HTWK

Abteilung Computergrafik
Fakultät Informatik und Medien

Computergrafik

Prof. Dr. Mario Hlawitschka

Sommersemester 2025

HTWK Leipzig, FIM

B Projektaufgaben

Im Sommersemester 2025 findet ein vorlesungsbegleitendes Projekt statt, in dem Renderingverfahren als Prüfungsvorleistung implementiert werden sollen. Die Aufgaben sollen in Gruppen von in der Regel drei Personen gelöst werden. Über die Aufteilung in der Gruppe, die genaue Umsetzung und die Programmiersprache entscheidet die Gruppe.

Ziel der Aufgabe ist es, dass jede Gruppe eine Software implementiert, die einen Renderer implementiert, der

- Geometriedateien einlesen kann,
- aus der Geometrieinformation, den Lichtquellen und der Kamera ein fotorealistisches Bild erzeugt,
- dieses Bild als Datei speichern kann oder auf dem Bildschirm anzeigt.

Die Implementierung kann frei nach eigenen Vorstellungen erfolgen. Die Quelltextausschnitte sollen als Hinweise dienen, wie so eine Struktur aussehen könnte.

Die Aufgaben sollten vorlesungsbegleitend bearbeitet werden. Einzelne Themen werden nach und nach in der Vorlesung behandelt.

B.1 Abschnitt 1: Datenhaltung der Geometrie

Grundlage jeder grafischen Ausgabe in der generativen Computergrafik ist eine Beschreibung der Geometrie. Wir benötigen also ein Grundgerüst, das Geometrie verarbeiten kann. Zur späteren Anzeige müssen Dreiecke in Verknüpfung mit Attributen auf den Dreiecken zugreifbar sein. Eine effiziente Speicherung der Dreiecke lässt sich auf unterschiedliche Arten realisieren. Eine einfache Version wäre die Speicherung der Dreiecke über die Koordinaten der Eckpunkte in einer Liste. Eine Alternative wäre die Speicherung aller Eckpunkte der Dreiecke in einer Liste und eine zweite Liste an Indizes, wo die Indizes angeben, welche Dreiecke verbunden sind.

Sinnvoll ist die Implementierung in einem Objekt, das die Daten hält und später durch Funktionen erlaubt, auf diese effizient zuzugreifen.

Im ersten Schritt sollte zum Beispiel die Möglichkeit gegeben sein, sich ein Dreieck nach dem anderen zurückzugeben, wobei das Dreieck zum Beispiel mit einer Farbe versehen ist.

Zum Testen ist es sinnvoll, Objekte aus Dateien einzulesen. Dafür bieten sich verschiedene einfache Datenformate an.

- OBJ: https://de.wikipedia.org/wiki/Wavefront_OBJ
- PLY: https://de.wikipedia.org/wiki/Polygon_File_Format
- STL: https://de.wikipedia.org/wiki/STL-Schnittstelle

B.1.1 Technische Umsetzung

Es ist ganz sinnvoll, sich für die Darstellung der Szene eine Struktur zu überlegen, die einerseits effizient, andererseits auch erweiterbar ist.

```
1 class Scene {
2     public:
3         std::vector<Object*> objects;
4         std::vector<Light> lights;       // Lichter kommen später
5         Camera camera;                   // Die Kamera kommt später
              hinzu
6 };
```

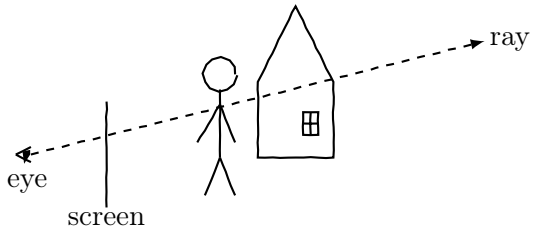
Aufgabe: – Erstellen Sie ein Grundgerüst für die Datenhaltung. Ermöglichen Sie das Anlegen von Objekten für die Geometrie und die Speicherung von Attributen auf den Eckpunkten oder Zellen. Ermöglichen Sie das Einlesen von Geometriedaten aus einem einfachen Dateiformat ihrer Wahl.

Bearbeitungszeit: 2 Woche(n)

B.2 Abschnitt 2: Die Kamera

Im einfachen Fall gehen wir von einer Lochkamera (Camera Obscura) aus. Hierbei fallen die Lichtstrahlen der Szene in die Öffnung der Kamera und treffen auf der Rückseite der sonst verschlossenen Box, wo sie das sichtbare Bild (auf dem Kopf und gespiegelt) darstellen. Analog dazu funktioniert das Modell des Auges des Betrachters vor dem Bildschirm: Licht trifft von der Szene durch einen Pixel auf dem Bildschirm in das Auge des Betrachters. Die Strahlen sind die gleichen wie im Kameramodell, jedoch entsteht das Bild in der Darstellung nicht auf der Netzhaut des Auges, sondern ist das, was auf dem Bildschirm dargestellt wird. Wir können dies durch folgende Information beschreiben:

- Den Augpunkt, einen Punkt, der die Position des Auges in der Welt darstellt (eye).
- Den Verschiebungsvektor vom Auge zur Mitte des Bildschirms (view) vor dem Betrachter. In der Regel nehmen wir an, dass dieser Vektor senkrecht zur Bildebene steht.
- Die Breite und Höhe des Bildschirms in Weltkoordinaten.
- Die Anzahl der Pixel des Bildschirms in beide Raumrichtungen.

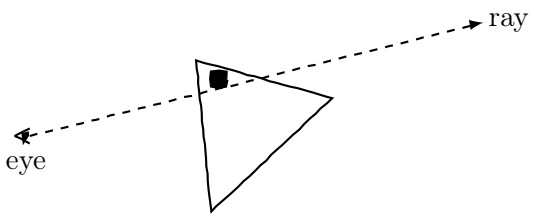


Weiter benötigen wir eine Repräsentation des erzeugten Bildes. Dieses Bild kann als 1D oder 2D Vektor gespeichert werden und sollte RGB-Komponenten der Farben enthalten. Während des Renderns (dem Erzeugen der Grafik) kann es sinnvoll sein, diese im Datentyp float abzuspeichern.

Eine Kamera könnte wie folgt implementiert werden:

```
1  // Beispiel für Daten der Kamera
2  class Camera {
3      public:
4          Vector3D eye;         // position of the camera
5          Vector3D view;        // direction vector from the eye to the
               center of the screen
6          float width;          // width in world coordinates of one pixel on
               the screen
7          float height;         // height in world coordinates of one pixel
               on the screen
8          int width_pixels;     // number of pixels on the screen in the
               horizontal direction
9          int height_pixels;    // number of pixels on the screen in the
               vertical direction
10
```

```
11          Vector3D get_pixel(int x, int y) const;
12          Ray get_ray(int x, int y) const;
13  };
14
15  // Ein Strahl ist durch einen Ursprung und eine Richtung beschrieben
16  class Ray{
17      public:
18          Point3D origin;
19          Vector3D direction;
20  };
21
22  // Trifft ein Strahl ein Objekt, so gibt es ein Hitpoint zurück, in dem
        wichtige Information über den Trefferpunkt enthalten sind.
23  // Diese werden dann in der aufrufenden Funktion verarbeitet
24  class Hitpoint {
25      public:
26          Point3D position;
27          float distance = std::numeric_limits<float>::max();
28          const Object* object = nullptr;
29  };
```



Aufgabe: – Fügen Sie eine Kamera für das Rendern in ihre Szene ein, Implementieren Sie ein Bildobjekt für die Anzeige der Daten, bestimmen Sie Sichtstrahlen durch die Pixel und berechnen den Strahlschnitt mit den Objekten aus der Datenstruktur. Testen Sie die Implementierung anhand einzelner Objekte in der Szene.

Bearbeitungszeit: 2 Woche(n)

B.3 Abschnitt 3: Bilderzeugung, Projektion oder Strahlschnitt

Um die Geometrie in die Bildebene zu bekommen gibt es zwei Verfahren, die beide ihre Vor- und Nachteile haben.

B.3.1 Projektion

Bei der Projektion wird die Geometrie mittels einer geometrischen Transformation in den Bildschirm, also in die Ebene, abgebildet. Da sich die Geometrie bei uns auf Dreiecke beschränkt, ist die nächste Aufgabe, das Dreieck in der Bildebene zu zeichnen. Dies erfordert eine »Rasterisierung« des Dreiecks, das ist die Abbildung des Dreiecks auf die Pixel des Bildschirms.

B.3.2 Strahlschnitt

Bei diesem Verfahren der Weg des Lichtes rückwärts vom Auge durch den Bildschirm in die Szene verfolgt. Auf dem Bildschirm ist das Dreieck sichtbar, das von dem Strahl vom Auge durch den Bildschirm zuerst getroffen wird.

Die Bilderzeugung geht in einer Schleife alle Pixel des Bildschirms durch, erzeugt für jedes Pixel einen Strahl vom Auge durch dieses Pixel in die Szene. Für diesen Strahl wird der Schnitt des Strahles mit dem Dreieck gesucht, das als zuerst vom Strahl geschnitten wird. Dieses Dreieck ist auf dem Bildschirm sichtbar. Das entsprechende Pixel im Bild wird anhand der Farbe des Dreiecks eingefärbt.

Am Ende dieser Aufgabe sollten Sie das erste Bild des Objektes sehen. Bei größeren Objekten wird die Berechnung länger dauern. Das Objekt ist in der Regel noch nicht besonders gut zu sehen, da die Beleuchtung fehlt.

```
// Grobübersicht Strahlverfolgung
HitPoint raytrace(const Ray& ray) {
    HitPoint hitpoint; // Startwert für distance
    for (auto &object : scene->objects) {
        auto hit = object.intersect(ray);
        if (hit && hit.distance > EPSILON && hit.distance <
           hitpoint.distance)
        {
            // neuen Treffer gefunden, der das bisherige Objekt verdeckt
            hitpoint = hit; //< neues Objekt merken
        }
    }
    return hitpoint;
```

```
void render()
{
    // doppelte For-Schleife, kann nativ parallelisiert werden
    for  (int y = 0; y < height; ++y) {
        for  (int x = 0; x < width; ++x) {
```

```
6               Ray ray  = scene->camera.getRay(x, y);
7               Color color;
8               HitPoint hitpoint = raytrace(ray);
9               if (hitpoint.distance < std::numeric_limits<float>::
                    max()) {
10                  color  = hitpoint.object->getColor(hitpoint);
11              } else {
12                  color  = scene->background; // Fallback, wenn nichts
                        getroffen wurde
13              }
14              image[y][x]  = color;
15          }
16      }
17  }
```

Aufgabe: – Fügen Sie eine Kamera für das Rendern in ihre Szene ein, Implementieren Sie ein Bildobjekt für die Anzeige der Daten, bestimmen Sie Sichtstrahlen durch die Pixel und berechnen den Strahlschnitt mit den Objekten aus der Datenstruktur. Testen Sie die Implementierung anhand einzelner Objekte in der Szene.

Bearbeitungszeit: 2 Woche(n)

B.4 Abschnitt 4: Beschleunigung

Die Geschwindigkeit des Verfahrens bisher hängt von zwei Dingen ab: der Anzahl der Pixel im Bildschirm und der Anzahl an Dreiecken in der Szene. Ziel dieses Abschnittes ist es, das Verfahren zu beschleunigen. Da wir die Anzahl der Pixel im Bildschirm nicht verringern wollen, müssen wir die Zahl der Dreiecksschnitte minimieren. Hierzu müssen wir also so effizient wie möglich unnötige Berechnungen entfernen, indem wir vorab die Dreiecke auswählen, die infrage kommen. Dazu bieten sich räumliche Suchstrukturen an, die entlang eines Strahls durchsucht werden können.

B.4.1 Aufteilung des Raumes in Würfelzellen

Unterteilt man den Raum in Würfel gleicher Größe, so lässt sich für einen Strahl effizient bestimmen, durch welche Würfel er geht.

In einem Vorverarbeitungsschritt ordnet man den Würfeln die Dreiecke zu (z. B. über Indexliste, die in den Würfeln gespeichert ist), die die Würfel schneiden. Beim Traversieren der Datenstruktur arbeitet man sich vom Auge entlang des Sehstrahls durch die Würfel, bis man das nächstgelegene Dreieck gefunden hat.

Unterteilt man den Raum in $10 \times 10 \times 10$ Würfel, so schneidet ein Strahl ca. 10–20 Würfel. Man erwartet eine Reduktion im besten Fall der zu betrachtenden Geometrie um Faktor 100, bei effizienter Implementierung einer Reduktion der Laufzeit auf fast ein Hundertstel.

B.4.2 Oc–Tree

Der Oc–Tree teilt den Raum rekursiv in acht gleich große Würfel. Die Anzahl der Unterteilungen kann von der Anzahl der in den Teilwürfeln zu berücksichtigenden Geometrieobjekten abhängen. Die Raumunterteilung und Zuweisung der Dreiecke entsteht erneut in einem Vorverarbeitungsschritt. Beim Strahlschnitt wird zuerst der äußere Würfel auf einen Treffer getestet, ist dies der Fall wird in die nächste feinere Ebene gesprungen und berechnet, welche der Teilwürfel infrage kommen und diese entlang des Strahles durchgegangen. Pro Teilwürfel wird das Verfahren rekursiv wiederholt, bis ein Dreieck gefunden wurde.

B.4.3 kd-Baum

Im Gegensatz zum Oc–Tree ist der kd–Baum ein balencierter Baum. Er ist unwesentlich komplizierter zu Erstellen und zu füllen, liefert aber durch die Balenciertheit eine gleichmäßigere Verteilung der Dreiecke im Baum und ist deshalb in der Regel schneller. `https://dcgi.fel.cvut.cz/home/havran/ARTICLES/cgf2011.pdf`

> **Aufgabe:** – Ergänzen Sie eine Beschleunigungsstruktur für ihre Daten, die die Strahlschnittberechnung beschleunigt. Testen Sie die Laufzeit im Vergleich zu der ersten Implementierung.
>
> Bearbeitungszeit: 2 Woche(n)

B.5 Abschnitt 5: Fiat Lux - Es werde Licht! (oder Schatten)

Bisher enthält unser Bild nur Information, welches Dreieck gesehen wurde. Mit einer natürlichen Beleuchtung hat das recht wenig zu tun. Wir benötigen in unserer Software also noch Information über Lichtquellen. Im einfachsten Fall sind dies Punktlichtquellen, von denen aus Strahlen in alle Richtungen des Raumes abgestrahlt werden.

Das einfachste Verfahren nutzt das Modell nach Phong zur Beleuchtung der Oberfläche, indem es vom Trefferpunkt die sichtbaren Lichtquellen identifiziert und das von dort eintreffende Licht mit den Oberflächeneigenschaften verrechnet, um festzustellen, wie viel Licht in die Richtung des Auges geht.

Phong–Beleuchtungsmodell (Teil 1)

Bei diesem Modell werden die Lichtquellen als Punktlichtquellen angenommen. Die Lichtquellen sind Teil der Szene und bei der Beleuchtungsberechnung muss ihre Information für jeden Trefferpunkt auf der Oberfläche des Objektes verfügbar sein.

Die Beleuchtungsberechnung ist in **??** dargestellt. Das Licht, das am Trefferpunkt in Richtung des Auges geht, wird als I bezeichnet und die Lichtquellen als L genannt. Hierbei wird das Licht aus verschiedenen Quellen eingesammelt.

Zunächst das direkte Licht von den Lichtquellen, wobei mit einem Strahl vom Trefferpunkt zu den Lichtquellen und einer vereinfachten Strahlschnittfunktion getestet wird, ob zwischen Trefferpunkt und Lichtquelle ein Objekt ist, das einen Schatten wirft. Das Licht jeder einzelnen Lichtquelle wird über die Entfernung der Lichtquelle abgeschwächt (je weiter die Entfernung der Lichtquelle, umso größer ist die Kugeloberfläche, auf die sich das Licht verteilt). Weiter wird der effektive Wirkungsquerschnitt der Fläche in Bezug auf die Lichtquelle benötigt. Dieser ist maximal bei senkrechtem Lichteinfall und nimmt dann bis zu flachem Lichteinfall auf der Oberfläche ab. Diese Abschwächung lässt sich über das Skalarprodukt der normierten Normalen und des Richtungsvektors zum Licht bestimmen. Zuletzt spielt die Oberflächeneigenschaft, gespeichert als Farbe der Oberfläche eine Rolle, die angibt, in welchen Farbkanälen im RGB-Modell das Licht erhalten bleibt und in welchen es abgeschwächt wird.

Weitere Modelle

Andere in der Vorlesung besprochene Modelle können den Fotorealismus erhöhen und tragen zu natürlicheren Bildern bei.

> **Aufgabe:** – Implementieren Sie ein Beleuchtungsmodell für die Oberflächen, sodass das Ergebnisbild einen 3D Eindruck erweckt.
>
> Bearbeitungszeit: 2 Woche(n)

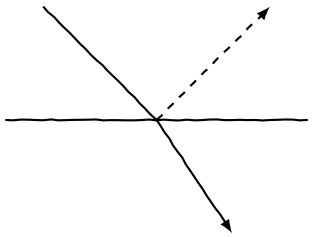
B.6 Abschnitt 6: Rekursive Verfahren - Spieglein, Spieglein an der Wand...

Kommen spiegelnde oder transparente Objekte in der Szene vor, so muss sichergestellt werden, dass das Licht korrekt berechnet wird. Dafür ergänzen wir das bestehende Modell.

B.6.1 Phong-Modell Teil 2

Die zweite Komponente stellt Reflexionen auf der Oberfläche dar. Dazu wird für jeden Trefferpunkt der Strahl bestimmt, der die ideale Spiegelung der zu berechnenden Lichtrichtung darstellt. Aus dieser Richtung muss das Licht eingesammelt werden, was durch einen rekursiven Strahlschuss erfolgt. Es wird also vom Trefferpunkt auf der Oberfläche ein neuer Strahl generiert, der dann mit genau der gleichen Funktion wie das Rendern der Pixel weiter in die Szene verfolgt wird. Die Rekursion kann abbrechen, wenn zu wenig Licht aus der Richtung erwartet wird, oder die vorgegebene Rekursionstiefe überschritten wird. In vielen Fällen reicht ein Wert von 3–5 aus. Bei stark spiegelnden Szenen muss ein höherer Wert gewählt werden.

Die Dritte Komponente ist die der Lichtbrechung. Ist das Material (halb-)transparent, so muss ein gebrochener Strahl anhand von Snells Gesetz bestimmt werden, der ebenfalls rekursiv in der Szene verfolgt wird.



Aufgabe: – Erweitern Sie das Verfahren um den rekursiven Strahlschuss um Spiegelungen und ggf. Transparenz zu unterstützen. Beeindruckend wirken die Ergebnisse in Szenen mit vielen mehreren spiegelnden Kugeln.

Bearbeitungszeit: 2 Woche(n)

B.7 Abschnitt 7: Das spezielle Etwas

Zum Abschluss soll das Modell weiter verbessert werden. Nutzen Sie eine weitere Optimierung zur Verbesserung der von Ihnen erzeugten Bilder aus der Vorlesung.

- Environment Maps
- Texturen
- Photon Mapping

- Radiocity
- Path Tracing
- Bi-Directional-Path-Tracing
- Metropolis Light Transport

Aufgabe: – Ergänzen Sie den Fotorealismus oder erhöhen die Geschwindigkeit Ihrer Implementierung durch einer von Ihnen gewählten Erweiterung des Verfahrens.

Bearbeitungszeit: 2 Woche(n)