

Dokumentation

Acht Schätze



186.831 – UE Computergraphik

26.06.2014

Pascal Plank
Bernhard Reisenberger

Inhaltsverzeichnis

Dokumenation.....	1
Acht Schätze.....	1
1 Allgemeines.....	3
2 Requirements.....	3
3 Arbeitsaufteilung.....	4
4 Levelgenerierung.....	5
5 Kollisionserkennung.....	6
6 Beleuchtung und Texturierung.....	6
7 Features.....	6
8 Effekte.....	7
9 Tools.....	7
10 Bibliotheken.....	7

1 Allgemeines

“Acht Schätze” ist ein klassisches Jump-and-Run-Spiel. Der Spieler kontrolliert eine Spielfigur bzw. Spielball und versucht möglichst schnell zum Ende des Levels zu gelangen. Auf der Strecke muss er verschiedenen Hindernissen ausweichen und Obst aufsammeln.

Die Basis-Datenstruktur unseres Spiels ist eine Mischung aus einer Liste (std::vector) und einem Szenegraph mit 2 Ebenen. Die einzelnen Instanzen der Mauer-Teilabschnitte sind in einer Liste gespeichert. Zusätzlich gibt es eine Liste mit allen Obststücken und eine mit allen Hindernissen. Jedes Objekt hat seine zugehörige Model-Matrix, um sie im Worldspace zu platzieren.

Sämtliche Szeneobjekte wurden von uns modelliert und texturiert. Die Objekte werden im Collada-Format gespeichert und mithilfe von Assimp in das Spiel geladen. Als Imagelader findet DevIL Verwendung.

Das Spiel wird mithilfe einer Third-Person-Camera gespielt. Mithilfe der Taste „F“ kann auf eine freibewegbare Kamera umgeschaltet werden. Gesteuert wird mit den üblichen WASD-Tasten und der Leertaste für Sprünge, sofern sich der Hero auf der Wall befindet.

Taste	Funktion
W	Bewegung nach vorne (relativ zur Blickrichtung)
A	Bewegung nach links (relativ zur Blickrichtung)
S	Bewegung nach hinten (relativ zur Blickrichtung)
D	Bewegung nach rechts (relativ zur Blickrichtung)
Space/Leertaste	Springen
F	Toggle zwischen Spiele- und freibeweglicher Kamera
R	Reset (falls man von der Mauer fällt)
ESC	Beendet das Spiel
F2	Frametime on/off
F3	Wireframe on/off
F4	Textur-Sampling-Quality: Nearest Neighbor/Bilinear
F5	Mip Mapping-Quality: Off/Nearest Neighbor/Linear
F6	Static Level of Detail-Qualität: Low/Medium/High
F7	Glow on/off
F8	Viewfrustum-Culling on/off
F9	Transparency on/off

2 Requirements

- free movable camera (Toggle mit „F“)
- Moving object: Bewegter Hero, rotierende Zutaten (Banane und Apfel)
- lighting and materials: Direktionale Lichtquelle (Sonne)

- Controls (siehe obige Liste)
- Gameplay: Wurde implementiert. Der 3D Aspekt kommt beispielsweise beim Herunterfallen von der Mauer, dem Hinaufspringen auf die seitlichen Mauerteile oder das Springen auf Hindernisse zum Tragen.
- Complex Objects: Mauer, Bananen, Äpfel, etc.
- hierarchical animated Objects: Äpfel und Bananen werden beim Aufsammeln relativ zur Kugel/Hero nach oben in einer rotierenden Bewegung verschoben und skaliert.
- View-Frustum-Culling: mit Bounding-Spheres
- Transparency: Sichtbar beim Einsammeln von Bananen und Äpfeln, sie werden immer transparenter je weiter sie nach oben fliegen.
- Experimenting with OpenGL: u.a. VBOs, VAOs, FBOs, Mip Mapping on/off, Texture-Sampling-Quality, etc. Die Texturparameter kann man am besten mit der frei bewegbaren Kamera nahe am Boden erkennen.

Zusätzlich:

- dynamisches Objektladen mittels Instanzen und JSON
- Collisiondetection mittels Bullet
- Sound (Hintergrundmusik) und Effekte

3 Arbeitsaufteilung

Pascal Plank	Bernhard Reisenberger
Effekte	
static LoD: 1 Punkt	Shadowmapping: 1,5 Punkte
Normalmapping (tangent-space): 1 Punkt	Glow: 1 Punkt
Weitere Spielelemente/Tätigkeiten	
Gameproposal/Design document	Gameproposal/Desing document/poster
Controls and its implementation: <ul style="list-style-type: none"> • ECS: Spielbeenden • WASD: Steuerung • F: freemovable cam • F2: Frametime on/off • F3: Wireframe on/off • F4: Textur-Sampling-Quality • F5: Mip Mapping-Quality • F6: static LoD • F7: Glow on/off • F8: Viewfrustum-Culling on/off • F9: Transparency on/off 	Controls and its implementation: <ul style="list-style-type: none"> • Reset mit R • Space/Leertaste: Springen • HUD/grafische Anzeige von Statistiken with Win/Loose-Screen
Frustum Culling debugging	Frustum Culling
Erstes Gamesetup und Integration von GLFW	Herosteuerung
Modelloading via Assimp	Collisiondetection

Modelling and Texturing: <ul style="list-style-type: none"> • wall-pieces (straight, start/end, curve) • ingredients (apple/banana in 3 geometrical resolutions) • obstacles (roll, grid, planks, sandheap) • Boundingboxes for all objects 	
textured model in C++/Opengl (non-inverleaved VBOs, interleaved one too)	
free moveable camera (F-Key)	
General game structure and textoutput	
Skybox	
Lightning (directional light – sun)	
Creating Level with Levelfile and Levelloader via json	
Moving objects	
Sound (FMOD) + Soundeffects	
Textureloading via DevIL	
Simples Punktesystem/Gameplay	
Hierarchical animations	
Transparency	
Debugging Shadowmapping	
Shader	
Shader: Textured shader	Shader: Shadow Mapping + Debug
Shader: Normalmapping	Shader: Glow+Blur+Mixer
Documentation for Submission 1	
Documentation for Submission 2	
Demo-Video + Screenshots	

4 Levelgenerierung

Das gesamte Level von Acht Schätze wird dynamisch aus einem JSON-Levelfile generiert. Die Mauer wurde hierzu aus einzelnen quadratischen Mauerelementen nahtlos aneinander gefügt. Da sich die einzelnen Mauerabschnitte wiederholen gibt es zur Zeit drei verschiedene „Mauerprototypen“:

- Beginn-Element (= Ende)
- gerades Element
- Kurve

Durch geeignete Rotationen und Translationen, welche über Positions-Befehle im JSON-File definiert sind, werden die Elemente in die richtige Richtung gedreht und im Spiel positioniert. Da sich die einzelnen Mauerstücke praktisch nur durch ihre Model-Matrix unterscheiden, wurde eine Form der Instanziierung verwendet. Hierzu werden für das gesamte Spiel nur die Basis-Mauerabschnitte geladen und über Instanzobjekte, die die zugehörige Model-Matrix zum tatsächlichen Mauerabschnitt enthält, die komplette Welt generiert. Dies hat zur Folge, dass für jedes Mauerstück nur mehr eine 4x4-Matrix von wenigen Bytes gespeichert werden muss, anstatt ein komplettes Modell von etwa 130KB (beschleunigt zusätzlich das Rendering).

Die gleiche Herangehensweise wurde für die zu einem Mauerabschnitt gehörenden Hindernisse und einzusammelnden Zutaten verwendet. Die entsprechenden Hindernis- und Zutaten-Befehle wurden auch im JSON-File hinterlegt und sind somit frei konfigurierbar.

5 Kollisionserkennung

Die Kollisionserkennung wird in unserem Spiel mithilfe der Bibliothek „Bullet“ realisiert. Sie wird vor allem für die Erkennung von eingesammelten Zutaten, das „Nicht-durch-den-Boden-fallen“, Gravitation und für die Hindernisse eingesetzt.

Im Bezug auf Hindernisse und die Mauer dient die praktisch ausschließlich als „Stopper“, sodass unser Hero nicht durch die Objekte hindurchgehen kann. Beim sammeln der Zutaten wird zusätzlich verarbeitet um welche Art von Zutat es sich handelt und entsprechend darauf reagiert.

Aufgrund der von uns benützten Form der Levelgenerierung (keine tatsächlichen Objekte sondern nur Instanzen von Prototypen) müssen die für die Kollisionserkennung bestimmten Objekte vorverarbeitet werden, sodass die einzelnen Vertices der Objekte von Bullet mithilfe der richtigen Model-Matrix auch an entsprechender Stelle in der Spielwelt verarbeitet werden.

6 Beleuchtung und Texturierung

Jedes Objekt des Spiels durch eine zentrale, direktionale Lichtquelle (Sonne) und Ambientes Licht beleuchtet. Für die Texturierung wurde herkömmliches UV-Mapping für sämtliche Modelle verwendet, welches auch beim Normalmapping verwendet wird.

7 Features

- Dynamische (und damit speicher- und verarbeitungsschonende) Generierung des Levels mittels JSON und Instanzen (nicht auf GPU)
- Physics und Collision-Detection
- Texturierung über UV-Mapping
- multiple Hindernis- und Zutatentypen
- Normalmapping
- HUD
- Glow
- static LoD
- Shadow-Mapping

- + alle verlangten Requirements und Features

8 Effekte

- Shadow Mapping: um der Szenerie einen besseren 3D-Eindruck zu verleihen, größtenteils basierend auf die in der Vorlesung gezeigten Vorgangsweise
- Normal Mapping: zur realistischeren Gestaltung von Mauer und Zutaten, basierend auf Techniken von UE Einführung in die Computergraphik. Die verlangte Berechnung im Tangentspace auf komplexen Objekten, die nicht auf den Achsen ausgerichtet sind, kann man gut an den drehenden Bananen oder Äpfeln erkennen wenn man die frei bewegbare Kamera einschaltet (F) und den Glow-Effekt ausschaltet (F7).
- Static LoD: zur Reduktion von Zeichenaufwand bei weit entfernten Zutaten (Apfel/Banane). Da es die Idee des Effekts ist, dass man ihn nicht bemerkt, kann man mit F6 zwischen 3 static LoD-Qualitätsstufen umschalten. Auch hier kann man den Effekt an den drehenden Bananen oder Äpfeln erkennen wenn man die frei bewegbare Kamera einschaltet (F) und den Glow-Effekt ausschaltet (F7).
- Glow: als „Special“ Effekt, basierend auf die in der VO gezeigten Techniken.

9 Tools

Für die Erstellung und Texturierung (mittels UV-Mapping) der Modelle (Mauer, Hindernisse, Zutaten, Hero) wurde Cinema 4D R12 verwendet.

Als Entwicklungstool wird Microsoft Visual Studio 2012 verwendet.

10 Bibliotheken

Folgende Bibliotheken finden bei unserem Spiel Verwendung:

- GLFW (<http://www.glfw.org/>)
- GLEW (<http://glew.sourceforge.net/>)
- DevIL (<http://openil.sourceforge.net/>)
- FMOD (<http://www.fmod.org/>)
- Bullet (<http://bulletphysics.org/wordpress>)
- Assimp (<http://assimp.sourceforge.net/>)
- glm (<http://glm.gtruc.net/>)
- jsoncpp (<http://jsoncpp.sourceforge.net/>)