OpenGL-basiertes Software Occlusion Culling zur Beschleunigung des 3D-Renderings großer Datenmengen und komplexer Szenen

Projekt-Inf 3D-Rendering

Dominik Sellenthin Christian Stegmaier Gariharan Kanthasamy

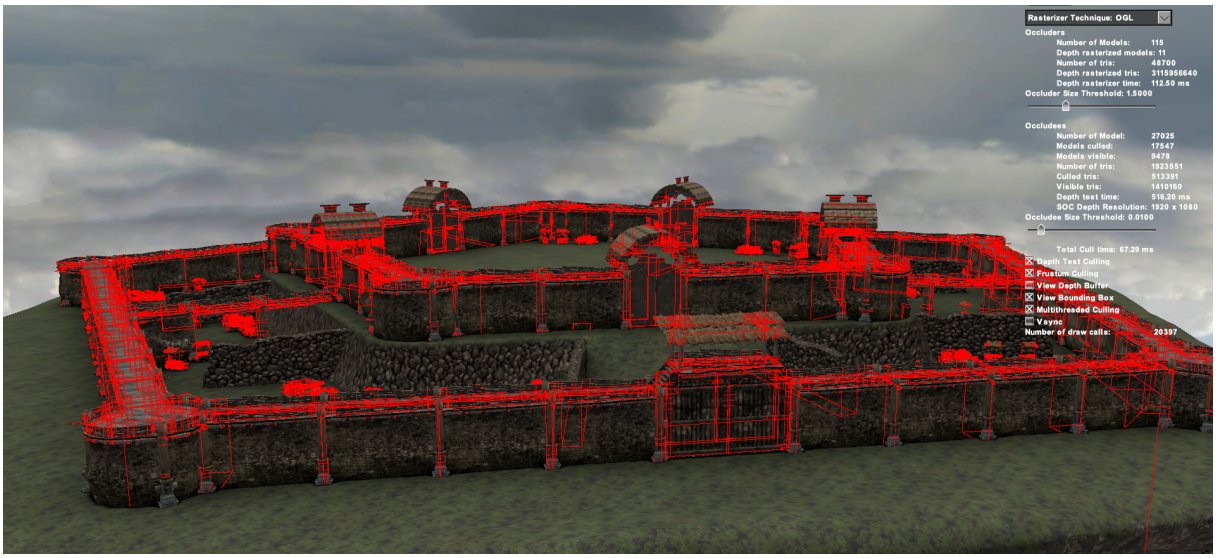

Abb. 1: Testszene.

Kurzbeschreibung—In Bereichen des 3D-Rendering, sei es in der wissenschaftlichen Visualisierung oder in Computerspielen, ist die Rechenleistung der Grafikkarte schnell an ihrer Grenze, während die CPU kaum in Anspruch genommen wird. Damit die GPU entlastet wird, wird ein Software-Rasterisierer eingesetzt, so dass ein Teil der Berechnungen in einem Vorverarbeitungsschritt auf die CPU ausgelagert wird. Um die vollständige Rechenleistung moderner Multi-Core CPUs zu gewährleisten, wird in dieser Arbeit ein bereits vorhandener Rasterierer, Mesa 3D, verwendet, der zusätzlich die OpenGL API zur Verfügung stellt. Ziel des Software-Rasterisierers ist es, komplett parallel zum GPU-Rendering des aktuellen Frames, in zwei Schritten festzustellen, welche Objekte in der kommenden Szene zu sehen sind. Im Wesentlichen gilt es dabei zuerst einen geeigneten Tiefenpuffer zu generieren und anschließend mittels Occlusion Queries alle Objekte zu bestimmen, die noch sichtbar sind (auch Z-Buffering genannt). Das Ergebnis der Occlusion Queries kann nun ohne nennenswerte Latenz verwendet werden, um der GPU mitzuteilen, welche Objekte gerendert werden sollen, so dass unnötiger Rechenaufwand der GPU vermieden wird (160).

1 Einleitung

Egal, ob im Bereich der wissenschaftlichen Visualisierung oder in anderen 3D-Rendering-Bereichen

2 Related Work

Das Paper „Masked Software Occlusion Culling “ von Hasselgren et al. implementiert einen neuen Algorithmus zur Berechnung de hierarchischen Z-Buffers in das Intel Occlusion Culling Framework. Da heutige Anwendungen immer höhere Ansprüche an die Leistung der Engines stellen wechseln diese von bisherigen Verfahren, wie dem vorherigen berechnen von Mengen die potentiell sichtbar sind, zu Algorithmen die Occluder in einen hierarchischen Z-Buffer rasterisieren. Der Z-Buffer wird hierbei mit einem Software Rasterisierer berechnet. Anstatt wie bisher alle Drawcalls an die GPU zu schicken kann nun mit dem Z-Buffer auf der CPU berechnet werden ob ein Occludee von einem Occluder verdeckt wird. Dadurch kann die Anzahl der Drawcalls an die GPU, unter Verwendung der CPU, verringert werden. In ihrem Paper stellen die Autoren einen Algorithmus vor, der effizient den hierarchischen Z-Buffer berechnen kann und somit die Gesamtperformance entscheidend verbessert. Anstatt wie in früheren arbeiten Pixel als kleinste Einheit zu betrachten benutzen die Autoren sogenannte Kacheln. Da AVX2 (Advanced Vector Extensions) es ermöglicht 8 SIMD Instruktionen mit 32-Bit Präzision auszuführen wurde für die Kacheln eine Größe von 32x8 Pixeln gewählt. Indem Bitmasken von rechts und links in die Kacheln geschoben werden wird am Ende eine Abdeckungsmaske erhalten, die angibt welche Pixel in einer Kachel verdeckt werden. Während ihr Algorithmus keine 100% Präzision garantiert, False Positives sind möglich, bewegt sich der Fehler in gleicher Größenordnugn wie bei bisherigen Algorithmen. Ein wichtiger Faktor für die Performance ihres Algorithmus ist die Reihenfolge in der die Objekte gerendert werden, die Objekte werden von vorne nach hinten gerendert. Durch diese Reihenfolge werden die wichtigsten Occluder als erstes dargestellt und für den Fall, dass die Occlusion Culling Time begrenzt ist, wird trotzdem ein nahezu optimales Ergebnis erzielt.

Als Basiswert für die Performance des Masked Occlusion Algorithmus (MOC) wurde ein normales Rendering ohne Occlusion Culling, aber mit Frustum Culling verwendet. Frustum Culling rendert nur Objekte die sich im Sichtbereich der Kamera befinden und kann somit je nach Kameraposition einen Teil der Objekte cullen. Als alternativer Algorithmus wird der „Hierarchical Z Buffer algorithm “ (HiZ) evaluiert. Alles Messungen wurden mit voller Auflösung (1920*1080

Institut für Visualisierung und Interaktive Systeme, Universität Stuttgart. Wintersemester 2018/2019.

Pixel) ausgeführt, außerdem wurde nur die Single-Core Performance betrachtet. Der MOC Algorithmus ist im Vergleich zum HiZ Algorithmus etwas vorsichtiger und cullt 2% weniger Dreiecke, erreicht allerdings trotzdem eine bessere Performance. Bei einem ersten Test mit einer kleinen Kamerafahrt im Intel Occlusion Culling Framework wurde die Framezeit, also die Zeit für die Berechnung eines Frames gemessen. Szene enthält 49k Dreieckige Occluder meshes. Während die Performance des Frustum Cullings sehr konstant bleibt, erkennt man bei den beiden Occlusion Culling Algorithmen größere Schwankungen, da ein großer Occluder im Vordergrund potenziell alle Occludees hinter ihm überdecken kann und damit die Berechnung stark vereinfacht. Mit den Occlusion Culling Algorithmen kann eine 1,5-7x schnellere Total Frame Time erreicht werden. Sind die Berechnungen sehr einfach und es können sehr viele Objekte gecullt werden, ist die Performance von HiZ und MOC sehr gut und fast identisch. Je komplexer die Berechnungen sind und je genauer alle Objekte auf potenzielles culling überprüft werden müssen, desto besser schneidet der MOC Algorithmus ab. In einzelnen Frames erreicht er so teilweise eine etwas mehr als doppelt so schneller Total Frame Time. In einem zweiten Test wurde eine wesentlich komplexere Szene mit 143k Occluder meshes für die Messung der Daten benutzt. Die Occlusion Culling Time des MOC Algorithmus ist ca. 10x so schnell wie die des HiZ Algorithmus. Die Autoren halten fest, dass eine 10-fache Beschleunigung durch ihren Algorithmus vermutlich nicht immer angenommen werden kann, ihr Algorithmus jedoch sehr robust gegenüber komplexen Occluder Strukturen ist. Um die Skalierbarkeit des MOC Algorithmus zu testen wurden 32k zufällige Occluder Dreiecke mit unterschiedlicher Größe erzeugt. Während die Performance bei einer Größe der Dreiecke von 10x10 praktisch identisch ist, gewinnt der MOC Algorithmus im Vergleich zum HiZ Algorithmus mit steigende Größe der Dreiecke immer mehr an Abstand. Als Endergebnis halten sie fest, dass ihr Algorithmus 3x schneller ist als die bisherigen Algorithmen und gleichzeitig nur einen geringen Memory Overhead hat. Mit ihrem Algorithmus können 98% aller Dreiecke gecullt werden.

Intel Paper [1]

SSE, AVX und maskedAVX

3 Software Occlusion Culling

Die Menge der Objekte, die es zu Rendern gilt, wird in zwei Mengen aufgeteilt. Zum einen gibt es die Occluder. Occluder sind eine Menge von Objekten, die groß genug sind, dass es wahrscheinlich ist, dass sie andere Objekte verdecken. Zum anderen gibt es Occludees. Occludees sind all diejenigen Objekte die potentiell von Occludern verdeckt werden (das heißt, sie beinhalten ebenfalls alle Occluder). Sowohl Occluder als auch Occludees liegen dabei in zwei Formen vor. Einmal als Mesh, das zur genauen Darstellung des Objekts in der gerenderten Szene dient und einmal in Form einer Axis Aligned Bounding Box (AABB), die sowohl zum Frustumculling als auch zum (Tiefen-)Rasterisieren verwendet wird. AABBs eignen sich wegen ihrer einfachen geometrischen Form sehr gut, um erste (grobe) Tests durchzufÃ¼hren, ob ein Objekt Ã¼berhaupt von der Kamera gesehen werden kann (Frustumculling) und dementsprechend fÃ¼r die folgende Rasterisierung beim Culling in Frage kommt.

FrustumCulling bevor jedem Schritt

Occlusion Culling besteht im Wesentlichen aus zwei Schritten. Als erstes wird der Tiefenpuffer auf Basis einer Occludermenge beschrieben. Occluder sind hierbei alle Objekte, (Occludees). Diese Occluder werden in einem ersten Renderingdurchlauf, allerdings ohne die Objekte zu zeichnen, rasterisiert und der Tiefenpuffer wird entsprechend der Occludermenge beschrieben.

Schritt zwei besteht darin Occlusion Queries durchzufÃ¼hren. Bei den Occlusion Queries werden die Bounding Boxes aller Occludees gegen den im vorherigen Schritt erstellten Tiefenpuffer getestet und es wird geprÃ¼ft, ob die Occludees den Tiefentest bestehen oder nicht, sprich, ob die Occludees von einem Occluder verdeckt werden oder sichtbar sind.

OcclusionQuery

4 Ergebnisse

Das ist ein Testsatz für Ergebnisse.

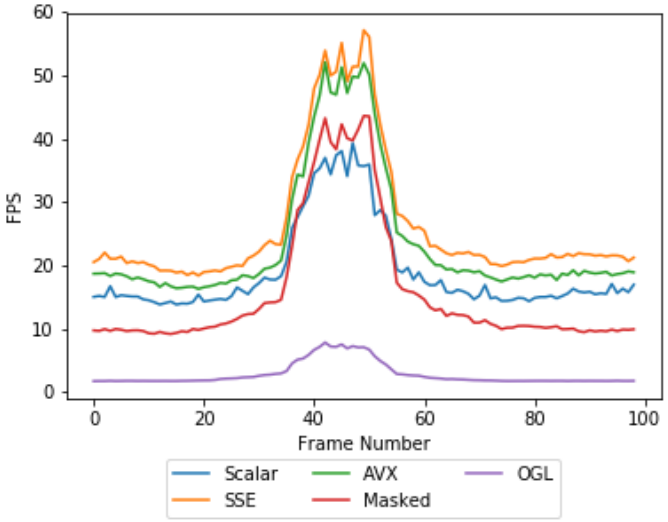

Abb. 2: Frame Number PNG

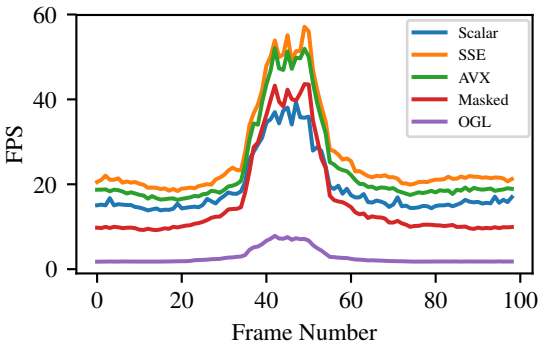

Abb. 3: Frame Number PGF

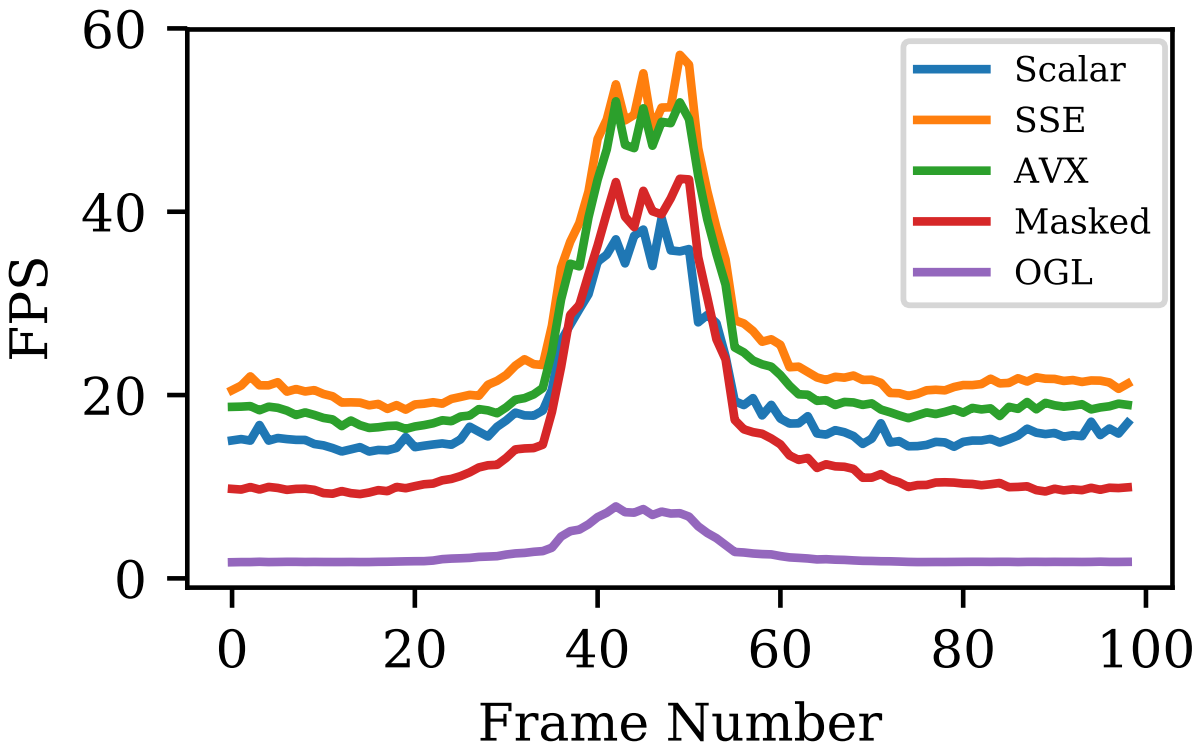

Abb. 4: Frame Number PDF

-

5 Fazit

Das ist ein Testsatz für das Fazit.

Literatur

[1] J. Hasselgren, M. Andersson, and T. Akenine-Möller. Masked Software Occlusion Culling. In U. Assarsson and W. Hunt, editors, *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2016.