

CG17 / Gruppe 2

Quidditch-Academy

Stephen Agyemang, 1047450
Wolfgang Gundacker, 8908802

Spielstart ist mit folgenden Argumenten möglich

-h, --height	setzt die vertikale Auflösung
-w, --width	setzt die horizontale Auflösung
-r, --refresh-rate	legt die Bildwiederholrate fest
-f, --fullscreen	legt für die Ausführung den Vollbildmodus fest
-h, --help	zeigt die Commandline-Optionen

Requirements

Gameplay

Auf einem Besen sitzend beginnt man das Spiel. Man nimmt den Quaffle auf und muss versuchen ihn durch die Ringe zu schießen. Ein Schuß wird mit einem Mausklick ausgelöst. Das Fliegen ist nur im freien Raum möglich, von den Begrenzungen und Stangen prallt man ab.

Complex Objects

Zum Laden von komplexen Objekte (Stadion, Tore, Spielball) wurde die Open Asset Import Library „Assimp“ verwendet. Die hierarchische Struktur aus der 3D Asset Szene wird in eine Tree Structure von SceneObjects übertragen.

Animated Objects

Der Ball fällt nach dem Wurf und kollidiert mit der Umgebung. Das Aufheben und Mitnehmen des Balls funktioniert durch einfaches Parenting. Sowohl die Spielerkamera und der Ball erben von der grundlegenden SceneObjects Klasse, somit wird die Ballposition ohne zusätzliche Arbeit an die Spielerposition angepasst. Zusätzlich gibt es noch hierarchisch animierte Hindernisse vor dem Tor, die das Erzielen von Punkten erschweren.

Frustum Culling

Frustum Culling wurde implementiert. Die Auswirkungen kann man überprüfen, indem man mit F1 den Hilfetext einblendet und mit F8 die Funktion ein bzw ausschaltet. Die Anzahl der gerenderten Objekte wird unterhalb des Hilfetextes angezeigt. Das Frustum Culling betrifft nur klassische Geometrie Objekte, Partikel, welche ohnehin sehr einfache Geometrie besitzen, werden nicht gecullt.

Experimenting with OpenGL

Blending, Mip Mapping und Texture-Sampling-Quality können wie gefordert durchgeschaltet werden (siehe unten „Steuerung im Spiel“ bzw. Helptext F1 im Spiel)

Freely movable camera

Da man das Spielfeld immer aus Spielersicht bewegt sich die Kamera automatisch mit der eigenen Position mit.

Moving Objects

Der Spielball kann aufgenommen und geworfen werden. Nach 3 Sekunden erscheint er wieder über der Spielfeldmitte.

Texture Mapping

Alle Objekte bis auf den Ball und Partikel sind texturiert. Es können Texturen für, Diffuse und Specular vorhanden sein. Die Texturen werden über die 3D Assets geladen das Setup erfolgt automatisch. Untexturierte Objekte (der Ball) haben ein Default-Material mit einer durchgängigen Diffuse / Specular Farbe.

Simple Lighting and Materials

Sowohl eine Lights- als auch eine Materialklasse sind implementiert. Derzeit wird eine Directional Lichtquelle verwendet. Wie im vorigen Punkt schon erwähnt wird bei Objekten ohne Texture eine Default-Material verwendet. Es wird das Blinn-Phong Beleuchtungsmodell verwendet.

Steuerung im Spiel

1. Tastatur

w	Beschleunigen nach vorne
s	Beschleunigen nach hinten
linke Maustaste	Ballwurf sofern der Ball mitgeführt wird
ESC	beendet das Spiel (derzeit ist eine max. Spieldauer von 30000 Sek. eingestellt)
F1	zeigt die Function-Key Belegung am Schirm an
F3	schaltet in den Wireframe-Modus und zurück
F4	Ändert die Texture-Sampling-Quality (Nearest → Bilinear →)
F5	Ändert die Mipmapping-Quality (off → Nearest → Linear →)
F6	startet ein Feuerwerk
F8	schaltet Frustum-Culling ein und aus (zeige Anzahl gerenderter Objekte mit F1)
F9	schaltet Blending ein und aus (sichtbar an Mond und Lensflares)

2. Maus

mit den Mausbewegungen steuert man die Blickrichtung.

Umsetzung

Es wurde eine Scenegrph Repräsentation der Szene implementiert. Objekte können durch parenting hierarchisch verknüpft und miteinander transformiert werden. Ein Modelloader basierend auf Assimp (<http://assimp.sourceforge.net/>) kann Szenen importieren und hierarchisch aufbauen. Als Image Loader wurde SOIL (<http://www.lonesock.net/soil.html>) verwendet. Als Matrix Library kam glm (<http://glm.g-truc.net/0.9.8/index.html>) zum Einsatz. Für die Textanzeige wurde Freetype (<https://www.freetype.org/>) gewählt.

Effekte

Es wurden Shadow-Mapping, Lensflares und GPU-Partikles.

Bei **Shadow Mapping** wird aus der Perspektive des Directional Light mittels orthographischer Position in eine Depth Map gerendert. Im darauffolgenden Renderpass wird die Depth Map verwendet um zu überprüfen ob das Objekt hinter einem anderen Objekt liegt. Es wird einfaches Depth Map PCF mittels Average Filtering umgesetzt.

Unsere **Lensflares** werden als einfache Quads mit Textur additiv auf den Rest der Szene geblendet. Für die 2D Position wird im Hauptprogramm die Screenspace Position der Lichtquelle (Mond) berechnet. Anschließend wird eine Linie von der Lichtquelle durch die Bildmitte gelegt. Entlang dieser Linie werden die einzelnen Flare Elemente positioniert und mit einem einfachen Shader gerendert. Die Distanz der Lichtquelle von der Bildmitte bestimmt die Intensität der Flares.

Die Mond wird ebenfalls als Quad mit Textur gerendert, wobei sich der Quad immer dem Spieler zuwendet und somit nie perspektivisch verzogen wird. Es wird eine Occlusion query durchgeführt wobei die tatsächlich gerenderten Fragments des Mond Quads gezählt werden. Erst wenn ein gewisser Grenzwert an sichtbaren Mond Fragments überschritten wird, werden die Flares gerendert.

Fällt ein Tor wird von der Mitte des Spielfelds ein **Partikelfeuerwerk** gestartet, das steil nach oben startet und nach einigen Sekunden in großer Höhe explodiert, wobei in mehreren Stufen neue Partikel gestartet werden. Die Partikelparameter werden durch Compute Shader jeden Frame upgedatet. Dabei kommt ein doppeltes Read/Write - Buffersystem zum Einsatz. Die vorigen Partikelparameter werden aus dem einen Buffer ausgelesen neu berechnet und anschließend in den zweiten Buffer geschrieben. Welcher Buffer ausgelesen / beschrieben werden soll, ändert sich in jedem Durchgang. Zudem wird die Zahl der lebenden Partikel jeden Frame festgehalten um im nächsten Updatevorgang nicht über den gültigen Bereich im Buffer zu hinaus zu laufen. Die aktuelle Partikelzahl wird auch für das Instancing verwendet um nur lebende Partikel zu rendern. Die Partikelgeometrie besteht aus einer einfachen Diamantform mit einer Farbe. In der Simulation wird Schwerkraft simuliert und Geschwindigkeiten neuer Partikel mit einer zufälligen Funktion berechnet.

Den Bloom Effekt setzen wir mittels mehrerer Framebuffer Objects um. In einem ersten Renderpass werden das Bild sowie ein Schwellwertbild in verschiedene Texturen gezeichnet. Das Schwellwertbild ist weiß an jenen Stellen an denen die Helligkeit im Originalbild über dem Schwellwertbild. Das Bild wird anschließend mehrmals mit einem Gaußfilter Shader weichgezeichnet und anschließend additiv mit dem Originalbild zusammengerechnet.

Verwendete Tools

Die Modelle wurden mit Blender und Cinema4D bearbeitet. Für Bilder wurde Photoshop verwendet.

Quellen

Das Stadion stammt von dieser Quelle

(<http://www.mediafire.com/file/e2narlumkv1165h/Quidditch+Stadium.zip>). Einige Codepassagen basieren auf Tutorials von (<https://learnopengl.com/>), wurden jedoch weitgehend verändert und erweitert. Die betreffenden Stellen sind annotiert.