

MorphingBlobs

Änderungen zur 2. Abgabe sind in blauer Farbe hervorgehoben.

Bug Fixes

- Die Auswahl von Einheiten durch Ziehen eines Rahmens funktioniert nun ordnungsgemäß und Präzise für Boden- und Lufteinheiten.
- Bei der Auswahl von Einheiten durch Ziehen eines Rahmens werden gegnerische Einheiten nicht mehr mitselektiert.
- Ein Bug im KI-Skript, welcher dafür gesorgt hat, dass die Computer zu schnell Kampfeinheiten bauen und angreifen, wurde gefixt

Anforderungen (3. Abgabe)

Gameplay

Gameplay und Steuerung sind vollständig implementiert. Auch die Kameragrenzen wurden angepasst, sodass der Rand des Spielfeldes nicht mehr „plötzlich fertig“ ist.

Nichttriviale Objekte / Animierte Objekte

Wie bereits bei der 2. Abgabe treten die Blobs hier als nichttriviale Objekte auf. Diese wurden noch weiter verbessert. Mehr Details unter „Techniken Features & Effekte“.

Beschleunigung der Sichtbarkeitsberechnung

Wie bereits bei der 2. Abgabe wird für Terrain und Blobs Viewfrustum-Culling benutzt, um nicht sichtbare Objekte von vornherein auszuschließen. Weiters werden alle Einheiten und Geschosse gegen den Fog of War gecullt.

Transparenzeffekte

Die Blobs werden leicht transparent gerendert.

Weiters kommen beim Angriff der Einheiten Blendingeffekte zur Anwendung.

Experimentieren mit OpenGL

Wie verlangt, werden verschiedene Möglichkeiten von OpenGL verwendet, die meisten können mittels der Funktionstasten umgeschaltet werden:

- Framerate & Statusanzeige der Funktionstasten (F2)
- Wireframemodus (F3)
- Texturfilter und Mipmapping (F4, F5, F7)
- VBOs mit VAOs werden sowohl vom Terrain als auch von den Blobs verwendet (nicht umschaltbar)
- Frustum-Culling (F8)
- Alpha-Blending der Blobs (F9)
- F10 bleibt nach wie vor dem Cheat-Modus vorbehalten ;-)

Techniken & Features & Effekte

Terrain:

- Die Höhenwerte des Terrains werden prozedural generiert, **zusätzlich wird eine Heightmaptextur aus einer Datei geladen und hinzuaddiert, um dem Spielfeld mehr Detailreichtum zu verleihen und den Spielfeldrand abzuschrägen.**
- Die Verteilung der 4 Texturen auf dem Spielfeld wird prozedural generiert
- Terrain ist aufgeteilt in kleinere „Patches“ welche einzeln gegen das Viewfrustum gecullt werden
- Der Shader mischt die 4 Texturen entsprechend einer Blendingtextur. Außerdem werden noch 4 weitere Texturen für ein simples Normalmapping verwendet
- **Es gibt nun 2 Texturesets (Rocky Grasslands & Crazy Asteroid), welche jeweils eigene Methoden zur Texturverteilung haben.**
- **Für beide Texturesets wird nun auch eine passende Skybox gerendert, welche man sehen kann wenn man den Kamerawinkel mittels mittlerer Maustaste verschiebt.**
- **Für den Spieler nicht einsehbare Bereiche werden mit einem reduzierten Diffuse-Wert gerendert, sodass sich eine dynamischere Beleuchtung ergibt, wenn sich die Einheiten bewegen-**

Blobs:

- Alle Blobs werden aus derselben Grundgeometrie generiert, nämlich einem horizontal und vertikal in eine bestimmte Anzahl von Segmenten unterteilte Plane.
- **Die Anzahl der Segmente eines gerenderten Blob wird in Abhängigkeit von der Distanz dynamisch erhöht/vermindert, wodurch sich variables LOD ergibt (Mesh-Auflösung von 8x8, 9x9, 10x10,, 31x31, 32x32 → maximum)**
- Der Vertexshader benutzt die x/y-Koordinate als Input für eine mathematische Funktion, welche die eigentliche Geometrie ~~und Normalvektoren~~ der Blobs generiert.
- **Die Normalvektoren werden nun im Fragmentshader generiert. Dies verhindert das „Beleuchtungs-popping“ beim Umschalten zwischen den LOD-Stufen (ein Popping der Geometrie ist auf Grund der ohnehin hohen Auflösung (fast) nicht erkennbar). Außerdem ist es so möglich, weitere Details auf die Oberfläche aufzubringen. Zum einen kommen somit automatisch mehr Details auf die Oberfläche, da die Funktionen, welche den Normalvektor generieren, eine wesentlich feinere „Auflösung“ haben als die Geometrie. Weiters wurde noch eine zusätzliche noise-Funktion erstellt, welche zum ermittelten Normalvektor hinzuaddiert wird und einen leichten, wabernden „Welleneffekt“ aufbringt (somit eine Art prozedurales Bump/Normalmapping).**
- Die mathematischen Funktionen verschiedener Modelle können interpoliert werden, dadurch ist es möglich, von einem Model stufenlos zu einem anderen Modell zu morphen
- Front- und Backface werden separat gerendert, um Bildfehler auf Grund der Transparenz zu verringern. **Außerdem wird vor dem Rendern der Blobs ein Depth-Sorting für diese durchgeführt, damit die transparenten Blobs korrekt gegeneinander geblendet werden.**
- Blobs werden einzeln gegen das Viewfrustum gecullt

Partikel- und Lichteffekte:

- Ein „Laserartiger“ Effekt wird für das Abbauen von Kristallen sowie für einige Angriffseffekte benutzt. Erreicht wird dieser Effekt durch mehrere Quads, welche mittels additivem Blending und mit zusätzlicher Texturanimation gerendert werden
- Die meisten Angriffseffekte werden durch viele Partikel (ebenfalls Quads mit additivem Blending) generiert, welche von einem virtuellen Projektil emittiert werden

KI:

- Computergegner werden durch eine (immer noch ziemlich blöde) KI gesteuert, welche mit ein wenig Übung aber von jedem nicht-Koreaner bezwungen werden kann. Das KI-Skript wird 1x pro Sekunde aufgerufen, um dem Computergegner Leben einzuhauchen.

Mathematisches:

Einige eigene, simple Klassen wurden für das Spiel geschrieben:

- Matrix: Nachbildung des OpenGL Matrix Stacks inklusive gängiger Funktionen (Perspective, Ortho, Translate, Rotate, Scale, Push, Pop)
- Frustum: Berechnet Frustum aus Modelview und Projection Matrix und berechnet Bounding Sphere Culling, gibt außerdem Distanz zu beliebigem Clip-Plane zurück.
- Vector: Simple 3D Vektor Klasse

Interface:

- Eine Font Rendering Klasse, welche eine Textur mit 2 ASCII Schriftsätzen verwendet
- Eine Interface Klasse, die simple, grundlegende Controls zur Verfügung stellt (Buttons, Tooltips)

Anforderungen (2. Abgabe - alt)

Entwurf der Datenstrukturen

Sämtliche Geometrie wird prozedural erzeugt, somit sind hierfür keinerlei spezielle Datenstrukturen notwendig. Die erzeugte Geometrie wird entweder:

- Bei Bedarf in ein temporäres Array gespeichert und gerendert
- Bei Objekterzeugung in ein VBO übertragen welches zum Rendern genutzt wird

Entwurf eines Kameramodells

Die frei bewegbare Kamera wird in einer Struct mit folgenden Parametern gespeichert, aus welchen die Modelview Matrix generiert wird:

- x, y, z: Das Zentrum des Bildschirms in Spielfeldkoordinaten
- rotV, rotH: Vertikale und Horizontale Rotation (x- /z- Achse)
- distance: Abstand der Kamera von der Zentrumposition

Außerdem wird aus diesen Werten on-the-fly die Kameraposition berechnet, um sicherzustellen, dass sich diese nicht unterhalb des Spielfeldes befindet.

Bewegte Objekte

Alle Spieleinheiten (Blobs) sind beweglich (eigene Einheiten durch den Spieler, gegnerische Einheiten durch die KI)

Implementierung von Texture Mapping

Das Terrain wird mit mehreren Texturen gerendert. Da das Terrain ein Grid aus Quads ist, bietet es sich an, die Texturkoordinaten direkt aus den x/y-Vertexkoordinaten zu generieren.

Beleuchtung und Materialien

Lichtquelle: Es gibt eine globale, direktionale Lichtquelle, welche schräg auf das Terrain strahlt.

Materialien: Es werden keine Materialien im eigentlichen Sinne verwendet, sondern unterschiedliche Shading-Methoden. Während das Terrain nur mit Ambient-und Diffuse gerendert wird (Specular macht dort wenig Sinn), werden die Blobs zusätzlich mit Specular gerendert.

Normalvektoren: Für das Terrain werden gemittelte Normalvektoren aus den umliegenden Punkten des Grids ermittelt. Für die Blobs werden die Normalvektoren im **Fragment**-Shader generiert

Steuerung

Die Steuerung ist ~~bereits weitestgehend~~ vollständig. ~~Lediglich~~ die Einheitenauswahl durch Ziehen eines Rahmens ~~funktioniert ebenfalls korrekt~~ ~~ist momentan nur provisorisch implementiert und~~ ~~funktioniert somit nur eingeschränkt.~~

Quellen und externe Libs

Externe Libs:

- SDL (Fenstererzeugung, Mouse- und Keyboardevents)
- SDL_mixer (Soundausgabe)
- GLEW (OpenGL Extensions)

Quellen:

- SDL/OpenGL 3.2 Grundgerüst/OpenGL Kontext Hack:
http://www.cg.tuwien.ac.at/courses/Realtime/opengl_3x_examples/sdl_ogl3.rar
- Terrain Normalmapping:
http://www.ozone3d.net/tutorials/bump_mapping.php
http://www.ozone3d.net/tutorials/mesh_deformer_p2.php
- Sowie weitere Quellen für verschiedene Kleinigkeiten