Dokumentation Aircade

Spielprinzip

Es gibt eine Berglandschaft, mehrere Hindernisse und Checkpoints. Die Checkpoints müssen mit dem Flugzeug in der vorgegebenen Zeit durchflogen werden. Je schneller das passiert, umso mehr Punkte können erreicht werden. Wenn man gegen die Landschaft oder ein Hindernis fliegt, ist das Spiel vorbei. Ebenso, wenn man den Checkpoint nicht in der angegebenen Zeit erreicht.

Features

- Ein Richtungspfeil, welcher auf den nächsten Checkpoint zeigt
- Ein “Interface”, über welches man die Punkte, Zeit, Geschwindigkeit und den Schub sieht

Steuerung

Funktion	Tastatur	Gamepad
Steuerung umschalten	Q	—
Nachtmodus ein/aus	N	—
Neigung rauf	W	Linker Steuerstick
Neigung runter	S	
Neigung links	A	
Neigung rechts	D	
Geschwindigkeit erhöhen	R	Rechter Steuerstick (rauf/runter)
Geschwindigkeit senken	F	
Lautstärke erhöhen	BILD UP	—
Lautstärke senken	BILD DOWN	—
Spiel pausieren	ESC	—
Spiel beenden (Pausemodus)	BACKSPACE	—

Design und Datenstruktur

Alle Objekte werden entweder texturiert oder deren Meshes enthalten Farbinformationen. Komplexere Objekte können auch texturiert und gefärbt sein. Jedes Objekt wird von einem SceneObject verwaltet oder davon abgeleitet. Im SceneObject werden das Modell, die Shader und die Transformationsmatrix gespeichert. Das Modell beinhaltet neben den Meshes noch die Materialien, in welchen die Textur und die Farbinformationen (Diffus, Specular, Shininess) gespeichert werden.

Beleuchtet wird die ganze Szene durch eine punktförmige Lichtquelle oder ein Spotlight.
Die Lichtquelle hat eine fixe Position, das Spotlight befindet sich auf der Spitze des Flugzeugs.

Komplexe Elemente

Komplexe Modelle sind, neben dem Terrain, beispielsweise die Bäume und Häuser.

Animierte Objekte

Diese Objekte wurden hierarchisch erstellt, indem das "Parent-Element" mehrere "Child-Elemente" besitzt und diese bei Transformationen mit updatet.

View-Frustum-Culling

View-Frustum-Culling wurde mit Hilfe von Boundingboxen und Bounding-Spheres implementiert. Dazu wurden jeweils eine Sphäre und eine Box verwendet und je nach Form des Objektes eine der beiden für die Überprüfung herangezogen. Das Terrain und der Spieler wurden von der Überprüfung ausgenommen, da der Spieler immer sichtbar ist und das Terrain so groß ist, dass es kaum vorkommt, dass es nicht in der Boundingbox liegt. Für Sphären wurde der Abstand der Mittelpunkte für die Überprüfung verwendet (Checkpoints). Bei der Boundingbox wurde anhand der Clipping-Planes des Frustums überprüft, ob die Box überhaupt im Frustum liegt. Wenn das der Fall ist, wurde überprüft, ob alle Punkte des Frustums außerhalb der Boundingbox liegen, da bei großen Objekten die Clippingplane zu Problemen führen würde. Es wurde hautsächlich das Tutorial von Lighthouse verwendet und für die Boundingboxen zusätzlich das Tutorial von iquilezles.

Kollisionserkennung

Die Kollisionserkennung wurde nur anhand der BoundingBox durchgeführt. Dazu wurden die Boundigboxen des Spielers und des Objektes, welches Überprüft wird, in das Modelkoordinatensystem transformiert. Dort wurde dann, ähnlich wie beim View-Frustum-Culling, überprüft, ob die Punkte des Objektes außerhalb der Boundingbox des Spielers liegen.

Für das Terrain und Wasser wurde eine eigene Kollisionserkennung implementiert. Diese Überprüft einerseits, ob die Flughöhe über dem Wasser liegt und andererseits, ob die Flughöhe über der Terrainhöhe liegt, über welcher sich das Flugzeug befindet.

Model Loader

Das Laden der Modelle wird unter Verwendung von Assimp durchgeführt. Das Modell kann mehrere Farben, Texturen und Meshes besitzen. Die geladenen Informationen werden dann in der Model und Mesh Klasse verwaltet und durch diese an OpenGL übergeben.

Effekte und Implementierung

Die unter “Experimenting with OpenGL” geforderten Punkte wurden alle implementiert (Mit Ausnahme der optionalen Hilfefunktion). Die Änderungen werden in der Console ausgegeben.

Implementierte Effekte:

- *Water (+Fresnel-Shading,Normal Mapping)*
- *Water Reflection*
- *Water Refraction*
  Das Wasser besitzt eine Normalmap sowie eine Distortionmap. Schattiert wird die Oberfläche mit Hilfe der Normalmap. Durch verändern der Texturkoordinaten mit der Distortionmap, wird eine bewegte Wasseroberfläche mit Wellen erzeugt. Für die Reflexion wird die gesamte Szene mit einer Clippingplane abgeschnitten und von einer transformierten Kameraposition aus in ein Framebuffer Object gezeichnet. Ähnliches geschieht auch bei der Brechung, nur dass diesmal lediglich Elemente der Szene welche sich unter der Clippingplane befinden in ein weiters Framebuffer Object gezeichnet werden. Die beiden daraus resultierenden Texturen werden im Fragment-Shader mit der Normalmap und der Distortionmap mithilfe von Fresnel-Shading kombiniert und erzeugen so die endgültige Farbe eines Fragments.

- *Spotlight*
  Das Spotlight wurde mit Hilfe des Light Casters Tutorial implementiert. Die Implementierung selbst wurde komplett im Fragmentshader eingebaut. Dort wurde ein Struct mit allen relevanten Parametern (Position, Richtung, Farbe, innerer/äußerer Kreis, Lichtabschwächung) angelegt. Die Position, Richtung und Farbe sind variabel und werden dem Shader übergeben. Die Größe des Spotlights wird über den inneren und äußeren Kreis geregelt, wobei die Intensität innerhalb des inneren Kreises 1 ist und im äußeren Kreis linear interpoliert wird. Damit das Licht nicht durch die komplette Scene leuchtet, wurde es über drei Koeffizienten (konstant, linear, quadratisch) abgeschwächt. Das Spotlight ist nur im Nachtmodus sichtbar (Taste N).

- *Shadow Volumes*
  Für die Shadow Volumes werden in einem Vorverarbeitungsschritt die Nachbarschaftsbeziehungen der Dreiecke des Modells ermittelt und deren Vertices geordnet in einem eigenem Buffer abgelegt.. Der Hauptteil der Berechnung findet im Geometry-Shader statt, welcher dynamisch das Volumen erstellt. Es wird mit GL_TRIANGLES_ADJACENCY gezeichnet, damit im Geometry-Shader die Nachbarn des Dreiecks übergeben werden. Dort wird die Silhouette ermittelt, indem überprüft wird, ob ein Dreieck und dessen drei Nachbarn in Richtung Lichtquelle schauen. Sieht das Dreieck zum Licht und ein Nachbar nicht, handelt es sich bei der betreffenden Ecke um eine Außenecke. Außenecken werden in Lichtrichtung zu einem Rechteck extrudiert, Zusätzlich berechnet der Geometry-Shader auch noch ein Front und Back-Cap. Das Volumen selbst wird in zwei Durchläufen mit dem Z-Fail-Test gezeichnet. Es werden mit Hilfe von Backfaceculling die vorderen und hinteren Dreiecke getrennt in den Stencilbuffer gezeichnet und dieser für jeden Pixel verringert beziehungsweise erhöht. Der Schatten wird danach gezeichnet, indem nur die Pixel gezeichnet werden, für welche ein Wert ungleich null im Stencilbuffer steht.
  Verwendete Links zum Erstellung der Shadow Volumes:
  http://ogldev.atspace.co.uk/www/tutorial39/tutorial39.html
  http://ogldev.atspace.co.uk/www/tutorial40/tutorial40.html

Verwendete Tools

Zum Erstellen der Modelle wurde Blender verwendet.

Verwendete Libraries (32bit binaries)

- GLFW 3.1.2: Für das Window-handling und die Tastatureingaben
- GLEW 1.13.0: Für zusätzliche OpenGL-Funktionalitäten
- GLM 0.9.7.4: Für mathematische Berechnungen und Datenstrukturen (Vektoren, Matrizen)
- Assimp 3.2: Zum Laden der Modelle in den Hauptspeicher
- FreeImage 3.17.0: Zum Laden der Texturfiles
- cmake 3.5.1: Zum Erstellen der Binaries
- Freetype 2.3.5-1: Zum Anzeigen von Text
- FMOD: Für die Soundwiedergabe