Programmierung einer Rendering Engine
Übungsabgabe

Fritz-Michael Gschwantner e0627004
Gabriele Hebart - e0626511

28. Februar 2007

1 Aufgabenstellung

Die Aufgabe bestand daraus drei Module einer Rendering Engine zu bauen, eines zur Generierung einer Geometrie, das nächste zur Veränderung dieser und das dritte Modul ist für das Rendern der Geometrie zuständig.

Das erste Modul soll einen Asteroidengürtel bestehend aus einer beliebigen Anzahl an Asteroiden sein. Die Asteroiden in diesem Gürtel unterscheiden sich in ihrer Größe als auch in der Geschwindigkeit in der sie sich bewegen.

Das Modul zur Veränderung der Geometrie soll View Frustum Culling mit Hilfe eines dynamischen Octrees implementieren, das heißt, dass in jedem Zeitschritt überprüft werden muss ob der Asteroid in einen anderen Node des Octrees verschoben werden muss.

Zum Schluss werden die tatsächlichen Asteroiden im Rendermodul gerendert.

Abbildung 1 veranschaulicht die Verbindung und den Aufbau der Module, sowie die Daten die von Modul zu Modul übergeben werden.

2 Algorithmen

2.1 Erzeugung der Geometrie

Die Asteroiden (Abbildung 2) basieren auf einer Kugelstruktur. Um die Vertexpositionen solch einer Kugel zu berechnen wurde ein Algorithmus aus dem Ogre Wiki verwendet[1]. Zuvor wird jedoch auch eine Art „fraktale Textur“ mit *Midpoint Displacement* errechnet. Hier wurde ein Algorithmus von Paul Martz verwendet: *Generating Random Fractal Terrain*[2]. Während der Berechnung der einzelnen Vertexpositionen werden auch die Texturkoordinaten

[1] `http://www.ogre3d.org/wiki/index.php/ManualSphereMeshes`
[2] `http://gameprogrammer.com/fractal.html`

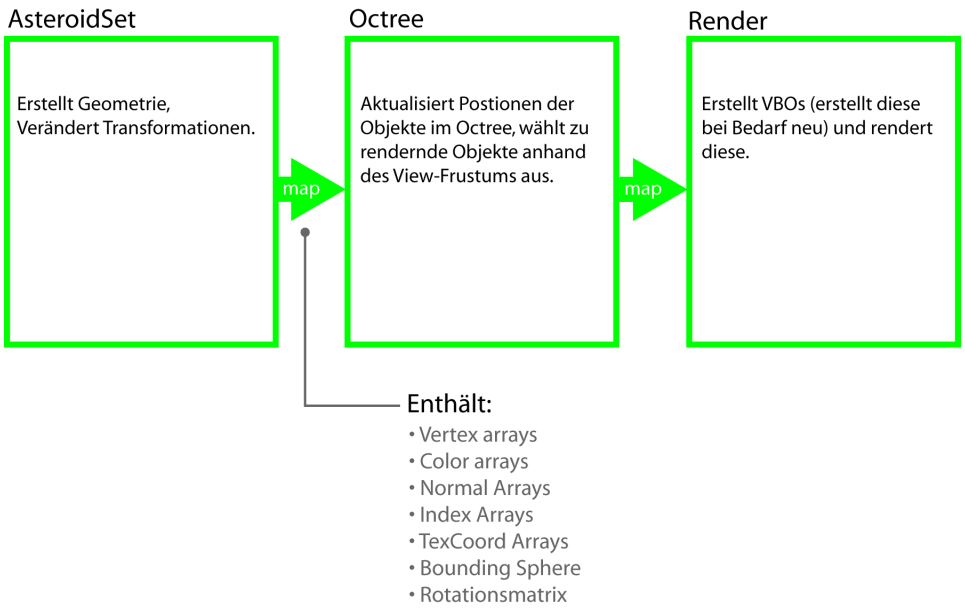


Abbildung 1: Prinzipieller Aufbau der Module und der Datenstruktur

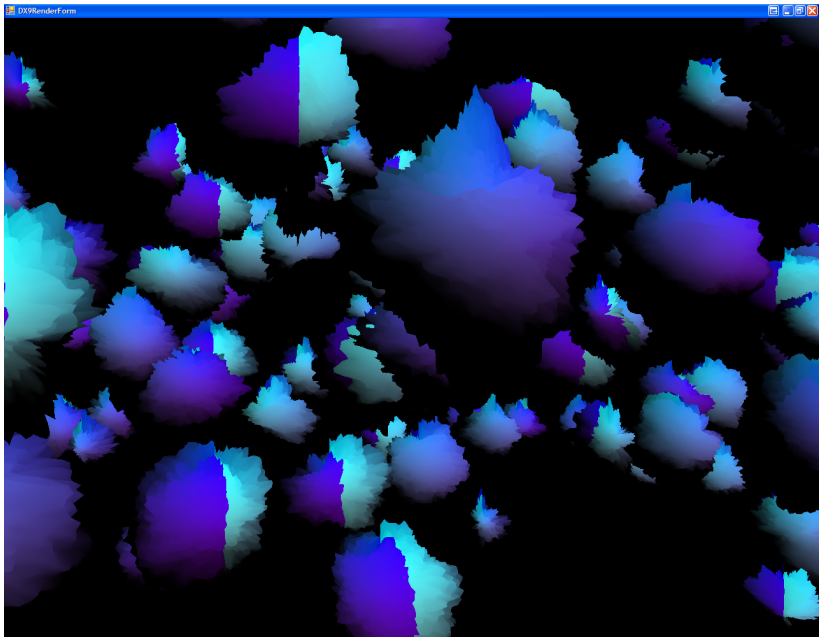


Abbildung 2: Durch Fraktale veränderte Kugeln: die Asteroiden.

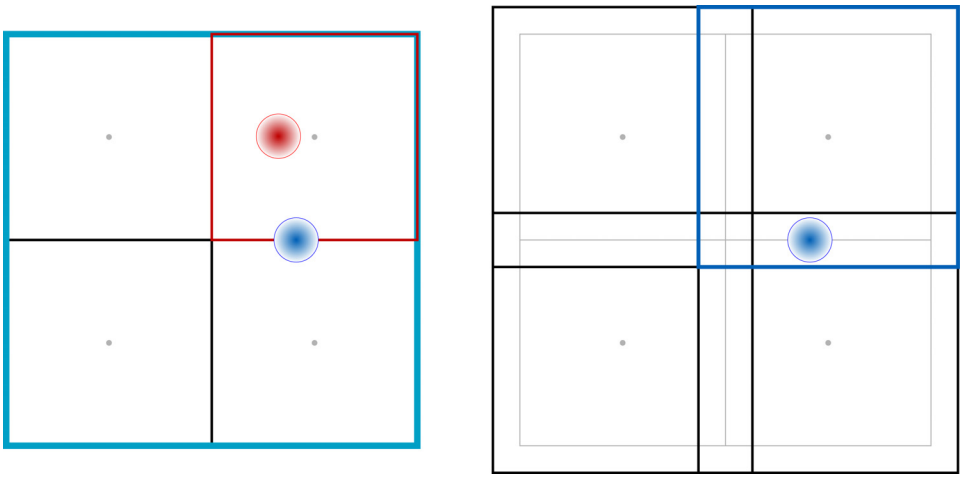


Abbildung 3: Einfache Illustration eines Quadtrees für dynamische Objekte mit und ohne Überlappung der Knoten

für die Kugel errechnet. Anhand dieser Texturkoordinaten werden anschließend Werte aus dem Fraktal genommen um damit die Vertexpositionen noch zusätzlich zu verändern.

2.2 Octree

Ein Octree erweitert das Prinzip des Quadtrees in die dritte Dimension. Anstatt eines Quadrats werden hier immer Würfel weiter unterteilt. Dies hat auch zur Folge, dass ein Octree viel mehr Knoten besitzt bzw. besitzen kann, da ein Würfel immer in acht weitere Regionen unterteilt werden kann.

2.2.1 Dynamic Octree

Der Octree soll dazu benutzt werden, Geometrie außerhalb des Sichtbereichs nicht an die Grafikkarte zu schicken. In unserem Fall handelt es sich dabei allerdings um bewegliche Objekte weshalb hier eine eher speziellere Version des Octrees eingesetzt werden muss. Das Problem hierbei ist einerseits, dass die Objekte im Octree immer wieder aktualisiert werden müssen, da sie durch ihre Bewegung in andere Knoten rutschen können. Und andererseits ist es in diesem Fall nicht sinnvoll die Geometrie aufzutrennen, um die einzelnen Vertices in den Knoten zu speichern. Dadurch entstehen Probleme in den Grenzregionen.

Wenn beispielsweise ein Objekt aus der Region seines Knotens austritt, kann das Objekt weder dem alten noch dem neuen Knoten eindeutig zugewiesen werden, solange sich das Objekt nicht vollständig in der Region eines Knotens befindet. Dies hat zur Folge, dass das Objekt zwangsläufig in dem bzw. einem der hierarchisch übergeordneten Knoten gespeichert werden

muss. Dies ist im linken Bild der Abbildung 3 illustruiert (der Einfachheit halber wird für die Darstellung nur ein Quadtree gezeigt). Während das rote Objekt noch mit seiner ganzen Bounding Sphere in den rechten oberen Subknoten passt, trifft dies für das blaue Objekt nicht mehr zu. Es muss daher im darüberliegenden Knoten gespeichert werden, dessen Bounding Box hier blau umrandet ist.

Dies führt dazu, dass sich viele Objekte nicht in kleineren Subknoten aufhalten können. Da die darüber liegenden Knoten natürlich auch größere Ausmaße haben, erhöht sich die Wahrscheinlichkeit dass diese, wenn auch nur teilweise, vom Sichtbereich geschnitten werden. Und dies führt wiederum dazu, dass viel mehr Objekte gerendert werden müssen, als eigentlich notwendig. Vor allem an den „Hauptachsen“ des Octrees ist dies sehr problematisch. Objekte die die Hauptachsen schneiden, müssen im obersten Knoten gespeichert werden, der quasi immer durch den Sichtbereich geschnitten wird.

Um bei einer Aktualisierung der Objekte im Octree nicht komplett vom Wurzelknoten aus durch traversieren zu müssen, werden in unserem Octree für jedes Objekt eine direkte Referenz auf den Knoten gespeichert. Somit kann schnell ermittelt werden, ob das Objekt sich noch in diesem Knoten befindet, und wenn nicht wird das Objekt an den darüber liegenden Knoten weitergeleitet.

2.2.2 Dynamic Overlapping Octree

Um das Problem mit den Regionsgrenzen zu umgehen gibt es die Möglichkeit, die Grenzen nicht sehr strikt zu halten, sondern mit den Nachbarknoten überlappen zu lassen. Dies ist im rechten Bild der Abbildung 3 illustriert. Dadurch, dass die Größe des Knotens, indem sich das blaue Objekt befindet, künstlich erweitert wurde, kann sich das Objekt immer noch in diesem Knoten befindet, obwohl es die ursprüngliche Grenze schon überschritten hat. Je nach Grad der Überlappung kann das Objekt dann entweder direkt in den nächsten Knoten zu rutschen, ohne zuerst in einem Elternknoten verweilen zu müssen.

2.2.3 Frustum Culling

Die einzelnen Nodes des Octress müssen dann auf ihre Sichtbarkeit überprüft werden, dazu wird im Renderermodul die Kameramatrix extrahiert und weiterverarbeitet. Aus dieser Matrix werden dann sechs verschieden Planes extrahiert (Lef, Right, Top, Bottom, Near, Far), dazu wurde Gil Gribbs und Klaus Hartmanns Artikel zu Hilfe genommen[3]. Die Nodes des Octrees werden dann darauf überprüft, ob sie vor, hinter oder auf den sechs verschiedenen Planes liegen. Liegen sie hinter allen Planes sind sie nicht sichtbar,

[3] `http://www2.ravensoft.com/users/ggribb/planeextraction.pdf`

liegen nur manche dahinter sind sie teilweise sichtbar etc. . Wird für einen Node festgestellt, dass dieser komplett im View Frustum liegt, werden seine Child Nodes nicht mehr auf Sichtbarkeit überprüft, da davon ausgegangen werden kann, dass diese auch sichtbar sind. Das Frustum Culling wurde mit Hilfe eines Artikels von Dion Picco[4] realisiert.

2.3 Rendering

Im Renderingmodul kontrolliert dann sowohl über die Geometrieerzeugung wie auch über deren Veränderung. Das Rendering-Modul übernimmt auch die Kameraeinstellungen und die Navigation.

Des weiteren übergibt der Renderer auch immer den aktuellen View Frustum, anhand dessen überprüft wird welche Nodes des Octrees sichtbar sind.

Zuerst werden sowohl der Octree als auch der Asteroidengürtel erzeugt, danach werden Frame für Frame die Module aneinandergekettet und die entsprechenden Maps weiter gegeben, je nachdem welche Module gerade eingeschaltet sind. Das Rendering Modul erhält dann eine Map mit den Objekten die Schluss endlich gerendert werden. Diese Liste wird nun abgearbeitet, zuerst wird überprüft ob schon ein Vertexbufferobject (Geometric Set) von dem Objekt verfügbar ist, ist es nicht, so wird es erzeugt und im Rendermodul mit der entsprechenden Id gespeichert, das VBO wird dann gerendert.

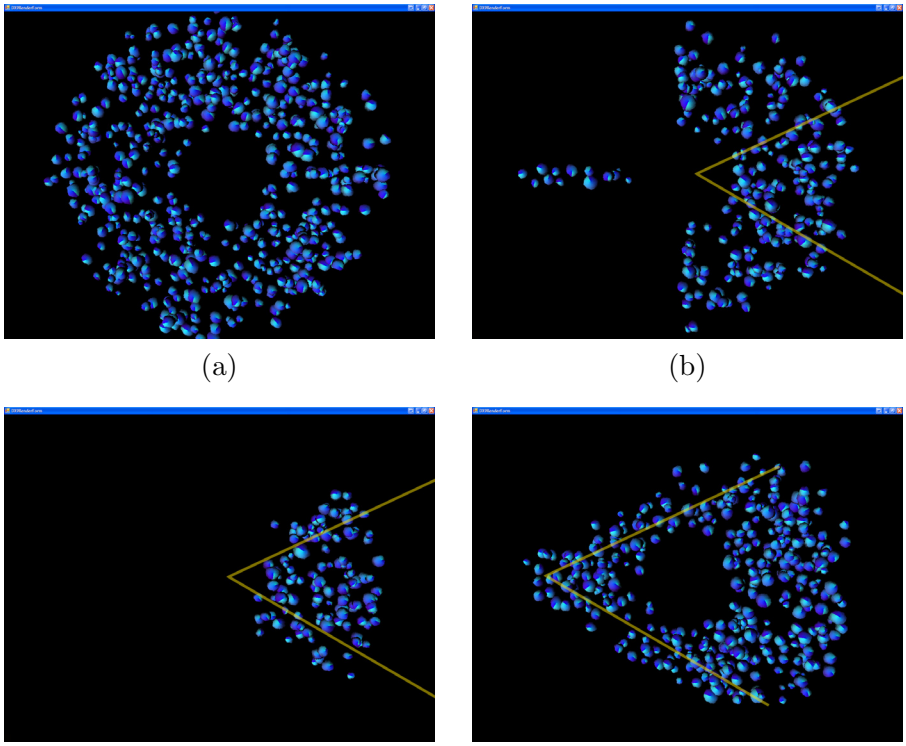
Die einzelnen Module können nach belieben ein- und ausgeschaltet werden, da das Ergebnis eines Moduls immer eine Map von Objekten ist, erhält der Renderer zum Schluss immer die gleiche Datenstruktur.

Der Asteroidengürtel wie auch der Octree können immer durch Tastatureingaben verändert werden, der Renderer bekommt diese übermittelt und verarbeitet sie entsprechend, und erzeugt dann zum Beispiel einen Asteroidengürtel mit einem verändertem Radius oder Anzahl an Asteroiden.

3 Analyse der Performance

In unserem Modul können sehr viele Parameter eingestellt werden. In diesem Testsetup gehen wir von 600 Asteroidenobjekten aus, bei einer Detailstufe von 64. Dies entspricht einer Anzahl von 2.535.000 Vertices. Folgende Tests sind in Abbildung 4 veranschaulicht, der jeweilige View Frustum ist mit gelben Linien (ungefähr) angedeutet. Die resultierenden *frames per second (fps)* Werte wurden auf einem Computer mit einem Athlon XP 2600+ und einer Radeon 9800 Pro Grafikkarte erreicht. In diesem Test befindet sich der Asteroidengürtel außerdem ausschließlich in der oberen Hälfte des Octrees, um Überschneidungen der Objekte mit den Hauptgrenzen etwas zu

[4] `http://www.flipcode.com/articles/article_frustumculling.shtml`



(a) (b)

(c) (d)

Abbildung 4: (a) Szene ohne Culling, (b) Culling mit Octree ohne overlapping, (c) Culling mit Octree mit overlapping, (d) Veränderte Position der Kamera, mit overlapping.

verringern.

- In Bild (a) wird die gesamte Szene ohne Verwendung des Octrees gerendert, also ohne View Frustum Culling. Ergebnis: maximal **10 fps**
- In Bild (b) wird der Octree mit einbezogen, allerdings mit abgeschaltetem Overlapping (Wert = 0.0). Man sieht hier deutlich, dass alle Objekte, die die „Hauptgrenzen“ durchqueren, immer gerendert werden, da diese Objekte sich zwischenzeitlich im obersten Knoten befinden. Links und rechts des View Frustum passiert ähnliches in den Unterknoten. Ergebnis: maximal **20 fps**
- In Bild (c) wurde der Overlapping Wert wieder erhöht (in diesem Fall auf 5.0, das heißt die Bounding Boxes der Knoten erhöht sich um den Wert 5.0). Dadurch erreicht man eine deutliche Reduzierung der Geometrie auf die Umgebung des Blickfeldes allein. Ergebnis: bis zu **30 fps**.
- Bild (d) zeigt noch eine Veränderung des Blickfeldes, sodass fast der gesamte Octree durchschnitten wird. Auch hier beschränkt sich die Geometrie nur auf die unmittelbare Umgebung, durch die Verwendung des overlapping Octrees. Da das Blickfeld aber nun einen sehr großen Teil des Asteroidengürtels umschließt, sinkt hier die Performance auf **14 fps**. Ohne overlapping schafft man hier wiederum nur **11 fps**.

Octrees sind für dynamische Objekte relativ aufwendig, die Prozessorlast ist hier deutlich höher, da Positionen der Objekte immer wieder neu in Betracht gezogen werden müssen (und das auch im komplett aufgebauten Octree), im Gegensatz zu statischen Objekten, wo sich die Position der Polygone im Octree nie ändert. Für den dynamischen Octree kann auch keine absolute Zahl (oder ähnliches) für die Performance genannt werden, da dies hier verstärkt vom Detailgrad des Octrees selbst und von der Anzahl der beweglichen Objekte abhängt.

Overlapping Octress können in manchen Situationen sogar eine schlechtere Performance liefern. Durch die Überlappung erhöht sich auch die Wahrscheinlichkeit, dass der View Frustum viel mehr Knoten schneidet, daher muss dieser Test auch viel öfters durchgeführt werden, welcher sehr kostspielig ist. Hier könnten noch weitere Maßnahmen getroffen werden, um gewisse Nodes schneller ausschließen zu können.