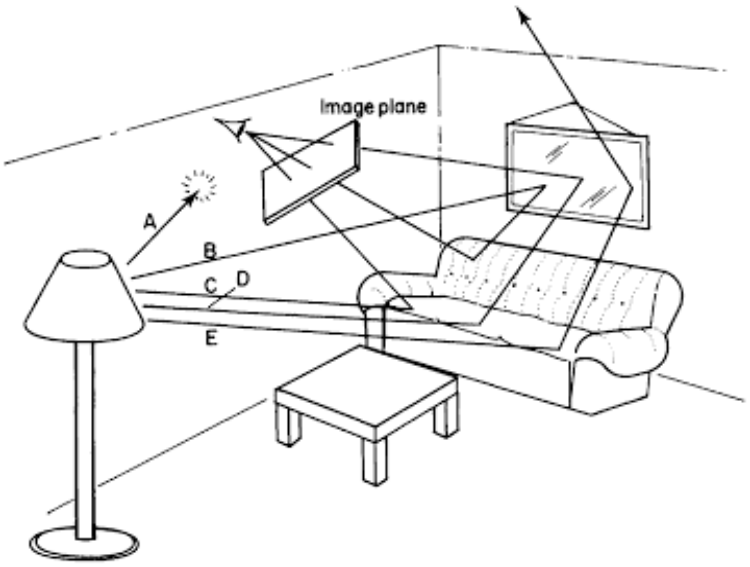
Projekt 1: Raytracer auf der GPU

Der Grundgedanke von Grafikkarten ist die Erzeugung von zweidimensionalen Bildern anhand von dreidimensionalen Objekten. Die Objekte werden durch ihre Oberfläche beschrieben, welche meist aus primitiven mathematischen Objekten besteht. Diese können Dreiecke, Linien oder beliebige Polygone sein. Das Verfahren zur Generierung des Bildes nutzt ein Rasterisierungsalgorithmus, um die einzelnen Pixel des Bildes einzufärben. Der größte Teil dieses Prozesses ist in Hardware auf der Grafikkarte implementiert.
Das Gegenstück zu diesem Verfahren stellt der Raytracer dar. Hier werden einzelne Strahlen von einem Augpunkt durch jedes Pixel des Bildes in die Szene gesendet. Diese Strahlen werden durch die Szene verfolgt und mit nahen Objekten geschnitten. Sobald ein Strahl auf ein Objekt auftrifft, werden mit Hilfe von Beleuchtungsmodellen und Materialeigenschaften entweder weitere Strahlen generiert, oder der zugehörige Pixel eingefärbt. Der Vorteil gegenüber der Rasterisierung ist die mögliche physikalische Genauigkeit, sowie die Vermeidung des Verdeckungsproblems bei komplexen Szenen.



Ziel

Das Ziel des Projekts ist die Implementierung eines Raytracers auf der GPU mit Hilfe von OpenCL oder CUDA. Die Szene soll aus mehreren Objekten bestehen, welche aus beliebigen Sichtrichtungen aufgenommen werden sollen. Der Einfachheit wegen kann die Szene nur aus impliziten Objekten wie Kugeln bestehen. Diese erlauben eine einfachere und schnellere Berechnung der Schnittpunkte. Die Anforderung an das Projekt ist die Verfolgung der Strahlen bis mindestens zum ersten Objekt. Weitere Verfolgung ist optional. Das Ergebnis soll über die Grafikpipeline mit OpenGL auf den Bildschirm ausgegeben werden.
Das Grundprinzip ist durch Sanders et. al. [1] erklärt. Für diverse Problemstellungen während der Implementierung bieten Singh et. al. [3] und Zlatuska et. al. [2] diverse Lösungen an.
Das Rendering des Raytracing beschränkt sich in diesem Projekt auf das Rendering von Punkten. Jeder Pixel in der Bildebene kann als einfacher Punkt an OpenGL weitergegeben werden. Man muss also jederzeit wissen, wo jeder Pixel der Bildebene sich in der Szene befindet.

Geschätzter Aufwand

- CUDA / OpenCL: 90 %
- OpenGL: 10 %

Themen

- Verfolgung der Strahlen vom Auge durch das Bild in die Szene
- Schnittpunktberechnung mit impliziten Objekten (Kugeln)
- Mehrere implizite Objekte (Kugeln)
- Optional: Schattenfühler zu Punktlichtquellen
- Optional: Supersampling

Evaluation

- Verschiedene Auflösungen der Bildebene (Anzahl Strahlen)
- Kamera bewegen und verschiedene Perspektiven rendern
- Anzahl Objekte und deren Positionen und Größen variieren
- Objekte verschieden einfärben
- Optional: Anzahl Punktlichtquellen und Positionen variieren

References

[1] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming.* Addison-Wesley Professional, 2010.

[2] Jag Mohan Singh and PJ Narayanan. Real-time ray tracing of implicit surfaces on the gpu. *IEEE transactions on visualization and computer graphics*, 16(2):261–272, 2010.

[3] Martin Zlatuška and Vlastimil Havran. Ray tracing on a gpu with cuda–comparative study of three algorithms. 2010.