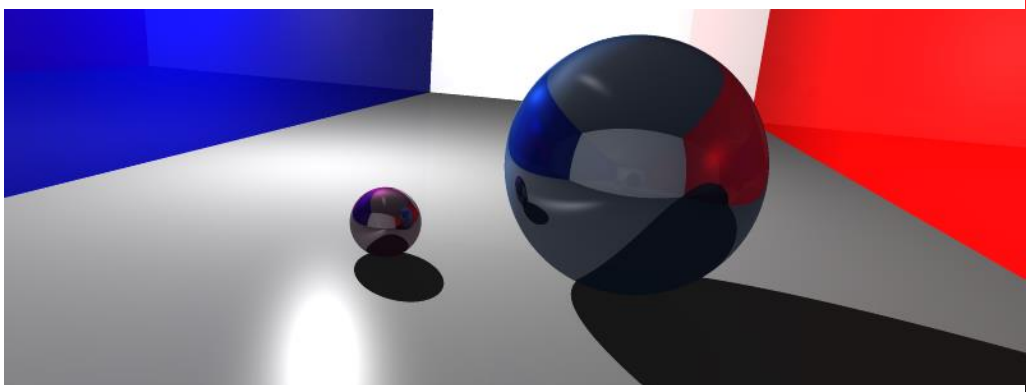

Abbildung 16: Shader mit Antialiasing

Durch diese Farbmittlung, wie sie oben beschrieben ist, können harte bzw. scharfe Kanten in der Szene weicher gezeichnet werden, was das Bild weniger pixelhaft und dadurch realistischer erscheinen lässt. Jedoch entstehen dadurch auch teils verwaschene Kanten in der Grafik. Dies ist vor allem der Fall, wenn mehrere Objekte in einem Pixel aufeinandertreffen und dadurch stark unterschiedliche Farben gemittelt werden müssen.

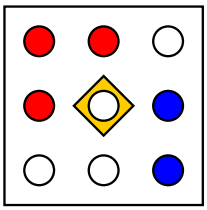

Abbildung 18: Darstellung eines einzelnen Pixels mit Multisampling

Der oben dargestellte Pixel veranschaulicht dieses Problem genauer. Hier treffen beispielsweise die Rote Wand, der weiße Boden sowie die blaue Spähre in der Szene aufeinander. Der mittlere Subpixel würde eigentlich einen Teil der weißen Wand zeigen. Durch die Bildung des Mittelwerts aller Farben erscheint dieser Pixel jedoch in einer Farbmischung aus rot, weiß und blau. Passiert dies nun an vielen Stellen im Bild können Kanten leicht verwaschen erscheinen. Als Lösung für dieses Problem kann die Auflösung des Bildes erhöht werden. Damit wird die anzuzeigende Fläche für einen Pixel kleiner, und damit auch die Wahrscheinlichten dafür, dass über viele unterschiedliche Farben gemittelt werden muss, geringer.

Eine weitere Möglichkeit verwischte Kanten zu reduzieren, ist die Gewichtung der einzelnen Subpixel mit einem individuellen Faktor. Dadurch könnte der Farbwert des mittleren Subpixels doppelt gewertet werden.

10 Sprunganimation

In der dargestellten Szene springen die Sphären auf und ab. Diese Sprunganimation wird erreicht, indem die z-Sphärenkoordinate (vertikal) anhand einer mathematischen Funktion verschoben werden. Die auf einem Sinus basierende Funktion wird im Folgenden dargestellt.

$$z_{Sphäre} = |Höhe * \sin(Zeit * x_{t-Faktor} + Phase)|$$

Formel 24: Formel zur Änderungen der z-Koordinate der Sphäre

$z_{Sphäre}$ beschreibt die y-Koordinate der Sphäre. Die Variable $Höhe$ verändert die Amplitude der Sinusfunktion, also die Sprunghöhe der Sphäre. $Zeit$ ist ein konstant zunehmender Wert, der durch $x_{t-Faktor}$ skaliert wird. Durch die Kombination von beiden kann die Frequenz der Sinusfunktion erhöht oder vermindert werden. Dies korrespondiert zu einer beschleunigten oder verlangsamten Sprungfrequenz der Sphäre. Des Weiteren kann eine Phasenverschiebung mit $Phase$ eingefügt werden als zusätzliche Möglichkeit die Sprungfunktionen von mehreren Sphären asynchron zu gestalten. Letztlich ist es notwendig vom Ergebnis aus $Höhe * \sin(Zeit * x_{t-Faktor} + Phase)$ den Betrag, also den absoluten Wert zu berechnen, damit die animierte Sphäre nicht im Boden der Szene

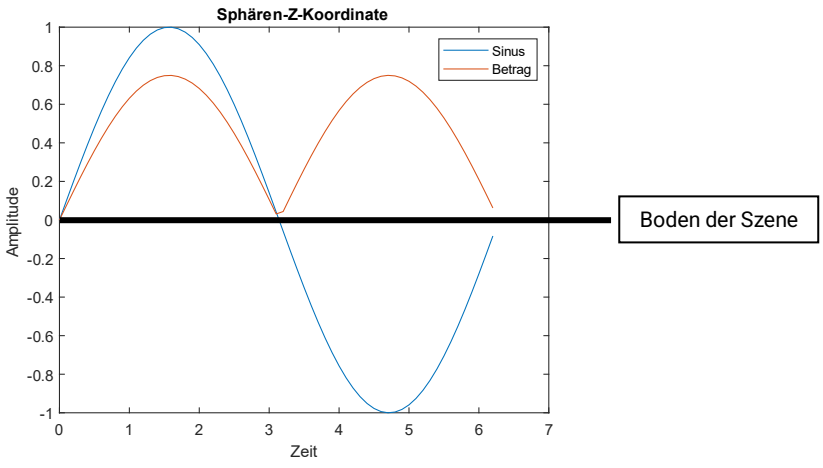

Abbildung 19: Sinusfunktion und Betrag einer Sinusfunktion in einem Koordinatensystem

verschwinden kann, wenn die Sinusfunktion ein negatives Ergebnis liefert. Dieser Sachverhalt wird in der folgenden Grafik verdeutlicht.

Zur besseren Lesbarkeit des Diagramms wurde für die orangene Funktion die Amplitude leicht vermindert.

Neben der Betragsfunktion muss noch beachtet werden, dass die Verschiebung der Z-Koordinate der Sphäre mit dem Radius der Sphäre addiert werden muss. Dies liegt daran, dass die Z-Koordinate der Sphäre von deren Mittelpunkt gemessen wird. Wird der Radius nicht aufaddiert versinkt die Sphäre, auch trotz Betrags-Sprung-Funktion bis zur Mitte im Boden.

11 Literaturverzeichnis

[1] „Core Language (GLSL),“ [Online]. Available: https://www.khronos.org/opengl/wiki/Core_Language_(GLSL)#Recursion. [Zugriff am 17. Januar 2023].

[2] „Drehmatrizen des Raumes $\mathbb{R}^3$,“ [Online]. Available: https://de.wikipedia.org/wiki/Drehmatrix#Drehmatrizen_des_Raumes_%E2%84%9D%C2%B3. [Zugriff am 17. Januar 2023].

[3] O. Aflak, „Medium,“ 26 07 2020. [Online]. Available: https://medium.com/swlh/ray-tracing-from-scratch-in-python-41670e6a96f9. [Zugriff am 25 12 2022].

[4] „Wikipedia,“ Wikipedia, 03 03 2021. [Online]. Available: https://de.wikipedia.org/wiki/Phong-Beleuchtungsmodell. [Zugriff am 5 01 2023].

[5] Learn OpenGL, „Learn OpenGL,“ Learn OpenGL, [Online]. Available: https://learnopengl.com/Lighting/Basic-Lighting. [Zugriff am 12 01 2023].

12 Abbildungsverzeichnis

Abbildung 1: Abstraktes Beispiel der Szenenzusammenstellung des Shaders 5
Abbildung 2: Nahezu gerade bzw. flache Oberflächen durch entsprechende Größe der Kreise 5
Abbildung 3: Projektionsebene als Raster 7
Abbildung 4: Zwei unterschiedliche Sichtwinkel der Kamera 9
Abbildung 5: Skizze zur Berechnung vom Abstand zur Projektionsfläche .. 10
Abbildung 6: Abbildung einer Sphäre mit Mittelpunkt A und Radius r 13
Abbildung 7: Mögliche Fälle für die Sphärenkollision 14
Abbildung 8: Sphäre wird von einem Strahl getroffen 16
Abbildung 9: Ein Strahl trifft mehrere Sphären zu bestimmten Distanz-/Zeitpunkten 18
Abbildung 10: Die einzelnen Komponenten des Phong Beleuchtungsmodells 19
Abbildung 11: Einfallender Strahl - vom Kollisionspunkt wird der Strahl direkt an die Lichtquelle weitergeleitet 19
Abbildung 12: Ergebnis des Skalarprodukts je nach Winkel zwischen Normalenvektor und Vektor zum Licht 21
Abbildung 13: Beispiel zweier Strahlen bei der Schattenberechnung 23
Abbildung 14: Abstrakte Darstellung der Reflexion eines Strahls 26
Abbildung 15: Abstrakte Darstellung von Antialiasing durch Mulisampling im Vergleich zum normalen Pixelraster 28
Abbildung 16: Shader mit Antialiasing 29
Abbildung 17: Shader ohne Antialiasing 29
Abbildung 18: Darstellung eines einzelnen Pixels mit Multisampling 30
Abbildung 19: Sinusfunktion und Betrag einer Sinusfunktion in einem Koordinatensystem 31

13 Formelverzeichnis

Formel 1: Parametergleichung für einen Strahl. Dieser ist abhängig vom Zeitparameter t 8
Formel 2: Vereinfachung der Parametergleichung für den Strahl 8
Formel 3: Abstand zur Projektionsebene 10
Formel 4: Berechnung der normalisierten Projektionsflächenbreite 10
Formel 5: Berechnung der Drehung in x-Richtung 11
Formel 6: Berechnung der Drehung in y-Richtung 11
Formel 7: Berechnung der Drehung in z-Richtung 11
Formel 8: Sphärengleichung 13
Formel 9: eingesetzter Strahl in die Sphärengleichung 14
Formel 10: quadratische Gleichung zur Berechnung des Faktors t 15
Formel 11: finale Formeln zur Berechnung der Kollisionsdistanzen t 16
Formel 12: Formel für den Normalenvektor der Oberfläche und den Vektor vom Objekt zum Licht 20
Formel 13: Farbkalkulation für Objektkollision außerhalb des Schattens 20
Formel 14: Formel für die Intensität des diffusen Lichts 21
Formel 15: Formel für den Kameravektor 22
Formel 16: Berechnung der Reflexion des Lichtstrahls 22
Formel 17: Berechnung des Verhältnisses zwischen Kameravektor und dem reflektierten Lichtstrahl 22
Formel 18: Finale Formel für die Intensität des spiegelnden Lichtanteils 22
Formel 19: Berechnung der Farbe im Schatten 24
Formel 20: Normalenvektor der Sphäre 25
Formel 21: Formel für den Richtungsvektor der Reflexion 25
Formel 22: Farbbestimmung der Reflexion durch aufaddieren der modifizierten Sphärenfarben 27
Formel 23: Addieren aller Subpixelfarben für die Antialiasing Funktion 29
Formel 24: Formel zur Änderungen der z-Koordinate der Sphäre 31