Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelor- /Projektarbeit (Studiengang, max. 2 Zeilen)

Titel der Arbeit Titel der Arbeit Titel der Arbeit Titel der Arbeit Titel der Arbeit Titel der Arbeit Titel der Arbeit Titel der Arbeit Titel der Arbeit Titel der (max. 4 Zeilen)

Autoren	Vorname Name Vorname Name
Hauptbetreuung	Vorname Name Vorname Name
Nebenbetreuung	Vorname Name Vorname Name
Industriepartner	Firmenname
Externe Betreuung	Vorname Name Vorname Name
Datum	01.01.2012

Bitte füllen Sie das Titelblatt aus und berücksichtigen Sie Folgendes:

- ➔ Bitte auf keinen Fall Schriftart und Schriftgrösse ändern. Text soll lediglich überschrieben werden!
- ➔ Bitte pro Tabellenzeile max. 4 Textzeilen!
- Vorlage: Haben Sie die richtige Vorlage gewählt? → Logo Institut/Zentrum
- Titel: Fügen Sie Ihren Studiengang direkt nach dem Wort „Bachelorarbeit“ ein (max. 2 Zeilen).
- Titel der Arbeit: Überschreiben Sie den Lauftext mit dem Titel Ihrer Arbeit (max. 4 Zeilen).
- Autoren: Tragen Sie Ihre Vor- und Nachnamen ein (bitte alphabetisch nach Name).
- Betreuer: Tragen Sie Ihren Betreuer / Ihre Betreuer ein (bitte alphabetisch nach Name).
- Nebenbetreuung: Falls Sie keine Nebenbetreuung haben → bitte ganze Tabellenzeile löschen.
- Industriepartner: Falls Sie keinen Industriepartner haben → bitte ganze Tabellenzeile löschen.
- Externe Betreuung: Falls Sie keine ext. Betreuung haben → bitte ganze Tabellenzeile löschen.
- Datum: Bitte aktuelles Datum eintragen.
- Schluss: Am Schluss löschen Sie bitte den ganzen Beschrieb (grau) und speichern das Dokument als pdf. ab.

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

……

Unterschriften:

……...

……

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Vorwort

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Inhaltsverzeichnis

1. Einleitung

1.1. Ausgangslage

3D-Applikation sind immer häufiger im Einsatz und werden auf diversen Geräten verwendet. Da diese Applikationen sehr rechenintensiv sind und gerade mobile Geräte in ihrer Rechenleistung beschränkt sind, ist Performance Optimierung im 3D-Rendering unabdinglich. Die Komplexität der Modelle hat dabei einen signifikanten Einfluss auf die Leistung. Eine Möglichkeit zur Optimierung ist das Generieren und Verwenden von vereinfachten Modellen. In diversen Rendering Engines [1] gibt es deshalb Möglichkeiten für das Verwenden von Level Of Details (LOD) Artefakten. Dabei werden abhängig von Parametern vereinfachte Varianten desselben Modelles verwendet. So kann z.B. ein Objekt in grosser Distanz vereinfacht dargestellt werden, ohne merkbare Auswirkungen auf die Qualität zu haben. Für Game Engines [2] wie Unreal oder Unity gibt es Möglichkeiten, um den Einsatz von LOD Artefakten zu vereinfachen. Im Web Bereich gibt es zur Zeit keine weit verbreitete Möglichkeit.

1.2. Zielsetzung

Ziel der Arbeit ist es, ein Tool zu entwickeln, das den Umgang mit LOD Artefakten im Web vereinfacht.

[1] Teilprogramm, das zuständig für die Darstellung von Grafiken ist
[2] Framework, das für den Spielverlauf und dessen Darstellung verantwortlich ist.

2. Grundlagen

2.1. Das Web

Im folgenden wird der Begriff "Das Web"vereinfacht für die Plattform von verschiedenen Web Technologien genutzt. Dies beinhaltet insbesondere vom W3C veröffentlichte Spezifikationen.

2.2. 3D-Rendering im Web

3D-Visualisierungen werden in vielen Branchen verwendet. So sind zum Beispiel im medizinischen Bereich 3D Scans hilfreich bei der Analyse von Verletzungen. Virtual Reality, CAD Anwendungen oder Computerspiele sind weitere Anwendunsgebiete, welche 3D Visualisierungen einsetzen. In vielen weiteren Bereichen sind dank leistungsstärkeren Geräten auch realere und somit komplexere Visualisierungen möglich.
Viele Anwendungen sind auf spezifische Hardware wie zum Beispiel die Spielkonsolen PlayStation oder Xbox angewiesen. Dies hat den Nachteil, dass die Verteilung der Software schwieriger ist, da spezifische Hardware notwendig ist. Selbstverständlich ist dies für gewisse Anwendungsgebiete kein Problem.
Für hardwareunabhängige Anwendungen eignet sich das Web hervorragend. Viele Benutzer haben Zugang zu einem Desktop, Tablet oder Mobiltelefon. Somit ermöglicht das Web Anwendungen mit weniger Aufwand einer grösseren Masse zugänglich zu machen. Heutzutage ist es auch möglich, komplexe 3D Visualisierungen im Web zu realisieren. Als Basis dafür dient meist das von der Khronos Group entwickelte WebGL, das von allen modernen Browsern unterstützt wird. WebGL ist eine low-level JavaScript API für 3D Visualisierungen. [1] Alternativ zu WebGL wird zurzeit ein weiterer Standard entwickelt: WebGPU. Dieser ist zur Zeit des Schreibens noch in Entwicklung und wird deshalb nicht weiter berücksichtigt, auch wenn ein grosses Potenzial vorhanden ist. [2]
Die Unabhängigkeit der Hardware bedeutet jedoch auch, dass Optimierung der Performanz in Webanwendungen unabdinglich ist, um allen Benutzern ein optimales Erlebnis zu ermöglichen. Im Vergleich zu fixen Hardware Anwendungen ist es realistisch, dass eine Web Anwendung sowohl auf einem leistungsfähigen Desktop Computer als auch auf einem günstigen Mobilgerät verwendet wird.
Zudem ist WebGL eine junge Technologie und wurde erst 2011 veröffentlicht – verglichen mit dem initialen Release Date von OpenGL [1] welches im Jahre 1992 publiziert wurde. [1, 3] Nicht nur das Alter, sondern auch die Natur der Web Plattform haben dazu beigetragen, dass WebGL ein langsames Wachstum zu Beginn verspürt hat. Um

[1] Spezifikation einer Programmierschnittstelle zur Entwicklung von 2D und 3D-Grafikanwendungen

einen Webstandard wie WebGL einsetzen zu können müssen alle grossen Browser die Spezifikation implementieren. So hat zum Beispiel Internet Explorer 10 keinen Support und es wurde erstmals Ende 2013 möglich im Internet Explorer 11 3D Anwendungen für ein breites Publikum zu entwickeln.

2.3. 3D-Modelle

Ein Modell stellt ein Objekt aus der realen Welt vereinfacht dar. 3D-Modelle können vereinfacht als Gruppe von Punkten definiert werden. Um im dreidimensionalen Raum Objekte visualisieren zu können sind mindestens drei Punkte notwendig. Punkte von 3D Modellen werden im folgenden als Vertex (Eckpunkte) bezeichnet und können als Vektor definiert werden. So können wir einen Vertex am Ursprung eines Koordinatensystems definieren als:

$$V = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Eine Sammlung von drei Vertices bildet ein Dreieck. Um komplexere Formen wie Vierecke (Quads) zu bilden werden jeweils Dreiecke kombiniert. Für eine Sammlung von Punkten wird generell der Term Polygon verwendet. Ein Modell besteht aus einer beliebigen Anzahl Polygone. Polygone bestehen aus einer theoretisch beliebigen Anzahl Vertices. Die Verbindung zwischen zwei Vertices ist eine sogenannte Edge (Kante). Verbindet man die Punkte eines Polygons und füllt die Fläche, ergibt sich schlussendlich ein Face (Fläche).

Weitere Attribute

Neben den geometrischen Attributen verfügt ein Modell ebenfalls über visuelle Attribute. So wird für jeden Vertex ein sogenannter Normal definiert. Ein Normal ist ein Vektor der im einfachsten Fall Perpendikular [2] zu den zwei an diesem Vertex verbundenen Edges verläuft. Normals werden häufig für das Berechnen von Reflexionen verwendet. Normals werden auch für Performanz Optimierungen eingesetzt, dazu mehr in Abschnitt 2.6.2. Um die Oberfläche von Modellen zu definieren wird häufig ein sogenanntes Texture Mapping [3] durchgeführt. Hierfür sind zusätzliche Informationen für ein Modell notwendig. In der Praxis finden viele weitere Methoden Anwendung, auf diese wird jedoch hier nicht weiter eingegangen.

Formate

Um ein 3D Modell in einer Anwendung zu verwenden, muss ein entsprechendes Format verwendet werden. Hierfür steht eine Vielzahl von Optionen zur Auswahl. Aufgrund der Menge wird hier jedoch nur oberflächlich auf die bekanntesten Formate eingegangen.

[2] Senkrecht
[3] Verfahren, mittels 2D-Bildern 3D-Objekte zu gestalten.

Wavefront OBJ ist ein offenes Dateiformat das von Wavefront Technologies 1989 entwickelt wurde. Das Format ist jedoch speichertechnisch ineffizient und verfügt zudem nur über ein limitiertes Feature Set. [4] FBX ist ein proprietäres Format, welches von Autodesk verwaltet wird. Das Verwenden von FBX Daten ist offiziell nur mit einer C++ FBX SDK möglich, welche für das Web nicht geeignet ist. Im Kontrast zu diesen beiden Formaten gibt es seit 2015 ein offenes und modernes sowie auf Speicherplatz Reduktion fokussiertes Format: glTF. Dieses Format wird von der Khronos Gruppe entwickelt und wird in dieser Arbeit als Austauschformat verwendet. [5]

glTF Die meisten 3D-Bearbeitungsapplikationen wie Blender (.blend), Maya (.ma) oder Lightwave3D (.lws) verwenden ihre eigenen proprietären Dateiformate. Deswegen brauchte es für jedes Eingangsformat für jedes Ausgangsformat einen *converter*, wie in Abbildung 2.1 ersichtlich.

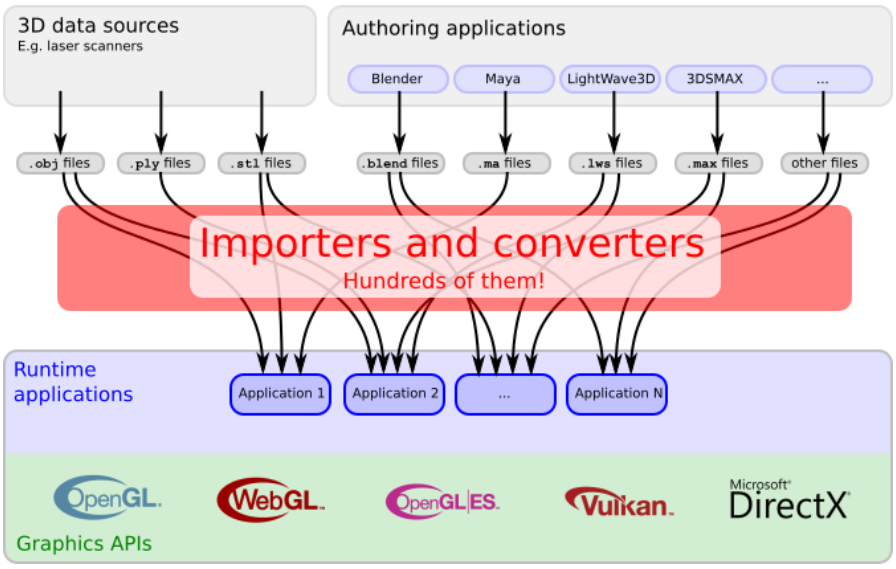

Abbildung 2.1.: Convertierungspipeline vor glTF [5]

Durch die immer breiter werdende Nachfrage nach 3D-Applikationen wurde ein Format benötigt, dass zum einen Applikationsunabhängig verwendet und zum anderen performant im Web eingesetzt werden kann. [5] Durch diese Abstraktion können hunderte *converter* vermieden werden und nur wenige, sehr spezifische Dateiformate benötigen noch einen einzigen *converter*. Diese neue Pipeline ist in Abbildung 2.2 dargestellt. [5]

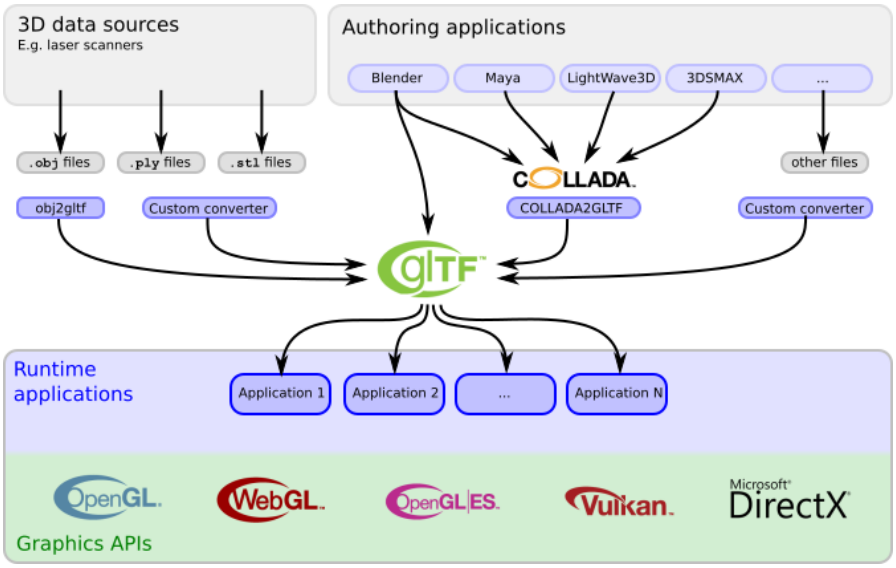

Abbildung 2.2.: Convertierungspipeline mit glTF [5]

Die Basisstruktur von glTF basiert auf JSON. Darin ist die komplette Szenerie des Modells beschrieben und in Abbildung 2.3 aufgezeigt.[5]

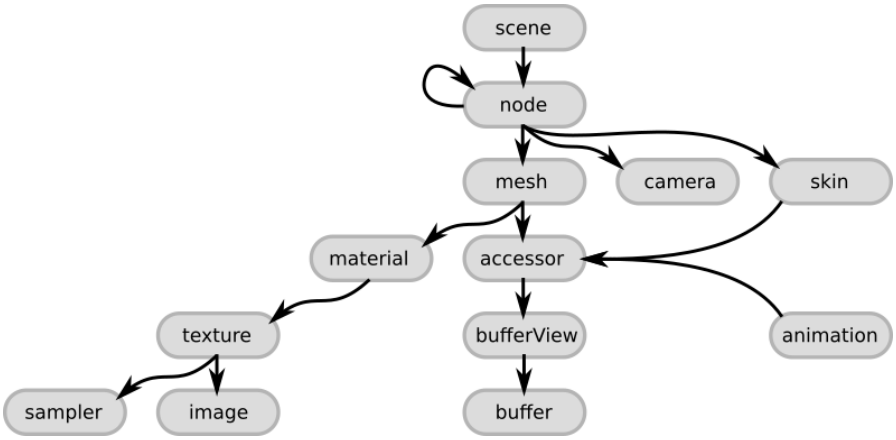

Abbildung 2.3.: glTF Datenstruktur [5]

Die wichtigsten Elemente dieser Struktur kurz erläutert:

- **Scene**: Einstiegspunkt der Szenerie und verweist auf alle Top-Level *nodes*.
- **Node**: Kann eine Transformation (Rotation, Translation oder Skalierung) beinhalten oder eine Kamera, *Skin*, Animation, ein *Mesh* oder *Childnodes* referenzieren.
- **Camera**: Definiert die Ansichtskonfiguration.

- **Mesh**: Beschreibt ein geometrisches Objekt und verweist auf *accessor*, welcher die effektiven geometrischen Daten beinhaltet und auf *material*, das beschreibt, wie das Objekt beim Rendern aussehen soll.
- **Accessor**: Verweis auf die Binärdaten (*BufferView*), welche die effektiven Eigenschaften für *meshes*, *skins* und *animations* kompakt beinhaltet.
- **BufferView**: Definiert eine Ansicht (Länge, Ort, Typ)auf einen *Buffer*.
- **Buffer**: Verweist auf einen Block von Binärdaten, welchen die effektiven Daten des 3D-Modells platzsparend beinhaltet.
- Weitere Elemente, welche im Rahmen dieser nicht relevant sind und nicht im Detail betrachtet werden, sind: *skin*, *material*, *animation*, *texture*.

2.4. Transformation von Modellen

Um ein Modell vereinfachen zu können, muss es verändert werden. Diese Veränderungen können in Basis Operationen vereinfacht erläutert werden.
Im Folgenden werden Transformationen mithilfe einer 2D Visualisierung erläutert. Das Modell könnte jedoch ebenfalls im dreidimensionalen Raum sein – die Funktionsweise bleibt identisch.
Für die folgenden Beispiele wird jeweils das Modell aus Abbildung 2.4 verwendet.

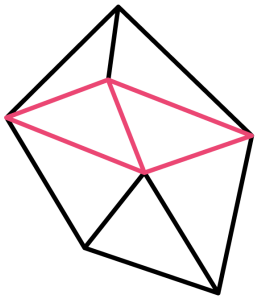

Abbildung 2.4.: Modell Basis

Edge Collapse

Hierbei werden zwei nebeneinanderliegende Vertices kombiniert, siehe Abbildung 2.5. Die Umkehrfunktion nennt man Vertex Split.

(a) Modell Ausgangslage (b) Edge Collapse

Abbildung 2.5.: Transformation mittels Edge Collapse

Halfedge Collapse

Hierbei wird ein Vertex direkt entfernt und alle Edges auf einen danebenliegenden Vertex zusammengelegt, siehe Abbildung 2.6.

(a) Modell Ausgangslage (b) Halfedge Collapse

Abbildung 2.6.: Transformation mittels Halfegde Collapse

Vertex Removal

Hierbei wird ein Vertex entfernt und das Resultat neu Trianguliert [4] . In Abbildung 2.7a wird der zentrale Vertex entfernt. Der neu triangulierte Polygon ist in Abbildung 2.7b ersichtlich.

[4]Verfahren, Polygone in Dreiecke aufzuteilen

(a) Modell Ausgangslage (b) Vertex Removal

Abbildung 2.7.: Transformation mittels Vertex Removal

2.5. Graphics Pipeline

Die Graphics Pipeline ist zuständig um eine definierte Szene auf einem Ausgabegerät zu visualisieren. Die verschiedenen Schritte werden im folgenden Abschnitt kurz erläutert. Es wird dabei ein simples 3D-Modell auf einen normalen 2D Display visualisiert. Die verschiedenen Schritte sind in Abbildung 2.8 aufgezeigt.

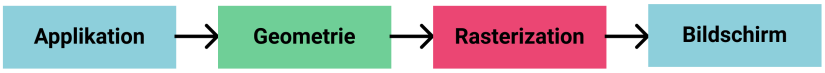

Abbildung 2.8.: Schritte einer Graphics Pipeline

Applikation

Im ersten Schritt wird die Szenerie aufbereitet, Modelle in den Arbeitsspeicher geladen, Positionen und Rotationen der Modelle definiert. Diese Schritte werden auf der CPU durchgeführt.

Geometrie

Anschliessend werden die Definitionen an die Geometrie Pipeline geliefert. Diese ist zuständig um die Modelle auszurichten, die Beleuchtung zu bestimmen und die 2D Projektion vorzunehmen.

Rasterization

Im letzten Schritt werden kontinuierliche Objekte zu diskreten Fragmenten verarbeitet. Der Anwender hat hierbei in vielen Engines die Möglichkeit, den Prozess mithilfe eines Fragmentshaders [5] anzupassen.

Bildschirm

Am Schluss wird das vom Rasterization Prozess generierte Bild auf dem Bildschirm dargestellt.

2.6. Einführung LOD

Als Level Of Detail (LOD) werden die verschiedenen Detailstufen bei der virtuellen Darstellung bezeichnet. Dies wird verwendet, um die Geschwindigkeit von Anwendungen zu steigern, indem Objekte im Nahbereich detailiert angezeigt werden; wohingegen Elemente im Fernbereich deutlich vereinfacht dargestellt werden.

2.6.1. Das Problem

Geometrische Objekte können zu Komplex werden, um jederzeit performant und interaktiv gerendert zu werden. Gerade wenn viele Objekte zur selben Zeit sichtbar sind, lohnt es sich, zu priorisieren und gewisse Objekte in reduzierter Qualität anzuzeigen. Im Idealfall geschieht dies jedoch, ohne dass der Anwender dies bemerkt.

2.6.2. Lösungsansätze

In diesem Abschnitt werden mögliche Ansätze erklärt, welche helfen sollen, die Render-Perfomanz zu erhöhen. Diese Arbeit konzentriert sich jedoch auf den Ansatz von Level of Detail; die anderen Ansätze werden nur kurz erläutert.

Level of Detail

Level of Detail auch bekannt als polygonale Simplifizierung, geometrische Simplifizierung oder Mesh Reduzierung basiert darauf, die Komplexität von Objekten zu reduzieren, welche weiter von der Kamera entfernt werden. Es gibt verschiedene Ansätze zur Generierung von LODs, welche in Abschnitt XY im Detail erläutert werden. Zudem braucht es einen berechenbare Methode, die Genauigkeit von Modellen zu definieren, um diese entsprechend anzuwenden. Schlussendlich ist dann noch zu definieren, ab wann welche Artefakte verwendet werden soll, basierend auf der Genauigkeit und der Komplexität.

[5] Generiert Farbdefinition für ein einzelnes Pixel mit zugehörigem Tiefenwert

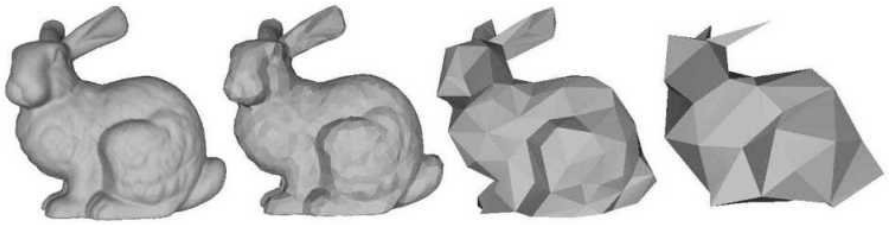

Abbildung 2.9.: Level Of Detail Visualisierung vier Hasen

Wie in der Abbildung 2.9 zu erkennen ist, wird von links nach rechts der Detailgrad und somit die Komplexität des Objektes reduziert. Sind es im Bild ganz links noch 69'451 Polygone, wird es bereits im ersten Schritt auf 2'502 Polygone reduziert. Dies ist eine enorme Reduktion von ca 96.5%. Im dritten Schritt wird die Anzahl Polygone wiederum um ca. 90% auf 251 reduziert. Schlussendlich hat das letzte Objekt noch 76 Polygone was knapp 0.1% der ursprünglichen Anzahl entspricht.

Frustum culling

Polygone, welche nicht im Kamera Frustum enthalten sind, werden bei dieser Methode nicht weiter prozessiert. Dies reduziert die Anzahl Polygone drastisch.

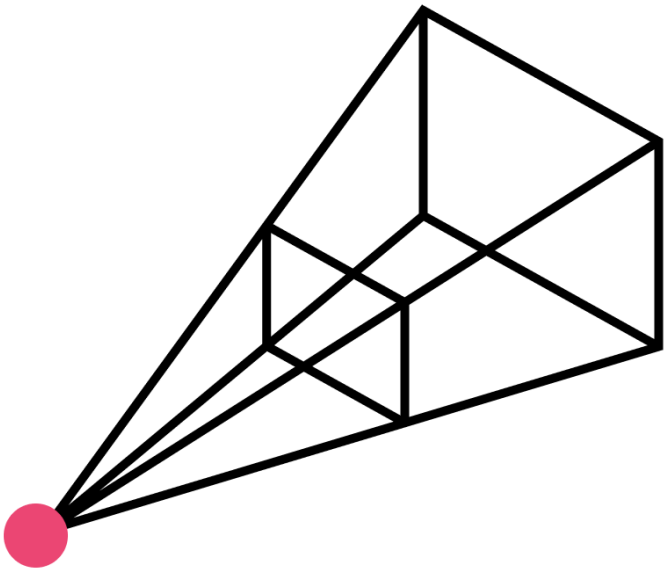

Abbildung 2.10.: Camera Frustum

Ein Frustum ist eine geometrische Form, die einer Pyramide ähnelt, welcher die Spitze abgeschnitten wurde. Das Kamera-Frustum bezeichnet den Raum, welcher von der Kamera aufgezeichnet wird. Dieser Breich startet eine gewisse Distanz von der Kamera entfernt und endet weit hinten wieder, da nicht bis ins unendliche Objekte sichtbar sind. Wie in Abbildung 2.10 zusehen ist, beginnt das Kamera-Frustum in der nähe der Kamera und dehnt sich ein wenig aus bis es gegen Ende des Spektrums beschränkt wird.

Occlusion culling

Polygone bzw. Objekte, welche komplett von anderen Objekten überdeckt werden, werden bei dieser Variante nicht prozessiert.

Backface culling

Bei dieser Methode wird berechnet welche Polygone zur Kamera orientiert sind. Alle Polygone, welche in die andere Richtung zeigen werden nicht gezeichnet. Dies ist nicht immer gewünscht, für die meisten Anwendungen ist diese Optimierung jedoch aktiviert. Als Grundlage für die Berechnung werden die Normals der Vertices berücksichtigt.

Parallel rendering

Auch bekannt unter Distributed Rendering ist der Einsatz von Techniken aus dem Parallel Programming in Visualisierungsanwendungen.

Image-based rendering

In gewissen Fällen kann das Modellieren übersprungen werden. So kann anhand von Bildmaterial eine 3D-Illusion erzeugt werden.

2.6.3. Verschiedene Ansätze zur Generierung von LOD Modellen

Es gibt verschiedene Ansätze, 3D-Modelle mittels LOD zu vereinfachen. In diesem Abschnitt werden einige davon detaillierter erläutert so wie ihre Vor- und Nachteile aufgezeigt.

Diskrete LOD (DLOD)

Bei diskreten LOD werden für ein detailliertes Modell mehrere weniger detaillierte Modelle verwendet. Abhängig von der Distanz zum Betrachter wird das optimale Modell gewählt.

+ Simplizität: Keine Anpassungen am Scene Graph [6] notwendig
− Harte Grenzen: Veränderung des Objektes kann merkbar sein
− Kein Clustering möglich: Probleme bei sehr grossen oder vielen kleinen Modellen

[6] Objektorientierte Datenstruktur zur Beschreibung von 2D- oder 3D-Szenarien

Kontinuierliche LOD (CLOD)

Im Gegensatz zu DLOD wird bei CLOD vereinfachende Veränderungen an einem Modell gespeichert.

+ Weiche Grenzen: Interpolation zwischen Auflösungen ist möglich
− Runtime Performance
− Kein Clustering möglich

Hierarchische LOD (HLOD)

Bei HLOD werden mehrere Objekte in einen Cluster gruppiert.

+ Clustering möglich

3. Vorgehen

3.1. Nutzen LOD

Um den Nutzen von LOD quantifizieren zu können, wird in einer ersten Phase ein Benchmark aufgestellt. Ziel ist es, das Laufzeitverhalten unter Einsatz eines optimierten Modells zu analysieren und somit den maximal möglichen Einfluss von LOD auf die Leistung klassifizieren zu können.

3.1.1. Definition Benchmark

Es gibt einige Ansätze, die das Durchführen eines Benchmarks vereinfachen. In diesem Abschnitt werden die verschiedenen erwägten Optionen erläutert und die Abgrenzungen aufgezeigt.

Browser Umgebung

Um den Umfang des Benchmarks überschaubar zu halten, wurde ausschliesslich ein Benchmark für Google Chrome entwickelt. Google Chrome basiert auf Chromium [1], dieselbe Engine, welche auch Microsoft Edge oder Opera verwenden. Einen Benchmark basierend auf Google Chrome liefert somit auch Indizien für diese beiden Browser, auch wenn gewisse Abweichungen möglich sind. Neben dem Marktführer Chrome sind Mozilla Firefox oder Safari von Apple ebenfalls Optionen. Jedoch wurde primär aufgrund des Marktanteils zugunsten von Google Chrome entschieden.

Automation

Um die Tests durchzuführen, wird ein Testautomationstool benötigt; unter anderem der Einsatz von Selenium wurde in Erwägung gezogen. Der Vorteil von Selenium ist insbesondere, dass der Benchmark für weitere Browser ausgeweitet werden könnte. Da jedoch das Analysieren der GPU Daten stark vom System abhängig ist und dafür zusätzliche Komplexität notwendig wäre, wird in diesem Benchmark die im Google Chrome integrierten Chrome DevTools eingesetzt. Selenium bietet zurzeit eine suboptimale Integration für das *Chrome DevTools Protocol*. Puppeteer [2], eine weitere Option für die Automation, ist eine Bibliothek, die eine vereinfachte Schnittstelle zu einer Headless Chrome Instanz bietet. Sie wird zudem direkt von Google entwickelt und bietet somit eine stabile Grundlage zur Kommunikation mit den DevTools.

[1] Open Source Browser-Projekt. Grundlage von Browsern wie Google Chrome, Edge oder Opera

[2] NodeJS Library, die eine API anbietet zum steuern von Chromium oder Chrome über das Chrome DevTools Protocol

Profiling

Die Chrome DevTools erlauben es, ein detailliertes Profil einer Applikation anzulegen. Im Profil befinden sich Informationen zu CPU/GPU Auslastung aber auch generelle Informationen bzgl. der Rendering Engine werden gesammelt. Die Analyse dieser Daten ermöglicht es, eine Aussage zum Laufzeitverhalten einer Applikation zu tätigen.

Testablauf

Bei einem Testablauf werden folgende Schritte durchlaufen:

1. Öffne Applikation in *Headless Chrome* Instanz.
2. Warte bis Seite geladen ist.
3. Starte *Profiling*.
4. Warte n Sekunden.
5. Stoppe *Profiling*.
6. Werte Daten aus.

Der Test erfasst kontinuierlich die FPS [3] . Zudem wird aus dem *Profiling* eine Kennzahl bzgl. der GPU Nutzung berechnet.

Analyse der Daten

Um eine zuverlässige Aussage treffen zu können, wird der Vorgang mehrfach wiederholt. Für jeden Durchlauf wird der Median der *FPS* Daten berechnet. Anschliessend wird die Varianz der FPS für die unoptimierten respektive optimierten Werte berechnet. Die Varianz dient als Kennzahl, um eine statistische Signifikanz nachweisen zu können.

3.2. Vergleich LOD Systeme

Die unterschiedlichen LOD Systeme bieten allesamt ihre Vor- und Nachteile. In diesem Schritt wird erläutert, welche Art LOD System in dieser Arbeit eingesetzt werden soll.

3.3. Vergleich LOD Algorithmen

Abhängig vom LOD System wird ein passender Algorithmus ausgesucht.

3.4. Pipeline Integration

Die Lösung soll in eine wiederverwendbare und konfigurierbare Pipeline integriert werden.

[3] Frames per Second. Bilder pro Sekunde. Gängige Art, die Performanz der Rendering Engine zu messen

3.5. Automatische Generierung von Detail Levels

Damit für den Endbenutzer die Konfiguration übersichtlich bleibt, soll der Einsatz von zum Beispiel heuristischen Methoden hilfreiche Basiskonfigurationen liefern. Zum einen geht es hier um das Festlegen der Thresholds, aber auch das Definieren des Dezimierungs- / Vereinfachungsfaktors spielt eine wichtige Rolle.

3.6. Levelwahl während Laufzeit

Die optimale Levelwahl kann stark von der Laufzeitumgebung abhängig sein. So sollte zum Beispiel auf mobilen Geräten früher ein einfacheres Modell gezeigt werden als auf leistungsstarken Geräten. Deshalb eignet sich eine Wahl der Levels zur Laufzeit, um ein optimales Erlebnis auf allen Geräten zu ermöglichen.

4. Resultate

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5. Diskussion und Ausblick

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6. Verzeichnisse

Literaturverzeichnis

[1] D. Jackson und J. Gilbert. (2021) *WebGL Specification.* [Online]. URL: https://www.khronos.org/registry/webgl/specs/latest/1.0/ [Stand: 26.03.2021].

[2] GPU for the Web Community Group. (2021) *GPU for the Web Community Group Charter.* [Online]. URL: https://gpuweb.github.io/admin/cg-charter.html [Stand: 26.03.2021].

[3] M. Segal, K. Akeley, C. Frazier, J. Leech und P. Brown. (2008) *The OpenGL® Graphics System: A Specification.* [Online]. URL: https://www.khronos.org/registry/OpenGL/specs/gl/glspec30.pdf [Stand: 26.03.2021].

[4] unknown. (2007) *Object Files.* [Online]. URL: http://www.martinreddy.net/gfx/3d/OBJ.spec [Stand: 26.03.2021].

[5] P. Cozzi und T. Parisi. (2016) *glTF 1.0 Specification.* [Online]. URL: https://github.com/KhronosGroup/glTF/tree/master/specification/1.0 [Stand: 26.03.2021].

Abbildungsverzeichnis

Tabellenverzeichnis

A. Anhang

A.1. Aufgabenstellung

Aufgabenstellung

A.2. Anhang 2

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.