Paper: Geometry Shader

Grass Shader

Wannes Vanderstappen – Juni 2010

2DAE-GD1

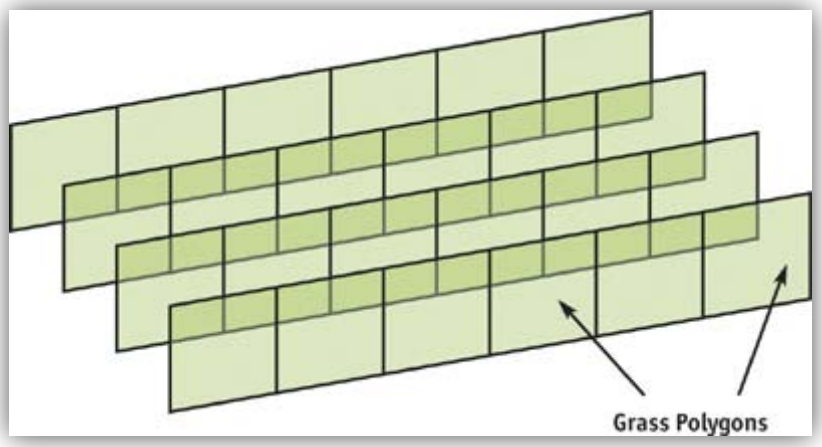
1. Algemeen

In games wordt voor een buitenomgeving vaak gebruik gemaakt van een terrein. Op dit terrein worden dan verschillende textures geplaatst om de indruk te wekken van bijvoorbeeld rotsen, zand, sneeuw of gras.

Om de indruk van gras nog meer te versterken wordt hiervoor vaak extra geometrie gebruikt. Met DirectX 10 is het mogelijk om deze geometrie op de grafische kaart aan te maken.

2. Plaatsing van het gras

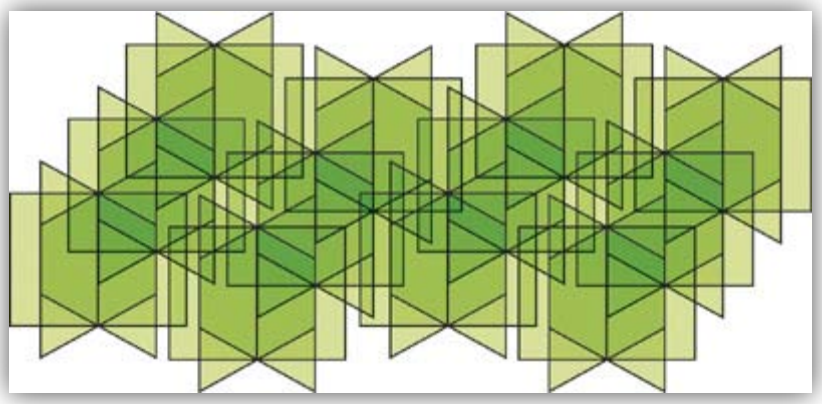
Het gras bestaat uit verschillende quads die een texture bevatten van gras (met alpha kanaal voor opacity). Wanneer je echter schuin op zo een vlak kijkt of erger nog, helemaal van de zijkant, dan merk je al snel dat het gras geen volume heeft. (zie figuur 1).



Figuur 1: Alle quads zijn in dezelfde richting gedraaid.

Om dat te voorkomen, worden de quads niet allemaal in dezelfde richting geplaatst. Dit kan op verschillende manieren gedaan worden.

Je kan bijvoorbeeld de quads telkens per drie quads in een soort ster-vorm plaatsen (zie figuur 2).



Figuur 2: Quads zijn geplaatst per groep van 3 in een ster-vorm.

Een andere manier (de manier die gebruikt is in deze shader) is om elke quad van het gras naar de camera te richten. Hierdoor zal je nooit de zijkant van een quad te zien krijgen. Om niet te hard te laten opvallen dat het aparte quads zijn die draaien, zijn ze zo geplaatst dat ze door elkaar komen. Deze techniek wordt billboarding genoemd (zie later) en wordt bijvoorbeeld ook gebruikt in The Elder Scrolls IV: Oblivion bij de bladeren van de bomen en struiken (zie figuur 3).

Figuur 3: De bladeren van de bomen in Oblivion lijken ook volume te hebben omdat ze geroteerd worden naar de camera.

3. Overzicht Shader

Het object dat met de gras-shader gerenderd wordt is in dit geval een terrein. Het terrein bevat per vertex de positie, normal en texture weights. Met behulp van de texture weights kan er gekeken worden of een bepaalde vertex gras voorstelt, rotsen, sneeuw, ...

Vertex Shader:

In de vertex-shader wordt de positie vermenigvuldigd met de world matrix. De normals en de texture weights worden gewoon doorgegeven naar de Geometry Shader.

Geometry Shader:

De quads waarop de gras-textures geplaatst worden worden op de GPU aangemaakt in de Geometry Shader. Hier is een kort overzicht van wat er in de Geometry Shader gebeurt:

- Per triangle van de mesh wordt gekeken of die triangle niet te ver van de camera ligt (enkel dichtbij wordt geometrie voor het gras aangemaakt).
- Er wordt ook gekeken of er op die positie wel degelijk gras moet zijn (via de texture weights).
- Als dit punt aan die voorwaarden voldoet, dan wordt op die plaats een quad aangemaakt die naar de camera gericht wordt. De Geometry Shader heeft enkel dan output. Voor de quad worden ook texture coördinaten berekend en doorgegeven naar de Pixel Shader.
- Het gras wordt geanimeerd om wind te simuleren.
- De normal van elke vertex van de quad is gelijk aan de normal van het punt waaruit de quad gemaakt is. Dit is handig bij belichting (zie Pixel Shader).

Pixel Shader:

In de pixel shader wordt een gras texture gesampled op de quads. Doordat de normal van de vertices gelijk is aan de normal van de vertex er onder van het terrein, kan er diffuse-lighting gedaan worden voor het gras. Het gras wordt dan op dezelfde manier belicht als het terrein. Omdat je niet op voorhand kan weten in welke volgorde de quads gerenderd zullen worden kan je niet zomaar alpha blending gebruiken. Er wordt in de plaats gebruik gemaakt van de functie clip(...) om zo de transparante delen niet te renderen. Om het gras toch zachtere randen te geven en om het er in de verte ook iets beter te laten uitzien kan er gebruik gemaakt worden van een 2de render Pass. In de eerste Pass is alpha blending disabled en de z-buffer enabled. Hier worden dan de pixels geclipped die bijvoorbeeld minder dan 30% zichtbaar zijn. In de tweede Pass is alpha blending enabled, maar schrijven naar de z-buffer disabled. Hier worden de pixels geclipped die meer dan 30% zichtbaar zijn.

4. De Geometry Shader

4.1 Beperken van de output

De shader bevat een variabele die de camerapositie bijhoudt. Hiermee kan de afstand berekend worden tot elk punt van de mesh (world-positie). In de Geometry Shader komen drie vertices binnen die een triangle van het terrein vormen. Die punten bevatten positie, normal en texture weights. Als output heeft de Geometry Shader maximum vier vertices die een quad voorstellen. Enkel wanneer de afstand tot een triangle kleiner is dan een zelfgekozen waarde en wanneer er op die triangle gras is (bijvoorbeeld textureweights.x > 0.6f) zal de Geometry Shader die quad maken. In de andere gevallen heeft de GS geen output. Dit zorgt voor een betere performance van het gras.

4.2 Billboarding

Billboarding is het draaien van een object in de richting van de camera, in dit geval de quads voor het gras. Om deze quads te kunnen draaien moeten we uiteraard eerst quads hebben.

Per triangle (hier de triangles van het terrein) wordt een quad aangemaakt in zijn lokaal assenstelsel met een bepaalde breedte en hoogte. (Indien de grootte van de triangles van het terrein te groot is zullen de quads voor het gras te ver uit elkaar staan. Er kunnen dan eventueel meerdere quads per triangle aangemaakt worden.) De texture coördinaten worden gewoon gespreid over de hele quad.

```
float Width = 1.2f;
float Height = 1.2f;


float4 v[4];
v[0] = float4(-Width*0.5f, 0,      0.0f, 1.0f);
v[1] = float4(+Width*0.5f, 0,      0.0f, 1.0f);
v[2] = float4(-Width*0.5f, Height, 0.0f, 1.0f);
v[3] = float4(+Width*0.5f, Height, 0.0f, 1.0f);

float2 texC[4];
texC[0] = float2(0.0f, 1.0f);
texC[1] = float2(1.0f, 1.0f);
texC[2] = float2(0.0f, 0.0f);
texC[3] = float2(1.0f, 0.0f);
```

Van deze quad wordt dan de world matrix opgesteld op de volgende manier:

De up-vector is steeds dezelfde: het gras is altijd naar boven gericht, anders zou het gras platliggen wanneer je er van bovenaf naar kijkt. In dit geval is dat dan:

```
float3 up = float3 (0.0f, 1.0f, 0.0f);
```

De look-vector kan worden berekend door het verschil van de positie van de camera en de world-positie van de plaats waar het gras komt:

```
float3 look = EyePos – “world-positie van de quad”;
```

Hiervan wordt dan de y-waarde op 0 gezet en het geheel genormalized:

```
look.y = 0.0f;
look = normalize(look);
```

De right-vector kan dan berekend worden door het cross-product van de up-vector en de look-vector:

```
float3 right = cross(up,look);
```

De world matrix wordt dan met deze vectoren en de world-positie samengesteld:

```
float4x4 W;
W[0] = float4(right, 0.0f);
W[1] = float4(up, 0.0f);
W[2] = float4(look, 0.0f);
W[3] = float4(“world-positie van de quad”, 1.0f);
```

Deze world matrix wordt vermenigvuldigd met de view-projection matrix om de world-view-projection matrix te vormen:

```
float4x4 WVP = mul(W,gViewProjection);
```

Van de vier berekende punten wordt nu de positie vermenigvuldigd met de world-view-projection matrix, de normal wordt geroteerd met het rotatiegedeelte van de world matrix en de texture coördinaten worden gewoon doorgegeven. Deze vertices worden toegevoegd aan de TriangleStream.

```
PS_INPUT_STRUCT gsOut;
for (int i = 0; i<4; ++i)
{
      gsOut.PosH = mul(v[i],WVP);
      gsOut.Normal = mul(“normal terrein op positie waar de quad komt”,
                                                (float3x3)gWorld);
      gsOut.Tex = texC[i];

      triStream.Append(gsOut);
}
```

4.3 Animatie

Om het gras wat gevarieerder te maken is de hoogte van het gras variabel gemaakt. Om dit te doen is een Texture2D variabele toegevoegd aan de shader. Deze texture bevat random waarden. In de Geometry Shader wordt naargelang de world-positie van het gras op een andere plaats van deze texture gesampled om een random waarde te verkrijgen. Deze waarde wordt gebruikt om de hoogte van het gras variabel te maken:

```
float random = grassOffsetsTex.SampleLevel(samLinear,float2(“x-waarde
world-positie”,”z-waarde world-positie”)/32.0f,0);
```

```
float Width = 1.2f + random;
float Height = 1.2f + random;
```

Aan de shader is ook nog een float variabele toegevoegd die per tick van de game verhoogd wordt. Deze variabele stelt een aantal radialen voor. Op deze manier kan makkelijk een sinus-achtige beweging gemaakt worden voor het gras. Bij het aanmaken van de vertices zorgt een combinatie van deze shadervariabele en de random voor een gevarieerde beweging:

```
float4 v[4];
v[0] = float4(-Width*0.5f, 0, 0.0f, 1.0f);
v[1] = float4(+Width*0.5f, 0, 0.0f, 1.0f);
v[2] = float4(-Width*0.5f+sin(grassOffset+3*random)*0.1f*random, Height,
0.0f, 1.0f);
v[3] = float4(+Width*0.5f+sin(grassOffset+3*random)*0.1f*random, Height,
0.0f, 1.0f);
```

5. Verbeteringen

Om het gras nog gevarieerder te maken zou in plaats van één texture een texture array kunnen gebruikt worden. Met de random waarden die al in de shader zitten zou dan willekeurig een texture kunnen gekozen worden.

Bronnen

K. Pelzer, 'Rendering Countless Blades of Waving Grass', http://http.developer.nvidia.com/GPUGems/gpugems_ch07.html

'Billboards Sample', http://creators.xna.com/en-us/sample/billboard

F. D. Luna, '*Introduction to 3D game programming with DirectX 10*', Texas, Wordware Publishing, 2008