Computergraphik 2 LU

SS 2007

Name	Matrnr	Kennzahl	E-Mail
Martin Wagner	0425090	532	martinwagner85@gmx.at
Katharina-Anna Wendelin	0425160	532	katharina.wendelin@gmail.com

Inhaltsverzeichnis

DragonWarrioR

Ziel des Spiels

In dem Spiel DragonWarrioR nimmt man als Spieler die Position eines Ritters in Ego-Shooter Perspektive ein. Das Ziel des Spiels ist es, den Weg durch das Burggartenlabyrinth zu finden und die Burg zu erreichen. Dabei muss man gegen Drachen kämpfen, die einem den Weg versperren.

Gameplay

Durch Collision Detection wird verhindert, dass man sich seinen Weg durch die Hecke bahnt. Somit muss man sich auf den Weg durch den Burggarten machen. Auch kann man nicht an den Drachen vorbeigehen, somit muss man, um sein Ziel zu erreichen, die Drachen besiegen.

Nichttriviale Objekte

In unserem Spiel gibt es einige Models, wobei es sowohl konvexe (z.B.: Kessel) als auch nichtkonvexe Objekte (z.B.: Schild, das nach innen gewölbt ist) gibt.

Models

Unsere Models (Drache, Schwert, Schild, Kessel, Burg) wurden mit 3d Studio Max gezeichnet, mit Milkshape animiert, als Quake III Arena MD3-File exportiert und dann mit einem selbstgeschriebenen Modelloader ins Spiel geladen. Dieser befindet sich in der Klasse Model und wurde mit Hilfe der Beschreibung der md3 Datenstruktur (siehe [Q1]) erstellt. Zusätzlich wurde noch ein Prinzessinenmodel aus dem Internet übernommen. Dabei handelt es sich um einen anatomisch korrekten Frauenkörper, auf den ein Kleid mit Milkshape gesetzt wurde. Das Frauenmodel und das Kleid wurden von diesen Quellen übernommen [Q5], [Q6].
Zu den animierten Objekten zählen:

- Drache
  3 Animationen: Gehen, Flügelausbreiten, Feuerspucken (Bewegen des Kiefers und Rotation des Kopfes um dem Ritter zu folgen)
- Schwert
  Schlaganimation (Klicken mit der rechten Maustaste)
- Schild
  Bewegen des Schilds vor den Körper (Klicken mit der rechten Maustaste)
- Burg
  Öffnen der Burgtore, sobald man sich der Burg nähert

Für die Animationen wurde mit Milkshape (z.B.: beim Drachen) ein Skelett aus Joints erstellt welches dann Frame für Frame bewegt wurde.

Beschleunigung der Sichtbarkeitsberechnung

Um für Performance-Gewinn zu sorgen, wurde Frustum Culling implementiert.

Transparenz-Effekte

Um die Lebensanzeige besser sichtbar zu machen, wurde ein schwarz-transparentes Rechteck unter die Anzeige gegeben. Auch die Partikel beim

Feuer und das Wasser haben einen transparenten Anteil, was alphageblendet wird.

Texturen
Die Texturen wurden aus dem Computerspiel Tactical Ops: Assault on Terror(Version 3.50) entnommen.

Umgebung

Die Koordinaten von Hecken, Wege, Gräser und Mauern werden aus einem Textfile ausgelesen. Die genaue Position der Faces wurde auf Grund einer mit der Hand gezeichneten Skizze erstellt.

Features des Spiels

- **HP-Leiste**
  In unserem Spiel gibt es einen Lebensbalken, der je nach verlorenen Lebenspunkten seine Farbe ändert. Bei 0 Lebenspunkten ist das Spiel verloren und das Programm wird beendet. Um die Sichtbarkeit auf allen Hintergründen gewährleisten zu können, wurde der Lebensbalken über einen grau-transparenten Balken gelegt.
- **Kessel**
  Überall im Spiel versteckt gibt es Zauberkessel, die ihre Farbe verändern. Nähert man sich einem Kessel, so kann man seine Lebenspunkte durch Drücken der Taste E auffüllen.
- **Burg**
  Das Ziel des Spiels ist es die Burg am Ende der Map zu erreichen. Nähert man sich dieser Burg, so öffnen sich die Tore und man kann eintreten. Somit ist das Spiel beendet.
- **Drache**
  Der Burggarten ist übersäht mit bösen Drachen, die den Ritter daran hindern wollen, seinen Weg zur Burg fortzusetzen. Sobald ein Drache den Ritter entdeckt, rennt er auf diesen zu, breitet seine Flügel aus, um ihn am Vorbeigehen zu hindern und beginnt Feuer zu spucken. Es ist nicht möglich, an dem Drachen vorbeizukommen, ohne diesen zu besiegen und ohne von diesem bemerkt zu werden, denn versucht man, sich von hinten an den Drachen heranzuschleichen, dreht sich dieser um und baut sich wieder vor dem Ritter auf und versperrt ihm den Weg. Schlägt man auf den Drachen mit dem Schwert ein, so verändert er die Farbe, je nachdem, wie oft er getroffen wurde. Sobald man den letzen, für den Drachen tödlichen Schlag, setzt, verschwindet der Drache.
- **Schwerthieb**
  Je länger der Schlag dauert, desto mehr Leben wird dem Drachen abgezogen. Außerdem gibt es noch eine Zufallskomponente, die zusätzlich den Drachen pro Schwerthieb Punkte abzieht.
- **Feuer**
  Bei dem Feuer handelt es sich um ein einfaches Partikelsystem, dass aus texturierten Quads besteht. Der Drache spuckt immer dann Feuer, wenn der Ritter in seiner Nähe ist. Dabei wird der Kopf des Drachen bewegt, um dem Ritter bei der Bewegung zu folgen. Wie lange und wann der Drache Feuer spuckt, hängt von einer Zufallsvariable ab.

- **Gehbewegung**
  Um die Fortbewegung des Ritters in 1st Person Perspektive realistischer zu gestalten, wurde eine Wippbewegung eingebaut. Dabei handelt es sich um eine periodische halbe Sinusschwingung, deren Amplitude zu y dazuaddiert wird. Diese Funktion ist abhängig von der Schrittdauer und ist in der Vorwärts, Rückwärts und Seitwärtsbewegung zu finden.
- **Frames per Second**
  Durch Drücken der Taste F2 werden die Frames per Second rechts oben am Bildschirm angezeigt.
- **Wireframe-Modus**
  Durch Drücken der Taste F3 kommt man in den Wireframe-Modus. Durch erneutes Drücken der F3 Taste verlässt man diesen Modus.
- **Texturqualität**
  Durch Drücken der Taste F4 kann man die Texturqualität bestimmen, wobei man hier zwischen nearest-neighbor und bilinear wählen kann.
- **Mip-Mapping**
  Durch Drücken der Taste F5 kann man Mip-Mapping ein- und ausschalten.
- **Pause**
  Durch Drücken der Taste P kann man das Spiel pausieren und durch erneutes Drücken das Spiel wieder fortsetzen.
- **View-Frustum Culling**
  Durch Drücken der Taste F8 kann Frustum Culling ein- und ausgeschaltet werden.
- **Transparenz**
  Durch Drücken der Taste F9 kann die Transparenz ein- und ausgeschaltet werden.

Spezialeffekte

Partikelsystem

Um es dem Drachen möglich zu machen, Feuer zu spucken, haben wir in unserem Spiel ein Partikelsystem eingeführt.
Als Grundlage haben wir uns an dieser Quelle orientiert [Q7], jedoch verwenden wir als Datenstruktur Arrays und berücksichtigen keine äußeren Einflüsse, da dies bei einem Feuerstrahl nicht wichtig ist.

Jeder unserer Drachen besitzt einen Emitter. Je nach Position des Kopfes wird dieser Emitter positioniert. Dieser Emitter erzeugt neue Partikel, sobald man sich in die Nähe des Drachen begibt.
Es kommen hierbei alpha-geblendete texturierte Quads zum Einsatz, die dem Betrachter zugewendet sind (sceen-aligned). Die Textur auf den Quads dient dazu, um „abrupten Rechtecke“, sondern gleichmäßig transparent verlaufende runde Partikel zu erzeugen.
Die Partikeltextur wurde mit Hilfe von Matlab erstellt und ist in groben Zügen eine cos-Funktion vom Abstand Texel zum Mittelpunkt in der Textur. Somit ist das Innere weiß, während das Äußere schwarz ist.

Per Pixel Lighting

Wir haben in unser Spiel Phong Shading eingebaut, um Oberflächen wie z.B.: das Schild oder den Kessel realistischer darzustellen. Für das Erstellen dieses Effekts wurden folgende Quellen verwendet:

- [Q2] für die mathematischen Grundlagen
- [Q3] für den Aufbau der Pipeline
- [Q4] für die richtige Verwendung der GLSL

Zusätzlich wurde die Library glew eingebunden.
Den Source Code für Vertex- und Fragmentshader, haben wir in zwei Textfiles geschrieben, welche wir in unserem „Data“ Ordner untergebracht haben.
Unser Vertexshader ist folgendermaßen aufgebaut:
Zuerst wird die Normale des Punktes und seine Position in den viewspace gebracht. Zusätzlich werden noch die Texturkoordinaten und der transformierte Punkt ausgegeben.
Im Fragmentshader findet die Berechnung für das Phong Shading folgendermaßen statt:
Die Normale, die aus dem Vertexshader übergeben wird, wird normalisiert.
Um die für das Phong Shading benötigte diffuse und spekulare Beleuchtung zu erhalten, verwenden wir diese Formel:

$$I = k_a I_a + k_d I_l (N*L) + k_s I_l (N*H)^{ns}$$

N..... normalisierte Normalvektor
L……. … ……Vektor, der zur Punktlichtquelle zeigt
V……. ……Vektor, der zum Betrachter zeigt
H……..halfvector H=(L+V)/(|L+V|)
k_a, k_d, k_s…….Reflexionskoeffizienten
I_a,I_l…… Intensitäten

Zusätzlich zu der Fragmentfarbe geben wir die Farbe und den Alpha-Wert der Texturkoordinate aus.
Der Shader selbst wird in unserem Hauptprogramm erstellt. Vertex- und Fragmentshader wird der Sourcecode, der aus den Textfiles ausgelesen wird, übergeben und anschließend werden die beiden kompiliert.
Danach wird ein Shaderprogramm erstellt, dem die Shader hinzugefügt werden. Das Programm wird nun verlinkt und mit glUseProgram(program) ausgeführt.

Wasser

Bei der Implementierung der dynamischen Wellengeometrie und der Texturanimation haben wir keine Referenzen zur Hilfe genommen.

In unserem Burggraben befindet sich Wasser, das sich wellenförmig von rechts nach links (Richtung Burg sehend) bewegt.
Es werden stets nacheinander Wellen erzeugt. Diese zeichnen sich durch eine unterschiedliche (zufällige) Wellenlänge und Höhe aus. Sobald Platz für eine neue Welle ist, wird diese erzeugt, und eine Welle, die aus dem sichtbaren Bereich verschwunden ist, wird gelöscht. Als Datenstruktur verwenden wir hierbei einen Vector.

Dieser Vector wird jedes Frame aktualisiert, wobei hierbei die Beginnkoordinaten der einzelnen Welle mit dem Zeitabstand zum vorherigen Frame transliert werden.

Die Wellenstruktur wird in einem Array abgespeichert, wobei jedes Element im Array den z-Wert der x.-Zeile erhält. Die Welle ist entlang der y-Achse gleich hoch. (x,y,z sind die 3 Welt-Koordinaten und x der Index des Arrays.) Diese Speicherung erfolgt, indem der Wellen-Vector durchiteriert wird, also alle aktuellen Wellen durchiteriert werden, und die Höhe an der Stelle (x) durch die Amplitude der Cosinus-Funktion mal der Wellenhöhe ermittelt wird. Am Rand des Grabens (x-Achse) wird der Wert im Array linear verringert, um ein abgehaktes Szenario zu verhindern.

Auf dieser dynamischen Wellengeometrie befinden sich 2 Texturschichten, wobei 1 in die x-Richtung und die andere in die y-Richtung in unterschiedlicher Geschwindigkeit voneinander, jeweils abhängig vom Zeitunterschied zwischen den Frames, translieren.

Beleuchtung

In unserem Spiel gibt es eine diffuse Punktlichtquelle, die sich über der Map befindet und in negative x-Richtung strahlt. Zusätzlich gibt es noch ein globales ambientes Licht.

Walk-through

Die folgende Skizze stellt den schnellsten weg da, um zum Ziel des Spiels zu gelangen. Möchte man sich einen Kessel ansehen nimmt man am besten den Weg zum Kessel in der linken unteren Ecke. Um sich die Wellenanimation anzusehen, braucht man nur diesem Walk-through zu folgen. Per pixel lighting kann man am besten am Schild erkennen. Das Partikelsystem wird sichtbar, sobald man auf den ersten Drachen trifft. Um schneller durch das Spiel zu kommen gibt es die beiden Cheat-Tasten „h“, um den Lebensbalken aufzufüllen und „k“ um einen Drachen in der näheren Umgebung zu töten.

Im Bild kennzeichnet D einen Drachen, K einen Kessel, S den Startpunkt und die blaue Linie stellt den kürzesten Weg (mit geringstem Widerstand) dar.

Steuerung

Tastatur

- W: Ritter läuft geradeaus
- A: Ritter läuft seitlich nach links
- D: Ritter läuft seitlich nach rechts
- S: Ritter läuft nach hinten
- E: Aktionstaste (in der nähe des Kessels: Lebenspunkte auffüllen)
- P: Pause/Pause aufheben
- K: Drache in näherer Umgebung töten
- H: Lebensbalken auffüllen
- F2: Frames per Second
- F3: Wireframe-Modus
- F4: Texturqualität ändern (nearest neighbor/ bilinear)
- F5: Mip-Mapping ein/aus
- F8 View frustum culling ein/aus
- F9 Transparenz ein/aus
- Esc: Spiel beenden

Nicht implementiert wurden Immidiate Mode / VBO / Display Lists.

Maus

- Maus nach links: Kamera bewegt sich gegen den Uhrzeigersinn
- Maus nach rechts Kamera bewegt sich im Uhrzeigersinn
- Maus hinauf: Kamera bewegt sich max. 25° nach oben
- Maus hinunter: Kamera bewegt sich max. 25° nach unten
- Linksklick: Schwerthieb
- Rechtsklick: Schild ziehen

Libraries

Für unser Spiel haben wir die von der CG2 Homepage vorgeschlagene GLUT-Library verwendet und zusätzlich die Library glew für die Implementation des Phong Shadings.

Quellen

[Q1] www.icculus.org/homepages/phaethon/q3a/formats/md3format.html

[Q2] Hearn, Baker: Computer Graphics with OpenGL, third edition

[Q3] http://www.lighthouse3d.com/opengl/glsl/index.php?pipeline

[Q4] http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf

[Q5] http://artist-3d.com/free_3d_models/dnm/model_disp.php?uid=161&ad=02anatomy_design.php&count=count

[Q6] http://artist-3d.com/free_3d_models/dnm/model_disp.php?uid=730&ad=02anatomy_design.php&count=count

[Q7] http://www.double.co.nz/dust/col0798.pdf