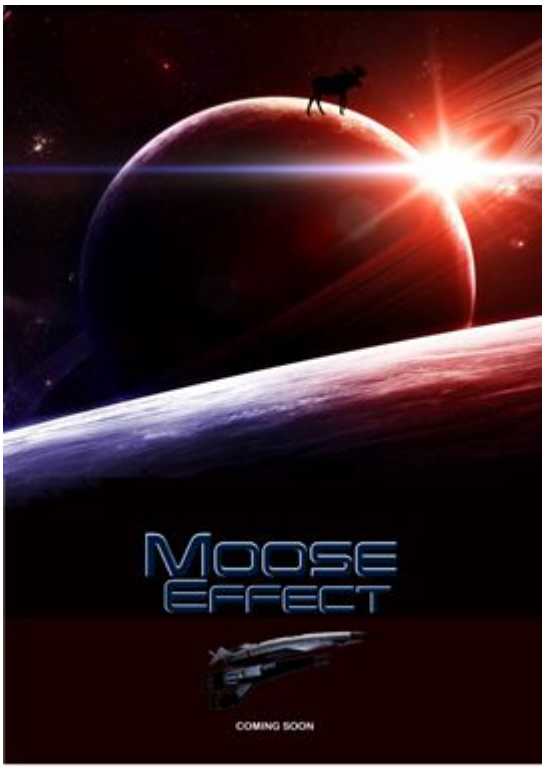
MOOSE EFFECT

Developer Team:
Novak Martin (1425662)
Gleichweit Julia (1325844)



Basic Gameplay

Der Spieler kontrolliert das Raumschiff von Mounty E. Python. Der Spieler muss dieses sicher durch die Meteoriten steuern und dabei herumfliegende Ressourcen einsammeln. Wurden genügend Ressourcen eingesammelt (5) kann das Raketenupgrade freigeschalten werden. Ist dieses freigeschalten hat der Spieler gewonnen.
Wenn das Raumschiff in einen Meteor fliegt nimmt es Schaden und der Spieler verliert eines seiner 3 Leben. Sind alle weg hat man verloren..

Der derzeitige Stand (d.h. Anzahl der Leben und Anzahl der Punkte) wird im Titel des Fensters aktualisiert.

Effekte

- **Shadow Maps (with PCF) (1.5) - Gleichweit**
  Es wurden zwei Tutorials (unter Referenzen) für die Implementation verwendet. Die Schatten wurden zusammengelegt mit dem TextureLightShader, d..h Objekte mit diesem Shader erhalten Texturen sind beleuchtet (DirectionalLight, Pointlight (blaue Kugel) und werfen Schatten bzw. befinden sich im Schatten. Die Szene wird zuerst aus der Sicht des Lichtes gerendert, um die ShadowMap zu erhalten und entscheiden mit diesen Werten, ob ein Schatten vorliegt oder nicht. (class ShadowMapShader, TextureLightShader). Wenn man über/unter einen Asteroiden fliegt solte man die Schatten gut sehen können.
- **Lens Flare (0.5) - Gleichweit**
  Angelehnt an die Referenz wurden drei Artefakte hinzugefügt (Werte und Positionen weitgehend durch Probieren entstanden), die sich jedoch vom Spieler wegbewegen. Um diese zu sehen wird die Kamera langsam um das LichtObjekt (blaue Kugel) bewegt. Normalerweise ist die Lichtquelle auch nur ein Billboard, hier jedoch eine blaue Kugel. Am besten ist der Effekt sichtbar, wenn man sich langsam nach hinten bewegt. Die Kugel befindet sich am Anfang hinter dem Spieler.
  Für das Anzeigen des Effektes werden einerseits die Werte aus der depthMap verwendet und andererseits der Blickwinkel auf die blaue Kugel.
- **Bloom (1.0) - Novak**
  Basierend auf dem verlinkten Tutorial wurde ein einfacher Bloom Effekt implementiert. Hierbei wird die Szene zuerst in zwei Framebuffer gerendert, einmal

Normal und einmal werden nur die Hellen Teile des Bildes extrahiert. Der helle Teil wird danach mehrmals durch einen Blur Shader gesendet, wo two pass Gaussian blur darauf angewendet wird. Das Resultat wird danach wieder mit der normalen Version des Renders kombiniert, was den klassichen Bloom Effekt erzeugt.

- **GPU-Particle System (+Compute Shader,Instancing) (1) - Novak**
- Das GPU basierende Partiklesystem wurde an eine implementation die auf github gefunden wurde angelehnt. Hierbei werden die originalen instancen der Partikel während des inits im computeShader angelegt. Simulation der Bewegung und das recyclen von “toten” partikeln passiert direkt im ComputeShader. Da die partikel nur eine Position haben, müssen sie beim Zeichnen erst von einem Geometry shader in Polygone umgewandelt werden. Das Erscheinen der Partikel kann unter Umständen etwas länger dauern, daher sollte man etwas warten, wenn man ein Objekt berührt.

Requirements

Complex Objects

Models wurden selbst mittels Blender erstellt. Das Flugzeug ist eine Kombinatione mehrere einzelner trivialer Objekte und bildet das geforderte komplexe Objekt.
Für das Laden der Modelle wird ASSIMP verwendet (ModelLoader.cpp). Der ModelManager verwaltet alle geladenen Modelle und verhindert das oftmalige Laden ein und desselben Modells. Alle Modelle wurden selbst mittels Blender erzeugt (Hilfestellung für die Asteroiden: How to Create a Realistic Asteroid using Blender - YouTube)

Die Asteroiden und das Raumschiff haben Texturen und verwenden den TextureLightShader. Die Ressource enthält nur ein Material (verwendet wird das default Jade Material). Die Texturen werden mittels eines TextureManager verwaltet, d.h. Texturen mit demselben Pfad werden nur einmal geladen. Die jeweiligen Objekte erhalten dann die entsprechende Referenz darauf.
Die gesamte Szene wird von einer Skybox mit Weltall CubeMap umgeben.

Welche Objekte werden beleuchtet oder texturiert?
Ein Pointlight befindet sich zu Beginn hinter dem Spieler in einer gewissen Höhe weiter oben. Zusätzlich wird die Szene mit einem Directional Light beleuchtet (dies ist dunkler gehalten aufgrund der Weltall-Atmosphäre). Das DirectionalLight im Spiel erzeugt die Schatten.
Die Bilder für die Texturen werden mittels FreeImage – Bibliothek geladen.
Alle Objekte in der Szene haben jeweils eine Textur.

Kollisionsabfrage
Für die Kollisionsabfragen ist Bullet zuständig. Mit dem Laden des Models wird zugleich auch die BoundingBox dieses erstellt. Diese Box wird dann von Bullet verwendet, um Kollisionen zwischen den Objekten zu erkennen.

Animated Objects

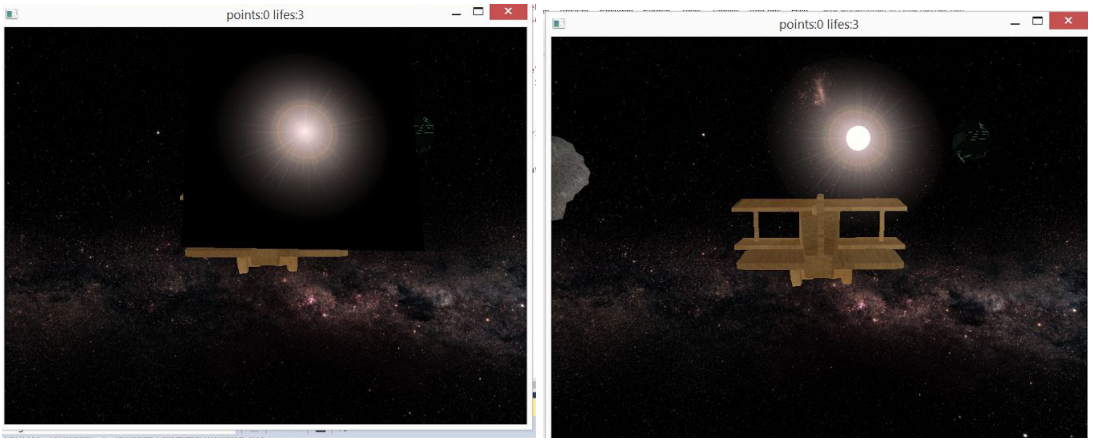
Ein Teil des Flugzeugs wurde mit einfacher hierarchischer Animation animiert. Diese Animation befindet sich am Kopfe des Flugzeuges: Der Propeller dreht sich.

View-Frustum-Culling

Wurde implementiert, mit dem ClipSpace-Ansatz von http://www.lighthouse3d.com/tutorials/view-frustum-culling/clip-space-approach-extracting-the-planes/ . Abfrage erfolgt im Screenspace, wenn zumindest bei einer plane alle Punkte außerhalb des Frustums sind ist dieses mit Sicherheit nicht sichtbar.

Experimenting with OpenGL

Wir verwenden auch VBO, VAO und FBOs. Beispielsweise in folgenden Klassen:
VBO: Skybox, Mesh
VAO: Mesh,
FBO: ShadowMap, Bloom
Blending wurde für Lensflare verwendet und ist am besten sichtbar, wenn sich das Flugzeug davor befindet.



Transparenz aus Transparenz an

Mip Mapping-Quality und Textur-Sampling-Quality wurde nicht implementiert, da nicht genau gewusst wurde wie/was dies umgesetzt werden sollte.

Controls
Grundsätzlich kann man sich frei durch die Szene bewegen. Das Spielerobjekt (hier das Flugzeug) befindet sich fix vor der Kamera und bewegt sich mit ihr. Die Implementierung der Kamera ist in lockedCamera.cpp zu finden (hier werden auch die entsprechenden Matrizen erzeugt). Die Steuerung der Kamera erfolgt mit der Maus. Bewegt man das Raumschiff nach vorne/hinten/links/rechts bewegt sich die Kamera automatisch mit.

Das Raumschiff wird mittels Tastatur gesteuert. Es kann in alle Richtungen bewegt werden, die -Blickrichtung kann mit der Maus verändert werden. Polling ist implementiert, d.h. es

kann gleichzeitig in zwei verschiedene Richtungen navigiert werden (z.B. nach vorne und links).

Raumschiff bewegen	**W/A/S/D**
Kamera drehen	Maus
Spiel beenden	**ESC**
Frametime ein-/ausschalten	**F2**
Wireframe ein-/ausschalten	**F3**
Bloom ein-/ausschalten	**F4**
Schatten ein-/ausschalten	**F5**
Frustum Culling ein-/ausschalten	**F8**
Transparenz ein-/ausschalten	**F9**

Libraries

- GLEW
  - Von Wiki „Tips und Tricks“ -> http://glew.sourceforge.net/
- GLFW
  - Von Wiki „Tips und Tricks“ -> http://www.glfw.org/
- GLM
  - Von Wiki „Tips und Tricks“ -> http://glm.g-truc.net/0.9.8/index.html
- ASSIMP
  - Von Wiki „Tips und Tricks“ -> http://assimp.sourceforge.net/
- FreeImage
  - Von Wiki „Tips und Tricks“ -> http://freeimage.sourceforge.net/
- Bullet Physics
  - Von Wiki „Tips und Tricks“ -> http://bulletphysics.org/wordpress/

Verwendete Tutorials / Papers / Bücher (Referencelist)

Shaderloader (shader.cpp)
https://learnopengl.com/#!Getting-started/Shaders

Objectloader (ModelLoader.cpp)
https://learnopengl.com/#!Model-Loading/Mesh
https://learnopengl.com/#!Model-Loading/Model

Camera (camera.cpp/lockedCamera.cpp)
https://learnopengl.com/#!Getting-started/Camera

Light (TextureLightShader.vert/.frag)
https://learnopengl.com/#!Lighting/Multiple-lights

Bullet Physics
Dokumentation im bullet/docs Ordner bzw.
http://www.bulletphysics.org/mediawiki-1.5.8/index.php/Documentation
https://www.raywenderlich.com/53077/bullet-physics-tutorial-getting-started

ShadowMap
http://ogldev.atspace.co.uk/www/tutorial42/tutorial42.html
https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping

Lens Flare
http://www.spieleprogrammierung.net/2010/05/opengl-3-lens-flares-city-lights-auf.html
http://www.emagix.net/academic/item/lens-flares

Bloom
https://learnopengl.com/#!Advanced-Lighting/Bloom

Partikle System
https://github.com/AlexEne/GL_Particles

SkyBox
https://learnopengl.com/#!Advanced-OpenGL/Cubemaps