Zweite Abgabe

# Flying Circus

16.06.2014

Technische Universität Wien
SS 2014
David Krywult 1147460
Daniel Witurna 1125818

## Umsetzung der Requirements

### Gameplay

Bei Flying Circus geht es darum der oder die Schnellste zu sein und alle Spielewelten zu meistern. Für jedes Level wird ein Highscore abgespeichert der innerhalb einer Spielesession beliebig oft unterboten werden kann. Sobald innerhalb einer Welt das Ziel erreicht wurde, kann das nächste Level geladen werden. Wenn alle Level absolviert sind, kann auf der Jagd nach Rekorden mit dem ersten Level wieder fortgesetzt werden.
Unsere Figur kann durch alle drei Dimensionen bewegt werden und sogar Loopings ziehen. Enge Passagen können bei entsprechender Geschwindigkeit im Messerflug bestritten werden. Im Falle eine Kollision mit dem Terrain hat der Spieler oder die Spielerin die Möglichkeit mittels ENTER-Taste an den Start zurückkehren.
Implementiert wurde das Flugmodell mit den Manipulationsfunktionen unseres Szenegraphen. Verschiedenen Parametern werden in der Update-Methode unseres Spielers passend zu der Frametime berechnet. Auch Steuerungseingaben werden im Spielerknoten (PlayerNode) behandelt.

### Effects

Als Schattierungstechnik hatten wir Anfangs Shadow-Mapping im Auge, welches wir allerdings nicht geschafft haben zu implementieren. Als unserer Meinung nach gelungenen Ersatz setzen wir nun auf **Lightmapping** mit separaten Texturen, die beim Laden eines Levels berechnet werden. Um ein besseres Geschwindigkeitsgefühl vermitteln zu können wird **Motion-Blur** auf die Umgebung des Spielers angewendet. **Spotlights** an der Spitze des Tiers sowie an den Flügeln erlauben eine bessere Einschätzung der Lage und des Abstands zu den Wänden. Um noch etwas Stimmung und leben in die Welt zu bringen benutzen wir ein **GPU-Particle-System** um tausende Schneeflocken zu erzeugen.

### Complex Objects

In Flying Circus kommen zwei Arten von komplexen Objekten vor: Modelle und das Terrain. Das Terrain wird über einen selbst geschriebenen Heighmap-Loader erstellt. Dabei werden in der Heightmap zusätzlich zu der Höhe auch Informationen wie die Position von Lichtern und dem Ziel-Tor gespeichert.
Die anderen Modelle sind im Wavefront-OBJ-Format gespeichert und werden mittels Assimp geladen. Alle Modelle werden mittels Lambert-Phong-Shading gezeichnet.

Die Modelle wurden alle mit einer Kombination von DOGA-CGA und Milkshape3D erstellt.

### Animated Objects

Um unserer Spielfigur mehr Leben einzuhauchen, trennten wir virtuell zuerst die Extremitäten ab um diese wiederum zusammenzufügen und animieren zu können. Je nach momentanen Benutzereingaben schwingen Arme und Beine in eine bestimmte Richtung. Auch die Tragfläche wurde in zwei Teile geteilt, welche je nach aktueller Fluglage eine andere Position einnehmen. Durch die Verwendung eines eigens geschriebenen Szenegraph ging dieser Punkt leicht von der

Hand. Im Spielerknoten wird je nach aktueller Tasteneingabe ein Motionwert erzeugt, der auf die Extremitäten angewandet wird.

### View-Frustrum-Culling

Im ersten Schritt werden von der Kamera anhand der Parameter, die zur Erzeugung der Projection-Matrix verwendet wurden, die Eckpunkte des View-Frustrums berechnet. Aus diesen lassen sich dann mit einfacher Vektormathematik die Ebenen extrahieren, die das View-Frustrum eingrenzen.
Für jedes Objekt wird dann berechnet, ob sich der Ursprung des Objekts im Frustrum befindet. Ist dem nicht so, dann wird berechnet, ob die Entfernung des Mittelpunkts vom Frustrum maximal den Radius des Objekts beträgt. Ist diese Entfernung größer als der Radius, dann wird das Objekt nicht gezeichnet.
Der Radius wird automatisch beim Laden des Objekts berechnet. Dazu wird das Fragment gesucht, dass am weitesten vom Ursprung des Objekts entfernt ist. Der Abstand vom Ursprung des Objekts zu diesem Fragment ist dann der Radius.

### Transparency

Um das Sichtfeld aus der Perspektive der Kamera zu erweitern haben wir uns dazu entschieden, die Flügel teilweise transparent zu gestalten. Ein direkter Vergleich des Unterschieds kann mittels der F9-Taste, die die Funktion aktiviert oder deaktiviert, getätigt werden.
Um das transparente Zeichnen zu ermöglichen gibt es einen eigenen Render-Pass, der nach allen anderen Passes (auch nach dem Motion-Blur-Pass) ausgeführt wird. In diesem Render-Pass werden nur transparente Objekte gezeichnet. Da sich bei uns die transparenten Objekte nicht überlappen können, ist es nicht notwendig, die Objekte der Tiefe nach zu sortieren.

### Experimenting with OpenGL

Vertex Buffer Objects und Vertex Array Objects werden so gut wie bei jeder Benutzung eines Shaders verwendet. Beispielhaft kann die Benutzung davon (nicht nur) in der Methode "createAndUploadVBO" innerhalb von "Surface.cpp" angesehen werden.
Frame Buffer Objects verwenden für die Berechnung der Lightmaps sowie für den Motion-Blur-Effekt. Für Mip-Mapping und Texture-Sampling-Quality haben wir einen Handler gebaut, der vor jedem Benutzen einer Textur aufgerufen wird und je nach Einstellungen die Eigenschaften setzt. Mittels den Tasten F4 und F5 können die Texture-Einstellungen umgeschalten werden.

## Features

- Eigene Collision Detection
- Eigenes Flugmodell mit Anlehnungen an die echte Luftfahrt
- Triplanares Mapping für das Terrain
- Schnee und Tageszeiten-Effekte
- Sehr freies Beleuchtungssystem

- Leicht um weitere Levels erweiterbar (alles was es braucht ist eine Heightmap, kann mit jedem besseren Bildeditor erstellt werden)
    - Roter Farbkanal für Höhe
    - Grün über einem Wert von 250 für Fackeln
    - Blau über einem Wert von 250 für das Ziel
    - Blau bei einem Wert von exakt 180 für einen Schnee-Emitter
    - Schatten werden während dem Laden eines Levels berechnet

## Beleuchtete Objekte

Alle Objekte, die keine Partikel sind, werden auf die gleiche Art beleuchtet. Es gibt ein recht flexibles Licht-System, dass es ermöglicht, eine beliebige Kombination aus Ambient-, Directional-, Point- und Spot-Lights als Nodes in der Spiel-Welt zu platzieren. Diese Lichter beleuchten dann alle Objekte in ihrem Einflussbereich rein über den Diffuse-Term der Lambert-Beleuchtung. Einzig das Ambient-Light beleuchtet über den Ambient-Term statt dem Diffuse-Term.
Die Objekte besitzen jeweils eine Diffuse-Textur, die die Farbe beinhaltet. Vertex-Farben gibt es nicht.
Das Terrain wird mit insgesamt fünf Texturen texturiert. Zuerst wird die Lightmap eingefügt. Diese beinhaltet vorberechnete Licht und Schatten. Dann wird die Diffuse-Textur dreifach genommen, um damit triplanares Textur-Mapping zu realisieren. Dabei wird statt uv-Koordinaten uvw-Koordinaten verwendet, wobei die w-Koordinate die Höhe des Terrains an dieser Stelle enthält. Dann wird jeweils ein Sample der Textur an den Koordinaten uv, vw und uw genommen. Je nach dem Normalvektor an dieser Position wird zwischen diesen drei Textur-Werten interpoliert. Auf diese Weise werden unschöne Streckungen an den steilen Heightmap-Wänden vermieden. Schlussendlich wird dann noch eine Schnee-Textur darübergelegt, wieder abhängig vom Normalvektor. An steilen Flächen gibt es demnach weniger Schnee, wärend es an flacheren Stücken mehr Schnee gibt.

## Externe Bibliotheken

- lodepng (http://lodev.org/lodepng/)
- assimp (http://assimp.sourceforge.net/)
- GLEW/GLFW/GLM

## Implementierung der Effekte

### Lightmapping + Separate Textures + In-Game Calculation

Um schönere Lichtverhältnisse zu bekommen verwenden wir Lightmaps. Darin werden die Beleuchtungswerte und Schatten der statischen Lichtquellen (bei uns das Ambient-Light und das Directional-Light) vorberechnet, so dass diese rechenintensiven Arbeiten nicht während dem Spiel passieren müssen.
Die Lightmap wird in einer seperaten Textur gespeichert. Diese wird im Beleuchtungs-Pass wie eine eigene Lichtquelle dazu addiert.

Findet sich eine Lightmap-Textur mit dem passenden Namen (z.B. “map1lightmap.png”) im Maps-Ordner, so wird diese geladen. Ansonsten wird die Lightmap beim Laden der Heightmap vom Spiel generiert.

### GPU Particle System

Das GPU-Particle-System wurde auf Basis zweier Tutorials (http://ogldev.atspace.co.uk/www/tutorial28/tutorial28.html, http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/) implementiert. Das Ergebnis der Implementierung sollten Kondensstreifen an den Flügeln des Spielers sein. Da es bei diesem Fall aber zu unterschiedlichen Problemen gekommen ist, haben wir uns für den Wettereffekt Schnee entschieden. Unsere Implementierung besteht einerseits aus dem Billboardshader, der für die konkrete Darstellung jeder einzelnen Schneeflocke verantwortlich ist, und aus den Particleshadern, welche im Geometryteil zahlreiche Partikel erzeugen. Um Schnee in unser Spiel zu bringen, fügen wir einen EmitterNode in unseren Szenengraph ein. Bei einem Update-Aufruf (der vor jedem Draw erfolgt) werden alle Partikel eines Emitternodes auf der GPU durchlaufen. Anfangs ist nur ein Partikel vom Typ "EMITTER_PARTICLE" vorhanden. Wird während einem Durchgang auf der GPU dieses Emitterpartikel behandelt, werden neue "SIMPLE_PARTICLE" erzeugt. Diese simplen Partikel werden dann zu einer tatsächlichen Schneeflocke gerendet. Wird bei einem Durchgang im Geometryshader ein einfaches Partikel behandelt, wird passend zu dessen aktueller Geschwindigkeit und Position eine neue Position errechnet. Hat das Partikel das Maximalalter überschritten, wird es zurück an den Anfang gesetzt. Auf der CPU-Seite wird für jeden Update-Aufruf ein Vektor mit drei zufälligen aber auf einen gewissen Bereich begrenzten Werten erzeugt. Dieser Vektor legt die Initiale Geschwindigkeit der einfachen Partikel fest und gewährleistet dadurch beim Erzeugen eine gewisse Streuung.

### Motion Blur + Radial Blur

Für den Motion Blur werden die nicht-transparenten Objekte in zwei FBO-Texturen gerendert. Die erste Textur beinhaltet dabei die normalen Farben, wärend die zweite Textur die Positionsänderungen jedes Pixels in Screen Space-Koordinaten seit dem letzten Frame beinhaltet.
In einem zweiten Render-Pass wird zuerst ein Blur-Vektor erstellt, der aus den Positionsänderungen (Motion Blur) und einem Vektor, der vom Zentrum des Bildschirms weg zeigt (Radial Blur) besteht. Die Länge des Radial Blurs ist von der Geschwindigkeit des Spielers abhängig. Ist der Spieler zu langsam, dann wird der Radial Blur komplett entfernt.
Die Farb-Textur aus dem letzten Render-Pass wird dann mehrfach entlang dieses Blur-Vectors gesampled und zwischen diesen Samples interpoliert, um einen Blur-Effekt entstehen zu lassen.

### Spotlight

Die Spot-Lights bauen auf den normalen Point-Lights auf, haben aber zusätzlich zu Helligkeit und Reichweite auch eine Richtung und zwei Winkel.
Für jedes Fragment, dass sich in der Reichweite des Spotlights befindet, wird ein Vektor erstellt, der von dem Licht zum Fragment geht, und dann das Dot-Product von diesem Vektor und dem

Richtungsvektor des Spotlights erstellt. Ist das Dot-Product kleiner als Kosinus des inneren Winkels, dann scheint das Licht mit voller Stärke. Ist das Dot-Product zwischen dem Kosinus des inneren und dem Kosinus des äußeren Winkels, dann wird zwischen voller Stärke und keinem Licht interpoliert, je nach dem wie nahe das Dot-Product dem Kosinus des inneren oder äußeren Winkels ist. Ist das Dot-Product größer als der Kosinus des äußeren Lichts, dann wird das Fragment nicht beleuchtet.

### Modeltools

Im ersten Schritt wurde DOGA CGA verwendet, da es sehr leicht zu bedienen ist. Leider kann DOGA CGA keine .obj-Dateien erstellen. Auch halten sich die Materialien auch nur schlecht exportieren. Deswegen haben wir Milkshape verwendet, um die Materialien nachzubearbeiten und die Objekte nach .obj zu exportieren.

## Steuerung und Ein-/Ausschalten von Funktionen

- WASD / Cursor-Tasten: Flugsteuerung
- F: Nebel an/aus
- L: Scheinwerfer an/aus
- N: Tag/Nacht
- Enter: Setzt den Spieler zurück
- F1 - keine Funktion
- F2 - Frame Time on/off
- F3 - Wire Frame on/off
- F4 - Texture-Sampling-Quality: Nearest Neighbor/Bilinear
- F5 - Mip Mapping-Quality: Off/Nearest Neighbor/Linear
- F6 - Motion Blur on/off
- F7 - keine Funktion
- F8 - Viewfrustum-Culling on/off
- F9 - Transparency on/off

## Steuerung per Joystick oder Gamepad

- Joystick: XY-Achsen für die Flugsteuerng
- Gamepad: Linker Analogstick für die Flugsteuerung
- Taste 1: nächstes Level (funktioniert nur am Ende eines Levels)
- Taste 2: Setzt den Spieler zurück

## Bekannte Bugs

- Wenn eine Map auf der linken und/oder rechten Kante der Heightmap einen Wert größer als 0 besitzt, dann wird eine Art Decke über das Level gerendert. Dies kommt aufgrund von einem Bug im Heightmap-Loader