Aufgabe 1: Simple Volume Renderer (25 Punkte)

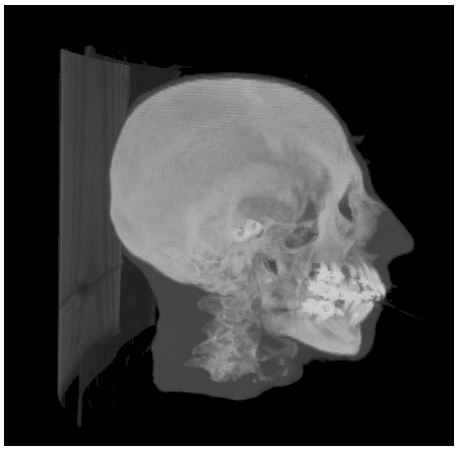
Mindestanforderung: (ToDo Hilfe im Code)

- Visualisierung eines Volumendatensatzes mittels Raycasting und MIP **(12 Punkte)**

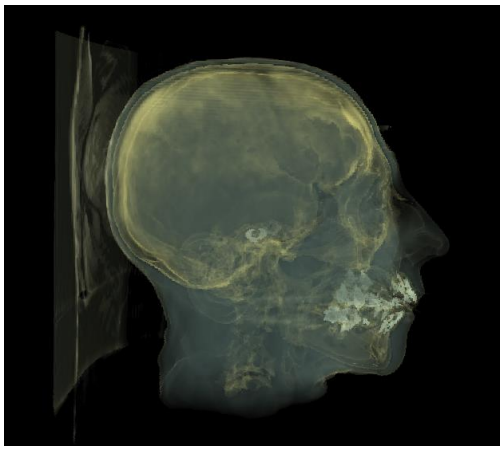
Weitere Punkte: (frei, ohne ToDo Hilfe)

- Alpha Compositing [Transferfunktion darf fix sein] **(6 Punkte)**
- Interaktive Tansferfunktion (Farbe + Dichte) **(7 Punkte)**

Diese Aufgabenstellung behandelt das Rendern von 3D Volumendaten. Das Rendern von Volumendaten ermöglicht es Bereiche in den Daten transparent oder halb-transparent darzustellen, um das Innere eines Datensatzes erforschen zu können. Dadurch können Datensätze in ihrer Gesamtheit besser verstanden werden.



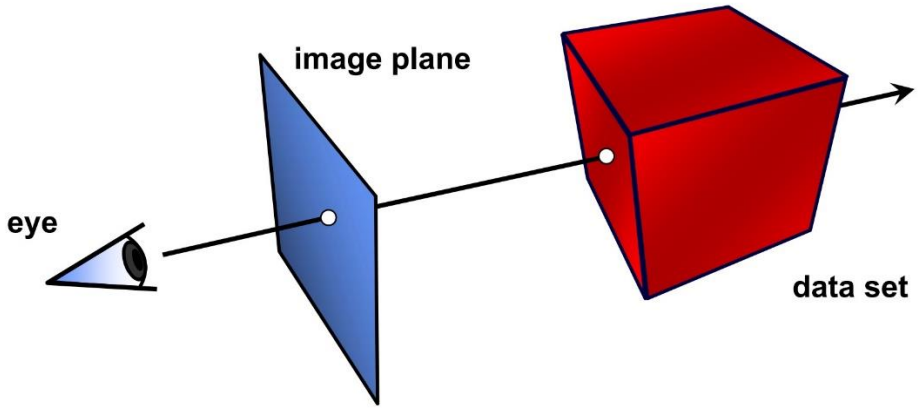
MIP



Alpha-Compositing (Beispiel)

In dieser Aufgabe soll ein einfacher Volume Renderer auf der **GPU** implementiert werden, der mit Hilfe von **raycasting** Volumendaten darstellen kann. Raycasting ist ein gebräuchliches Verfahren, um ein Bild aus Volumendaten zu berechnen, ohne vorher geometrische Primitive (z.B. Polygone) aus den Volumendaten generieren zu müssen.

Favoritenstraße 911 / E193-02,A-1040 Wien,Austria
Tel. +43 (1) 58801-193201, Fax +43 (1) 58801-193209
www.cg.tuwien.ac.at



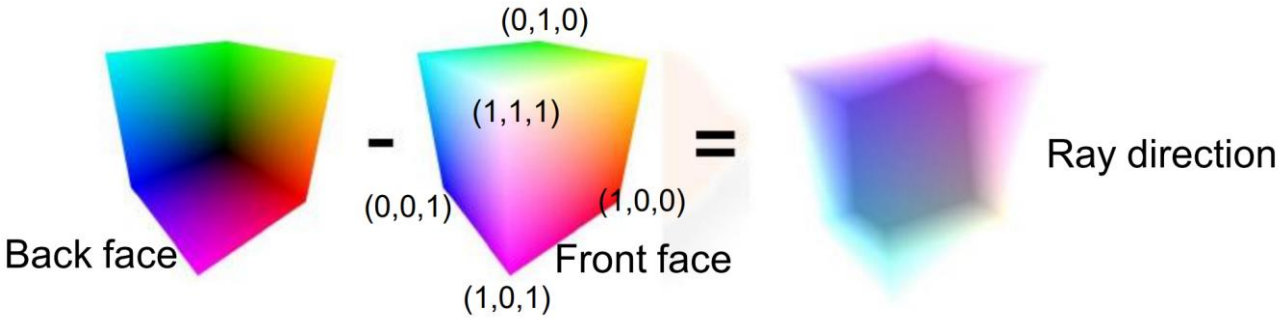
Um ein Bild mittels raycasting zu erzeugen wird pro Pixel ein Blickstrahl (ray) durch die Bildebene geworfen (casting). Der Blickstrahl wird mit dem Volumen geschnitten. Die Volumsdaten sind in einem kartesischen 3D-Gitter angeordnet. An den Gitterpositionen sind Voxelinformationen gespeichert (Dichtewerte). Entlang des Blickstrahls wird an regelmäßigen Positionen (samples) die Dichte ausgelesen, und je nach Kompositionsverfahren ein Farbwert für den aktuellen Strahl ermittelt. Eines der einfacheren Kompositionsverfahren ist die **Maximum-Intensity-Projektion** (MIP), bei der immer nur das Sample mit dem größten Dichtewert verwendet wird, um den Farbwert des Strahls zu ermitteln.

Bei MIP wird entlang des Strahls immer nur der größte Dichtewert berücksichtigt. Eine Alternative dazu ist **Alpha-Compositing** (Info), wo alle Werte entlang des Strahls akkumuliert werden. Der Farb- und Transparenzwert jedes Samples ergibt sich anhand einer vorher definierten Transferfunktion. Diese ordnet jedem Dichtewert im Volumen eine Farbe und Transparenz zu. Diese Werte sollen dann entlang des Strahls mittels des Over-Operator (Info) kombiniert werden. Es wird empfohlen Front-To-Back-Compositing (Info) zu verwenden, damit die Berechnung früher abgebrochen werden kann, wenn der maximale Alphawert erreicht wurde (Early Ray Termination).

Vorgehensweise

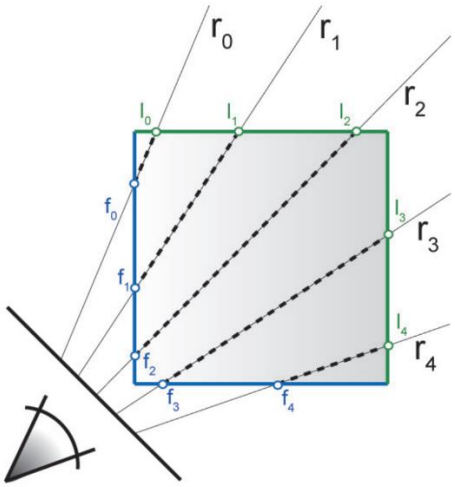
Die grundlegende Idee von GPU-based raycasting ist, dass das ganze Volumen als 3D Textur auf der GPU gespeichert wird und in einem fragment shader Strahlen durch das Volumen zu schießen, um den jeweiligen Farbwert zu berechnen. Jeder Pixel entspricht dabei einem Strahl. Hierfür gibt es verschiedene Implementierungen. In dieser Aufgabe soll ein multi-pass Ansatz implementiert werden. Das Rendern des Volumens läuft also in mehreren Renderschritten ab und die Zwischenergebnisse sollen in Framebuffer Objects (FBO) gespeichert werden.

Für das raycasting müssen zuerst die Richtungen der Strahlen berechnet werden. Eine einfache Methode um an die Anfangs- und Endpunkte von den Strahlen zu kommen ist die front faces und back faces von der Bounding Box des Volumens zu rendern und dann in einen FBO abzuspeichern.



[Krugger and Westermann 2003]

Zieht man die Koordinaten der front faces von den Koordinaten der back faces ab bekommt man die Richtung der geschossenen Strahlen. Die 3D Koordinaten des Volumens sind hier mit RBG-Farbwerten illustriert.



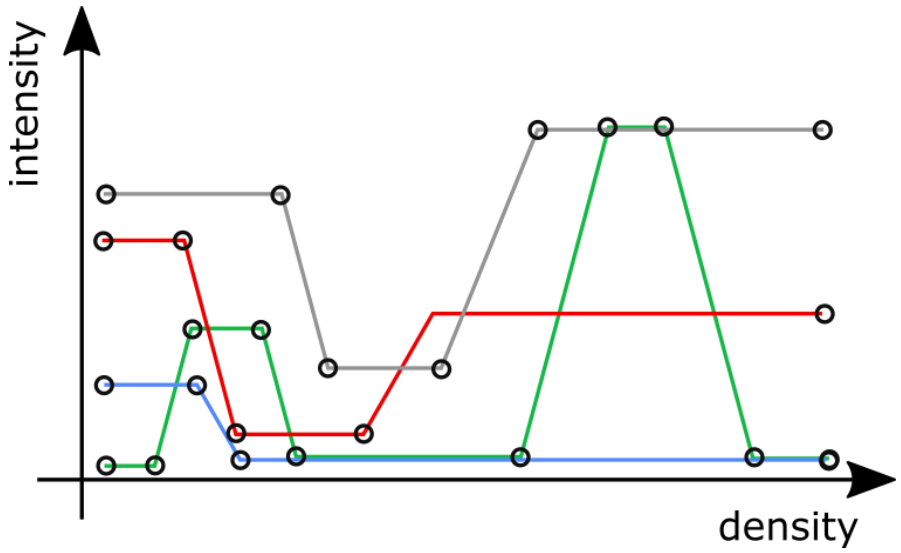
In dem nächsten Schritt soll das raycasting durchgeführt werden. Um das raycasting zu starten kann einfach ein Viereck gerendert werden, welches die ganze Bildfläche umfasst. Beim raycasting wird dann in regelmäßigen Schritten entlang der Strahlen die Dichtewerte abgetastet (Im Bild: Start bei $f_0 - f_4$ und Ende bei $l_0 - l_4$). Die Abtastrate hat hierbei Einfluss auf die Performance und die Qualität des resultierenden Bildes.

Zusammenfassung:

1. Generierung der front faces: Render die Koordinaten der front faces in eine Textur
2. Generierung der back faces: Render die Koordinaten der back faces in eine Textur
3. Raycasting: Benutze die Startpostition von den front faces und taste das Volumen entlang der Strahlrichtung ab
4. Compositing: Benutze ein compositing Verfahren (MIP, Alpha Compositing), um die Farbwerte zu berechnen

Interaktive Transferfunktion:

Transferfunktionen werden benutzt um bestimmte Funktionsbereiche (Bereiche von Dichtewerten) auf bestimmte Materialien (Farbwerte, Opazität) abzubilden. Mit solchen Transferfunktionen kann man z.B. bestimmen, dass Dichtewerte, die typisch für Knochen sind, eine andere Farbe bekommen als solche die typisch für Gewebe sind. Für die Implementierung als fixe Transferfunktion kann man diese Werte einmalig direkt im Sourcecode festlegen. Für die interaktive Transferfunktion soll ein QT Widget implementiert werden, dass in der Lage ist Funktionen mit Hilfe von Kontrollpunkten anzupassen. Wie so etwas aussehen könnte ist hier zu sehen:



Hier werden die verschiedenen Dichtewerte auf verschiedene Intensitäten der Farben Rot, Grün und Blau, sowie der Opazität abgebildet. Diese Funktionen sollen dann dem Shader übergegen werden, damit dieser sie beim Rendern benutzen kann.

Hinweise:

Für die Implementierung wird ein Framework angeboten, welches die grundlegenden Funktionen bereits zur Verfügung stellt. Das Framework wird als Visual Studio Projekt bereitgestellt und basiert auf OpenGL und QT. QT bietet Unterstützung bei der GUI und Wrapper für OpenGL Funktionen an, welche das Programmieren mit OpenGL vereinfacht. Für das Projekt muss QT nicht installiert werden. Alle notwendigen Bibliotheken sind direkt im Framework mitgeliefert.

Die wichtigsten Punkte beim implementiern sind die Folgenden:

Für das rendern müssen die shader *cube.glsl* und *raycasting.glsl* implementiert werden. Die Shader können zur Laufzeit angepasst werden. So können Änderungen direkt getestet werden. Kompilierfehler werden in der Konsole ausgegeben.

In der Klasse GLWidget muss die Rendering Routine (*paintGL*) implementiert werden.

Zusätzliche Referenzen:

GPU Gems: Volume Rendering Techniques
http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch39.html

Direct Volume Rendering
https://graphics.ethz.ch/teaching/former/scivis_07/Notes/stuff/StuttgartCourse/VIS-Modules-06-Direct_Volume_Rendering.pdf

Advanced Illumination Techniques for GPU-Based Volume Raycasting
https://www.vrvis.at/publications/pdfs/PB-VRVis-2008-027.pdf

Real-Time Volume Graphics
https://pdfs.semanticscholar.org/3661/5eccb10d00be0c26cebec4793f7d34864219.pdf

4D Volume Rendering
http://www.nvidia.com/content/GTC/documents/1102_GTC09.pdf

Acceleration Techniques for GPU-based Volume Rendering
https://www.evl.uic.edu/cavern/teranode/research/shalini/papers/Viz/Other/acceleration_gpu_rendering_viz03.pdf