Dokumentation zur Hausarbeit im Fach Computergrafik: Raytracer

Matr. Nr. 9431638

25.01.2025

1 Beschreibung der Implementierung

Die vorliegende Hausarbeit dokumentiert die Implementierung eines einfachen Raytracers, der grundlegende Prinzipien der Strahlverfolgung zur Bildsynthese verwendet. Der Raytracer ist in Python 3.12.2 geschrieben und nutzt die Bibliotheken PyOpenGL, NumPy und GLFW, um die Funktionalität von Rendering und Interaktion zu realisieren.

PyOpenGL dient als Schnittstelle zur OpenGL-API, die für die GPU-beschleunigte Grafikverarbeitung verwendet wird. NumPy wird zur effizienten Verarbeitung numerischer Daten eingesetzt, insbesondere für Vektor- und Matrixoperationen. GLFW übernimmt das Management des Fensters und der Benutzereingaben, einschließlich der Steuerung mittels Tastatur und Maus.

1.1 Installation und Ausführung

Zur Ausführung des Raytracers sind die folgenden Schritte erforderlich:

1. Abhängigkeiten installieren:

```
$ pip install -r requirements.txt
```

2. Das Programm kann mit dem folgenden Befehl gestartet werden:

```
$ python main.py
```

Nach dem Start öffnet sich ein Fenster, in dem die Szene mit mehreren beleuchteten Kugeln und einer Ebene gerendert wird.

1.2 Steuerung

Der Benutzer kann sich innerhalb der Szene mittels folgender Tasten bewegen:
W, A, S, D – Bewegung entlang der X- und Z-Achse.
Leertaste – Aufwärtsbewegung entlang der Y-Achse.
Shift (Umschalttaste) – Abwärtsbewegung entlang der Y-Achse.
Mausbewegung – Drehung der Kamera entlang der Yaw- und Pitch-Achse.

1.3 Beispiel-Screenshot

Nachfolgend ist ein Standbild der gerenderten Szene dargestellt:

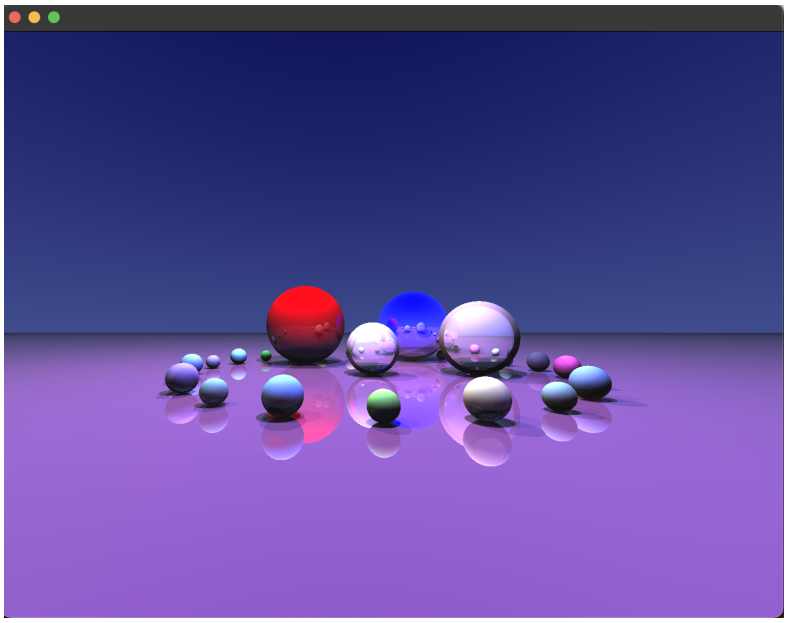

Abbildung 1: Beispiel-Screenshot

Der Raytracer unterstützt grundlegende Materialeigenschaften und Beleuchtungseffekte, darunter direkte Beleuchtung, Spiegelungen und Brechungen. Um die Rahmen der Arbeit nicht zu sprengen, wurden nur einfache Objekte wie Kugeln implementiert. Weitere Details zu den theoretischen Hintergründen und der Implementierung werden in den nachfolgenden Kapiteln behandelt.

2 Theoretische Grundlagen

Um diesen Raytracer zu implementieren, wurden folgende theoretische Grundlagen verwendet:

2.1 Strahlengleichung

Die Strahlengleichung ist eine fundamentale Formel, die zur Implementierung eines Raytracers verwendet wird. Sie beschreibt die Bewegung eines Lichtstrahls als eine unendliche Linie im 3D-Raum und stellt die Grundlage für die Berechnung von Schnittpunkten zwischen dem Strahl und geometrischen Objekten dar. Die Strahlengleichung wird allgemein wie folgt definiert:

$$\vec{P}(t) = \vec{O} + t\vec{D} \tag{1}$$

Hierbei bedeuten:
$\vec{P}(t)$: Die Position eines Punktes entlang des Strahls als Funktion des Parameters t, wobei t den Abstand vom Ursprung des Strahls angibt,
$\vec{O}$: Der Ursprung des Strahls, also der Ausgangspunkt, von dem der Strahl emittiert wird,
$\vec{D}$: Die Richtung des Strahls, dargestellt als ein normalisierter Vektor. Ein Vektor ist normalisiert, wenn seine Länge $|D|$ gleich 1 beträgt, also $|\vec{D}| = 1$. Dies wird durch $\vec{D}_{norm} = \frac{\vec{D}}{|\vec{D}|}$ berechnet,
t: Ein skalärer Parameter, der bestimmt, wie weit entlang der Richtung $\vec{D}$ der Punkt $\vec{P}(t)$ vom Ursprung $\vec{O}$ entfernt ist.

2.2 Strahl-Schnittberechnungen mit Kugeln

Die Hauptobjekten im vorliegenden Raytracer sind die Kugeln. Die implizite Kugelgleichung lautet:

$$|\vec{P} - \vec{C}|^2 = R^2 \tag{2}$$

Hierbei bedeuten:
$\vec{P}$: ein Punkt auf der Kugeloberfläche, $\vec{C}$: Mittelpunkt und R: Radius.

Setzt man die Strahlengleichung 1 in die Kugelgleichung, bekommt man eine Schnittbedingung:

$$|\vec{O} + t\vec{D} - \vec{C}|^2 = R^2 \tag{3}$$

Die euklidische Norm kann durch das Skalarprodukt ausgedrückt werden, sodass sich nach Ausmultiplizieren folgende quadratische Gleichung ergibt:

$$(\vec{D} \cdot \vec{D})t^2 + 2(\vec{D} \cdot (\vec{O} - \vec{C}))t + (\vec{O} - \vec{C}) \cdot (\vec{O} - \vec{C}) - R^2 = 0. \tag{4}$$

Die Anzahl der Schnittpunkte zwischen dem Strahl und der Kugel wird durch die Diskriminante Δ der quadratischen Gleichung bestimmt. Falls $\Delta < 0$, existiert keine reelle Lösung, was bedeutet, dass der Strahl die Kugel nicht schneidet. Falls $\Delta = 0$, gibt es genau einen Schnittpunkt, d. h., der Strahl tangiert die Kugel. Falls $\Delta > 0$, gibt es zwei Schnittpunkte, was bedeutet, dass der Strahl die Kugel an zwei Stellen durchdringt.

2.3 Gerade-Ebene Schnittpunkt

Die allgemeine Gleichung für eine Ebene im Normalenform lautet:

$$\vec{N} \cdot (\vec{P} - \vec{P}_0) = 0, \tag{5}$$

wobei $\vec{N}$ ein Normalenvektor der Ebene ist und $\vec{P}$ ein beliebiger Punkt auf der Ebene. Setzt man die Strahlengleichung in die Ebenengleichung ein, ergibt sich:

$$\vec{N} \cdot (\vec{O} + t\vec{D} - \vec{P}_0) = 0. \tag{6}$$

Aufgelöst nach t:

$$t = \frac{\vec{N} \cdot (\vec{P_0} - \vec{O})}{\vec{N} \cdot \vec{D}}. \tag{7}$$

Falls der Nenner $\vec{N} \cdot \vec{D} = 0$ ist, verläuft der Strahl parallel zur Ebene und es existieren entweder keine oder unendlich viele Schnittpunkte. In diesem Fall gibt es keine eindeutige Lösung für t.

2.4 Fresnel-Schlick-Approximation

Die Fresnel-Schlick-Approximation beschreibt das Reflexionsverhalten von Licht an der Grenzfläche zweier Medien mit unterschiedlichen Brechungsindizes.

Die Fresnel-Gleichungen für Reflexion und Transmission:

$$R_s = \left(\frac{n_2 \cos\theta_i - n_1 \cos\theta_t}{n_2 \cos\theta_i + n_1 \cos\theta_t} \right)^2 \tag{8}$$

$$R_p = \left(\frac{n_1 \cos\theta_i - n_2 \cos\theta_t}{n_1 \cos\theta_i + n_2 \cos\theta_t} \right)^2 \tag{9}$$

Der durchschnittliche Reflexionskoeffizient ergibt sich aus:

$$R = \frac{R_s + R_p}{2}. \tag{10}$$

Dabei sind: n_1 und n_2 die Brechungsindizes der Medien, θ_i der Einfallswinkel, θ_t der Brechungswinkel gemäß Snelliusschem Gesetz.

Falls der Einfallswinkel θ_i den kritischen Winkel überschreitet, tritt Totalreflexion auf, und der gesamte Lichtanteil wird reflektiert. Dies geschieht, wenn:

$$\sin\theta_t > 1 \tag{11}$$

2.5 Beer-Lambert-Gesetz

Das Beer-Lambert-Gesetz beschreibt die Abschwächung (Attenuation) von Lichtintensität, wenn Licht durch ein absorbierendes Medium propagiert. Die Lichtintensität I nach einer bestimmten Strecke d in einem Medium mit Absorptionskoeffizient α folgt der exponentiellen Abschwächung:

$$I = I_0 \cdot e^{-\alpha d} \tag{12}$$

wobei: I_0 die ursprüngliche Lichtintensität ist, α der Absorptionskoeffizient des Mediums, d die zurückgelegte Strecke im Medium, $e^{-\alpha d}$ der Dämpfungsfaktor durch Absorption ist.

3 Algorithmen und Umsetzung

Dieses Kapitel beschreibt die Umsetzung des Raytracers, benutzte Algorithmen und dessen Architektur.

3.1 Architektur

Die Architektur des Raytracers folgt einem Client-Server-Modell, bei dem die CPU (Client) Steuerungsaufgaben übernimmt und die GPU (Server) die rechenintensive Bildsynthese durchführt. Die Abbildung 2 verdeutlicht die hierarchische Beziehung zwischen den Komponenten und ihre Ablaufreihenfolge.

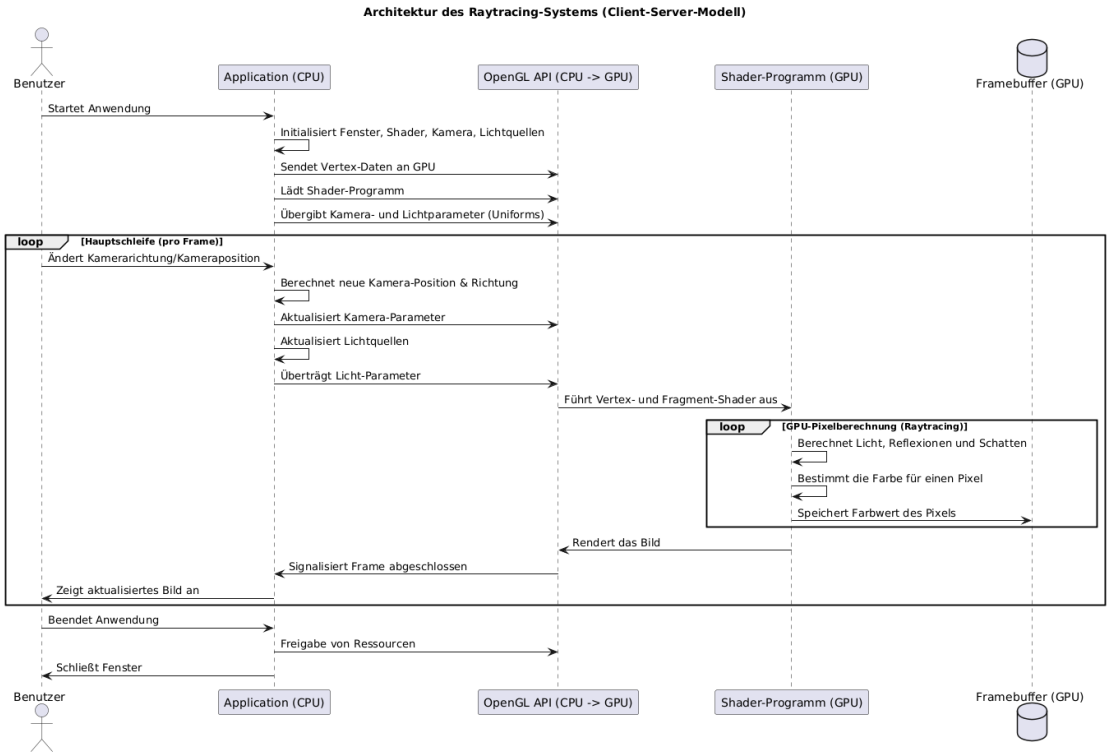

Abbildung 2: Sequenzmodell der Architektur: CPU (Anwendungslogik) und GPU (Raytracing-Kern)

Die Datei *main.py* ist der Einstiegspunkt des Programms. Sie erstellt eine Instanz der Klasse *Application*. Diese Klasse übernimmt die zentrale Steuerung des Programms. Sie integriert alle Komponenten, wie zum Beispiel die Lichtquellen und die Kamera. In der *Application*-Klasse werden der Startpunkt und die Richtung der Kamera sowie die Positionen der Lichtquellen und deren Farbe definiert. Diese Komponenten werden durch Callbacks in jedem Frame aktualisiert.

Der Hauptablauf wird in der *Application*-Klasse verwaltet. Innerhalb dieses Hauptloops werden die Eingabedaten von der Kamera verarbeitet (wie zum Beispiel die aktualisierte

Position und Blickrichtung), ebenso wie die Daten der Lichtquellen, von denen einige rotieren. Diese Daten werden über die OpenGL-API an den GPU-Prozessor weitergegeben. Dazu wird die *glGetUniformLocation*-Methode verwendet, um die Position der Uniform-Variablen im Shader bei der Initialisierung zu erhalten, und die *glUniform*-Methoden, um die Werte an die Shader zu übermitteln. Hier endet der Teil des Programms, der auf der CPU ausgeführt wird. Ab diesem Punkt erfolgt die weitere Verarbeitung auf der Grafikkarte. Der Code, der auf der GPU ausgeführt wird, ist hauptsächlich der Shader-Code, der die eigentliche Berechnung und Darstellung der Grafik übernimmt.

CPU-seitige Verarbeitung:
Die *Application*-Klasse ist für die Verwaltung des gesamten Programms zuständig. Sie ist verantwortlich für die Initialisierung der Fenster, das Erstellen von Buffern und das Laden der Shader. Die *process_input*-Methode verarbeitet Benutzereingaben wie Tastatureingaben und Mausbewegungen, die die Kamera steuern.

OpenGL-API (Uniform-Variablen und Shader):
Die GPU wird über OpenGL mit den aktualisierten Werten für die Kamera und Lichtquellen versorgt. Die *glGetUniformLocation*-Funktion sucht nach den Speicheradressen der Uniform-Variablen im Shader. Mit *glUniform*-Funktionen werden die entsprechenden Werte an die Shader weitergegeben.

GPU-seitige Verarbeitung (Shader-Code):
Der eigentliche Rendering-Prozess, der die Grafik darstellt, erfolgt im Shader-Code. Hier wird die Geometrie gerendert und das Licht auf die Objekte angewendet. In diesem Raytracing-System wird der Vertex Shader nicht für die Transformation komplexer 3D-Modelle genutzt, sondern hat eine minimalistische Aufgabe: Er sorgt dafür, dass ein Fullscreen-Quad (ein Rechteck, das den gesamten Bildschirm abdeckt) korrekt dargestellt wird. Dazu nimmt der Shader 2D-Vertex-Koordinaten als Eingabe und setzt die z-Koordinate auf 0.0, sodass das Quad in der Bildebene bleibt. Die eigentliche Raytracing-Berechnung erfolgt anschließend im Fragment Shader, der für jeden Pixel Licht, Schatten und Reflexionen berechnet.

3.2 Algorithmus

Der Raytracing-Algorithmus verfolgt Lichtstrahlen durch die Szene und simuliert deren Wechselwirkungen mit Objekten. Dabei werden Reflexion, Brechung, Schatten und Beleuchtung berücksichtigt.

Jeder Pixel des Bildes repräsentiert einen Primärstrahl, der von der Kameraposition $\vec{r_o}$ (Ray Origin) in eine berechnete Richtung $\vec{r_d}$ (Ray Direction) ausgesendet wird. Die Strahlrichtung wird mithilfe einer Lochkamera mit einstellbarem Blickwinkel (FOV) abgeleitet:

```
rd = normalize(camera_dir + uv x focal)
```

Hierbei wird der Parameter $focal = tan(\frac{FOV}{2})$ zur Perspektivprojektion verwendet.

Für jeden Primärstrahl wird der nächstgelegene Schnittpunkt mit der Szene bestimmt. Die Schnittberechnung erfolgt für verschiedene Geometrien, wobei Kugeln durch die Lösung der quadratischen Gleichung bestimmt werden, die sich aus der Einsetzung der

Strahlengleichung in die Kugelgleichung ergibt. Die Berechnung für Ebenen erfolgt durch Einsetzen der Strahlengleichung in die Ebenengleichung und Lösen nach t. Falls kein Schnittpunkt existiert, wird der Himmelshintergrund gerendert, dessen Farbe sich dynamisch anhand von Lichtverhältnissen und Umgebungseffekten anpassen kann.

Trifft der Strahl auf eine Oberfläche, wird die Farbe aus direkter Beleuchtung und Materialeigenschaften berechnet. Die Lichtquellen werden auf ihre Sichtbarkeit überprüft, indem geprüft wird, ob Objekte den Lichtweg blockieren, was für die Schattenberechnung wichtig ist. Die Beleuchtung setzt sich aus ambientem Licht, diffuser Reflexion nach dem Lambert'schen Reflexionsmodell und spiegelnder Reflexion basierend auf der Fresnel-Schlick-Approximation zusammen. Weiche Schatten entstehen durch die Verteilung der Lichtquellen, während globale Beleuchtung Lichtstrahlen berücksichtigt, die von verschiedenen Oberflächen reflektiert werden und die Helligkeit der Szene beeinflussen.

Reflektierende und transparente Materialien erfordern zusätzliche Strahlen, wobei sekundäre Strahlen für Reflexion und Brechung berechnet werden. Ein reflektierter Strahl entsteht nach dem Reflexionsgesetz. Falls das Material transparent ist, wird die Brechungsrichtung über das Snelliussche Gesetz berechnet, während Totalreflexion mit der Fresnel-Gleichung überprüft wird. Die Strahlverfolgung erfolgt rekursiv bis zur maximalen Anzahl an Reflexionen *maxBounces*.

Die berechneten Lichtanteile jeder Reflexion oder Brechung werden entlang des Strahlpfades akkumuliert. Das Beer-Lambert-Gesetz wird verwendet, um Lichtabsorption innerhalb transparenter Materialien wie Glas zu simulieren. Zudem könnte ein Monte-Carlo-Sampling-Verfahren genutzt werden, um eine genauere Farb- und Helligkeitsberechnung durch stochastische Methoden umzusetzen. Am Ende der Berechnung ergibt sich die endgültige Pixel-Farbe aus der Summe der gesammelten Lichtanteile. Dieses Algorithmus wird dann (für das Fenster mit der Auflösung 1200×800) 960.000 mal für jeden Pixel innerhalb eines Frames durchgeführt.

3.3 Datenstrukturen

Zur Verwaltung der Objekte in der Szene wurden verschiedene Datenstrukturen verwendet. Die folgende Abbildung 3 zeigt die verwendeten Structs.

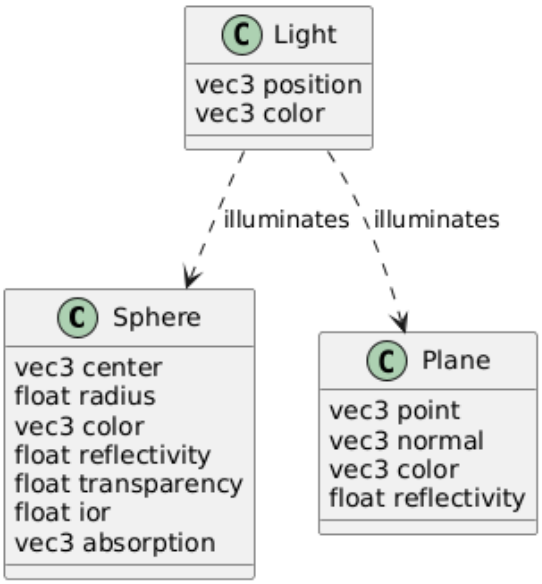

Abbildung 3: Datenstrukturen

Die Struktur Sphere speichert beispielsweise die Position, den Radius, die Farbe und Reflexions-, Transparenz- Absorptions- sowie IOR-eigenschaften einer Kugel. Die Struktur Plane beschreibt ebene Fläche mit einer Position und Normalenrichtung. Lichtquellen werden durch Light definiert, die ihre Position und Farbe speichern, die durch Pythonprogramm aktualisiert werden.