C.E.S. Dokumentation

Implementierungsbeschreibung:

Frei bewegbare Kamera:
Es gibt ein eigenes Kamera Objekt das sich auf ein Target zentrieren lässt. Die Kamera lässt sich um das Target drehen.

Bewegende Objekte:
- Der Charakter lässt sich steuern. Elemente vom Charakter bewegen sich. Dazu wird eine Objekthierarchie verwendet um Objekte gruppieren zu können.
- Die einsammelbaren Objekte drehen sich.
- Eine schwebende/fahrende Plattform kann den Spieler über ein Hindernis befördern.

Textur Mapping:
Zum Laden von Textur-Bildern wird die SDL_Image Library verwendet.
Texturen werden in einem Shader mittels texture-coordinates welche in den .obj Dateien gespeichert sind gemappt.

Einfache Beleuchtung und Materials:
Omnidirektionale Lichtquellen vorhanden.
Eigenschaften der Lichtquellen:
- ambient
- color
- position
- intensity
- constant attenuation
- linear attenuation
- quadratic attenuation

Shader für multiple Lichtquellen inkl. Texturierung (blinn-phong illumination, gouraud shader).
Materialeigenschaften werden derzeit fix gesetzt (stärke der specular reflection).

Overlay-Text:
Mit Hilfe der FreeType Library wurden Overlay-Text realisiert der für die GUI und zusätzliche Informationen verwendet wird, z.b. auch beim ein/ausschalten einer Funktion wird kurz eine Meldung eingeblendet.

Animations-System:
Es wurde ein selbstentwickeltes flexibles und robustes Animations-System implementiert. Der Held besitzt eine Idle- und eine Walk-Animation zwischen denen fließend überblendet wird, so geht er beispielsweise nur kleine Schritte wenn man sich langsam fortbewegt oder dreht. Es könnten aber beliebig viele weitere Animationen integriert werden die dynamisch mit dem überblendungsmechanismus kombiniert werden könnten (z.B. könnte der Held während man läuft mit dem Kopf auf etwas fokusiert bleiben). Weiters ist zu beachten das bei den Animationen eine vielzahl von Parametern gleichzeitig dynamisch modifiziert werden, z.B. die Neigung und Position einzelner Elemente, auch werden unterschiedliche Funktionskurven zur Berechnung der Bewegungen eingesetzt.

Collision-Detection and Response:
Es wurde eine eigene Collision-Detection implementiert (für Hilfestellungen siehe Referenzen). Dabei werden Bounding-Spheres und Ellipsoide sowie Triangle-Meshes verwendet um akkurat Kollisionen erkannt werden können und entsprechend auf diese reagiert werden kann.

Steuerung:

Kamera drehen mit W,A,S,D
Laufen mit den Pfeiltasten.
Springen mit F.
Beenden mit ESC.
Zoom rein/raus mit Punkt und Minus.
Position reseten mit 0.
Bounding-Spheres ein/aus mit 1.
LightSource modelle ein/aus mit 2.
CollisionDetection ein/aus mit 3.
Shadow-Volume-Preview ein/aus mit 4.
Schatten ein/aus mit 5
Normal mapping ein/aus mit 6
Bloom und HDR ein/aus mit 7
Alternative Steuerung mit 9
Hilfe anzeigen/ausblenden mit F1
Frames per second anzeigen/ausblenden mit F2
Wireframes ein/aus mit F3.
Texture Sampling Nearest/Bilinear mit F4
Mimap Sampling Off/Nearest/Bilinear/Trilinear mit F5
Free-Flight-Mode mit Enter (Fliegen dann mit der Sprungtaste 'F').
Es existiert eine config-Datei um z.B. Fullscreen zu aktivieren.

“Features” des Spiels:

- Ein steuerbarer Charakter
- Eine steuerbare Kamera die dem charakter folgt.
- Eine kleine Spielwelt, die den ersten Level des Spiels repräsentiert.
- Texturierte Modelle.
- 2 Lichtquellen.
- Volumetrische stencil-shadows
- Normalmapping
- Bloom
- HDR
- Objekthierarchie: Bsp: der charakter kann bewegt werden und elemente des charakters bewegen sich unabhängig mit.
- Shader für multiple lichtquellen inkl. Texturierung (blinn-phong illumination, gouraud shader)
- Modelle werden aus .obj dateien geladen und verwenden neben vertices auch normals und texture-coordinates.
- Es gibt ein Kollision-Detection und Response System. Zuerst werden bounding-spheres überprüft und anschließend erst detailiert Ellipsoid-triangle tests durchführt (was später noch optimiert wird). Das Kollidierende Objekt wird entsprechend der Kollisions-Position und des Geschwindigkeits-Vektors repositioniert.
- Es gibt einsammelbare Gegenstände und einen Score der mitzählt.
- Man kann als GameOver bedingung in den Abgrund stürzen, der Spieler wird dann zur Zeit wieder zum Anfang zurück gesetzt.
- Es gibt Bewegende Objekte die mit dem Spieler interagieren können, exemplarisch eine schwebende/fahrende Plattform.
- Ein eigenes Animations-System inkl. Animationen
- Overlay-Text
- Selbst gezeichnete Texturen und Normal-Maps auf dem Haupt-Charakter.

Wie und welche Objekte beleuchtet und/oder texturiert werden:
Alle Objekte im Spiel haben eine Textur zugewiesen bekommen. Beleuchtung wird mittels omnidirektionalen Lichtern umgesetzt, wobei eine Lichtquelle sich mit dem steuerbaren Charakter mitbewegt. Zur Texturierung werden Texturkoordinaten verwendet welche in den .obj Dateien der Modelle gespeichert sind.

Effekte:

Bloom Effekt:

Der Bloom effekt wurde durch eine Kombination eines Brightpassfilters in Verbindung mit einem Gaußfilter implementiert. Zuerst wird der Brightpassfilter auf das gerenderte Bild angewandt und auf das Ergebnis des Brightpassfilters, mittels Gaußfilter, ein Blur erzeugt. Der Blur wird in 4 verschiedenen Bildgrößen angewandt und am ende werden diese auf das ursprüngliche Bild addiert. Dadurch wird der Bloomeffekt erzeugt.
Man kann den Effekt vorallem bei Lichtquellen oder bei den Aufsammelbaren Objekten beobachten.

Referenzen:
http://en.wikipedia.org/wiki/Bloom_%28shader_effect%29 – generelles Verständnis
http://wesleygamedesigner.blogspot.co.at/2013/02/graphics-in-video-games-part-1-bloom.html Codebeispiel und zum Verständnis
http://prideout.net/archive/bloom/ - eine gute Erklärung wie das Prinzip von Bloom funktioniert
http://kalogirou.net/2006/05/20/how-to-do-good-bloom-for-hdr-rendering/ - Ebenfalls eine gute Erklärung und ein schönes Anschauliches Beispiel für die Umsetzung von Bloom. Grundsätzlich habe ich auch auf diese Art meinen Bloom implementiert.

HDR:

High Dynamik Range Rendering ist mittels einer Tonemap in Verbindung mit Float Vertex Buffern implementiert. Zuerst wird das gerenderte Bild in einen Float Buffer geschrieben. Aus der Textur wird CPU seitig die maximale sowie die durchschnittliche Helligkeit berechnet, diese werden im weiteren Schritt dazu verwendet eine Reichweite für die Tonemap zu definieren. Damit werden gerenderte Bilder bei denen sehr helle Bildbereiche auftreten etwas dünkler und dunkle Bilder etwas heller um so den Kontrast zu stärken.
Man kann dies vorallem beobachten wenn ein Übergang von einem Bild mit hellen Bildbereichen zu einem Bild mit hauptsächlich dunklen Bildbereichen stattfindet. Da der Charakter eine eigene Lichtquelle besitzt ist dies nicht immer eindeutig zu erkennen, man kann den Effekt beobachten wenn man direkt von oben hinabsieht und der Kopf die eigene Lichtquelle verdeckt. Weiters kann man den Effekt gut beobachten wenn ein helles Objekt einen größeren Teil des Bildes einnimmt z.b. wenn ein leuchtender Würfel groß genug auf dem Bildschirm ist, wird sich die Helligkeit des Bildes anpassen.
Leicht beobachten kann man die Effekte Allgemein indem man sie mittels Keybinding an und ausschaltet. (5 für Schatten, 6 für Normal Mapping, 7 für Bloom und HDR)

Referenzen:
http://de.wikipedia.org/wiki/High_Dynamic_Range_Rendering - Grundverständnis
http://en.wikipedia.org/wiki/Tone_mapping - Grundverständnis
http://transporter-game.googlecode.com/files/HDRRenderingInOpenGL.pdf – Gute Erklärung für den Vorgang des Tonemappings und HDR, nach diesem PDF habe ich mich teilweise auch bei der Implementation gerichtet
http://proquest.tech.safaribooksonline.de/book/programming/opengl/9781782167020/firstchapter Kapitel 5 – gute Erklärungen zur Implementation + Beispiele. Die Berechnung der Helligkeit des

Bildes, ist angelehnt auf die Beispiele aus diesem Buch.

Volumetrische Stencil-Shadows (z-fail):

Bei Shadow-Volumes werden pro Modell und Lichtquelle geometrische Objekte generiert welche die Volumen der Schatten repräsentieren. Dazu wird die Silhouette des Modells in Relation zur Lichtquelle berechnet und extrapoliert. Anschließend werden die Schatten-Objekte in den Stencil-Buffer gerendert um letztlich nur dort beleuchtete Modelle zu zeichnen wo das Licht der Lichtquelle auch hin kommen kann.
Die Kamera kann sich problemlos in einem Schattenvolumen befinden. Es werden von allen Lichtquellen Schatten geworfen. Es wird berücksichtigt ob Lichtquellen auf Schatten anderer Lichtquellen Licht werfen.

Referenzen:
Sehr hilfreich beim Verstehen von z-fail:
http://http.developer.nvidia.com/GPUGems3/gpugems3_ch11.html

Normal Mapping:

Normal-Maps speichern Normal-Vektoren für jeden Pixel in einer Textur. Beim Rendern eines Objekts kann im Shader diese Information genutzt werden um die Oberflächennormalen des Objekts und damit die Beleuchtung jedes Fragements entsprechend zu modifizieren. Dazu müssen die Normalen allerdings in Object-Space konvertiert werden. Wird haben nach einiger Recherche eine Methode implementiert die es erlaubt die dafür notwendige Matrix direkt im Shader zu berechnen ohne auf zusätzliche Tangenten-Informationen angewiesen zu sein, was den einsatz von Normal-Maps wesentlich flexibler gestaltet. Die Normal-Maps können sehr gut am Hauptcharakter begutachtet werden, vor allem wenn die Lichtverhältnisse sich ändern.

Referenzen:
Normal Mapping without precalculated Tangents:
http://www.thetenthplanet.de/archives/1180
NVIDIA Texture Tools for Adobe Photoshop (for creating normal maps)
https://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop

Zusätzliche Libraries und Referenzen:

- SDL Image Library - https://www.libsdl.org/projects/SDL_image/
- Abhängigkeiten der SDL Image Library.
- GLEW - http://glew.sourceforge.net/
- GLFW - http://www.glfw.org/
- Zum Lernen von OpenGL und GLSL wurde das OpenGL Redbook verwendet, daher kann es sein dass teilweise Ähnlichkeiten zu Beispielen dort bestehen - http://www.opengl-redbook.com/
- Um Bilder mit Hilfe von SDL Image zu laden wurde ausgiebig auf www.stackoverflow.com recherchiert, daher könnte dieser Code starke Ähnlichkeiten zu Beispielen dort aufweisen.
- Die Collision Detection und Response wurden nach folgendem Paper implementiert:
- [K. Fauerby, 25.07.2003, “Improved Collision detection and Response”, http://www.peroxide.dk]
- FreeType: Font engine und Rasterizer http://www.freetype.org/freetype2/
- Text-Rendern mit Hilfe von FreeType:
- http://en.wikibooks.org/wiki/OpenGL_Programming/Modern_OpenGL_Tutorial_Text_Rendering_01
- Verwendete Fonts:
- http://www.fontspace.com/gnu-freefont/freesans
- MipMaps:
- http://www.learnopengles.com/android-lesson-six-an-introduction-to-texture-filtering/
- http://gregs-blog.com/2008/01/17/opengl-texture-filter-parameters-explained/
- http://www.opengl.org/wiki/Common_Mistakes#Automatic_mipmap_generation

Models:
Modelle sowie die UV Maps wurden mit Maya erstellt.