Programmiersprachen und Softwarentwicklung
Sommersemester 2014

Aufgabensammlung 8

Dieses Aufgabenblatt betrachtet den Softwareentwicklungsprozess am Beispiel einer Implementation eines Ray-Tracing-Systems. Arbeiten Sie in Gruppen von zwei Studierenden. Benutzen Sie git zur *Source-Code-Verwaltung*. Ein freies `git`-Repository können Sie sich zum Beispiel unter `https://github.com/` oder `https://bitbucket.org/` einrichten. Die Verwendung einer Source-Code-Verwaltung ist Pflicht. Ohne diese findet keine Abnahme statt!
Nutzen Sie zur Entwicklung ihres Ray-Tracing-Systems das vorgestellte Framework. Dieses ist ein bereits initialisiertes git-Repository. Vermeiden Sie die Verwendung von Raw-Pointern. Nutzen Sie die in der Vorlesung vorgestellten *Smart-Pointer* wie z.B. `std::shared_ptr`.

Bitte melden Sie sich für Konsultationstermine per Email (**andreas.bernstein@uni-weimar.de** an. Eine vorzeitige Abgabe des Ray- Tracers ist ebenfalls möglich. Bei Fragen können Sie auch jederzeit vorbeikommen oder eine Email schreiben.

Die Bewertung des Ray-Tracing-Systems erfolgt am **1. September** und am **2. September**. Melden Sie ihre Gruppe für die Bewertung der Aufgaben bis zum **28. August** per Email bei Frau Hansens (**maria-theresa.hansens@uni-weimar.de**) an. Bereiten sie zu diesem Termin eine kurze Präsentation (ca. 15 Minuten) vor, in der sie auf Schwierigkeiten, Lösungen bei der Umsetzung sowie die Arbeitsteilung innerhalb der Gruppe eingehen.

Aufgabe 8.1

Implementieren sie ein Ray-Tracing-Programm mit folgenden Eigenschaften:

- Einlesen einer Szene im SDF-Format und rendern dieser Szene,
- eine Szene kann aus beliebig vielen Objekten bestehen,
- mindestens achsenparalleler Quader, Kugeln werden unterstützt,
- jedes Objekt kann ein eigenes Material haben,
- eine Szene kann von beliebig vielen Lichtquellen beleuchtet werden; es wird ein Beleuchtungsmodell mit ambienter, diffuser und spekularer Reflexion angenommen,
- der Beobachter befindet sich im Ursprung und blickt entlang der negativen z-Achse.

Nutzen Sie die Ressourcen der vorangegangenen Aufgabenblätter, die Folien der Vorlesung zum Raytracer und beachten Sie beim Entwurf bereits die folgenden Aufgaben, insbesondere Aufgabe 7.2. Verwenden Sie die im Framework bereitgestellten Klassen. **[100 Punkte]**

Aufgabe 8.2

Implementieren Sie das in der Vorlesung vorgestellte Composite-Pattern. Dies ermöglicht eine hierarchische Szenenbeschreibung Damit kann die Szene durch einen Baum beschrieben werden, dessen innere Knoten *Composites* und dessen Blätter die primitiven Objekte (Kugeln, Quader, etc ...) sind. Erweitern Sie dazu auch ihren SdfLoader.

```
define shape box    rbottom -100 -80 -200 100 80 -100 red
define shape sphere bsphere 0 0 -100 50 blue
# composites
define shape composite root rbottom bsphere
```

[15 Punkte]

Aufgabe 8.3

Erweitern sie das Ray-Tracing-System um Translation, Skalierung und Rotation. Dazu muss die Basisklasse der `shape`-Hierarchie mit einer akkumulierten Transformationsmatrix `world_transformation` und deren Inversen (`world_transformation_inv`) als Attribute ausgestattet werden.
Vor der Schnittberechung muss die Inverse der Transformationmatrix auf den Strahl angewendet werden.
Um die Strahlposition und die Strahlrichtung mit der Matrix multiplizieren zu können, müssen Sie sie zunächst in homogene Koordinaten konvertieren. Danach führen Sie die Schnittberechung mit dem transformierten Strahl aus. Der berechnete Schnittpunkt wird dann ins Weltkoordinatensystem transformiert und wieder in einen `glm::vec3` konvertiert. Sehen Sie sich dazu S.26–S.32 in den Folien zum Ray-Tracing an.
Beachten Sie, dass Sie für die Rücktransformation der Normalen die Transponierte der Inversen benötigen.
Erweitern Sie ihren SDF-Parser um Anweisungen, die die verschiedenen Transformationen ermöglichen. Beispielsweise:

```
# geometry
define shape box    rbottom -100 -80 -200 100 80 -100 red
define shape sphere bsphere 0 0 -100 50 blue
# composites
define shape composite root rbottom bsphere
```

```
# transform name    tranformation parameter
transform   rbottom scale         2 4 2
transform   rbottom translate     3 0 2
transform   rbottom rotate        45 1 0 0
```

[40 Punkte]

Aufgabe 8.4

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 8.1 um Schatten. **[15 Punkte]**

Aufgabe 8.5

Implementieren Sie das in der Vorlesung vorgestellte Kameramodell. Erweitern Sie dazu ihren SDF-Parser um folgendes Kommando:

`camera <name> <fov-x> <eye> <dir> <up>`

Definiert eine Kamera mit Namen `<name>` und dem horizontalen Öffnungswinkel `<fov-x>` an der Position `<eye>` mit der Blickrichtung `<dir>` und dem Up-Vektor `<up>`.

[20 Punkte]

Aufgabe 8.6

Erstellen Sie eine einfache Animation.
Schreiben Sie dazu ein Program, welches für jeden Frame eine SDF-Datei generiert. Sie können zum Beispiel die Kamera um ihre Szene kreisen lassen oder Sie bewegen Objekte. Die Animation sollte mindestens 5 Sekunden lang sein und 24 Frames pro Sekunde haben.
Generieren Sie anschließend aus den SDF-Dateien mit ihrem Raytracer die Bilder ihrer Animation. Die Bilder können Sie zum Beispiel mit ffmpeg zu einem Video zusammenfügen.

```
# -r -- framerate
# -i -- input
ffmpeg -r 24 -i bild%d.png animation.mp4
```

Rendern Sie mit 24 Bildern pro Sekunde. Erstellen Sie eine mindestens 5 Sekunden lange Animation (140 Bilder). **[20 Punkte]**

Aufgabe 8.7

Erweitern Sie die `shape`-Hierarchie mit den Primitiven Dreieck, Kegel und Zylinder. **[15 Punkte, optional]**

Aufgabe 8.8

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 8.1 um Spiegelung. **[20 Punkte, optional]**

Aufgabe 8.9

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 8.1 um Refraktion.

Hinweis: Sie können dafür ihre Materialbeschreibung um einen Brechungsindex und einen Transparenzkoeffizienten (`opacity`) erweitern. **[25 Punkte, optional]**

Aufgabe 8.10

Verbessern Sie die visuelle Qualität ihres Ray-Tracing-Systems durch ein Anti-Aliasing. Verwenden Sie dabei zur Berechnung eines Pixels 4 oder mehr Subpixel und interpolieren Sie zwischen diesen. **[20 Punkte, optional]**

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 rbottom rotate   45  0 0 1
transform rbottom translate    0 0 -10
transform bsphere rotate   45  0 0 1
transform bsphere 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
# camera xform
transform eye rotate -45 0 1 0
transform eye translate 100 0 100
# ... and go
```

```
render eye image.ppm 480 320
```