Hochschule Bremen
Fakultät 4 – Elektrotechnik und Informatik
Internationaler Studiengang Medieninformatik (B.Sc.)

Exposé

Anwendung von Hardware-Raytracing zur Optimierung der Treffererkennung

– Eine Untersuchung zur Leistungssteigerung und Präzisionsverbesserung in Echtzeit-Anwendungen –

Kevin Kügler (*399027*)

4. Juni 2024 (Version 1.00)

1 Einleitung

In der Welt der Videospiele und erweiterten Realitätsanwendungen (XR) stellen die steigende Komplexität und die Anforderungen an die Treffererkennung eine Herausforderung dar. Traditionelle CPU-basierte Methoden stoßen zunehmend an ihre Grenzen, was die Systemperformance und die Spielerfahrung beeinträchtigt. Diese Arbeit erforscht die Möglichkeit, die Treffererkennung durch den Einsatz von Raytracing-Kernen auf GPUs zu realisieren, um die Präzision zu steigern und die CPU-Last zu verringern.

2 Problemstellung und Lösungsansatz

2.1 Problemstellung

Die Problematik der Treffererkennung in interaktiven Echtzeit-Umgebungen verschärft sich mit der zunehmenden Komplexität von Spielwelten und der fortlaufenden Entwicklung realistischerer 3D-Modelle. Die CPU ist in einer modernen Spiel-Engine oft mit mehreren Aufgaben belastet, einschließlich KI-Steuerung, Spiellogik und der Verwaltung von Benutzerinputs, wodurch die Ressourcen für die Treffererkennung limitiert sind.

Die traditionelle Hit-Scan-Methode, die auf der CPU ausgeführt wird, stößt bei der Skalierung auf Grenzen. Typischerweise werden Hitboxes in Form von Quadern, Sphären und Kapseln zur Vereinfachung des 3D-Modells verwendet. Der Strahl von seinem Startpunkt in die gegebene Richtung projiziert. Trifft der Strahl auf eine Hitbox, wird die Berechnung beendet und Trefferinformationen zurückgegeben. Alternativ kann das 3D-Modell selber als Hitbox fungieren, allerdings mit deutlich erhöhtem Rechenaufwand.

Dies führt zu einem Dilemma: Die Erhöhung der Präzision der Treffererkennung verlangt mehr Rechenzeit, was wiederum die Bildwiederholrate und damit die Immersion und Interaktion beeinträchtigen kann.

2.2 Lösungsansatz

Im Rahmen der Arbeit wird das Hit-Scan-Verfahren auf die GPU verlagert, um die Rechenleistung der spezialisierten Raytracing-Kerne moderner Grafikkarten zu nutzen. Diese Anpassung zielt darauf ab, die Geschwindigkeit und Präzision der Treffererkennung zu steigern, während gleichzeitig die Belastung der CPU reduziert wird.

Die Verlagerung des GPU-basierten Hit-Scan-Verfahrens soll mit der Vulkan-API, speziell Vulkan Ray Queries, realisiert werden. Ray Queries ermöglichen den Einsatz der Raytracing-Kerne außerhalb von Rendering-Shadern. Sie können für beliebige Zwecke benutzt werden. So auch für Treffererkennung.[1]

Mehrere Testszenen mit variierender Anzahl von statischen 3D Modellen, bewegend und stehend, sollen der Evaluierung dienen: Durch Vergleichen der Performance des Verfahrens auf der GPU mit dem bestehenden CPU-basierten Ansatz in Unreal Engine 5[2] wird überprüft, ob die GPU-Implementierung eine schnellere Verarbeitung ermöglicht. Dieser Vergleich soll die Vorteile der GPU-Nutzung für Echtzeit-Treffererkennung in interaktiven Anwendungen verdeutlichen. Der GPU-basierte Ansatz wird in einer zuvor eigens entwickelten Rendering-Engine verwirklicht.

[1] https://www.khronos.org/blog/ray-tracing-in-vulkan/#blog_Ray_Queries
[2] https://www.unrealengine.com/en-US/unreal-engine-5

3 Literaturübersicht

Diese Übersicht an Literatur beschreibt Grundlagen und aktuellen Stand zu Raytracing, Echtzeit-Kollisionsdetektion, Vulkan sowie die Verwendung von Vulkan Ray Queries und Compute Shader.

- **RTX on—The NVIDIA Turing GPU**[1] John Burgess beschreibt die Architektur der NVIDIA Turing GPU und ihre neuen Features wie RT Cores und Tensor Cores, die speziell für Raytracing und Deep Learning entwickelt wurden. Der Artikel beleuchtet die technischen Details und die Leistungsfähigkeit dieser GPU-Generation.
- **Mastering Graphics Programming with Vulkan**[2] In diesem Buch von Marco Castorina wird die Programmierung von Grafikanwendungen mit Vulkan von Grund auf erläutert. Es behandelt zudem Ray Queries.
- **Introduction to Compute Shaders**[3] Matthäus G. Chajdas bietet in diesem Artikel eine Einführung in Compute Shader. Er erklärt die Grundlagen, Anwendungsgebiete und die Implementierung von Compute Shadern.
- **Real-Time Collision Detection**[4] Christer Ericsons Buch ist ein umfassendes Werk zur Echtzeit-Kollisionserkennung. Es bietet theoretische Grundlagen und praktische Implementierungstechniken für effiziente Kollisionserkennung in Spielen und Simulationen.
- **NVIDIA TURING GPU ARCHITECTURE**[5] Dieses Whitepaper beschreibt die Turing GPU-Architektur, die erstmals Echtzeit-Raytracing in Hardware ermöglicht. Es werden die neuen RT Cores und Tensor Cores sowie ihre Anwendungen in Grafik und Deep Learning detailliert erläutert.
- **Ray Tracing Gems II**[6] Shirley et al. kompilierten eine Sammlung von Artikeln zur nächsten Generation des Raytracings mit DXR, Vulkan und OptiX. Behandelt werden fortschrittliche Techniken und Optimierungen für hochqualitatives und Echtzeit-Rendering sowie Grundlagen zu Raytracing
- **vk_mini_path_tracer**[7] Neil Bickford beschreibt in seinem Tutorial detailliert eine Implementation eines Path-Tracers mithilfe von Vulkan Ray Queries.

4 Tools

Hardware:

- CPU - AMD Ryzen 9 7900X
- GPU - Nvidia RTX 4070

Implementationstools:

- Microsoft Visual Studio 2022 - C++ Entwicklungsumgebung
  - Microsoft Visual C++ Compiler
  - C++ 20 Standard
- NVIDIA Nsight Graphics - GPU-Debugging
- Unreal Engine 5 - Game-Engine

Software-Bibliotheken:

- Eigens erstelle Vulkan Rendering-Engine

Versionsverwaltung:

- Git - Versionskontrollsoftware
- Fork - Grafische Benutzeroberfläche für Git

5 Vorläufige Gliederung

Nachfolgend die vorläufige Gliederung der Thesis. Es gilt zu beachten, dass vor allem die prototypische Realisierung sich radikal von der Planung unterscheiden kann.

Eigenständigkeitserklärung

Abstract

1. Einleitung
   - 1.1. Problemfeld
   - 1.2. Ziele der Arbeit
   - 1.3. Lösungsansatz
   - 1.4. Aufbau der Arbeit
2. Grundlagen
   - 2.1. 3D-Meshes
     - 2.1.1. Static Meshes
     - 2.1.2. Skeletal Meshes
   - 2.2. Kollisionserkennung/Treffererkennung
     - 2.2.1. Bounding-Boxes
     - 2.2.2. Treffererkennungs-Algorithmus
   - 2.3. Raytracing
     - 2.3.1. Hit-Scan Verfahren
   - 2.4. Hardware-Raytracing
   - 2.5. Bounding Volume Hierarchies
   - 2.6. GPU-Architektur
     - 2.6.1. Scheduler
     - 2.6.2. Compute Units
     - 2.6.3. Pipeline
   - 2.7. GPU<->CPU Kommunikation
     - 2.7.1. PCI-Express
   - 2.8. Render-Hardware-Interface
     - 2.8.1. Vulkan
       - 2.8.1.1. Pipeline
       - 2.8.1.2. Queue

6 Zeitplanung

Geplanter Starttermin: 25. Juni 2024

Bearbeitungsdauer: 9 Wochen

Tabelle 1 stellt die geplanten Arbeitspakete und Meilensteine dar:

Tabelle 1: Arbeitspakete und Meilensteine

M1	Offizieller Beginn der Arbeit	25.06.2024
	• Verfassen von Kapitel 1 (Einleitung)	1 Woche
M2	Recherche & Grundlagen	02.07.2024
	• Recherche (Hardware-)Raytracing • Recherche Treffererkennung • Recherche Vulkan & Compute Shader • Verfassen von Kapitel 2 (Grundlagen) • Verfassen von Kapitel 3 (Verwandte Arbeiten)	2 Wochen
M3	Konzeption	16.07.2024
	• Entwurf des Renderers und Raytracing-Shaders • Entwurf der Testszenen • Verfassen von Kapitel 4 (Konzeption)	2 Wochen
M4	Implementation Prototyp	30.07.2024
	• IDE Einrichten • C++ Implementation • Profiling • Testszenenaufbau und Profiling • Verfassen von Kapitel 5 (Prototypische Realisierung)	2 Wochen
M5	Evaluation	13.08.2024
	• Verfassen von Kapitel 6 (Evaluation) • Verfassen von Kapitel 7 (Zusammenfassung & Ausblick)	1 Woche
M6	Erste Fassung vollständige Thesis	20.08.2024
	• Korrekturlesen • Drucken & binden lassen	1 Woche
M7	Abgabe der Thesis	27.08.2024

Literatur

Gedruckte Quellen

[1] John Burgess. „RTX on—The NVIDIA Turing GPU“. In: *IEEE Micro* 40.2 (2020), S. 36–44. DOI: 10.1109/MM.2020.2971677.

[2] Marco Castorina. *Mastering Graphics Programming with Vulkan. Develop a modern rendering engine from first principles to state-of-the-art techniques*. Hrsg. von Gabriel Sassone. 1. Birmingham: Packt Publishing, 2023. 1382 S. ISBN: 978-1-80324-479-2.

[4] Christer Ericson. *Real-Time Collision Detection*. Taylor & Francis Group, 2004, S. 632. ISBN: 9781000750553.

[5] NVIDIA Corporation. *NVIDIA TURING GPU ARCHITECTURE*. Sep. 2018. URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf (besucht am 27. 05. 2024).

[6] Adam Marrs. *Ray Tracing Gems II. Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Hrsg. von Peter Shirley und Ingo Wald. [Erscheinungsort nicht ermittelbar]: Springer Nature, 2021. 1858 S. ISBN: 978-1-4842-7184-1.

Online-Quellen

[3] Matthäus G. Chajdas. *Introduction to compute shaders*. Juli 2018. URL: https://anteru.net/blog/2018/intro-to-compute-shaders/ (besucht am 27. 05. 2024).

[7] Neil Bickford. URL: https://nvpro-samples.github.io/vk_mini_path_tracer/ (besucht am 27. 05. 2024).