Programmieren II - Projektarbeit

Prof. Dr. Helmut Neemann

Raytracing

Organisatorisches

- Die Abgabe des Programmentwurfs erfolgt spätestens am Freitag, den 19.12.2014 als gepackte Quellen im ZIP-Format.
- Die Bearbeitung der Aufgabe erfolgt in Gruppen von jeweils zwei Studierenden. **Bitte vermerken Sie Ihre Namen und Ihre Matrikelnummern als Kommentar im Kopf jeder Quelldatei**.
- Relevant für die Bewertung Ihres Programmentwurfs sind lediglich die Quelldateien, da diese bei der Bewertung komplett neu übersetzt werden und die Übersetzbarkeit der Quellen ein Bewertungskriterium darstellt.
- Jede Gruppe fertigt eigenständig eine individuelle Lösung der Aufgabenstellung an und reicht diese wie oben angegeben ein.
- Die geforderte Funktionalität muss von Ihnen selbst implementiert werden.
- Da der Raytracing-Algorithmus sehr einfach ist, finden sich dutzende Implementierungen im Netz. Sie können sich Anregungen aus diesen Projekten holen, allerdings muss in diesem Fall die Herkunft der Ideen bzw. von Teilen des Quellcodes im Kommentar Ihrer Quelldatei(en) vermerkt sein.

Viel Spaß und viel Erfolg bei der Bearbeitung!

Aufgabenstellung

Es soll ein minimaler Raytracing-Algorithmus implementiert werden. Die Anwendung soll in der Lage sein, eine beispielhafte Szene (Abbildung 1) zu berechnen, welche mindestens folgendes beinhaltet:

1. Zwei Kugeln unterschiedlicher Farbe.
2. Die Kugeln sollen auf einer Ebene liegen.
3. Es soll eine punktförmige Lichtquelle verwendet werden, die außerhalb der sichtbaren Szene platziert ist.
4. Die Kugeln sollen einen Schatten werfen.
5. Es soll das Phong-Beleuchtungsmodell für alle Oberflächen verwendet werden.
6. Alle sichtbaren Oberflächen sollen leicht spiegelnd sein.

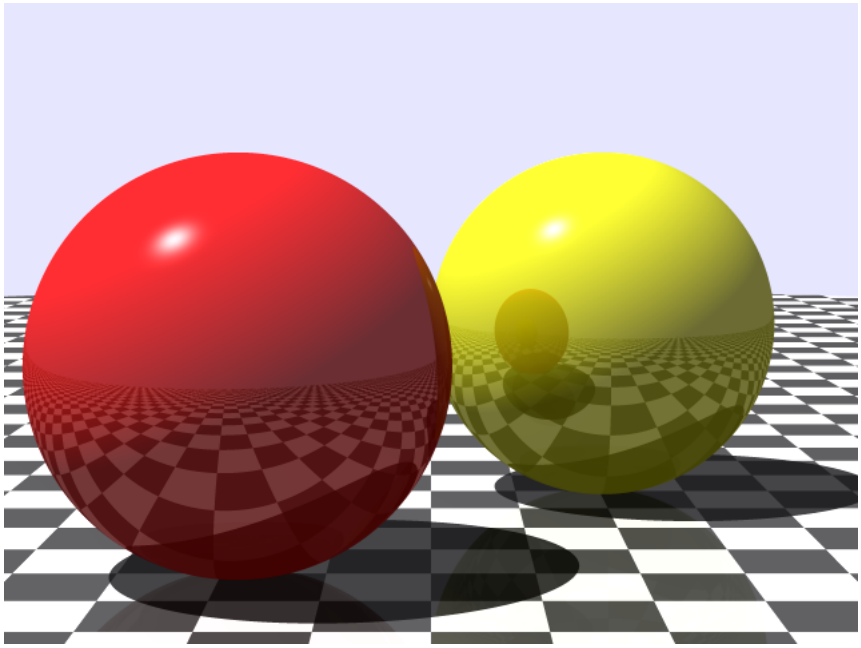

Abbildung 1: Beispielgrafik incl. der optionalen Features Kantenglättung und Bodentextur.

7. Die Kugeln sollen so platziert sein, dass sie als Spiegelung deutlich in der jeweils anderen Kugel zu sehen sind.

Anforderungen

- Der Code soll im Paket `de/vorlesung/projekt/[Gruppen-ID]` liegen, damit alle Lösungen parallel im Source-Tree gehalten werden können.
- Es dürfen keine Pakete Dritter verwendet werden! Einzige Ausnahme sind Pakete zur Vereinfachung der Tests.
  Empfohlen sei hier `github.com/stretchr/testify/assert`.
- Ausgegeben werden sollen PNG-Grafiken. Diese können mit den Paketen `image` und `image/png` erzeugt werden.
- Es sollen Benchmarks implementiert werden. Diese sollen weitere Beispielgrafiken im System-Temp-Ordner (verfügbar per `os.TempDir()`) erzeugen.
- Das Main-Paket soll nur den Code zur obigen Szenenbeschreibung enthalten (Platzieren der Objekte, Kamera und Lichtquelle, Start des Renderers.) Die eigentliche Funktionalität soll in entsprechenden Paketen organisiert werden
- Ein Einlesen einer Szenenbeschreibung aus einer Beschreibungsdatei o.ä. ist nicht erforderlich.

- Das Programm soll ohne die Angabe von Aufruf-Parametern starten, und eine PNG-Grafik der Größe 640x480 Pixel ähnlich der Abbildung 1 im aktuellen Ordner erzeugen.
- Implementiert werden müssen zumindest der Schnitt mit einer Kugel und mit einer Ebene.
- Da Spiegelung gefordert ist, muss das rekursive Raytracing implementiert sein.
- Es soll Testgetrieben entwickelt werden: Es sollen alle wichtigen Funktionen durch geeignete Testfälle abgedeckt sein, die mit `go test` gestartet werden können.
- Alle Testfälle müssen erfolgreich durchlaufen werden.
- Der Code soll mit `gofmt` formatiert sein.

Bewertungskriterien

Ihr Programmentwurf wird nach folgenden Kriterien (mit abnehmender Gewichtung aufgeführt) bewertet:

1. Abbildung der Anforderungen (s.o.)
2. Strukturierung und Konsistenz des Quellcodes
3. Ausreichende Testabdeckung des implementierten Codes. (gemessen mit `go test -cover`)
4. Sinnhaftigkeit der Tests (Sonderfälle, Grenzfälle usw.)
5. Qualität von Kommentaren und Dokumentation
6. Benutzerfreundlichkeit der Schnittstellen (APIs)
7. Optionale Features

Optionale Features

- Kantenglättung durch Überabtastung (Für jedes Pixel werden z.B. 25 unterschiedliche Sehstrahlen berechnet und die gemittelte Farbe angezeigt).
- Schachbrett-Textur auf der Ebene z.B. als Volumentextur (jedem Raumpunkt wird eine Farbe zugeordnet)
- Parallelisierung des Rendering-Vorganges um Multi-Core Prozessoren auszunutzen.
- Berücksichtigung von Lichtbrechung, um Linsen darstellen zu können.
- Ergänzung weiterer Körper (z.B. Zylinder, Würfel, Polygone u.s.w.)
- Ergänzung weiterer Volumen-Texturen (Holz oder Marmor)
- Ergänzung von Rausch-Funktionen wie z.B. “Perlin noise“ für realistischere Holz-

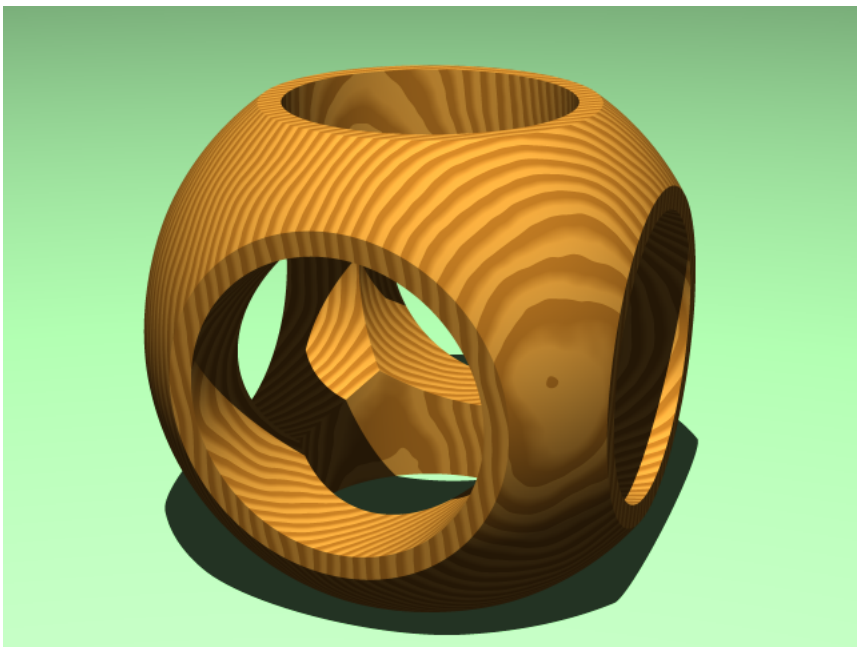

Abbildung 2: Eine dreifach durchbohrte Holzkugel

oder Marmortexturen.

- CSG-Algorithmen (Constructive Solid Geometry) für die Berechnung von boolschen Volumenoperationen auf Körpern wie z.B. “Kugel UND NICHT Zylinder“, Abbildung 2.
- Implementierung von Beschleunigungsverfahren wie z.B. “Teilszenen-Einkugelung“ (bounding volume hierarchies), Abbildung 3.
- Alles was Ihnen sonst noch so einfällt. Bitte sprechen Sie mich vor der Implementierung an!

Link-Sammlung

- `http://de.wikipedia.org/wiki/Raytracing`
- `http://de.wikipedia.org/wiki/Phong-Beleuchtungsmodell`
- `http://www.youtube.com/watch?v=94ilBqGUmyU`
  2 Minuten-Video über das Raytracing der Die Illusions-Schmiede GmbH
- `https://www.youtube.com/watch?v=IyUgHPs86XM`
  Vortrag über die physikalischen Grundlagen von John Carmack (Mitgründer von ID-Software) auf der QuakeCon 2013

Abbildung 3: Ein fraktaler Baum mit 29524 Zweigen und 39366 Blättern, 640x480 Pixel, 9 fache Überabtastung, berechnet in 50 sec. (Intel Core i5-3230M CPU @ 2.60GHz)

- `http://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html`
  Eine knappe Zusammenstellung aller mathematischen Grundlagen.