Praktikum Computergrafik

Steven Schlegel

Abteilung für Bild- und Signalverarbeitung

Betreuer: Steven Schlegel (schlegel@informatik.uni-leipzig.de)

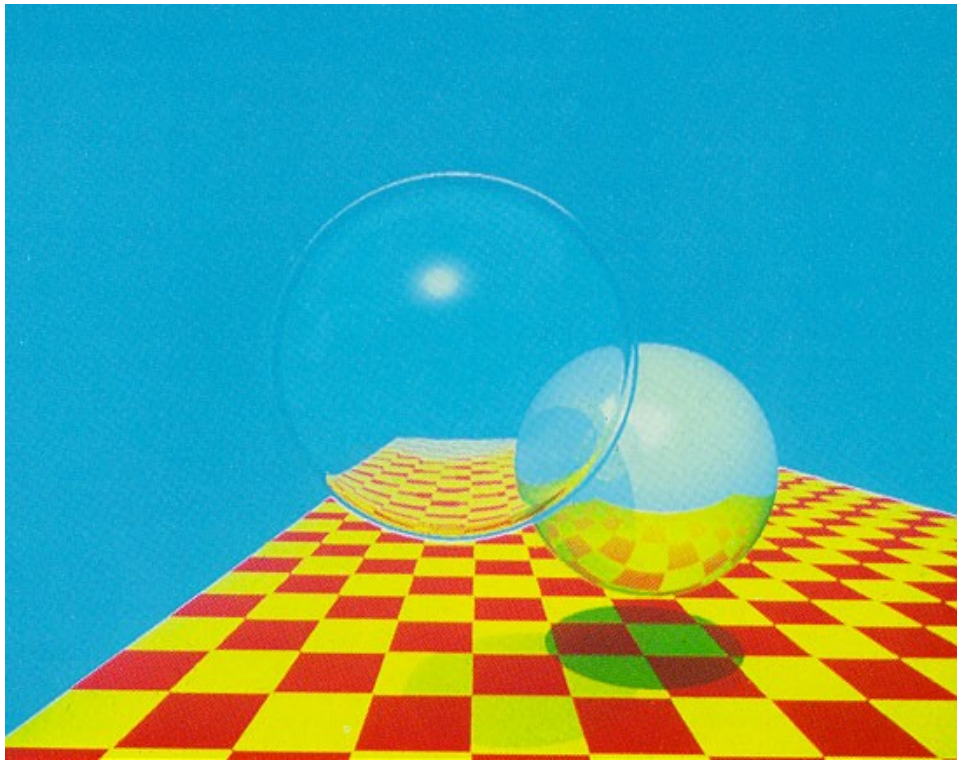
Aufgabe

- Ziel des Praktikums ist es, einen Raytracer zu schreiben, der eine vorgegebene Szene rendert.

- Dabei wird auf folgende “Features” Wert gelegt:
  - Schattenberechnung
  - spiegelnde Flächen
  - transparente Flächen

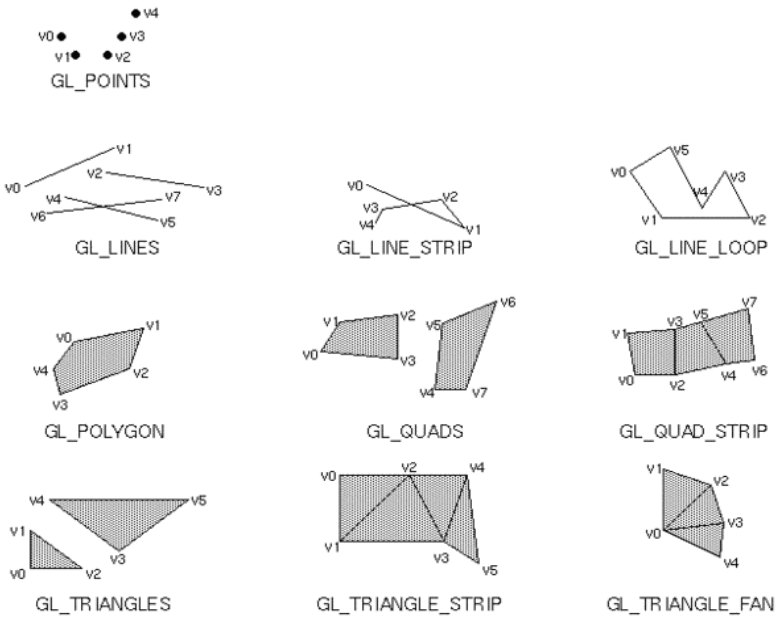
- Die Szene besteht aus Dreicken mit bestimmten Oberflächeneigenschaften (Farbe, Transparenz...) und Informationen über die Lichter, welche die Szene beleuchten.

Raytracing - Beispiele



Grafikprimitive

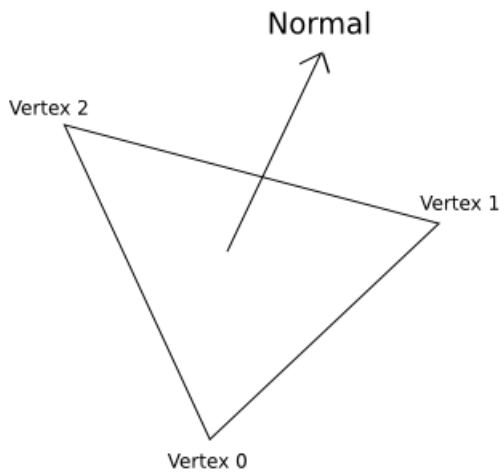
- Jedes Objekt wird durch Grafikprimitive dargestellt
- In OpenGL z.B. gibt es folgende Grafikprimitive



- Hier: Beschränkung auf Dreiecke

Grafikprimitive

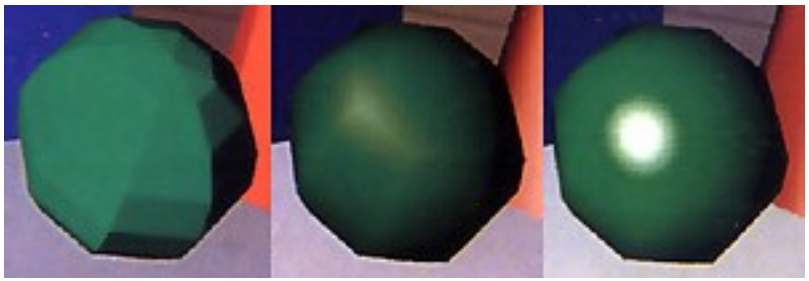
- Dreiecke bestehen aus den Positionsvektoren der Eckpunkte und (midestens) einer Oberflächennormalen:



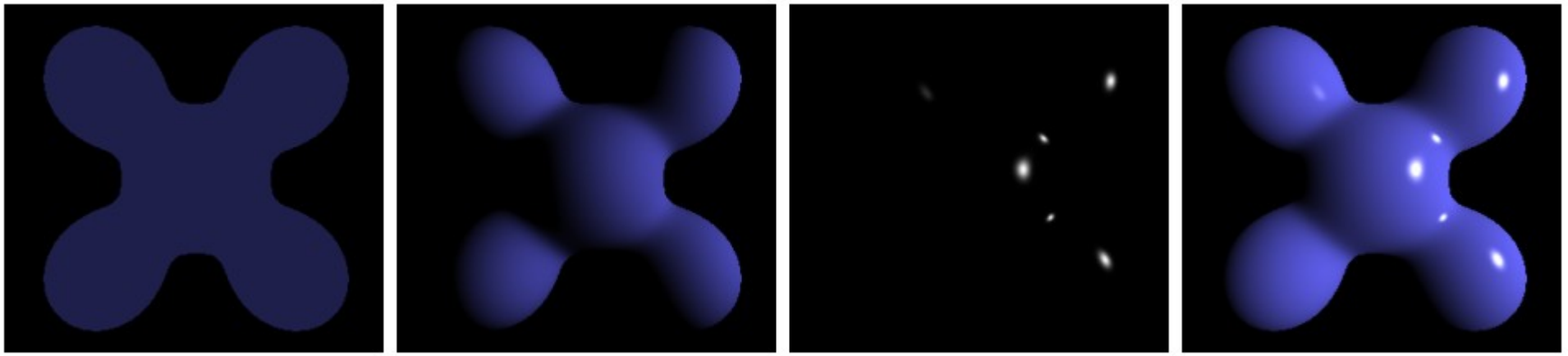
- Anmerkung: Die Normale(n) können über das Kreuzprodukt der Dreiecksvektoren ausgerechnet werden

Shading - Grundprinzip

- Die Normale bestimmt, wohin die Dreiecksfläche zeigt, was wiederum wichtig für die Farbberechnung unter Lichteinfall ist.
- Hat man eine Normale für jedes Dreieck (siehe Abbildung auf vorheriger Folie), erhält man FLAT-Shading. Dabei ist jedes Dreieck einzeln sichtbar.
- Ist an jedem Positionsvektor eine Normale definiert, so kann entweder der Farbwert an den Ecken des Dreiecks ausgerechnet werden und innerhalb interpoliert werden (Gouraud-Shading) oder die Normale für jede Position auf dem Dreieck interpoliert werden (Phong-Shading), wodurch jeweils ein glatter Farbverlauf entsteht
- Für den Raytracer wird das Phong-Shading benutzt.



Exkurs: Farbberechnung - Das Phong Modell



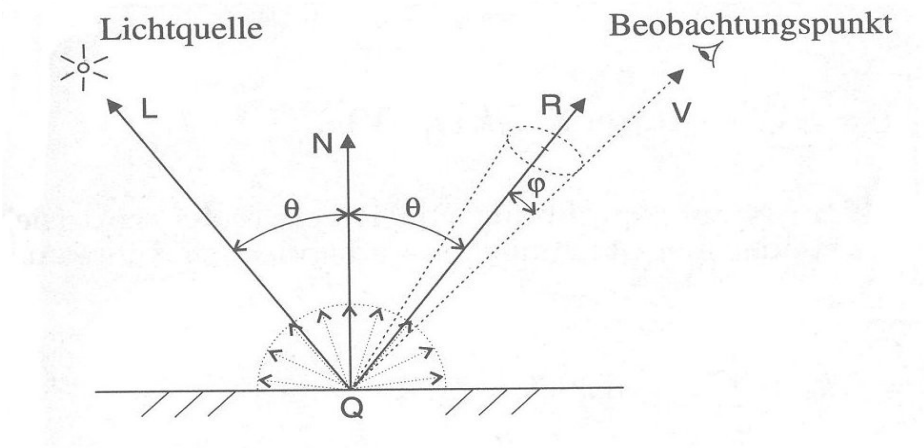
Das Phong-Modell ist ein **lokales** Beleuchtungsmodell.
Die Farbe eines Objektes bestimmt sich aus den Materialeigenschaften, des Objektes sowie der Farbe des Lichtes, welches das Objekt beleuchtet.

Materialeigenschaften bestehen aus drei Komponenten:

- → ambienter Farbanteil ("Hintergrundfarbe" - unabhängig von der Beleuchtung)
- → diffuser Farbanteil (die Farbe, die durch die direkte Beleuchtung entsteht)
- → specularer Farbanteil (Glanzlichter – simulieren sich im Objekt spiegelnde Lichtquellen)

Durch diese Komponenten wird die Farbe bestimmt, welche das Objekt reflektiert.

Exkurs: Farbberechnung - Das Phong Modell



Q: Punkt auf Objektoberfläche
N: Flächennormale in Q
L: Vektor von Q zu einer Punktlichtquelle
V: Vektor von Q zum Augpunkt
R: Reflexionsvektor von L

$$I = I_a k_a + I_d * k_d * \cos(\Theta) + I_s * k_s * \cos^k(\phi))$$

mit:

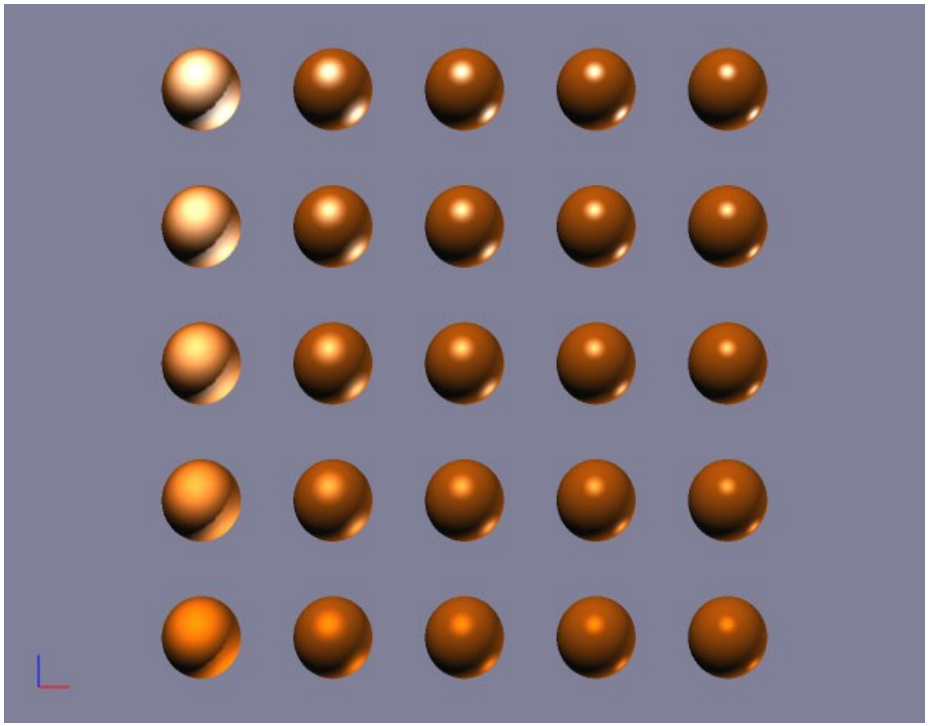
I_a	:	Intensität des ambienten Lichts
I_d	:	Intensität des diffusen Lichts
I_s	:	Intensität der spiegelnden Lichts
k_a, k_d, k_s	:	Reflexionskoeffizienten des Objektes (ambient, diffus, specular)
k	:	Grad des Oberflächenglanzes (z.B. k=1 für eine matte, k=100 für eine glänzende Oberfläche)

Berechnung: $\cos(\Theta) = L * N$ und $\cos^k(\phi) = R * V$

Die Intensität wird für alle 3 Farbanteile (R,G,B) separat ausgerechnet und bestimmt so die endgültige Farbe an Punkt Q.

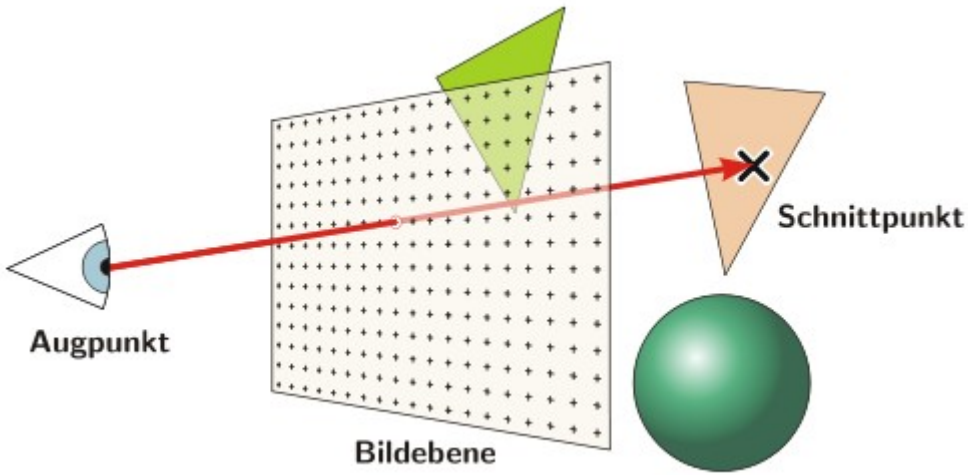
Shininess

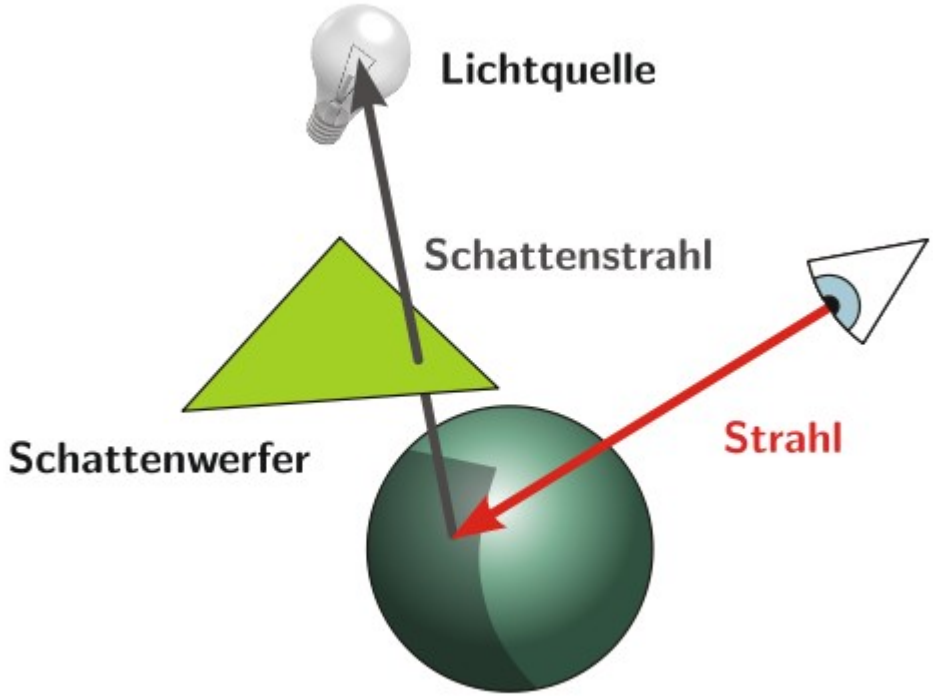
- Auswirkung der Shininess



Von links nach rechts beträgt die Shininess 5, 25, 45, 65 und 85. Von unten nach oben geht die spiegelnde Farbe von der selben Farbe, wie die difuse Farbe nach weiß über.

Raytracing - Prinzip

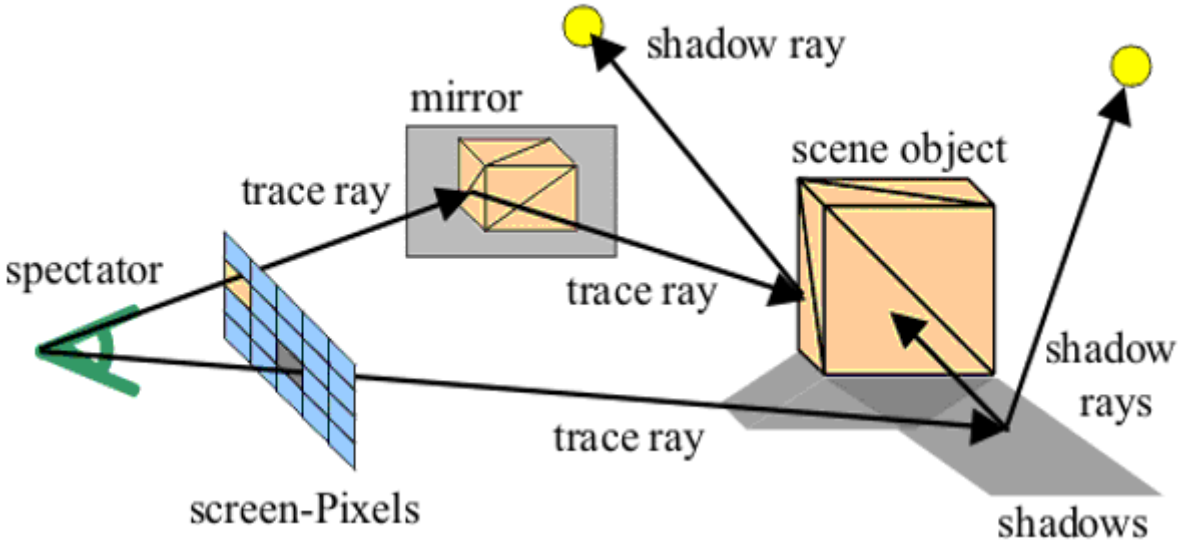




- 1. Schnittpunkttest (RayCasting)
- 2. rekursive Strahlverfolgung (RayTracing) für Schatten, Lichtreflexion und -brechung

(Skizze: nur Schattenberechnung)

Raytracing - Prinzip

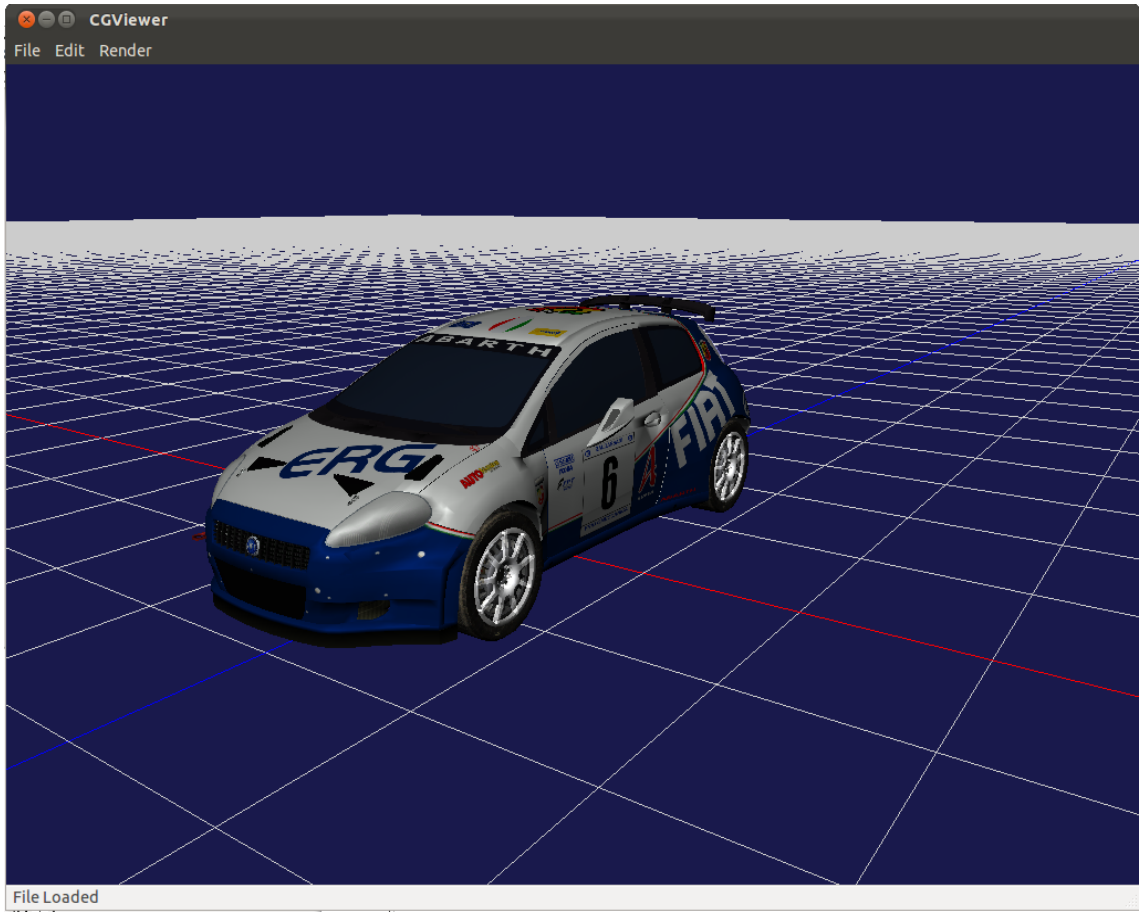


PSEUDOALGORITHMUS:

```
Color raytrace { ray, depth}
{
        Für jede Lichtquelle:
        {
                bestimme den nächstgelegenen Schnittpunkt des Strahls mit dem Objekt;
                if (kein Schnittpunkt)
                        color = background;
                else
                {
                        berechne Schattenstrahl (zur Lichtquelle);
                        If (kein Objekt zur Lichtquelle)
                                color = vollständiger lokaler Anteil;
                        berechne reflektierten Strahl;
                        ref_color = raytrace (reflektierter Strahl, depth-1);
                        berechne gebeugten Strahl;
                        trans_color = raytrace( gebeugter Strahl, depth-1);
                        color =  color + ref_color + trans_color;
                }
        }
        return color;
}
```

CG Viewer

Tool / Code-Framework für das Praktikum



geschrieben in C++ mit Qt
(http://doc.qt.nokia.com/)

CGViewer

Er ist in der Lage **triangulierte** Objekte im OBJ-Format (Infos zum OBJ-Format unter http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/ und http://local.wasp.uwa.edu.au/~pbourke/dataformats/mtl/) zu importieren. Solche 3D-Modelle können z.T. kostenlos im Internet heruntergeladen werden, z.B. unter http://www.gfx-3d-model.com/.
Hinweis: Diese Modelle sind nicht immer trianguliert und darüberhinaus schwankt ihre Qualität stark. Von daher empfiehlt es sich, sie in ein 3D-Bearbeitungsprogramm (z.B. Blender www.blender.org) zu laden, evtl. zu bearbeiten und dann trianguliert ins OBJ-Format zu exportieren.

Es sind einfache Transformationen der Objekte möglich, sowie das Hinzufügen von Lichtquellen. Die Szene kann abschließend in ein einfaches Ascii-Format exportiert werden. Die Spezifikation des Formates können Sie auf der Praktikumswebseite herunterladen.

Diese gespeicherte Szene können Sie wiederum erneut laden und auch mit Hilfe Ihres Raytracers rendern.

Tutorial

- Modell in den CGViewer einladen
- Szene bearbeiten, Lichtquellen hinzufügen usw.
- Szene exportieren

Quellcode

- Wichtig für den Raytracer:
  - **CGMath.h**: Vectorklasse, Skalar- und Kreuzprodukt, Datenstrukturen (Lightsource, Material, Triangle)
  - **Raytracer.h**: std::vector<Triangle> triangles, std::vector<Lightsource> lights, QColor backgroundcolor, ambientlight
  - **Raytracer.cpp**: QColor raytrace(Vector start, Vector dir, int depth), Supersampling, Parallelisierung

1. Aufgabe

- a.) Implementierung Schnitt-Test
- b.) Beschleunigung