DeepL 订阅 DeepL Pro 以编辑此演示文
访问 www.DeepL.com/Pro，了解更多信息。

第五章

寒霜中的地形渲染
使用程序性着色器
溅射

约翰-安德森 [1]

DICE

5.1 简介

许多现代游戏都发生在户外环境中。虽然对几何 LOD 解决方案有很多研究，但在实时应用中使用的纹理和阴影解决方案通常是相当基本和不灵活的，这往往导致近处、远处或高角度的细节缺乏。

地形纹理的一个比较常见的方法是，将低频细节的低分辨率唯一颜色图（图 1）与高频细节的多个平铺细节图结合起来，在与相机的一定距离内进行混合。这给艺术家们提供了对低频的良好控制，因为他们可以随心所欲地绘制或生成颜色地图。

[1]电子邮件：johan.andersson@dice.se

对于细节贴图，有多种方法可以使用。在《*战地 2*》中，一块 256 平方米的地形可以有多达六个不同的平铺细节图，这些细节图通过一个或两个三分量的独特细节遮罩纹理混合在一起（图 4），控制各个细节图的可见性。艺术家可以像绘制彩色地图一样绘制或生成细节遮罩。

图1.地形颜色图来自战地2

图2.战地》的俯瞰图。坏公司的景观

图3.战地》的特写镜头。坏公司的景观

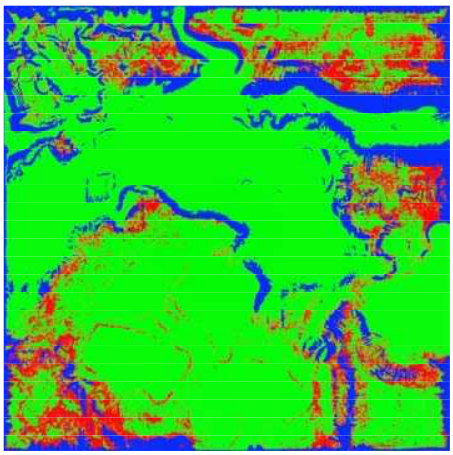
所有这些传统的地形纹理和渲染方法在未来有几个潜在的问题，我们想在开发 *Frostbite* 引擎时尝试解决或改进这些问题。

图4.战地2》中的地形细节遮罩纹理。RGB通道强度代表3个独立的平铺细节纹理的可见性。

纹理需要大量的内存，是一个固定的功能和内存成本。在地形

上的不同区域和材料上改变着色和合成可能是很困难的，而且计算量也很大。

我们的主要问题是，它们是静态的。从《*战地 1942*》开始，我们就希望能够在几何和纹理上破坏地形，但还没有足够的性能或内存来支持高度场的任意几何破坏。通过动态改变压缩的颜色贴图和细节遮罩纹理来扩展破坏的纹理合成方法也不是真正可行的。为被破坏的材料（如开裂的柏油路面或烧焦的草）添加更多的同步细节贴图变化也不可行。



在我们希望能够破坏地形的同时，我们也希望在减少内存使用量的同时，从总体上提高地形的视觉质量。传统的地形纹理合成方案，如
战地 2》独特的彩色地图和细节遮罩

但是，为不同的材料进行不同的和专门的着色和纹理是一件非常好的事情，而且通常是游戏中普通网格的着色器的方式，以尽可能地提高内存和计算效率。

例如：如果我们想在岩石表面使用视差遮蔽贴图（[Tatarchuk06]），我们不想为所有其他不需要的材质支付计算视差遮蔽贴图的性能成本。同样的，如果我们有一个覆盖了大部分关卡的海底材质，但阴影和纹理的质量并不重要，因为它将被海面部分遮挡住。在这种情况下，我们希望不必支付存储颜色贴图和细节遮罩纹理的费用，因为该材料可以用一些着色的细节贴图来近似。

将地形着色器专门用于不同的地形材料开辟了许多有趣的可能性，在这一章中，我们描述了基于这一想法的地形渲染系统和技术，用于 DICE 的 Frostbite 引擎，在《*战地*》中使用。*坏公司》中使用*。

5.2 地形纹理和阴影

Frostbite 中地形纹理的基本思想是允许艺术家为单个地形材料创建具有任意动态纹理合成的专门着色器，并根据材料的性质使用最适合的方法将它们分布在地形上。

5.2.1 基于图形的表面着色器

为了让艺术家们能够轻松地　进行实验　，并　为单个地形材料创建自定义着色器，我们利用内部的高级　着色框架，允许通过图形表示而不是代码来编写表面着色器（图 5）。更多细节见[AT07]。

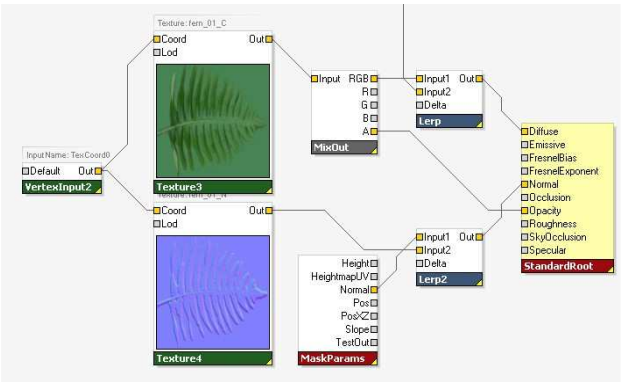


图5.基于图形的表面着色器的例子

这种基于图形的方法有多种好处，包括编写地形着色器。

- 对艺术家友好。我们的艺术家中很少有人知道 HLSL，通过标准的对话框来调整数值和颜色，而不是写文字，这是一个很大的生产力提升。
- 灵活性。程序员和艺术家都可以轻松地暴露和封装新的节点和阴影功能。
- 以数据为中心。它很容易自动处理或转换着色图中的数据，这可能是非常强大的，基于代码的着色器很难做到。

着色框架通过一个复杂但强大的离线处理管道生成结果着色方案和实际的像素和顶点着色器，以便在游戏中使用。该框架根据高级别渲染状态组合生成着色器。系统有许多可用的状态，如光源的数量和类型、几何处理方法、效果和表面着色器。

管道生成的着色解决方案被游戏运行时间所使用，该运行时间通过以下方式在多个平台上实现

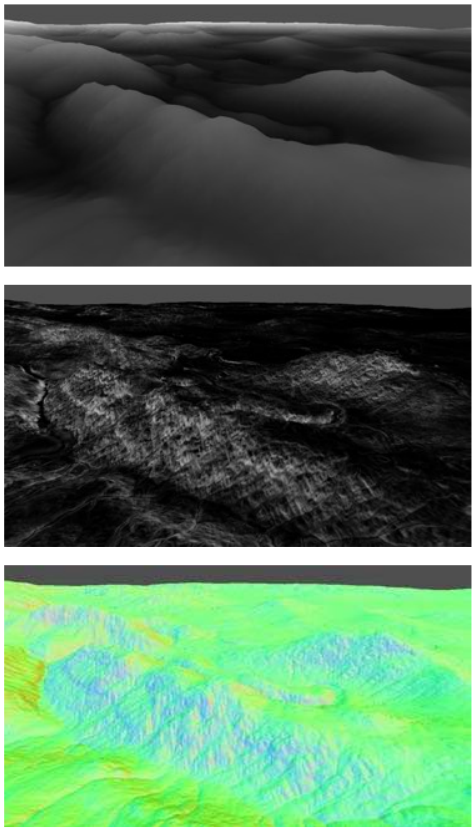


图6*.地形每像素参数：高度（上）、坡度（中）和法线（下）。*

DirectX9、Xbox 360、PlayStation 3 和 Direct3D10 的渲染后端。它处理向 GPU 调度命令的工作，并可以通过遵循包含确切设置渲染的说明的着色方案来相当薄。

在实现基于图形的着色器开发的同时，我们意识到需要在我们的框架中支持灵活而强大的基于代码的着色器块开发。通常情况下，艺术家和程序员都可能想利用自定义的复杂功能，并在整个着色器网络中重复使用它们。作为这个问题的解决方案，我们引入了*实例着色器*--具有明确的输入和输出的着色器图，可以在其他着色器内实例化和重复使用。通过这个概念，我们可以分层封装部分着色器，并创建非常复杂的着色器图形网络，同时仍然是可管理和高效的。这种功能使得着色器网络可以很容易地被扩展。

大部分的地形着色和纹理功能是通过实例着色器实现的。管道中的一般数据转换和优化能力（主要是在图层面上）被用来结合多种技术来创建长的单通道着色器。

5.2.2 程序性参数

在过去十年中，消费类 GPU 的计算能力一直超过摩尔定律，图形芯片每一代都变得越来越快。同时，内存大小和带宽的增加并不匹配 GPU 计算能力的跳跃。意识到这一趋势，尝

试在着色器中计算大部分的地形着色和纹理合成，而不是将其全部存储在纹理中，是有意义的。

有许多有趣的程序性技术用于地形纹理和生成，但大多数都需要多遍渲染到缓存的纹理中以实现实时使用，或者是昂贵且难以正确的 mipmap（如 GPU Wang Tiles 中的[Wei 03]).

出于性能方面的考虑，我们选择从一个非常简单的概念开始，即计算并向基于图形的地形着色器暴露三个程序参数（图 6）。

- 高度（米
- 坡度（0.0=0 度，1.0=90°）。
- 正常（世界空间

由于地形是基于高度场的，所以对地形上的每一个像素的参数计算都很简单和快速。

高度是一个归一化的 16 位高场纹理的双线性样本，然后按地形的最大高度进行缩放。

正态可以用多种方式计算，我们发现一个简单的四样本交叉过滤对我们来说足够好了（*清单 1*）。
斜率是 1 减去法线的 *Y* 分量。

```
sampler bilinearSampler。
Texture2D 高度图。

float3 filterNormal(float2 uv, float texelSize, float texelAspect)
{ float4 h;
        h[0]  =  heightmap.Sample(bilinearSampler,  uv  +  texelSize*float2(  0,-1)).r  *
texelAspect; h[1] = heightmap.Sample(bilinearSampler, uv + texelSize*float2( -1, 0))。r *
texelAspect; h[2] = heightmap.Sample(bilinearSampler, uv + texelSize*float2( 1, 0)).r *
texelAspect; h[3] = heightmap.Sample(bilinearSampler, uv + texelSize*float2( 0, 1)).r *
texelAspect;
     float3 n;
    n.z = h[0] - h[3]。
    n.x = h[1] - h[2]。
    n.y=2。
    return normalize(n); }
```

清单1*.高度图法线交叉过滤着色器（Direct3D 10 HLSL）。*

5.2.3 屏蔽

地形着色器决定了一个材料的外观，而且，如果需要的话，还决定了它出现的位置。

例如，假设我们有一个山体材质着色器，我们想让它在地形的斜坡上可见。这可以通过两种方法来实现。一种方法是使用灰度蒙版纹理，可以在地形上手动绘制（或在其他程序中

生成），完全控制材质出现的位置。注意，我们必须为这个遮罩的纹理付出内存成本的代价（因为所有的纹理合成都是在运行时完成的）。

我们支持的另一种方法是让着色器本身计算它应该出现的位置。在这种情况下，对于一座山来说，可以用着色器中的程序性坡度参数来计算一个简单的斜坡函数，在指定的最小和最大坡度之间掩盖山体，并有一个线性过渡（图 7 和 8）。这种方法也是许多离线地形渲染和生成程序的基础，如[Terragen*]。

从着色器中的程序斜率计算出来的遮罩的分辨率受到高度场的分辨率的限制。因此，在极端的特写镜头下，由于高度场的双线性纹理放大，遮罩会变得模糊不清。这可能会在材料之间产生沉闷和典型的不自然的平滑过渡。在使用低分辨率的基于图像的绘画蒙版时也会出现同样的问题。

我们可以通过在每个材料的基础上向计算的蒙版添加细节来改善平淡的过渡。我们可以在必要时在每个材料的基础上通过混合平铺的细节掩模或程序性噪声（如分数布朗运动）来为计算的掩模增加细节。
在像素着色器中计算噪声可以产生高质量，并且在现代 GPU 上可以达到相当快的速度（[Tatarchuk 07]），但是对于我们想要计算多种材料的多个八度空间的目的来说，计算上仍然是很困难的。

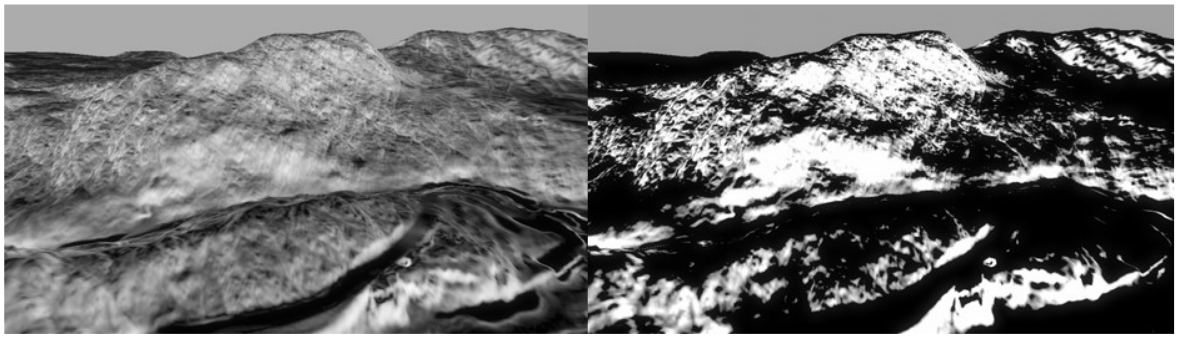


图7.地形坡度参数（左）。在着色器中计算的山体遮罩（右）。

图8.没有(左)和有(右)山地材料的地形，使用坡度参数的计算掩码

fBm 也可以在着色器中 "计算"，方法是预先生成一个特定时期的离线噪声纹理，并对每个倍频程进行采样，而不是用算术方法计算噪声函数。这没有那么灵活，而且限制了范围，但可以更快，而且仍然有用。

在我们的案例中，我们用一种更有效、更简单的方法来增加大部分材料的蒙版细节。 我们编写或（在着色器中）重用平铺的灰度纹理作为细节掩膜纹理，并将它们与具有各种功能的低分辨率掩膜结合起来（图 9）。这样做的好处是需要很少的纹理取数（与基于纹理的 fBm 方法相比），而且在 ALU 操作的复杂性上也很灵活（与基于 ALU 的噪声相比），因此是一个很好的折中办法。它还通过创建细节遮罩纹理和选择如何组合遮罩，让艺术家对细节转换有很好的控制。

Adobe® Photoshop®混合模式*叠加*（清单 2）对于结合两个蒙版纹理和增加细节非常有用。它不影响基础程序蒙版为 0.0 或 1.0 的区域，因此蒙版的基本形状得以保留。我们几乎只使用它与简单的乘法和线性混合在一起，但当然也可以使用任何混合模式。

图9.带有程序性坡度遮罩的地形特写（顶部）。程序性遮罩与平铺的细节遮罩纹理混合，使用叠加混合模式
(中间)。平铺的细节遮罩纹理（底部[45])

将多个细节遮罩纹理与所有其他纹理放在一起，会很快吃掉性能和纹理采样器（Shader Model 3 中只有 16 个）。为了改进这一点，我们在重用普通颜色纹理的通道，甚至是已经在着色器中使用的法线贴图方面取得了一些很好的效果，可能是对纹理坐标进行了不同的平铺，重新映射范围或改变对比度（*清单 2*），以获得一个归一化的遮罩值并使用它来代替额外的纹理。

```
float overlayBlend(float value1, float value2, float opacity)
{
        float blend = value1 < 0.5 ? 2*value1*value2 : 1 - 2* (1-value1) * (1-value2) ;
return lerp(value1, blend, opacity);
}
float scaleContrast(float value, float contrast)
{
    return saturate((value-
0.5)*contrast+0.5); }
```

***清单2**.叠加混合和对比HLSL函数。在归一化的[0,*
1]范围内工作，可以很容易地扩展到任意维度（例如颜色）。

5.2.4 静态稀疏掩码纹理

有许多地形材料不能以纯粹的程序方式生成，特别是在只使用基本参数时，如高度、坡度和法线。图 1 中远处的空地就是一个很好的例子，它们是人工创造的，关卡设计师和艺术家希望完全控制其形状和位置。

为了方便这一点，我们支持在我们的编辑器工具中或在 Photoshop 中手动为单个地形材料在地形上绘制任意的灰度掩码。

为了节省内存，所有绘制的蒙皮纹理都存储在一个稀疏的四叉树纹理表示中，只存储唯一的 32×32 像素的瓷砖。这可以说是一个很大的胜利，因为通常没有一个地形材料遮罩能覆盖整个地形（图 10），而这些空白区域就不会占用任何内存[1]。四树表示法还允许降低蒙皮纹理中那些总是从远处看的区域的分辨率。

为了获得最佳的纹理资源利用率和性能，四个四树掩码纹理被打包到一个 64 位指示纹理的 R、G、B 和 A 通道中，一个 32 位四树级纹理和一个 DXT5A/BC4 图集纹理（图 11）。

指向纹理存储了一个归一化的 XZ 索引，用于地图集中的瓦片。

[1] 并非完全正确，指示性纹理仍然占用内存

四叉树级别的纹理存储了瓦片在四叉树中的哪个级别，在从世界空间位置计算纹理坐标时使用。

在*清单 3* 中，包含了 4x 稀疏四树掩码纹理采样着色器。

***图10**.叶子地形材料的源面具纹理*

***图11**.*
2048x2048灰度掩码纹理图集，32x32瓦片

在创建面具纹理图集并从中取样时，必须注意垫高瓷砖的边界，以防止在使用双线性过滤时在边缘出现过滤伪影。否则，图集中的部分边界瓦片会在边缘漏掉，导致瓦片的边界出现难看的线条伪影。

在 Direct3D10 和 Xbox 360 上，可以使用纹理阵列来代替图集，然后双线性过滤会自动正常工作，不需要任何额外的填充。不幸的是，纹理阵列在 Direct3D10 中的限制是 512 片（瓦片），在 Xbox 360 上是 64 片，这限制了它们在这种情况下的作用，除非瓦片被分割成多个纹理阵列。

```
sampler pointSampler;
sampler bilinearSampler;
Texture2D levelsTexture;
Texture2D indicesTexture。
Texture2D atlasTexture。

空白的 QuadTreeMasks(
        in float2 posXZ, in float2 heightmapUV, in
float2 atlasSize, in float2 invAtlasSize,
        in float
maskIndirectionResolution, out float4
outMasks)
{
        float4 indices = indicesTexture.Sample(pointSampler, posXZ);
float4 levels = levelsTexture.Sample(pointSampler, posXZ) 。

    levels *= 255.0f; //解压 [0,1] -> [0,255] 8 位
    [开卷]
        for (int i=0; i<4; i++)
    {
                float2 uv =
frac(heightmapUV*maskIndirectionResolution/levels[i]); uv *= invAtlasSize;

        float2 索引。
        index.x = floor(fmod(indices[i], atlasSize.x));
index.y = floor(indices[i] * invAtlasSize.x)。

        uv += index*invAtlasSize。

        outMasks[i] = atlasTexture.Sample(bilinearSampler, uv)。
    }
}
```

***清单3**.四树纹理采样着色器（Direct3D 10 HLSL），输出是4个单独的遮罩值。注意：瓦片之间的填充不包括在内。*

5.2.5 销毁面具

当地形的某个区域受到地面破坏的影响时，高度场中该区域周围的像素会被更新。我们希望，在同一时间，能够改变纹理合成，使该特定区域的其他地形材料可见，以显示例如烧焦的泥土（图 12）。

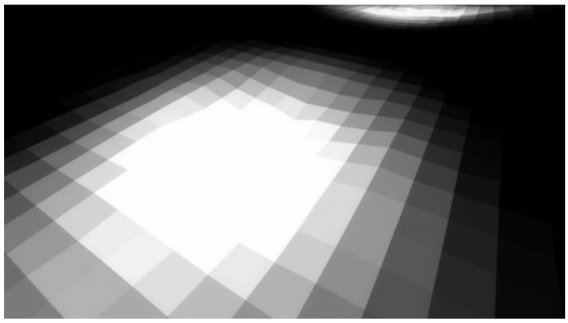


图12*.地面破坏掩蔽* ***图13****.破坏性掩膜（道路上的烧焦的泥土材料有指向性为清晰起见进行过滤)*

我们通过动态渲染纹理掩码贴纸来实现这一点，该掩码贴纸覆盖了整个可以被破坏的地形（图 13）。破坏遮罩的分辨率很低，每米 4 像素，因为遮罩只需要在受地面破坏影响的区域包含粗糙的圆形渐变。更多的细节可以在着色器中添加，其方式与我们在本章前面描述的程序化蒙版和彩绘蒙版纹理添加细节的方式类似。

尽管如此，即使是合理的低分辨率，纹理的内存占用也成为一个问题。在一个 2048×2048 米的可破坏地形区域，每米 4 个像素的未压缩 8 位掩码纹理需要$(2048*4)^2$ 字节=64MB。这在任何平台上都是不可取的。

在我们的案例中，地面破坏的最坏情况其实并不是 100%的地形区域可以被完全破坏，并且需要同时进行掩蔽。我们可以摆脱的百分比要低得多，也许是 10%。但是我们并不想限制在地形的什么地方发生破坏，所以这 10%的破坏面积可以任意地分散在整个地形上。

这与我们之前遇到的静态稀疏掩码纹理的情况类似，不过在这种情况下是动态纹理。因此，我们选择的节省内存的方法是在 GPU 上为地面破坏创建并逐步更新一个动态稀疏掩码纹理（图 14）。

图14*.动态稀疏遮蔽纹理图集渲染目标（左）。动态掩膜嵌套纹理在地形上的顶部投影（右），B和G通道是归一化的XY索引，进入图集中的瓦片。地形上的三个独立区域受到了地面破坏的影响。*

我们不需要改变破坏掩码的分辨率，与静态掩码纹理相比，所以我们可以使用一个固定的网格结构来处理嵌套纹理，没有四树级别的纹理，使得纹理采样着色器更快。

当一个地面破坏事件被触发并且高度场被更新时，我们检查弹坑覆盖的区域是否被分配到稀疏纹理中。如果没有，我们就在图集中分配一个或多个新瓦片，并将图集瓦片的 XY 索引存储在 CPU 复制的间接纹理中。然后，该定向纹理被复制到 GPU 上。

每个弹坑都被表示为一个小型的 2D 纹理映射贴花，通过设置视口以匹配贴花，然后渲染该贴花内的所有贴花，将其渲染到破坏掩码纹理图集贴花中。由于每一帧分配或更新的瓦片非常少，但陨石坑和分配的瓦片总量可能很高，这种增量更新可以说是一个很大的胜利。

5.3 地形着色器合成

地形上的任何区域都可以有多个重叠的地形材料，需要一起合成。这些材料被指定为一个严格的顺序，决定了哪种材料位于哪种材料的上面。

渲染材质的一个简单实现是在帧缓冲器中使用 alpha 混合法（如[Bloom00]）对地形材质进行合成。这样的实现方式将按照从后到前的顺序浏览每个地形材质，渲染与该材质相关的所有地形几何，并将其输出混合到前一个材质的输出之上。

然而，这样的多通道方法也有不少缺点。

- 帧缓冲器带宽。地形覆盖了屏幕的大部分，我们添加的材料越多，每个像素必须被读出和写回帧缓冲区的次数就越多，这就消耗了内存带宽。*图14.2种地形材料（红色和*
- 几何图形的过度绘制。与任何*绿色）创建3个地形材料的多*通道技术一样，*由于重叠（黄色）的几何体组合*被多次渲染。由于我们要渲染的地形有大量的三角形的良好细节和 GPU 顶点吞吐量的增长速度不如像素吞吐量，这可能成为一个大瓶颈。
- 重复的着色器计算。地形材料着色器中的许多计算，如地形法线，将在每一次传递中重新计算，这是很昂贵的。
- 固定功能的混合模式。内置的混合模式不是很灵活，特别是与着色器相比。有很多有趣的方法来非线性地组合地形的自然纹理

相反，我们将所有的地形材料着色器自动组合成大的单通着色器，并在着色器中进行合成。这可以通过在材质之间共享着色器计算来实现最佳性能，同时只渲染一次几何体和像素。

为了实现合成，我们有一个预处理步骤，分析地形并收集所有在地形的每一块上使用的地形材料组合。这个过程考虑到了地形材料的分布掩码，当多种材料在同一个补丁上重叠时（图 14），所有的材料都将被包括在内。

然后，这些信息被用来为找到的每一个地形材料组合创建复合着色器，并保存一个参考复合着色器的网格，以便运行时知道在地形上的哪个补丁和区域使用哪个着色器。

考虑到着色器的图形表示，自动创建复合着色器是出乎意料的简单。地形材质着色器的输出被重新路由到预先创建的合成着色器的输入，该着色器结合所有的材质并输出最终的颜色。

重复的资源（纹理、采样器和常量）和复合着色器中相同的图形子树或代码会被一般的着色器图形编译器自动删除。

图15*.使用程序化着色器拼接，将约15个地形材质着色器遮蔽并组合起来的地形俯视图。*

为了进一步提高性能，在复合着色器中严格使用了动态流控制，以跳过材料完全被其他材料覆盖的区域的计算和纹理获取。这在所有平台上都是一个很大的胜利。

我们把这种地形纹理和着色的方法称为*程序性着色器拼接（procedural shader splatting）*，从这个意义上说，我们是将程序性着色器任意地 "拼接 "在地形的不同区域和相互之间，然后将它们全部组合起来，以实现高效渲染（图 15）。

5.4 地形渲染

地形剔除和 LOD 是通过一个帧与帧之间的连贯四叉树结构完成的，每个节点都知道节点内高度场区域的最大和最小高度。当高度场因地面破坏而改变时，节点的最小高度可能会改变。

四叉树中所有可见的叶子节点都被渲染成固定的 33 x 33 顶点网格。顶点网格被存储在一个共享的顶点缓冲区中，网格顶点只包含一个 4 字节的 UV 坐标，该坐标在网格上给出一个[0,1]的参数化。这个参数化在顶点着色器中被转换为高度场和世界空间，并用于通过高度场纹理获取顶点的地形高度。因为顶点网格与高度场对齐，所以可以使用点过滤，这对不支持顶点着色器中纹理的双线性过滤的 GPU（GeForce 6 和 7）来说是个好处。

在不能有效支持顶点纹理获取的平台和显卡上，我们有一个 33×33 顶点的顶点缓冲池，它以 LRU 为基础按需分配给可见的四叉树节点，并由 CPU/SPU 线程通过采样高度场填充。

固定的顶点网格分辨率是很重要的，它能够以固定的成本和质量支持最坏情况下的任意地面破坏。这 "浪费 "了非改变的平坦区域的三角形，但我们发现这个成本是值得的，因为这个方法的简单性和通用性。

5.4.1 几何学 LOD

如图 16 所示，在使用固定网格分辨率的四维树叶时，不同 LOD 的相邻四维树斑会在更详细的斑块中产生 Tjunctions。

如果地形高度场在 t-junction 附近变化，在详细补丁（高 LOD）中 t-junction 的三角形在高度场中的采样高度将与低 LOD 中它们旁边的三角形不一样。这就在地形上产生了孔洞，这些孔洞可能很明显，特别是如果天空等明亮的颜色被渲染到地形下面。为了消除地形上的潜在孔洞，我们需要消除 T 字路口。
我们可以这样做，首先要求所有的四边形树节点与它的邻居最多有 1 级的差异。然后用一个单一的三角形取代构成详细补丁中每一个 T 形结点的两个三角形，这个三角形只使用邻近的低 LOD 中也有的顶点（图 17）。这些顶点通过我们要求的最大级别差异保证存在。

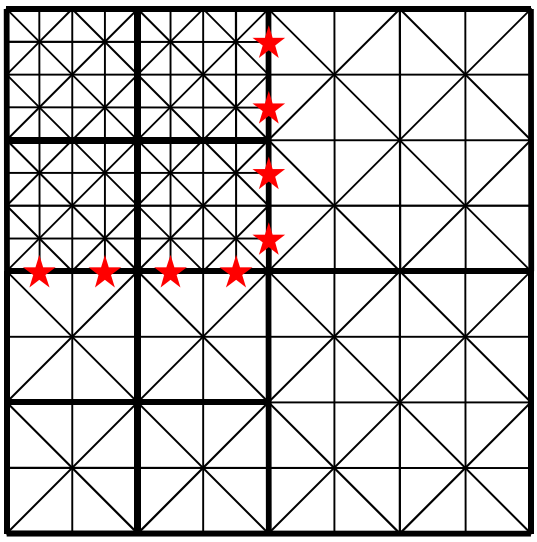


图16 *.有四种树的斑块*

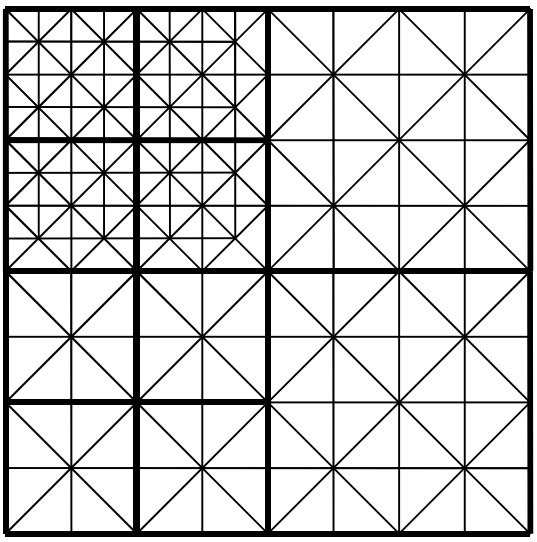


图17 *.三角形创造T-*

不同的LOD创建T型交叉点，并 在红星 的地形上的最高LOD孔中移除交叉点 补丁

为了支持具有最大级别差异限制的四叉树节点的所有可能组合，我们所需要的是 9 种不同的索引缓冲器排列组合（图 18）。

- 一种是顶点网格中的所有三角形都完好无损的包络，当邻居补丁是同一级别时，就会使用这种包络。这是最常见的情况。
- 在补丁的四条边上去掉 T 字形三角形的四种排列方式

- 移除两边相邻的 T 字形三角形的四种排列组合

我们不需要所有可能的 16 种排列组合（所有边上的 T 型交叉点单独移除或保留）的原因是我们选择从详细的补丁中移除几何体，而不是在较低的 LOD 补丁中添加几何体（这也是可行的）。四叉树叶子节点的两边总是共享父节点，因此也共享 LOD，这意味着我们不需要从一个补丁的两个以上相邻的边上移除 t-junctions。

这种带有多个索引缓冲区的技术有点类似于[Dallaire06]，但使用的是四叉树而不是相同大小的补丁。

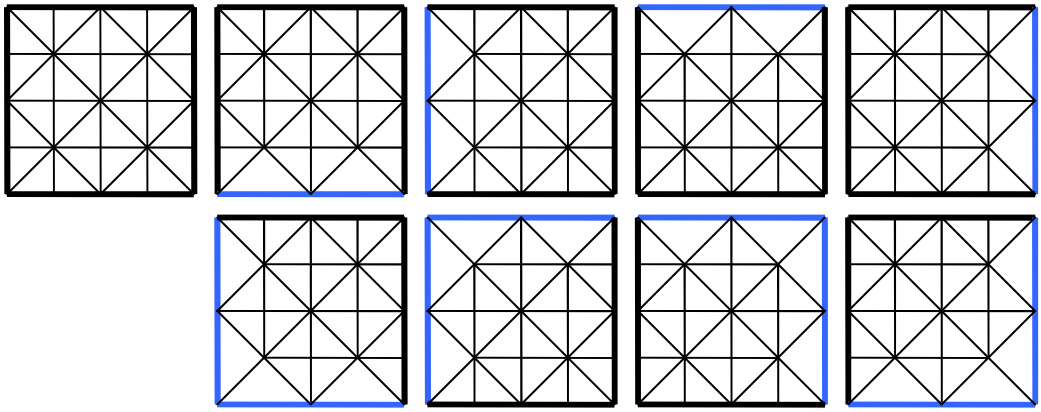


图18.无T结的LOD转换所需的9种几何排列方式

5.5 矮树丛

无论我们在地形上有多么先进的着色、纹理和光照，由于高度场作为几何体的固有限制，它在近处看起来仍然不自然（图 19）。

我们想要的是细节几何和网格，用灌木、草、植物、石头和碎片来填充地形，创造一个更丰富的环境（图 20）。由于高度场的几何形状和地形的材料和纹理会因为地面的破坏而发生变化，这种细节几何形状也需要能够被更新。

图19.无景观灌木丛

图20.有矮树丛的景观

无论是从时间管理还是数据管理的角度来看，在整个地形上手动放置单个石头或植物都不是一个可行的方法。但在实践中，关卡设计者甚至不需要或不喜欢这种控制量。

灌木丛的几何尺寸也相当小，最大约 1 米，而我们希望它非常密集，每平方米最多有几个实例。这使得在一个大型的 2×2 或 4×4 公里的地形上，仅仅存储和加载实例数据（转换）就成了问题。

自动程序化生成的放置（程序化实例化）可以解决内容工作流程和内存存储问题。

实例数据的程序化生成既可以作为离线的预处理，也可以作为运行时的按需步骤。我们选择后者，因为它对内存和磁盘存储的要求明显较低，而且可以使我们轻松地动态再生受地面破坏影响的区域。

5.5.1 方法

在以前的游戏中，如*《RalliSport Challenge 2》和《Battlefield 2》*，我们根据单独的材料索引 CPU 纹理在地形上的顶层投影，程序性地生成了灌木实例数据。这些地图索引了艺术家定义的矮树丛材料，其中包含分布设置，如分布的网格、密度（每平方米的数量）、随机比例范围、动画设置等。

该系统运行良好，但材料指数图有三个主要的限制，我们希望在《寒霜》中解决。

- *树下的材料不能重叠*。用不同类型的矮树丛涂抹一个区域是很麻烦的，而且有局限性，因为你需要克隆原来的材料，并在材料中添加新类型的几何图形来分布。
- *树下的材料与底层的地形材料是完全分开的*。如果一个地区的地形纹理被重绘成泥土而不是草，那么灌木丛材料索引图也必须被手动重绘。
- *分辨率和破坏*。有了动态地面破坏，我们需要有一个更高的分辨率，这些纹理的成本内存。

由于我们现在通过着色器对地形材料和纹理进行贴图、着色和分配，所以使用同样的程序性着色器溅射系统来生成矮树丛是很自然的。然后，所有已经存在的地形材料可以自动地在它们的特定区域分布矮树，而只需在内容上做最少的工作。

地面破坏和重叠材料也已经是一般地形材料遮蔽的一部分，所以它是非常合适的。

5.5.2 生成

由于矮树丛的小规模和高密度，我们只在内存中生成并保留靠近玩家（和其他重要视口）的区域。这是通过一个基本的网格结构来实现的，其中 16 x 16 米的矮树丛网格单元是在景观周围移动时从一个固定的单元池中动态分配和取消分配的。

为了防止用游戏板或（更糟糕的）鼠标快速旋转视图时的性能下降；单元的分配是以视口原点的 2d *xz* 距离为基础进行的，而不是当单元在视口果壳中可见时。单元视口地壳检查仍然可以用来区分哪些单元需要尽快生成，哪些*应该*生成以进一步平衡多帧的生成成本。

每个单元包含一个区域内的灌木网状结构类型列表和一个包含所有实例数据的顶点缓冲器。实例数据通常只是一个 4 x 3 的世界变换，被压缩为 fp16 值，以节省内存并提高 GPU 性能。

当一个单元变得可见或受到地面破坏的影响时，我们用 MRT 渲染出 4-12 个材料遮罩值以及地形法线，并在单元区域上自上而下地投影到 2-4 个 ARGB8888 64x64 同步渲染目标（图 21）。使用的着色器是以类似于地形着色器合成着色器的方式自动离线生成的。

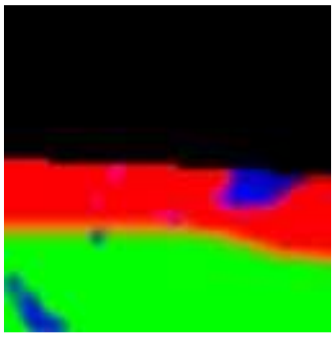


*图21。4通道生长不良掩膜纹理在黑色没有生长不足的*单元空间上（左）。*底层生长来自地形的单元法线图（右）。*

当纹理被 GPU 渲染后，我们将其锁定（或者在 Direct3D10 中，将其复制到一个暂存纹理），以便由 CPU 线程或 SPU 进行处理。

CPU 处理以随机抖动的网格模式扫描每个树下材料的纹理，网格大小取决于材料密度设置。在每个采样点，材料掩码纹理被读取，并进行俄罗斯轮盘赌，以确定是否应该在该点放置一个灌木丛实例。

如果它通过了，地形法线图就会被用来旋转或倾斜实例以适应地面。

随机抖动的网格模式的工作原理是在网格上生成统一的点，并以最大半格的长度随机地偏移这些点，给出一个统一但不同的分布。与普通的伪随机分布相比，这减少了实例的重叠，这在视觉上和草地等材料的性能上都很重要。

为了在一个单元内生成伪随机数时得到确定的结果，单元在网格结构中的位置被散列并作为种子使用。这在本地客户端重新生成单元时很重要，但在网络的多个客户端运行时也很重要，这样每个人都能看到相同的几何图形。

在 Direct3D10 中，整个生成步骤可以使用流输出（[Blythe06]）转移到 GPU 上，以卸载 CPU 并减少生成的延迟。

5.5.3 渲染

生长环境下的细胞生成后，渲染它们很容易。

树下网格是具有任意表面着色器的低聚网格（图 22），使用流实例化进行渲染。它们使用阿尔法测试或阿尔法覆盖，并在每个单元的基础上按从前到后的顺序渲染，以改善分层 Z-cull，尽管纹理中大量的小细节使得分层 Z-cull 不是非常有效。

通过使用表面着色器框架和运行时间，矮树丛将获得与引擎中任何其他表面相同的每像素照明和阴影，这看起来不错，使其更容易与环境的其他部分融合。

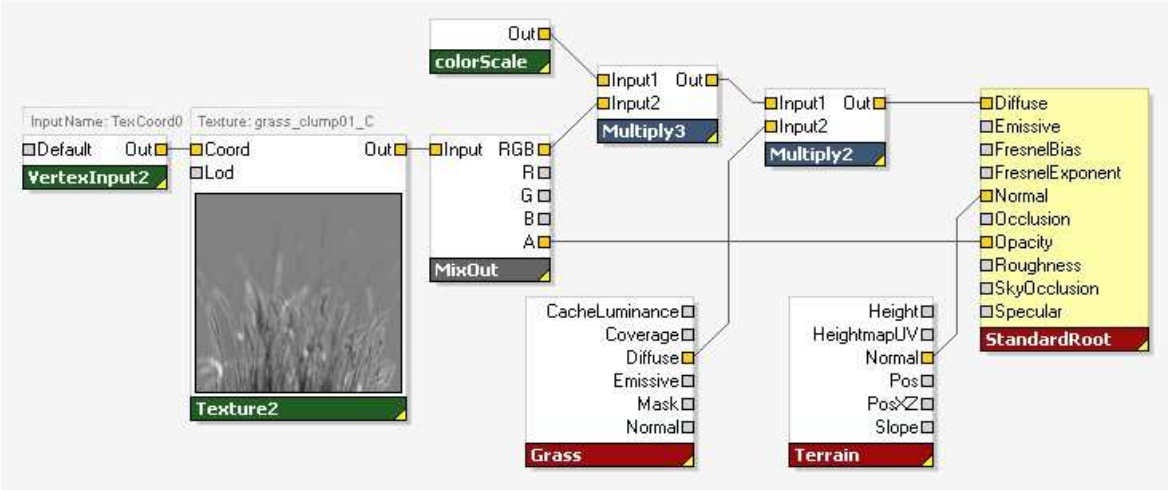


图22. 草地网格的树丛表面着色器。通过将其颜色贴图与实际地形草丛着色器的漫反射颜色进行合成，与地形融为一体。照明使用高度场的法线，看起来和地形一样。

5.6 结论

我们提出了一种灵活的地形渲染框架和技术，称为*程序化着色器 Splatting*，其中基于图形的表面着色器控制地形纹理的合成和分布，以允许地形材料被单独专用，以平衡性能、内存、视觉质量和工作流程。

该技术使我们能够支持对地面破坏的动态高度场修改，同时在远处和近处都保持较高的视觉质量和较低的内存使用。

树丛的程序化实例被整合到系统中，使用地形材料分布和着色器是一个非常强大的工具，也是增加视觉细节的简单方法，在内存和内容创建方面成本都很低。

然而，这种技术也有一些固有的缺点。

- *性能*。由于几乎所有的纹理合成都是在运行时在着色器中完成的，而不是存储在离线创建的颜色地图中，所以这种方法一般来说比传统的固定方案（如*《战地 2》*的成本要高（由于着色器指令数和纹理获取数）。

- *复杂的工作流程*。虽然艺术家仍然可以选择绘制遮罩纹理和颜色地图，但要真正利用该系统，他们需要将其与程序性着色结合起来，而程序性着色是不熟悉的，而且不像纹理那样有固定的成本。另一方面，程序元素可以更容易地在多个地形上共享和重复使用。

该技术和框架所具有的灵活性使其成为一个伟大的可扩展平台，以便在未来整合有趣的着色技术和纹理方案。

5.7 参考文献

[At07] Andersson, J., tatarchuk, N. 2007.Frostbite 渲染架构和实时程序性着色和纹理技术。AMD 赞助的会议。GDC 2007。2007 年 3 月 5 日至 9 日，加利福尼亚州旧金山。http://ati.amd.com/developer/gdc/2007/Andersson-Tatarchuk-FrostbiteRenderingArchitecture（GDC07_AMD_Session）.pdf

[BLOOM00] BLOOM, C. 2000.通过在帧-缓冲区中的混合进行地形纹理合成（又称 "溅射 "纹理）。2000 年 11 月 2 日。网站。http://www.cbloom.com/3d/techdocs/splatting.txt

[blythe06] Blythe, D. 2006.Direct3D 10 系统。在 ACM 图形交易会议记录中（SIGGRAPH'06 会议记录），第 724-234 页，波士顿，马萨诸塞州。

[dallaire06] Dallaire, C. 2006.用于生成地形瓦片索引缓冲区的二元三角树。Gamasutra 文章。http://www.gamasutra.com/features/20061221/dallaire_01.shtml

[tatarchuk06] tatarchuk, N. 2006.带有近似软阴影的动态视差闭塞映射。In proceedings of AMD SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp.63-69, Redwood City, CA.

[tatarchuk07] tatarchuk, N. 2007.噪声的重要性：快速、高质量的噪声。会议会议。GDC 2007。2007 年 3 月 5 日至 9 日，加利福尼亚州旧金山。http://ati.amd.com/developer/gdc/2007/Tatarchuk-Noise(GDC07-D3D_Day).pdf

[TERRAGEN*] TERRAGEN by Planetside Software. http://www.planetside.co.uk/terragen/

[WEi04] WEI, L. 2004.图形硬件上基于瓷砖的纹理映射。在 ACM *SIGGRAPH/Eurographics 图形硬件会议*论文集中，第 55-63 页。法国格勒诺布尔。