Programmiersprachen und Softwarentwicklung
Sommersemester 2019

Aufgabensammlung 7

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 von git ist Pflicht. Ohne vollständige git-History 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 bei **adrian.kreskowski@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 **02. September** und am **03. September**. Bei Bedarf kann das Ray-Tracing-System auch jederzeit nach individueller Absprache **vorher** abgenommen werden.

Melden Sie Ihre Gruppe (max. 2 Personen) für die Bewertung der Aufgaben bis zum 26. August per Email bei Frau Hansens (maria-theresa.hansens@uni-weimar.de) an. Bereiten Sie zu diesem Termin eine kurze Präsentation (z.B. in PowerPoint) vor (ca. 15 Minuten), in der Sie auf Schwierigkeiten, Lösungen bei der Umsetzung sowie die Arbeitsteilung innerhalb der Gruppe eingehen. Illustrationen der Entwicklung Ihres Systems bieten sich hier ebenfalls an.

Aufgabe 7.1

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 Punktlichtquellen beleuchtet werden; es wird das in der Vorlesung vorgestellte Beleuchtungsmodel benutzt
- Objekte werfen Schatten
- der Beobachter befindet sich im Ursprung und blickt entlang der negativen z-Achse.
- Den finalen Farbwert c_{ldr} berechnen Sie mit der Formel $c_{ldr} = \frac{c_{hdr}}{c_{hdr}+1}$ aus dem vom Raytracer berechneten Farbwert c_{hdr} Diese Farbe speichern Sie in der Ausgabedatei bzw. zeigen Sie im Window an. https://en.wikipedia.org/wiki/Tone_mapping
- vollständige git-History

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.

Commiten Sie jeden erfolgreichen Implementierungsschritt in git und kommentieren Sie diesen passend. **[80 Punkte]**

Aufgabe 7.2

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 7.1 um Spiegelung.
[10 Punkte, optional]

Aufgabe 7.3

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 7.1 um Refraktion.

Hinweis: Sie können dafür Ihre Materialbeschreibung um einen Brechungsindex und einen Transparenzkoeffizienten (`opacity`) erweitern. Informieren Sie sich für diese Aufgabe selbstständig über das *Selliussche Brechungsgesetz* (*Snell's Law*) . **[10 Punkte, optional]**

Aufgabe 7.4

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 7.5

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 in Form von `glm::mat4` Matrizen ausgestattet werden.
Vor der Schnittberechung muss die Inverse der Transformationmatrix auf den Strahl angewendet werden, um den Strahl vom Weltkoordinatensystem in das lokale Objektkoordinatensystem zu überführen.
Um die Strahlposition und die Strahlrichtung mit der Matrix multiplizieren zu können, müssen Sie zunächst als homogene Koordinaten ausgedrückt werden. Beachten Sie den Unterschied zwischen der w-Koordinate für Punkte und für Richtungsvektoren, wie in der Vorlesung beschrieben.
Schreiben Sie eine Hilfsfunktion `transformRay`, welche die Transformation des Strahls übernimmt:

```
Ray transformRay(glm::mat4 const& mat, Ray const& ray);
```

Danach führen Sie die Schnittberechung mit dem transformierten Strahl im Koordinatensystem des Modells durch.
Der berechnete Schnittpunkt wird dann vom Objektkoordinatensystem zurück ins Weltkoordinatensystem transformiert und abschließend wieder in einen `glm::vec3` konvertiert. Informationen dazu finden Sie in den Folien zum Ray-Tracing.
Beachten Sie, dass Sie für die Rücktransformation der Normalen die Transponierte der Inversen World-Transformation $(m^{-1})^T$ benötigen benötigen.
Nur so erhalten Sie korrekte Beleuchtung bei non-uniformer Skalierung.
Erweitern Sie Ihren SDF-Parser um Anweisungen, die die Erzeugung der verschiedenen Objekt-Transformationen ermöglichen.

Beispiel:

```
# geometry
define shape box    rbottom -100 -80 -200 100 80 -100 red
```

```
define shape sphere bsphere 0 0 -100 50 blue

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

[50 Punkte]

Aufgabe 7.6

Implementieren Sie das in der Vorlesung vorgestellte Composite-Pattern. Das Composite-Pattern ermöglicht eine hierarchische Szenenbeschreibung in Form eines Baumes. Die inneren Knoten des Baumes sind dabei Objekte des Typs *Composite* und dessen Blätter die Objekte Ihrer geometrischen Primitive (Kugeln, Quader, etc ...).
Erweitern Sie für die Unterstützung von Composites Ihren SDFLoader wie folgt.

```
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
```

Durch die Implementierung des Composite-Patterns sind Sie nun in der Lage die gesamte Szene ausgehend von einem einzelnen Wurzelknoten auszudrücken. Ersetzen Sie deshalb Ihren bisher genutzten `Shape`-Container zur Verwaltung Ihrer Szenen-Objekte im `struct Scene` durch einen einzigen `shared_ptr<Shape>`, welcher den Wurzelknoten der Hierarchie referenziert. Ausgehend von diesem Knoten bauen Sie Ihre Szene auf.

Hinweis: Ein Beispiel für eine Vererbungshierarchie unter Benutzung des Composite-Patterns finden Sie in den Vorlesungsfolien `Vererbung I (OO-7)`.

Hinweis: Um Ihre Composite-Pattern-Implementierung zu testen müssen Sie eine Transformationshierarchie bauen die ausreichend tief ist (mehr als ein Level von aufeinanderfolgenden Transformationen).

[10 Punkte, optional]

Aufgabe 7.7

Erstellen Sie eine einfache Animation aus Ihren gerenderten Einzelbildern.
Schreiben Sie dazu ein Programm, welches für jeden Frame eine SDF-Datei generiert. Sie können zum Beispiel die Kamera um Ihre Szene kreisen lassen oder Objekte durch die Szene bewegen. Die Animation sollte mindestens 5

Sekunden lang sein und mit einer Bildwiederholungsrate von 24 Bildern pro Sekunde abgespielt werden.
Generieren Sie anschließend aus den SDF-Dateien mit Ihrem Raytracer die Bilder Ihrer Animation. Die Bilder können Sie zum Beispiel mit dem Kommandozeilentool *ffmpeg* (https://www.ffmpeg.org/) zu einem Video zusammenfügen.

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

Erstellen Sie nach der Erzeugung Ihrer Einzelbilder (mindestens 120) nun wie im Beispiel gezeigt Ihre Animation. **[10 Punkte]**

Aufgabe 7.8

Erweitern Sie die `Shape`-Hierarchie um die Primitive **Dreieck**, **Kegel** und **Zylinder**. **[15 Punkte, optional]**

Aufgabe 7.9

Verbessern Sie die visuelle Qualität Ihres Ray-Tracing-Systems durch einfaches Anti-Aliasing. Verwenden Sie dazu zur Berechnung eines Pixels 4 oder mehr Subpixel und interpolieren Sie die resultierenden Farben zwischen diesen. **[3 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] [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 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 .2 100
# 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 Gradmaßund 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 sun    500 800 0   1.0  1.0  1.0  10
define light spot1 -500 800 0   0.8  0.8  0.8  10
ambient 0.1  0.1  0.1
# 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
```