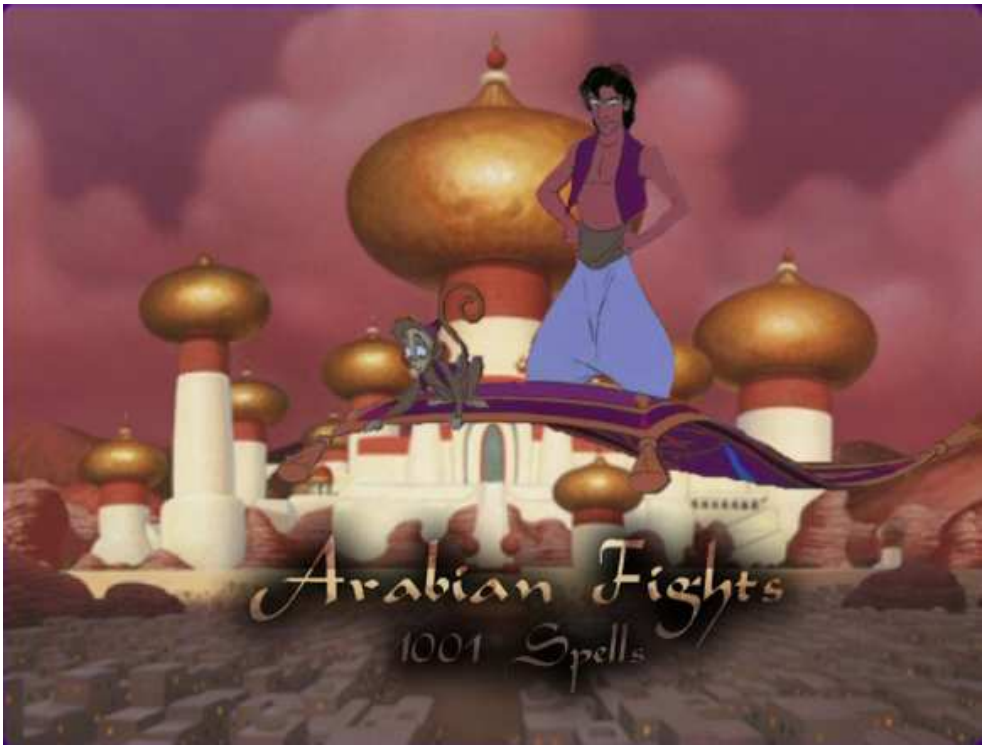


Computergrafik 2 Laborübung

Sommersemester 2008

Zweite Abgabe

Thomas Mühlbacher 532 0625075 e0625075@student.tuwien.ac.at
Clemens Arbesser 532 0625176 e0625176@student.tuwien.ac.at

Arabian Fights

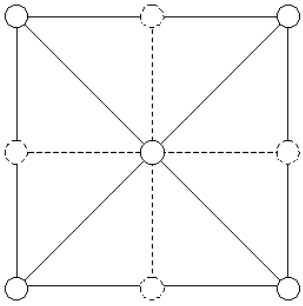
1001 Spells

In diesem Dokument geben wir einen Überblick über bereits implementierte Features und deren Funktionsweise. Außerdem werden wir noch geplante Features anführen, sowie den Status der Umsetzung des Spielkonzepts.

Bereits umgesetzte Features

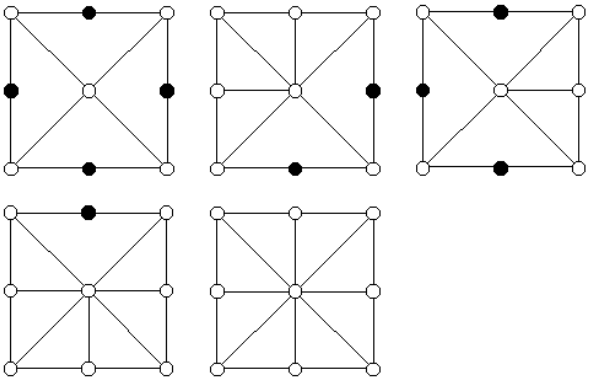
Continuous LOD Terrain

Das Terrain wird in einem Quadtree abgespeichert – das Besondere daran ist, dass die Squares nicht einheitliche Größen haben. Stattdessen wird versucht, in „geometriearmen“ Regionen die Squaregröße groß zu halten, während in geometriereicheren Regionen weiter unterteilt wird. Jeder Square hat jedoch dasselbe Aussehen:



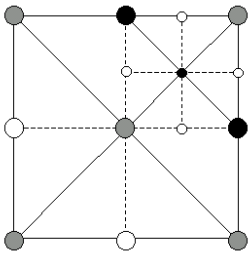
3x3 heightfield. Die gepunkteten Linien/Vertices sind optional

Die oben gezeigten optionalen Vertices können aktiviert/deaktiviert werden. Zum Rendern (dazu weiter unten mehr) eignet sich diese Darstellung sehr gut – ein einziger triangle-fan mit dem mittleren Punkt als Zentrum reicht, um dieses Heightfield darzustellen:



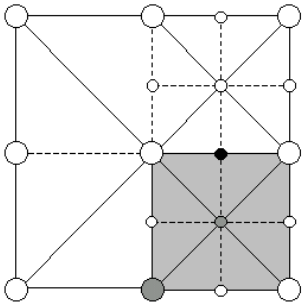
V*erschiedene Render-Szenarien. Deaktivierte Vertices sind schwarz dargestellt.*

Zusätzliche Geometrie erhält man entweder durch aktivieren der optionalen Vertices oder durch rekursives Aufteilen einer der 4 „subsquares“:



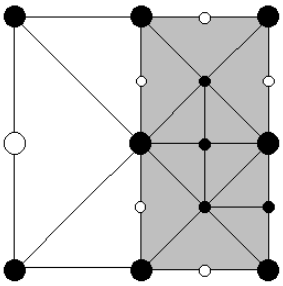
Wie man erkennt, weiß man beim Unterteilen des quadtree bereits, dass die grau markierten Vertices aktiviert sind, man muss also nur die schwarz markierten vertices des subsquare aktivieren.

Will man die optionalen Vertices eines subsquares aktivieren, muss man folgendes bedenken:



Der schwarz markierte Punkt des NE-subquads soll aktiviert werden. Man erkennt, dass auch der entsprechende Vertex des SE-subquads aktiviert werden muss.

Nachdem man nun weiss, welche Vertices aktiviert sind, kann der Quadtree auf eine einfache rekursive Weise gerendert werden. Zunächst werden alle Kinder des aktuellen Square gezeichnet, danach werden alle Vertices gezeichnet, die nicht Teil eines der Kinder waren:

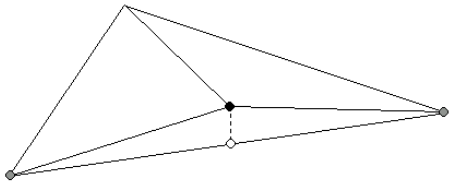


Render-Szenario einer 3x3 heightmap, die aktivierten Vertices sind schwarz.
Die grau unterlegten squares werden durch einen rekursiven Render-Aufruf gezeichnet, die weiss unterlegten hingegen vom ursprünglichen Aufruf.

Den wichtigsten Teil haben wir jedoch noch ausgelassen – die Entscheidung, ob ein Vertex aktiviert wird, und die Entscheidung, ob eine Unterteilung des quads erfolgen soll.

Vertices aktivieren / deaktivieren

Die Grundlage zu dieser Entscheidung bildet folgende Überlegung:



Vertex-Interpolations-Fehler. Falls ein Vertex aktiviert/deaktiviert wird, so ändert sich das Aussehen des Meshes. Die größte Änderung liegt dabei genau bei dem aktvierten Vertex, und ist durch die gestrichelte Linie dargestellt.Der Fehler ist dann der Unterschied zwischen der wahren Höhe (schwarz) und der Höhe des ursprünglichen Vertex (weiss). Man beachte, dass der ursprüngliche Vertex nur zwecks besserer Anschaulichkeit eingeführt wurde.

Nun können wir die Entscheidung über Aktivierung/Deaktivierung eines Vertex folgendermaßen treffen:

Wir verwenden eine Näherung der Distanz des Vertex vom Spieler (L1-norm), den errechneten Vertex-Interpolations-Fehler und einen konstanten Detail-Threshold:

L1 = max(abs(vertx - viewx), abs(verty - viewy), abs(vertz - viewz));
*enabled = error * Detail-Threshold < L1;*

Mit anderen Worten erklärt:
Für eine gegebene View-Distanz ist der größte Vertex-Interpolations-Fehler, der toleriert wird, z / DetailThreshold.

Liegt das relevante Polygon nahe zum Betrachter, ist L1 klein und es wird nur ein sehr kleiner Fehler toleriert -> Aktivierung eines optionalen Vertex zur Fehler-Verkleinerung.

Unterteilung des Quadtrees

Kann das Heightfield durch Aktivieren der optionalen Vertices nicht genau genug rekonstruiert werden, müssen die Squares im Quadtree feiner unterteilt werden.
Um festzustellen, ob unterteilt werden muss, führen wir einen „Box-Test“ durch:
Gegeben sei eine achsenparallele Box, die einen Teil des Terrains umschließe (zB. einen Quadtree-Square), sowie der größte Vertex-Interpolations-Fehler im Inneren dieser Box.
Ist dieser maximale Fehler noch zu groß, muss der Square feiner unterteilt werden (es wurden bereits alle optionalen Vertices aktiviert).

Dieser einfach erscheinende Test hat einige für den Programmierer sehr angenehme Eigenschaften. Einerseits reicht es, geometriearme bzw. nicht sichtbare Teile des Terrains sehr grob aufgelöst darzustellen, ohne dass die Qualität allzu stark beeinflusst wird. Zudem ist dieser Quadtree beinahe beliebig skalierbar, da die Performance nicht von der absoluten Größe des gesamten Levels abhängt, sondern nur von der Größe der geometriereichen, nahe zum Betrachter liegenden Teile und von der globalen Detail-Threshold-Konstante.

Referenzen

Wir möchten uns an dieser Stelle herzlich bei Thatcher Ulrich (lead programmer for Slingshot Game Technology) bedanken, der uns die Details dieses Algorithmus erklärte und uns auch Tipps für die Implementierung gab. Für die Erstellung dieses Algorithmus gab er folgende Ressourcen an:

[1] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust and Gregory A. Turner. "Real-Time, Continuous Level of Detail Rendering of Height Fields". In *SIGGRAPH 96 Conference Proceedings*, pp. 109-118, Aug 1996.

[2] Mark Duchaineau, Murray Wolinski, David E. Sigeti, Mark C. Miller, Charles Aldrich and Mark B. Mineev-Weinstein. "ROAMing Terrain: Real-time, Optimally Adapting Meshes." *Proceedings of the Conference on Visualization '97*, pp. 81-88, Oct 1997.

[3] Stefan Röttger, Wolfgang Heidrich, Philipp Slusallek, Hans-Peter Seidel. *Real-Time Generation of Continuous Levels of Detail for Height Fields. Technical Report* 13/1997, Universität Erlangen-Nürnberg.

[4] Seumas McNally. http://www.LongbowDigitalArts.com/seumas/progbintri.html . This is a good practical introduction to Binary Triangle Trees from [2]. Also see http://www.TreadMarks.com , a game which uses methods from [2].

[5] Ben Discoe, http://www.vterrain.org . This web site is an excellent survey of algorithms, implementations, tools and techniques related to terrain rendering.

Diese gehen jedoch einerseits über den Umfang dieser Abgabe, und andererseits über den Umfang der Laborübung hinaus.

Frei bewegliche Kamera

Wie in der ersten Abgabe geschildert kann sich der Spieler frei bewegen. Wir sind derzeit noch am Testen, ob eine Rotation um die z-Achse der Kamera (dh. Neigung nach links/rechts) notwendig ist, und haben sie in der derzeitigen Version ausgeschalten.
Sämtliche Rotationen erfolgen mit einer vereinfachten Matrixmultiplikation, wobei jedoch nicht die Viewmatrix selbst rotiert wird, sondern nur der Kamera-Vektor. Nach dessen Rotation wird die neue Perspektive einfach mit einem gluLookAt - Befehl gesetzt. Dies hat den Vorteil, auf einfache Weise jederzeit auf alle Parameter der Kamera zugreifen zu können.

Skybox

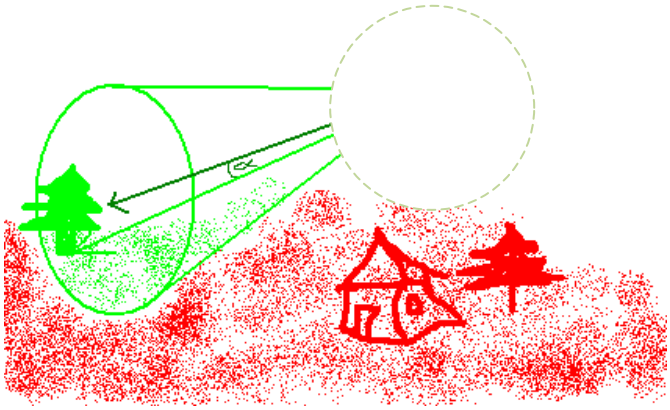
Das Level wird durch eine Skybox abgerundet, die derzeit noch im immediate-Mode gerendert wird – dies sollte performance - technisch jedoch kein großes Problem darstellen. Wir sind noch auf der Suche nach orientalisch - anmutenden Skyboxes für die finale Version.

Collision Detection

Bereits im Spiel enthalten ist eine einfache Kollisions-Abfrage mit Bounding-Spheres (Schuss<->Spieler/Gegner) und mit dem Terrain (Spieler/Objekt<->Terrain). Dabei können Objekte, je nach Typ, unterschiedlich auf eine Kollision reagieren. Für die finale Abgabe ist eine Interpolation der Terrain-Höhen vorgesehen, damit der Spieler nicht über Kanten des Terrains fliegen muss.

Frustum Culling

Zur Performanceverbesserung haben wir ein einfaches Kegel-Frustum-Culling implementiert, welches nicht sichtbares Terrain und Objekte schon frühzeitig aus der Berechnungspipeline entfernt:



In einfachen Worten: Alle Objekte, für die der Vektor zum Betrachter mit dem Kamera-View-Vektor einen Winkel einschließt, der kleiner als der halbe Öffnungswinkel des (grünen) Kegels ist, sind sichtbar, alle anderen nicht.
Ein kleines Problem stellen damit Objekte dar, die nahe aber seitlich zum Betrachter liegen und somit diesen Winkel überschreiten, daher gänzlich verschwinden. Daher haben wir alle Objekte, die in einer gewissen Kugel um den Betrachter liegen, immer gezeichnet (gestrichelte Kugel).

HUD

Am oberen Bildschirmrand befindet sich das HUD, das aus einer Gesundheits-Anzeige, einer Mana-Anzeige, den aktuellen Zaubersprüchen und einer Minimap besteht. Die Anzeigen sind zur Zeit noch nicht mit dem Spiel synchronisiert, wird der Spieler getroffen, wird dies nur in der Konsole ausgegeben.

Minimap

Die Minimap zeigt die Umgebung des Spielers, sowie Dekor-Objekte (grün) und Gegner (schwarz). Die Position des Spielers wird durch ein Kreuz markiert. Der Kartenausschnitt rotiert zudem mit dem Spieler mit, sodass die Sichtrichtung des Spielers mit der Karte übereinstimmt.

FBX-Models

Bereits in dieser Version enthalten ist ein FBX-Loader, der in der Lage ist, sowohl statische als auch animierte Modelle zu laden. Diese werden derzeit im Immediate-Mode gerendert. Als Vorlage diente einerseits das offizielle Beispiel des FBX-SDK's, und andererseits ein Loader, den uns freundlicherweise ein Absolvent dieser LVA zur Verfügung gestellt hat.

Gegner und Decor-Objekte

Zur Demonstration wurden einige Dekor-Objekte und Gegner gezeichnet, die mit dem Spieler interagieren – dh. mit diesem kämpfen können.
Das Rendering erfolgt durch die Berechnung eines Szenegraphen, der Zeiger auf alle derzeit sichtbaren Objekte enthält. Ob ein Objekt sichtbar ist, entscheidet das oben angeführte Frustum-Culling. Die 3D-Modelle werden mit Maya erstellt und umfassen Gebäude, Vegetation und Gegner. Unter Umständen wird auch der Spieler selbst modelliert.

Geplante Features

Models laden

Spielkonzept fertig umsetzen

Leveldesign

Effekte:

- Schatten
- Partikeleffekte (Zaubersprüche, evtl. Gras)
- Animierte Gegner
- Wasser

Bekämpfen der Gegner mit Zaubersprüchen

Ziel des Spiels wird die Vernichtung aller (bzw. eines gewissen Prozentsatzes) Gegner des jeweiligen Levels sein. Dem Spieler stehen dafür 9 verschiedene Zaubersprüche zur Verfügung:

-**Feuerball:** Der einfachste Angriffszauber. Geringer Schaden, hohe Reichweite
-**Feuerball-Salve :** Viele Feuerbälle in schneller Abfolge. Mittlerer Schaden, hohe Reichweite
-**Blitz:** Starker Angriffszauber, wird nicht von Abprall reflektiert. Mittlere Reichweite
-**Meteor:** Ultimativer Projektilangriff, wird jedoch von Abprall reflektiert. Hohe Reichweite

-**Geschwindigkeit:** Erhöht die Fluggeschwindigkeit um das 2-3 fache
-**Mana – Besitz:** Nimmt neutrales Mana (siehe weiter unten) in Besitz
-**Heilung:** Heilt den Spieler
-**Abprall:** Lässt für geringe Zeit Projektile am Spieler abprallen.
-**Kreatur beschwören:** Der Spieler beschwört eine Kreatur, die für eine gewisse Zeit an seiner Seite eigenständig Gegner bekämpft und somit zur Unterstützung im Kampf dient.

Zauber unterschieden sich in folgenden Parametern:

Modell: Das Aussehen des Geschosses
Manakosten: Die Manamenge, die bei Benutzung vom Manavorrat des Spielers abgezogenen wird
Reichweite: Maximale Flugweite (0 falls spielerbezogener Zauberspruch)
VerfügbarAb: Manamenge, die der Spieler gesammelt haben muss, bis ihm der Zauberspruch zur Verfügung steht.

Ausserdem ist geplant, den Spieler in der finalen Version beim Zielen zu unterstützen. Sollte dies aus Zeitgründen nicht möglich sein, wird ein Fadenkreuz gezeichnet werden. Zu beachten ist weiters, dass dem Spieler zu Anfang nicht alle Zaubersprüche zur Verfügung stehen werden – diese muss er im Verlauf des Spieles sammeln.

Mana-Sammeln

Gegner hinterlassen nach ihrem Ableben goldene Kugeln, die ihre magische Energie repräsentieren. Die goldene Farbe bedeutet, dass sie noch nicht vom Spieler in Besitz genommen wurden. Mit dem Zauberspruch „Mana-Besitz“ kann der Spieler die Manakugeln auf seine Spielerfarbe umfärben und sie so seinem Manavorrat hinzufügen. Durch Erhöhen seines Manavorrates erhält der Spieler Zugriff auf stärkere Zaubersprüche. Das Aktivieren von Zaubersprüchen kostet den Spieler Mana, das ihm von seinem aktuellen Vorrat abgezogen wird. Dieser lädt sich mit der Zeit wieder auf (schneller, falls sich der Spieler in der Nähe seines Ausgangspunktes befindet).