Prof. G. Zachmann
D. Mohr

TU Clausthal
Institut für Informatik
13. Juli 2010

Sommersemester 2010

Übungen zu Computergraphik II - Blatt 6

Abgabe am Donnerstag, den 15. 07. 2010, 10:15 Uhr

Aufgabe 1 (Raytracing: Schnittest, *2 Punkte*)

In der Vorlesung wurde (ab Folie 55) die geometrische Methode zur Schnittberechnung einer Kugel mit dem Strahl erklärt.

Geben Sie die algebraische Methode an.

Aufgabe 2 (Raytracing: Instancing, *2 Punkte*)

Beim Instancing (Folie 66) wird der Strahl für den Schnittest in die kanonische Form eines Objektes transformiert und danach der Schnittest durchgeführt. Die Rückkonvertierung von t' nach t wurde in der Vorlesung nicht angegeben. Ergänzen Sie dies.

Aufgabe 3 (Raytracing: Metaballs, *10 Punkte*)

In dieser Aufgabe soll ein Raytracing-Framework um Metaballs erweitert werden. Das Grundgerüst *RayTracingTemplateMB* finden Sie wie immer auf der Vorlesungshomepage.

Einführung in das Framework:

Das Programm besteht aus vier wesentlichen Komponenten:

- Die **GUI** 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-Bibliohtek 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 GUI 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`, `meta-balls.xml` und `stein-bach.xml`) im Unterordner `scenes` an.
- **Mathematische Hilfsklassen:** Einige einfache Template-Klassen zur Vereinfachung von Berechnungen:
  - `VectorT`: Template für n-dimensionale Vektorarithmetik
  - `MatrixT`: Template für quadratische n×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 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 einen Strahl vom Augpunkt durch einen gegebenen Pixel (x, y).

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 vollständige implementiert.

Ihre Aufgaben:

Implementieren Sie die Funktion `Metaball::intersect()` in `Metaball.cpp`. Verwenden Sie als Potentialfunktion die in der Vorlesung auf Folie 78 definierte Exponentialfunktion:

$$p(r) = e^{-br^2}$$

Als Blending verwenden Sie die Addition aus Folie 76:

$$P(\mathbf{x}) = \sum_{i=1}^{n} a_i \; p(\| \mathbf{x} - \mathbf{x}_i \|)$$

Damit ergibt sich die Isofläche

$$K = \{\mathbf{x} \mid P(\mathbf{x}) - \tau = 0\}$$

Die Nullstellen eines Metaballs dürfen Sie durch Brute-Force-Sampling entlang des Strahls abschätzen. (Gerne dürfen Sie auch eine etwas ausgefeiltere Methode implementieren.)

Der Isowert τ, für jeden Ball i die Feldstärke a_i, der Skelettpunkte x_i und die Bandbreite b_i der Potentialfunktion werden in der XML-Datei festgelegt. Im Sourcecode finden Sie diese in `Metaball::m_isovalue` und `Metaball::m_balls` wieder. Im OpenGL-Vorschaufenster werden die Skelettpunkte x_i durch kleine Kugeln visualisiert.