选题：Voxel Cone Tracing

成员：李博文 陈福康

使用方法：

实现 UI 操作，包含鼠标和键盘控制，具体操作方法见 README 和 demo

概述：

全局光照是真实图像合成的一个重要元素，但其计算代价昂贵，高度依赖于场景的复杂性和所涉及表面的 BRDF。2012 年的 GTC 上提出了一种实时计算间接照明的新方法，该方法基于场景几何和反射亮度的离散体素表示。在该方法提出了一种新的体锥跟踪算法，利用稀疏体素，有效地收集间接光照和评估间接可见性。

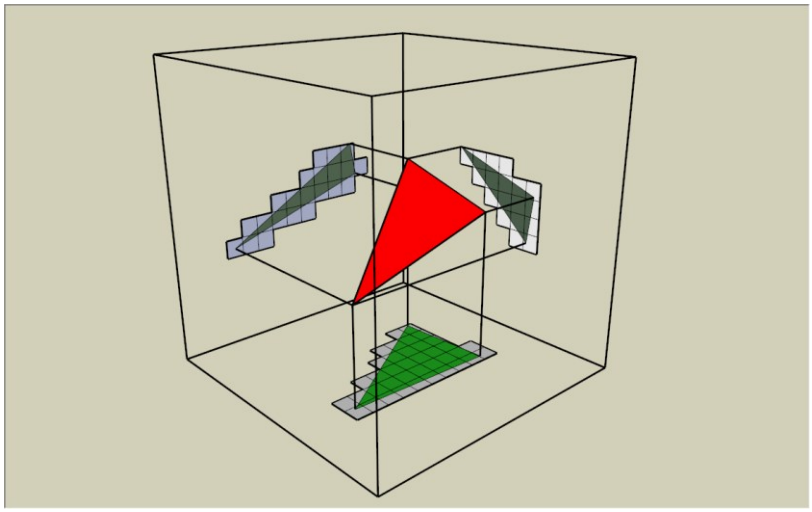
主要任务：

1.体素化

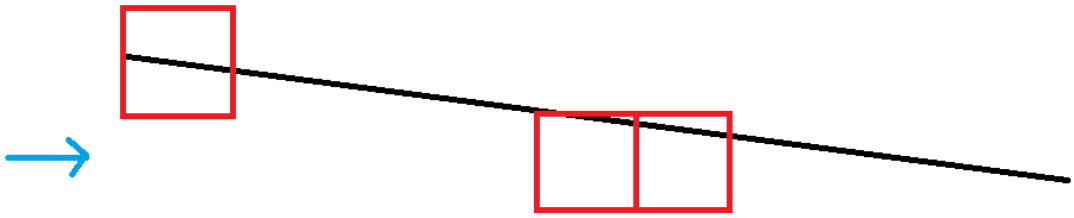
2.光线计算

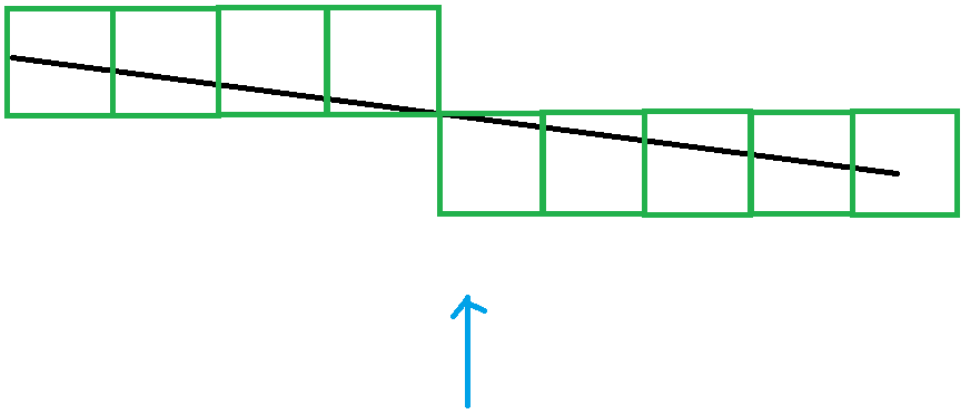
一、体素化：

我们的体素化是基于 GPU 计算的：首先计算出需要体素化模型的包围盒，然后将模型投影到包围盒的某个面上，经过渲染管线的光栅化插值操作，我得到了模型的每个片元及相应的坐标，根据这个顶点坐标标记三维空间数组的相应位置（就是投影的过程），最后把相关信息存到一个 3D 纹理里面。可以看到，整个流程思路非常清晰，但是为了消除缝隙或者孔洞，我们还要做一些处理。



在下图中，一个从左边投影，一个从下面投影，我们可以清晰地看见不恰当的投影方式会产生相当多的缝隙或者孔洞：





所以我们选择投影面积最大的方向进行体素化(即选择法线和投影方向的夹角最小的情况)。

```
/* 选择有效面积最大的那个方向，确定最终要使用的projection矩阵 */
if(normalX >= normalY && normalX >= normalZ)
    frag.axis = 1;
else if (normalY >= normalX && normalY >= normalZ)
    frag.axis = 2;
else
    frag.axis = 3;
```

二、光线计算：

1. 直接反射光：Phong 光照模型+阴影贴图

漫反射：

```
// 直接漫反射
float cosTheta = max(0, dot(N, L));//入射光线和法线余弦值
vec3 directDiffuseLight = ShowDiffuse > 0.5 ? vec3(visibility * cosTheta) : vec3(0.0);//是否让直接漫反射光可见
```

镜面反射：

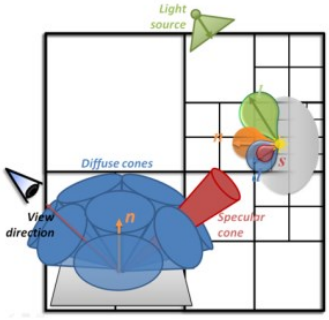
```
//直接镜面反射
vec3 reflectDir = normalize(reflect(-L, N));
float spec = pow(max(dot(E, reflectDir), 0.0),Shininess);
vec3 directSpecularLight = vec3 (spec * visibility);
directSpecularLight = ShowDirectSpecular > 0.5 ? directSpecularLight : vec3(0.0);
```

其中 visibility 代表从阴影贴图中求得的可见度，这里利用 PCF 从阴影贴图中采样计算以求得软阴影，效果表明阴影贴图分辨率越高，效果越好，当设为 4096*4096 时，已经有很好的效果。

```
float countShadow(float bias)  //bias是偏移值，防止阴影走样
{
   float current=Position_depth.z/Position_depth.w;
   float shadow = 0.0f;
   int radius = 2;
   for(int x=-radius;x<=radius;x++)   //这里用的是PCF,采样空间3x3
   {
      for(int y=-radius;y<=radius;y++)
      {
         vec2 offset = vec2(1.0/shadowMapRes * x, 1.0/shadowMapRes * y);
         float closest = texture(ShadowMap, vec2(Position_depth.xy+offset)).r;
         if(current - bias > closest)
            shadow+=0.0f;
         else
            shadow+=1.0f;
      }
   }
   shadow/=(2*radius+1)*(2*radius+1);
   return shadow;
}
```

2. 间接反射光：

利用之前建立的包含可见度信息的体素结构，来近似场景中经过直接照射后的新“光源”分布，这样经过对 diffuse 和 specular 的锥体中的光线进行收集，得到的就是二次反射光线下该物体的颜色，这个颜色值作为间接光分量，还要与对应的遮蔽因子相乘才是真正进入眼中的颜色值。



漫反射锥体：6*60 度，加权：1*0.25+5*0.15
镜面反射锥体：该方向与视角方向的角平分线应当是该片元处的法线，且三线共面，孔径大小依材质的光泽度而定。
环境光遮蔽：简化为环境光与漫反射遮蔽因子相乘

注意：间接反射光需要乘上对应的遮蔽因子作为最终求和时的反射光分量。

```
//间接漫反射
vec4 indirectDiffuseLight = vec4(0.f);
for(int i = 0; i < NUM_CONES; i++) //叠加六个锥体
{
   indirectDiffuseLight += coneWeights[i] * coneTrace(normalize(TBN * coneDirections[i]), 0.577);
}
occlusion = 1 - indirectDiffuseLight.a;
indirectDiffuseLight = ShowIndirectDiffuse > 0.5 ?  indirectDiffuseLight :vec4(0.0);
```

```
//间接镜面反射
 vec3 inlightDir = normalize(reflect(-E,N));
 vec4 tracedSpecular = coneTrace(inlightDir, tan(0.3)); //  0.07 = 8 degrees angle
 tracedSpecular = ShowIndirectSpecular > 0.5 ?  tracedSpecular : vec4(0.0);
 specularOcclusion = 1 - tracedSpecular.a;
```

```
//环境光
vec3 ambientLight =AmbientOcclusion>0.5? ambientFactor * materialColor.rgb* occlusion :ambientFactor * materialColor.rgb ;
```

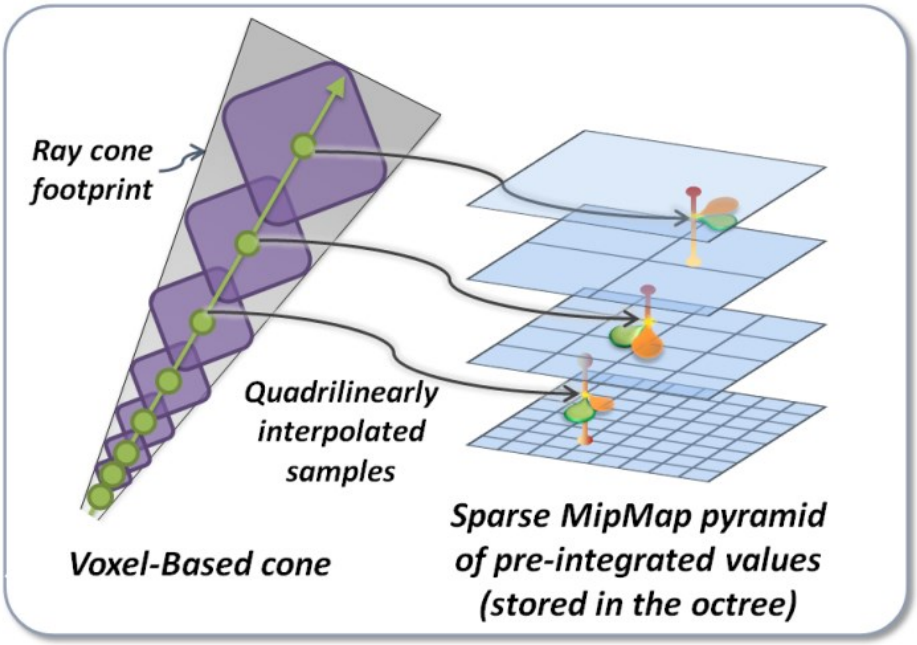
3. 分量求和:

```
// 漫反射求和
diffuseReflection =  (directDiffuseLight +    occlusion * indirectDiffuseLight.rgb ) * materialColor.rgb;
```

```
//镜面反射求和
specularReflection =   (tracedSpecular.rgb+ specularOcclusion * directSpecularLight) * specularColor.rgb ;
```

```
//光总量求和
color = vec4(ambientLight + diffuseReflection + specularReflection, alpha);
```

4. 体素锥追踪



从顶点附近一个体素距离的起始点开始沿中轴线步进，在到达最远距离或者已经完全被遮蔽后停止，光线依次积累这些体素上的遮蔽因子和颜色，可以认为是二次光源的光线颜色沿该方向照射到片元表面，调整步长、最远距离、可见度阈值、遮蔽因子衰减系数等以求最佳效果。

```
  //沿锥累积
vec4 coneTrace(vec3 direction, float tanHalfAngle) {
  //累积变量初始化
  float lod = 0.0; //mipmap级别
  vec3 color = vec3(0);
  float alpha = 0.0;
  float occlusion = 0.0;

  float voxelWorldSize = VoxelGridWorldSize / VoxelDimensions;  //单个体素的边长
  float dist = voxelWorldSize; // 避免自遮蔽
  vec3 startPos = Position_world + Normal_world * voxelWorldSize; //避免在平坦表面形成自遮蔽

  while(dist < MAX_DIST && alpha < ALPHA_THRESH ) {
    // 最小取样直径（三角形的底与体素大小间取大值)
    float diameter = max(voxelWorldSize, 2.0 * tanHalfAngle * dist);
    float lodLevel = log2(diameter / voxelWorldSize); //使用对数计算取样级别
    vec4 voxelColor = sampleVoxels(startPos + dist * direction, lodLevel);//取对应体素的颜色

    // 沿锥体中心线方向叠加
    color +=  (1.0 - alpha) * voxelColor.rgb;
    occlusion += ( (1.0 - alpha) * voxelColor.a) / (1.0 + 0.03 * diameter);
    alpha +=  (1.0 - alpha) * voxelColor.a;
    dist += diameter;
  }
```

由于体素直接存在 3D 纹理中通过 glGenerateMipmap()函数生成了渐远纹理，因此直接通过世界空间中的坐标来计算在体素空间中的坐标即可，然后取相应级别的纹理。

```
vec4 sampleVoxels(vec3 worldPosition, float lod) {
  //求在体素空间中的坐标以及属性
  vec3 voxelTextureUV = worldPosition / (VoxelGridWorldSize * 0.5);
  voxelTextureUV = voxelTextureUV * 0.5 + 0.5;
  return textureLod(VoxelTexture, voxelTextureUV, lod);
}
```

三、UI 界面：

```
Vendor: NVIDIA Corporation
Renderer: GeForce MX250/PCIe/SSE2
Version: 4.5.0 NVIDIA 452.41
GLSL: 4.50 NVIDIA
Initializing VCT
Loading UI ...
checking src/Shaders/standard.vert ...
checking src/Shaders/standard.frag ...
checking program ...
checking src/Shaders/voxelization.vert ...
checking src/Shaders/voxelization.frag ...
checking src/Shaders/voxelization.geom ...
checking program ...
checking src/Shaders/shadow.vert ...
checking src/Shaders/shadow.frag ...
checking program ...
Loading objects...
Loading done!
```

具体操作见图中 USE GUIDE 以及 README.txt.

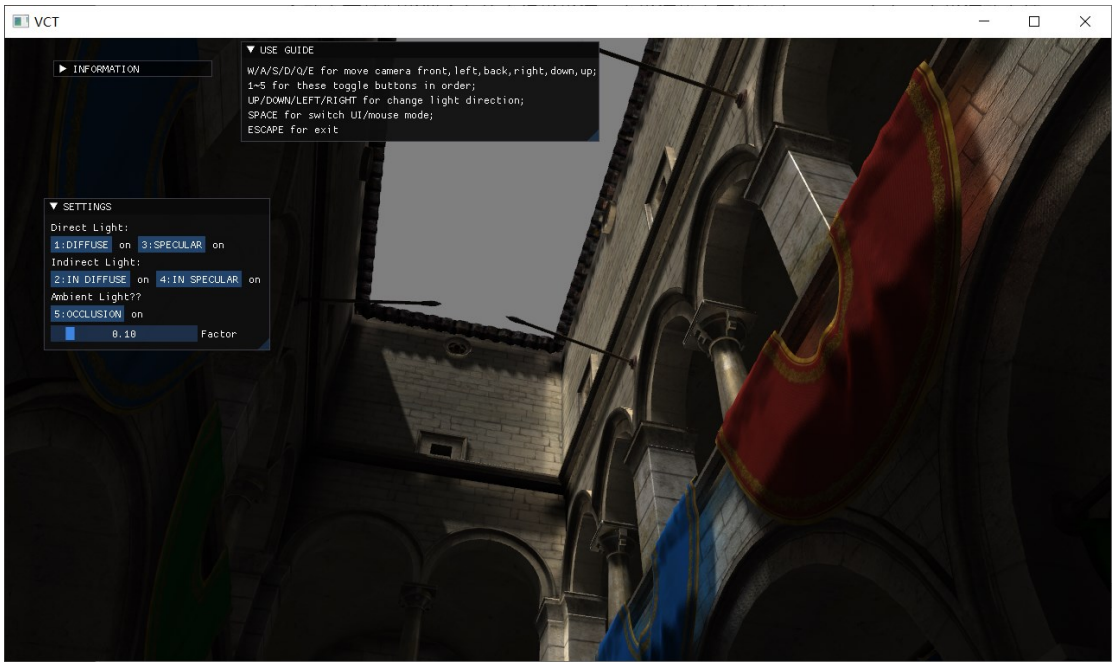
四、最终效果：

参数配置

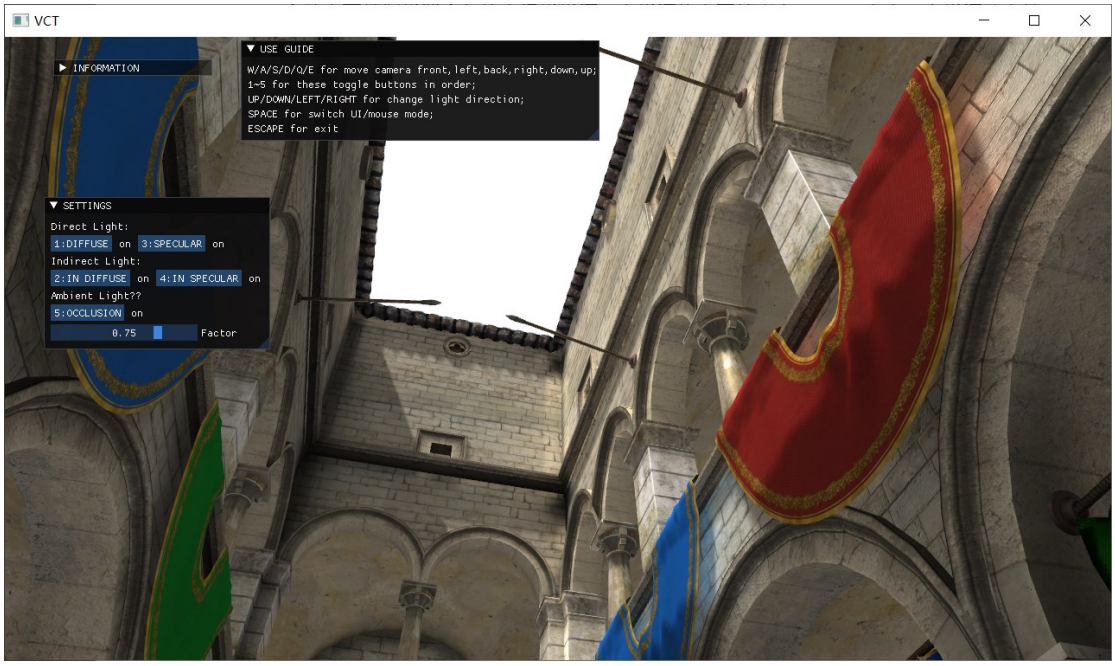
一个动态可控制光照方向的平行光光源，在光源静态下帧率在 144Hz 左右，在光源动态控制下帧率在 60Hz 左右（体素参数为 128 * 128 * 128，在 256 * 256 * 256 下可达 15Hz；阴影贴图分辨率参数为 4096 * 4096，在 8092 * 8092 下帧率稍稍降低）

效果举例：可以调整环境光因子模拟月光下、日光下的效果，如下：

较暗的环境光：



较亮的环境光：



其他具体效果请见 demo.

五、参考文献：

[1] Crassin C, Neyret F, Sainz M, et al. Interactive indirect illumination using voxel cone tracing[C]//Computer Graphics Forum. Wiley/Blackwell (10.1111), 2011, 30(7): 1921-1930.