DHBW
Duale Hochschule
Baden-Württemberg

Computergrafik

RayTracing-Algorithmus

Benotete Abgabe

von

8978851

9482203

und 9741250

Abgabedatum:10.01.2021

Dozent	Johannes Riester
Bearbeitungszeitraum	6 Wochen

Inhaltsverzeichnis

Abbildungsverzeichnis 3
1 Theoretische Prinzipien 4
1.1 Grundlagen Raytracing 4
1.2 Phong 4
1.2.1 Ambient Beleuchtungswert 4
1.2.2 4
1.2.3 Specular Beleuchtungswert 5
1.2.4 Zusammensetzung der Beleuchtungswerte 6
2 Algorithmische Umsetzung 7
2.1 Kollisionserkennung von Rays mit Objekten 7
2.2 Quadratische Lichtquelle 7
2.2.1 Berechnung der Schatten 7
2.2.2 Berechnung der Beleuchtung 9
3 Implementierung 11
3.1 Funktionen 11
3.2 Klassenmodell 12
3.3 Mögliche Optimierungen 13
3.3.1 Performance 13
3.3.2 Bildgebung 14
Quellenverzeichnis 15

Abbildungsverzeichnis

- Abbildung 1: Bild mit Umgebungsbeleuchtung 4
- Abbildung 2: Bild mit diffuser Beleuchtung 5
- Abbildung 3: Bild mit „Specular“-Beleuchtung 6
- Abbildung 4: Zusammengesetztes Bild 6
- Abbildung 5: Bild mit 1 systematischen Shadow Ray 8
- Abbildung 6: Bild mit 4 systematischen Shadow Rays 8
- Abbildung 7: Bild mit 9 systematischen Shadow Rays 8
- Abbildung 8: Bild mit 8 zufälligen Shadow Rays 8
- Abbildung 9: Bild mit 9 systematischen und 8 zufälligen Shadow Rays 9

1 Theoretische Prinzipien

1.1 Grundlagen Raytracing

1.2 Phong

Beim Phong-Beleuchtungsmodell wird die Beleuchtung eines Punktes im Raum aus den drei Werten, “Ambient”, “Diffuse” und “Specular, zusammengesetzt. Dabei wird nicht das Ziel verfolgt, die Realität möglichst realistisch abzubilden, sondern ein einfaches und schnell zu berechnendes Verfahren zu verwenden.

1.2.1 Ambient Beleuchtungswert

Der “Umgebungswert” eines Punkts ist nur abhängig von dem Ambient Wert der Oberfläche und dem Ambient Wert der Lichtquelle, bzw. der Lichtquellen. Er stellt somit eine konstante Beleuchtung des gesamten Raums dar, unabhängig von der Position der Lichtquelle oder des Beobachters.

$$Beleuchtung_{ambient} = Objekt_{ambient} * Lichtquelle_{ambient}$$

In Abbildung 1 sieht man ein Bild was ausschließlich mit ambienter Beleuchtung erzeugt wurde. Alle Wände sind gleichmäßig beleuchtet.

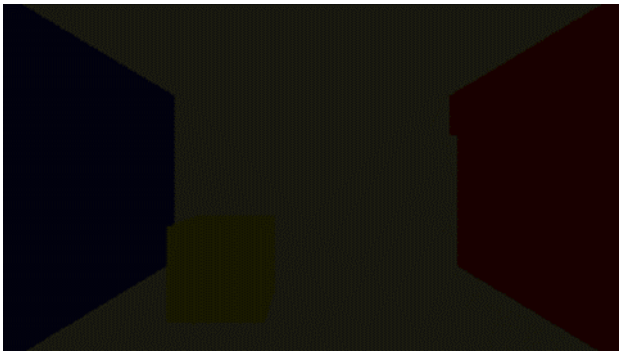


Abbildung 1: Bild mit Umgebungsbeleuchtung

1.2.2 Diffuse-Beleuchtungswert

Der diffuse Beleuchtungswert ist abhängig von den „Diffuse“-Werten der Lichtquelle , dem Objekt, sowie dem Winkel der Oberfläche zur Lichtquelle. Steht der

Normalenvektor der Oberfläche des Objekts auf dem Vektor zur Lichtquelle ist der diffuse Wert maximal.

$$Beleuchtung_{diffuse} = Objekt_{diffuse} * Lichtquelle_{diffuse} * (Richtung_{licht} \cdot Normalenvektor_{oberfläche})$$

In Abbildung 2 ist ein Bild zu sehen, welches nur durch diffuses Licht beleuchtet wurde. Die Flächen sind jetzt unterschiedlich beleuchtet und man sieht auf der dem Licht abgewandten Seite der Quader kein Licht, da das Skalarprodukt aus der Richtung des Lichts und des Normalenvektors der Oberfläche <= 0 ist.

Da der „Ambient"- und der „Diffuse"-Wert nicht von der Position des Betrachters abhängen, kann man diese Werte bei einer Echtzeitanwendung im Voraus berechnen.

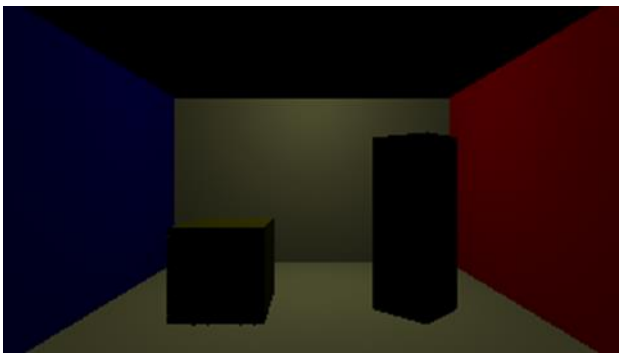


Abbildung 2: Bild mit diffuser Beleuchtung

1.2.3 Specular Beleuchtungswert

Der „Specular"-Beleuchtungswert soll spiegelnde, besonders helle Punkte einer Oberfläche betonen, in denen der Winkel zum Licht und zum Betrachter von der Oberflächennormale relativ ähnlich sind. Zur Berechnung wird die optimale Reflektionsachse bestimmt und mit der realen Reflektionsachse (der Oberflächennormale) verglichen.

$$Reflektionsachse_{optimal} = Richtung_{licht} + Richtung_{betrachter}$$

Je näher sich diese Werte sind (also je größer das Skalarprodukt zwischen den beiden Werten), desto höher ist der Specular-Beleuchtungswert für diesen Punkt. Dieses Skalarprodukt wird dann noch mit einem Wert „shinyness" potenziert. Durch einen höheren „shinyness"-Wert erhält man eine Fläche die glänzender, bzw. Spiegelnder erscheint, während ein niedriger Wert zu einer eher matten Oberfläche führt.

$$Beleuchtung_{specular} = Objekt_{specular} * Lichtquelle_{specular} * \left(Reflektionsachse_{optimal} \cdot Reflektionsachse_{real}\right)^{shinyness}$$

Abbildung 3 zeigt das Bild nur mit „Specular“-Beleuchtung. Besonders an der Lichtquelle kann man die „harten“ Reflektionen gut sehen.

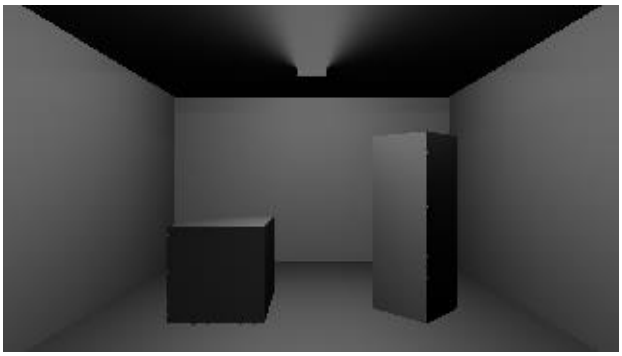


Abbildung 3: Bild mit „Specular“-Beleuchtung

1.2.4 Zusammensetzung der Beleuchtungswerte

Für jeden Punkt werden die drei Werte addiert. Dadurch ergibt sich das Bild in Abbildung 4. Durch das Anpassen von den Parametern kann das Bild entsprechend verändert werden.[1]

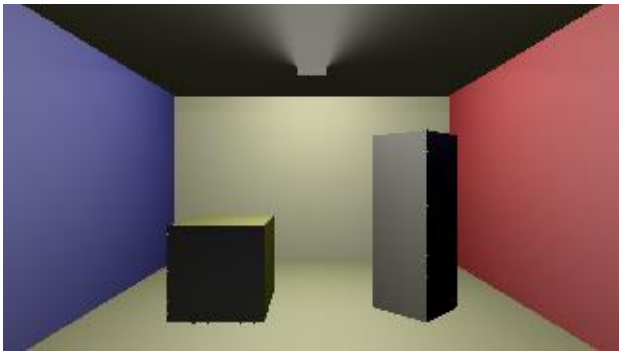


Abbildung 4: Zusammengesetztes Bild

[1] Vgl. (Phong, 1975) und https://medium.com/swlh/ray-tracing-from-scratch-in-python-41670e6a96f9

2 Algorithmische Umsetzung

2.1 Kollisionserkennung von Rays mit Objekten

2.2 Quadratische Lichtquelle

Da eine quadratische Lichtquelle gefordert ist, aber die in Kapitel 1 beschriebene Phong-Methode nur mit eindimensionalen Lichtquellen arbeitet, stellt das Berechnen der Beleuchtung eines Punktes ein besonderes Problem dar. Zusätzlich ist der einfache Ansatz zur Schattenbestimmung eines Punktes durch Werfen eines Strahls zur (punktförmigen) Lichtquelle, bedingt durch die zweidimensionale Ausdehnung der Lichtquelle, nicht möglich. Um weiche Schatten zu erhalten muss also ein anderer Ansatz gewählt werden.

2.2.1 Berechnung der Schatten

Zur Berechnung der Schatten wird ein primitiver Algorithmus verwendet, der die Schattenbestimmung von einer eindimensionalen Lichtquelle auf eine mehrdimensionale Lichtquelle portiert. Statt zu überprüfen, ob zwischen zwei Punkten ein Objekt liegt, wird überprüft wie viele Strahlen ungestört von einem Punkt auf dem Objekt zu mehreren anderen Punkten auf dem Licht reichen. Es werden also von dem zu prüfenden Punkt mehrere „Schattenfühler“ (Shadow Rays) zur Lichtquelle geschickt und aus dem Anteil der angekommen Fühler ein Schattenwert für diesen Punkt berechnet.
TODO: Skizze mit Zeichentablet einfügen
Diese Methode approximiert die Lichtfläche nur durch mehrere Punkte und stellt daher kein perfektes Ergebnis dar und nur stufenförmige Schatten (siehe Abbildung 6). Die Punkte, durch die die Lichtquelle repräsentiert wird, können dabei systematisch(siehe Abbildung 5 bis Abbildung 7), zufällig(siehe Abbildung 8**Fehler! Verweisquelle konnte nicht gefunden werden.**) oder gemischt(siehe Abbildung 9) gewählt werden. Für den finalen Render wurden jedoch nur systematische Shadow Rays benutzt, da diese für ein schöneres Bild sorgen und die ungenutzten zufälligen Shadow Rays später zu Fehlern im Bild geführt haben (Siehe Abbildung 8 und Abbildung 9).

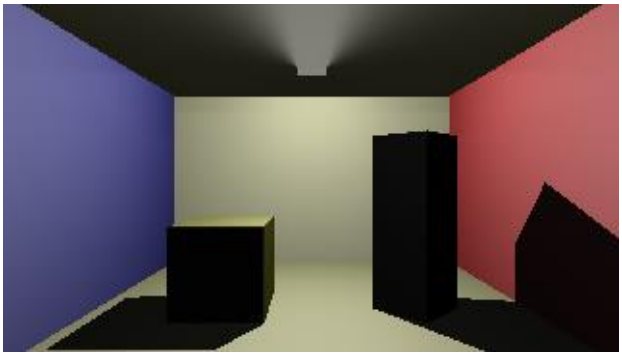


Abbildung 5: Bild mit 1 systematischen Shadow Ray

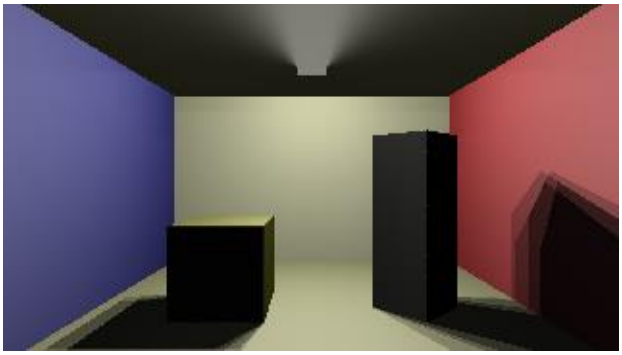


Abbildung 6: Bild mit 4 systematischen Shadow Rays

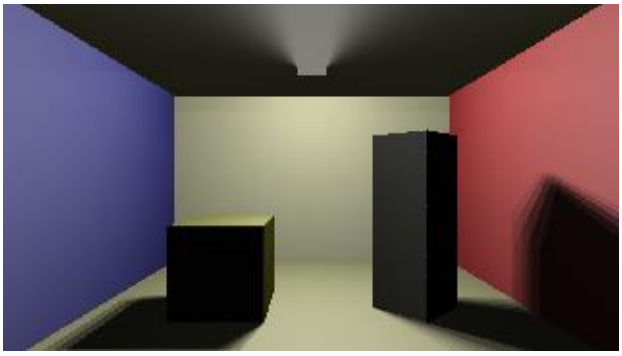


Abbildung 7: Bild mit 9 systematischen Shadow Rays

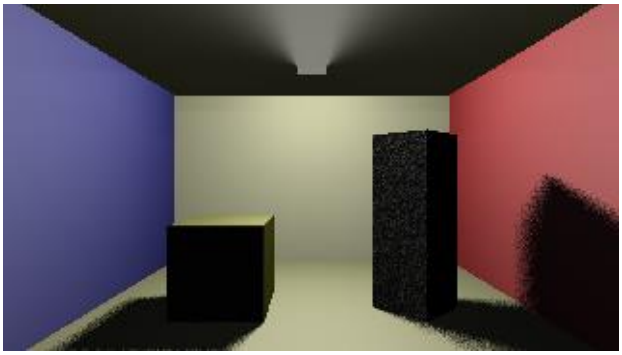


Abbildung 8: Bild mit 8 zufälligen Shadow Rays

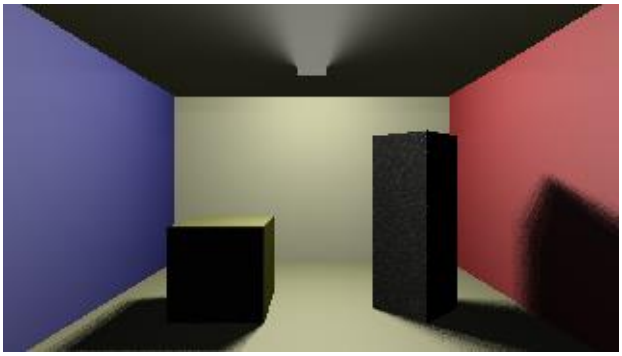


Abbildung 9: Bild mit 9 systematischen und 8 zufälligen Shadow Rays

2.2.2 Berechnung der Beleuchtung

Bei der Berechnung der Beleuchtung für eine quadratische Lichtquelle nach dem Phong-Modell ist nur der „Diffuse“- und „Specular“-Wert wichtig, da nur diese von der Position des Lichts abhängen. Für die Berechnung dieser Werte wird daher der für den Punkt auf dem Objekt der optimale Punkt auf der Lichtfläche zur Berechnung verwendet.

Bei der Berechnung des Specular-Wertes wird anhand der realen Reflektionsachse (dem Normalenvektor der Oberfläche) ein reflektierter Ray von der Kamera aus gebildet. Wie weit man diesem Ray folgen muss um auf der Lichtebene zu landen wird mit folgender Formel berechnet:

$$n = \frac{Normal_{licht} \cdot Stütze_{Licht} - Normal_{licht} \cdot Quelle_{ay}}{Normal_{licht} \cdot Richtung_{ray}}$$

Wenn der Ray parallel zur der Lichtebene ist ($Normal_{licht} \cdot Richtung_{ray} = 0$) wird zu der Richtung des Rays ein kleiner Offset hinzugefügt, um den Schnittpunkt nahe Unendlich zu legen. Der herausgefundene Schnittpunkt wird anschließend durch eine Basisänderung so vom dreidimensionalen in den zweidimensionalen Raum projiziert, so dass ein Wert im Bereich $0 <= x, y <= 1$ einem Punkt auf der Lichtebene innerhalb des Lichts entspricht (Für die Transformation wird die Pseudoinverse einer Matrix, die aus den aufspannenden Vektoren des Lichts besteht, benutzt). Ist n positiv wird der zu dem Schnittpunkt nächste Wert im Bereich $0 <= x, y <= 1$ rücktransformiert, wenn n negativ ist, wird der am weitesten entfernteste Punkt rücktransformiert. Dieser rücktransformierte, nun wieder im dreidimensionalen Raum liegende, Punkt wird dann für die Berechnung der optimalen Lichtachse verwendet.

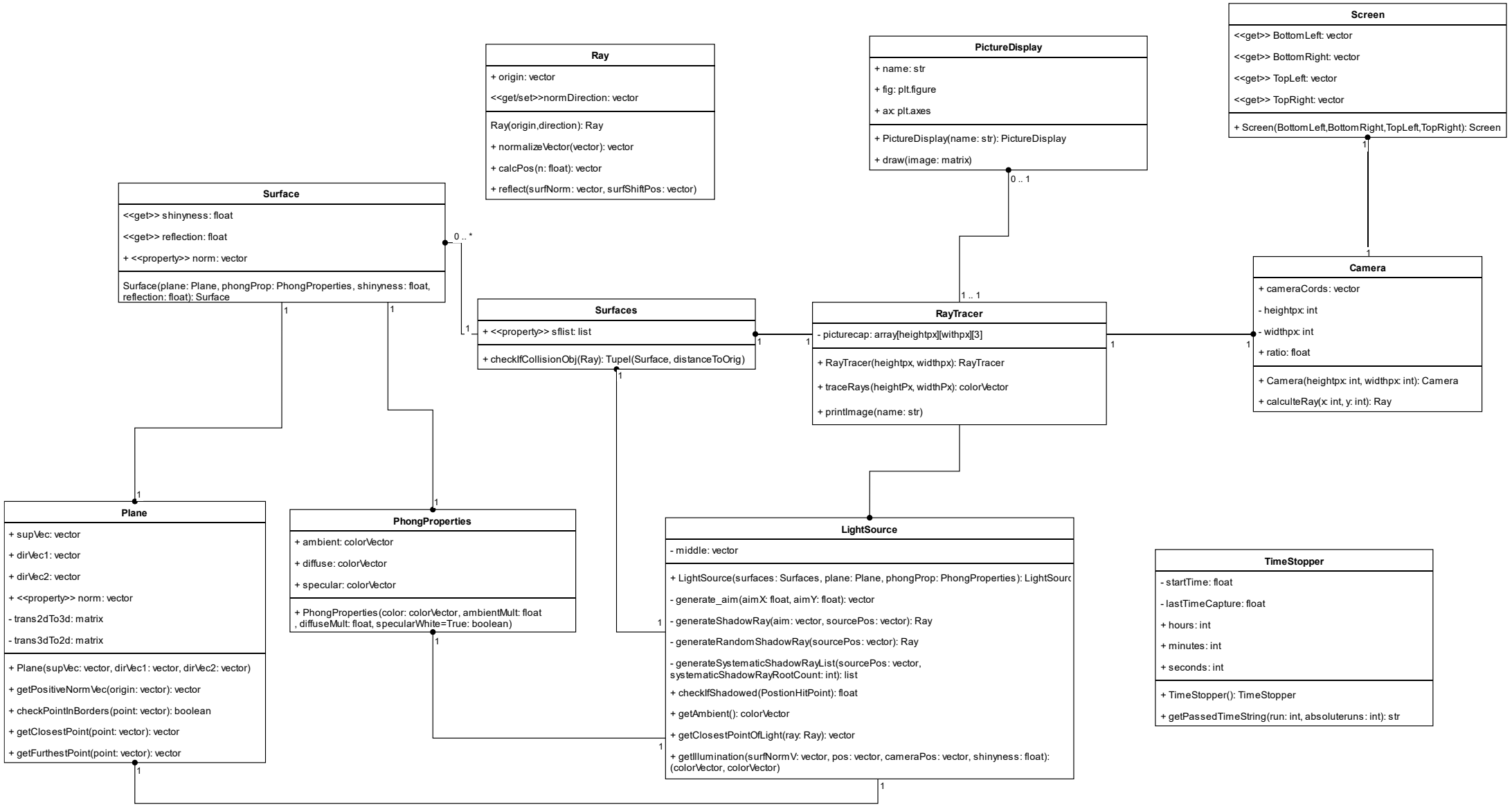
Die Berechnung für die optimale Position des Lichts für den diffusen Wert verläuft ähnlich, nur wird hier als initialer Ray nicht der reflektierte Vektor von der Kamera verwendet, sondern der Normalenvektor der Oberfläche.
//TODO: ganz viele skizzen einfügen

3 Implementierung

3.1 Implementierung des 3-dimensionalen Raums

3.2 Funktionen

3.3 Klassenmodell



3.4

3.5 Mögliche Optimierungen

Neben der leichten Erweiterbarkeit des Programms durch seinen objektorientierten Aufbau, kann man das Programm auch noch abwandeln, um eine schnellere Berechnung zu erreichen. Zusätzlich kann man das bilderzeugende Verfahren anpassen, um realistischere oder interessantere Bilder zu bekommen.

3.5.1 Performance

Das implementierte Programm ist denkbar ineffizient. So braucht die Berechnung eines Bildes in Full-HD (1080*1920px) mit 64 Shadow Rays und einer maximalen Raytracing-Tiefe von 4 (also maximal 256 Shadow Rays pro Pixel) etwa 15 Stunden auf einer modernen CPU. Dies liegt daran, dass Python als interpretierte Sprache langsamer ist als eine Sprache die als Maschinencode ausgeführt wird, wie etwa C.
Trotzdem kann man durch konsequentes Nutzen von in C implementieren Modulen von Python, wie etwa Numpy, und das Verwenden von in C umgesetzten builtins von Python, wie der map() Befehl. Durch die Ersetzung von einigen Schleifen durch map() konnte die Laufzeit um fast ein Zehntel reduziert werden.
Neben der effizienteren Ausnutzung der vorhandenen Ressourcen kann man die vorhandenen Ressourcen erhöhen, indem man mehr Prozessorkerne zur Berechnung einsetzt. Durch den Einsatz des Python Moduls „multiprocessing“ (Standard Python Modul) konnte die Berechnungszeit auf einem System mit vier Kernen ca. auf ein Drittel reduziert werden.

Berechnungsverfahren	Zeit
Single Process	~15:00
Multi Process	~6:20

Tabelle 1: Vergleich der Rechenzeiten eines 480*853px Bildes mit 16 Shadow Rays

Da multiprocessing kein erlaubtes Modul ist, gibt es im Github einen extra Branch der dieses Modul verwendet.

3.5.2 Bildgebung

Quellenverzeichnis

Phong, B. T. (1975). *Illumination for Computer Generated Pictures.*