# Mesh View Terrain Rendering

潘李亮

2010/6/19

本文借鉴目前的地形渲染和地形数据管理调度的解决方案，提出一种简单易实现的无限大地形的数据调度和加载方案。

# 综述

大地形渲染技术在游戏和GIS/VR等领域中有重要的应用，地形渲染的技术在最近十多年中随着图形硬件技术的发展也经历很多次革新，从最早的专注于降低每一个不必要渲染三角形的ROAM算法，到后来为了提高每个batch三角形数量而提出的ROAM 2.0，GeoMipmap，Inter-Lock等算法，以及最近流行的专注于数据管理和调度Mega-Texture/Virtual Texture算法。

因为硬件的发展，三角形数量不再是渲染地形的瓶颈。相反，古老的LoD算法中，过多的draw Primitive调用和超大的数据如何有效的从外存调度到显存已经成了目前大地形渲染的核心问题。

并且，在Gis应用中，通常表示地形高度的DEM和表示地表贴图的影像数据如何形成一套有效的，一致的管理算法也是我们需要考虑的问题。

本文的目是提出一套简单，有效的方案，用一致的管理DEM数据（高度图）以及遥感影响数据（地表纹理贴图）。

# 现有方案

John Camark 的id-tech5 的技术demo中给我们展示了一个无比细致的室外场景，该演示使用了id-software的最新技术，Mega-Texture, Mega Textue严格的来说是一种纹理的调度和管理方案，但是Mega-Texture最早的起源于SGI的Clipmap[参考文献1]技术，而Clipmap技术经过多年的发展，已经被成功移植到GPU上[参考文献2]。该方案是一个非常有效的管理和调度超大规模地形和影像数据的方法。GPU-Base clipmap的基本思想是将高度图以clipmap的方式进行管理，并在cpu端生成14个block的vertex-buffer，这些vertex buffer中只保存(x,y)的坐标，高度z则在vertex shader中通过采样clipmap来获取。生成14个block的原因一方面在于LoD的考虑，另一方面方便进行视锥裁剪。

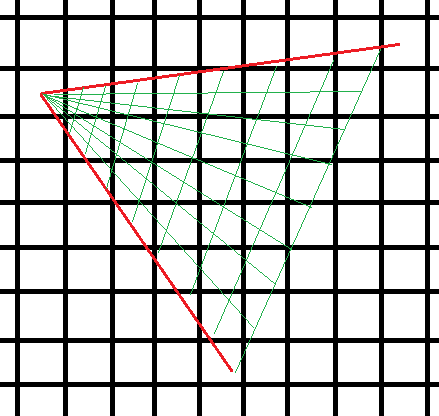
这里我还要提一个非常古老的地形技术—Voxel，当年风靡一时的对战游戏<Delta force>给我的震撼依然难忘，该游戏使用的地形渲染技术正是Voxel，在当年使用Voxel来绘制地形是3D硬件没有普及的时候的不得已而为之。不过voxel的有些特性特别适合做LoD，在Voxel算法中，根据当前屏幕的分辨率，确定体素采样的分辨率，比如屏幕的分辨率为1280x720，那么每一条采样线的分辨率就为720个点，从视点开始，一直采样到远裁剪面，视线方向上的采样间隔随着采样线离视点的距离而变宽。而离开视点越远，每条采样线跨过的距离也越长（透视原因），因此离开视点越远，采样点越稀疏，从而实现了LoD。

# Mesh View方式

## Mesh View

考察现在DX9.0c/DX10.1/DX11的硬件特性，displacement map早已经绝对的主流配置，这意味着使用GPU的clipmap技术完全已经是具有实用价值的。

上一节中提到的Voxel技术其实可以认为是通过一个规则的Mesh, 该Mesh如下图



红色的线为视锥的区域，绿色的线组成一个网格，计算出这个绿色网格中每一个点的高度值，最终的结果，就是这个地形渲染的一个View。

结合Voxel和Clipmap，我们将DEM保存成R32F格式的float texture，使用一个或者多个任意的Mesh，并把这些Mesh放在以视点为中心的位置，通过这些Mesh上(x,y)的值来对DEM进行采样，生成一个全新的Mesh。只要这些Mesh满足远离中心位置的越稀疏，靠近中心越稠密，就能形成LoD。

## 数据调度和渲染

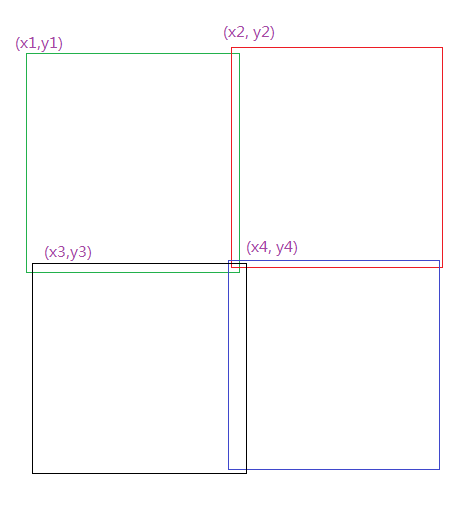
数据调度算法直接影响到数据可以以何种方式去组织，在Clipmap算法中，除了最高层的Clipmap Pryramid以外，Clipmap stack中的每一层，都是有重复的，如果一个点落在金字塔的底层，那么它一定可以在分辨率低的那一层找到。应该说，这个规则也限制了clipmap更自由的调度---在更新的时候，基本上Clipmap stack里的每一个纹理都需要更新。



图片来自参考文献1

Clipmap的数据调度的另外一个限制是选择的Clip size直接影响到该系统能调度的最大数据量。2048x2048大小的Clipmap大约需要132M的显存，最大能调度68T的纹理（单张68T纹理，分块保存，这也是改进后的算法称为Mega-Texture的原因）。这个大小的纹理也许对游戏已经非常足够了，但是对类似一个国家的地表渲染，也许会是一个问题。

这里我们采用更简单的方法，除了Clipma Pyramid以外的数据，这些数据的每一块都带有自己的位置和大小信息，如下图



地形/影像数据块的排列。

各个块之间允许有重叠现象，甚至每个块的大小还可以不一样，这点非常符合卫星遥感图像的需求，遥感图像通常不会严格的重合和无重叠。

根据硬件能力不同，同一个时刻，只保存视点周围3x3，或者 5x5个左右的地形块。这些地形块组成一个地形块缓冲池。根据视点的位置和移动趋势，动态的加载和卸载这个池中的数据。在渲染的时候，vertex shader会判断每一个输入的点，根据缓冲池中的地形块的位置和大小信息，决定从哪一块中进行采样。伪代码如下：

PS\_INPUT main( VS\_INPUT input ){

PS\_INPUT output = (PS\_INPUT)0;

float3 \_Pos = float3(CameraPos.xy , 0.0) + input.Pos.xyz;

Pos = mul( matWorld , float4(\_Pos ,1) ) .xyz ;

float \_h = 0.0f; float div = 0.0f;

for(int i = 0 ; i < nDEM.x ; i ++ ){

if( PtInRegin(\_Pos , DEMRegion[i] ) {

float w = DEMRegion[i].z - DEMRegion[i].x;

float h = DEMRegion[i].w - DEMRegion[i].y;

float \_x = \_Pos.x - DEMRegion[i].x;

float \_y = \_Pos.y - DEMRegion[i].y;

float2 texCoord = float2( \_x /w , \_y / h ) ;

float \_v = DEMTextures[i].SampleLevel(DefaultClampSampler, texCoord , 0 ).x;

if(\_v > 0 ){ div += 1.0f; \_h += \_v; }

}

}

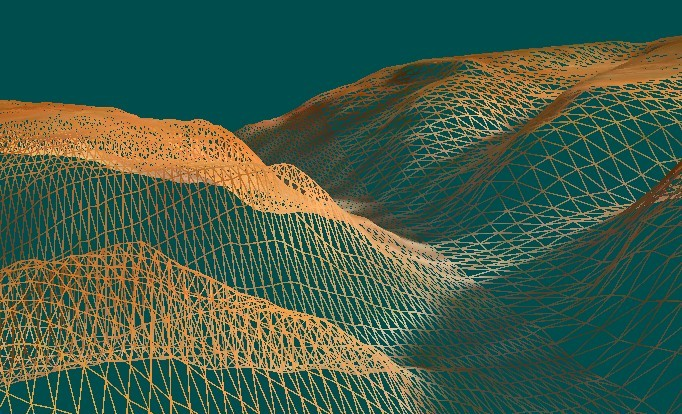
if(div > 0.0f) output.z = \_h / div ; //计算高度

else output.z = 0.0f; //该点不落在任何一块地形上，给个默认值

//….. 其它运算

}

对于纹理数据，可以用同样的算法，唯一不同之处在于，vertex shader的采样是displacement map，而pixel shader中则是texture map。下图是渲染结果，采用的是一个圆形的Mesh.



采用圆形Mesh的渲染结果

## 抖动

抖动也是本方案需要重点考虑的问题，网格在世界空间中移动，每个点的高度如果在每次移动中都变化，则会让渲染结果有抖动现象。简单的解决方法是，只在移动一定距离后才移动Mesh。

## 优势

本方案的最大优势在于实现简单，对数据的限制比较少，并且能用统一的算法对DEM和影像进行处理和渲染。由于采用的Mesh本身可以不存在裂缝，因此在任何情况下，该算法都不会产生裂缝，并可以根据硬件能力的不同，动态的决定使用不同细致程度和不同形状的Mesh，本人在ATI HD 5770的显卡上使用300x300的Grid Mesh(18万个三角形)渲染的时候，能轻松达到170fps。

另外，因为相对clipmap中的block，我们采用的是mesh，这更符合一般渲染引擎的场景管理算法，在需要把地形系统与其它的模块进行整合的时候将会变得非常的自然。同样的，block的裁剪也可以简单的使用已有的场景管理器对mesh的裁剪算法而不需要专门实现针对地形块的裁剪版本。

# 优化以及后续工作

## 优化

本文成文之际，我的Mesh View地形渲染刚刚集成进我的引擎中，可以优化和改进的方地方还非常的多。

首先，我们考虑如何减少纹理数量的问题，我们知道D3D9最多只支持8个纹理， 而一般我们至少要在缓冲池中保存9个以上的纹理，这意味着如果不减少纹理的数量，我们将无法使用d3d9来实现该算法。我们可以考虑动态的将一个大纹理划分为1024x1024的区域，每一个地形块或者影像块加载的时候都被重采样成该分辨率的，使用LockRect来更新某个区域。一般主流硬件至少都支持4096x4096分辨率（大部分支持8192x8192）。采用该方法，只需要一个4096x4096的纹理，就能满足一个4x4缓冲池的需求。

另外，Direct3D 10支持Texture Array。采用该方法，也能减少纹理个数的使用。

如果地形系统需要支持高级光照，那么如何生成地形的法向量，也是一个需要考虑的问题，我们知道通过对高程的差分运算，可以得到高程的法向量，我们可以在Vertex shader或者geomerty shader中来完成这个动作，不过考虑到法向量需要进行平滑运算，在计算的时候，应该取改点周围的点来进行差分运算，以保证法向量的连续和平滑。

此外，如何减少纹理采样次数，如何减少dependent vertex fetch的次数，也是一个非常值得优化的地方。

## 后续工作

Tessellation：DX11中引入的Tessellation是非常有效的提高几何精细度的技术，在本文的方案中，加入tessellation，可以有效的提高地表的细致程度，同时还可以根据不同的参数，动态的调整某个区域的LoD等级。Tessellation可以用DX11的tessellation单元，也可以使用DX10.1的compute shader。同时DX10.0的geometry shader也可以用来生成简单的网格。

# 参考文献

1. <The Clipmap: A Virtual Mipmap>,Christopher C. Tanner, Christopher J. Migdal, and Michael T. Jones,Silicon Graphics Computer Systems,1998
2. Terrain Rendering Using GPU-Based Geometry Clipmaps,<GPU GEMS 2>,Hoope,NVIDIA,2005
3. NVIDIA, SDK10,clipmaps