Universität Leipzig
Institut für Informatik
Abteilung für Bild- und Signalverarbeitung
Praktikum Computergrafik

SoSe 2024

Baldwin Nsonga
Lydia Müller

Aufgabenblatt 3

Sichtbarkeit durch Raycasting

Der zentrale Inhalt dieser Aufgabenserie wird es sein, Raycasting zu implementieren. Im Kontext der Sichbarkeit wird durch Raycasting berechnet, welche Objekte von einem von der Kamera ausgehenden Strahl getroffen werden und somit sichtbar sind. Dabei sollen Schnittberechnungen mit Kugeln und Dreiecken möglich sein. Informieren Sie sich über Raycasting ausgehend von der Vorlesung. Dieses Aufgabenblatt setzt voraus, dass Aufgabenblatt 2 vollständig gelöst wurde.

Folgendes muss in dieser Aufgabe implementiert werden:

1. *main(...)*: Kommentieren Sie den Code wieder ein, der für die Erzeugung der Kamera *cam* und dem Hinzufügen der Kamera zur Szene verantwortlich ist. Die Kameraposition ist im Beispielbild: [0,0,200].

2. *main(...)*: Erzeugen Sie 2 Objekte der Klasse *Sphere* und fügen Sie dieses mittels *Scene::addSphere(...)* zur Scene hinzu. Laden Sie zusätzlich drei Würfel.

3. *main(...)*: Führen Sie alle notwendigen Transformationen an den Modellen durch (siehe Abbildung 1).

4. *main(...)*: Erzeugen Sie ein Objekt der Klasse *SolidRenderer* und führen Sie *SolidRenderer::renderRaycast()* aus.

5. Die Funktion *SolidRenderer::renderRaycast()* ist gegeben und führt *SolidRenderer::computeImageRow(size_t row)* aus. Die Aufteilung in Zeilen ermöglicht eine einfach Parallelisierungsstrategie. Im Quelltext steht Ihnen die OpenMP parallelisierte Berechnung (auskommentiert) zur Verfügung.

6. Um optimierten Code mit Prallelisierung zu erhalten, sollten Sie mit *ccmake* (benötigt ggf. ein zusätzliches Paket) den *CMAKE_BUILD_TYPE* auf *Release* stellen.

7. Implementieren Sie die Methode *SolidRenderer::computeImageRow(size_t rowNumber)*. Hier soll für jede Splate innerhalb der Zeile ein Strahl generiert werden. Hierfür steht Ihnen die Funktion *Camera::getRay(size_t column, size_t row)* zur Verfügung. Befassen Sie sich mit der Struktur *HitRecord*. Dieses Objekt wird zum Speichern notwendiger Informationen für einen Strahl genutzt. Erzeugen sie für einen Pixel einen *HitRecord* und initialisieren Sie die für das Raycasting notwendigen Variablen (color, parameter, triangleId, sphereId). Innerhalb dieser Methode soll mit dem Aufruf von *Scene::intersect(...)* überprüft werden, ob ein Objekt getroffen wurde. Ist dies der Fall, setzen Sie die Farbe eines Pixels mit *Image::setValue(...)*. Wird nichts getroffen, so soll das Pixel die Hintergrundfarbe (z.B. weiß: *[1,1,1]*) erhalten.

8. Implementieren Sie die Methode *Scene::intersect(...)*. Gehen Sie dabei über alle Kugeln ihrer Szene und rufen Sie für jede Kugel *Scene::sphereIntersect(...)* auf. Bei einem Treffer muss der *HitRecord* aktualisiert werden. Gehen sie analog in allen Modellen Ihrer Szene über alle Dreiecke und rufen Sie *Scene::triangleIntersect(...)* auf. Beachten Sie dabei, dass Sie temopär ein Dreieck in Abhängigkeit ihrer Transformationen erzeugen müssen. Wenn eine Kugel **oder** ein Dreieck von einem Strahl getroffen wurde, gibt die Methode *true* zurück. Der Parameter *epsilon* kann genutzt werden um numerische Ungenauigkeiten zu behandeln.

9. Implementieren Sie den Schnittpunkttest für einen Strahl mit einer Impliziten Kugel *Scene::sphereIntersect(...)*. Das Verfahren aus der Vorlesung bzw. Praktikum ist zu verwenden. Weitere Hinweise finden Sie im Netz.

10. Implementieren Sie den Schnittpunkttest für einen Strahl mit einem Dreieck *Scene::triangleIntersect(...)*. Orientieren Sie sich dabei an der Vorlesung (Sichtbarkeit) bzw. der Folien aus dem Praktikum. Weitere Hinweise finden Sie im Netz.

11. *main(...)*: Erzeugen Sie Objekte der Klasse *Material*. Ein Material für das Bunny-Modell, 4 Materialien für das Cube-Modell und zwei weitere Materialien für die Kugeln. Setzen Sie die Farben des Materials und fügen Sie das Material mit Hilfe von *setMaterial* einem Modell/Kugel hinzu. (Beispiel Farben - Bunny: [0,1,0]; Cubes: [0.9,0.9,0.3], [0.9,0.4,0.3], [0.1,0.0,0.0], [0.9,0.9,0.9]; Kugeln: [0,0,1], [0,1,1])

12. Bei einem Treffer sollen die Materialeigenschaften des getroffenen Objektes mit der kürzesten Distanz zur Kamera ausgelesen werden und die Pixelfarbe entsprechend dieser Eigenschaften gesetzt wird. Dies soll umgesetzt werden, indem bei einer positiven Evaluierung der Funktion *Scene::intersect(...)* innerhalb der Funktion *SolidRenderer::computeImageRow(size_t rowNumber)* die Funktion *SolidRenderer::shade(HitRecord &r)* ausgeführt wird. In der Funktion *shade(HitRecord &r)* wird geprüft ob im *HitRecord* ein Treffer mit einem Objekt verzeichnet ist. Ist dies der Fall, so wird aus dem *Hitrecord* die Materialeigenschaften ausgelesen. In *shade* wird dann die *color* des *HitRecord* geschrieben. *Image::setValue(...)* wird in *SolidRenderer::computeImageRow(size_t rowNumber)* mit der aktualisierten *HitRecord* Farbe ausgeführt. Dieser Schritt präzisiert Schritt 7, in welchem nur zwischen *Treffer* und *kein Treffer* unterschieden wird. Beispielergebnis: Abbildung 1.

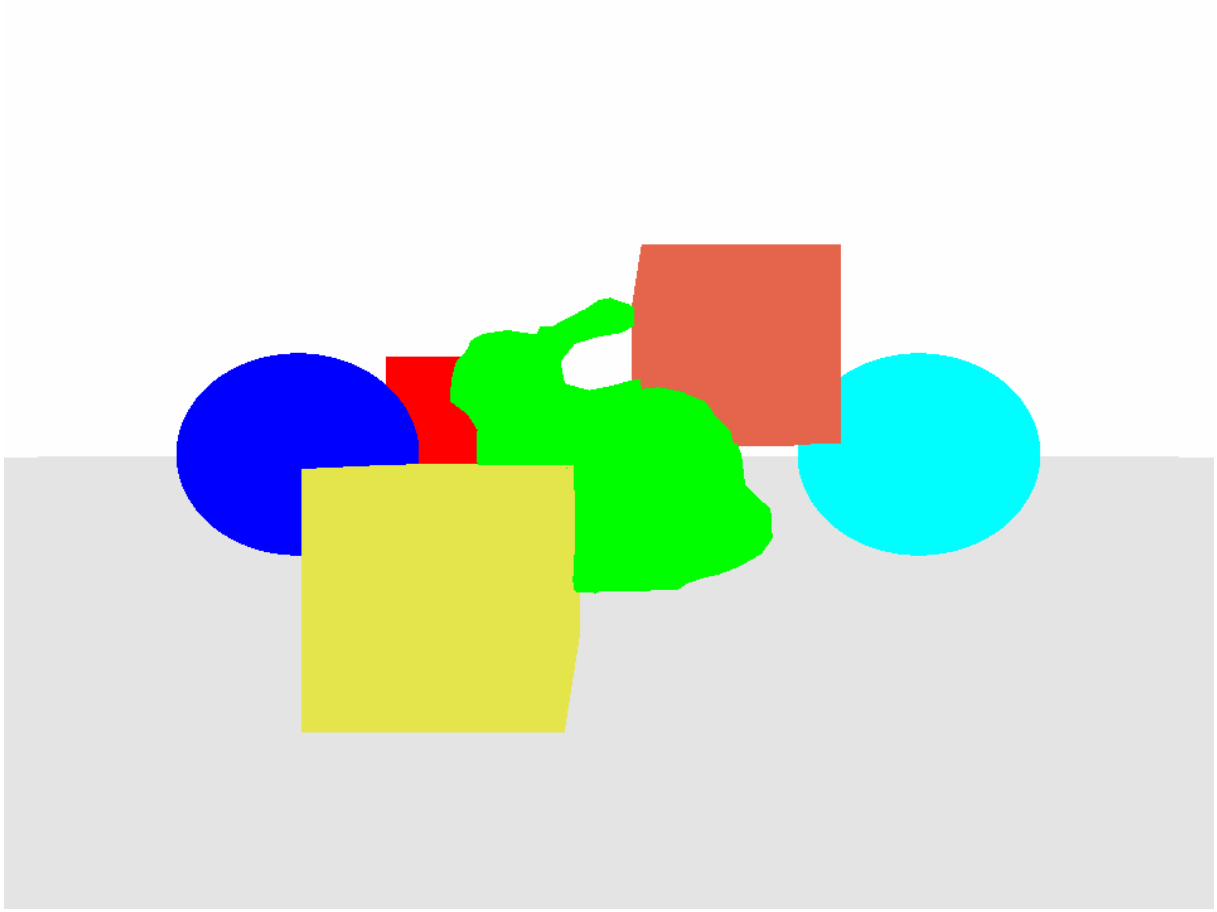

Abbildung 1: Zielsetzung der Sichtbarkeitsberechnung durch Raycasting (Beispiel) Anmerkung: Kugel 1: Position = [-150, 0, -30], Radius = 50; Kugel 2: Position = [150, 0, -30], Radius = 50. Das Bunny-Modell wurde 10 Einheiten in negativer y Richtung und 30 Einheiten in negativer z Richtung verschoben und um 170 Grad um die y-Achse rotiert (bitte beachten Sie, dass das Modell um die eigene Achse gedreht wurde). Folgendes wurde für die Cubes durchgeführt: Cube 1 Translation: [-60, -50, 0]; Cube 2: Translation: [60, 50, -50]; Cube 3: Translation: [-80, 10, -100]; Cube 4 Skalierung: [500, 0.01, 500], Translation: [0, -100, 0].