Raytracing Aufgabe 01 - Vorbereitung

Benjamin Schurian, Clemens Pade, Robert Dimitrov
Computergraphik I

WS2015/16

1 Einführung

Diese Aufgabe dient als eine Art Vorbereitung für die spätere Entwurfsphase des Raytracers. Die Studenten sollen sich mit dem Java Image-API vertraut machen und dabei die grundlegenden Klassen und Methoden kennenlernen, die sie im Laufe des Semesters benutzen werden. Auch die in den früheren Semestern erworbenen Kenntnisse zu den Themen Lineare Algebra und Geometrie sollen bei der Aufgabe aufgefrischt werden. Außerdem werden erste praktische Erfahrungen mit der Versionsverwaltungssoftware "Git" und mit dem Textsatzsystem LaTeX gesammelt.

2 Teilaufgabe 1

In der ersten Teilaufgabe ist ein Programm zu schreiben, mit dem man Bilder von der Festplatte auswählen und öffnen kann. Das geöffnete Fenster sollte die Größe des Bildes besitzen.

Die von uns geschriebene Klasse `ImageOpen` basiert auf zwei anderen Klassen – `java.imageio.ImageIO` und `javax.swing.FileChooser`, die beide sehr benutzerfreundlich und leicht zu bedienen sind und uns daher keine Schwierigkeiten bereitet haben. Die Implementierung der ersten Teilaufgabe ist relativ straight-forward und hat nicht so viel Zeit gekostet. Länger hat das Einlesen ins Thema gedauert.

- **Teammitglied:** Benjamin
- **Zeitbedarf:** 2-3 Stunden (zusätzlich 2 Stunden für die später beschriebenen Probleme).

3 Teilaufgabe 2

Die zweite Teilaufgabe besteht in der Implementierung eines Programms, das ein Fenster mit der Größe 640x480 Pixel öffnet und auf dessen Fläche ein schwarzes Image gezeichnet wird, das einen diagonalen roten Strich beinhaltet. Dabei wird zunächst ein Bild erzeugt, dessen Pixel alle mit Schwarz initialisiert werden. Anschließend wird die Diagonale pixelweise gezeichnet. Bei Veränderung der Größe des Fensters sollte sich das Bild an die neue Größe anpassen. Außerdem soll das Bild auch als .PNG oder .JPG gespeichert werden können.

Zum Zeichnen des Bildes haben wir die Klasse `ImagePaint` geschrieben. Mit der Methode `paintImage()` dieser Klasse wird ein Bild mit den oben genannten Eigenschaften erzeugt. Die Methode `save()` dient zum Speichern des Bildes. In der Klasse `ImageTest` wird das Fenster erzeugt, in welchem das Bild angezeigt wird.

Die Klasse `ImagePaint`, die von `JPanel` erbt, besitzt eine Variable vom Typ `BufferedImage`. Diese Image wird in der Methode `paintImage()` initialisiert. In der Methode werden folgende Variablen erzeugt und initialisiert:

int w: speichert die Breite des `ImagePaint`-Objekt
int h: speichert die Höhe des `ImagePaint`-Objekt
int black: repräsentiert die Farbe schwarz als int Final
int red: repräsentiert die Farbe rot als int

`WritableRaster` raster: speichert das Raster des Bildes

`ColorModel` model: speichert das Farbmodel des Bildes

`Object` r: verweist auf ein Array mit den Farbwerten eines Pixels im jeweiligen Farbmodell

Mit der Methode `setDataElements()` wird ein Rechteck mit den übergebenen Maßen und Farbwerten gezeichnet. Anschließend wird innerhalb einer geschachtelten for-Schleife eine rote Diagonale gezeichnet.

In der Methode `save()` ist der Algorithmus zum Speichern des Bildes implementiert. Zuerst wird ein Speichern-Dialog erstellt. Anschließend wird mit einem Filter festgelegt, welche Dateitypen in dem Dialog angezeigt werden. Je nachdem welche Taste(Speichern, Abbrechen) gedrückt wurde, hält der int-Wert returnVal eine Int-Zahl. Wenn die Taste speichern gedrückt wurde wird der Pfad der angegeben Datei in einem String gehalten. Mit einer If-Bedingung wird geprüft, ob der Dateiname mit .jpg oder .png endet. Anschließend wird eine neue Datei erstellt und mit `ImageIO.write()` das Bild in die Datei geschrieben.

Die geerbte Methode `paintComponent` sorgt dafür, dass das Bild neu gezeichnet wird, wenn sich das Fenster in seiner Größe ändert.

In der Testklasse wird ein Fenster erzeugt und die Größe des Fensters festgelegt. Anschließend wird eine Menüzeile plus Menüeintrag erzeugt und hinzugefügt. Ein `ImagePainter`-Objekt und ein `ActionListener` werden erzeugt und miteinander bekannt gemacht. In der Methode `actionPerformed()` wird die Methode `imagePaint()` aufgerufen, wenn sich dieFenstergröße verändert. Anschließend wird das `ImagePainter`-Objekt auf die Fensterfläche gesetzt und das Fenster sichtbar gemacht.

Es ist uns leider nicht gelungen, ein Fenster zu erzeugen, das auch nach Aufruf der Methode `pack()` in der vorher eingestellten Größe anzuzeigen, wahrscheinlich, weil das Image nicht in das `ImagePainter`-Objekt geadded wird und somit das Objekt am Anfang die Größe null hat. Auch haben wir es nicht geschafft, das Bild an das veränderte Fenster anzupassen ohne immer wieder ein neues Image-Objekt zu initialisieren.

- **Teammitglied:** Clemens
- **Zeitbedarf:** Diese Aufgabe hat sehr viel Zeit in Anspruch genommen, da unser Wissen über die benötigten Java-Klassen zum Zeichnen und Verarbeiten von Bildern noch sehr begrenzt ist. Besonders ist die Funktionsweise der `paintComponent`-Methode schwer zu verstehen.

4 Teilaufgabe 3

In der dritten Teilaufgabe ist eine Matrizen- und Vektorenbibliothek mit den folgenden vier Klassen zu implementieren:

- `Mat3x3` - eine Matrix mit drei Zeilen und drei Spalten;
- `Vector3` - ein Vektor (x,y,z) im 3D-Raum;
- `Point3` - ein Punkt (x,y,z) im 3D-Raum;
- `Normal3` - ein Normalenvektor (x,y,z);

Diese Klassen verfügen über Methoden für die grundlegenden Operationen der linearen Algebra. So mussten z.B. Funktionen für die Matrixmultiplikation mit einem Vektor, einem Punkt und einer Matrix, die Addition und das Kreuzprodukt zweier Vektoren und das Skalarprodukt eines Vektors und einer Normalen / eines anderen Vektors usw. geschrieben werden.

Die Implementierung dieser Bibliothek ist uns nicht schwer gefallen, da es sich meistens um mathematische Formeln handelte, die man in den Handouts für Mathematik II und im Buch *"Ray Tracing from the Ground Up"* gut strukturiert und ausführlich erklärt finden konnte. Trotzdem hat diese Teilaufgabe viel Zeit und Konzentration gekostet, weil jedes Element nach der Code-Style-Guideline dokumentiert werden sollte und manche Methoden aus langen Berechnungen mit vielen Variablen bestanden, die leicht zu verwechseln sind und aus diesem Grund mehrmals überprüft werden sollen.

Zudem mussten wir bei manchen Methoden nach der richtigen Formel recherchieren. Ein Beispiel dafür ist die `reflectedOn(Normal3 n)` Methode, für die im Netz zwei fast gleiche Formeln zu finden sind, die aber zu verschiedenen Ergebnissen führen.

Zu dieser Teilaufgabe gehört auch das Vorführen einer Reihe Testberechnungen, die die vier implementierten Klassen auf Fehler prüfen. Diese Berechnungen finden in der Klasse `CalculationTests` statt.

- **Teammitglied:** Robert
- **Zeitbedarf:** ca. 6 Stunden

5 Probleme

Ein sehr großes Problem entstand, als eins der Gruppenmitglieder einen sehr unbedeutenden Merge-Konflikt hatte. Er hatte versucht, ihn mit einem speziellen Tool in der Eclipse-IDE namens "Team Synchronizing" aufzulösen. Er dachte, das Tool würde genauso funktionieren wie ein normaler Merge nach einem pull oder fetch: Merge-Konflikte werden im Code angezeigt und dann dort bearbeitet. Aber das war nicht der Fall. In "Team Synchronizing" stehen die zwei unterschiedlichen Versionen nebeneinander und man kann etwas von der origin-Version ins lokale Repo kopieren, aber keine Änderung, außer Löschen der Datei, wird seiner Erfahrung nach im lokalen Repo festgeschrieben. Am Tag dadrauf hat er einfach alle Dateien die nicht mergen konnten, gelöscht und neu von origin gepullt.

Das hat einen guten Teil seiner Zeit an diesem Tag gekostet. Da er den Tag noch etwas Dringendes erledigen musste, aber seine Commits nicht selber pushen konnte, hat er einfach alle veränderten Dateien einem Teammitglied geschickt, damit er sie, selbst wenn es der Fall sein sollte das er sie nicht pushen kann, sie sich zumindest angucken kann.

Es hat sich später herausgestellt, dass man alternativ in Eclipse die herkömmliche merge-Funktion benutzen kann.

Die Einarbeitung in LaTeX hat auch viel Zeit in Anspruch genommen, was aber nicht als "Problem" bezeichnet werden soll.

6 Quellen und Literatur

- http://docs.oracle.com/javase/8/docs/api/
- http://math.stackexchange.com
- Handouts Mathematik II - Medieninformatik Bachelor – Prof. Dr. Marlene Müller (Beuth Hochschule für Technik Berlin)
- Ray Tracing from the Ground Up – Kevin Suffern