OpenGL EERT
- Einzelprojekt -

Universität Oldenburg
Wintersemester 2008/2009

Robert Schadek

28. Januar 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 neue Szenen darstellen kann, 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. Wie die letztendliche Abgabe aussehen wird kann ich zu diesem Zeitpunkt noch nicht sagen, da sich die Szenen wie bereits angesporchen ohne ändern des Quellcodes anpassen lassen sollen. Aller Voraussicht werde ich aber eine Variante der beigelegten Testszene verwenden, da diese sich sehr gut dazu eignet die Fähigkeiten von EERT zu demonstrieren.

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, aber doch zu jeder Grafik-Engine dazugehören. Außerdem wollte ich es ermöglichen Szenen von großen Polygonenumfang zu zeichnen. (> 1Mil Dreiecke)

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. Und mir die Arbeit einfach zu machen habe ich mich mir das .obj Format als Vorbild genommen.

2.3 Editor

Um nicht alle Informationen in der Szenendatei von Hand einzutragen, habe ich mit ein kleines Programm geschrieben welches mir eine Szenen zufällig anhand von bestimmen Parametern erzeugt.

2.4 Objektloader

Der Implementierte Objektloader ist in der Lage jede Form von Mesh zu laden, solange sie nur aus Dreiecken besteht. 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.

2.6 Per Pixel Lightning

Wie die Überschrift es bereits andeutet, soll das fertige Programm die Beleutung durch Per-Pixel Lightning realisieren. Aus dem einfachen Grund, da dies zur Zeit stand der Technik ist. Sollte ich die Zeit finden, werde ich versuchen zusätzlich, dass Per Pixel Lightning noch mit Normal-Maps zu verbinden.

2.7 Objektinstanzen

Unter Objektinstanzen hat man eine Technik zu verstehen, die es mir ermöglichteinen Mesh nur einmal im Speicher zu halten, ihn aber an vielen Stellen der Szene mit verschiedenen Eigentschaften zu zeichnen. Dies ist Sinnvoll, die Objekte die ich in einer Szene laden alleine ca. 2MB groß sind. Nimmt man nun an ich würde diese Objekte nun 400 mal laden um sie an 400 Stellen zu zeichnen bräucht ich alleine ca. 800MB Speicher nur für die Objektdaten.
Dank der Objektinstanzen muss ich die Objektdaten nur einmal laden und für jede Instanz einen Postionssowie Rotationsvektor.

2.8 UV-Texturing

Beim UV-Texturing oder wie es auch genannt wird UV-Mapping, wird ein komplexes 3D Objekt derart auseinander gefallt, dass es 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.

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, dass 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 es ein paar hundert sind. Betrachtet man nun mehrere diese Stufen fällt es dem Benutzer nicht auf, dass bei bestimmen Abständen von der Kamera eigentlich verschiedene Objekte gezeichnet werden.

2.10 Shadow Volumes

Als technisches Highlight soll in der finalen Version von EERT das Schattenverfahren Shadow Volumes implementiert werden. Im Prinzip ist dieses bereits in dieser Version implementiert, allerdings funktionert es noch nicht. Darum ist es nicht 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 ist 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. Es ist darauf zu achten was für eine Rekursionstiefe man wählt da die Anzahl der Knoten mit 8^n wächst.

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 das 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 sich immernoch im Frustum befinden werden alle Objekte dies 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.

3 Bedienung

3.1 Args Argumente

Für die Ausführung von EERT ist es erforderlich zwei Argumente an die Laufzeitumgebung zu übergeben. Das erste Argument gibt den Dateinamen der Szenenfile an. Die einzige zur Zeit unterstützte Szene ist SuzannTest6.eob. Das zweite Argument gibt an ob EERT im Fullscreen-Mode starten soll. Damit dies geschieht muss das Argument Fullscreen heißen. Wird das zweite Argument nicht übergeben wird eine Standart-Auflösung ausgewählt. Der empfohlende Aufruf von EERT sieht somit folgendermaßen auf.

java -jar Eert SuzannTest6.eob

3.2 EERT Steuerung

Nach einem Click in das aktive Fenster lässt sich die Kameraausrichtung durch klicken und gleichzeitiges Bewegen der Maus verändern. W A S D ändern die Position der Kamera. Zusätzlich kann man sich mit der Leertaste sowie der C-Taste nach Oben und Unten bewegen. Die Taste U aktivert den Fullscreen-Mode, allerdings wird hierbei die Szene von vorne gestartet. Es lassen sich Laufzeitinformationen über die I-Taste einblenden.