动态场景面光源光照加速计算方法的研究与实现

第一章 绪论

1.1 论文的选题背景及来源

真实感绘制作为图形学的重要组成部分，广泛应用于娱乐，医疗，军事。特别是最近两年间VR的

@阴影在光照计算中使用的位置@

第二章 研究现状与相关技术

@什么是软阴影和硬阴影@

@透视投影方法@

第三章 精确的面光源阴影计算方法

3.1 引言

@精确的面光源阴影是什么及重要性 ， 限定1离散化面光源@

@目前现有的计算方法， RT 耗时，优化后的RT不能支持全动态场景@

@本章给出的方法@

本章方法的贡献主要有两点：

第一 。。。

第二．。。。

3.2 4D光栅化平面算法概述

本算法的输入为：一个由三角形组成的3D 场景S，一个矩形面光源*L0L1L2L3* , 以及一个给定的视点V。我们让相机从给定的视点V观察3D场景S，屏幕上每个像素都会对应到三维空间中的一个点（这里我们只考虑小孔成像@参考文献@），该点称为屏幕像素的采样点。采样点处的光照值将会决定屏幕上像素的颜色值。在使用光照模型（@Phone光照参考@）进行光照值的计算时，我们需要提前得到采样点处的阴影值。4D光栅化平面算法的目的就是求解所有采样点处的阴影值。正如引言中所述，软阴影的阴影值范围为[0, 1]，0 表示该像素处于全阴影区，1表示该像素被面光源*L0L1L2L3* 完全照亮，其他值表示该像素处于半影区。4D光栅化平面算法会将输出的阴影值保存在一张2D纹理中。

首先考虑由单个点光源*L*生成硬阴影的算法。我们可以将屏幕像素对应的采样点和三角形投影到一个帧缓存（被均匀划分成许多个矩形格子的图像平面）中。如果采样点*S* 的投影和三角形*T*的投影都覆盖了帧缓存的同一个像素*p*，我们再测试三角形*T*是否遮挡了采样点*S*。如果不使用帧缓存，我们就需要测试所有的三角形和采样点的组合。使用了帧缓存后，我们只需要测试所有三角形和采样点组合中的很小一部分。在这里我们只是使用帧缓存排除很大一部分不可能产生遮挡的三角形和采样点的组合，之后还需要对剩下的每一个三角形和采样点组合的精确地判断出遮挡关系。

我们将上面的方法推广到矩形面光源软阴影的计算中。矩形面光源可以看作是点光源在两个自由度上移动产生的。随着点光源的移动，采样点和三角形在帧缓存上的投影也会随之移动。我们将投影移动所经过的位置称作其在4D光栅化平面上的投影。使用采样点和三角形在4D光栅化平面投影的关系，我们就可以得到需要测试遮挡关系的采样点和三角形的组合，而不用对所有的采样点和三角形的组合进行测试。

算法的具体步骤如下：

**Algorithm 1** 4D-rasterization for fast soft shadow rendering

**Input:** 3D scene *SCE* modeled with triangles, light rectangle *L*, output view *V*, light discretization resolution *m* x *n*, 4D rasterization framebuffer resolution *w* x *h*.

**Output:** *SCE* rendered from *V* with an *m* x *n* occlusion mask of *L* for each output sample *s*.

1. **STEP 1: render output image without shadows**
2. Render *SCE* from *V* to preliminary image *I*
3. **STEP 2: define frame buffer for 4D rasterization**
4. Define plane *P* for 4D rasterization frame buffer *FB*
5. **for each** pixel *s* in *I* **do**
6. *FB.aabb* = *FB.aabb* ∪ Proj4(*s.xyz*, *L*, *P*)
7. **STEP 3: assign output image samples to *FB* pixels**
8. **for each** output image sample *s* in *I* **do**
9. *s.aabb* = Proj4(*s.xyz*, *L*, *FB*)
10. **for each** *FB* pixel *p* in *s.aabb* **do**
11. *p.sampleSet* = *p.sampleSet* ∪{*s*}
12. **STEP 4: assign triangles to *FB* pixels**
13. **for each** triangle *t* in *S* **do**
14. *t.aabb* = Proj4(*t.v*0, *L*, *FB*) ∪ Proj4(*t.v*1, *L*, *FB*) ∪
15. Proj4(*t.v*2, *L*, *FB*)

16. **for each** *FB* pixel *p* in *t.aabb* **do**

1. *p.triangleSet* = *p.triangleSet* ∪ {*t*}

18. **STEP 5: computation of light occlusion masks**

19. **for each** *FB* pixel *p* **do**

20. **for each** triangle *t* in *p.triangleSet* **do**

21. **for each** output sample *s* in *p.sampleSet* **do**

22. **for** *i* = 1 to *m* **do**

23. **for** *j* = 1 to *n* **do**

24. *s.mask*ij |= Occlusion(*L*ij, *t*, *s.xyz*)

该算法由5个步骤组成，而这5个步骤可以分为两大部分，STEP 1~4 用来构建4D光栅化的帧缓存，STEP 5根据帧缓存进行阴影值计算，下面是各个步骤的详细解释。

**STEP 1** 从给定的视点*V*绘制一遍场景，从而得到屏幕像素对应的采样点。同时在这一步骤中我们使用Defer Shading 保存了采样点的法线，纹理等信息。关于Defer shading在第@@节有详细解释。

**STEP 2** 用来确定4D光栅化平面（下面简称*FB）*的具体参数。首先*FB*所在平面是平行于矩形面光源。且其位置离矩形面光源足够远，以保证场景中的物体在矩形面光源与*FB*所在平面之间。在实现中*FB*所在平面离面光源的位置是由场景轴向包围盒（简称AABB）对角线的长度决定的（第4行）。*FB*上格子的划分的轴向与矩形面光源的轴向相同。*FB*上格子划分的范围由**STEP 1** 中采样点在*FB*平面上投影的范围（投影方法见@ 3.3节@）决定（第5-6行）。

**STEP 3** 和 **STEP 4** 将采样点和三角形加入到*FB*像素链表中，这样就隐含地得到需要进行测试的采样点和三角形组合。一个采样点会被加入到其投影覆盖的所有的FB像素中（第8-11行），一个三角形的投影范围被定义为3个顶点在*FB*上投影的轴向包围盒（第13-17行）。具体的投影方法将在3.3节解释。

**STEP 5** 为每个屏幕像素的采样点计算一个记录遮挡信息的Bit-map。在4D光栅化帧缓存的每个像素中，存储了一个三角形链表和一个采样点链表。对于这两个链表组合出所有的三角形和采样点对，函数Occlusion(*L*ij, *t*, *s.xyz*)会检查*m x n*个点光源中的每个光源采样点Lij，屏幕像素采样点s是否被三角形t遮挡了从点光源采样点Lij处射来的光。如果遮挡了，函数Occlusion(*L*ij, *t*, *s.xyz*) 返回1，否则返回0。每个三角形对s求出的结果最终通过逻辑“与”操作进行合并。

本节介绍了算法的大致步骤，下面一节给出投影的具体实现方式。

3.3 4D光栅化投影方法

首先定义三维点的4D光栅化投影方法。如图 1所示，矩形*L0L1L2L3*表示场景中的面光源，*S*为三维世界中的一个点。从矩形面光源的四个顶点*L0*，*L1*，*L2*和*L3*处向点*S*连线，直线*L0S*，*L1S*，*L2S*和*L3S*与*FB*所在平面交于*L0`*，*L1`*，*L2`*和*L3`*四个点。之前在3.2节4D光栅化平面算法概述中已经定义*FB*平面是与面光源*L0L1L2L3*所在平面平行的。*L0L1*和*L0`L1`*是平面*L0L1S*与平面*L0L1L2L3*和*FB*平面的交线，因此*L0L1*∥*L0`L1`*，同理可得*L1L2*∥*L1`L2`*，*L2L3*∥*L2`L3`*和*L3L0*∥*L3`L0`*，矩形*L0`L1`L2`L3`*表示三维点*S*在*FB*平面上的4D光栅化投影。并且由于投影矩形*L0`L1`L2`L3`*与矩形*L0L1L2L3*具有相同的轴向。而*FB*划分像素的轴向与矩形*L0L1L2L3*相同，因此投影矩形*L0`L1`L2`L3`*与*FB*划分像素的轴向相同，这点为编程实现提供了便利。

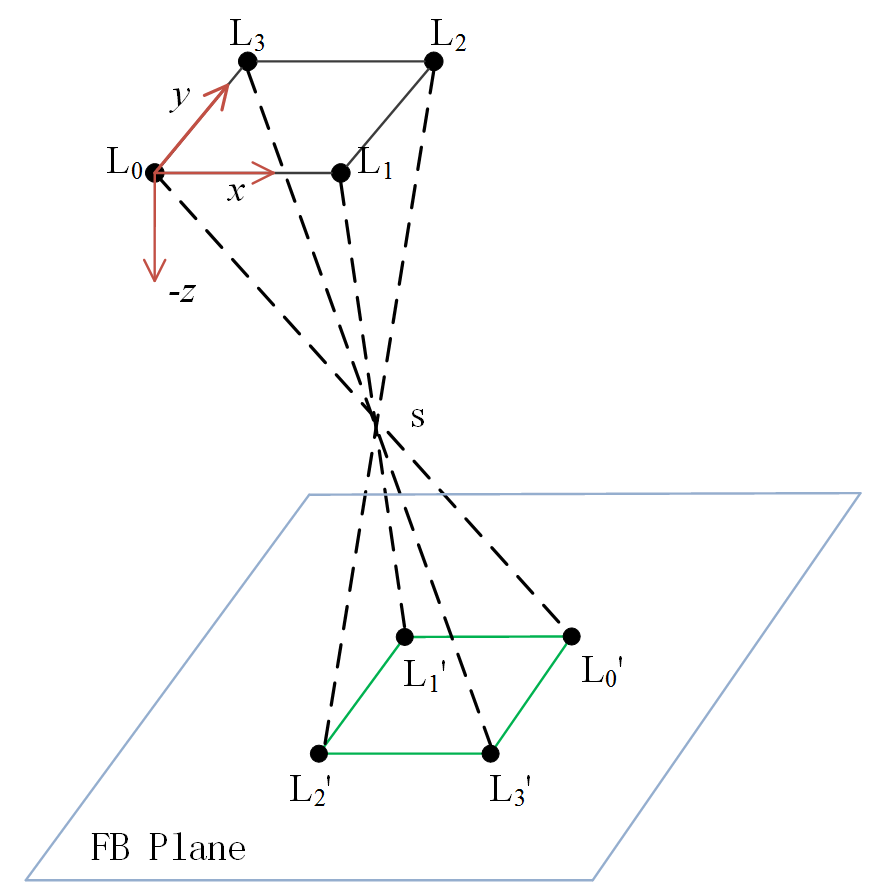


图 1对三维点进行4D光栅化投影示意图

帧缓存的格子划分范围是由屏幕像素采样点的投影范围决定的。@图X @表示2D *FB*的构建过程。在这幅2D图中采样点*S1*在平面*P*上的投影是*BC*。从*A*到*D*包括了所有采样点投影，因此*AD*表示了帧缓存划分格子的范围。

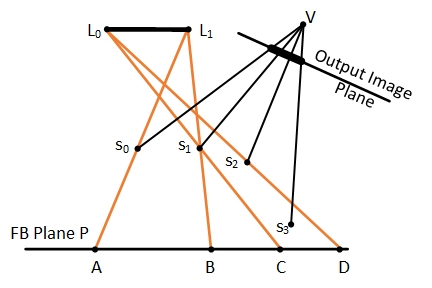


图 2帧缓存所在平面P平行于矩形光源L0L1。采样点S1在P上的投影是BC。从A到D包括了所有采样点投影。

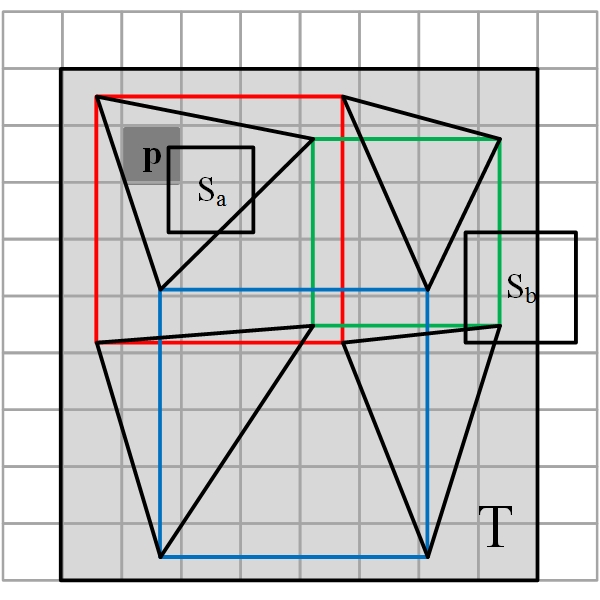


图 3三角形的投影范围

三角形的4D光栅化投影定义为三角形三个顶点的4D光栅化投影的轴向包围盒。@如图@中，三角形T三个顶点的投影分别为红色，绿色和蓝色三个矩形表示。三角形T的投影范围为红色，绿色和蓝色三个矩形的轴向包围盒（灰色填充的矩形）。

图 3同时表示了三角形*T*和采样点像素*S*之间关系的建立过程。*p*表示*FB*的一个像素。*S*和*T*的投影都覆盖了像素*p*，因此*S*和*T*都会加入到像素*p*的链表中。在算法1 的STEP 5 中，会产生S和T的组合，从而会判断S与T的遮挡关系。

3.4 CUDA实现细节

3.4.1 4D光栅化投影公式的推导

在3.3节中我们定义了4D光栅化对三维空间一点的投影：从矩形面光源的四个顶点向点*S*作4条直线，这4条直线与FB平面有四个交点。然后按次序连接这4个交点可以得到FB平面上的一个矩形投影。与透视投影公式推导类似@2.x节中透视投影@，4D光栅化投影在X轴和Y轴方向上是独立的。这里首先给出X轴投影的公式：

为了方便计算，坐标系原点放在L0处，坐标系的x轴朝向L1，y轴轴朝向L3，z轴垂直于光源平面向上（如图 1对三维点进行4D光栅化投影示意图所示）。

如图 4中所示L0L1表示面光源其宽度为*w*，面光源与FB平面的距离为d，对三维点s（x, y, z）投影后投影点为L0`，L1`，投影矩形的宽度为*w`*。

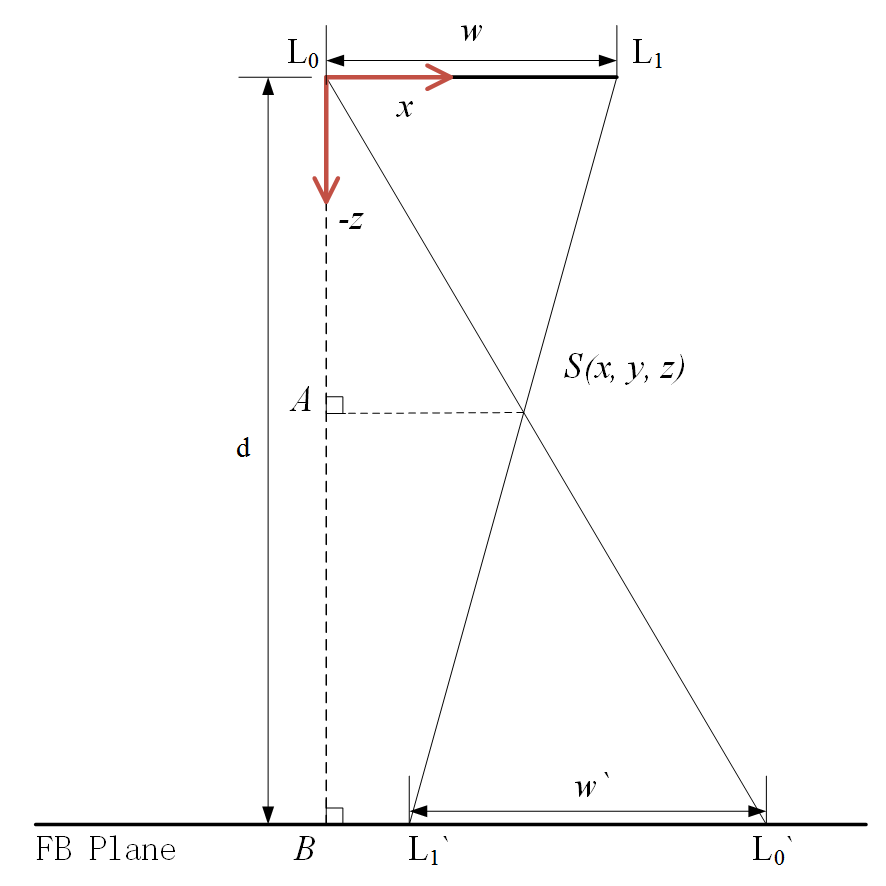


图 4 4D光栅化投影公式推导示意图

图中辅助线L0B ⊥ FB 平面，辅助线AS⊥L0B。由于△L0AS ∽ △L0B L0` 可得：

代入数值得

由于L0L1 ∥FB 平面 所以有 △ L0S L1 ∽ △ L0`S L1` 可得：

3.4.2 GPU中建立链表

3.4.3 GPU 实现规约操作

3.5 光照计算

3.6 实验效果与分析

3.7 本章小结

第四章 精确的面光源阴影计算方法的效率优化

第五章 面光源光照计算方法

结论与展望