Sommersemester 2014

Aufgabensammlung 7

Die Aufgaben werden am **9. Juli** in der Übung bewertet. Diese Aufgaben betrachteten den Softwareanalyse- und Softwaredesignprozess am Beispiel eines Ray-Tracing-Systems.

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

Aufgabe 7.1

Erklären Sie den in der Übung am **2. Juli** vorgestellten Ray-Tracing-Algorithmus. Nutzen Sie dazu die Skizze und erweitern Sie diese.

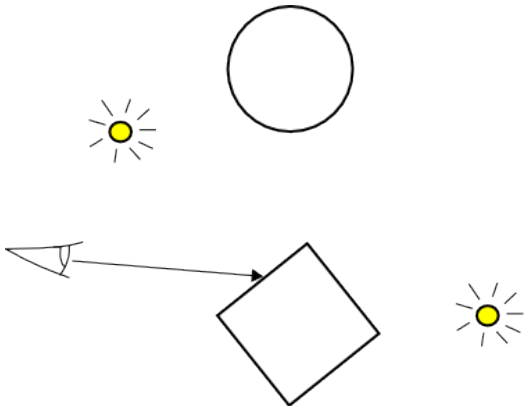

Abbildung 1: Strahlverlauf

[10 Punkte]

Aufgabe 7.2

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

- Eine im SDF-Dateiformat (siehe Anhang) gegebene Szene kann eingelesen und mittels Ray-Tracing gerendert werden.

- Die Szene kann aus beliebig vielen geometrischen Objekten bestehen. Jedes dieser Objekte kann eine Kugel, ein Quader oder ein Dreieck sein und ein eigenes Material haben.
- Eine Szene wird von Lichtquellen beleuchtet. In der Szene gibt es beliebig viele diffuse Punktlichtquellen und eine ambiente Grundbeleuchtung.
- 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 ermittelte Farbwert dem Pixel zugewiesen
- Das fertige 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 welche Verantwortlichkeiten ebenfalls als Klasse abgebildet werden sollten.

Hinweis: Verwechseln Sie Objekte nicht mit Klassen. Stellen Sie sich ggf. Personen mit jeweils einer Verantwortlichkeit vor, die die erforderlichen Aufgaben erfüllen. **[20 Punkte]**

Aufgabe 7.3

Definieren Sie die Fähigkeiten bzw. Verantwortlichkeiten ihrer Klassen und listen Sie diese auf. Diese Auflistungen dienen als Vorlage für ihre Tests.

1. window:
   - Gibt das berechnete Bild auf dem Monitor aus
2. `SDFloader`:
   - ...
3. ...

[10 Punkte]

Aufgabe 7.4

Clonen oder forken Sie das Repository https://github.com/vrsys/programmiersprachen-raytracer mit `git`.
Erstellen Sie einen Unterordner `build/` im *programmiersprachen-raytracer* Projektverzeichnis. Wechseln Sie in einem Terminal in diesen Ordner und rufen Sie `cmake ..` auf. Anschließend rufen Sie `make` auf. Dies kompiliert das Framework und Beispielprogramme. Sie können nun das Programm `raytracer` aufrufen: `./build/build/Release/raytracer`.

Fügen Sie das DTO `Ray` und die Klasse `Sphere` zum Unterordner `framework` hinzu.

```
#include <glm/glm.hpp>

struct Ray
{
  glm::vec3 origin;
  glm::vec3 direction;
};
```

Danach müssen Sie wieder `cmake` Aufrufen. Implementieren Sie für die Klasse `Sphere` die Methode `intersect`. Informationen zum Schnitt einer Kugel mit einem Strahl finden Sie auf den Seiten 20 – 22 der Vorlesung zum Raytracer (2.Juli 2014). Dokumentation zur verwendeten `glm`-Bibliothek finden Sie im `doc/` Ordner als auch im Internet. **[10 Punkte]**

Aufgabe 7.5

Implementieren Sie eine Klasse `Material`. Sie soll die Membervariablen *name* vom typ `string`, *ka*, *kd*, *ks* vom typ `color` und *m* vom typ `float` besitzen. Die Klasse soll Standard-, User-Konstruktor sowie Getter für die Attribute besitzen. Implementieren Sie außerdem den Streamoperator (`operator<<`) zur Ausgabe. **[10 Punkte]**

Aufgabe 7.6

Legen Sie eine Textdatei mit folgendem Inhalt an:

```
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
```

Lesen Sie diese Datei ein und erzeugen Sie die Materialien, die Sie in einem geeigneten Container speichern. Geben Sie den Inhalt des Containers anschließend aus. **[10 Punkte]**

Aufgabe 7.7

Konkretisieren Sie den Verantwortungsbereich `SDFloader` aus Aufgabe 7.3 mit ihren in 7.6 gewonnen Erkentnissen.

SDF file → `SDFloader` → output

Überlegen Sie sich, wie das Resultatobjekt aussehen könnte, welches der `SDFloader` erzeugt und spezifizieren Sie dieses genauer. **[5 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.

- Umbrello UML (Linux/KDE)
- http://staruml.sourceforge.net/en/download.php (Windows)
- http://download.gentleware.biz/poseidonCE-5.0.0.zip (JAVA)

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 Objekt Definition. 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] [La] [Ld]`

Definiert eine Lichtquelle mit Namen `<name>` an der Position `[pos]`. `[La]` und `[Ld]` bezeichnen den ambienten und diffusen Term der Lichtquelle. **Hinweis:** Die Summe der ambienten Terme aller definierten Lichtquellen ergibt den ambienten Term zur Beleuchtung der ganzen 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 .8 .8 .8
# camera
define camera eye 45.0
# ... and go
render eye image.ppm 480 320
```

Extended Scene Description Format (eSDF)

Aufbauend auf SDF werden folgende Erweiterungen definiert:

`define shape composite <name> <child> ...`

Definition einer Gruppe von `Shapes` mit Namen `<name>` und einer Liste von mindestens einem oder mehreren Objekten. Gruppen, wie auch Primitivobjekte, müssen eindeutige Namen haben!

`transform <object> translate [offset]`

Translation eines Objektes `<object>` um den Vektor `[offset]`.

`transform <object> rotate <angle> [vector]`

Rotation eines Objektes `<object>` um den Winkel `<angle>` im Gradmaß und den Vektor `[vector]`.

`transform <object> scale <value>`

Uniforme Skalierung eines Objektes `<object>` um den Wert `<value>`.

Beispiel: Eine einfache Szene in eSDF

```
# materials
define material red   1 0 0   1 0 0   1 0 0   10
define material blue  0 0 1   0 0 1   0 0 1   10
# geometry
define shape box    rbottom -100 -80 -200   100 80 -100 red
define shape sphere bsphere    0   0 -100   50          blue
# composite
define shape composite root rbottom bsphere
# scene xform
# transform root scale      2 2 2
transform root rotate   45  0 0 1
transform root translate    0 0 -10
# lights
define light ambient amb                 0.1  0.1  0.1
define light diffuse sun    500 800 0    1.0  1.0  1.0
define light diffuse spot1 -500 800 0    0.8  0.8  0.8
# camera
define camera eye 60.0 480 320
# camera xform
transform eye rotate -45 0 1 0
transform eye translate 100 0 100
# ... and go
render eye image.ppm
```

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