# 19.1高度图

地形渲染的实现思路是：以一个平面网格为基础（图19.1上图），通过调整网格顶点的高度（即，*y*坐标）模拟地形起伏，使网格表现出从高山到河谷的平滑过渡（图19.1中图）。当渲染网格时，我们可以使用一幅精致的纹理来表现沙滩、草地、石壁、雪山等自然地貌（图19.1下图）。

****

**图19.1 （上图）三角形网格。（中图）通过为三角形网格指定顶点高度来模拟高山与河谷之间的平滑过渡。（下图）带有光照和纹理的地形。**

**学习目标**

1．学习如何为一个地形生成高度信息，形成高山与河谷之间的平滑过渡。

2．了解如何为地形添加纹理。

3．使用硬件曲面细分绘制一个带有连续LOD（level of detail）的地形。

4．了解如何在地形表面上放置摄像机或其他物体。

我们使用高度图（heightmap）描述地形中的山川起伏。高度图是一个矩阵，它的每个元素指定了地形网格中的一个特定顶点的高度。也就是，每个网格顶点都有一个与之对应的高度图元素，第*ij*个高度图元素提供了第*ij*个顶点的高度。在图像处理软件中，高度图一般显示为灰阶图像（grayscale map），其中，黑色表示最低海拔高度，白色表示最高海拔高度，不同亮度的灰色表示介于中间的海拔高度。图19.2展示了几个高度图的例子，以及使用这些高度图生成的地形。

****

**图19.2 高度图的例子。可以看到，高度图中的明暗区域描述了不同的高度变化，生成了不同的地形表面。**

当我们在磁盘上存储高度图时，通常只会为高度图中的每个元素分配一个字节的存储空间，也就是说，高度的取值范围是[0,255]。该范围足以表达地形中的高度变化。不过，在实际应用中，我们可能需要对该范围进行扩展，使它与我们的3D空间大小匹配。例如，当我们将3D空间的测量单位设为英寸时，一个字节根本无法表示小于0英寸或大于255英寸的高度值。基于上述原因，我们将使用浮点数存储那些载入到应用程序中的高度元素。这样，我们就可以有效地扩展[0,255]区间，把它缩放到任何大小都不成问题；而且，这样还可以对高度图进行过滤，生成整数之间的高度值。

**注意**：我们在6.11节中使用数学函数创建过一个“地形”，它是一个以代码来生成地形的例子。不过，它的用处不大，因为函数根本无法精确描述你实际想要的地形。高度图具有很好的灵活性，它可以由美术师来编辑和绘制。

## 19.1.1 创建高度图

高度图可以由代码或图像处理软件（比如Adobe Photoshop）生成。我们可以在Photoshop中使用滤镜生成各种不同的杂乱图案，作为高度图的底图，然后使用各种绘图工具做一些手工修改，从而得到一幅不错的高度图。而且，模糊滤镜可以有效地缓解高度图中的粗糙边缘。

你也可以使用某些工具软件来创建高度图，比如Terragen（<http://www.planetside.co.uk/terragen/>）可以通过程序生成高度图，它还提供了一个修改高度图的工具，也可以输出软件生成的高度图，然后导入到另一个图像处理程序（例如Photoshop）中进行手工修改。Bryce（<http://www.daz3d.com/i.x/software/bryce/>）也自带很多生成高度图的程序算法和高度图编辑器。Dark Tree（<http://www.darksim.com/>）是一个强大的纹理设置工具，也可用于生成高度灰度图。

在完成高度图的绘制之后，你需要把它保存为一个8位RAW文件。RAW文件只是把图像的二进制数据直接保存成文件，不添加任何编码信息。这样我们可以很容易地把图像载入到程序中。如果你使用的软件在保存RAW文件询问是否加入一个文件头，那么请选择“不加入文件头”。图19.3展示了Terragen的导出对话框。

****

**图19.3 （左图）地形生成器以算法方式生成随机地形，它还提供了画刷工具，可以让你手工雕刻地形。（右图）Terragen的导出对话框。注意，请在导出方式中选择8位RAW格式。**

**注意**：高度图不一定存储为RAW格式；你可以根据需要选择任何一种图像格式。RAW格式只是我们可以使用的图像格式中的一种。我们使用RAW格式，主要是因为很多图像处理软件都支持这种格式，而且它用起来非常简单，我们可以很容易地把RAW文件载入到演示程序中。本书中的演示程序使用8位RAW文件（即，高度图中的每个元素均为8位整数）。

**注意**：如果256个高度级对于你的需要来说过于粗糙，那么可以考虑使用16位高度图，通过16位整数来存储高度值。Terragen支持16位RAW高度图。

## 19.1.2 载入RAW文件

由于RAW文件是一个连续的字节块（其中的每个字节是一个高度图元素），所以我们可以很容易地使用std::ifstream::read 方法把它读入内存中来。下面是具体的实现代码：

void Terrain::LoadHeightmap()

{

 // 存储高度数据的数组

 std::vector<unsigned char> in( mInfo.HeightmapWidth \* mInfo.HeightmapHeight );

 // 打开文件.

 std::ifstream inFile;

 inFile.open(mInfo.HeightMapFilename.c\_str(), std::ios\_base::binary);

 if(inFile)

 {

 // 读取RAW字节.

 inFile.read((char\*)&in[0], (std::streamsize)in.size());

 // Done with file.

 inFile.close();

 }

 // 将数组中的数据复制到一个浮点数组并进行缩放

 mHeightmap.resize(mInfo.HeightmapHeight \* mInfo.HeightmapWidth, 0);

 for(UINT i = 0; i < mInfo.HeightmapHeight \* mInfo.HeightmapWidth; ++i)

 {

 mHeightmap[i] = (in[i] / 255.0f)\*mInfo.HeightScale;

 }

}

**mInfo**是**Terrain**类的一个成员变量，它是一个**InitInfo**结构体实例。该结构体描述了地形的各种属性：

struct InitInfo

{

 std::wstring HeightMapFilename; // 高度图文件名称

 // 地形纹理的文件名称

 std::wstring LayerMapFilename0;

 std::wstring LayerMapFilename1;

 std::wstring LayerMapFilename2;

 std::wstring LayerMapFilename3;

 std::wstring LayerMapFilename4;

 std::wstring BlendMapFilename;

 float HeightScale;

 UINT HeightmapWidth;

 UINT HeightmapHeight;

 float CellSpacing;

};

注意：读者可以回顾5.11节，查阅有关网格构造的更多信息。

****

**图19.4 网格属性。**

## 19.1.3 平滑处理

8位高度图的一个缺点是它只能表示256个离散的高度级。因此，我们无法模拟图19.5a所示的高度值；只能得到图19.5b的高度值。这样创建出来的地形会比我们实际想要的地形“粗糙”。而且，当高度值舍入之后，我们将无法恢复原始的高度值。不过，我们可以对图19.5b进行平滑处理，使它接近于图19.5a。

****

**图19.5 (a)[0, 255]区间中的浮点高度值。(b)高度值舍入为近似整数。**

那么，我们所要做的就是通过读取RAW文件，把高度图载入到内存中。然后把字节数组复制给一个浮点数组，提高数值存储的精确度。接着，在浮点高度图上应用一个滤镜，对高度图进行平滑处理，降低邻接元素之间的高度差。该滤镜算法非常简单，它通过计算一个像素及其8个邻接像素的平均值来生成该像素的新颜色（参见图19.6）：



****

**图19.6 第*ij*个顶点的高度等于高度图中的第*ij*个像素及其8个邻接像素的平均值。**

注意，在高度图边缘上的像素没有8个邻接像素，我们在处理这些像素时只能按照邻接像素的实际数量来计算平均值。

下面的函数用于计算高度图中的第*ij*个像素的平均值：

bool Terrain::InBounds(int i, int j)

{

 // 如果ij是有效的索引则返回true; 否则返回false.

 return

 i >= 0 && i < (int)mInfo.HeightmapHeight &&

 j >= 0 && j < (int)mInfo.HeightmapWidth;

}

float Terrain::Average(int i, int j)

{

 // Function computes the average height of the ij element.

 // It averages itself with its eight neighbor pixels. Note

 // that if a pixel is missing neighbor, we just don't include it

 // in the average--that is, edge pixels don't have a neighbor pixel.

 //

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

 // | 1| 2| 3|

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

 // |4 |ij| 6|

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

 // | 7| 8| 9|

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

 float avg = 0.0f;

 float num = 0.0f;

 // Use int to allow negatives. If we use UINT, @ i=0, m=i-1=UINT\_MAX

 // and no iterations of the outer for loop occur.

 for(int m = i-1; m <= i+1; ++m)

 {

 for(int n = j-1; n <= j+1; ++n)

 {

 if( InBounds(m,n) )

 {

 avg += mHeightmap[m\*mInfo.HeightmapWidth + n];

 num += 1.0f;

 }

 }

 }

 return avg / num;

}

当像素在高度图上时，函数**inBounds**返回**true**；反之返回**false**。这样，当我们对一个边缘像素的邻接像素进行采样时，如果某个邻接像素不在高度图范围内，那么**inBounds**会返回false，我们不把它计入平均值——因为它不存在。

要对整个高度图进行平滑，我们只需要为每个高度图像素计算平均值：

void Terrain::Smooth()

{

 std::vector<float> dest( mHeightmap.size() );

 for(UINT i = 0; i < mInfo.HeightmapHeight; ++i)

 {

 for(UINT j = 0; j < mInfo.HeightmapWidth; ++j)

 {

 dest[i\*mInfo.HeightmapWidth+j] = Average(i,j);

 }

 }

 // 将过滤后的值替换旧值.

 mHeightmap = dest;

}

## 19.1.4 高度图Shader资源视图

我们在下一节会看到，要支持曲面细分和位移映射，需要在shader程序中采样高度图。所以，需要创建一个高度图的shader资源视图。创建资源视图现在我们已经很熟悉了，唯一的一个技巧就是为了节省内存，我们使用了16位浮点数代替了32位。可以使用XNA库**XMConvertFloatToHalf**函数将32位浮点数转换为16位浮点数。

void Terrain::BuildHeightmapSRV(ID3D11Device\* device)

{

 D3D11\_TEXTURE2D\_DESC texDesc;

 texDesc.Width = mInfo.HeightmapWidth;

 texDesc.Height = mInfo.HeightmapHeight;

 texDesc.MipLevels = 1;

 texDesc.ArraySize = 1;

 texDesc.Format = DXGI\_FORMAT\_R16\_FLOAT;

 texDesc.SampleDesc.Count = 1;

 texDesc.SampleDesc.Quality = 0;

 texDesc.Usage = D3D11\_USAGE\_DEFAULT;

 texDesc.BindFlags = D3D11\_BIND\_SHADER\_RESOURCE;

 texDesc.CPUAccessFlags = 0;

 texDesc.MiscFlags = 0;

 // HALF定义在xnamath.h中，用来存储16位浮点数.

 std::vector<HALF> hmap(mHeightmap.size());

 std::transform(mHeightmap.begin(), mHeightmap.end(), hmap.begin(), XMConvertFloatToHalf);

 D3D11\_SUBRESOURCE\_DATA data;

 data.pSysMem = &hmap[0];

 data.SysMemPitch = mInfo.HeightmapWidth\*sizeof(HALF);

 data.SysMemSlicePitch = 0;

 ID3D11Texture2D\* hmapTex = 0;

 HR(device->CreateTexture2D(&texDesc, &data, &hmapTex));

 D3D11\_SHADER\_RESOURCE\_VIEW\_DESC srvDesc;

 srvDesc.Format = texDesc.Format;

 srvDesc.ViewDimension = D3D11\_SRV\_DIMENSION\_TEXTURE2D;

 srvDesc.Texture2D.MostDetailedMip = 0;

 srvDesc.Texture2D.MipLevels = -1;

 HR(device->CreateShaderResourceView(hmapTex, &srvDesc, &mHeightMapSRV));

 // SRV saves reference.

 ReleaseCOM(hmapTex);

}