EZG Projektdokumentation MGS1 2015/16

Hannes Höttinger - 1510585004

FH Technikum Wien, Game Engineering und Simulation, Wien, AUT

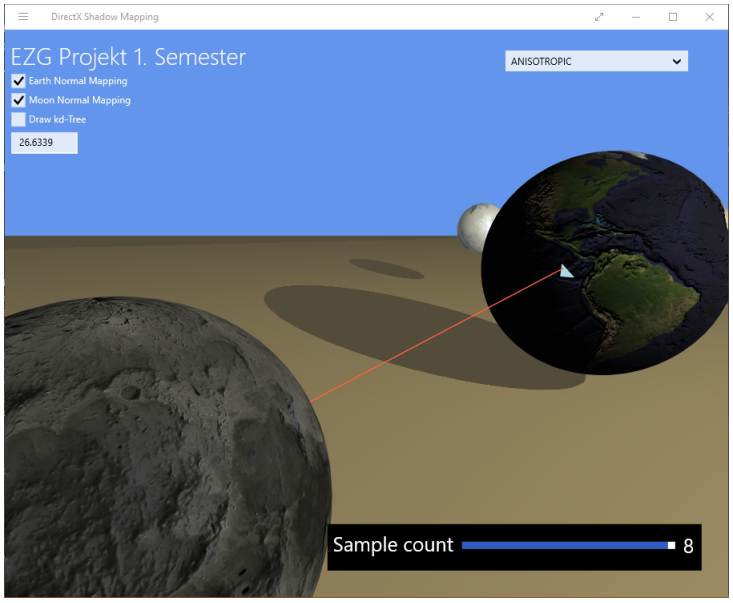

Abbildung 1: 3D-DirectX Szene mit Point-To-Point Messung, Shadow Mapping, Anti-Aliasing und Normalmapping

1 Aufgabenstellung

DirectX 3D - Visualisierung mit folgenden Inhalten:

1. 3 Objekte im Raum
2. Kochanek-Bartels Spline Interpolation für eine Kamerafahrt mit mindestens 5 Punkten + Quaternion Squad Interpolation
3. Shadow Mapping mit veränderbarer Lichtquelle
4. Bump/Normal Mapping umschaltbar
5. Texture Filtering umschaltbar + MipMapping
6. Anti-Aliasing Quality Levels/Sample Count umschaltbar
7. Ray-Tracing mittels kd-Tree und Visualisierung des getroffenen Triangles; Camera-To-Point und Point-To-Point Distanzmessung

2 Bedienung

Eingabe	Steuerelement
[W] / [S]	Kamera vorwärts / rückwärts
[A] / [D]	Kamera links / rechts
[↑] / [↓]	Kamera Rotation x-Achse
[←] / [→]	Kamera Rotation y-Achse
[Q] / [E]	Kamera Rotation z-Achse
[I] / [K]	Lichtquelle vorwärts / rückwärts
[J] / [L]	Lichtquelle links / rechts
[Z] / [H]	Lichtquelle Rotation x-Achse
[U] / [O]	Lichtquelle Rotation y-Achse
[N] / [M]	Lichtquelle Rotation z-Achse
[Enter]	Kamera zum Startpunkt der Kamerafahrt setzen
[F1]	Kamerafahrt starten
[1][2][3][4][5]	Neue Interpolationspunkte für Kamerafahrt festlegen
[Space]	Umschalten zwischen Camera-To-Point und Point-To-Point Messung
[RMT]	Ray an aktuelle Mausposition

Tabelle 1: Steuerelemente der DirectX Szene

Weitere Steuerelemente sind durch das XAML-Overlay gegeben. Die gesamte Szene ist als Microsoft Universal App aufgebaut. Das GUI wird durch XAML Steuerelemente realisiert. Die App kann somit theoretisch auf allen Windows Geräten eingesetzt werden. Normal Mapping von Mond und Erde können hier bedient werden, sowie die verschiedenen Filtermodi und das Anti-Aliasing (Samplecount) (siehe Abbildung 2). Weiters wird in einer Textbox die aktuell gemessene Distanz (Camera-To-Point bzw. Point-To-Point) ausgegeben. Der kd-Tree wird bei Programmstart berechnet und für die Darstellung aufbereitet. Über die Checkbox kann die Visualisierung aktiviert/deaktiviert werden.

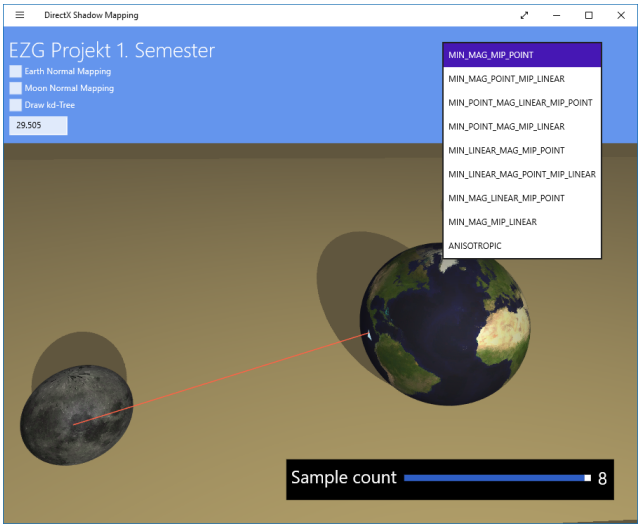

Abbildung 2: XAML-Steuerelemente - Combobox mit allen Filtermodi

3 Implementierung

3.1 Kamera

Es wurde eine First-Person Kamera implementiert. Je nach Eingabe durch den User wird die Kamera um einen gewissen Winkel rotiert, abhängig von der Dauer der Eingabe (gedrückt halten) und der Samplingrate. Dies wird mittels Euler-Winkel realisiert, die wiederum in Quaternions für die Rotationsmatrix umgerechnet werden. Es erfolgt eine erneute Berechnung der Viewmatrix.

Die automatische Kamerafahrt wird mittels Kochanek-Bartels Spline Interpolation realisiert. Die notwendigen Formeln wurden von Tomas Akenine-Moller (2008) verwendet. Die Quaternion Interpolation erfolgt mittels der Squad-Methode. Diese Methode wurde mittels DirectX spezifischer Funktionen implementiert: *XMQuaternionSquadSetup* und *XMQuaternionSquad*.

Listing 1: Squad Quaternion Interpolation for Rotation and Kochanek-Bartel Spline Interpolation for Translation

```
void kochanekSplineandSquadInterpolation(const ↩
    CameraViewPoint& point0, const CameraViewPoint& point1,↩
     const CameraViewPoint& point2, const CameraViewPoint& ↩
    point3, float s)
{
    float tension = 0.0f, bias = 0.0f, cont = 0.0f;
    XMVECTOR starttang, endtang;
    XMVECTOR qRot1, qRot2, qRot3;

    // Kochanek-Bartels spline tangents
    starttang = (((1 - tension)*(1 + bias)*(1 - cont)) / 2)↩
         * (point1.Eye - point0.Eye) + (((1 - tension)*(1 -↩
         bias)*(1 + cont)) / 2) * (point2.Eye - point1.Eye)↩
        ;
    endtang = (((1 - tension)*(1 + bias)*(1 + cont)) / 2) *↩
         (point2.Eye - point1.Eye) + (((1 - tension)*(1 - ↩
        bias)*(1 - cont)) / 2) * (point3.Eye - point2.Eye);

    // Kochanek-Bartel Spline for Position interpolation
    eye = (2 * pow(s, 3) - 3 * pow(s, 2) + 1) * point1.Eye ↩
        + (pow(s, 3) - 2 * pow(s, 2) + s) * starttang + (-2↩
         * pow(s, 3) + 3 * pow(s, 2)) * point2.Eye + (pow(s↩
        , 3) - pow(s, 2)) * endtang;
    //Quaternion Squad Interpolation for Quaternion ↩
        rotation to desired positions
    XMQuaternionSquadSetup(&qRot1, &qRot2, &qRot3, point0.↩
        quaternion, point1.quaternion, point2.quaternion, ↩
        point3.quaternion);
    quaternion = XMQuaternionSquad(point1.quaternion, qRot1↩
        , qRot2, qRot3, s);
}
```

3.2 Schatten

Zuerst wird die Szene von dem Blickpunkt der Lichtquelle gerendert. Als Lichtquelle wurde ein Spotlight verwendet mit einer eigenen Projektionsmatrix = "Lichtkegel". Es werden jedoch nur die Tiefenwerte gespeichert und die Szene auf die ShadowMap gerendert. Anschließend wird die gesamte Szene nochmals gerendert, diesmal jedoch mit den gespeicherterten Tiefenwerten. Im Pixelshader wird nun ein Test durchgeführt ob das Objekt vor der Lichquelle bzw. sich im Schatten befindet. Wenn das Objekt nicht im Schatten liegt, wird eine Lichtberechnung mit diffuser und spekularer Komponente durchgeführt. Schattenobjekte werden von der ShadowMap gesampelt und nur deren Textur und ein Ambient Anteil berechnet.

3.3 Normal-Mapping

Die Normalmap wird dazu verwendet, die Lichtintensität jedes Pixels zu berechnen und damit die tatsächliche Pixelfarbe zu bestimmen. Zuerst werden die Werte aus der Normalmap-Textur in eine Range von [-1,+1] gebracht. Anschließend wird für jedes Pixel ein neuer Normalvektor berechnet. Der projezierte Lichtstrahl wird je nach Aufprallpunkt abhängig von der Normalmap reflektiert. Die resultierende Intensität wird für die weitere Lichtberechnung verwendet und führt somit zu einer Tiefeninformation der Textur (vgl. Abbildung 1 und 2). Das Normalmapping ist mittels XAML Checkbox ein/ausschaltbar, um den Effekt besser zu visualisieren.

3.4 Anti-Aliasing

Die Anordnung und Anzahl der Abtastpunkte spielt bei Anti-Aliasing eine große Rolle für das Aussehen des resultierenden Bildes. Um Anti-Aliasing zu aktivieren, muss vorher die Grafikkarte nach den unterstützten Quality Levels und deren Sample Count abgefragt werden. Dies wird mittels der Funktion *CheckMultisampleQualityLevels* realisiert. Der Slider im XAML-GUI wird daraufhin angepasst und es können jeweils nur die unterstützten Werte eingestellt werden. Um nun Multisampling zu aktivieren ist es notwendig ein Rendertarget mit *D3D11_RTV_DIMENSION_TEXTURE2DMS* zu erzeugen. Es wird auf eine Offscreen Textur gerendert, die mit dem Befehl *ResolveSubresource* auf den Backbuffer aufgelöst wird. Sobald der Slider verwendet wird, wird ein neues Rendertarget mit der eingestellten Anti-Aliasing Stufe erzeugt.

Weiters werden die Ergebnisse des Grafikkartenchecks in der Debug Konsole ausgegeben. Beispielausgabe Intel HD 4000:

MSAA 1X supported with 1 quality levels
MSAA 2X supported with 1 quality levels
MSAA 4X supported with 1 quality levels
MSAA 8X supported with 1 quality levels

3.5 MipMapping

Beim MipMapping wird je nach tatsächlicher Größe einer Textur im gerenderten Bild eine unterschiedlich große Version einer Textur ausgewählt. Die kleineren Versionen derTextur werden schon vor Programmlaufzeit berechnet und müssen nur aus dem *.dds Container geladen werden. Nun ist es nur noch notwendig die MipMapping Tiefe auszuwählen (LOD = Level of Detail):

Listing 2: DirectX MipMapping Sampler

```
// allow use of all mip levels
samplerDesc.MinLOD = 0;
samplerDesc.MaxLOD = D3D11_FLOAT32_MAX;
```

3.6 Picking

Es wird ein Ray erstellt, indem die Mauskoordinaten mit Hilfe der Projektionsmatrix und der Inversen View-Matrix in den Worldspace übertragen werden. Anschließend wird die rayDirection sowie rayOrigin berechnet. rayOrigin ergibt die Kameraposition (bei Camera-To-Point Messung, bei Point-To-Point wird rayOrigin mit dem gefundenen ersten Punkt überschrieben). Für die Richtung wird einfach ein z-Wert in Richtung der Szene gewählt (-1.0f). Für nähere Details: siehe Code + Kommentare.

Der Intersectiontest wird mittels Ray-Triangle Intersection durchgeführt. Für diesen Test wird der Ray auf die Triangleebene projeziert und mittels baryzentrischer Koordinaten festgestellt, ob der Punkt indem der Ray die Ebene schneidet innerhalb des Dreiecks liegt. Als Nebenprodukt dieses Verfahrens erhält man den exakten Schnittpunkt des Rays auf dem getroffenen Triangle. Dieser Punkt wird als Endpunkt für den gezeichneten Ray verwendet (siehe Abbildung 3). Weiters wird das getroffene Triangle des Objekts markiert.

3.7 kd-Tree

Um nun nicht alle Triangles der Szene (gespeichert in WorldSpace Koordinaten) mit dem Ray-Triangle Intersection Test prüfen zu müssen wird er durch einen kd-Tree geschickt. Der Baum wird statisch zu Programmbeginn berechnet.

Zum Aufbau des Baumes wird zuerst die größte Expansionsrichtung aller Dreiecke gesucht, um die Splitplane-Richtung auszuwählen, anschließend die Dreiecke am Median in zwei Hälften geteilt (wobei Dreiecke, die auf beiden Seiten liegen, in beide Hälften gespeichert werden), und für diese Dreiecksmengen rekursiv wieder der selbe Algorithmus ausgeführt. Dies passiert so lange, bis eine Dreiecksmenge nur noch 5 oder weniger Dreiecke enthält, eine Tiefe von 32 erreicht wurde, oder bei der Erstellung der Hälften eine oder beide Hälften wieder alle Dreiecke enthalten. Beim Traversieren des Baumes werden durch Schnitt des Strahls mit den Splitplanes des KD-Trees nur die Weltbereiche durchsucht, durch die der Strahl auch wirklich verläuft. Innerhalb eines Bereiches wird der

Strahl dann mit allen Dreiecken, die er noch nicht geschnitten hat, geschnitten und dadurch bei Treffer der Schnittpunkt errechnet. Es werden die jeweiligen Splitting-Planes für die Visualisierung verwendet. Weiters wird eine Zeitmessung für das Raycasting durchgeführt. Dafür wird der Ray mit allen Triangles getestet und mit dem Raycasting mithilfe des KD-Trees verglichen. Es ergibt sich folgendes Messergebnis für 11904 Triangles:

Raycast all triangles [ms]: 00:00:00.029;
Raycast triangles with kd-tree [ms]: 00:00:00.007;

4 Renderszene

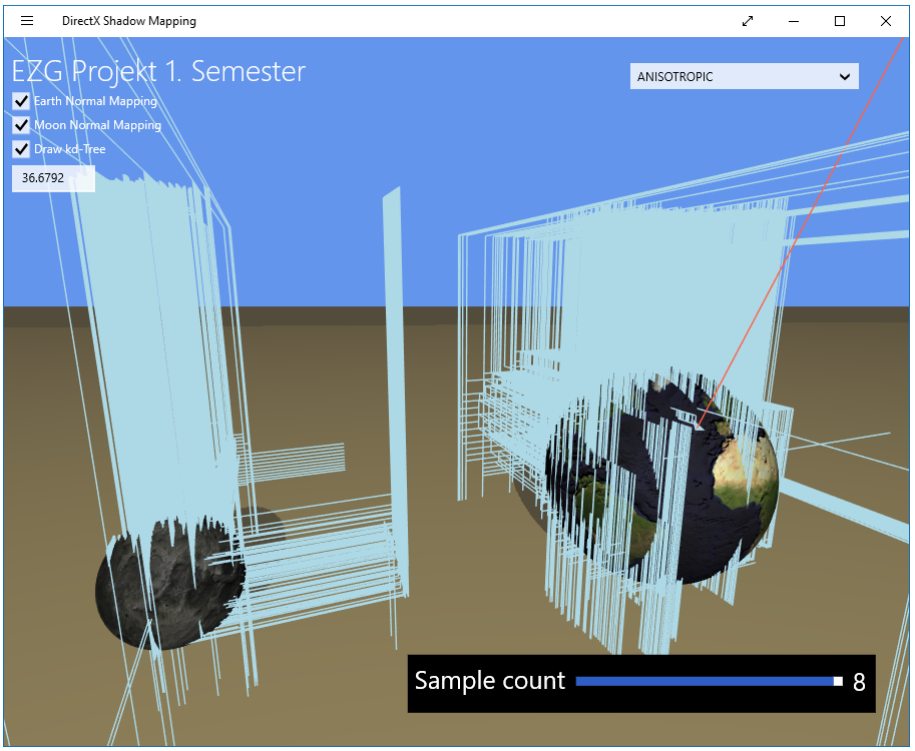

Abbildung 3: Renderszene mit Camera-To-Point Distanzmessung und eingezeichnetem kd-Tree

Literatur

Tomas Akenine-Moller, Eric Haines, N. H. (2008). *Real-time rendering.* A.K. Peters, 3rd edition.