#### 一种手游中通过预烘焙植被信息提升植被渲染效率的方法

#### 一、 缩略语和关键术语定义

#### 二、背景技术

##### 1. 相关技术背景

游戏中的植被（包括树木和灌木等）一般通过插片的形式来实现，通常具有很高的面数。另外，由于其拓扑几何结构的复杂性，渲染植被时，需要配合开启alpha test。alpha test会影响GPU的preZ流程，降低渲染速度。 在移动平台上，alpha test会带来极大的渲染开销，受限于移动平台的性能，功耗和发热等问题，需要严格控制场景中植被的面数。

##### 2. 与本发明相关的现有技术

1. billboard技术。即公告板技术。对于远处的植被，取其一个方向拍一张简单的照片，将其贴到游戏中的一块公告板上，通过代码控制其不断旋转，以使公告板总是面向场景的摄像机。此种方案开销相对来说比较小，但是因为信息有限，烘焙时没有做处理，会有噪点多，黑边，lod切换移位等问题。同时烘焙出的billboard没有法线信息和自阴影信息，无法参与光照计算， 所以显示效果很差。此外，由于只有一个方向上的信息，对于对称性不佳的植被模型，会有穿帮的现象出现。在移动平台上，由于面数的限制，要求billboard显示距离相对于PC平台小很多，所以还是需要保证基本的效果，能够参与光照计算。
2. impostor技术。与billboard技术相比，取64个方向烘焙植被的信息，存储了64个方向上的信息。但是由于烘焙的方向和信息过多，需要存储的图片数量也变的更多。在移动平台上，加载过多的图片会带来极大的内存开销，低端机器可能会出现内存不足的情况，同时会导致很大的带宽开销。由于上述限制，由于烘焙出来的图片数量过多，导致每张图片不能过大，显示距离稍微拉近时，显示模糊粗糙，效果很差。
3. 上述两种技术，都不能进行实时阴影计算，所以没有自阴影信息，导致渲染出来的植被过于平整，跟有阴影计算时差距大。虽然可以用AO(ambient occlusion)来替代阴影，但AO需要美术来制作，导致增加制作成本的增加。

#### 三、本发明技术方案的概述

##### 本发明索要解决的技术问题

* 本发明提供了一种新型的billboard烘焙方式，只烘焙植被八个方向上的信息，同时保证烘焙出来的billboard也能参与光照阴影计算，保证显示效果。在移动平台上可以近距离用billboard替代植被模型，显著降低植被面数，提升性能，降低功耗。

##### 本发明核心方案内容

* 为解决传统billboard方案的问题，本文提供了一种新型的billboard信息烘焙方案，包括：
* (1) 除了Base贴图外，烘焙记录模型的法线信息，以参与光照计算。
* (2) 为了有效解决噪点问题，采用超采样技术，烘焙一张尺寸比较大的图，然后手动生成对应目标尺寸的结果图。
* (3) 通过引入后处理流程，加入扩边逻辑，以解决现实效果里面的黑边问题。
* (4) 只烘焙8个方向上的信息，相比于上述的impostor技术，减少了生成图的数量，相对于简单的 billboard技术，能够有效避免穿帮问题。
* (5) 为了解决billboard没法进行阴影计算的问题，烘焙时记录遮挡信息，利用球谐函数投影，存储遮挡阴影信息，运行时可以从球谐函数中还原出阴影信息，进行光照计算，保证显示效果。

##### 本发明技术方案带来的有益效果

* (1) 性能上的提升。
* 通过烘焙植被模型八个方向上的信息，用作植被模型的最后一级lod。由于用来显示烘焙信息的billboard的mesh（即网格）只有四个顶点两个三角面，所以用来替代模型时，可以极大的减少了场景中植被的面数，提升性能。
* (2)效果上的提升。
* 加入了法线和阴影的烘焙逻辑。所以即使billboard模型只有两个三角面，也能够参与光照计算。球谐函数编码的自阴影信息用于还原阴影，没有渲染阴影的开销，可以得到近似阴影渲染的效果。同时烘焙了八个方向上的信息，运行时根据当前视角选择最合适的方向，能够有效的避免lod切换时穿帮问题。保证显示效果后，可以适当的拉近billboard的显示距离，进一步降低移动平台上的开销。
* (3)效率上的提升。
* 烘焙过程不需要美术的参与，无需美术单独制作其他的额外的信息图，阴影等信息通过光追的方案计算出来，编码在球谐函数中，无需美术额外制作AO等信息图。

#### 四、本发明技术方案的具体实施例

1. 图一是本发明提供的billboard烘焙流程图。具体流程说明如下：

S101: 计算自遮挡阴影信息。模型加载完毕后，对模型上的每一个顶点，以顶点位置为起点，在球形空间中发射多条射线，判断射线是否相交，记录顶点在各个方向上的遮挡情况。计算得到各个方向上的遮挡信息后，利用球谐函数投影，将遮挡信息用二阶球谐函数编码，编码后的遮挡信息球谐系数存放在网格的顶点信息当中，烘焙时提取阴影球谐函数，存储到烘焙图里，后续用于自阴影的光照计算。

S102: 计算摄像机烘焙时的位置和朝向。首先计算待烘焙模型的包围球信息，取包围球上平行于地面的球面，将该球面八等分，八等分后的八个位置分别代表烘焙时相机需要放置的位置。 为了避免移位问题，在进行烘焙时，相机朝向的并不是包围球的中心的方向，而是模型的中心方向。只有这样，烘焙出来信息会存储在图的正中央的位置，从而可以避免从植被模型从正常模型billboard模型时出现位移问题。

S103: 初始化烘焙渲染管线，初始化渲染管线中渲染对象的大小。 然后加载需要烘焙的植被模型，包括模型的网格信息，贴图信息，球谐函数信息等。

S104：进行烘焙。首先将Base贴图信息烘焙到第一张图上。然后还原法线信息，将Tangent space中的法线还原到模型的局部空间(local space)中，为了运行时效率，法线贴图中存储的是局部空间中的法线信息。然后取前面步骤预计算的网格顶点上的遮挡信息编码的球谐系数，将球谐系数进行插值后烘焙到第三张图上。

S105: 噪点处理。为了减少烘焙生成图的噪点信息，设置渲染管线时，通常设置烘焙出来的图的尺寸是最后用于渲染的图的尺寸两倍或者4倍。得到烘焙出来的图后，运用超采样技术，将烘焙出来的大尺寸的图缩小到用于渲染时的图的大小，可以有效的去除图中的噪点信息。

S106: 扩边处理。由于渲染管线的mipmap机制，如果不对噪点处理后的烘焙结果图进行扩边处理，在绘制时模型边缘会出现黑边。扩边时，先拿到base图里面的alpha通道的信息。如果烘焙出来的base图里面的某个像素的alpha通道的值为0，则寻找距离该像素最近的像素，取最近像素的RGB三个通道的颜色值作为该像素的颜色值。对于烘焙出来的法线图和球谐信息图，也是根据对应位置上base图里面的alpha通道的信息，进行扩边处理。Base图不会进行alpha通道的扩边，只进行RGB三个通道的扩边处理。而法线图和球谐信息图则会对生成图的四个通道都进行有效的扩充。

S107: 移动到下一位置，继续进行烘焙流程，直到烘焙完成，输出烘焙结果图。

S108: billboard网格顶点信息生成。billboard的网格顶点信息需要根据烘焙的植被模型大小信息来生成。为了提升效率，在保证显示效果的前提下，要尽可能的缩小生成的billboard网格信息。生成的billboard网格越小，进行alpha test的像素的数量越少，效率越高。由于烘焙时摄像机的设置是按照模型包围球的大小来设定的，为了避免billboard显示时的异常拉伸，billboard网格顶点信息的生成需要根据模型包围球来计算。billboard模型网格的顶点位置根据其包围盒和包围球的比例来计算。billboard网格顶点位置计算伪代码如下：

boxExtent = boxMax - boxMin;  
float xzExtent = std::max(boxExtent.x, boxExtent.z);  
xzExtent /= 2.0f;  
  
float bias = 0.0f;  
if (xzExtent < mCameraDistance)  
{  
 bias = mCameraDistance - xzExtent;  
}  
  
boundingSphereBoxMax = vec3(boundingSphereCenter.x + boundingSphereRadius, boundingSphereCenter.y + boundingSphereRadius, boundingSphereCenter.z + boundingSphereRadius);  
boundingSphereBoxMin = vec3(boundingSphereCenter.x - boundingSphereRadius, boundingSphereCenter.y - boundingSphereRadius, boundingSphereCenter.z - boundingSphereRadius);  
  
vertexPositions = {  
 // front panel  
 { boundingSphereBoxMin.x + bias, boxMin.y, 0.0f }, // low left  
 { boundingSphereBoxMax.x - bias, boxMin.y, 0.0f }, // low right  
 { boundingSphereBoxMin.x + bias, boxMax.y, 0.0f }, // up left  
 { boundingSphereBoxMax.x - bias, boxMax.y, 0.0f }, // up right  
};

S109: billboard网格uv信息生成。 uv的计算需要根据模型包围盒和包围球的相关信息确定。 根据billboard模型的长度d与包围球半径的r的比值以及模型包围盒高度h和模型包围球半径r的比值，计算billboard模型四个顶点的uv取值。具体计算伪代码如下：

boxExtent = boxMax - boxMin;  
float diameter = 2.f \* boundingSphereRadius;  
  
// for front panel  
float xzExtent = std::max(boxExtent.x, boxExtent.z) / 2.0f;  
float xzUVBias = (1.0f - (xzExtent / boundingSphereRadius)) / 2.0f;  
xzUVBias = std::max(0.0f, xzUVBias);  
  
float upYUVBias = std::max(0.0f, ((boundingSphereRadius + boundingSphereCenter.y) - boxMax.y) / diameter);  
float downYUVBias = std::max(0.0f, (boxMin.y - (boundingSphereCenter.y - boundingSphereRadius)) / diameter);  
uvs = {  
 { xzUVBias, 1.0f - downYUVBias}, // lower left  
 { 1.0f - xzUVBias, 1.0f - downYUVBias },// lower right  
 { xzUVBias, upYUVBias }, // upper left  
 { 1.0f - xzUVBias, upYUVBias } // upper right  
};

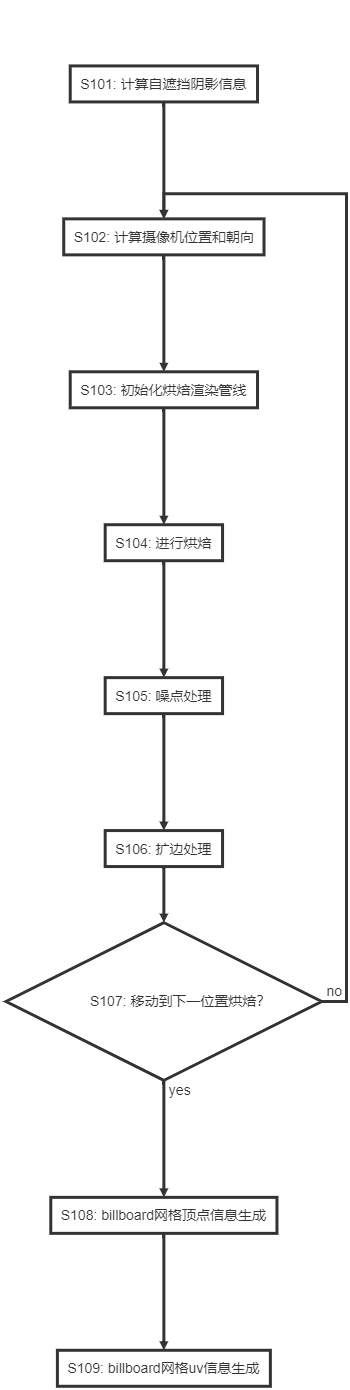
烘焙过程记录了八个方向上模型的Base，法线和球谐函数编码的自遮挡阴影信息。运行根据模型朝向和摄像机朝向的夹角，选定一个方向的烘焙图进行显示。具体billboard在游戏中显示时计算过程如下图2所示。渲染流程具体说明如下：

S201: 计算时，使billboard总是朝向摄像机方向。首先计算出模型到摄像机的方向向量。根据模型Y轴和计算出来的朝向，计算X轴，然后计算模型位置信息。具体计算伪代码如下所示：

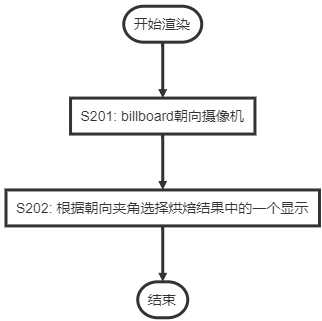
// 计算billboard模型的x,y,z三个朝向轴  
zAxis = CameraPos.xyz - modelPos.xyz;  
zAxis.y = 0.0; //忽略y分量  
yAxis = modelRotYAxis;  
xAxis = cross(yAxis, zAxis);  
  
worldPos = inPos.x \* xAxis + inPos.y \* yAxis + inPos.z \* zAxis;  
worldPos \*= modelScale;

S202: 根据当前朝向和摄像机朝向，计算夹角。根据夹角，从烘焙的八个方向中选择一个烘焙结果进行显示，具体计算伪代码如下所示：

// 计算模型到摄像机的朝向：
  
viewDirection = CameraPos.xyz - worldPosition.xyz;
  
// 忽略y分量
  
viewDirection.y = 0.0;
  
  
// 只取xz方向
  
vec2 modelForward = normalize(ve2(modelForward.z, modelForward.x));
  
vec2 camForward = normalize(vec2(viewDirection.z, viewDirection.x));
  
  
// 计算夹角
  
cosTheta = dot(modelForward, camForward);
  
  
// 计算 [0, 360]范围内的夹角
  
vec2 zVec\_t = vec2(zVec.y, -zVec.x);
  
pAngle = dot(fVec, zVect);
  
theta = acos(cosTheta);
  
if(pAngle < 0.0)
  
 theta = 2 \* PI - theta;
  
theta += PI / 8.0;
  
// 计算渲染图的index
  
Index = static\_cast<int>(std::floor(theta \* (8.0f / PI) \* 0.5f + 0.00001)) % 8;



#### 图1：烘焙流程图



#### 图2：渲染流程图