OpenGL EERT
- Einzelprojekt -

Universität Oldenburg
Wintersemester 2008/2009

Robert Schadek

21. Februar 2009

Inhaltsverzeichnis

1 Einleitung, Projektziel

Der Name meines Projektes lautet EERT. Dies steht für EERT enhanced rendering technology. Die Idee hinter dem Projekt ist jene, dass ich ein Programm schaffen wollte, welches in der Lage ist neue Szenen darstellen zu können, ohne jedes mal neu kompiliert zu werden. Außerdem wollte ich Frustum Culling implementieren, da dies, wie ich finde, zu jeder Grafikanwendung gehört die Echtzeit fähig sein will. Außerdem wollte ich in der Lage sein eine Szene darzustellen die theoretisch aus mehr als einer Millionen Triangle besteht und dies mit Frameraten über 100FPS.

2 Techniken

2.1 Allgemein

Zu den verwendeten Techniken kann man allgemein sagen, dass ich nicht versucht habe, auf biegen und brechen, jede vorgestellte Technik aus der Vorlesung in mein Projekt einzubauen. Ich habe vielmehr versucht Techniken zu implementieren, die nicht vorgestellt wurden, allerdings angesprochen.

2.2 Szenenbeschreibung

Meine Szenenbeschreibung sieht so aus, dass ich mir eine Dateistruktur ausgedacht hab in der man speichern kann welches Object geladen wird, welche Texturen zu ihm gehören, wo sich die Objektinstanzen befinden, wie sich jene verschieben und rotieren usw. Um mir die Arbeit einfacher zu machen habe ich mich mir das .obj Format als Vorbild genommen. Die Szenendatein tragen die Endung .eob was für eert objekt seht.
Intern sind diese wie folgt aufgebaut:

l float float float float float float
Erzeugt ein direktionales Licht, wobei die ersten drei floats die Richtung und die zweiten drei die Frabe angeben.

p long float float float float float float float float
Erzeugt einen Pfad auf dem sich die Kamera bewegt. Der long Wert gibt an wie lange es in Millisekunden dauert bis sich die Kamera vom ersten bis zum dritten Vektor bewegt hat. Es müssen mindestens drei Vektoren oder anderes gesagt neun floats angegeben werden. Es können dann zusätzlich jeweils drei

weitere Vektoren angegeben werden.

a float float float float float float float float float
Erzeugt einen Pfad auf den die Kamera guckt, während sie sich über den unter p deklarierten Pfad bewegt. Es sind hier genau so viel Vektoren zu deklarienen wie für den Pfad. Die Bewegungsgeschwindigkeit ist jene unter p deklarierte.

o *.obj *.obj *.obj *.obj *.obj *.obj
Dies erzeugt ein Objekt die sechs *.obj geben jeweils die Dateinamen der Objekte an. Wobei diese aufgrund des Verwendeten LOD Verfahrens(siehe Unten) vom hochauflösend absteigent sortiert sind.

ot *.png *.png *.png *.png *.png *.png
Hiermit werden die Texturen für das als nächste Deklarierte Objekt gesetzt. Wie bei den Objekten sollten sechs Auflösungen für die verschiedenen LOD Auflösungen vorgehalten werden.

oi int float float float float float float float float float float float float
Eine mit oi beginnende Zeile beschreibt eine Objektinstanz. Das int zu beginn gibt Auskunft über, dass dazugehöhrige Objekt. Die nächsten drei floats beschreiben den Startvektor gefolgt von der Startrotation und der Konstanten Rotation. Alle folgenden drei floats beschreiben eine Bewegungsrichtung.

2.3 Editor

Der Editor entstand, als es klar wurde, dass ich eine Vielzahl von Objektinstanzen in meiner Beispielszene darstellen wollte. Hätte ich diese per Hand geschrieben, hätte ich für jede Objektinstanz mindestens neun floats eingeben müssen. Um dies ein wenig zu vereinfachen, habe ich einen kleinen Editor geschrieben der dies übernimmt. Die Hauptaufgabe des Editors ist es Positionen innerhalb der Szene zu finden in denen nicht bereits ein Objekt plaziert ist. Aufgrund von Webstart ist es nicht möglich den Editor zu demonstrieren. Dies ist deshalb der Fall, da Webstart nicht den Zugriff auf das lokale Datensystem erlaubt und die durch den Editor erstelle Datei noch ein wenig manuellen Eingriff benötigt, bis sie darstellbar ist.

2.4 Objektloader

Der Implementierte Objektloader ist in der Lage jede Form von Mesh zu laden, solange sie nur aus Dreiecken besteht und geschlossen sind. Die Geschlossenheite ist Voraussetzung für Shadow Volumen. Wie oben bereits angedeutet lade ich Objekte des Typs **.obj**. Ich habe dieses Format gewählt, da es recht einfach zu verstehen ist und für meine Zwecke vollkommen ausreicht.

2.5 Szenenmusik

Damit die Szene nicht zu steril wirkt, habe ich mit Hilfe der Jlayer Libray die Wiedergabe von MP3 Datein implementiert. Da dies kein Feature von OpenGL ist, werde ich hier nicht weiter drauf eingehen.

2.6 Per Pixel Lightning

Wie die Überschrift es bereits andeutet, ist die Beleutung durch Per-Pixel Lightning realisieren. Aus dem einfachen Grund, da dies zur Zeit Stand der Technik ist. Da die Demoszene ein Skybox enthällt die einen Himmel darstellt, hab ich das Per Pixel Lightning für direktionales Licht implementiert.

2.7 Objektinstanzen

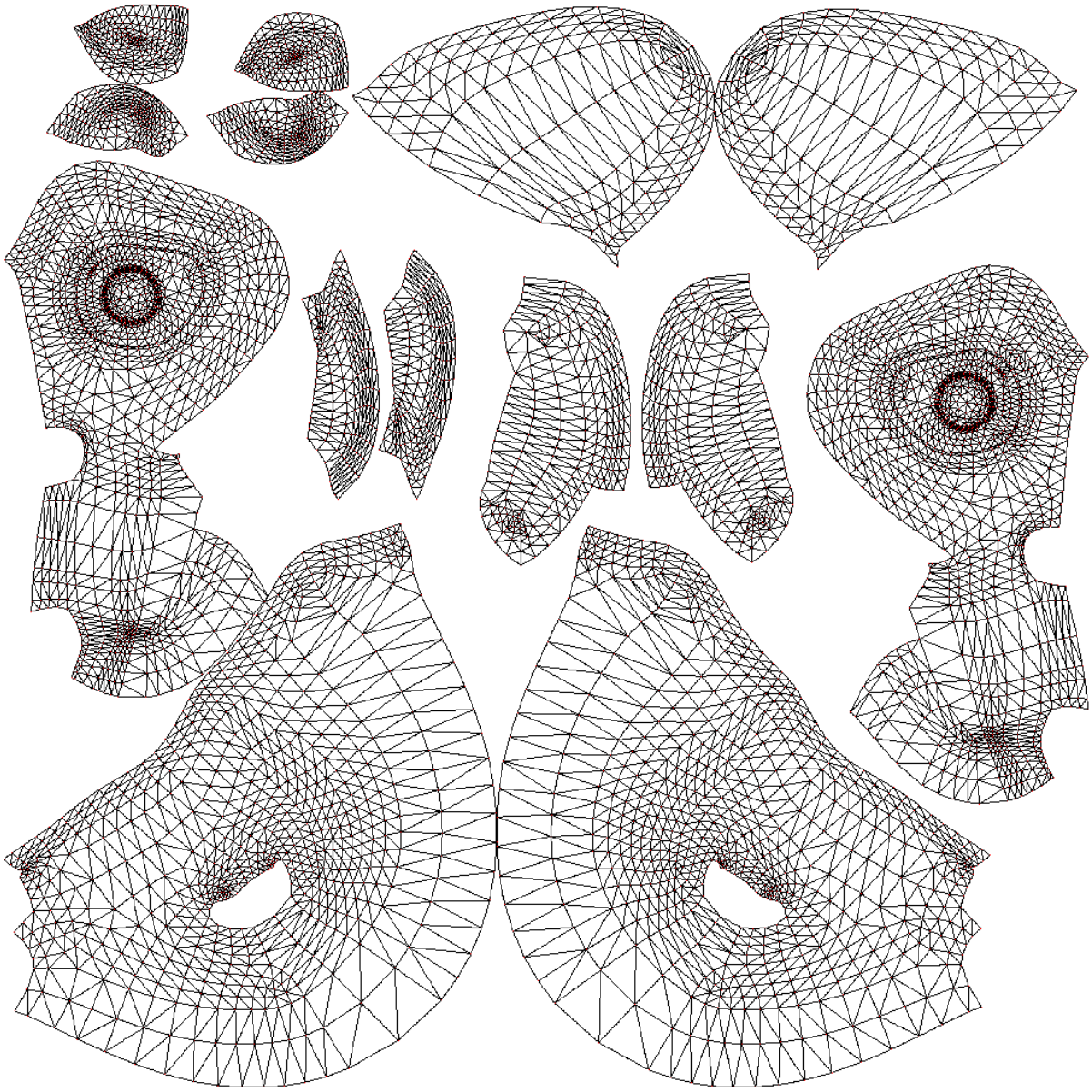
Unter Objektinstanzen hat man eine Technik zu verstehen, die es mir ermöglicht einen Mesh nur einmal im Speicher zu halten, ihn aber an vielen Stellen der Szene mit verschiedenen Eigentschaften zu zeichnen. Dies ist Sinnvoll,da die Objekte die ich in einer Szene laden alleine ca. 2MB groß sind. Nimmt man nun an man würde dieses Objekt 400 mal laden um es an 400 Stellen zu zeichnen bräucht ich alleine ca.

800MB Speicher nur für die Objektdaten. 400 mal deshalb da ich in der Demoszene 400 Objektinstanzen darstelle.
Dank der Objektinstanzen muss ich die Objektdaten nur einmal laden. Für jede Instanz muss prinzipell nur noch ein Postions- sowie Rotationsvektor gespeichert werden.

2.8 UV-Texturing

Beim UV-Texturing oder wie es auch genannt wird UV-Mapping, wird ein komplexes 3D Objekt derart auseinander gefallt, dass er sich auf einer 2D Fläche sprich Texture darstellen lässt. Dies ermöglicht es Objekte zu texturieren ohne dabei auf den Hilfsmittel von OpenGL zurückzugreifen. Dies macht Sinn, da es mit diesen Hilfsmitteln nicht möglich ist, die Texturen beliebig komplex auf Objekten abzubilden. Das UV-Mapping steht in enger Verbindung mit dem Objektloader, da beim Erstellen der Objekte bereits die sogenante UV-Map erstellt werden muss. Dies kann dann später in einem beliebigen Zeichnenprogramm bearbeitet werden.

Hier ein Beispiel für eine solche UV-Texture.



2.9 Level of Detail

Level of Detail kann man als Mipmapping für Dreiecke verstehen. Ich setze voraus, dass jedes Objekt welches in EERT darstellt werden soll, in sechs Auflösungen vorliegt. Beispielhaft von 10000 bis 300 Dreiecken. Dies mache ich mir so zu nutze, indem ich sage, wenn ein Objekt so weit von der Kamera entfernt ist, dass es nurnoch wenige Pixel auf dem Bildschirm einnimmt, brauch es nicht aus mehrere tausend Dreiecken bestehen, es reicht wenn jenes aus ein paar hundert besteht. Betrachtet man nun mehrere diese Stufen fällt es dem Benutzer nicht auf, dass bei bestimmen Abständen von der Kamera eigentlich verschiedene Objekte gezeichnet werden. Mit Abstand ist die Euklidische Distanz zwischen Kamerapostion und Objektmittelpunkt gemeint.

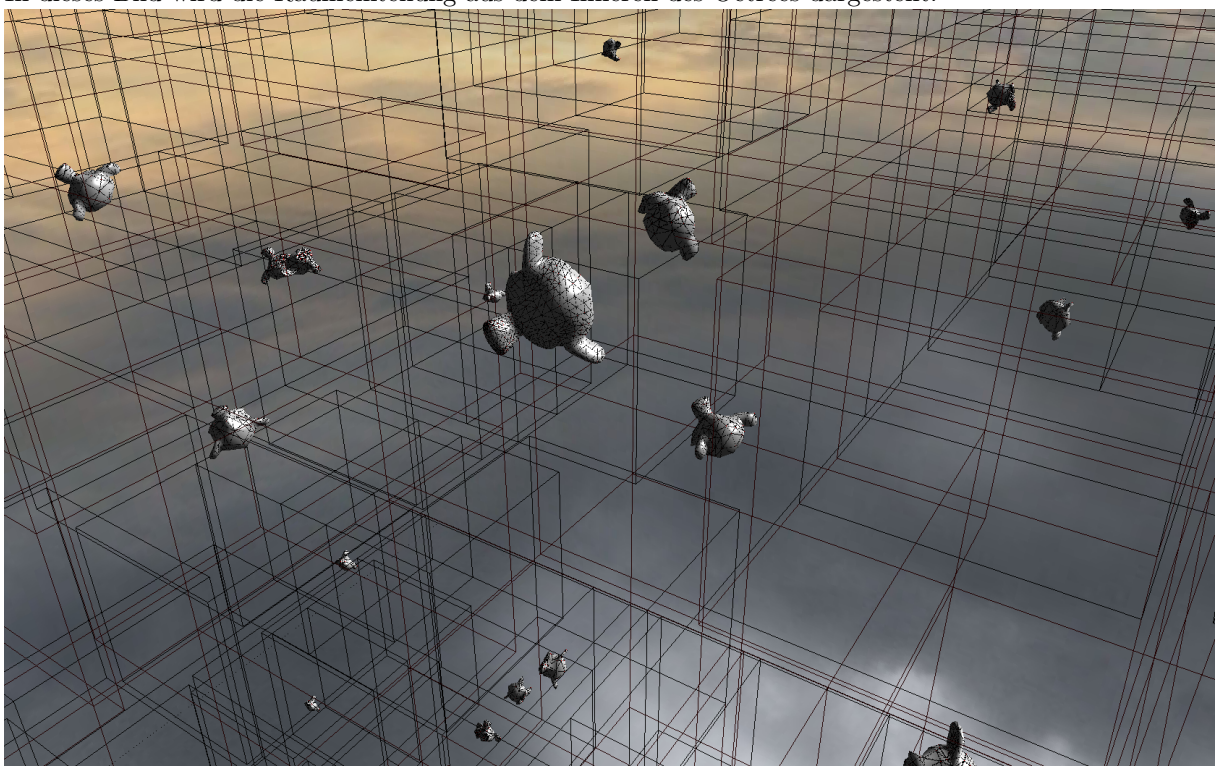
2.10 Shadow Volumes

Shadow Volumes ist eine Technik zum darstellen von Schatten. Die Technik funktioniert so, dass ausgehend von einer Lichtquelle die Siluette eines jeden Objekt extrudiert wird und der dadurch entstandene Körper als nicht durch die entsprechende Lichquelle beleutet wird. Eine genauere Beschreibung würde den Umfang dieses Dokuments platzen lassen. Dieses Feature ist aufgrund eines Fehler nicht in der Engine aktiviert.

2.11 Octree

Die Idee hinter dem Octree ist es, den Raum in dem sich die zu zeichnen Objekte befinden in acht Unterräume aufzuteilen. Nachdem der Raum das erste mal aufgeteilt wurde wird überprüft welche Objekte in welche Unterraum liegen. Danach wird jeder Unterraum wiederrum in acht Unteräume aufgeteilt und es wird erneut überprüft welche Objekte sich in ihm befinden. Dies wird solange fortgeführt bis eine bestimmte Rekursionstiefe erreicht ist oder ein Unterraum leer ist. Es ist darauf zu achten was für eine Rekursionstiefe man wählt da die Anzahl der Knoten mit $O(8^n)$ wächst.

In dieses Bild wird die Raumeinteilung aus dem Inneren des Octrees dargestellt.

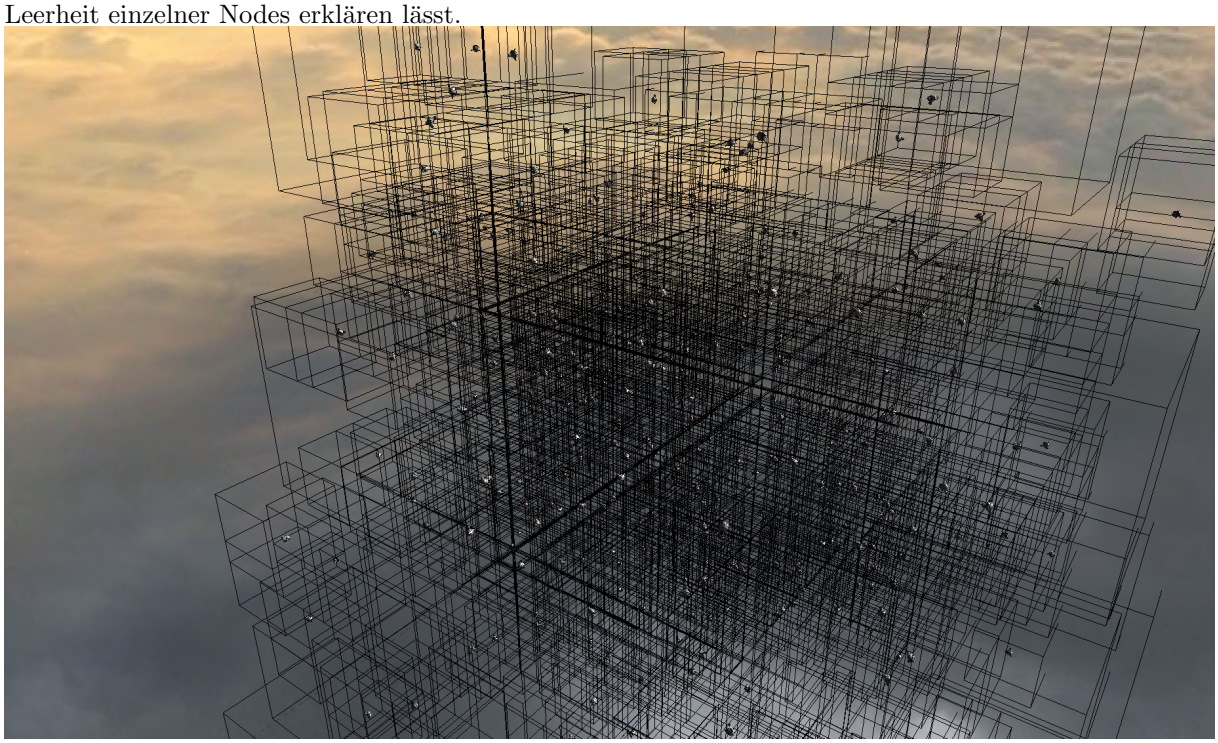


Beim Rendern überprüft man nun, ob sich die Root-node im Frustum der Camera befindet, ist dies nicht so ist der Rendervorgang bereits vorbei, da es ausgeschlossen ist, dass sich irgendeine weitere Node und

somit irgendein Objekt im Frustum befindet. Sollte sich eine Node im Frustum befinden werde alle ihre Kinder überprüft. Dies geschieht solange bis die Überprüft Node keine Kinder mehr hat, sollte sie dies jeweilige Node immernoch im Frustum befinden werden alle Objekte die sich in ihr befinden gezeichnet.

Dies ermöglicht ein überaus effektives Frustum Culling. Die Datenstruktur ist zudem so schnell aufzubauen, dass es m"öglich ist, diese jedem Frame neu aufzubauen. Somit können alle Objekte frei bewegt werden.

Die Datenstruktur eignet sich nicht nur für das Frustum Culling zusätzlich ist es auch eine effizente Technik für das Überprüfen von Kollsionen.
Dieses Bild zeigt den Octree von außen. Wie man sieht es er kein vollständiger Würfel was sich durch die Leerheit einzelner Nodes erklären lässt.



3 Zusammenfassung

Durch die oben beschrieben Techniken ist EERT in der Lage eine Szene die theoretisch aus 4.05 Millionen Dreicken besteht mit durchschnittlich 200 FPS bei einer Auflösung von 1920x1200 zu rendern. Somit wurde das angestrebte Projektziel erreicht. Die Zahl 4.05 Million kommt daher das 400 Objekt die jeweils eine Auflösung von 10.138 Dreiecken besitzten gezeichnet werden.