Institut für Visualisierung
und Interaktive Systeme

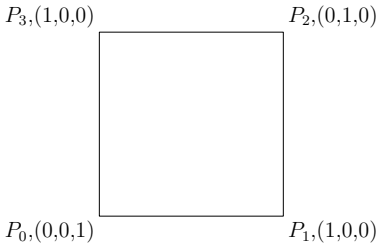
Dr. Guido Reina Dr. Michael Krone

5. Übungsblatt zur Vorlesung Computergraphik im WS 2017/18

Abgabe am Mittwoch, 29.11.2017 12:00 Uhr
Besprechung am Montag, 04.12.2017 17:30 Uhr

Aufgabe 1: Baryzentrische Koordinaten [3 Punkte]

1. Wieviele baryzentrische Koordinaten benötigt man für ein n-dimensionales Simplex?
2. Gegeben sei ein Quadrat für dessen Eckpunkte Farbwerte (R, G, B) im Wertebereich $[0, 1]$ definiert sind:



   Zerlegen Sie das Quadrat gedanklich in zwei Dreiecke entlang der Diagonalen von P_3 nach P_1 und interpolieren Sie die Farbwerte baryzentrisch. Beschreiben Sie (kurz!) in Worten die Verläufe der Farben zwischen den Eckpunkten (siehe auch nächste Teilaufgabe).
3. Wie ändert sich der Farbverlauf, wenn Sie das Quadrat entlang der anderen Diagonalen (P_2, P_0) teilen?
4. Die geometrische Interpretation baryzentrischer Koordinaten im Dreieck lässt sich auf Rechtecke erweitern. Überlegen Sie sich, wie Sie baryzentrische Koordinaten direkt in einem Rechteck ermitteln, ohne es vorher in Dreiecke aufzuteilen. Wieviele baryzentrische Koordinaten benötigen Sie und wie lauten diese an den Eckpunkten? Lässt sich das Verfahren problemlos auf Vierecke im Allgemeinen übertragen?
5. Wie würden Sie baryzentrische Koordinaten innerhalb eines Tetraeders ermitteln?

Aufgabe 2: Raytracing [7 Punkte]

In dieser Aufgabe sollen Sie einen einfachen CPU-Raytracer implementieren. Dafür finden Sie in Ilias ein dreiteiliges Programmgerüst, das wie folgt aufgebaut ist:

main Hier wird die Szene erstellt und das Raytracing durchgeführt.

sceneobject Enthält eine abstrakte Klasse `SceneObject`, die ein Objekt in der Szene repräsentiert. Außerdem die konkreten Objekttypen `Plane` und `Sphere`, die von `SceneObject` erben und zwei Methoden implementieren: `intersect(...)`, in der ein Schnitt des Objektes mit einem Strahl berechnet bzw. überprüft wird und `getSurfaceColor(...)`, eine Methode welche die Oberflächenfarbe des Objektes zurückgibt.

util Enthält diverse Utility-Klassen und -Funktionen: 2D-/3D-Vektorklassen mit gängigen mathematischen operationen, eine Ray-Klasse die einen Strahl repräsentiert, Funktionen zur Erzeugung und Vergleich von PPM-Bildern, sowie Zufallszahlengenerierung.

Das Programm erzeugt nach erfolgreichem Ausführen ein Ergebnisbild `result.ppm`. Dieses sollte im Auslieferungszustand des Programmgerüstes komplett schwarz sein.

Aufgabe 2.1: Strahlerzeugung und -verfolgung [4 Punkte]

Implementieren Sie zunächst die Strahlerzeugung und -verfolgung in `main.cpp`. Gehen Sie dabei wie folgt vor (die entsprechenden Stellen im Code sind mit `TODO` 2.1.n markiert):

1. Erzeugen Sie in der Methode `render(...)` für jeden Pixel auf der Bildebene einen Sichtstrahl, der die Kameraposition als Ursprung hat und durch den Mittelpunkt des Pixels geht. Benutzen Sie die vorgegebenen Parameter der Kamera und Bildebene. Rufen sie mit jedem so erzeugten Strahl die Methode `castRay(...)` auf.

2. Rufen Sie in der Methode `castRay(...)` die Methode `trace(...)` auf. Sollte der Strahl ein Szenenobjekt treffen, berechnen Sie außerdem den Schnittpunkt und geben die Farbe des Objektes an dem getroffenen Punkt zurück.

3. Iterieren Sie in der Methode `trace` über alle Szenenobjekte. Sollte mindestens eines der Objekte getroffen werden, speichern sie die Referenz auf das getroffene Object, welches sich am nächsten zur Kamera befindet in `hitObject`, sowie den Abstand zu diesem Schnittpunkt in `t_near`.

So lange lediglich die `intersect(...)`-Methode für Ebenen implementiert ist, sind auch nur diese Szenenobjekte im Ausgabebild (vgl. Abbildung 1) zu sehen.

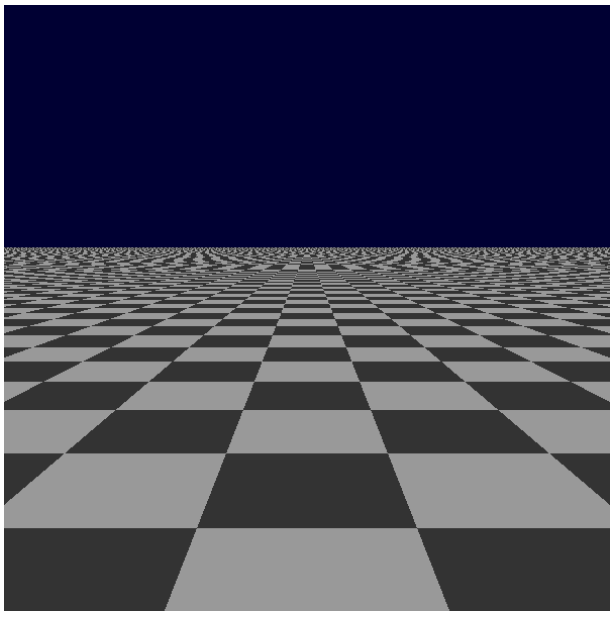


Abbildung 1: Referenzausgabe für Strahlerzeugung und -verfolgung. Nur die Ebene ist sichtbar.

Hinweis: Sie können Ihre Implementierung mit Hilfe des Tests `TEST_RAY_GENERATION` überprüfen, welcher das Ausgabebild auf Pixelebene gegen das Referenzbild vergleicht.

Aufgabe 2.2: Strahl-Kugel-Schnitt [3 Punkte]

Implementieren Sie die `intersect(...)`-Methode für Kugel-Objekte in der `Sphere` Klasse (in `sceneobject.cpp`), indem Sie den Schnittpunkt für Strahlen mit Kugeln berechnen.

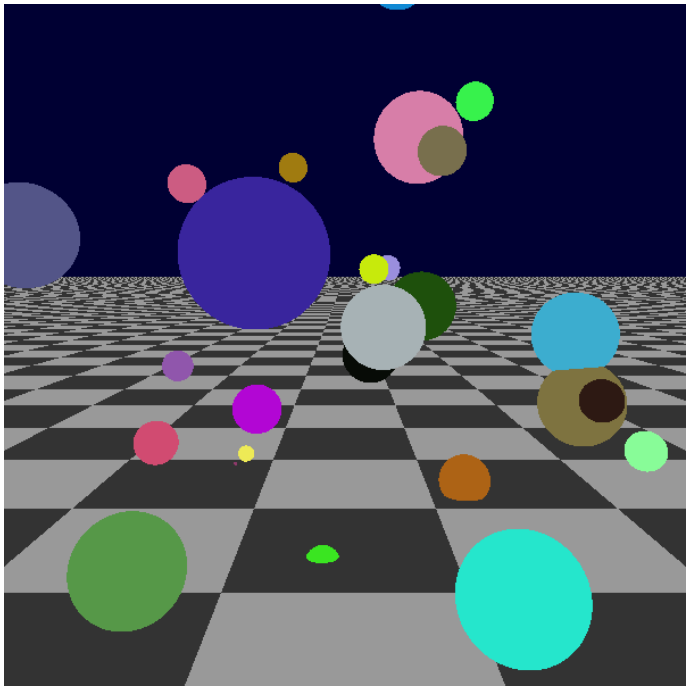


Abbildung 2: Referenzausgabe für die Strahl-Kugel-Schnittberechnung.

Hinweis: Sie können Ihre Implementierung mit Hilfe des Tests `TEST_SPHERE_INTERSECT` überprüfen, der das Ausgabebild auf Pixelebene gegen das Referenzbild (vgl. Abbildung 2) vergleicht.