# 11.2 树广告牌演示程序

## 11.2.1 概述

当树与观察点的距离很远时，我们可以通过广告牌（billboard）技术来提高渲染效率。也就是，我们只在一个四边形上绘制树的3D图片，而不是渲染一个完整的3D树模型（参见图11.2）。从远处看，你根本分辨不出是否使用了广告牌。不过，你必须确保广告牌始终面对摄像机（否则个假象就会被拆穿）。

****

**图11.2 带有alpha通道的树广告牌纹理。**

假设*y*轴垂直向上，*xz*平面为地平面。树广告牌总与*y*轴对齐，只在*xz*平面上面对摄像机。图11.3展示了鸟瞰视图中的几个广告牌的局部坐标系——注意，广告牌始终“面对”摄像机。

****

**图11.3 广告牌面对摄像机。**

这样，只要在世界空间中给定广告牌的中心位置**C**=(Cx,Cy,Cz )和摄像机的位置**E**=(Ex,Ey,Ez)，我们就有足够的信息来描述广告牌相对于世界空间的局部坐标系：



**v** = (0, 1,0)

**u** = **v** × **w**

给定上述局部坐标系以及公告牌的大小，公告牌四个顶点的坐标就可以由以下方式确定（如图11.4所示）：

v[0] = float4(gin[0].CenterW + halfWidth\*right - halfHeight\*up, 1.0f);

v[1] = float4(gin[0].CenterW + halfWidth\*right + halfHeight\*up, 1.0f);

v[2] = float4(gin[0].CenterW - halfWidth\*right - halfHeight\*up, 1.0f);

v[3] = float4(gin[0].CenterW - halfWidth\*right + halfHeight\*up, 1.0f);

****

**图11.4 根据公告牌的局部坐标系统和大小计算四个顶点的位置**

注意，每个广告牌的局部坐标系统都不一样，我们必须为每个广告牌分别计算广告牌矩阵。

在本例中，我们将创建一个点图元列表（**D3D11\_PRIMITIVE\_TOPOLOGY\_POINTLIST**），每个点图元的位置都会比地形网格略高一些。这些点描述了我们所要绘制的广告牌的中心位置。在几何着色器中，我们将这些点扩展为广告牌四边形，并计算广告牌的世界矩阵。图11.5展示了该程序的屏幕截图。

****

**图11.5 树广告牌演示程序的屏幕截图。**

如图11.5所示，本例建立在第9章的“Blend”演示程序基础之上。

**注意**：公告牌的CPU实现版本要将公告牌的四个顶点放置在一个动态顶点缓冲中，当相机移动时，顶点坐标就需要通过CPU进行更新，然后使用**ID3D11DeviceContext::Map**方法发送到GPU上。这个方法必须将每个公告牌的四个顶点绑定到IA阶段，并更新动态缓冲区时，这是非常耗时的。而使用几何着色器的方法，GPU会计算公告牌的四个顶点并使之朝向相机，所以只需使用静态缓冲区即可，而且，每个公告牌只需使用一个顶点，所需内存也较少。

## 11.2.2 顶点结构体

我们用下面的顶点结构体来描述广告牌的位置：

struct TreePointSprite

{

 XMFLOAT3 Pos;

 XMFLOAT2 Size;

} ;

const D3D11\_INPUT\_ELEMENT\_DESC InputLayoutDesc::TreePointSprite[2] =

{

 {"POSITION",0,DXGI \_FORMAT\_R32G32B32\_FLOAT,0,0,D3D11\_INPUT\_PER\_VERTEX\_DATA,0},

 {"SIZE",0,DXGI\_FORMAT\_R32G32\_FLOAT,0,12,D3D11\_INPUT\_PER\_VERTEX\_DATA,0}

} ;

该顶点结构体存储了广告牌在世界空间中的中心位置、宽度和高度（单位以世界空间为准），这样就可以使几何着色器知道广告牌应该被放在哪里，以及扩展为多大尺寸（参见图11.6）。通过改变每个顶点的尺寸，可以很容易地让广告牌呈现出各种不同的大小。

****

**图11.6 将一个点扩展为一个四边形。**

除纹理数组（参见11.3节）外，“Tree Billboard”示例中的其他 C++代码都是普通的Direct3D代码（创建顶点缓冲区、effect、调用绘图方法等等）。所以，我们现在将讲解的重点转向tree.fx文件。

## 11.2.3 Effect文件

由于这是我们的第一个几何着色器程序，所以我们把整个效果文件的内容都列了出来，以使你更清楚地看到顶点着色器、几何着色器、像素着色器以及其他效果对象是如何协同工作的。这个effect文件中还有一些我们未讨论过的对象（**SV\_PrimitiveID**和**Texture2DArray**）；这些内容会在稍后进行讲解。现在，我们主要讲解几何着色器程序**GS**，它按照11.2.1节讨论的方法，把一个点扩展为一个四边形，并将四边形的法线方向对准摄像机所在的位置。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// TreeSprite.fx by Frank Luna (C) 2011 All Rights Reserved.

//

// Uses the geometry shader to expand a point sprite into a y-axis aligned

// billboard that faces the camera.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "LightHelper.fx"

cbuffer cbPerFrame

{

 DirectionalLight gDirLights[3];

 float3 gEyePosW;

 float gFogStart;

 float gFogRange;

 float4 gFogColor;

};

cbuffer cbPerObject

{

 float4x4 gViewProj;

 Material gMaterial;

};

cbuffer cbFixed

{

 //

 // 计算方块上的纹理坐标

 //

 float2 gTexC[4] =

 {

 float2(0.0f, 1.0f),

 float2(0.0f, 0.0f),

 float2(1.0f, 1.0f),

 float2(1.0f, 0.0f)

 };

};

// Nonnumeric values cannot be added to a cbuffer.

Texture2DArray gTreeMapArray;

SamplerState samLinear

{

 Filter = MIN\_MAG\_MIP\_LINEAR;

 AddressU = CLAMP;

 AddressV = CLAMP;

};

struct VertexIn

{

 float3 PosW : POSITION;

 float2 SizeW : SIZE;

};

struct VertexOut

{

 float3 CenterW : POSITION;

 float2 SizeW : SIZE;

};

struct GeoOut

{

 float4 PosH : SV\_POSITION;

 float3 PosW : POSITION;

 float3 NormalW : NORMAL;

 float2 Tex : TEXCOORD;

 uint PrimID : SV\_PrimitiveID;

};

VertexOut VS(VertexIn vin)

{

 VertexOut vout;

 // 直接将数据传送到几何着色器

 vout.CenterW = vin.PosW;

 vout.SizeW = vin.SizeW;

 return vout;

}

 // 因为我们将一个点扩展为一个方块（4个顶点），

 // 所以每次调用几何着色器输出顶点的最大数量为4。

[maxvertexcount(4)]

void GS(point VertexOut gin[1],

 uint primID : SV\_PrimitiveID,

 inout TriangleStream<GeoOut> triStream)

{

 //

 // 计算方块在世界空间中的局部坐标系统，

 // 公告牌与y轴对齐，且面对相机。

 //

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

 float3 look = gEyePosW - gin[0].CenterW;

 look.y = 0.0f; // 公告牌与y轴对齐，只在xz平面上面对摄像机

 look = normalize(look);

 float3 right = cross(up, look);

 //

 // 在世界空间中计算顶点坐标

 //

 float halfWidth = 0.5f\*gin[0].SizeW.x;

 float halfHeight = 0.5f\*gin[0].SizeW.y;

 float4 v[4];

 v[0] = float4(gin[0].CenterW + halfWidth\*right - halfHeight\*up, 1.0f);

 v[1] = float4(gin[0].CenterW + halfWidth\*right + halfHeight\*up, 1.0f);

 v[2] = float4(gin[0].CenterW - halfWidth\*right - halfHeight\*up, 1.0f);

 v[3] = float4(gin[0].CenterW - halfWidth\*right + halfHeight\*up, 1.0f);

 //

 // 将方块顶点转换到世界空间，并以三角形带的形式输出

 //

 GeoOut gout;

 [unroll]

 for(int i = 0; i < 4; ++i)

 {

 gout.PosH = mul(v[i], gViewProj);

 gout.PosW = v[i].xyz;

 gout.NormalW = look;

 gout.Tex = gTexC[i];

 gout.PrimID = primID;

 triStream.Append(gout);

 }

}

float4 PS(GeoOut pin, uniform int gLightCount, uniform bool gUseTexure, uniform bool gAlphaClip, uniform bool gFogEnabled) : SV\_Target

{

 // Interpolating normal can unnormalize it, so normalize it.

 pin.NormalW = normalize(pin.NormalW);

 // The toEye vector is used in lighting.

 float3 toEye = gEyePosW - pin.PosW;

 // Cache the distance to the eye from this surface point.

 float distToEye = length(toEye);

 // Normalize.

 toEye /= distToEye;

 // Default to multiplicative identity.

 float4 texColor = float4(1, 1, 1, 1);

 if(gUseTexure)

 {

 // Sample texture.

 float3 uvw = float3(pin.Tex, pin.PrimID%4);

 texColor = gTreeMapArray.Sample( samLinear, uvw );

 if(gAlphaClip)

 {

 // Discard pixel if texture alpha < 0.05. Note that we do this

 // test as soon as possible so that we can potentially exit the shader

 // early, thereby skipping the rest of the shader code.

 clip(texColor.a - 0.05f);

 }

 }

 //

 // Lighting.

 //

 float4 litColor = texColor;

 if( gLightCount > 0 )

 {

 // Start with a sum of zero.

 float4 ambient = float4(0.0f, 0.0f, 0.0f, 0.0f);

 float4 diffuse = float4(0.0f, 0.0f, 0.0f, 0.0f);

 float4 spec = float4(0.0f, 0.0f, 0.0f, 0.0f);

 // Sum the light contribution from each light source.

 [unroll]

 for(int i = 0; i < gLightCount; ++i)

 {

 float4 A, D, S;

 ComputeDirectionalLight(gMaterial, gDirLights[i], pin.NormalW, toEye,

 A, D, S);

 ambient += A;

 diffuse += D;

 spec += S;

 }

 // Modulate with late add.

 litColor = texColor\*(ambient + diffuse) + spec;

 }

 //

 // Fogging

 //

 if( gFogEnabled )

 {

 float fogLerp = saturate( (distToEye - gFogStart) / gFogRange );

 // Blend the fog color and the lit color.

 litColor = lerp(litColor, gFogColor, fogLerp);

 }

 // Common to take alpha from diffuse material and texture.

 litColor.a = gMaterial.Diffuse.a \* texColor.a;

 return litColor;

}

//---------------------------------------------------------------------------------------

// Techniques--just define the ones our demo needs; you can define the other

// variations as needed.

//---------------------------------------------------------------------------------------

technique11 Light3

{

 pass P0

 {

 SetVertexShader( CompileShader( vs\_5\_0, VS() ) );

 SetGeometryShader( CompileShader( gs\_5\_0, GS() ) );

 SetPixelShader( CompileShader( ps\_5\_0, PS(3, false, false, false) ) );

 }

}

technique11 Light3TexAlphaClip

{

 pass P0

 {

 SetVertexShader( CompileShader( vs\_5\_0, VS() ) );

 SetGeometryShader( CompileShader( gs\_5\_0, GS() ) );

 SetPixelShader( CompileShader( ps\_5\_0, PS(3, true, true, false) ) );

 }

}

technique11 Light3TexAlphaClipFog

{

 pass P0

 {

 SetVertexShader( CompileShader( vs\_5\_0, VS() ) );

 SetGeometryShader( CompileShader( gs\_5\_0, GS() ) );

 SetPixelShader( CompileShader( ps\_5\_0, PS(3, true, true, true) ) );

 }

}

## 11.2.4 SV\_PrimitiveID

在本例中，几何着色器包含了一个由**SV\_PrimitiveID**语义修饰的特殊的无符号整数参数。

[maxvertexcount(4)]

void GS(point VS\_OUT gIn[1],

uint primID : SV\_PrimitiveID,

inout TriangleStream<GS\_OUT> triStream);

当指定该语义时，输入汇编器阶段会为每个图元自动生成一个图元ID。当调用draw方法绘制*n*个图元时，第1个图元被标记为0，第2个图元被标记为1，依次类推，直至最后一个图元被标记为*n*−1。图元ID只有在每一次绘制调用中才是唯一的。在本例中，几何着色器没有使用图元ID（虽然它可以使用这个ID）；几何着色器把图元ID写入了输出顶点，传给了像素着色器阶段。像素着色器通过图元ID来建立广告牌和纹理数组之间的对应关系，这些内容将在下一节中讲解。

**注意**：当没有几何着色器时，图元ID参数可以添加到像素着色器的参数列表中：

float4 PS(VertexOut pin, uint primID : SV\_PrimitiveID) : SV\_Target

{

 // Pixel shader body...

}

不过，当有几何着色器时，图元ID参数必须定义在几何着色器的参数列表中。然后，几何着色器可以使用图元ID或者把图元ID传给像素着色器阶段（或者两者皆有）。

**注意**：输入装配器还可以生成一个顶点ID。要使用这个ID，必须在顶点着色器的签名中添加一个由**SV\_VertexID**语义修饰的无符号整数参数。

下面的顶点着色器签名说明了应该如何完成这一工作：

VertexOut VS(VertexIn vin, uint vertID : SV\_VertexID)

{

 // vertex shader body...

}

在每次调用**Draw**方法时，所要绘制的顶点都会被加上0、1、…、*n*−1这样的ID标记，其中*n*表示当前绘图调用中的顶点数量。在调用**DrawIndexed**方法时，顶点ID是相应的顶点索引值。