

Sommersemester 2012

Übungen zu Computergraphik II - Blatt 5

Abgabe am 25. 05. 2012

Aufgabe 1 (Raytracing, 10 Punkte)

In dieser Aufgabe soll ein Raycasting-Framework zu einem vollständigen Raytracer erweitert werden. Dazu müssen einige Funktionen des Frameworks ergänzt werden. Das Grundgerüst finden Sie wie immer auf der Vorlesungshomepage.

Nehmen Sie sich bitte zunächst etwas Zeit um sich in das Gerüst des Raytracers einzuarbeiten. Neben den Hinweisen auf diesem Blatt sollten auch die Kommentare im Quelltext beim Verständnis helfen.

Das Programm besteht aus vier wesentlichen Komponenten:

- **GUI:** Die Benutzeroberfläche basiert auf Qt4. Neben einem Fenster, in dem das Ergebnisbild des Raytracings angezeigt wird, gibt es noch ein Fenster, welches die Szene in OpenGL darstellt. Diese Ansicht dient zur Modifikation der Szenen, zum Debugging, zur Veranschaulichung des Raytracing-Prozesses und auch zur Verdeutlichung der Unterschiede zwischen globaler und lokaler Beleuchtungsberechnung.

Die Klassen für die grafische Oberfläche und die Verwendung von Qt und OpenGL sind jedoch nicht notwendig zum Verständnis des Raytracers. Deswegen sollten Sie diese Dateien am besten zunächst unbeachtet lassen.

- **XML-Parser:** Das Programm liest Szenen im XML-Format ein. Dazu wird eine frei erhältliche XML-Bibliothek verwendet. Für Windows liegt dem Framework eine fertig kompilierte dll-Datei bei. Für Linux-Benutzer steht im `xmlParser`-Unterverzeichnis ein `Makefile` bereit. Ebenso wie bei der Benutzeroberfläche, ist ein tieferes Verständnis der Parser-Bibliothek nicht notwendig zum Verstehen des Raytracers.

Das Format der xml-Dateien ist weitgehend selbsterklärend. Schauen Sie sich dazu am besten die beiliegenden Beispielszenen (`objects.xml`, `glass-spheres.xml`, `metal-spheres.xml` und `steinbach.xml`) an.

- **Mathematische Hilfsklassen:** Einige einfache Template-Klassen zur Vereinfachung von Berechnungen:
 - `VectorT`: Template für n-dimensionale Vektorarithmetik
 - `MatrixT`: Template für quadratische $n \times n$ -Matrizen
 - `Matrix33T`: Spezialisierung von `MatrixT` für 3×3 -Matrizen
 - `ColorT`: Template für RGB-Farbarithmetik
- **Raytracer:** Die Grundfunktionalität des Raytracers ist in der Klasse `Raytracer` konzentriert. In der Funktion `Raytracer::render()` werden die Strahlen für die einzelnen Pixel erzeugt und mittels der Funktion `Raytracer::traceRay()` durch die Szene verfolgt. Die von Ihnen zu implementierende Funktion `Raytracer::shade()` wertet das lokale (Phong-)Beleuchtungsmodell in

einem Punkt der Szene aus. Außerdem wird in dieser Funktion überprüft, ob sich der Punkt im Schatten befindet.

Die Klasse `PinholeCamera` implementiert eine einfache Lochkamera. Die wichtigste Funktion stellt `PinholeCamera::generateRay()` dar. Sie generiert für einen gegebenen Pixel (x,y) einen Strahl durch den Augpunkt.

Die Klasse `Ray` dient zur Repräsentation eines solchen Lichtstrahls. Strahlen werden durch Startpunkt und Richtung definiert. Darüber hinaus bietet die Klasse auch Funktionen zur Berechnung gebrochener und reflektierter Strahlen.

Die Szenendefinition (Objekte, Materialien, Lichtquellen) wird durch virtuelle Basisklassen realisiert:

- **Surface:** Virtuelle Basisklasse für geometrische Objekte. Alle abgeleiteten Klassen müssen eine Funktion `intersect()` zur Berechnung des Schnitts eines Strahls mit dem Objekt, zur Verfügung stellen. Die abgeleiteten Klassen `Plane`, `Sphere` und `Checkerboard` sind bereits vollständig implementiert.
- **Shader:** Virtuelle Basisklasse für Materialien. Die abgeleitete Klasse `PhongShader` ist bereits vollständig implementiert. Die Funktion `PhongShader::shade()` berechnet das lokale Phong-Beleuchtungsmodell.
- **Light:** Virtuelle Basisklasse für Lichtquellen. Die abgeleiteten Klassen `PointLight` und `DirectionalLight` sind bereits vollständig implementiert.

Ihre Aufgaben:

- a) Implementieren Sie die Funktion `Raytracer::shade()` in `Raytracer.cpp`. In dieser Funktion werden die Schattenstrahlen für die Szene erzeugt und die (Phong-)Beleuchtung für das entsprechende Pixel gesetzt, falls es nicht im Schatten liegt.
Wenn diese Funktion richtig implementiert wurde, sollte die Beleuchtung ungefähr so aussehen wie im OpenGL-Fenster (+zusätzlicher Schatten).
- b) Nun fehlen noch die fürs Raytracing wichtigen Reflexionen. Um diese hinzuzufügen, implementieren Sie die Funktion `Ray::reflectedRay()` in `Ray.cpp`. Die Funktion soll aus den Input-Parametern einen perfekt reflektierten Strahl berechnen. Verwenden Sie diese Funktion um in `Raytracer::traceRay()` rekursiv den Farbanteil des gespiegelten Strahls zu berechnen und addieren Sie den erhaltenen Wert zum Farbwert hinzu.
- c) Neben Reflexionen tragen gebrochene Strahlen zum realistischen Eindruck von Raytracing-Bildern bei. Implementieren Sie dazu die Funktion `Ray::refractedRay()` in `Ray.cpp`. Die Funktion soll einen durch die Materialparameter bestimmten gebrochenen Strahl erzeugen. Verwenden Sie diese Funktion um in `Raytracer::traceRay()` rekursiv den Farbanteil des gebrochenen Strahls zu berechnen und addieren Sie den erhaltenen Wert zum Farbwert hinzu.