Endabgabe EZG

Beschreibung

Bei dem Programm handelt es sich um eine Anwendung welche mittels DirectX 11 einige Techniken des Real Time Renderings in einer Szene mit mehreren Models verbindet.

Verwendet werden unter zum Beispiel: Diffuse Lighting, Shadow Mapping (Multiple Shadows), Normal-Mapping, und Multi-Sampling.

Weiters lädt die Anwendung Models aus .obj-Files, realisiert eine Kamerafahrt mittels eines Kochanek-Bartels Interpolators. Alle Models in der Szene werden in einen KD-Tree aufgenommen, welcher für das Raycasting verwendet wird.

Tastenbelegung

Aufgrund der vielzahl an Interaktionen sind viele Tasten belegt worden.

W, A, S, D	Bewegungstasten
Pfeiltasten	Rotationstasten
Z	Wireframe-Mode
R	Resetten der Wegpunkte (Kamerafahrt)
E	Setzen eines neuen Wegpunktes
T	Starten der Kamerafahrt
Y	Mehr Bumpiness
X	Weniger Bumpiness
B	Bounding-Boxes Rendermode
;	Durchschalten der Samples (MSAA)
.	Durchschalten der Quality Levels
M	Aktivieren der neuen Samples plus QL
1	Position für Licht 1 setzen
2	Position für Licht 2 setzen
Space	RayCast absetzen

Kamerafahrt

Die Kamerafahrt befindet sich in der Klasse “CameraClass”. Mittels der Kochanek Bartels Formel werden mehrere Punkte zwischen den gesetzten Wegpunkten interpoliert. Ebenfalls zwischen den Blickrichtungen wird interpoliert. Hierzu werden Hilfvektoren erstellt und in die Hermite Funktion übergeben.

Zuerst gab es einige Probleme mit den Quaternions für die Rotationsinterpolation der fliegenden Kamera zwischen den Wegpunkten. Allerdings hat sich das mit zunehmendem Wissen über die Funktionsweise von Quaternionen gelegt.

Schatten

Für die Schattenberechnung werden die Models in der Szene zuerst in eine DepthTexture gerendert. Die ViewMatrix befindet sich an stelle der Lichtquelle und schaut in eine bestimmte Richtung. Da es sich um Directional Light handelt, ist die Projection Matrix orthogonal. Dies simuliert eine weit entfernte Lichtquelle wie die Sonne.

Im Pixelshader wird anschließend geschaut, ob der derzeit berechnete Pixel sich im Schatten befindet. (Auf der DepthTexture zu sehen ist.) Abhängig davon wird die endgültige Farbe des Pixels mit oder ohne Diffusem Licht berechnet.

Dieser Vorgang wird für zwei Lichtquellen hintereinander ausgeführt. Für jede weitere Lichtquelle muss eine weitere DepthTexture als RenderTarget angelegt werden.
Die Klassen “DepthShaderClass” und “RenderTexture” sind für die Schatten verantwortlich. Es werden außerdem die Shader “DepthVertexShader” und “DepthPixelShader” verwendet.

Spezielle Probleme hat mir der zu wählende “Bias” bereitet, welcher die Ungenauigkeiten in der Depthberechnung ausmerzen soll. Zuerst war der Wert so hoch gesetzt, dass laut dem Pixelschader alle Triangle iim Schatten gelegen sind. Später hat sich dann “Shadow Acne” in der kompletten Szene gebildet. Letzten endes war es ein Trial and Error verfahren um den richtigen Wert zu finden.

Normal Mapping

Für das Normal Mapping werden beim einlesen der Models aus den .obj-Files weitere wichtige Vektoren (Tangente, Binormale, Normalvektor) berechnet. Mithilfe dieser ändern sich die Einfallwinkel des Lichts auf dem Model per Pixel und es können auf ebenen Polygonen sogar 3-Dimensionale Effekte entstehen. Obwohl sich die Geometrie des Objektes nicht ändert, erhält man viel mehr Realismus.
Das Berechnen der Vektoren findet in der Klasse “ModelLoader” statt.

Ausnahmsweise hatte ich keine Probleme beim implementieren der Normal Maps.

MSAA (MultiSampling)

Um das Anti-Aliasing mittels Multisampling zu realisieren und ein Umschalten der Samples und des QualityLevels zur Laufzeit seamless zu ermöglichen wird die komplette Szene auf eine Texture gerendert (RenderTarget) welche am Schluss in den BackBuffer geschrieben wird.

KD-Tree und RayCasting

Um einen KD-Tree aufzubauen werden alle geladenen Triangle in einen Rekursiven Aufruf der KDNode::Build funktion übergeben. Die Abbruchbedingungen haben hier ein paar Probleme gemacht, da es zu Stack-Overflows kommen kann. Gelöst wurde dies mittels den Bedinungen “Triangle pro Bounding Box < 100” und “Depth < 100”.
Die jeweilige Bounding Box wird immer an der längsten Achse (X, Y, Z) geteilt. Triangles welche durchschnitten wurden, werden in beide resultierende Bounding Boxen aufgenommen. (Fehlerursache des Stackoverflows)

Nach dem Erstellen des KD-Trees kann ein Ray von der Kameraposition in Sichtrichtung losgeschossen werden. Das näheste getroffene Triangle wird in der Ansicht farblich markiert. Sobald zwei unterschiedliche Dreiecke getroffen wurden, wird die Distanz zwischen beiden Ausgegeben und ein Strahl zwischen ihnen eingezeichnet.