Programmiersprachen und Softwarentwicklung
Wintersemester 2009/2010

Aufgabensammlung 11

Die Aufgaben 11.1 bis 11.6 werden am **19. Januar** in der Übung bewertet. Diese Aufgaben betrachteten den Softwareanalyse- und Softwaredesignprozess am Beispiel eines Ray-Tracing-Systems.

Alle weiteren Aufgaben können in Gruppen von bis zu drei Studenten umgesetzt werden und befassen sich mit der Implementation eines Ray-Tracing-Systems. Arbeitsgruppen sollten eine *Source-Code-Verwaltung* verwenden. Die Auswahl eines *Source-Code-Verwaltungstools* (svn, cvs oder git) wird ihnen überlassen. Bitte sprechen Sie ihre Wahl jedoch mit dem Übungsleiter und/oder einem Tutor vor der Benutzung ab.

Am **16. März** und **17. März** erfolgt die Bewertung der Aufgaben 11.7 bis 11.13. Der Termin ist pro Arbeitsgruppe vorher per Email zu vereinbaren. Jede Gruppe sollte ihre Implementation mit einem **30-minütigen Vortrag** vorstellen und dabei auf Probleme und Lösungen bei der Implementierung sowie die Arbeitsteilung innerhalb der Gruppe eingehen und die anschließenden Fragen (ebenfalls ca. 30 Minuten) beantworten.

Aufgabe 11.1

Der Raytracer berechnet eine numerische Annäherung an die *Rendering Equation*. Diese beschreibt die ausgehende Strahldichte eines Punktes als Summe von ausgestrahlter und reflektierter Strahldichte. Die Rendering Equation kann in folgender Form aufgeschrieben werden:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\mathcal{H}^2} f_r(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| \mathrm{d}\omega_i \qquad (1)$$

Hierbei bezeichnet p den betrachteten Oberflächenpunkt, L die Strahldichte (Radiance) und f_r die bidirektionale Reflektanzverteilungsfunktion. Die Strahldichte wird im Raytracer als RGB Wert dargestellt. Erklären Sie die Rendering Equation und das in der Vorlesung vorgestellte Beleuchtungsmodell. Erklären Sie den Zusammenhang zwischen beiden. Nutzen Sie dazu die Skizze und erweitern Sie diese.

[10 Punkte]

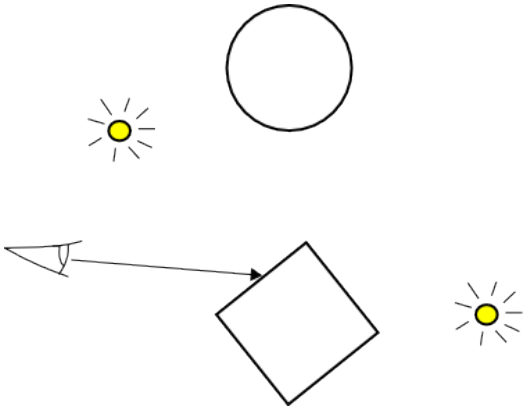

Abbildung 1: Strahlverlauf

Aufgabe 11.2

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

- Eine im SDF-Dateiformat 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 11.3

Definieren Sie die Fähigkeiten bzw. Verantwortlichkeiten ihrer Klassen und listen Sie diese auf:

1. window:
   - ▶ Gibt das berechnete Bild auf dem Monitor aus
2. fileloader:
   - ▶ ...
3. ...

[10 Punkte]

Aufgabe 11.4

Konkretisieren Sie den Verantwortungsbereich *fileloader* aus Aufgabe 11.3.

SDF file → *fileloader* → output

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

Aufgabe 11.5

Erstellen Sie ein detailliertes UML-Flussdiagramm für die Beleuchtungsberechnung eines ermittelten Schnittpunktes. Grundlage dafür ist das in der Vorlesung vorgestellte Beleuchtungsmodell.
Beachten Sie, dass in die Berechnung die ambiente und alle diffusen Lichtquellen eingehen. Berücksichtigen Sie auch Spiegelung und Schatten in dem Diagram. Unterscheiden sie dabei explizit zwischen Strahlverfolgung und Beleuchtungsberechnung.
Definieren sie anhand der notwendigen Informationen für die Ermittlung des *nächsten* Schnittpunktes das Interface für die abstrakte Methode `shape::intersect` und spezifizieren deren Rückgabe genauer. **[20 Punkte]**

Aufgabe 11.6

Richten sie ein Projekt für den Raytracer ein und kompilieren sie das bereitgestellte Framework.

Hinweis: Die Abhängigkeiten GLUT und Boost sind für Windows dabei bzw. müssen unter Linux/MacOS installiert werden. **[5 Punkte]**

Aufgabe 11.7

Entwickeln 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. Gehen Sie von ihren in UML formulierten Softwareentwurf aus. **[80 Punkte]**

Aufgabe 11.8

Ermöglichen Sie eine hierarchische Szenenbeschreibung. Die Szene wird dabei durch einen Baum beschrieben, dessen innere Knoten *composites* und dessen Blätter die primitiven Objekte (Kugeln, Quader, etc ...) sind. Nutzen Sie das Composite-Pattern zur Realisierung.
Erweitern Sie die entsprechende Basisklasse und erlauben Sie die Transformation (Translation, Skalierung und Rotation) ihrer geometrischen Objekte und der Composite-Objekte.
Erweitern Sie Ihre Primitive um die akkumulierte Transformationsmatrix `world_transformation` und ihre Inverse. Vor der Schnittberechung muss diese Transformation auf den Strahl angewendet werden. Der berechnete Schnittpunkt wird dann zur Beleuchtungsberechnung mit der Inversen zurück in Weltkoordinaten transformiert. Beachten Sie, dass Sie für die Rücktransformation der Normalen die Transponierte der Inversen benötigen.
Erweitern Sie ihren SDF-Parser um das Manipulieren dieser Transformationen zu erschließen, 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 rotatex        45
```

[60 Punkte]

Aufgabe 11.9

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 11.7 um Schatten.

[20 Punkte]

Aufgabe 11.10

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 11.7 um Spiegelung.

[40 Punkte]

Aufgabe 11.11

Erweitern Sie die `shape`-Hierarchie mit Dreiecken.

[20 Punkte]

Aufgabe 11.12

Erweitern Sie das Beleuchtungsmodell aus Aufgabe 11.7 um Refraktion.

Hinweis: Sie können dafür ihre Materialbeschreibung um einen Brechungsindex und einen Transparenzkoeffizienten (`opacity`) erweitern. **[40 Punkte, optional]**

Aufgabe 11.13

Optimieren Sie ihren Ray Tracing Algorithmus, indem Sie eine Bounding-Box-Hierarchie verwenden. **[40 Punkte, optional]**

Aufgabe 11.14

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]**

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.
Umbrello UML (Linux/KDE) und StarUML (Windows) sind im Pool installiert. Beide Programme sind auch in der Lage C++-Code zu generieren. Nutzen sie 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
```