Übung 09: Raytracing in Vulkan

Ansprechpartner: Joseph Heetel auf Fakultäts-Discord; joseph.a.heetel@stud.hs-kempten.de

0 Beispiele installieren

Klonen Sie das Git Repository von https://github.com/Joseph-Heetel/VulkanRT-Educational:

```
git clone https://github.com/Joseph-Heetel/VulkanRT-Educational --recursive
```

Für Raytracing in Vulkan wird als Minimum die Vulkan SDK Version 1.2 benötigt. Stellen Sie sicher das Sie ein entsprechendes SDK installiert haben: https://www.lunarg.com/vulkan-sdk/
Die installierte Version kann nach Öffnen des Programms „Vulkan Configurator“ im Menüpunkt Vulkan Info überprüft werden.

Das Repository ist ein CMake Projekt, also kann dies nach Belieben direkt mit Visual Studio bearbeitet, direkt kompiliert oder zu anderen Projekttypen konvertiert werden.

Auf Windows sollte damit das Projekt schon laufen, fürs Installieren auf Linux werden noch weitere Dependencies benötigt, siehe https://github.com/SaschaWillems/Vulkan/blob/master/BUILD.md.

Die folgenden Aufgaben werden in den Ordnern „vulkanrt-exercise1“ und „vulkanrt-exercise2“ bearbeitet. „vulkanrt-reference“ ist eine beinahe unveränderte Version des Beispielprojektes „raytracing-reflections“ von Sascha Willems.

Das Projekt ist ein Klon von einer Vulkan-Beispiel-Bibliothek, die von Sascha Willems erstellt und gepflegt werden. Das ursprüngliche Repository befindet sich hier: https://github.com/SaschaWillems/Vulkan

Jeder dieser Ordner wird von CMake mit einem Batch- (auf Linux Shell-) Skript bestückt, welches alle Shader im Shader-Unterverzeichnis mithilfe des „glslc.exe“ Tools in Ihrem lokalen Vulkan SDK kompiliert.

Aufgaben 1, 2 und 3 sind getrennt voneinander bearbeitbar.

0.1 Troubleshooting

Build tools for v142 not found … retarget solution: Hier im Visual Studio installer das Toolset nachinstallieren.

1 Vertraut machen mit Funktionsweise von Raytracing-Shadern

Schauen Sie sich im Projektordner das Projekt „vulkanrt-reference“ genauer an. Dieses Programm zeichnet mithilfe von Vulkan Raytracing Reflektionen. Starten Sie das Programm. Öffnen Sie die Shader-Quelldateien und erschließen Sie sich die Funktionsweise. Ein „.rgen“-Shader wird zuerst aufgerufen, und erzeugt die rays. Ein „.rchit“-Shader wird für den ersten Intersect eines Rays mit Geometrie aufgerufen. Ein „.rmiss“-Shader wird aufgerufen, wenn ein Ray keine Intersects gefunden hat. Suchen Sie online nach ihnen unbekannten Keywords welche Sie im Shader finden.

Nun beantworten Sie folgende Fragen (Ein Satz/Stichpunkt pro Frage):

- Was ist die Funktion eines RayPayloads?
- Wie können Sie die Position des Schnittpunktes in Weltkoordinaten innerhalb eines „.rchit“-Shaders berechnen?
- Wie ist in dieser Implementierung die für Raytracing charakteristische Rekursion realisiert?
- Wo könnten Sie in dieser Implementierung die Farben der Skybox anpassen?

2 Schatten in Vulkan-RT

Machen Sie sich mit dem Einbinden von Shaderprogrammen, bzw. der Shader Binding Table vertraut. Hilfreiche Website von Nvidia zu dem Thema: https://nvpro-samples.github.io/vk_raytracing_tutorial_KHR/#shaderbindingtable.

Folgend wird Shader Binding Table zu „SBT“ abgekürzt.

Öffnen Sie das Projektverzeichnis „vulkanrt-exercise1“. Hier muss lediglich in den Dateien „exercise1.cpp“ und im Shader-Unterverzeichnis in „closesthit.rchit“ und „shadow.rmiss“ gearbeitet werden.

Der Ablauf, um ein neues Schatten Shaderprogramm einzubauen, kann hier: https://nvpro-samples.github.io/vk_raytracing_tutorial_KHR/#shadows gut beobachtet werden. Jedoch unterscheiden sich die verwendeten Helper-Klassen und Methoden (die Grundstruktur) zu Sascha Willems fundamental. Folgende Schritte werden Sie dabei unterstützen den Code in das Beispielprojekt zu transferieren:

Suchen Sie nach dem ersten TODO in der C++ Quelldatei. Da nun ein zweiter Miss-Shader eingebaut werden soll (Siehe ASCII-Zeichnung vom SBT) muss ein zusätzlicher Handle und entsprechend Speicher reserviert werden.

Das zweite TODO in der gleichen Datei markiert die Stelle, an der Sie den entsprechenden Shader laden und in die Liste der Shaderstages und Shadergroups hinzufügen sollen.

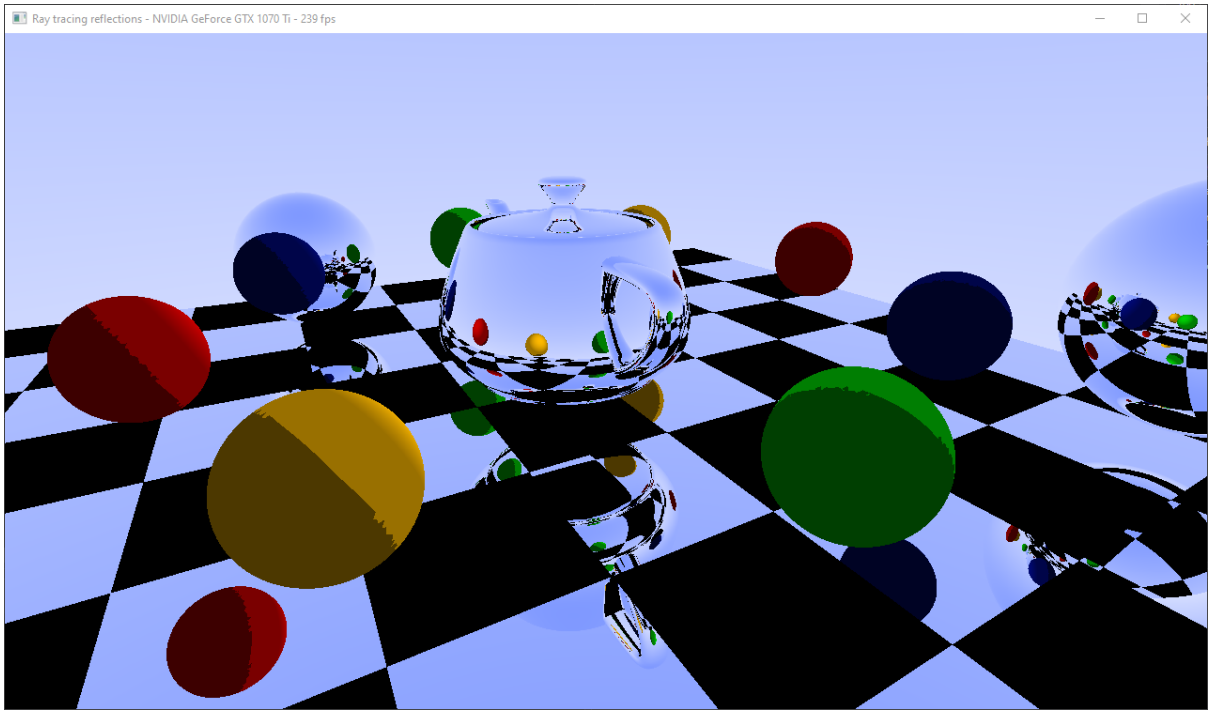
Damit ist der Teil im Quellcode fertig. Falls flackernde eckige Artefakte beim Ausführen des Programms erscheinen ist was schief gelaufen. Ohne Anpassen des Shadercodes sollte das Programm noch ausführbar sein und problemlos zeichnen. Eventuell helfen Errormessages der Vulkan Validation layer aus der mit dem Programm startenden Konsole weiter.

Anschließend muss der zusätzliche Shader „shadow.rmiss“ befüllt werden. Wie das aussieht verrät die oben verlinkte Nvidia Website (dieser Teil ist extrem simpel, nicht wundern).

Zuletzt muss der „closesthit.rchit“ shader angepasst werden. Hier soll an der markierten Stelle ein zusätzlicher raycast erfolgen, um die Sichtbarkeit der Lichtquelle festzustellen. Wie dies funktioniert verrät auch die bereits bekannte Website.

Dran denken mit der Batch/Shell Datei die Shader zu kompilieren, bevor das Projekt startet!

Folgendermaßen soll das ganze schließlich aussehen:



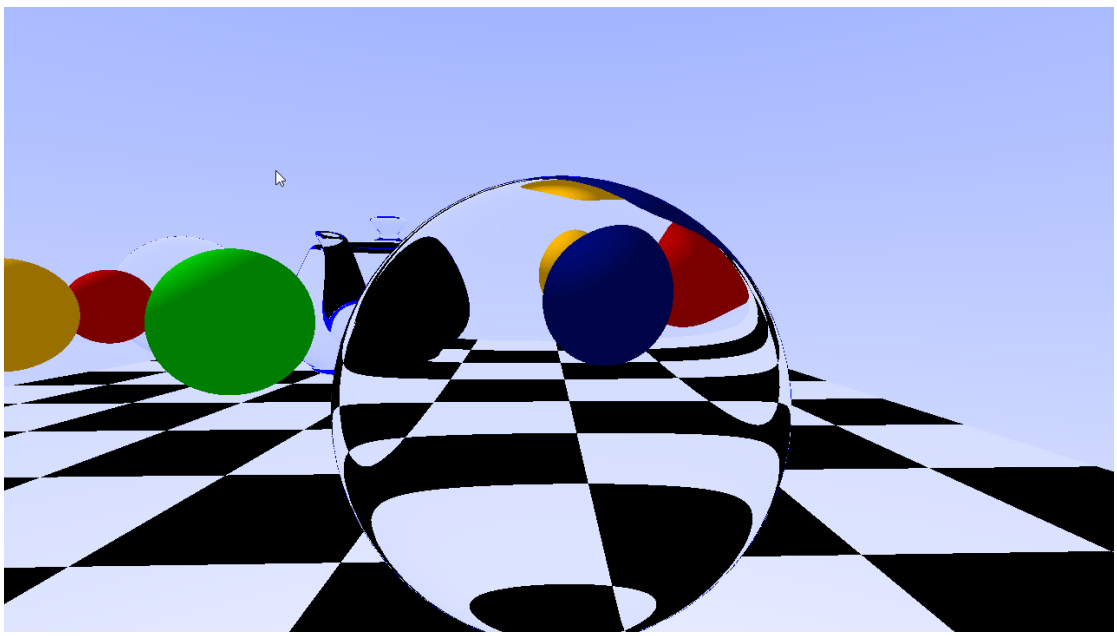
2.1 Schatten - Artefakte

Eventuell können Sie Artefakte erkennen. Erklären Sie das Zustandekommen und Beschreiben Sie grob mögliche Vorgehensweisen, um diese zu vermeiden.

3 Glass Effekt in Vulkan-RT

Bearbeiten Sie das Projekt „vulkanrt-exercise2“ Hier müssen nur Änderungen im Shader „raygen.rgen“ gemacht werden. Hier ist die Rekursion mit einer for-Schleife gelöst. Sie müssen also an der entsprechend markierten Stelle Ray Origin und Ray Direction korrekt setzen, um Lichtbrechung zu erhalten. Der Ablauf, um ein Glas-Effekt einzubauen, kann hier: https://iorange.github.io/p02/TeapotAndBunny.html#7-ice gut beobachtet werden.

Probieren Sie verschiedene Brechungs-Indizes aus.



3.1 Schwarze Kanne?

Überlegen Sie sich, wieso die Kanne schwarz ist, wenn man durch die Glas-Kugeln schaut. Erarbeiten Sie Lösungsvorschläge.