Sommersemester 2019

Aufgabensammlung 6

Die Aufgaben werden am **10. Juli** (Mittwoch) in der Übung bewertet. Diese Aufgaben betrachteten den Softwareanalyse- und Softwaredesignprozess am Beispiel eines Ray-Tracing-Systems.

Sie können diese Aufgaben zu zweit bearbeiten und auch präsentieren.

Benutzen Sie für dieses Aufgabenblatt weiterhin das raytracer-Framework, welches Sie bereits im Aufgabenblatt 5 benutzt haben. (fork: https://github.com/vrsys/programmiersprachen-raytracer.git)
Pushen Sie die Änderungen in Ihren github-Fork:

```
git push
```

Bei Fragen und Anmerkungen schreiben Sie bitte eine Email an adrian.kreskowski@uni-weimar.de .

Aufgabe 6.1

Erklären Sie den in der Übung vorgestellten Ray-Tracing-Algorithmus. Zeichnen Sie dazu beispielhaft Strahlverläufe in die Skizze aus Abbildung 1 ein.

[5 Punkte]

Aufgabe 6.2

Modellieren Sie ein Ray-Tracing-System mit folgenden Eigenschaften:

- Die Szene kann aus **beliebig vielen geometrischen Objekten** bestehen. Jedes dieser Objekte kann eine **Kugel**, eine **Box** oder ein **Dreieck** sein und ein eigenes Material haben, welches sich jedoch mit anderen Objekten **geteilt werden können** soll.
- Die Objekte der Szene sollen in einem gemeinsamen Container verwaltet werden
- Eine Szene wird von Lichtquellen beleuchtet. In der Szene gibt es **beliebig viele diffuse** Punktlichtquellen und **eine ambiente** Grundbeleuchtung.

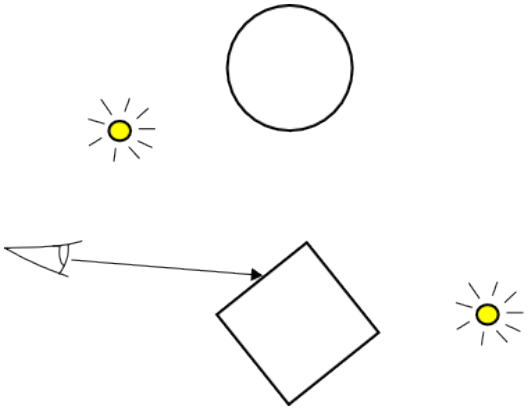

Abbildung 1: Strahlverlauf

- Die Kamera befindet sich im Ursprung und blickt entlang der negativen z-Achse. Die Kamera besitzt einen horizontalen Öffnungswinkel und eine beliebige Bildauflösung.
- Für jeden Pixel des Bildes wird ein Strahl generiert, das Ray-Tracing durchgeführt und der dadurch ermittelte Farbwert dem Pixel zugewiesen
- Das fertig gerenderte Bild wird auf dem Bildschirm ausgegeben und als Datei gespeichert.

Erstellen Sie anhand dieser informellen Beschreibung ein UML-Klassendiagramm. Überlegen Sie welche Klassen Sie benötigen und bilden Sie Verantwortlichkeiten auf Methoden und Eigenschaften als Membervariablen ab. Die Eigenschaften der entsprechenden Objekte können dem SDF-Format im Anhang entnommen werden.

[10 Punkte]

Aufgabe 6.3

Fügen Sie eine rein virtuelle Methode

```
HitPoint intersect(Ray const& ray, float& t) = 0;
```

zur Klasse `Shape` hinzu. Implementieren Sie für die Klasse `Box` die Methode `intersect`, welche die intersect-Methode der Basisklasse überschreibt. Informationen zum Schnitt einer Box mit einem Strahl finden Sie in der Vorlesung zum Ray-Tracing. Testen Sie Ihre Implementierung.

Da Sie die `intersect`-Methode für die Klasse Sphere bereits im letzten Aufgabenblatt implementiert haben, fügen Sie nun das Schlüsselwort `override` zu der bereits definierten Methode hinzu.

Committen Sie regelmäßig Ihre Fortschritte in git.

[8 Punkte]

Aufgabe 6.4

Implementieren Sie ein Struct `Material`. Es soll die Membervariablen *name* vom Typ `string`, die Materialkoeffizienten *ka*, *kd*, *ks* vom Typ `color` und den Spekularreflexionsexponenten *m* vom Typ `float` besitzen.
Implementieren Sie außerdem den Streamoperator (`operator<<`) zur Ausgabe.
Ersetzen Sie die Membervariable vom Typ *Color* in der Basisklasse Shape durch einen Member vom Typ `std::shared_ptr<Material>` und passen Sie die Interfaces dementsprechend an.

Commiten Sie Ihre Änderungen. **[5 Punkte]**

Aufgabe 6.5

Schreiben Sie eine freie Funktion, welche den Dateipfad zu einer SDF-Datei (siehe Anhang) übergeben bekommt und daraus die Elemente der Szenenbeschreibung einließt, die entsprechenden Objekte erzeugt und im Struct `Scene` in entsprechenden Container speichert.
Sie können sich für diese Aufgabe auf das Einlesen der Materialien, die von den Objekten geteilt werden sollen, beschränken.
Lesen Sie anschließend eine SDF-Datei mit folgendem Inhalt ein:

```
define material red   1 0 0 1 0 0 1 0 0 20
define material green 0 1 0 0 1 0 0 1 0 50
define material blue  0 0 1 0 0 1 0 0 1 10
```

Erzeugen Sie die entsprechenden `std::shared_ptr<Material>`-Objekte und speichern Sie diese in der Szene in einem geeigneten Container.
Speichern Sie für diese Aufgabe die `std::shared_ptr<Material>` jeweils in

- einem `std::vector<std::shared_ptr<Material>>`,
- einem `std::set<std::shared_ptr<Material>>` und
- einer `std::map<std::string, std::shared_ptr<Material>>`.

Implementieren Sie nun drei Suchfunktionen, jeweils den Namen eines Materials übergerben bekommen, und den `std::shared_ptr<Material>` des gesuchten Materials zurückgibt, falls es existiert hat, ansonsten einen `nullptr`.

Achten Sie darauf, dass Sie für die effiziente Suche auf Ihrem `std::set` mittels std::set::find(/*...*/) eine Vergleichsfunktion der Form

```
bool operator<(std::shared_ptr<Material> const& lhs,
std::shared_ptr<Material> const& rhs)
{return lhs->name < rhs->name;};
```

implementieren müssen.

Beachten Sie weiterhin, dass die `find`-Methode auf dem `std::set` ein Objekt des value-types des sets übergeben bekommt. Dementsprechend müssen Sie für eine Suche mittels `find` eine Instanz der Materialklasse erstellen. (http://www.cplusplus.com/reference/set/set/find/).

Diskutieren Sie die Rechenkomplexität der Suchfunktion und des Speicheraufwands für alle drei Varianten ausführlich!

[10 Punkte]

UML-Werkzeuge

Der Softwareentwicklungsprozess kann durch Zeichenprogramme mit UML-Diagrammelementen, durch integrierte Entwicklungsumgebungen, vom Reverse Engineering über Refactoring und Patternanwendung bis zur Codeerzeugung unterstützt werden.
Nutzen Sie z.B. eines der folgenden Programme um UML-Sequenzdiagramme zu erstellen.

- https://www.draw.io
- Umbrello UML (Linux/KDE)
- http://staruml.io
- Poseidon UML: http://www.gentleware.com/downloadcenter.html

Laden und Speichern in Poseidon UML ist auch mit der Community Edition möglich. Kommerzielle und verbreitete UML-Werkzeuge sind Borland Together und IBM Rational Rose.

Scene Description Format (SDF)

SDF ist eine einfache, zeilenorientierte, imperative Sprache zur Beschreibung einer Szene. Eine Zeile ist ein Statement und beschreibt ein Kommando vollständig mit allen notwendigen Argumenten. Alle Komponenten eines Statements sind durch Leer- und/oder Tabulatorzeichen getrennt.
Alle Zahlenangaben erfolgen in Gleitkommadarstellung. Argumente in `<>` bezeichnen einen einzelnen Wert, Argumente in `[]` bezeichnen drei Werte, z. B. die Koordinaten eines Punktes oder Vektors. Folgende Statements sind definiert:

```
# <text>
```

Kommentarzeile, d.h. die ganze Zeile wird ignoriert.

```
define <class> <name> <arg> ...
```

Allgemeine Form einer Objektdefinition. Eine neue Instanz vom Typ `<class>` wird an den Namen `<name>` gebunden und mittels der Argumente `<arg> ...` initialisiert. Objekte müssen eindeutige Namen haben!

```
define shape sphere <name> [center] <radius> <mat-name>
```

Definiert eine Kugel mit Namen `<name>`, Mittelpunkt `[center]`, Radius `<radius>` sowie dem Material `<mat-name>`.

```
define shape box <name> [p1] [p2] <mat-name>
```

Definiert einen achsenparallelen Quader mit Namen `<name>`, den sich gegenüberliegenden Eckpunkten `[p1]` und `[p2]` sowie dem Material `<mat-name>`.

`define material <name> [Ka] [Kd] [Ks] <m>`

Definiert ein Material mit Namen `<name>` und die Koeffizienten für ambiente (`[Ka]`), diffuse (`[Kd]`) und spiegelnde (`[Ks]`) Reflexion. `<m>` ist der Exponent für die spiegelnde Reflexion.

`define light <name> [pos] [color] [brightness]`

Definiert eine Lichtquelle mit Namen `<name>` an der Position `[pos]` und der Farbe `[color]`. Die Helligkeit der Lichtquelle wird mit `[brightness]` bezeichnet. Die Intensität der Lichtquelle ist dann `brightness * color`.

`ambient [ambient]`

Definiert den ambienten Term zur Beleuchtung der gesamten Szene.

`camera <name> <fov-x>`

Definiert eine Kamera mit Namen `<name>` und dem horizontalen Öffnungswinkel `<fov-x>`. Die Kamera befindet sich im Nullpunkt und blickt in Richtung der negativen z-Achse.

`render <cam-name> <filename> <x-res> <y-res>`

Erzeugt ein Bild der Szene aus Sicht der angegebenen Kamera (`<cam-name>`) und legt es in der angegebenen Datei (`<filename>`) ab. Die Auflösung des Bildes (`<x-res> <y-res>`) wird in Pixel angegeben.

Beispiel: Eine einfache Szene in SDF

```
# materials
define material red  1 0 0 1 0 0 1 0 0 1
define material blue 0 0 1 0 0 1 0 0 1 1
# geometry
define shape box    rbottom -100 -80 -200 100 80 -100 red
define shape sphere bsphere 0 0 -100 50 blue
# light - from right above
define light sun 1000 700 0 .2 .2 .2 100
# camera
define camera eye 45.0
```

```
# ... and go
render eye image.ppm 480 320
```

Bei Fragen und Anmerkungen schreiben Sie bitte eine Email an adrian.kreskowski@uni-weimar.de .