Marvin Weisbrod
744411
Yannik Hupfer
770088

Shaderkonzepte für Spieleentwicklung
Dokumentation zum Praktikumsprojekt
Collector Drone

02.12.2022

1 Konzept

Die Idee für dieses Projekt war, ein kleines Spiel umzusetzen, bei dem der Benutzer mit einem Quadrocopter durch eine Szene mit vielen Hochhäusern fliegt und Collectibles einsammelt. Das Projekt sollte auf der vorhandenen Codebasis des Praktikums aufsetzen und dieses um diverse Engine- und Rendering-Features erweitern.

2 Gameplay

Zu Beginn befindet sich die Drone in der Mitte auf dem Boden der Szene. Ein Timer fängt direkt an, die Zeit zu messen. Der Spieler muss nun so schnell wie Möglich alle Torten, welche in der Szene verteilt sind, einsammeln. Sobald alle Torten eingesammelt wurden, wird die Zeit gestoppt. Der Spieler kann jederzeit die Szene zurücksetzen um von Vorne zu beginnen.

2.1 Steuerung

Zusätzlich zum Steuern der Drone, ist es auch möglich, sich einzelne Aspekte des Deferred Renderings anzeigen zu lassen, wie sich folgender Tabelle entnehmen lässt:

Eingabe	Funktion	Eingabe	Funktion
W	Nach Vorne	1	Dronenkamera
A	Nach Links	2	Freie Kamera
S	Nach Hinten	3	Rendering: Default
D	Nach Rechts	4	Rendering: Deferred Only
Q	Links Rotieren	5	Rendering: Position
E	Rechts Rotieren	6	Rendering: Normals
L.Shift	Nach Oben	7	Rendering: Albedo
L.Ctrl	Nach Unten	8	Rendering: Specular
Space	Level Zurücksetzen	9	Rendering: Emissive
T	Tag/Nacht Wechseln	0	Rendering: Shine

3 Features

3.1 Drone

Die Drone kann per Tastatursteuerung bewegt werden. Sie beschleunigt in die gewählte Richtung, bis sie ihre Maximalgeschwindigkeit erreicht. Wenn keine Eingabe auf einer Eingabeachse registriert wird, steuert die Drone von selbst der Bewegung entgegen, um von alleine abzubremsen. Die Drone wird abhängig von ihrer aktuellen Beschleunigung animiert. Dies hat keinen Einfluss auf das Flugverhalten, sondern sorgt nur für eine ansprechende Visualisierung der Beschleunigung. Als Modell wurde eine Drone aus einem Asset Pack [2] aus dem Unity Asset Store verwendet.

3.2 Stadt mit Collision

Um die Implementierung der Collision möglichst simpel zu halten, werden axis-aligned bounding boxes (AABBs) verwendet. Dies setzt voraus, dass möglichst Box-ähnliche Gebäude verwendet werden. Daher wurde ein einfaches low-poly Gebäude[4] verwendet und in der Szene verteilt. Nachdem die Drone einen neuen Bewegungsvektor berechnet hat, wird die neue Position mit allen AABBs verglichen und die Drone bei Kollision gestoppt. Ein Abprallen der Drone wäre durch eine kompliziertere Berechnung unter Einbeziehung aller involvierten Rotationen möglich gewesen, jedoch wurde aus Zeitgründen vermieden. Die Kollision mit dem Boden kann jedoch auf solche Weise einfach berechnet werden, da weder Boden noch Drone an X und Z Achse rotiert werden. Dadurch ist es möglich, am Boden entlang zu rutschen, wie es auch bei Gebäuden wünschenswert gewesen wäre.

3.3 Collectibles

Als Collectibles wurde das Modell einer Torte[3] verwendet, welches sich durch Animation des Transforms dreht und hüpft. Für die Collision werden wie auch für die Stadt AABBs verwendet.

3.4 Text Rendering

Zur Anzeige von Texten wurde eine Font Map erstellt, welche alle benötigten Zeichen enthält. Diese wurde mit einem Bitmap Font Generator[1] erstellt und enthält einen monospace font, dessen Zeichen ein Seitenverhältnis von 1:2 haben. Diese Fontmap wird als Textur an den Textrenderer gegeben, welcher ein Rechteck (zwei Dreiecke) zeichtnet und dieses anhand der Anzahl der Chars im Text und der Textgröße in Vertex-Shader entsprechend streckt. Im Fragment-Shader wird über die UV-Koordinaten und Textlänge bestimmt, welches Zeichen an die Fragment-Stelle gehört. Es werden neue UV-Koordinaten für den einzelnen char berechnet und anhand dieser aus der Fontmap der korrekte Farbwert gelesen.
Konkret bedeuted dies, dass ganze Strings von bis zu 64 chars als zwei Dreiecke gerendert werden können. Danach muss ein neuer Text String verwendet werden, um mehr Text anzuzeigen. Auch mehrzeiliger Text ist aktuell nicht möglich. Allerdings ist es möglich, den Text links- oder rechtsbündig sowie mittig zu einem Ankerpunkt zu platzieren. Sowohl horizontal als auch vertikal. Dies geschieht dadurch, dass im Vertex-Shader einer der Dreieckspunkte auf die angegebene Textposition gesetzt wird und alle anderen relativ dazu anhand der gewünschten Bündigkeit (Bei zentrietem Text sitzen *alle* Dreieckspunkte relativ verschoben zur Textposition).

3.5 Skybox

Das Programm rendert nach allen anderen Objekten eine Skybox in den Hintergrund. Hierfür wurde eine Counter Strike Mod[5] verwendet, da keine andere passenden Skyboxen gefunden werden konnten. Für die Implementierung der OpenGL-Features wurden hauptsächlich das Tutorial von LearnOpenGL[6] verwendet.
Die Textur mit ihren sechs Seiten wird als $GL\_TEXTURE\_CUBE\_MAP$ an die aktive Textur gebunden, wodurch mit einem $samplerCube$ im Fragment-Shader über die UV-Koordinaten die Farbe aus der Textur gelesen wird.

3.6 Nachtmodus

Per Knopfdruck kann zwischen einem Tag- und einem Nachtmodus gewechselt werden. Tagsüber wird mit einem Directional Light das Sonnenlicht simuliert. Nachts wird das Sonnenlicht stark reduziert und die Drone schaltet einen Scheinwerfer an, um in der Stadt navigieren zu können. Darüber hinaus gehen nachts mehrere Punktlichter an, welche die Gebäude verschiedenfarbig anstrahlen.

3.7 Deferred Rendering

Das Projekt implementiert Deferred Rendering, um die Beleuchtung nicht mehr für jedes Objekt, sondern nur noch für jeden Pixel berechnen zu müssen. Die Szene wird in zwei Schritten gerendert. Zuerst werden beleuchtungsrelevante Informationen in mehrere Buffer eines ”GBuffers”gespeichert. Im zweiten Schritt wird die Beleuchtung für jeden Pixel berechnet, ausgehend von den gespeicherten Informationen im GBuffer.
Für den GBuffer werden fünf Rendertargets erstellt:

1. *Position:* Die Position des Objektes wird in Weltkoordinaten gespeichert.
2. *Normal:* Die Normale des Objektes wird ebenfalls in Weltkoordinaten gespeichert.
3. *Color:* Die Farbe des Objektes wird gespeichert.
4. *Emissive:* Die emissive Farbe des Objektes wird gespeichert.
5. *Specular + Shininess:* Hier werden zwei Komponenten in einer Textur zusammengefasst. Im R-Kanal steht der speculare Anteil, im G-Kanal die Shininess des Objektes. Diese wird durch 100 geteilt, um einen Wert zwischen 0 und 1 zu erhalten. Dabei wird angenommen, dass die Shininess 100 nicht überschreitet. Das ist notwendig, da Texturwerte zwischen 0 und 1 gespeichert werden.

Die Tiefe jedes Pixels wird in einem Renderbuffer gespeichert. Da die Anwendung nicht auf die Tiefe selber zugreifen muss, ist kein zusätzliches Rendertarget notwendig.
Der Rendervorgang vereint Deferred Rendering mit Forward Rendering:

1. *Geometry Pass:* Alle Objekte in der Szene werden mit einem speziellen Shader gerendert, der die relevanten Informationen in den GBuffer schreibt.
2. Abhängig von dem aktuell gewählten Visualisierungsmodus gibt es zwei Renderpfade:

   (a) *Light Pass:* Wenn die Szene normal dargestellt werden soll, werden hier alle benötigten Uniforms für den Lichtshader gesetzt, sodass dieser die Beleuchtung für jeden Pixel berechnen kann. Dazu wird ein Rechteck, dargestellt durch zwei Dreiecke, über das ganze Fenster gezeichnet. Der Lichtshader kann nun auf den GBuffer an dem bestimmten Pixel zugreifen und die Beleuchtung berechnen.
   (b) In diesem Fall ist ein anderer Visualisierungsmodus eingestellt. Dafür wird die Textur, in dem die zu visualisierende Komponente gespeichert ist, gebunden und an einen speziellen Shader übergeben, zusammen mit Informationen darüber, welche Kanäle dargestellt werden sollen. Hier wird ebenfalls ein Rechteck über das ganze Fenster gezeichnet, wodurch der Shader die Komponente darstellen kann.

3. Dieser Schritt ist nur notwendig, wenn der Standardvisualisierungsmodus ausgewählt ist. Hier wird Forward Rendering benutzt, um die Skybox zu rendern. Diese kann nicht mit Deferred Rendering gezeichnet werden, da sie nicht beleuchtet werden soll. Dafür wird der Tiefenbuffer des GBuffers in den Standardtiefenbuffer kopiert, damit die Skybox nur hinter allen anderen Objekten gezeichnet wird. In diesem Schritt könnten auch andere Objekte mit Forward Rendering gerendert werden, die nicht mit Deferred Rendering gerendert werden können, wie beispielsweise transparente Objekte.

4. Zum Schluss wird noch der Text über allen Objekten gerendert.

Der GBuffer kann durch verschiedene Visualisierungsmodi angezeigt werden:

- *Default:* Im Standardmodus werden alle Objekte beleuchtet dargestellt.
- *Deferred Only:* Hier werden nur Objekte im Deferred Rendering dargestellt. Im Projekt wird deshalb nur die Skybox ausgeblendet.
- *Position:* Die gespeicherte Position in Weltkoordinaten wird angezeigt. Da Farbwerte zwischen 0 und 1 gespeichert werden und die Stadt Koordinaten hat, welche größer als 1 sind, werden viele Objekte einfarbig dargestellt, mit Ausnahme kleiner Bereiche um die x- und z-Achse.
- *Normal:* Die Normale in Weltkoordinaten wird dargestellt.
- *Albedo:* Die Albedo wird dargestellt, im Projekt ist das größenteils der Farbwert einer Textur.
- *Specular:* Der spekulare Anteil wird angezeigt. Im Projekt hat lediglich die Drone eine spekulare Textur, daher werden in in diesem Modus die Gebäude schwarz dargestellt.
- *Emissive:* Die emissive Farbe wird hier angezeigt. In diesem Projekt hat nur die Drone im vorderen Bereich, der nach vorne zeigt (man sieht in daher nicht), eine emissive Farbe, daher wird in diesem Modus alles schwarz angezeigt.
- *Shininess:* Hier wird die Shininess der Objekte dargestellt. Sowohl die Gebäude als auch der Boden und die Drone haben jedoch den gleichen Wert, daher wird hier alles gleichfarbig dargestellt.

Literatur

[1] Codehead. *Codehead's Bitmap Font Generator*. Online, aufgerufen am 19.02.2021. URL: http://www.codehead.co.uk/cbfg/.

[2] Mario Haberle. *Drone Controller Demo*. Online, aufgerufen am 19.02.2021. URL: https://assetstore.unity.com/packages/tools/physics/drone-controller-demo-134089.

[3] Hallow. *Cake Model*. Online, aufgerufen am 19.02.2021. URL: https://www.models-resource.com/pc_computer/portal/model/15917/.

[4] richardhind. *low poly apartment building Free low-poly 3D model*. Online, aufgerufen am 19.02.2021. URL: https://assetstore.unity.com/packages/tools/physics/drone-controller-demo-134089.

[5] Syco. *Town Skybox*. Online, aufgerufen am 19.02.2021. URL: https://gamebanana.com/textures/download/200.

[6] Joey de Vries. *Cubemaps*. Online, aufgerufen am 19.02.2021. URL: https://learnopengl.com/Advanced-OpenGL/Cubemaps.