

## Übungen zu Bildverarbeitung und Graphikprogrammierung in C++

### Aufgabe 28 (H) Offscreen Rendering und Post Processing

Für viele Anwendungen in der Visualisierung, Bildverarbeitung und wissenschaftlichen Rechnens auf der GPU wird das endgültige Resultat durch mehrere Passes auf der GPU berechnet. Um die Berechnungen mit maximaler Genauigkeit durchzuführen werden so genannte Offscreen Render Targets genutzt. In dieser Aufgabe werden Sie eine Möglichkeit kennenlernen um eine OpenGL Szene in ein Offscreen Render Target zu zeichnen und anschliessend mit GLSL Shadern in einem Post Processing Pass zu filtern.

Für die Bearbeitung dieser Aufgabe wird Ihnen eine Beispielapplikation zur Verfügung gestellt in der Sie noch einige Funktionen implementieren müssen. Für Offscreen Rendering wurde die Klasse `FramebufferObject` zur `GLLib` hinzugefügt. Die Klasse von Aaron Lefohn et al bietet einen effizient programmierten Wrapper um die OpenGL Framebufferobjekte an.

- Vervollständigen Sie die Implementierung der Funktion `initFBO` in dem Sie Code für die Erzeugung und Initialisierung eines Framebufferobjects hinzufügen. Fügen Sie die Textur `renderTex` an den Attachment Point `GL_COLOR_ATTACHMENT0_EXT` und den Renderbuffer `depthRB` an den Attachment Point `GL_DEPTH_ATTACHMENT` an. Prüfen Sie ob das Framebufferobject korrekt initialisiert ist mit der Methode `IsValid` und geben Sie etwaige Fehlermeldungen auf `std::cout` aus.
- Vervollständigen Sie die Implementierung der Funktion `offscreenPass`. Für den offscreen Render Pass soll der Viewport auf die Höhe und Breite des Render Targets gesetzt werden. Fügen Sie Code hinzu der dies realisiert und danach wieder den Viewport auf seinen ursprünglichen Wert zurück setzt. Fügen Sie Code zum Offscreen Rendering hinzu. Das Framebuffer Objekt sowie das gegebene GLSL Programm `OffscreenPass` sollen gebunden werden, bevor die Szene mittels der Funktion `renderGeometry` gerendert wird.
- Vervollständigen Sie die Implementierung der Funktion `display`. In `display` soll zuerst in einem offscreen Pass eine Szene in eine Texture gerendert werden. Danach soll das Ergebnis auf dem Standard OpenGL Framebuffer angezeigt werden. Füge Sie dazu Code zum Zeichnen eines texturierten Quads hinzu das den gesamten sichtbaren Bereich des Viewports ausfüllt.
- Schreiben Sie einen OpenGL Fragment Shader (Datei `passthrough.fs`) um die Textur auf dem Framebuffer anzuzeigen. Bestimmen Sie dazu die Farbe des Fragments über einen Lookup aus der 2D Texture (siehe GLSL Befehl `texture2D(sampler2D, vec2)`, builtin Variable `gl_TexCoord`, sowie vorgegebene uniform Variable `sampler0`).
- Implementieren Sie einen OpenGL Fragment Shader (Datei `rgbToGrayscale.fs`) für die Umrechnung von RGB nach Grayscale nach folgender Formel

$$Luminance = 0.299 * r + 0.587 * b + 0.144 * g$$

- f) Implementieren Sie einen OpenGL Fragment Shader (Datei **blur.fs**) zum Blurren des Bildes. Dazu soll das Bild mit einer 3x3 Filtermaske gefaltet werden, die den Mean der 9 Pixel berechnet. Die Texture Koordinaten Offsets für die Maske sind in der Variable tc\_offsets gegeben.
- g) Implementieren Sie einen OpenGL Fragment Shader (Datei **sharpen.fs**) zum Schärfen des Bildes. Dazu soll das Bild mit folgender 3x3 Filtermaske gefaltet werden

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Die Texture Koordinaten Offsets für die Maske sind in der Variable tc\_offsets gegeben.

- h) Implementieren Sie einen OpenGL Fragment Shader (Datei **dilation.fs**) für Dilation in einer 8 Pixelnachbarschaft.
- i) Implementieren Sie einen OpenGL Fragment Shader (Datei **erosion.fs**) für Erosion für eine 8 Pixelnachbarschaft.
- j) Implementieren Sie einen OpenGL Fragment Shader (Datei **laplacian.fs**) für Kantendetektion mittels dem Laplace Operator.
- k) Implementieren Sie einen OpenGL Fragment Shader (Datei **prewitt.fs**) für Kantendetektion mittels der Prewitt Masken für horizontale und vertikale Kanten/Gradienten.

3x3 Maske für horizontale Kanten

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

3x3 Maske für vertikale Kanten

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Die resultierende Farben eines Fragments soll durch die Wurzel der Summe der quadrierten Anteile des Gradienten in horizontaler und vertikaler Richtung bestimmt werden. Die Texture Koordinaten Offsets für die Maske sind in der Variable tc\_offsets gegeben.

- l) Implementieren Sie einen OpenGL Fragment Shader (Datei **sobel.fs**) für Kantendetektion mittels der Sobel Masken für horizontale und vertikale Kanten/Gradienten.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

3x3 Maske für vertikale Kanten

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Die resultierende Farben eines Fragments soll durch die Wurzel der Summe der quadrierten Anteile des Gradienten in horizontaler und vertikaler Richtung bestimmt werden. Die Texture Koordinaten Offsets für die Maske sind in der Variable tc\_offsets gegeben.

## Aufgabe 29 (H) Volume Rendering

In der Vorlesung wurde GPU Raycasting für Volume Rendering besprochen. In dieser Aufgabe werden Sie einen GLSL Fragment Shader für Raycasting auf der GPU schreiben. Sie bekommen zur Bearbeitung der Aufgabe eine bereits lauffähige Applikation mit Umgebung (Datensatz, Transferfunktionen und Startskript). Ihre Aufgabe ist einen GLSL Fragment Shader (Datei raycast.frag) zu implementieren der einen Strahl durch das Volumen castet und dabei das Volume Rendering Integral berechnet. Das Template wurde auf den Rechnern im Praktikumsraum getestet und ist dort einwandfrei startbar. Sollten bei der Bearbeitung der Aufgabe Probleme wegen durch ihre GPU nicht unterstützte Features auftreten, so können Sie die Aufgabe auf jeden Fall im Praktikumsraum lösen.

Bevor Sie mit der Bearbeitung der Aufgabe beginnen, hier noch Hinweise zum Benutzen der Umgebung. Das Zip File **RaycastTemplate.zip** enthält drei Verzeichnisse.

- **bin** Dieser Ordner enthält das Executable, die notwendigen DLLs, die GLSL Shader sowie das Startskript **runRaycaster.bat**.
- **tfn** Der Ordner enthält verschiedene Transferfunktionen. Sie können über eine Änderung des zweiten Parameter für das Programm raycastVRT im Startskript geändert werden.
- **volumeDataSets** Der Ordner enthält einen CT Datensatz eines menschlichen Schädels.

Mit folgenden Tasten können Sie Parameter des Renderers ändern, oder sich anzeigen lassen.

- w - verringere Sampling Rate
- e - erhöhe Sampling Rate
- t - Aktiviere/Deaktiviere Transfer Funktions Display. Achtung, bei aktivierten Transfer Funktion Display kann das Volumen nicht im Raum rotiert oder verschoben werden.
- o - Aktiviere/Deaktiviere Octree Optimierungen für Empty Space Skipping.
- Leertase - Aktiviere/Deaktiviere Display der Zwischenschrittergebnisse. Damit kann überprüft werden ob die Frontfaces/Backfaces und Ray Direction korrekt berechnet wurden.
- m - Aktiviere Animation des Volumes.
- b - Aktiviere/Deaktiviere Drahtgitter Bounding Box Visualisierung
- f - Aktiviere/Deaktiviere Frames per Second Anzeige
- Cropping entlang der Hauptachsen
  - Mit 1,2,3 wählen sie jeweils die 1.,2. oder 3. Hauptachse aus
  - A - bewegt Minimum entlang aktivierter Achse auf Maximum auf der Achse zu
  - a - bewegt Minimum entlang aktivierter Achse weg vom Maximum auf der Achse
  - s - bewegt Maximum entlang aktivierter Achse auf Minimum auf der Achse zu
  - S - bewegt Maximum entlang aktivierter Achse weg vom Minimum auf der Achse

In den folgenden Teilaufgaben werden Sie nun Schritt für Schritt die noch fehlende Funktionalität für Volume Rendering implementieren. Für jede Unterfunktion sind bereits die Rümpfe vorgegeben, der Hauptschader liest bereits die Start Position des Strahls (aus ray\_start) und Richtung (aus ray\_dirTex) aus. Anbei Erläuterungen zu den an den Shader übergebenen Variablen.

- **sampler1D transfer\_function** - Sampler für 1D RGBA Texture für Klassifikation mittels 1D Transferfunktion
- **sampler2D ray\_start** Sampler für 2D RGBA Texture für Ray Startpunkte. Jedes Texel enthält den berechneten Start Punkt in **ObjectSpace!** (nicht **Texture Koordinaten**).
- **sampler2D ray\_dirTex** Sampler für 2D RGBA Texture für Ray Richtungsvektoren. Jedes Texel enthält den berechneten Richtungsvektor (nicht normalisiert, sondern direktes Ergebnis von  $\text{RaySegmentEnd} - \text{RaySegmentStart}$  in `ray_dirTex.rgb`, sowie Länge in `ray_dirTex.a`) in **ObjectSpace!** (nicht **Texture Koordinaten**).
- **sampler3D volume** Sampler für 3D Texture. Enthält die Volumendaten. Zugriffe müssen über korrekte Texture Koordinaten erfolgen.
- **float sample\_distance** Distanz zwischen zwei benachbarten Sample Points auf dem Strahl.
- **int max\_samples** Maximale Anzahl Samples entlang eines Strahls.
- **vec3 texScaleFactors** Vektor für Umrechnung von Koordinaten im Volumen in **ObjectSpace** in gültige 3D Texture Koordinaten.

a) Definieren sie ein Struct Ray das folgende wichtige Information für einen Strahl durch das Volumen kapselt,

- aktuelle Position des Strahls (Texture Koordinaten)
- Richtungsvektor des Strahls (Texture Koordinaten)
- Einen Vektor für das Bewegen des Sample Points von der aktuellen Position auf dem Strahl zum nächsten Sample Point entlang des Strahl (Vektor ist parallel zu Strahlrichtung, Länge entspricht Distanz zwischen zwei benachbarten Sample Points auf dem Strahl) (Texture Koordinaten)
- Ingesamte Länge des Strahls von Eintrittspunkt in das Volumen bis zum Austrittspunkt (Texture Koordinaten)
- Distanz zwischen zwei Sample Points (Texture Koordinaten)
- Bisher zurückgelegte Distanz entlang des Strahls (Texture Koordinaten)

Definieren Sie ein Funktion `initRay`, die folgende Parameter bekommt

- `vec3 rayStart` - Startposition des Strahls im Volume (Object Koordinaten)
- `vec3 rayDir` - Richtungsvektor des Strahls (Object Koordinaten)
- `vec3 scaleFactors` - Skalierungsfaktoren für Umrechnung von Object Koordinate in Texture Koordinate durch Multiplikation
- `float sample_distance` - Abstand zwischen zwei Sample Points

und eine neue korrekt initialisierte Variable vom Typ Ray zurückgibt.

b) Vervollständigen Sie die Implementierung der Funktion **raycast**. In dieser Funktion soll das Volumen Rendering Integral durch Aufsummieren der Sample Werte entlang des Strahls im Volumen berechnet werden. Implementieren Sie hierfür eine For-Schleife die von 0 bis `max_samples` iteriert und dabei den Sample Point entlang des Strahls bewegt. Samplen sie mittels der Funktion **texture3D** das Volumen (3D Texture) an der aktuellen Sample Position auf dem Strahl. Berechnen Sie das Update des Volumen Rendering Integrals innerhalb der Schleife über folgende Formel

$$\begin{aligned} \text{currentSample.rgb*} &= \text{currentSample.a}; \\ \text{sum} &= (1.0 - \text{sum.a}) * \text{currentSample} + \text{sum}; \end{aligned}$$

Beachten Sie bei Ihrer Implementierung das Sie mittels `break` die Schleife verlassen sollten falls die auf dem Strahl zurückgelegte Strecke grösser ist als die zuvor berechnete Länge des Strahlsegments. Als weitere Optimierung können Sie **Early Ray Termination** implementieren indem Sie die Schleife verlassen falls **sum.a > 0.99**.

- c) Vervollständigen Sie die Implementierung der Funktion **raycastTF**. Für eine verbesserte Visualisierung und interaktive Klassifizierung soll nun eine 1D Transferfunktion integriert werden. Die Berechnung des Volume Rendering Integrals ist dafür zu erweitern. Der RGBA Wert des aktuellen Samples wird nun nicht mehr direkt aus dem Volumen genommen sondern über einen Lookup in der Transferfunktionstabelle. Fügen Sie dazu Code ein um zuerst das Volumen an der aktuellen Position zu sampeln und dann die RGBA Werte für **currentSample** über eine Lookup in der 1D Texture **transfer\_func** nach der Formel zu bestimmen.

$$\begin{aligned} \text{volSample} &= \dots // \text{Sample Volumen an aktueller Position} \\ \text{currentSample} &= \text{texture1D}(\text{transfer\_func}, \text{volSample.a}) \end{aligned}$$

(Anmerkung: Da für die Aufgabe mit einem einkanaligen Volumen gearbeitet wird, haben die RGBA Kanäle beim Sampeln der 3D Texture alle denselben Wert. Deshalb die eindimensionale Transferfunktion.)