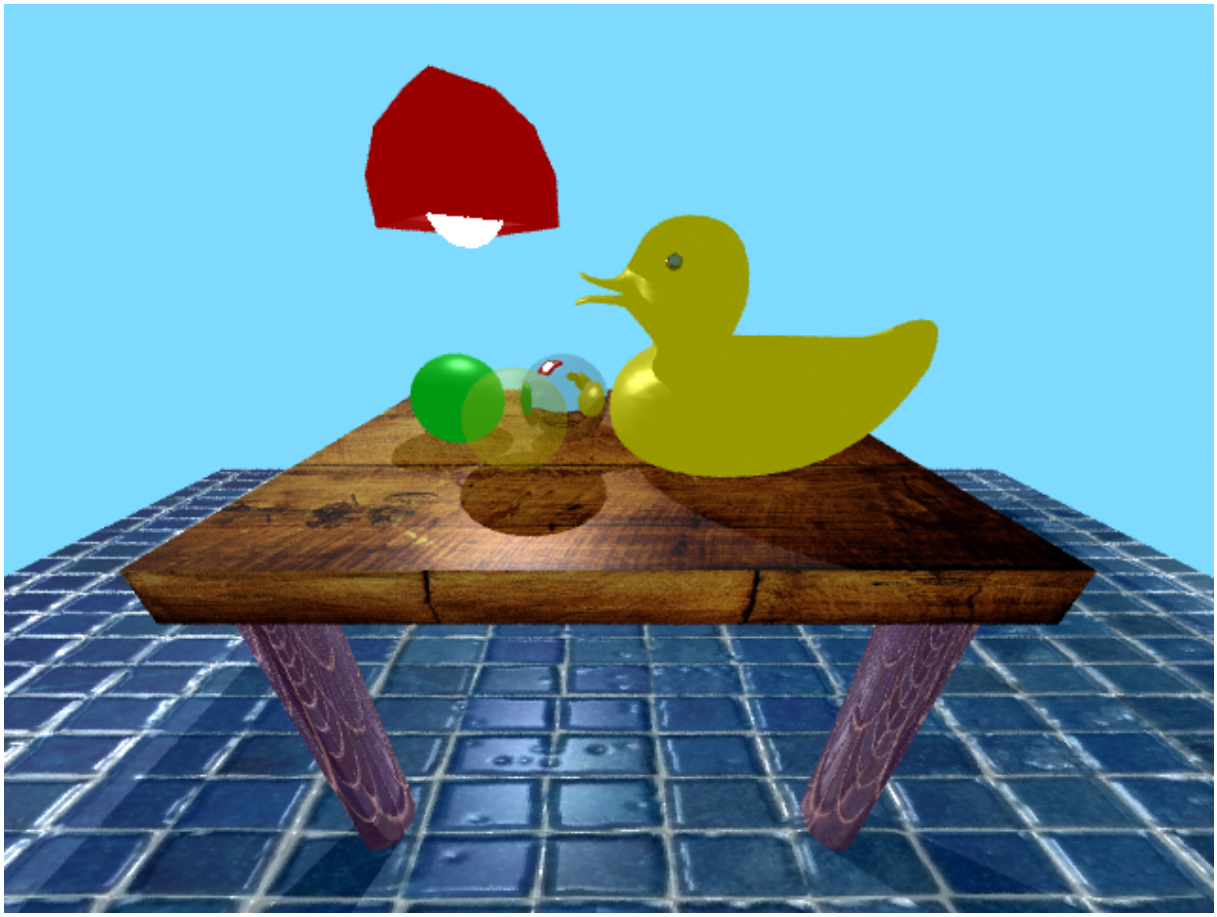
Implementierung eines einfachen Raytracers



Praktikum WS 2011

Philipp Ruchti

16. Januar 2012

Raytracing

Bei der gewohnten Erstellung eines Bildes, wie man es beispielsweise von OpenGL kennt werden die Vertices einer 3D-Szene mit Hilfe einer Projektionsmatrix auf eine Bildfläche projiziert und damit die Farbe eines Bildpunktes bestimmt. Beim Raytracing werden im Gegensatz hierzu Strahlen von der Kamera durch die Bildfläche geschickt und diese mit der 3D-Szene geschnitten. Diese Technik bietet die Möglichkeit auch Reflektionen, Schatten und viele andere Lichteffekte darzustellen. Im Gegensatz zur ersten Technik bietet Raytracing die Möglichkeit realistischere Bilder einer 3D-Szene zu berechnen ist jedoch rechenaufwendiger. Im Verlaufe dieses Praktikums wurde ein einfacher Raytracer implementiert, welcher die grundlegenden Funktionalitäten bietet eine 3D-Szene abzubilden. Im folgenden werden die dafür verwendeten Techniken kurz erläutert und einige Ergebnisbilder gezeigt.

Homogene Notation

Um 3D-Positionen und Verschiebungen zu repräsentieren eignet sich die sogenannte homogene Notation. Hierzu werden 3D-Vektoren um eine vierte Komponente erweitert. Ein euklydischen Punkt $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$ wird hierbei durch den homogener Punkt (x, y, z, w) beschriebne. Diese vierte Komponente w beschreibt eine Skalierung des Vektors. Punkte haben als 4.Komponente einen Wert $\neq 0$. Vektoren welche eine 4. Komponente $= 0$ aufweisen sind Richtungen, welchen eine unendliche Länge aufweisen und sich nicht verschieben lassen. Mit dieser Notation ist es möglich mit Hilfe von 4x4-Transformatrizen Punkte und Vektoren zu rotiern sowie Punkte zu verschieben. Eine Matrix die einen Punkt mit einer Rotationsmatrix $\mathbf{M}$ rotiert und um (x, y, z) verschiebt sieht homogen wie folgt aus:

$$\begin{pmatrix} & & & x \\ & \mathbf{M} & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Platzierung von Objekten und der Kamera

Um Objekte und Kamera platzieren und verschieben zu können verwendet man homogene Matrizen. Hierbei erhält man das gleiche Ergebnis wenn man die Kamera in einer Richtung bewegt, oder alle Objekte in die entgegengesetzte Richtung bewegt, da für die Sicht in die Szene lediglich die relative Position von Kamera und Objekten eine Rolle spielt. Um die Verwendung von AABBs zu vereinfachen wurde in dieser Implementierung die Kamera verschoben, beziehungsweise wurden die Strahlen nach Berechnung aus dem Ursprung und in kanonische Richtung mit Hilfe einer Transformationsmatrix verschoben und rotiert.

Strahlen - Objekt Schnittpunkt

Um Objekte in einem Raytracer anzeigen zu können müßen Schnittpunkte von Strahlen (engl. ray) mit den unterschiedlichen Objekten berechnet werden. Hierbei ist in der Regel lediglich der erste Schnittpunkt auf einem Strahl von Interesse.

Implizite Oberflächen: hier Kugeln und Zylinder

Implizite Oberflächen werden durch mathematische Gleichungen $f(x, y, z)$ beschrieben, welche für alle Punkt die auf der Oberfläche liegen $= 0$ ergeben. Für Kugeln benötigt man hierzu lediglich den Mittelpunkt der Kugel, sowie einen Radius. Die Punkte (x, y, z) welche die Gleichung

$$f(x, y, z) = (x - u)^2 + (v - v)^2 + (z - w)^2 - r = 0$$

erfüllen bilden die Oberfläche einer Kugel mit Radius r und Position (u, v, w). In dieser Arbeit wurden des weiteren noch Zylinder implementiert, bei welchen die Oberfläche ebenfalls auf eine solche Weise beschrieben werden kann.

Parametrische Oberflächen: hier Drei- und Vierecke

Parametrische Oberflächen sind Oberflächen welche durch eine Funktion $f(x, y)$ repräsentiert werden. Um die Kollision mit einer solchen Oberfläche zu berechnen muss man ein lineares Gleichungssystem lösen. Für Dreiecke bilden die Punkte welche die Gleichung

$$o + t \cdot d = (1 - b_1 - b_2)p_0 + b_1 * p_1 + b_2 * p_2$$

unter der Bedingung $b_1 \geq 0$ und $b_2 \geq 0$ und $b_1 + b_2 \leq 1$ erfüllen die Oberfläche. Für Vierecke ändert sich lediglich der letzte Teil der Bedingung zu $b_1 < 0$ und $b_2 < 0$.

AABBs

Zur Beschleunigung der Schnittpunktberechnung kann um komplexere Objekte ein Boundingvolume gelegt werden. Die Kollisionsberechnung mit diesem ist hierbei einfach und schnell zu berechnen und so müssen Strahlen, die das Boundingvolume nicht treffen nicht mit jedem Objekt des Inhaltes getestet werden. In der hier vorliegenden Implementierung wurden Axis-Aligned-Boundingboxes implementiert. Hierzu werden so genannte *slabs* berechnet, Intervalle entlang einer Achse. Mit Hilfe dieser *slabs* läßt sich feststellen ob die Linie das Boundingvolume schneidet.

Phong Shading und Phong Beleuchtungsmodell

Trift ein Strahl ein Objekt so wird an diesem zur Berechnung der „gesehenen“ Objektfarbe sowohl die Farbe des Objektes beachtet als auch die Beleuchtung mit Hilfe der verschiedenen Lichtquellen und deren Farben berechnet, dies übernimmt das Beleuchtungsmodell. Hierbei entsteht das Licht aus drei verscheidenen Komponenten. Die erste Komponente ist hierbei das ambiente Umgebungslicht. Die Zweite ist das diffuse Richtungslicht. Die Dritte Komponente ist das spekulare Glanzlicht. Für die Berechnung des hier verwendeten Beleuchtungsmodells wird neben dem Kollisionpunkt auch die Normale N des Punktes benötigt, dieses wird beim Phong Beleuchtungsmodell über das Objekt interpoliert. Zur Berechnung der Farbe werden sowohl die entsprechenden Komponenten der Farbe des Objektes $\text{Farbe}_{\{ambient,diffuse,specular\}}$ verwendet als auch die entsprechenden Farbkomponenten $E_{\{ambient,diffuse,specular\}}$ der Lichtquelle. Wobei die [Wort für tiefgestelltes Zeugs] die jeweiligen Farbkomponenten des Lichtes beschreiben. Die Beleuchtung an einer Stelle wird nach folgenden Formel berechnet:

$$\text{Farbe} = \text{Farbe}_{ambient} \otimes E_{ambient} + \text{Farbe}_{diffuse} \otimes E_{diffuse} \cdot (N \cdot L) + \text{Farbe}_{specular} \otimes E_{specular} \cdot (R \cdot V)^m$$

Hierbei ist N die Normale am Objekt, L die Richtung in welcher die Lichtquelle liegt R die Reflektionsrichtung des Strahles sowie V die Sichtrichtung auf das Objekt, also der Richtungsvektor des Strahles. $\otimes$ ist die komponentenweise Multiplikation der einzelnen Vektorwerte der Farben. m beeinflusst die Größe der Glanzlichtanteile. Hat die Szene mehr als eine Lichtquelle so entsteht das resultierende Licht als Durchschnitt aller Farbbeiträge der einzelnen Lichtquellen.

Schatten

Punktlichtquellen und gerichtete Beleuchtung

Erweitert man obiges Modell um eine Prüfung ob die entsprechenden Lichtquellen gesehen werden, so kann man leicht Schatten erzeugen. Hierzu wird bei einer Punktlichtquelle ein Strahl vom Punkt des Objektes, dessen Farbe soeben bestimmt wird, zur Lichtquelle geschickt und überprüft ob dieser auf dem Weg zum Licht ein Objekt trifft, ist dies der Fall so wird das Licht nicht gesehen. Handelt es sich um eine gerichtete Beleuchtung so darf kein Objekt in diese Richtung liegen, egal in welchem positivem Abstand.

Texturen

Statt an einem Kollisionspunkt eines Strahles mit einem Objekt eine Objekt globale Farbe zurückzugeben kann diese auch aus einer Textur gelesen werden. Bei Drei- und Vierecken ist dies am einfachsten, da man hier die Baryzentrische Koordinaten bereits bei der Kollisionsberechnung ermittelt und diese direkt zum Zugriff auf ein Bild verwenden kann. Bei Kugeln bedarf es einer aufwendigeren Berechnung des Punktes auf der Kugel in Koordinaten des Bildes.

Sampling

Durch wenige oder gar nur einen Strahl welche durch die Bildpixel gesendet werden kann es bei kleinen Objekten oder feinen Strukturen dazu kommen, dass diese nicht oder Falsch dargestellt werden. Diesen Effekt kann man dadurch umgehen, dass man mehr Strahlen durch ein Bildpixel sendet und aus diesen danach einen Farbwert rekonstruiert. Um den Effekt des Aliasings, bei welchem bei zu geringer Abtastrate, Zahl der Strahlen, statt einer feinen Struktur Muster entstehen welche nicht vorhanden sind, zu reduzieren ist es zudem ratsam die Strahlen nicht geordnet anzulegen. Werden jedoch viele Farbwert für ein Pixel berechnet und lediglich gemittelt, so wird das Bild unscharf. Für diesen hier erstellten Raytracer wurden verschiedene Samplingmethoden zum Vergleich implementiert.

Random Sampling

Beim Random Sampling werden durch ein Pixel mehrere Strahlen gesendet, die Position dieser ist hierbei zufällig und kann so dazu führen, dass eine unregelmäßige Abdeckung verschiedener Bereiche des Pixels erfolgt.

Stratified Sampling

Beim Stratified Sampling wird das Pixel in mehrere regelmäßige *Strata* eingeteilt, in jedem dieser Subregionen wird nun zufällig eine Position für einen Strahl ermittelt. Im Gegensatz zum Random Sampling verbessert sich die Abeckung des Pixels, da so keines der *Strata* nicht beachtet wird.

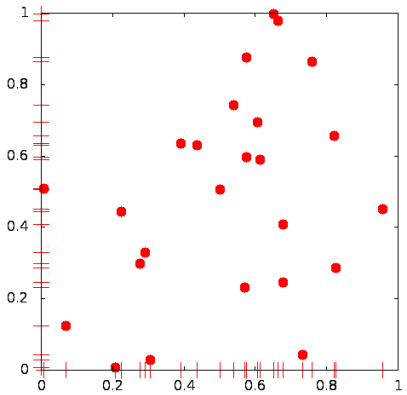
Poisson Sampling

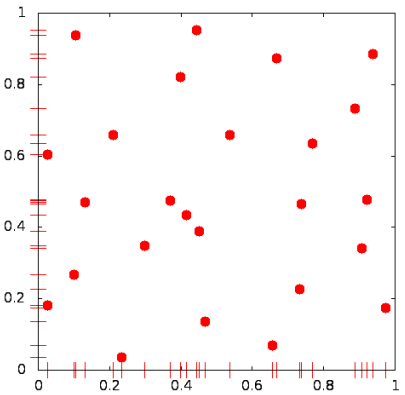
Für den hier implementierten Raytracer wurde das Verfahren des Poisson Sampling gewählt. Hierbei werden zufällig über das Pixel verteilt Orte gewählt, durch welche Strahlen gesandt werden sollen. Jedoch werden nur solche Positionen verwendet, die zu ihren Nachbarn einen gewissen Mindestabstand aufweisen. So wird die Häufung der Strahlen an einer Stelle vermieden.

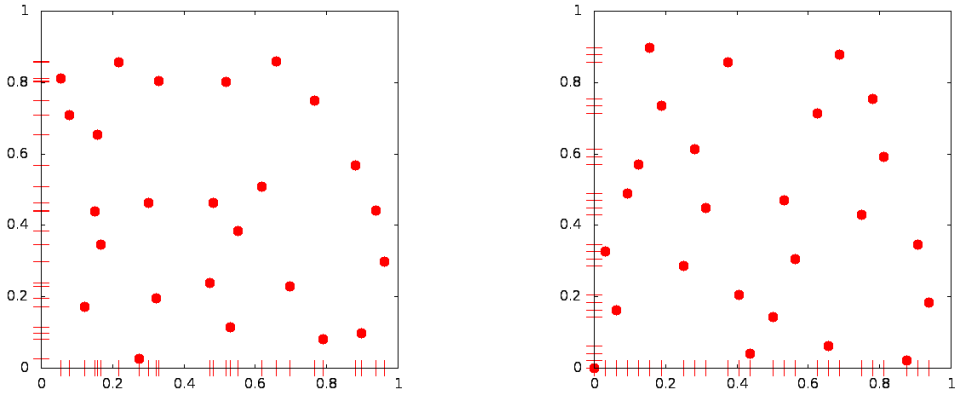
Halton Sampling

Das ausgeklügeletste Sampling-Verfahren welches hier implementiert wurde ist das sogenannte Halton Sampling. Bei diesem Verfahren wird die Missverteilung des Positionen für die Strahlen minimiert. Hierzu werden sowohl die X- als auch die Y-Komponenten der Position des Strahles anhand einer Hammerlesy Sequenz gewählt. Eine Hammerlesy Sequenz liefert Werte welche zwischen 0 und 1 liegen und diesen Bereich zu jedem Zeitpunkt möglichst gleich-verteilt abgedecken. Mit diesem Verfahren wird gewährleistet das der Bereich des Pixels möglichst gleichmäßig abgedeckt wird.

Ergebnisse der verschiedenen Sampling Methoden







Die von den verschieden Verfahren berechneten Positionen für Strahlen innerhalb eines Pixels. Von oben links nach unten rechts: Random Sampling, Stratified Sampling, Poisson Sampling, Halton Sampling (mit $p_1 = 2$, $p_2 = 7$). Zusätlich wurden die Positionen auch auf die X- und Y-Achse projiziert, so das einfacher ersichtlich ist ob das Pixel so in beiden Dimensionen ausreichend und gleichverteilt abgedeckt wird.

Rekonstruktion

Um aus den Farbwerten, welche die verschiedenen Strahlen die für den Bereich eines Pixels berechnen wurden einen konsistenten Farbwert zu wählen bedarf es einer Rekonstruktionstechnik. Diese gewichtet die Farbwerte anhand eines Kernels $f(d)$ und normalisieren am Schluss das Ergebnis.

Box Rekonstruktion

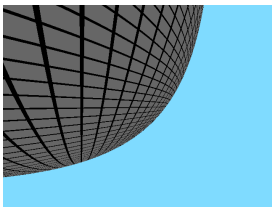
Ein einfaches Rekonstruktionsverfahren ist die Box-Rekonstruktion. Hierbei werden alle Farbwerte gleich gewichtet (f(d) = 1) und durch die Anzahl der Farbwerte geteilt, es wird also ein Mittelwert über alle vorhandenen Farbwerte der Strahlen gebildet.

Mitchell Rekonstruktion

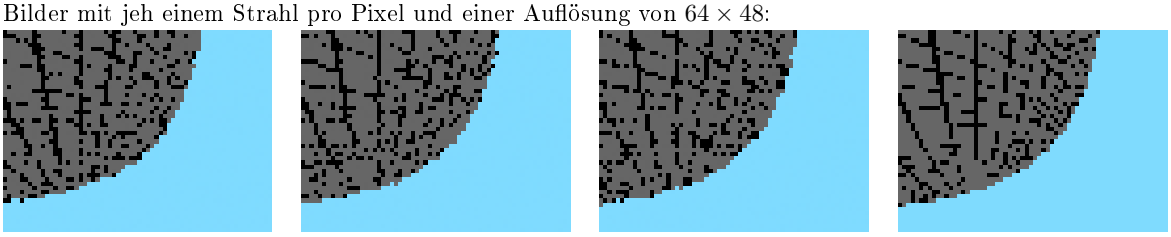
Die Mitchell Rekonstruktion gewichtet die Farbwerte mit einer Standart-Normalverteilung ihres Abstandes zum Pixelzentrum. Somit werden Farbwerte die nah an der Pixelmitte liegen stärker beachtet.

Vergleich verschiedener Sampling- und Rekonstruktionstechniken

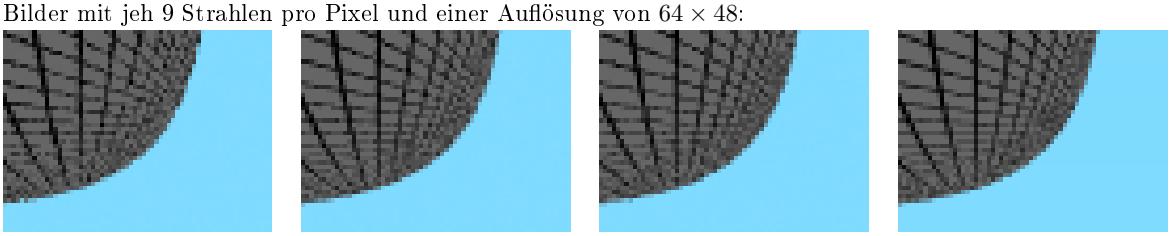
Bild mit 640×480 Pixeln:



Bilder mit jeh einem Strahl pro Pixel und einer Auflösung von 64×48:



Bilder mit jeh 9 Strahlen pro Pixel und einer Auflösung von 64×48:



Zu sehen sind hier von links nach rechts: Random Sampling, Stratifeid Sampling, Poisson Sampling und Halton Sampling jeweils mit Mitchell Rekonstruktion. Es ist gut zu erkennen das da Auftreten und Ausbleiben der Lininen beim Random Sampling mit einem Strahl pro Pixel sehr zufällig ist und diese beim Halton Sampling bereits konsistenter Auftreten oder Ausbleiben. Mit mehreren Strahelmn pro Pixel verschwimmen die Lininen bei allen Verfahren.

Hier noch mehr Vergleiche, aber lieber mit 4 rays/pixel

Ende Text.

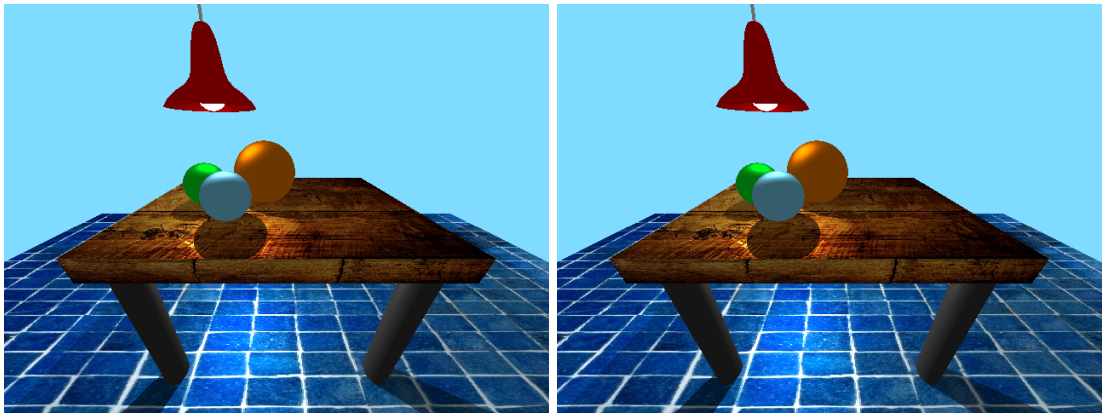
Beschleunigung der Schnittpunktberechnung

Zur Beschleunigung der Schnittpunktberechnung mit komplexeren Objekten eignen sich neben den oben genannten Boundingvolumes hierarchische Zerlegungen der zu zeichnenden Objekte oder des Zeichenraumes. In dieser Implementierung wurden die Objekte zerlegt. Hierzu wurde eine kd-Baum ähnliche Boundingvolumen-Hierarchie implementiert. Diese zerteilt rekursiv das zu zeichnende Objekt in je zwei Teil-Hierarchien.

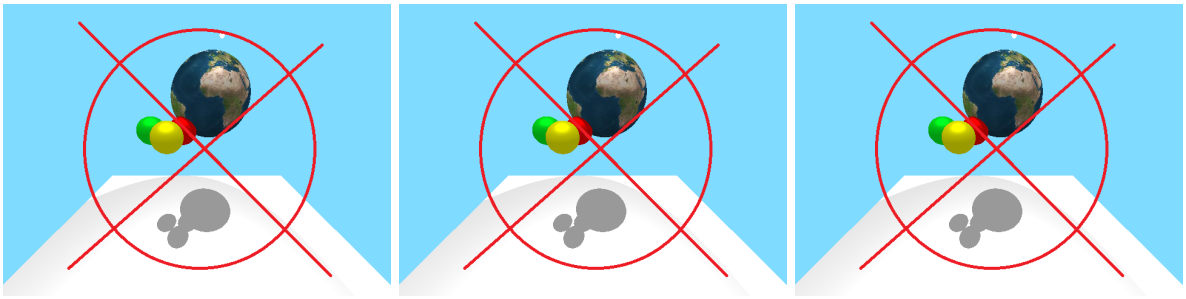
kd-Baum-Hierarchie

Ein kd-Baum zerlegt eine das komplette Objekt umschließende AABB mit Hilfe verschiedener Ebenen rekursiv in zwei Teil kd-Bäume. Schneidet der Strahl einen solchen Teilbaum nicht, so kann dieser von der Schnittpunktberechnung komplett ausgeschlossen werden und so die Berechnung erheblich beschleunigt werden. Der kd-Baum trennt hierbei in wechselnder Reihenfolge das (Teil-)Objekt in Teilräume bezüglich einer zu einer der Achsen orthogonalen Ebene. Zur einfacheren Traversierung der Datenstruktur wurden statt die Teilbäume mit Hilfe einer Ebene zu trennen AABBs für die zwei Teil-Hierarchen berechnet und verwendet. So teilt auch die hier implementierte kd-Baum-Hierarchie das Objekt in oben genannter Weise in je zwei Teil-Hierarchien. Für beide Teileobjekte werden nun AABBs berechnet die alle Primitive umschließen. Durch diesen Trick kommt es dazu dass sich die AABBs zum Teil überlagern, jedes Primitiv aber nur in einem der zwei Teil-Hierarchen gespeichert werden muss. Wenn ein Strahl nun mit der Hierarchie geschnitten werden soll, so wird zuerst geprüft ob der Strahl die alles umschließende AABB schneidet, Wenn ja wird rekursiv jeweils geprüft mit welcher der zwei Teil-Hierarchien der Strahl schneidet. So können schnell große Teile des Objektes verworfen werden. Danach werden nur die Um die Korrektheit und Effizienz zu überprüfen wurden folgende zwei Bilder gerendert. Die Lampe besteht hierbei aus [x] Dreiecken. Das linke Bild wurde hierbei ohne die Datenstruktur erstellt und benötigte 212

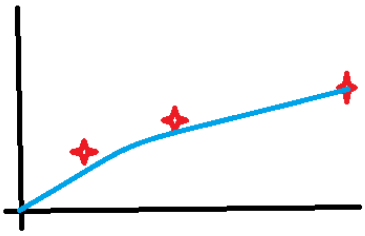
Sekunden zur Erstellung, das rechte Bild benötigte mit Datenstruktur lediglich 10 Sekunden.



Des Weiteren wurde getestet wie die Laufzeit bei Verwendung von zusätlichen Primitiven skaliert. So wurden verschieden viele Instanzen des selben Objektes in ein und die selbe Datenstruktur integriert.



Hierbei benötigten die Bilder in der Erstellung von links nach rechts: 10, 12 bzw 15 Sekunden. [ANPASSEN!]



Trägt man diese Zeiten in ein Diagram ein so kann ein anähernd logarithmisches Verhalten in der Anzahl der Primitiven vermutet werden. [ANPASSEN!]

Zusätzliche Funktionalität

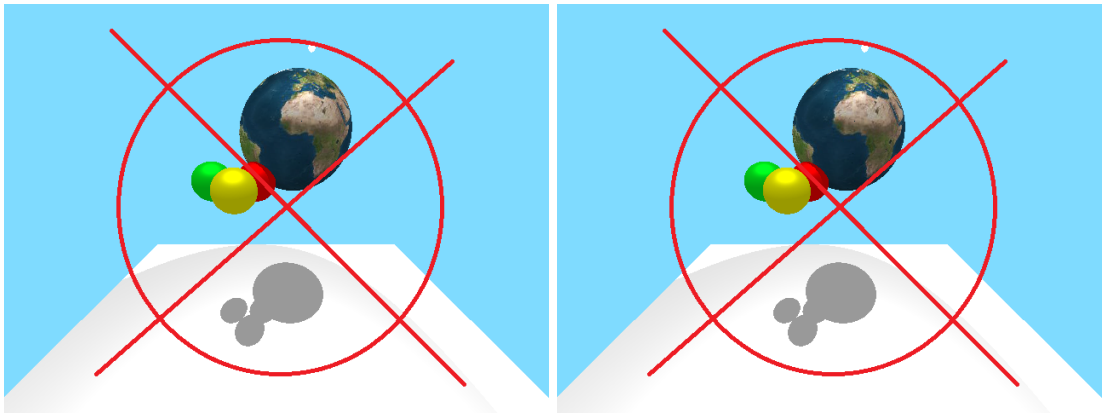
Reflektionen und Transparenz

Versendet man an einem Kollisionspukt eines Strahls mit der Szene rekursiv zwei weitere Strahlen so kann man einfache Transparenz und Reflektionen erzeugen. Hierzu wird ein Strahl in Reflektionsrichtung und ein Strahl durch das Objekt gesendet. Je nach Gewichtung der drei entstehenden Farben ensteht der Anschein von Reflektionen oder Transparenz.

Bump-Mapping

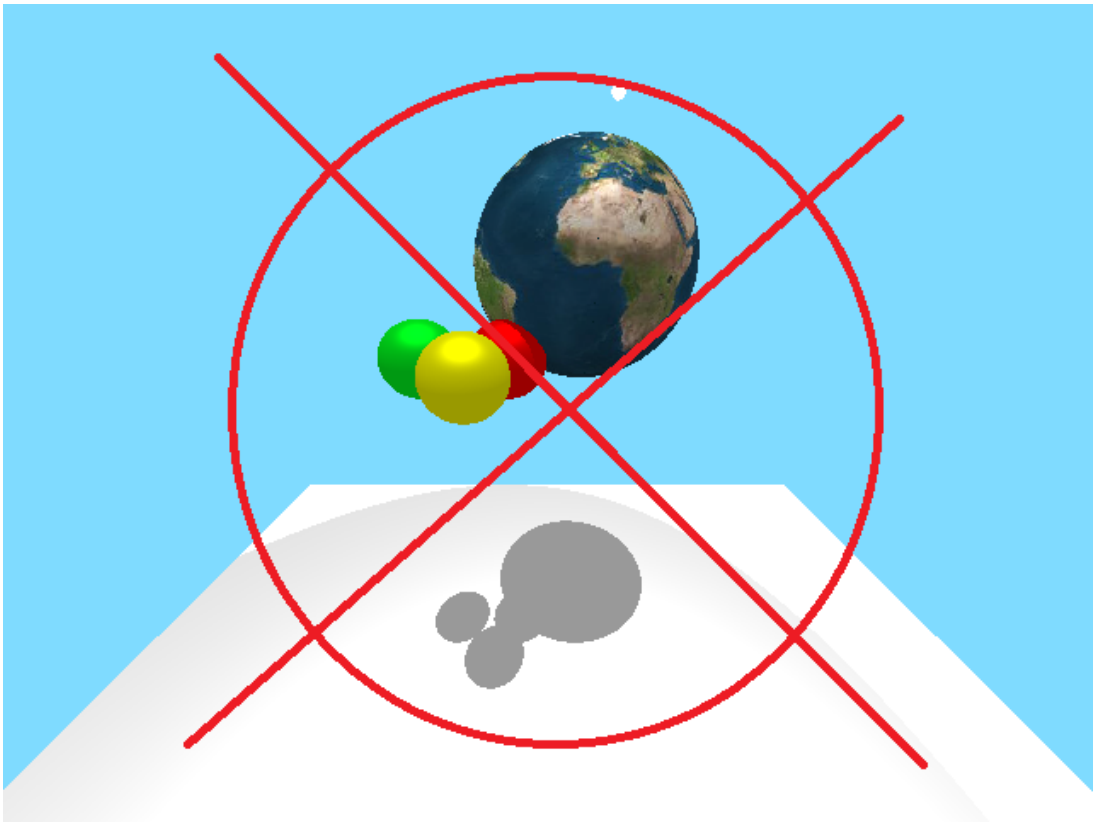
Ähnlich wie bei der Verwendung von Texturen zur Bestimmung der Farbe können Grauwertbilder auch zur Modifikation der Objektnormale verwendet werden. Durch die neue Normale werden Licht- und Schattenefekte erzeugt die den Eindruck von Struktur auf einer glatten Fläche vermitteln.Links ein Bild ohne

eine Bump-Map, rechts mit:



Ergebnisse

Ein Eindruck des implementierten Programmes:



Quellen

C++Mathe-Bibliothek

Für die hier verwendete Mathematik wurde die in dieser Implementierung die Mathebibliothek des Lehrstuhles für Mustererkennung und Bildverarbeitung von Prof. Brox verwendet. Copyright: Prof. Brox

Weiterer Code

Jeglicher weiterer Code wurde persönlich angefertigt. Qt und C++ wurden verwendet.

Texturen und Modelle