Übung 1 - Computergrafik-I, WS 2015/16

Christoph Stumpe, Fabian Wendland, Martin Zier
Beuth Hochschule für Technik Berlin

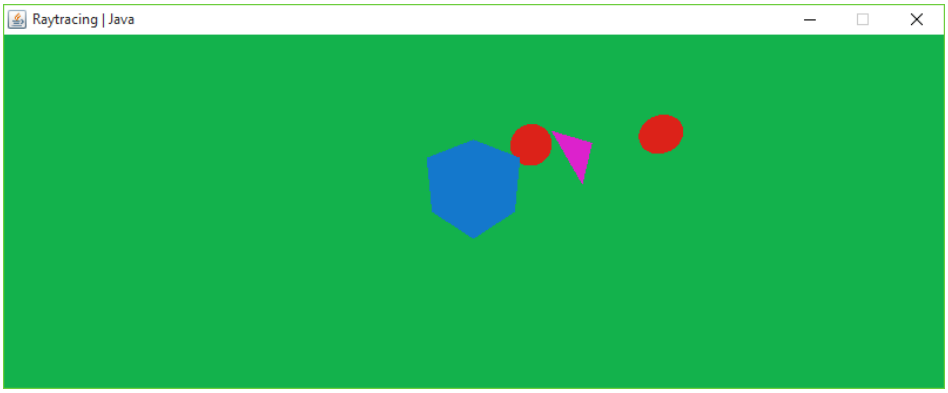

Abbildung 1: *Übungsinhalt der zweiten Aufgabe*

Inhaltsverzeichnis

1 Einführung

Die Aufgaben der zweiten Übung umfassen die ersten Darstellungen von Bildern und rendern von simplen Objekten, darunter Ebenen, Kugeln, Dreiecke und eine Axis-Aligned Box. Dabei wird das gesamte Grundprogramm des Raytracers geschrieben und auf die Vektorbibliothek aus der ersten Übung zurückgegriffen.

2 Aufgabenstellung

Alle dargestellten Aufgaben beziehen sich auf die Implementierung mit Java sofern nicht anders gekennzeichnet.

2.1 Strahlengenerierung

Das Objekt `Ray` repräsentiert einen ausgesendeten Strahl, der von der gewählten Kamera ausgeht. Es kann daraus erechnet werden, an welchem Punkt im Raum sich der Strahl befindet für ein bestimmtes `t` oder wie groß das `t` an einem bestimmten Punkt ist.

2.2 Kameras

Es soll eine abstrakte Klasse `Camera` erstellt werden, die bereits Informationen für eine `OrthographicCamera` und eine `PerspectiveCamera` beinhaltet. Diese zwei erbenden Klassen erweitern die Superklasse um die benötigten Klassenattribute und können `Rays` generieren.

2.3 Farben

Die Klasse `Color` soll eine RGB-Farbe mit separaten `double`-Werten repräsentieren, wobei jeder Farbkanal `>= 0` sein muss. Diese Farben können miteinander addiert, subtrahiert und multipliziert werden, welches für spätere Übungen verwendet wird.

2.4 Geometrie

Für die Geometrie wird eine Superklasse `Geometry` erstellt, aus der `Plane`, `AxisAlignedBox`, `Triangle` und `Sphere` erben. Die Geometrie-Objekte können ein neues, separates `Hit`-Objekt zurückliefern, wenn ein ausgesenderter `Ray` auf das Objekt trifft.

2.5 Welt

Eine neue Klasse `World` beinhaltet alle geometrischen Informationen und eine Hintergrundfarbe. Diese Hintergrundfarbe wird benutzt, wenn ein Strahl kein Objekt trifft.

2.6 Raytracer

Das Herzstück der Anwendung ist der `Raytracer`, der über alle Pixel der zu renderenden Ebene iteriert und die Berechnung der Farbwerte einer Szenerie durchführt.

2.7 OpenCL Implementierung

In dem vorherigen Papier zur Übung 1 wurde eine Python-Implementierung vorgeschlagen. Unsere Gruppe hat sich danach entschieden den Raytracer in OpenCL zu implementieren. Die Vorbereitungen für ein Rendering in OpenCL werden bereits getroffen.

3 Lösungsstrategien

Da die Komplexität des Problems nur sehr klein ist und die benötigte Implementierung mehr eine Fleißaufgabe (oder Tippaufgabe) ist, hat einer die Tracer-Applikation geschrieben, einer geprüft und der letzte bereits einen C++-Host und OpenCL-Kernel angelegt.

4 Implementierung und Bearbeitungszeit

Da es bereits eine sehr strikte systematische Planung vorgegeben wurde, musste keine große Vorplanung der Implementierung getroffen werden.

4.1 Strahlengenerierung

Die Strahlengenerierung erwies sich als sehr einfach.

Bearbeitungszeit: 0,5 - 1 Stunde

4.2 Kameras

Durch unentdeckte Fehler der Vektorenbibliothek hat das Debugging der Kameras viel Zeit benötigt. Diese Fehler hätten mit intensiveren Prüfungen im Vorfeld der ersten Übung vermieden werden können.

Bearbeitungszeit: 2 - 3 Stunden

4.3 Farben

Die Farbberechnung ist mit unter den einfachsten Aufgaben der Übung gewesen.

Bearbeitungszeit: 1 Stunde

4.4 Geometrie

Da die Geometrien umfangreich sind und sich vor allem bei der Axis-Aligned Box einige Probleme aufgetreten sind, erwies sich die Bearbeitung der Geometrien weitaus länger als gedacht. Durch den Hinweis der Gleitkomma-Fehlerberechnung wurden die Probleme jedoch gelöst und die Fehlerbilder wurden aufgehoben.

Bearbeitungszeit: 4 Stunden

4.5 Welt

Auch die Klasse `World` ist ohne versteckte Probleme zu bearbeiten gewesen. Durch eine `ArrayList` gibt es eine erweiterbare Liste und die Verwendung von einer solchen Collection ist für den Moment vollkommen ausreichend für die Performance.

Bearbeitungszeit: 30 Minuten

4.6 Raytracer

Der Raytracer wurde im Verlauf immer wieder angepasst, da hauptsächlich die dynamische Weltveränderung Anfangs nicht möglich war.

4.7 OpenCL Implementierung

Die OpenCL Implementierung ist noch in Bearbeitung - die hauptsächlichen Problematiken sind folgende:

- **Datentransport von Host zu Kernel:** Es stellt sich heraus, dass es nur sehr schwierig ist, Daten vollständig und funktionell zu übertragen, so dass der OpenCL Kernel sinnvoll diese Daten weiter verarbeiten kann. Das Problem ist bis jetzt nicht gelöst.
- **Interoperation OpenGL & OpenCL:** Uns scheint es sinnvoll ein Fenster mit OpenGL darzustellen - dadurch wird auch die Grafikhardware optimal ausgenutzt und durch viele built-in Features von SDL (Simple DirectMedia Layer) ist die Bilddarstellung einfach und möglichst effizient.

Da beide großen Probleme noch in Bearbeitung sind und für uns C und C++ recht neu sind, wird die vollständige Implementierung beider Übungen noch für unbestimmte Zeit dauern.

5 Aufgetretene Probleme der Implementierung

Das größte Problem ist das Debugging des Raytracers. Es ist nur sehr schwierig festzustellen, wo die Fehlerquelle ihren Ursprung hat. Möglicherweise sind vorgefertigte Unit-Tests oder Grenzfall-Vorgaben sinnvoll, die so lokal wie möglich typische Problemfälle aufzeigen und die Eingrenzung des Problemursprungs hilfreich sein können.

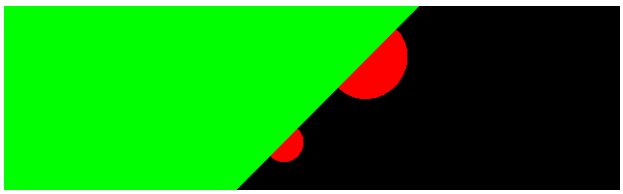

Abbildung 2: *Ein Fehler im Kreuzprodukt eines* `Vecot3s` *führte zur Verdrehung der Kamera*

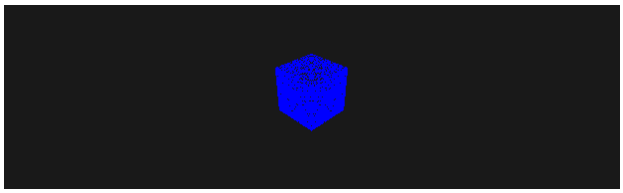

Abbildung 3: *Durch Gleitkommafehler entstand ein "Rauschenäuf der Box.*