**Graphics Programming Projektreflexion**

**Kurs: GPR5300.S2**

**Name: Franz Mörike**

**Abgabedatum: 19.08.2022**

**Wörter: 1164**

**Inhalt:**

1. **Herangehensweise**
2. **Überlegungen zu möglichen Optimierungen**
3. **Probleme die auftraten und wie sie gelöst wurden**
4. **Herangehensweise:**

Zunächst habe ich als Grundlage die Vorlesungen von David Hackbarth genommen und anhand von diesen parallel zu ihm das Framework mit Direct 3D 9 und später Direct 3D 11 umgesetzt. Das finale Framework sollte komplett auf D3D11 basieren, da dieses mehr Modifizierungsmöglichkeiten in den einzelnen render-stages bietet.

**Grundsätzlicher Aufbau:**

Der grundsätzliche Aufbau des Projekts basiert auf OOP Prinzipien. Es gibt eine Main.cpp in der alle erstellten Objekte deklariert, instantiiert und mit der entsprechenden Init() Methode initialisiert. Es werden schrittweise ein Window, Direct 3D, (eine Skybox), mehrere Objekte, die Kamera, time management, das Material des meshes bzw. Objektes und die Beleuchtung gesetzt. Danach ist der update-loop des Fensters, indem Anpassungen der Objekte zur runtime aufgerufen werden. Zuletzt werden alle Objekte bzw. Pointer zu Objekten deinitialisiert und aufgeräumt, um keine Fragmente auf dem Speicher zu hinterlassen. Um Texturen zu laden wurde der WICTextureLoader verwendet. Alle klassenübergreifenden oder außerhalb der jeweiligen Klasse benutzten Komponenten sind zugreifbar über Getter.

**Einzelne Komponenten:**

**Window:**

Im Window wird von Grund auf ein neues Fenster erstellt. Es werden einige Einstellungen vorgenommen, wie beispielsweise passende Auflösung im Verhältnis zum Bildschirm, Hintergrundfarbe, Anzeigemodus (fullscreen, windowed). Außerdem verfügt das Window über den Update-Loop, der alle anderen instantiierten Objekte zur runtime an Änderungen anpasst. Die WndProc beinhaltet Aktionen, die am Fenster vorgenommen werden und wie mit diesen umgegangen wird. Zum Beispiel wenn das rote ‚X‘ in der Ecke gedrückt wird, soll das Fenster zerstört und das Programm beendet werden. Oder auch bei der Veränderung der Fenstergröße wird diese dementsprechend angepasst.

**Direct3D:**

In der Direct3D Klasse wird zunächst in Init() das d3ddevice Objekt (für Zugriff auf D3D-Methoden) und die swap-chain (tauscht front und backbuffer), sowie ihre Einstellungen erstellt. Es gibt die Möglichkeit einzustellen, welche Direct 3D Versionen unterstützt werden sollen. Auch das renderTargetView, depthStencilView (für 3D rendering), rasterizer (z.B. für backface culling) und viewPort (für das Fenster) werden mithilfe passender descriptions eingestellt und erstellt.

Damit die Anzeige passend gerendert wird, wird vor dem Rendern der Objekte im update-loop die renderTargetView und depthStencilView in StartFrame() zurückgesetzt. Am Ende jedes frames wird in EndFrame() durch die swapChain front und backBuffer ausgewechselt. Dort kann auch vsync aktiviert werden, um die Bildwiederholrate des Bildschirms mit der Grafikkarte zu synchronisieren (Nur empfehlenswert, sollte die Grafikkarte mehr frames berechnen können, als der Bildschirm darstellen kann).

**Mesh:**

Zunächst werden in der Init() alle Daten im mesh gesetzt, wie Position, Rotation und Skalierung. Daraufhin werden vertex- und index-buffer erstellt und anhand des enums kann der Typ des meshes bestimmt werden. Die erstellten vertices repräsentieren die Ecken des meshes, durch deren Verbindung ein mesh aufgespannt wird. Durch die indices wird die Reihenfolge der Verbindung der vertices bestimmt. Nachdem die buffer erstellt wurden, wird die Position des meshes in der worldMatrix gespeichert.

Das Mesh greift auf die Vertex Klasse zu, in der verschiedene Konstruktoren vorhanden sind, um das Mesh mit Position, Farbe, Normal- und UV-Koordinaten zu erstellen.

In der Update() werden alle zur runtime vorgenommenen Änderungen bestimmt, sowie die worldMatrix dementsprechend aktualisiert. Durch Drücken von Tasten können Eigenschaften des mesh, wie Position, Rotation und Skalierung zur runtime angepasst werden. Die Bedienung findet sich in der angehängten readme Datei.

In der Render() werden die gesetzten buffer an den InputAssembler gesendet, sowie die Beschaffenheit der Verbindung der vertices bestimmt (z.B. triangleList, triangleFan, …). Zuletzt wird das mesh gezeichnet

**Camera:**

In der Kamera Init() werden zunächst die view transformation matrix, sowie die perspective projection matrix berechnet. Diese werden benötigt um den Blick der Kamera auf die gerenderten Objekte zu bestimmen.

Die UpdateProjectionMatrix() ermöglicht, dass wenn die Größe des Fensters verändert wird, die projection matrix der neuen Größe entsprechend angepasst wird.

**Time management:**

In der Zeit Klasse wird eine eigene Form der deltatime berechnet, um die Geschwindigkeit von vorgenommenen Änderungen während der runtime, wie z.B. Rotation des meshes, anzuzeigen. Dies ermöglicht auch die Berechnung und Ausgabe der FPS in der Konsole.

**Material:**

In der Init() des Materials werden VertexShader und PixelShader initialisiert. Es gibt die Möglichkeit Farb-shader, Textur-shader oder beleuchtete Lighting-shader zu laden. Der color shader berechnet nur Farbwerte an jeder Position und verrechnet diese mit der Textur. Der texture shader ermöglicht voriges und zusätzlich Anpassungen der Textur durch UV-Koordinaten (tiling, offsets). Im lighting shader wird mithilfe der normals auch noch die externe Beleuchtung der Textur berechnet (keine self-illumination). Beim Erstellen des VertexShaders wird das Format der vertices bestimmt, damit die im Mesh gesetzten vertices in den shadern korrekt angesprochen werden kann. Außerdem wird ein matrix buffer erstellt, sowie die Textur mithilfe des WICTextureLoaders und der sampler state gesetzt. Der sampler state bestimmt die Umsetzung der UV-Koordinaten.

In der Render() Methode werden input layout, vertex shader, pixel shader und matrix buffer aktualisiert. Außerdem wird die Textur an den shader geschickt und sampler state aktualisiert.

**Lighting:**

Im Light wird in der Init() übergebene Daten gesetzt (Richtung, Farbe, Intensität des Lichts). Danach werden diese an einen neuen Buffer übergeben, um außerhalb auf diese Daten zugreifen zu können.

In der Render() wird dieser buffer an den pixel shader übergeben, um dort im lighting shader die Beleuchtung zu berechnen.

1. **Überlegungen zu möglichen Optimierungen:**

Ich habe unter anderem das Framework möglichst modifizierbarer gebaut insbesondere beim Mesh kann man Eigenschaften über MeshData setzen und diese werden dann in der Init() im Mesh selbst gesetzt. In MeshData können außerdem schon bestimmt werden, wie hoch verticesCount und indexCount sein sollen. Dadurch muss es nicht jedes Mal händisch innerhalb der CreateIndexBuffer() bzw. CreateVertexBuffer() angepasst werden. Damit bekannt ist, welches mesh gerendert werden soll, habe ich ein enum eingefügt, welches auch in MeshData gesetzt werden kann. Es verfügt außerdem über viele Methoden, die Modifizierungen nach Initialisierung ermöglichen (SetScale(), SetPosition(), etc.) Außerdem wurde das Input-System ausgebaut und verbessert.

Zu Optimierung von Fehlerüberprüfung wurde eine optimierte Version des ErrorHandlings aus der Vorlesung umgesetzt. Jeder Error besitzt eine eineindeutigen Zahlencode und der Name des Errors verweist auf die Stelle der möglichen Ursache. Auch die template Methode SafeRelease wurde optimiert, sodass in der Funktion selbst direkt auch ein ErrorCode geworfen werden kann. Es wurden zusätzlich weitere Templates implementiert, die helfen den Code übersichtlicher zu halten.

Durch die mesh-Klasse soll später noch eine Skybox anhand einer cubemap erstellt werden können. Die Grundsätze sind implementiert, aber die Skybox ist noch nicht gesetzt.

1. **Probleme die auftraten und wie sie gelöst wurden:**

Da ich die vertices und indices individuell je nach übergebenem Typ an Mesh setzen wollte, wurde dies in eine externe Funktion verlagert. Ich hatte daraufhin das Problem, dass man in dieser Methode kein Array zurückgeben kann, welches aber von data.pSysMem erwartet wurde. Dies wurde durch pointer gelöst. Außerdem trat das Problem auf, dass das Mesh vom Cube zuerst falschrum gerendert wurde, also die Innenseiten zu sehen waren, anstatt der äußeren Flächen. Dagegen invertieren der normals und anpassen der Reihenfolge der vertices. Ich hätte gern noch mehrere verschiedene meshes eingebaut, wie Sphären und Ähnliches, leider haben wir im Unterricht größtenteils nur planes und cubes behandelt.