# 6.10 山峰与河谷演示程序

本章还包含了一个“山峰与河谷”的例子。它使用了与颜色立方体演示程序相同的Direct3D方法，只是它绘制的几何体更复杂一些。它主要讲解的是如何使用代码来生成三角形网格；这种几何体在实现地形渲染和水体渲染时非常有用。

实数函数*y* = *f*(*x , z*)可以生成一个“漂亮的”曲面。我们可以通过构造一个*xz*平面上的网格来模拟该曲面，其中每个四边形都由两个三角形构成。然后，我们将每个网格点代入该函数；参见图6.8。

****

**图6.8 （上）建立*xz*平面上的网格。（下）将每个网格点代入函数*f*(*x* ,*z*)，得到*y*坐标。通过将大量的点(*x*, *f*(*x, z*), *z*)连接起来，即可形成上述曲面。**

## 6.10.1 生成网格顶点

下面的主要任务是创建*xz*平面上的网格。一个包含*m*×*n*个顶点的网格可以生成(*m* − 1)× (*n*− 1)个多边形（或单元格），如图6.9所示。每个多边形由两个三角形组成，一共2×(*m* − 1)× (*n*− 1)个三角形。如果网格的宽度为*w*、深度为*d*，则*x*轴方向上的单元格间距为*dx* = *w*/(*n*-1)、 轴方向上的单元格间距为*dz*=*d*/(*m*-1)。我们从左上角开始生成顶点，逐行计算每个顶点的坐标。在*xz*平面上，第*ij*个网格顶点的坐标为：

**v*ij*= (−0.5w + j ∙ dx , 0.0 , 0.5d – i ∙ dz)**

****

**图6.9 网格结构。**

下面的代码实现了这一工作。

void GeometryGenerator::CreateGrid(float width, float depth, UINT m, UINT n, MeshData& meshData)

{

 UINT vertexCount = m\*n;

 UINT faceCount = (m-1)\*(n-1)\*2;

 //

 // 创建顶点

 //

 float halfWidth = 0.5f\*width;

 float halfDepth = 0.5f\*depth;

 float dx = width / (n-1);

 float dz = depth / (m-1);

 float du = 1.0f / (n-1);

 float dv = 1.0f / (m-1);

 meshData.Vertices.resize(vertexCount);

 for(UINT i = 0; i < m; ++i)

 {

 float z = halfDepth - i\*dz;

 for(UINT j = 0; j < n; ++j)

 {

 float x = -halfWidth + j\*dx;

 meshData.Vertices[i\*n+j].Position = XMFLOAT3(x, 0.0f, z);

 meshData.Vertices[i\*n+j].Normal = XMFLOAT3(0.0f, 1.0f, 0.0f);

 meshData.Vertices[i\*n+j].TangentU = XMFLOAT3(1.0f, 0.0f, 0.0f);

 // Stretch texture over grid.

 meshData.Vertices[i\*n+j].TexC.x = j\*du;

 meshData.Vertices[i\*n+j].TexC.y = i\*dv;

 }

 }

}

**GeometryGenerator**是一个工具类，用于生成诸如网格、球、圆柱体、盒子之类的几何形状，在本书的其他示例中都会用到这些形状。这个类在系统内存中生成数据，我们必须将这些数据复制到顶点和索引缓冲中。**GeometryGenerator**创建的某些顶点数据在后面的章节中才会用到，这个演示程序不会用到，所以也无需将这些数据复制到顶点缓冲中。**MeshData**结构体用于存储顶点和索引的集合列表。

class GeometryGenerator

{

public:

 struct Vertex

 {

 Vertex(){}

 Vertex(const XMFLOAT3& p, const XMFLOAT3& n, const XMFLOAT3& t, const XMFLOAT2& uv)

 : Position(p), Normal(n), TangentU(t), TexC(uv){}

 Vertex(

 float px, float py, float pz,

 float nx, float ny, float nz,

 float tx, float ty, float tz,

 float u, float v)

 : Position(px,py,pz), Normal(nx,ny,nz),

 TangentU(tx, ty, tz), TexC(u,v){}

 XMFLOAT3 Position;

 XMFLOAT3 Normal;

 XMFLOAT3 TangentU;

 XMFLOAT2 TexC;

 };

 struct MeshData

 {

 std::vector<Vertex> Vertices;

 std::vector<UINT> Indices;

 };

…

};

## 6.10.2 生成网格索引

在完成顶点的计算之后，我们必须通过索引来定义网格三角形。我们再次从左上角开始逐行遍历每个四边形，通过计算索引来定义构成四边形的两个三角形。如图6.10所示，对于一个由*m*×*n*个顶点构成的网格来说，两个三角形的线性数组索引为：

△ABC = (i∙n+j , i∙n + j + 1, (i + 1) ∙n + j)

△CBD = ((i +1) ∙n + j , i∙n + j + 1 ∙ (i + 1) ∙n + j + 1)

****

**图6.10 第*ij*个四边形的顶点的索引。**

下面是对应的代码：

meshData.Indices.resize(faceCount\*3); // 3 indices per face

// 遍历所有四边形并计算索引

UINT k = 0;

for(UINT i = 0; i < m-1; ++i)

{

 for(UINT j = 0; j < n-1; ++j)

 {

 meshData.Indices[k] = i\*n+j;

 meshData.Indices[k+1] = i\*n+j+1;

 meshData.Indices[k+2] = (i+1)\*n+j;

 meshData.Indices[k+3] = (i+1)\*n+j;

 meshData.Indices[k+4] = i\*n+j+1;

 meshData.Indices[k+5] = (i+1)\*n+j+1;

 k += 6; // next quad

 }

}

## 6.10.3 代入高度函数

创建了网格之后，我们要从MeshData中提取顶点元素，将平面网格转换为代表山丘的曲折表面，并基于顶点的高度（*y*坐标）设置它们的颜色。

struct Vertex

{

 XMFLOAT3 Pos;

 XMFLOAT4 Color;

};

void HillsApp::BuildGeometryBuffers()

{

 GeometryGenerator::MeshData grid;

 GeometryGenerator geoGen;

 geoGen.CreateGrid(160.0f, 160.0f, 50, 50, grid);

 mGridIndexCount = grid.Indices.size();

 //

 // 在每个顶点上附加高度函数。此外，还根据顶点的高度设置它们的颜色：

 // 沙滩为沙的颜色，小山为绿色，山顶为白色的雪。

 //

 std::vector<Vertex> vertices(grid.Vertices.size());

 for(size\_t i = 0; i < grid.Vertices.size(); ++i)

 {

 XMFLOAT3 p = grid.Vertices[i].Position;

 p.y = GetHeight(p.x, p.z);

 vertices[i].Pos = p;

 // 根据顶点高度设置颜色

 if( p.y < -10.0f )

 {

 // 沙滩色

 vertices[i].Color = XMFLOAT4(1.0f, 0.96f, 0.62f, 1.0f);

 }

 else if( p.y < 5.0f )

 {

 // 淡绿色

 vertices[i].Color = XMFLOAT4(0.48f, 0.77f, 0.46f, 1.0f);

 }

 else if( p.y < 12.0f )

 {

 // 深绿色

 vertices[i].Color = XMFLOAT4(0.1f, 0.48f, 0.19f, 1.0f);

 }

 else if( p.y < 20.0f )

 {

 // 棕色

 vertices[i].Color = XMFLOAT4(0.45f, 0.39f, 0.34f, 1.0f);

 }

 else

 {

 // 白色

 vertices[i].Color = XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f);

 }

 }

 D3D11\_BUFFER\_DESC vbd;

 vbd.Usage = D3D11\_USAGE\_IMMUTABLE;

 vbd.ByteWidth = sizeof(Vertex) \* grid.Vertices.size();

 vbd.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER;

 vbd.CPUAccessFlags = 0;

 vbd.MiscFlags = 0;

 D3D11\_SUBRESOURCE\_DATA vinitData;

 vinitData.pSysMem = &vertices[0];

 HR(md3dDevice->CreateBuffer(&vbd, &vinitData, &mVB));

 //

 // 将所有网格的索引放入一个索引缓冲中

 //

 D3D11\_BUFFER\_DESC ibd;

 ibd.Usage = D3D11\_USAGE\_IMMUTABLE;

 ibd.ByteWidth = sizeof(UINT) \* mGridIndexCount;

 ibd.BindFlags = D3D11\_BIND\_INDEX\_BUFFER;

 ibd.CPUAccessFlags = 0;

 ibd.MiscFlags = 0;

 D3D11\_SUBRESOURCE\_DATA iinitData;

 iinitData.pSysMem = &grid.Indices[0];

 HR(md3dDevice->CreateBuffer(&ibd, &iinitData, &mIB));

}

****

**图6.11 山峰演示程序的截图。**

这个程序所用的函数*f*(*x*,*z*)由以下代码给出：

float HillsApp::GetHeight(float x, float z)const

{

 return 0.3f\*( z\*sinf(0.1f\*x) + x\*cosf(0.1f\*z) );

}

这个函数生成的图形看起了就像是山峰和山谷（见图6.11），程序的其他部分与上一节颜色正方体的示例类似。