ID Tech 5 中"Megatexture"针对地形的 D3D9 基本实现原理

姚勇
H3D
2007－8

本文对 ID SOFTWARE 使用的"megatexture"技术，针对地表贴图应用的 D3D9 的硬件实现，做了概要式的介绍。并对技术的实现，优化，扩展，和实用性进行了刨析。转载请注名作者和 H3D。

一，综述

ID TECH5是PC 3D游戏之父和DOOM之父John Carmack最新推出的一项新的游戏制作技术。核心内容为一种命名为"Megatexture"的动态贴图管理技术。实际上是一种动态卸载和装载渲染资源的技术统称。Idsoftware DOOM3引擎 的lighting/shadow/shader方案统一以后，同UNREAL3一样，面对次世代游戏对资源的无穷尽需求，需要尽快建立起一种实际工程技术方案，加快构建3D虚拟现实的生产。

图片来源：Quake Wars

这个技术最主要方向就是要把VR3D建造从有限的显卡内存中释放出来。把美术从繁重的贴图重用绘制中解脱出来。因为VR3D主要资源消耗都在贴图上，所以这个技术主要针对

texture。同时geometry也可以作为动态资源。 一旦rendering相关资源作为动态流进行管理，所带来的另一个好处就是可以让VR世界变为动态的。随时改变texture/geometry。

总得来讲，ID这项针对工程应用技术主要致力于解决：1-3DVR资源需求的爆炸增长和有限显卡显存的矛盾；2-美术资源需求的爆炸增长，以及生产力低、成本高昂的矛盾

在ID TECH5演示中，实时演算播放了一个细节非常丰富的室外地形，同时结合了一个小型的室内场景。所有模型和贴图制作，按照Carmack介绍，只让几个美术花了4天时间。就制作出20G的游戏美术资源。而游戏引擎负责把几十G的资源动态装载到显卡进行渲染。

本文就以下几个问题做初步讲述：1，综述地形渲染以及贴图技术；2，Clipmap介绍；3，基于D3D9的解决方案描述；4，优化和扩展；5，方案可行性

1-1，Mega Texture

Megatexture 的具体实现，在 DOOM3 中的地形绘制已经加入。具体实现细节简单描述如下：

DOOM3 的地形贴图最大支持 32768x32768 。即 5.46 G。在 DOOM3 中，以 128x128 大小的块进行储存便于快速读取。以最大 2048x2048 的 clipmap 尺寸进行处理。其实可以计算出在 D3D9 标准的显卡上，normalmap+6 clipmap stack 的最大理论支持贴图数据大小为（2048 * 2^6 ）^2 *4= 68T bytes. 这个大小针对现在的 PC 硬盘是绰绰有余的。

DOOM3 使用这张全局 metatexture 纹理渲染地形，不使用 LOD，GEOM-MORPHING。

1-2，传统地形贴图渲染

当使用 Tiling 方式绘制贴图，等于把贴图图素(texel)密度成倍扩大。以重复纹理信息量的代价保证在一定分辨率下屏幕上贴图的精细程度。在绘制大范围地形时，使用不 Tiling 的全局贴图无法表现贴地处的地表细节。所以使用一些可以重复(Tiling)的不规则图案的贴图来进行模拟。

1-2-1 Tiling 地形贴图

由于地形贴图尺寸很大，所以无法使用全局贴图。而是把多层 Tiling 的纹理使用 alpha 通道互相融合起来。诸如 WOW[1]，天堂 2 等大型室外地形渲染多采用此技术。此技术在每一层 Tiling 贴图依然用到了一张全局 alpha 贴图。以及一张全地形唯一的静态光影贴图。在 multitexture 中根据显卡的 multitexture 处理单元数量，进行 multi-pass 和 multitexture 的混合渲染。

比如，混合绿草，土路，野花的一块室外地形，需要至少 3 层地表贴图。分别是一块 256x256 的草贴图，128x128 的土路贴图，和 128x128 的野花贴图。每层纹理使用 Tiling 模式进行绘制，在绘制的同时，每层纹理带有一张 ALPHA 灰度图（第一层不用）。用来表明本层纹理在融合中所占的比重。把 3 张地表贴图，2 张 ALPHA 贴图，以及一张 LIGHTMAP 贴图混合起来，就有如下地表效果。可以看出土路和草地之间的均匀过渡。

图片来源：H3D

如果进行优化，4 层 ALPHA 可以混合称为一张贴图。Lightmap 一张。那么也至少需要 2 张全局贴图进行地表绘制。

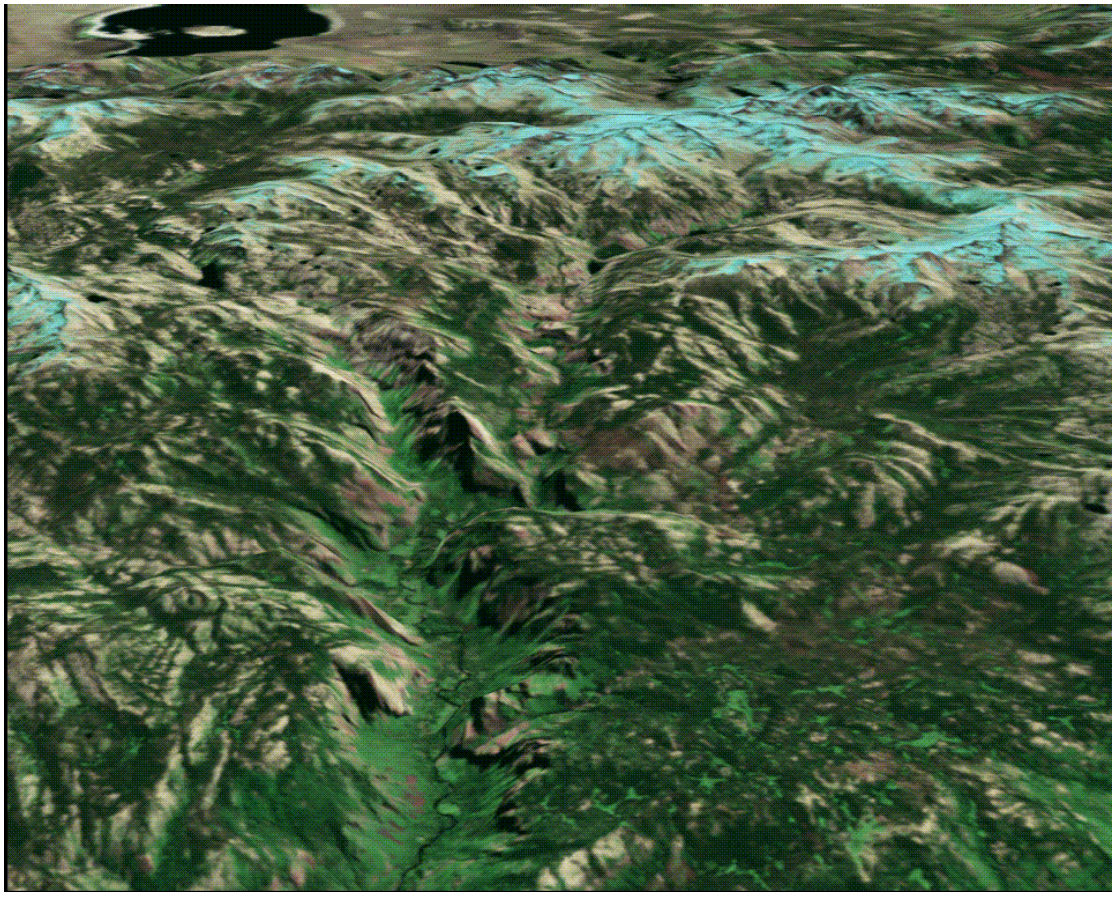
1-2-2 全局地表纹理混合 Tiling 细节贴图[2]

在更大广度的地形渲染，有时候需要一种更加快速的方法。在 FAR CRY 引擎中，使用了这个技术。简单描述为，使用一张全局地表纹理，渲染地表所有植被和光影信息。在离视点近的范围内，使用一张表现当地地表细节的贴图，以 Tiling 方式进行融合叠加。这样在细节度上，以 Tiling 方式的贴图以高细节图素纹理，掩盖了下面那一层全局贴图的粗糙颗粒。

总的来讲，分层 Tiling Alpha 混合地表贴图，和全局地表混合 Tiling 细节贴图，两种方法都是靠贴图的 Tiling 来增大视点附近贴图图素的密度。防止比屏幕分辨率小的 2 图素之间进行线性差值。影响真实感。

二，Clip maps 基本介绍

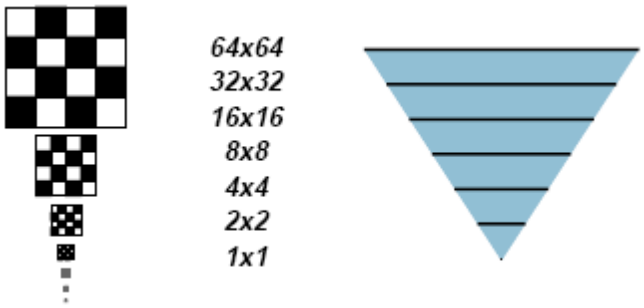
针对受显卡显存制约，无法使用大尺寸贴图进行绘制，1998年SGI公司发表了一篇名为<The Clipmap: A Virtual Mipmap>[3]的论文。论文里详细阐述了一种以有限硬件性能，来实现相对无限大贴图的渲染。论文是一整套硬件与软件结合的方案。它可以实现以一平方米一个图素，用单一贴图进行整个地球表面的渲染。并且结合硬件数据传输带宽，以及相应优化方法，鲁棒的保证以任意速度漫游时保持在60Hz(针速率)。下面先对此技术做一定简要介绍。在第三节给出基于D3D9硬件的解决方案思路。下图为Clipmap实现的美国国家地图实时漫游的一小部分。美国约塞米蒂国家公园(Yosemite National Park)南部的一半。整个美国使用一张170G的贴图。在显卡中的clipmap贴图缓存为16M。这个贴图内存用量在今天的PC硬件上是完全可以实现的。



图片来源：<The Clipmap: A Virtual Mipmap>

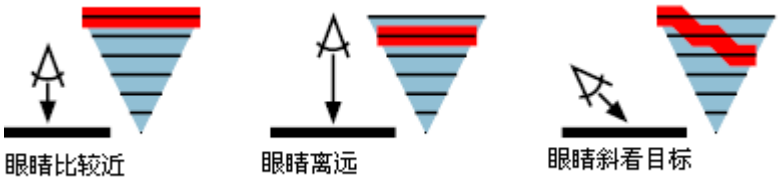
2-1，Mipmap 工作原理及其分析

我们先从 mipmap[4]工作原理分析开始。如图：



图片来源：<The Clipmap: A Virtual Mipmap>

Mipmap 的工作原理是，把一张贴图按照 2 的倍数进行缩小。直到 1X1。把缩小的图都储存起来。在渲染时，根据一个象素(pixel，注意 pixel 和 texel 区别。Pixel 是屏幕上一个点。Texel 是贴图上一个图素)离眼睛位置的距离，来判断从一个合适的图层中取出 texel 颜色赋给象素。D3D 和 OGL 都有想对应的 API 控制接口。如下图



图片来源：<The Clipmap: A Virtual Mipmap>

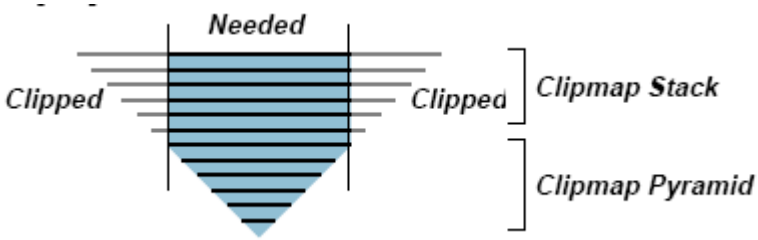
通过观察 mipmap 工作原理我们可以发现，硬件总是根据眼睛到目标的距离，来选取最适合当前屏幕象素分辨率的图层。假设有一张 32768x32768 mipmap 贴图。当前屏幕分辨率为 1024×1024。眼睛距离物体比较近时，mipmap 最大也只可能从 1024×1024 的 mipmap 图层选取 texel。再次，当使用三线性过滤(trilinear)时，最大也只访问到 2048x2048 的图层选取 texel，来和 1024x1024 图层中的图素进行线性插值。

这样一个基本事实告诉我们，在使用任何尺寸贴图渲染任意距离的物体时，贴图采样只会用到不大于屏幕分辨率 2 倍的 mipmap 层。根据这个事实，我们可以创建 clipmap 对大尺寸贴图进行渲染时的优化。

2-2，Clipmap 原理

2-2-1 Clipmap 构建

在原始 clipmap 论文中，clipmap 在显卡内存中的构建如图：



图片来源：<The Clipmap: A Virtual Mipmap>

Clipmap Pyramid（clipmap 棱椎)是我们 PC 显卡中常见的 mipmap 形式。 Clipmap Stack 是不同于 mipmap 实现的地方。横线绘制的棱锥代表完整贴图的 mipmap 形式。但是在显存中，大于 clipmap Pyramid 最高层尺寸的贴图，都以 clipmap pyramid 尺寸存放，但是相应的，clipmap stack 中的贴图代表的原始贴图面积，一层比一层小。比如 clipmap pyramid 最大 2048x2048。在之上一层，mipmap 表达是 4096x4094 图素。在 clipmap stack 中，还是储存 2048x2048 大小的贴图。显然此代表的就是 mipmap 尺寸 4096 的一半边长，即原始面积的 1/4 大小。依次推上去，每层都是下层代表面积的 1/4。Clipmap pyramid 最大边长即为 **Clipsize**。

Clipmap 贴图的中心点，在整张贴图的位置坐标，我们称为 **ClipCenter**。ClipCenter 由当前摄像机在整个世界中的相对位置来决定。

实际中，目前 D3D9 级别的 PC 显卡硬件并不支持这种储存形式的 mipmap。具体实现方法将在第三章讲解。

2-2-2 渲染 Clipmap

在渲染中，针对一个 Pixel，根据当前 Pixel，找到相应的 clipmap 层，如果处于 clipmap pyramid，直接按照 mipmap 传统方式采样取图素。如果处于 clipmap stack，则把贴图坐标根据当前 clipmap stack 代表整体面积的比例，进行缩放，得到对应位置图素。这样就可以正确渲染屏幕上的任意物体。如下图：

图片来源：<The Clipmap: A Virtual Mipmap>

把 CLIPMAP 所有图层绘制出来，从正上方看，就是象北京二环内，二环，三环，四环为分割。绘制中心区域的贴图分辨率最大，但是代表的面积最小。绘制最外围四环之外的贴图分辨率最小。是因为到了 clipmap pyramid 层，一层就代表一张全局贴图。但是分辨率已经被缩小了很多倍。而分辨率最大的 clipmap stack 顶层，由于尺寸必须保持 clipmap pyramid 的最大尺寸，所以所代表的面积就小了很多。这种折中正是由于基于绘制屏幕上的任何物体都不可能使用大于屏幕分辨率的贴图尺寸，所以在显存中只提供刚刚满足绘制在一针内，从一个视点看过去的所有场景（地形）分辨率的贴图内容。

2-2-3 climap LOD 计算以及每层贴图坐标计算

给定一个根据ClipCenter换算过的贴图坐标，下面需要计算出在屏幕上的一个物体的象素，从clipmap中选取一层，索引贴图图素。由mipmap LOD（LOD数在本文用俗称层数描述）计算公式得知，clipmap LOD计算和mipmap一样。并且可以保证任意pixel计算出来的climap层必然落在clipmap stack中。一般PC 3D显卡硬件计算都采用Heckbe算法。简单描述如下：

LOD = log2[f(x, y)]; f(x,y)为屏幕x,y轴方向贴图坐标变化率的最大值。贴图坐标变化率计算由屏幕象素X,Y坐标分别递进一个象素单位的变化投影到贴图坐标系得出。具体描述参考[5]。

2-3，存储效率

Clipmap由于采用clipmap stack和clipmap pyramid结合的方式，使用clipsize为2^m的clipmap，把一张2^n边长贴图容量从$(4^{n+1} - 1)/3$ 字节下降为$4^m(n - m + 4/3) - 1/3$ 字

节。更具体的数据如下：

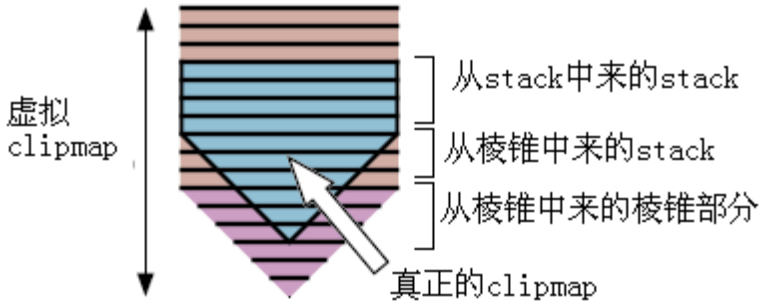
类型/边长	512	1024	4096	32768	67108864
Full Mipmap	682KB	2.7MB	42.7MB	2.7GB	10923TB
512 Clipmap	682KB	1.1MB	2.2MB	3.7MB	9.1MB
1024 Clipmap	682KB	2.7MB	6.7MB	12.7MB	34.7MB
2048 Clipmap	682KB	2.7MB	18.7MB	42.7MB	131.7MB

可见clipmap极大降低了显卡内存对渲染超大容量贴图的要求。目前D3D9的主流显卡从128M到256M不等。在1024的Clipsize下，几乎所有显卡都可以满足要求。

2-4，绘制整个地球表面的方案--虚拟 clipmap（简述）

当绘制国家或者星球表面这样规模的实时渲染时，整张贴图尺寸为 2^26 的平方。这样一来，需要在显卡硬件内部计算的工作，诸如贴图坐标换算等都难以在精度为 IEEE 32 FLOAT 下正确进行。

在原有 clipmap 基础上，在 U,V 方向加入一个坐标偏移向量。在 clipmap stack 的纵向方向，加入一个偏移数值。形成如下图解构:



图片来源：<The Clipmap: A Virtual Mipmap>

系统只要能够保证从图中的 3 种情况中正确寻址，就能够保证超大贴图的 clipmap 渲染 。

三，基于 D3D9 的简化实现

从之前对 Clipmap 的描述中可以看出。现代 D3D9 显卡具备了渲染 clipmap 的大部分功能。唯一欠缺的是 clipmap stack 的模拟。因为 clipmap pyramid 就是一张标准的 mipmap 贴图。所以 D3D9 只要能够把 clipmap stack 模拟出来，使用 shader model3 可以很容易的实现 clipmap。

3-1，Multitexture

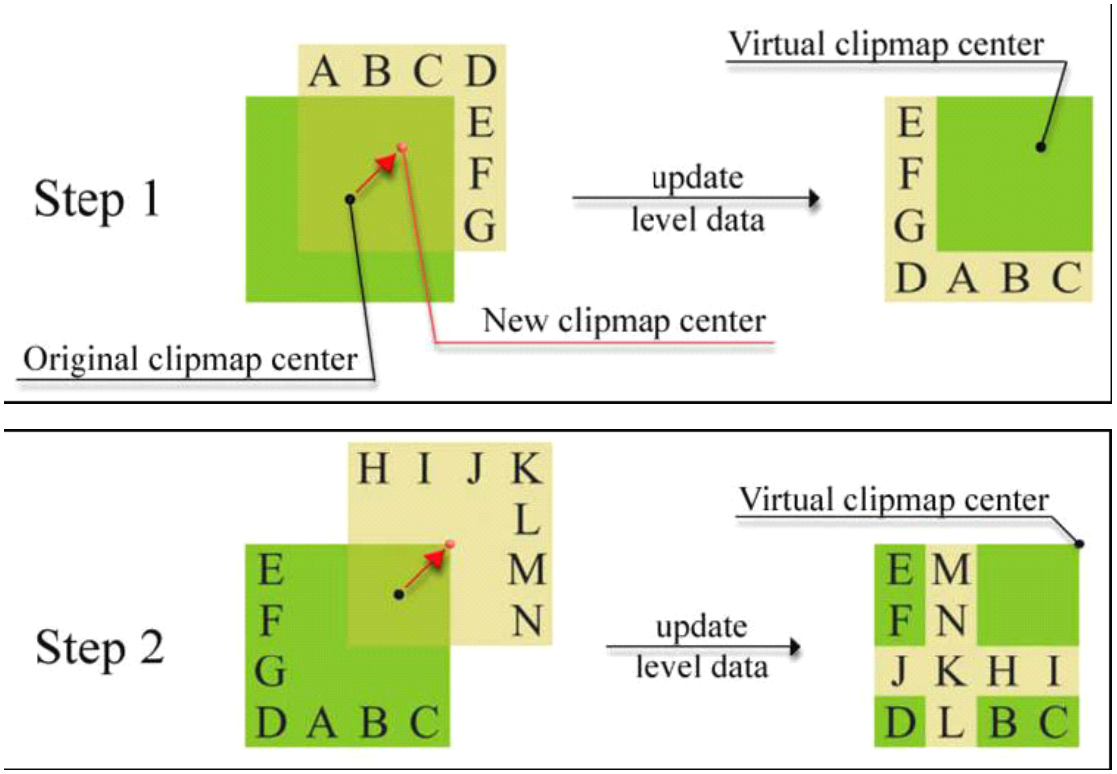
我们利用显卡硬件的 multitexture 模拟 clipstack。这是 D3D9 简化实现的最基本思想。PC 显卡支持 clip pyramid。在 pixel shader 中，如果想利用 clipmap stack，就必须有某种途径让 pixel shader 使用除了 clip pyramid 之外的贴图。自然我们想到了使用 multitexture。D3D9 标准显卡应该支持最少 8 层纹理单元（不支持的硬件很快会绝迹，所以可以不考虑）。假设

我们不使用法线贴图进行凹凸象素光照。

使用第 8 层装载 clip pyramid，使用 2048x2048 32bit DDS 贴图。从第一层到第七层我们都可以使用 clip stack 贴图。这样，分别装载 4096, 8192, 16384,32768,65536,131072,262144 边长 mipmap 的 clipmap。每一层占用 2048x2048*4*/4*1.33 /1024 /1024 = 5.32Mb 。这样一共 8 层贴图总共 43MB 贴图显存，可以容纳一张 274TB 的贴图。当我们使用每层 1024×1024 贴图时，使用 12MB 显存就可以模拟 68TB 的贴图。显然对于游戏来讲绰绰有余。注意这里只是简化计算，没有把 trilinear 考虑进去。

3-2，Toroidal addressing

在第四章我们将考虑优化和动态管理 clipmap。我们知道用 MULTITEXTURE 模拟的 clipmap 只是当前视角观察的贴图用量。当摄像机移动时，clipmap 的贴图组必须同时更新。除了最底层 clipmap pyramid 贴图不用更新，其他几张 clipmap stack 贴图必须根据摄像机移动，改变 ClipCenter 位置。然后从磁盘上把 clipcenter 周围的贴图调入。这涉及到显存写入问题。可以想见当所有 clipmap stack 贴图同时频繁装载总共十几 MB 容量的内存，对于磁盘 IO 和显存带宽都是极大考验。所以针对 clipmap 更新特点，在贴图寻址方面采用 wrap 循环寻址。然后对贴图的更新采用如下方式：



图片来源：NVSDK

3-3，pixel shader 实现思想概要

由于有了第二章 climpa 渲染的大概描述，以及前两节关于 clipmap stack 和贴图寻址方式，我们利用 shader model 3 的 pixel shader 功能可以直接实现具体算法。描述如下：

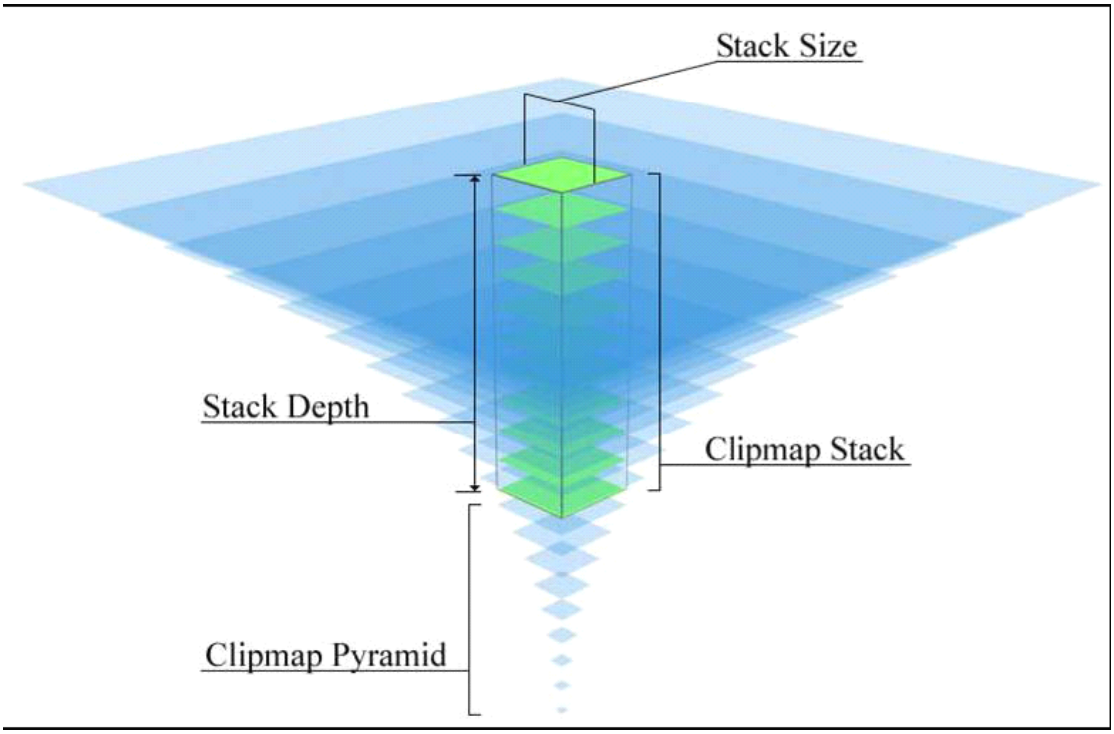
1，计算 mipmap LOD。
2，计算光照
3，假如应该采样 clipmap pyramid，直接采样，混合光照，完成
4，假如应该采样 clipmap stack 贴图，重新计算贴图坐标
5，根据 mipamapLOD 选择相应的 stack 贴图。采样，混合光照，完成
首先，计算 mipmap LOD。这里可以采用三线性插值。
例如：

```
PSOut PS_Trilinear(PSIn input)
{
    PSOut output;

   float2 dx = ddx( input.texCoord * g_TextureSize.x );
    float2 dy = ddy( input.texCoord * g_TextureSize.y );
    float d = max( sqrt( dot( dx.x, dx.x ) + dot( dx.y, dx.y ) ) , sqrt( dot( dy.x, dy.x ) + dot( dy.y, dy.y ) ) );

    // 计算出
    float mipLevel = log2( d );
    float blendGlobal = saturate(g_StackDepth - mipLevel);
    float4 color0 = PyramidTexture.Sample( samplerLinear, input.texCoord );

    If (blendGlobal)//如果使用了全局 clipmap pyrimad，及早退出
    {
        output.color = color0;
    }
    Else
    {//使用了 clipmap stack 贴图，计算贴图坐标，然后动态分支判断应该采样第几层贴图
        // This fractional part defines the factor used for blending
        // between two neighbour stack layers
        float blendLayers = modf(mipLevel, mipLevel);
        blendLayers = saturate(blendLayers);

        int nextMipLevel = mipLevel + 1;
        nextMipLevel = clamp( nextMipLevel, 0, g_StackDepth - 1 );
        mipLevel = clamp( mipLevel, 0, g_StackDepth - 1 );

        // Here we need to perform proper scaling for input texture coordinates.
        // For each layer we multiply input coordinates by g_ScaleFactor / pow( 2, layer ).
        // We add 0.5 to result, because our stack center with coordinates (0.5, 0.5)
        // starts from corner with coordinates (0, 0) of the original image.
        float2 clipTexCoord = input.texCoord / pow( 2, mipLevel );
        clipTexCoord *= g_ScaleFactor;
```

```
        //只是简单的双线性采样
         if (mipLevel==0)
        {
              //use clipStackTexture0.Sample(...)
        }
        If (mipLevel ==1)
        {
              //use clipStackTexture1.Sample(...)
        }
        }
Return output;
}
```

3-4，D3D10 的实现【7】

D3D10 引入了一种叫 Texture Array 的功能。可以在 pixel shader 中直接利用一个整数下标来寻找所需要的贴图，然后进行采样。思路同上，只不过不用动态分支，直接进行数组索引后采样。



图片来源：NVSDK
图中绿色部分就是 texture array。

四，一些优化和扩展

Clipmap 在显存中的存在，只是应对某一个固定摄像机位置和角度的场景贴图需求。当摄像机移动时，必须对 clipmap 贴图进行更新。具体更新方法第三章已经提过。但是，在摄像机每一针运动都使用磁盘 IO 和阻塞渲染管道的内存搬运，是不具实用性的。所以在实用中，必须对 clipmap 动态更新进行高度优化。这也是此技术的核心所在。

同时，对 clipmap 的一些扩展是为了增强游戏的表现力。以下 4-6 到 4-9 的扩展方面没有进行任何实际论文和实例支持，只是一种对现有技术扩充的预想和估计。希望专家给与更深入的建议。

4-1，动态 clipmap 管理

Clipmap 的动态管理，必须满足几个条件。1，摄像机移动时的平滑实时渲染；2，不能对摄像机的移动速度和位置有所限制；3，动态装载更新应该是外部不可见，自动进行的。

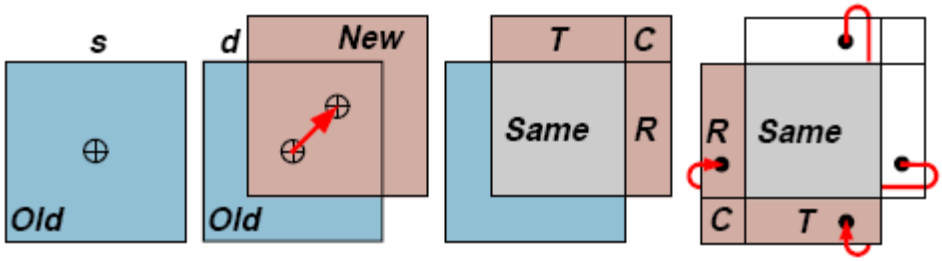
4-2，2 级 cache

由一般异步动态装载卸载应用技术得知，在大数据量进行从磁盘到内存的交换时，必须有足够的缓冲。由于磁盘到内存，内存到显存的带宽大不一样，磁盘到内存比内存到显存慢得比较多。所以，对于从硬盘到显卡内存的缓冲，需要存在至少 2 步 CACHE。

第一步，从磁盘到主内存的缓冲。对 clipmap stack 贴图周边的图素，必须预先读入一些到主存 CACHE 中来。并且这个动作应该在整个游戏期间保持不停顿。预先读入的数据根据玩家移动方向做一定预测。

在模拟星球表面的海量贴图库中进行更新，同时还要考虑不同磁盘阵列相应速度等等。所以磁盘到主存的缓冲策略非常重要。

第二步，从主存装入内存，每次更新每个 clipmap stack 贴图周边的半圈贴图内容。



图片来源：<The Clipmap: A Virtual Mipmap>

棕色区域是每次更新的内容。

4-3，多线程 IO 控制

在主循环中对贴图进行磁盘 IO，解压，然后写入 clipmap stack 贴图缓存，会同时阻碍 CPU 和 GPU 的工作流水线。使得处理器停滞，影响效率。所以，对磁盘到主存的读取 IO，解压工作，可以放在另外一个线程中进行。写入贴图显存的工作由于本身会阻断 D3D9 级别

显卡的流水线。但是，可以利用一些策略延缓一次性装入所有 CLIPMAP STACK 贴图的并发。这将在 4-6 讲解。

4－1 中提到的摄像机平滑移动渲染，主要是指对 clipmap stack 贴图更新的步骤一定要短，不能极大幅度影响主渲染循环的执行。在固定带宽内，对 clipmap stack 贴图边带更新的数据量并不很大。基本可以保持一个平滑渲染的过程。

在 WINDOWS 系统，使用无缓冲的异步 IO 模式是最适合此应用的。且文件以硬盘簇大小的整数倍放置，读取效率最快。

4-4，使用 MaxTextureLOD

在 4－1 中提到，摄像机以任意速度移动，都不能影响渲染的平滑。我们采用如下策略。

1，当摄像机以设计速度移动时，clipmap 正常更新。

2，当摄像机超过设计速度，clipmap 更新速度无法赶上摄像机移动速度。则自动调整 max texture LOD。Max texture LOD 的意义是 clipmap 工作在最高 clipmap stack 的贴图层数。这个参数同时影响 clipmap upload 和渲染 pixel shader 中 stack 选取。降低最高细节 stack 意味着降低 climap update 的负荷。

3 当出现极端情况时，所有 clip stack 更新都赶不上 camera 移动，则只使用 clipmap pyramid 贴图进行渲染。虽然图象贴图分辨率急剧下降，但是保证了 camera 以任意速度漫游 。

4-5，对地形几何渲染扩展--geometry clipmap

参考 GPU GEMS2 的 geometry clipmap 介绍文章【6】。简单描述为：使用 clipmap 进行顶点生成。把高度图做为 clipmap 进行更新，然后做为 vertex texture 的源。用以生成地形网格。这么做的好处是，地形网格削减被放入了 climap 的 LOD 过程中。而不用 CPU 做任何额外计算工作。并且支持相对无限大地形。动态地形改变。

4-6，double clipmap buffer，无效边带

4-3 指出，使用多线程把磁盘 IO 和解压缩，可以促进 CPU 和 GUP 协同工作。在对磁盘文件进行预读取时有着极大好处。在对 JPEG 进行分块读取和 REALTIME 解压缩时，可以在多核系统上获得额外的负载减轻。

Clipmap 原先的设计，存在一个叫无效边带的技术。它可以使得 SGI 硬件底层一边渲染一边进行边带的贴图更新。等于是主存到显存的第 0 级 CACHE。PC 和有的 CONSOLE 显卡不具备这样的功能。对于写入显存操作，都是把一张贴图显存 LOCK 住，进行写入。在这个期间此贴图无法进行渲染。但是受此启发，我们可以把 CACHE 的粒度放为每层 CLIPMAP STACK 贴图。结合多线程与 DOULBE BUFFER 思想，方法如下：

1，在需要创建 clipmap stack 贴图时，对每层贴图创建一个 BACK BUFFER。

2，在渲染 CLIPMAP 同时，多线程进行另外一组 stack texture 的更新。虽然可以预知假如写入一张贴图要阻断整个 GPU 工作管道的话，但是由于多线程把一组 stack texture 按顺序更新。所以避免了在主循环并发一次性更新整个 stack texture array。如果显卡可以同时渲染和写入 back buffer 的话，则效率应该会提高。

3，每针渲染完毕，调换 STACK BACK BUFFER 到前台。

这个方法 DOULBE 了 CLIPMAP 占用内存。但是在多核环境下，尤其显卡支持渲染时并发写入贴图显存功能的话，CLIPMAP 更新效率会大大提高。内存的使用，前面进行过计算，适合游戏的贴图，采用 1024 大小的 DDS CLIPMAP，显存占用不大。

4-7，支持非 2 的 N 次方全局贴图

在游戏中，一个全局的世界如果只能是 2 的 N 次方大小，会极大限制美术工作。如果是正方形世界，贴图只能是 4G，16G，64G（都乘 1.33)这个数量级别。考虑引入 MxN 大小的贴图。M 和 N 都不是 2 的幂。

我们在之上取任意一点 P，k<Px<M-k, k<Py<N-k， k=2^q(q=1,2,...),k 为接近分辨率的一个 2 的幂整数。围绕 P 建立一个边长为 k 的正方形。以 k 正方形为基础，创建 clipmap。这个 clipmap 代表在 P 点绘制需要的贴图内容。

随着摄像机移动，原先不用更新的 clipmap pyramid 也需要根据 ClipCenter 移动进行更新。保持贴图棱锥还是以 2 的幂进行更新。换句话说，除了 clipmap stack 贴图更新，clipmap pyramid 贴图以及其 mipmap 层，都需要更新。

这样一来，我们就可以用 clipmap 进行任意长宽世界贴图的渲染。

4-8，支持法线贴图进行象素级别光照

在 ID TECH5 演示中，地表都具有凹凸贴图象素光照。在这里无法给出确切的实现方法。只是尝试列出可能的实现方案。希望更进一步的探讨。

法线贴图同样使用一张全局贴图。

1， multipass 方法。使用第二张 clipmap 装载法线贴图。

2， 在更新 diffuse color clipmap 时，利用 3 张 clipmap stack 贴图的 ALPHA 通道，装载法线贴图的 z,y,z。或者只装载 x,y（在 pixel shader 计算 z)。然后在 pixel shader 重组，进行光照计算。2 层 stack clipmap 代表一层 normalmap。我们只在近距离使用 normalmap，远处就用一个 RGB(0.5,0.5,1)的蓝色象素代替。在超出 clipmap stack 视野区域使用 mipmap 的 trilinear 插值。

3， 使用 clipmap 其中的一层或者两层装载法线贴图，因为远距离地形景物不需要象素级别的光照。所以直接对近景进行象素光照即可。凹凸象素光照消失的地方使用一些三线性插值，平滑过渡。

4-9 非地形应用

Clipmap 主要用于大规模地形渲染的贴图调度。在此基础上，对非地形场景的渲染，应用 clipmap 也称为可能。Clipmap 实际是对贴图数据库进行了一种线性组织和查找。可以根据简单的 x,y 坐标以及 u.v 贴图坐标定位所需要贴图在 clipmap texture database 的位置。地形应用的天然方便之处在于，地形网格的空间位置本身就说明了其贴图纹理在 clipmap texture database 中的位置。

根据此思想，如果能够在场景中固定几何物体信息中标致好所在 clipmap 数据库中的位置。并且有效组织场景中的几何体贴图存放位置也和其世界空间位置相关。这样就能在几何体与 clipmap 之间建立起一种联系。随着摄像机的移动，clipmap 贴图不断更新，相应场景

中的物体就被赋上材质。当视角远离物体，此物体贴图不再被使用，从显存中清除。这样就实现了非地形的海量几何物体贴图的 clipmap 管理。

此思想只是停留在思考阶段。需要在今后的工作中实践。

五，是否值得

5-1，地形渲染精度和容量

我们来计算一下实际游戏大概需要的贴图容量。

假设地形，高度图的一个象素代表一个格子。一个格子代表现实一米。屏幕分辨率 r=1024x1024。人眼离地面最低距离 d=1 米。FOV=90。离地面最近距离为人眼直视下方。根据透视关系，可以推算出在屏幕上 2 米的格子将会撑满整个屏幕。

一张非 TILING 的贴图，平铺在地下，我们可以计算出，如果需要屏幕的一个象素恰好被一个贴图图素覆盖(mipmapLOD=0)。则需要 2x2 的格子贴一张 1024 非 TILING 的贴图。

对于一个格子贴 512 象素才不会被 filter 进行插值模糊。这个贴图细节度是非常惊人的。按这个计算，要一张全局 RGB24BIT 贴图贴满 1 平方公里的区域，就需要 1024*1024*512*512*3*1.33 字节＝ 1Tb 的 RGB 贴图。粗略估计 JPEG 大概为 40－50G 之间。但是 1T 的海量贴图相当于一张 1024x1024 图的 26 万倍，这基本不太可能由美术手工完成。而且，一公里的距离根本不足以表现大型世界。假设人站在 512，512 的坐标，500 米的直线距离，景物远远没有达到人眼视界之外。人眼在晴天看远处山脉可以有十几公里的视界。就算在远处不用精细的贴图，但是人既然能够移动过去，那么必然要制作这么大范围的全局贴图。利用 512 象素代表一米显然是不现实的。

如果我们认为不需要那么高的精度。观察 WOW，粗略估计可以接受的精度为，128-256 象素代表一米(WOW 基本为 256)。我们计算人眼离地 1 米时一个图素覆盖 4 个象素插值的情况：128*128*1024*1024*3*1.33 = 65G. JPEG 估计 3G。4 平方公里就是 48G。

对于付出如此巨大代价，在一平方公里之内，让美术进行了相当于一万六千多张 1024x1024 地图的手绘，实时渲染进行非常复杂的 CACHE 和多线程异步操作，得到的效果只是 4 个象素模糊一个图素的图形细节。而且如果游戏想承载 4 平方公里的地域，基本就已经超出目前 CONSOLE 和 PC 能够接受一个游戏容量的范围。即便是蓝光，也快到达了极限。而 4 平方公里对次世代游戏来说，简直是九牛一毛。有很多游戏（CRYSIS）对于表现几十平方公里都已经非常得心应手。数量也在几个 G 之内。所以单纯使用 clipmap 实现连续大型地形渲染，以目前硬件容量，还是比较吃力。需要其他优化手段配合。

接下来计算贴图更新速度需求。假设人类行走 1.5 米/秒.如果按照斜前方行走。1 米 128 图素。 Clipmap 为 1024。Clipmap stack 为 6 层。在一秒人类行走经过的地域，以 1024 为边界，正方形边带图素个数大约为 1.5/sqrt(2) *128* 1024 * 2 ＝ 2172Kb。1 秒钟需要更新的图素内存为 6x2172x3 = 39096Kb 。大约 40MB。假设 30Hz 刷新，一针里更新的数据为 39096/30 = 1.3Mb。一针一 MB 的数据流量，也是非常客观。

5-2，对传统地形渲染方法进行 STREAMING

5-1 节对使用全局贴图进行大范围渲染的数量和最小贴图传输量做了估计。接下来我们

看一下，应用 clipmap 思想，传统地形渲染方法是否有可扩展的空间。

在 1-2-1 和 1-2-2 使用 tiling 的两种地形渲染方法中，都提到要应用至少 1－2 张全局地表纹理贴图。在使用全局贴图绘制过程中，完全可以借鉴 clipmap 的技术。把全局 alpha 和 lightmap，或者全局 diffuse 颜色图放入 clipmap 管理。由于其他图层都是靠 tiling 实现。理论上可以实现贴图密度无限制，贴图大小无限制的全面解决方案。而制作全局 diffuse 颜色图和制作 clipmap 全局图一样。Alpha map 靠手工绘制。Lightmap 可以自动生成。并不会比只使用一张全局地形贴图麻烦多少。

六，结论

Clipmap 极大的释放了显存限制，把应用对贴图细节的无限增长需求转化为对 clipmap 实时更新的技术解决方案。此应用不仅可以用于大规模地形渲染，也可以应用与大型场景中固定几何物体的贴图管理。

Clipmap 可以做为一种空间和贴图数据库的对应管理系统。并且在象素级别进行 LOD。结合 geometry clipmap 的应用，可以相对提高虚拟现实制作的生产力。

同时 Clipmap 的更新成为了一个新的瓶颈。在 PC 硬件架构上，需要极大精力进行更新的优化 。

另外，在对次世代游戏质量和渲染精度的标准下，对 clipmap 采用全局世界纹理的可行性做了分析，并提出了质疑。同时结合传统地形渲染方法，给出了与 clipmap 相结合的可能性。

来信请给 puzzy4d@yahoo.com.cn，欢迎讨论。

索引

【1】WORLD OF WOLFCRAFT,http://www.wowchina.com, Blizzard,2004

【2】Far Cry, http://www.crytek.com/, Crytek,2007

【3】<The Clipmap: A Virtual Mipmap>,Christopher C. Tanner, Christopher J. Migdal, and Michael T. Jones,Silicon Graphics Computer Systems,1998

【4】EWINS Jp，WALLER MD，WHITE M，et a1．MIP—Map Level Selection for Texture Mapping[J]．IEEE TRANSACTIONS ON VISUAIAZATION AND COMPUTE R GRAPHICS，1998，4(4)

【5】Mipmap 映射技术中 d 值计算方法的探讨,韩慧健 ，徐振中 ，张诚洁,计算机应用,第 24 卷第 12 期,2004

【 6 】 Terrain Rendering Using GPU-Based Geometry Clipmaps,<GPU GEMS 2>, Hoope,NVIDIA,2005

【7】NVIDIA, SDK10,clipmaps