

Bear-In, Bear-Out

Projekt Dokumentation

Martin Smiech (1426853) und Dea Cizmic (1425788)

Implementierung

- Frei bewegliche Kamera
 - Die Kamera folgt dem Bären (wenn locked)
 - Die Kamera kann durch Halten der linken Maustaste (wenn nicht gelockt) frei gedreht werden (um die eigene Position) und in alle Richtungen blicken.
 - Die Kamera kann unlocked werden (Drücken der L-Taste) und dann kann man sich mit der Kamera alleine durch die Welt bewegen. ("spectate" mit AWSD-Tasten)
- Bewegliche Objekte

Die Bewegung der Objekte wurde durch einfache Matrixmultiplikation zwischen Modellmatrix und Transformationsmatrizen implementiert. Dazu wurden die vordefinierten Methoden wie scale/rotate von glm verwendet. Derzeit bewegt sich nur der Würfel (Rotation um die eigene Achse) und der Bär.
- Texture Mapping wurde so implementiert wie in der Vorlesung vorgezeigt. Zusätzlich wurde DevIL für das Laden der Bilder der Texturen benutzt.
- Beleuchtung und Materialien (siehe Effektliste und für grundsätzliches Licht 1. Submission)
- Controls

Der Bär kann mit den Pfeiltasten sowie mit "Space" gesteuert werden. Mit der linken/rechten Pfeiltaste dreht sich der Bär in die entsprechende Richtung. Mit der oberen und unteren Pfeiltaste bewegt er sich nach Vorne/Hinten und mit der Space-Taste springt der Bär nach oben und fällt entsprechend mit Zeit und Gravitation wieder herunter.
- Gameplay

Es werden die Modelle der Rennstrecke und des Bären mittels Assimp geladen und so Positioniert, dass sich der Bär beim Start der Strecke befindet. Es wurde eine vereinfachte win/lose condition mithilfe eines Würfels implementiert. Der Würfel soll das Ziel darstellen. Wenn der Bär sich auf dem Würfel mit dem Futter befindet, wird auf dem Bildschirm „Game Won!“ ausgegeben und wenn der Bär ins Wasser fällt oder mit einem Stein kollidiert wird "Game Over!" ausgegeben.

Features

- Laden von Szenen aus COLLADA-Files
- Ein komplexes Modell für den Bären (siehe Quellen)
- Eine Spielwelt (die auf eine Quelle basiert und erweitert/verändert wurde)
 - Hindernisse (Felsen, siehe Quellen) und Boni (Fische, ebenfalls Quellenangabe)
- Bär bewegt sich "frei" (nicht durch Wände/Boden) in dieser Welt
- Kann ein Ziel erreichen (als Würfel dargestellt, Text Anzeige in der Mitte des Displays erfolgt)
- Kann verlieren in dem er mit einem Stein kollidiert oder ins Wasser fällt (dargestellt als eine transparente Ebene).
- Die Lichtquelle wird gerendert und als Würfel dargestellt.
- HDR (Toggle F10) und Bloom (Toggle F11) verschönern die Darstellung der Szene, Sonne leuchtet
- Die Beine des Bären werden beim bewegen des Bären animiert.
- In der Szene liegt eine Transparente Ebene die Wasser darstellen soll.

Beleuchtung der Objekte

Zunächst wird die Szene aus der Perspektive (orthogonale Projektion, weil parallele Lichtquelle -> "Sonne", also keine perspektivische Verzerrung) gerendert um so eine Depth Map zu erstellen um nachher basierend darauf Schatten zu erzeugen.

Um die Objekte zu Beleuchten wurde ein Blinn-Phong-Shader (Erläuterung dazu im 1. Submission Dokument) implementiert. Dazu werden Vertexkoordinaten und -Normalen im Vertex-Shader zu Weltkoordinaten transformiert und interpoliert an den Fragment-Shader gesendet. Diese beiden Shader heißen bei unserer Implementierung `main_shader.vert` und `main_shader.frag`. Dieser Shader zeichnet für weitere Verarbeitung das grundsätzliche Ergebnis in den Farbpuffer (Color Buffer Attachment 0) und die hellsten Stellen des Szenenausschnitts in einen separaten Farbpuffer (Attachment 2).

Dieses Beleuchtungs- und Schattierungsmodell unterstützt ambientes, diffuses und spekuläres Licht. Alle Objekte können bei uns über eine Textur verfügen. Das Ergebnis der Lichtberechnung (Summe der drei zuvor genannten Lichtarten) wird mit der Farbe der Textur (UV-Koordinaten für den bestimmten Vertex im Fragment-Shader) multipliziert. Zusätzlich dazu wird noch die Alpha-Komponente zum Darstellen der Transparenz benutzt.

Um diese Beleuchtung und Schattierung/Schatten schnell zu sehen, wurde eine Punktlichtquelle über der Welt platziert. Diese wird als Würfel dargestellt. Diese ist nur zu Illustrationszwecken ein Objekt "in der Nähe" ist aber effektiv unendlich weit entfernt bei der Lichtberechnung.

Wenn der Bär sich in Richtung des Ziels bewegt, wird er von vorne beleuchtet und wenn die Kamera im richtigen Winkel ist, sieht man ein Glänzen auf dem Bären (specular, analog auf dem Boden). Unter und hinter dem Bären sieht man dann einen Schatten, der durch ShadowMapping erzeugt wird.

Effekte

- Shadow Mapping

So wie in der Vorlesung erklärt wird zuerst eine Depth Map (1. Durchlauf, `depth_shader`) erzeugt in dem die Szene aus der Position der Lichtquelle gerendert wird. Danach wird diese Depth Map an den Shader weiter übergeben wo dann die Tiefenwerte ausgelesen werden und verglichen werden. Alle Punkte die im 2. Durchlauf (also im `main_shader`) einen größeren Tiefenwert haben, als die entsprechende Stelle in der Depth Map, sind schattiert und somit dunkel/schwarz.

- Normal Mapping

Für Normal Mapping werden zuerst die Tangenten und Bitangenten eines Punktes mithilfe von Assimp berechnet (es gibt ebenfalls eine alternative Implementierung dieser Berechnung die mit Quellenangabe in unserem Code zu finden ist, welche verwendet wird, wenn Assimp keine Tangenten/Bitangenten berechnet). Diese werden dann an den Shader weitergegeben. Im Shader wird dann die Tangent-Bitangent-Normal Matrix berechnet. Diese braucht man um die Normalen aus der Normal Map (wird als Textur gebunden) von Tangent- in Worldspace zu transformieren um die weiteren Berechnungen wie zuvor durchzuführen. Normal Mapping ist am besten an den Steinen und an dem Würfel zu sehen.

- HDR

Um das Licht bzw. die Lichtwerte in unserer Szene anzupassen und um helle Objekte verhältnismäßig besser sichtbar zu machen, verwenden wir HDR mit Reinhard Tonemapping. Dadurch können höhere Helligkeitswerte, also Werte die den LDR (Low Dynamic Range [0.0, 1.0]) Bereich übersteigen, berücksichtigt werden. Um das zu bewerkstelligen, werden (wie zuvor beim `main_shader` erklärt) werden besonders helle Töne in einen separaten Buffer geschrieben und in der Nachbearbeitung (`post_processing_shader`) gemappt entsprechend eines exposure Wertes.

- Bloom

Der Überblendungseffekt "Bloom" baut bei uns auf den im HDR Rendering eingeführten Aufspalten der Renderziele auf. Er verwischt mittels Gauß-Filter (Zwei-Passfilter, 5 horizontale und 5 vertikale Durchläufe) die Hellen stellen des Bildes. Dieses verwischte Ergebnis wird ebenfalls in einem eigenen Color Buffer (als Textur im Nachbearbeiten verwendet) gespeichert. Die Werte dieses Buffers/Textur werden anschließend zu den Helligkeitswerten im Post Processing addiert und in den LDR Bereich gemappt bzw. geclamped (auf 1.0 abgeschnitten) falls HDR aktiv bzw. nicht aktiv ist.

- Animated Textures

Auf dem Wasser im Spiel wird ein animiertes Video einer Wasseroberfläche abgespielt. Zusätzlich dazu wird zur Motivation des Bären ein Video auf einer Leinwand abgespielt.

Tools

Für das Bearbeiten und erstellen der 3D Modelle wurde Blender benutzt. Für die Physik wurde die neueste (Open-Source) Version von Nvidia PhysX verwendet. Zum Debuggen der Physik-Engine wurde Nvidia PhysX Visual Debugger benutzt.

Debug Features (Additional Controls)

- F1 = Hilfestellung
- F2 = Frametime ON/OFF
- F3 = Wire Frame ON/OFF
- F4 – Texture-Sampling-Quality: Nearest Neighbor/Bilinear
- F5 – Mip Mapping-Quality: Off/Nearest/Linear
- F6 – Normal Mapping
- F7 – Shadow Mapping
- F8 – Viewfrustum-Culling ON/OFF
- F9 – Transparency ON/OFF
- F10 – HDR ON/OFF
- F11 – BLOOM ON/OFF
- L – Lock/Unlock Camera
- R – „Cheat“ Reset Game Over/Game Won Lock

Bibliotheken

- Assimp 3.2 für das Laden der 3D Modelle (bzw. Szene)
 - <http://www.assimp.org/>
- DevIL 1.7.8 für das Laden von Bildern (bzw. Texturen)
 - <http://openil.sourceforge.net/>
- glew 1.13.0 für das Laden von OpenGL und Verfügbarkeitsüberprüfung
 - <http://glew.sourceforge.net/>
- glfw 3.1.2 für das Fenster- und Input-Handling
 - <http://www.glfw.org/>
- glm 0.9.7.3 als Mathematik-Bibliothek (Vektoren und Matrizen angelehnt an OpenGL)
 - <http://glm.g-truc.net/0.9.7/index.html>
- zlib 1.2.8 als Kompressionsbibliothek, wird von den anderen verwendeten Bibliotheken eingesetzt
 - <http://zlib.net/>
- Nvidia PhysX 3.3.4 eingesetzt zur Physik Simulation als auch Kollisionsberechnung
 - <https://developer.nvidia.com/physx-sdk>

Quellen

Verwendete Quellen für C++ und GLSL-Shader wurden an den Orten gekennzeichnet und genannt an denen sie verwendet wurden. Beispielsweise wurde das Laden der Modelle aus Assimp mit Hilfe von <http://www.nexcius.net/2014/04/13/loading-meshes-using-assimp-in-opengl/> (u.a.) bewerkstelligt und an der entsprechenden Stelle im Code gekennzeichnet.

Weitere Quellen/Referenzen siehe Code im Git-Repo.

Zusammenfassend wurden das CGUE-Wiki, die Vorlesungen und die Folien dazu als Erstquellen verwendet. Weitere Quellen die darüber hinausgehen sind (bis auf wenige Ausnahmen, die im Code an ansprechenden Stellen gekennzeichnet sind):

- Buch: "Learning Physics Modeling with PhysX" von "Krishna Kumar"
- Buch: "OpenGL SuperBible" (nur als Nachschlagewerk)
- <http://www.learncpp.com/> (nur als Nachschlagewerk)
- <http://www.opengl-tutorial.org/>
- <http://ogldev.atspace.co.uk/>
- <https://open.gl/>
- <http://learnopengl.com/>
- <http://www.lighthouse3d.com/tutorials/>

Darüber hinaus wurden Modelle/Meshes aus dem Internet verwendet. Diese wurden für unsere Anwendungsfälle angepasst. Die Quellen dazu sind folgende:

- BearBrown - <http://tf3dm.com/3d-model/bear-98965.html>
- SnowTerrain - <http://tf3dm.com/3d-model/snowy-terrain-46334.html>
- Rock - <http://tf3dm.com/3d-model/rock-86533.html>
- Salmon - <http://tf3dm.com/3d-model/salmon-44387.html>