

Submission 2: Dokumentation

-

Extremely Crazy TU Stress (ECTS)

CG UE, SS2015

Hanna Huber, 0925230
Patrick Wahrmann, 1327120

Spielanleitung

Bei *Extremely Crazy TU Stress* (= ECTS) geht es um das Sammeln von ECTS Punkten und das damit einhergehende Bestehen des Semesters.

Einerseits muss man, um nicht zu verhungern, regelmäßig Nahrung in Form von Energiepunkten, die als wunderschöne drehende Diamanten in die Szene eingebettet sind, zu sich nehmen. Berührt man einen Diamanten, sammelt man direkt Energiepunkte.

Andererseits muss man nach demselben Prinzip Wissenspunkte sammeln, die durch Glühbirnen dargestellt sind, um sich auf Prüfungen vorzubereiten.

Dabei hilft es, sich zu überlegen, wo man sich im Freihaus am besten stärken bzw. Wissen aneignen kann.

Sobald man glaubt genug gelernt zu haben, kann man zu einer Prüfung antreten. Zur Prüfung kommt es, sobald man sich einem der zwei Professoren (Prof. Wimmer im Erdgeschoss des grünen Bereichs und der Meister beim [FH1](#)) nähert (aber Vorsicht - wenn man ihnen zu nahe kommt, fühlen sie sich - wie Studenten - bedrängt). Dabei werden einem unterschiedlich viele Fragen (in Form von fliegenden Fragezeichen) an den Kopf geworfen. Um diese zu beantworten, muss der Professor mit dem gesammelten Wissen beeindruckt (=abgeschossen) werden. Manche Fragen sind besonders hinterhältig gestellt und stehen der erfolgreichen "Kommunikation" mit dem Professor im Wege... Die Anzahl der nötigen Wissenspunkte entspricht dem Wert der Prüfung in ECTS-Punkten. Wer die Prüfung verpatzt, kann nach entsprechender Wartezeit wieder antreten (Die STEOP wird vorausgesetzt, bei der Anzahl der Antritte sind die Professoren deshalb nicht so streng).

Das Semester dauert 150 Sekunden. In dieser Zeit müssen beide Prüfungen positiv absolviert werden, damit das Semester bestanden ist (und nicht die Familienbeihilfe verloren geht).

Ein weiterer Aspekt des Spiels ist, dass man nicht über die (mangels genügend Lernräumen) am Boden lungernden Studenten stolpern darf. Sollte man über einen Studenten stolpern, schlägt sich der darauffolgende Konflikt auf die Körperenergie...

Implementierung

Beim Start des Spiels wird die main Methode in der Main-Klasse aufgerufen, woraufhin zuerst in init() OpenGL, Modelle und alles andere initialisiert wird und dann die Game-Loop gestartet wird. In der Game-Loop werden die Methoden update(), der Z-Buffer-Pass und der Shading-Pass aufgerufen.

Alle im Spiel verwendeten Objekte werden mit der SceneImporter-Klasse importiert und im ObjectManager gespeichert und verwaltet.

Die Klasse SceneObject ist die Superklasse aller im Spiel vorkommenden Gegenstände. Diverse Kategorien an Gegenständen werden von Subklassen repräsentiert. Alle Gegenstände die eine Oberfläche (Meshes) haben werden als Geometry gespeichert. Im Gegensatz dazu existieren auch Gegenstände ohne Oberfläche wie der (First-Person-)Player, die Kamera und Lichter. Doch auch die Geometries werden weiter unterteilt: So gibt es zB. Items oder Environment-Objekte.

Jede Geometry hat eine Liste von Meshes gespeichert, ein Mesh wiederum besteht aus Vertices, Faces und ähnlichen für das Rendern relevanten Daten.

Die Collision Detection wird in der Physics-Klasse behandelt. Sobald der Spieler ein Item aufammelt, über einen Studenten stolpert oder einem Professor zu nahe kommt, wird dies hier erkannt. Mehr dazu im Abschnitt Bibliotheken/Bullet.

Die Klasse ShadowMap ist für das Shadowmapping zuständig, initialisiert die nötigen Matrizen, den Z-Buffer-Shader, den Framebuffer und die Cubemap, bereitet den Z-Buffer- und den Shading-Pass vor und beendet diese. Sie speichert auch den Index der Lichtquelle, für die das Shadowmapping realisiert wurde.

Das Rendering von Text (zum Beispiel der Energy-Punkte) passiert in TextManager. Dieser wird am Anfang mit Hilfe von TrueType initialisiert. Anschließend werden alle Buchstaben auf jeweils einem genau angepassten Rechteck als Textur mit dem TextShader auf den Bildschirm gerendert. Die Implementierung stützt sich auf ein sehr gutes [Tutorial](#).

Effekte

- ❖ **Omnidirectional Shadowmapping:** Das Shadowmapping wurde für eines der Punktlichter (in main durch *sm_light_id* gekennzeichnet) implementiert. Die euklidische Distanz zum nächsten Objektpunkt aus Sicht der Lichtquelle wird mithilfe eines Framebuffers mit Depthattachment in einer Cubemap gespeichert.

Die Berechnung erfolgt im ZBufferShader (inkl. GeometryShader). Für den Shadingpass speichern die beiden Fragmentshader die Cubemap als uniform (samplerCubeShadow).

Da der Tiefenwert im ZBuffershader manuell gesetzt wird, muss dieser, um Shadowacne zu vermeiden, zusätzlich skaliert und ein Offset hinzugefügt werden.

Zusätzlich werden für PCF weitere Texturparameter bei der Initialisierung gesetzt.

Abgesehen von den [Folien](#) im CG-WIKI wurden folgende Quellen verwendet:

- learnopengl.com: [Point Shadows](#)
- Peter Houska: [Omnidirectional Shadows](#)

- ❖ **Blooming:** Wenn man in einem sonst dunklen Bereich eine besonders helle Stelle (zum Beispiel durch ein Fenster in die helle Natur) betrachtet, fällt auf dass der helle Bereich vor dem Auge / der Kameralinse verschwimmt. Dieser Effekt wird mittels Blooming simuliert. Die Realisierung erfolgt mittels Post Processing und Framebufferobjects (FBOs). In einem ersten Schritt werden zuerst alle besonders hellen Pixel aus dem gerenderten Bild gefiltert (im ThresholdShader) und in ein FBO gespeichert. Als nächstes wird zuerst mit einem 5x1 Gauß-Filter in eine Richtung geblurt und dann mit einem 1x5 Gauß-Filter in die andere Richtung (GaussShaderHorizontal & GaussShaderVertical). Anschließend wird das geblurte Bild mit der ursprünglichen gerenderten Szene kombiniert (FboShader). Dieser Effekt kann mit F7 ein- und ausgeschaltet werden. Zum Beispiel kann er beim Eingang des Freihauses beobachtet werden, wo das ganze Tageslicht von draußen hineindringt und die armen Studenten blendet.

Abgesehen von den [Folien](#) im CG-WIKI wurden folgende Quellen verwendet:

- learnopengl.com: [FrameBuffers](#)
- nvidia GPUGems: [Chapter 21. Real-Time Glow](#)
- <http://prideout.net/archive/bloom/>

- ❖ **Animated Textures:** Animierte Texturen werden verwendet um Videos oder Bildabfolgen auf Objekte aufzutragen. In *ECTS* ist die animierte Textur im Erdgeschoss des grünen Bereichs im Freihaus zu sehen. Auf einem Monitor wird das Intro-Video von “*Bad Bob*” (Bad Bob ist ein zweidimensionales Android-Spiel, das im Rahmen der Multimedia UE (188.912) im WS14 von Patrick Wahrmann und Martin Ruiss entwickelt wurde) abgespielt.

Die Implementierung der animiertenTexturen ist prinzipiell simpel aufgebaut. Die Klasse AnimatedTextureShader basiert auf dem normalen TextureShader, der auch zum Beispiel für alle Wände im Spiel zuständig ist. Dynamisch wird in

jedem Framedurchlauf des Spiels abhängig von der Zeit berechnet welcher Video-Frame derzeit gezeigt werden muss. Dieser wird dann als Textur gebündelt und gezeichnet.

Features

- ❖ **Sammle Energiepunkte**
- ❖ **Sammle Wissenspunkte**
- ❖ **Vermeide Zusammenstöße mit Studenten/Professoren**
- ❖ **Trete zur Prüfung an**
- ❖ **Fliegende Fragen**
- ❖ **Beeindrucke die Professoren mit deinem Wissen**
- ❖ **Info-Anzeige**
- ❖ **Wireframe-Modus**
- ❖ **Transparente Items**
- ❖ **Schatten**
- ❖ **Bloom**
- ❖ **Variable Texturesampling-/Mipmapping-Qualität**
- ❖ **View-Frustum-Culling**
- ❖ **Kopfschüttelnde Professoren**

Requirements

- ❖ **Animierte Objekte:** Die Professoren drehen sich in Erwartung der nächsten Prüfung im Kreis und schütteln vor Sorge um das Bildungsniveau der Studierenden den Kopf. Zwei Meshes (Kopf+Haare) bewegen sich zusätzlich zur Bewegung des gesamten Objekts (Rotation um ein definiertes Zentrum) relativ dazu (lokale Rotation).
- ❖ **Beleuchtung und Materialien:**
 - Materialien werden durch Shader repräsentiert. Die wichtigsten Rollen spielen der TextureShader und der BlinnPhongShader, der alle Materialien übernimmt, die keine Textur haben. Bei beiden Shader-Arten wird das Licht berücksichtigt, das in Form mehrerer Pointlights existiert. Sowohl die Farbe des Lichts, als auch die Entfernung wirkt sich auf die Darstellung des Materials aus. Derzeit verwenden der Boden, die Wände, die Studenten und die Professoren einen TextureShader. Die Decke, Energiediamanten und Glühbirnen verwenden den BlinnPhongShader.

- Die Lichtquellen werden mittels Assimp importiert. Lichter sind in der Klasse Pointlight implementiert. Dabei werden konstante, lineare und quadratische Attenuation bereits berücksichtigt, um die Lokalität des Lichts zu simulieren. Die Parameter des Lichts werden den Shadern übergeben und berücksichtigt. Jedes Stockwerk wird von ungefähr 4 bis 5 Lichtern ausgeleuchtet und zusätzlich sind noch zwei schwache Lichter über dem Gebäude platziert, um auch dunklere Stellen ähnlich wie beim Three-Point-Lighting auszuleuchten.
 - Für eine Lichtquelle (Index *sm_light_id*) wird der Farbwert zusätzlich skaliert, je nachdem, ob der entsprechende Objektpunkt im Schatten liegt oder nicht. Die (wieder durch die Farplane dividierte) euklidische Distanz wird dafür als Referenzwert an die samplerCubeShadow-Textur übergeben. Diese liefert den passenden Skalierungswert zurück.
 - ❖ **Steuerung:** Der fürs Gameplay relevante Teil der Steuerung wird in *Physics::update* verarbeitet:
 - Rotation: gesteuert vom User mithilfe des Cursors; verarbeitet in *Player->handleInput()*.
 - Translation: gesteuert vom User mithilfe der WASD-Tasten und der Leertaste; verarbeitet in *Physics::movePlayer*.
 - Schießen: gesteuert vom User mithilfe der linken Maustaste; verarbeitet in *Physics::throwBullet*; setzt voraus, dass genügend Wissenspunkte vorhanden sind; reduziert die Wissenspunkte um 1 (pro Geschoss)
- Zusätzlich kann der User bestimmte Features ein- und ausschalten. Dieser Input wird in der Klasse UserInput verarbeitet.
- F2: Info anzeigen on/off
 - F3: Wireframe-Modus on/off
 - F4: Texturesampling-Methode linear/nearest neighbor
 - F5: Mipmapping-Modus linear/nearest neighbor/off
 - F6: Anfänger/Turoren-Modus (unendlich viel Zeit und Energy)
 - F7: Bloom on/off
 - F8: ViewFrustum-Culling on/off
 - F9: Transparenz on/off

Modellerstellung

Die Modelle wurden mit [Blender](#) erstellt. Das Freihaus-Modell besteht hauptsächlich aus Ebenen mit einer Wand- (oder Boden) Textur. Die Wandtextur wurde mit einem praktischen Tool namens [Fresh Paint](#) auf einem Microsoft Surface erstellt. Anzumerken ist, dass Blender aufgrund von Bugs diverse Probleme verursachte, da es beim Exportieren ständig Texturen vermischte, die teilweise manuell in .dae-Dateien korrigiert

werden mussten. Zur Übersicht war es jedoch sehr praktisch, dass man bei Blender Objekte in Gruppen einteilen kann, die dann einzeln exportiert oder angezeigt werden können. Somit war es möglich auch Objekte im Erdgeschoss anzuzeigen indem man das darüberliegende Geschoss unsichtbar machte.

Zusätzliche Bibliotheken

- ❖ **FreeImage:** Zum Laden von Bilddateien für Texturen wird FreeImage (<http://freeimage.sourceforge.net>) verwendet. In `Texture::loadFile(path)` wird die Bilddatei mithilfe von FreeImage-Befehlen eingelesen, konvertiert und als entsprechende Datenstruktur an den Konstruktor zurückgegeben, wo die eigentliche Textur mithilfe von OpenGL-Befehlen erstellt wird.
- ❖ **Assimp:** Zum Importieren der Modelle wird Assimp (<http://assimp.sourceforge.net>) verwendet. Der Zugriff auf Assimp erfolgt ausschließlich in der Klasse `SceneImporter`. Die Methode `importFrom()` ist die einzige die von außen zugreifbar ist. Über sie wird der Befehl gegeben aus einer bestimmten Datei mithilfe von Assimp Objekte zu importieren. Die Methode liefert einen Vector mit importierten Objekten (Geometries) zurück. In der Methode wird mit Hilfe von Assimp eine `aiScene` importiert, die dann in `iterateOverAllNodes()` rekursiv durchlaufen wird. Derzeit werden alle Dateien inklusive Materialien berücksichtigt. Die Materialien werden in Shader umgewandelt und in den dazugehörigen Geometries gespeichert. Falls notwendig werden dazu auch Texturpfade ausgelesen, die dann im Konstruktor des `TextureShaders` mit Hilfe von FreeImage geladen werden (siehe oben).
- ❖ **GLFW:** GLFW (<http://www.glfw.org/>) wird verwendet, um die Interaktion mit dem Benutzer zu ermöglichen. Dazu gehört das Erstellen des OpenGL-Kontext (Version, Core Profile) und des Fensters samt Einstellungen (Größe, Vollbildmodus), sowie die Benachrichtigung über die Benutzereingabe (Steuerung, Beenden des Programms).
- ❖ **GLEW:** Auf diverse Funktionalität von OpenGL wird mit GLEW zugegriffen (<http://glew.sourceforge.net/>).
- ❖ **GLM:** GLM (<http://glm.g-truc.net/0.9.6/index.html>) stellt nützliche mathematische Datenformate (`vec3`, `vec2`, `mat4`, ...) und Funktionen (`length`, `distance`, `normalize`,...) zur Verfügung und ermöglicht die Verwendung der Operatoren `+`, `-`, `*` für arithmetische Operationen zwischen Vektoren/Matrizen.
- ❖ **Bullet:** Für physikalisch korrekte Bewegungen (Gravitation, Kollisionen, Wurfparabeln) wird die Bullet-Bibliothek (bulletphysics.org) verwendet. Für alle geometrischen Objekte gibt es entsprechende physikalische Objekte (`btRigidBody`s, etc), die die Physikklasse speichert. Für Kollisionen werden Callbacks verwendet. Bei Professoren wird zwischen Körper

(Kollision->Punkteabzug) und näherer Umgebung (Kollision->Prüfungsbeginn) des Professors unterschieden. Die Glühbirnen werden mithilfe eines Impulses in Blickrichtung geworfen. Die Position des Players wird mithilfe der Funktion *btRigidBody->translate* verändert.

- ❖ **Freetype:** Freetype (<http://www.freetype.org/index.html>) wird verwendet, um Text am Bildschirm anzuzeigen. Freetype generiert aus einer TrueType-Schrift leicht weiterverwendbare Bitmaps aller benötigten Zeichen. Auch sehr große Projekte (Linux, iOS, Android,...) greifen auf Freetype zurück.

Externe Dateien

- ❖ **Schriftarten:** Die AveraSans-Fontfamilie stammt von <http://openfontlibrary.org/de/font/averia-sans/> und die doch nicht benutzte Dotrice-Fontfamilie von <http://openfontlibrary.org/de/font/dotrice>
- ❖ **Modelle:** Als Basis-Modell für die Studenten diente [dieses Modell](#) und die Professoren basieren auf einem [Lego-Männchen Mesh](#). Die bei den Professoren verwendeten Texturen sind von [hier](#) und [hier](#).