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@

@CUDA 中原子操作会降低并行效率@

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

3.3.1 4D光栅化投影方法的定义

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。

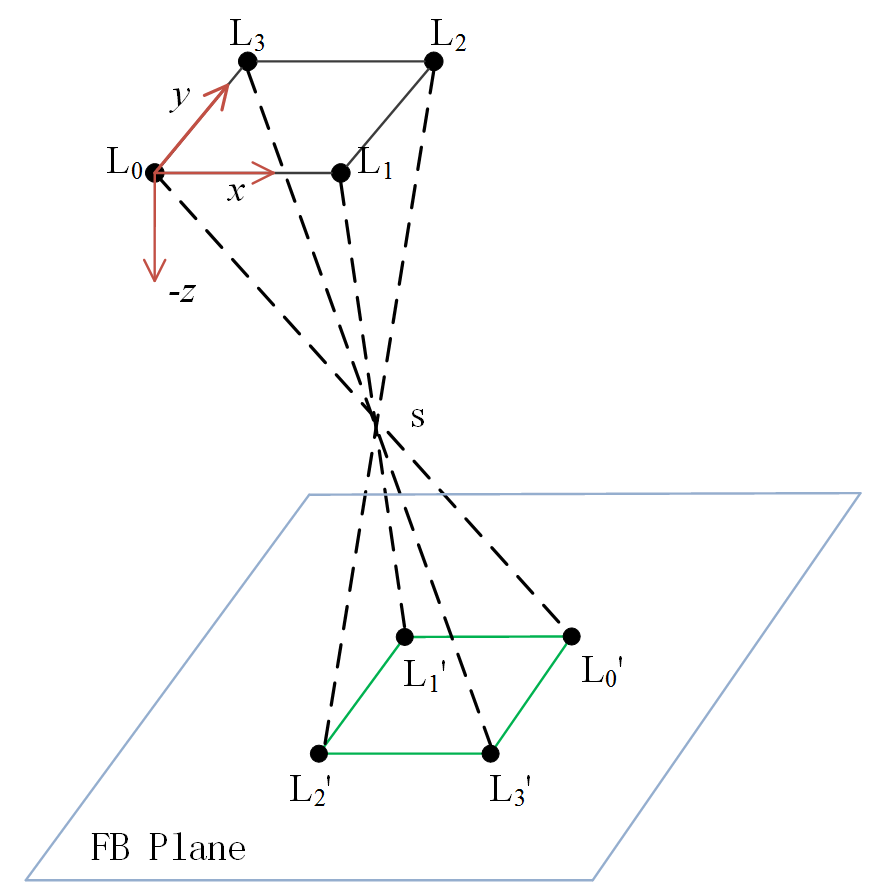


图 1对三维点进行4D光栅化投影示意图

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。

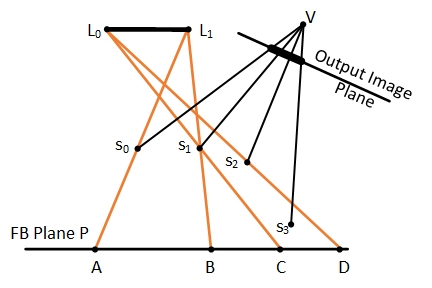


图 2帧缓存所在平面P平行于矩形光源L0L1。采样点S1在P上的投影是BC。从A到D包括了所有采样点投影。

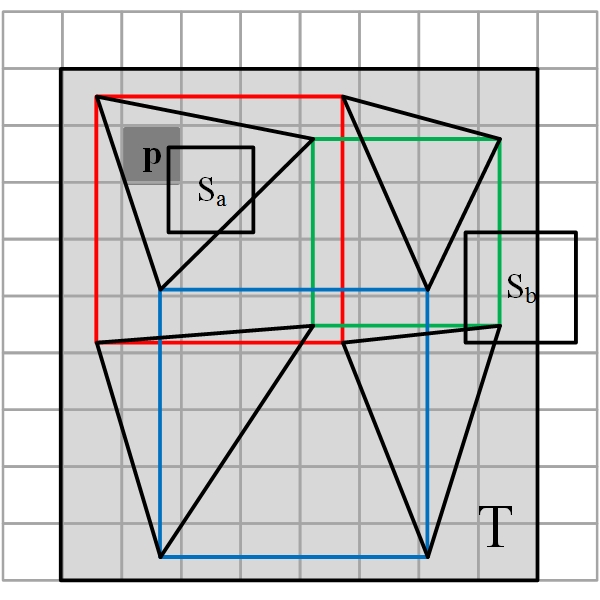


图 3三角形的投影范围

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。

3.3.2 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`*。

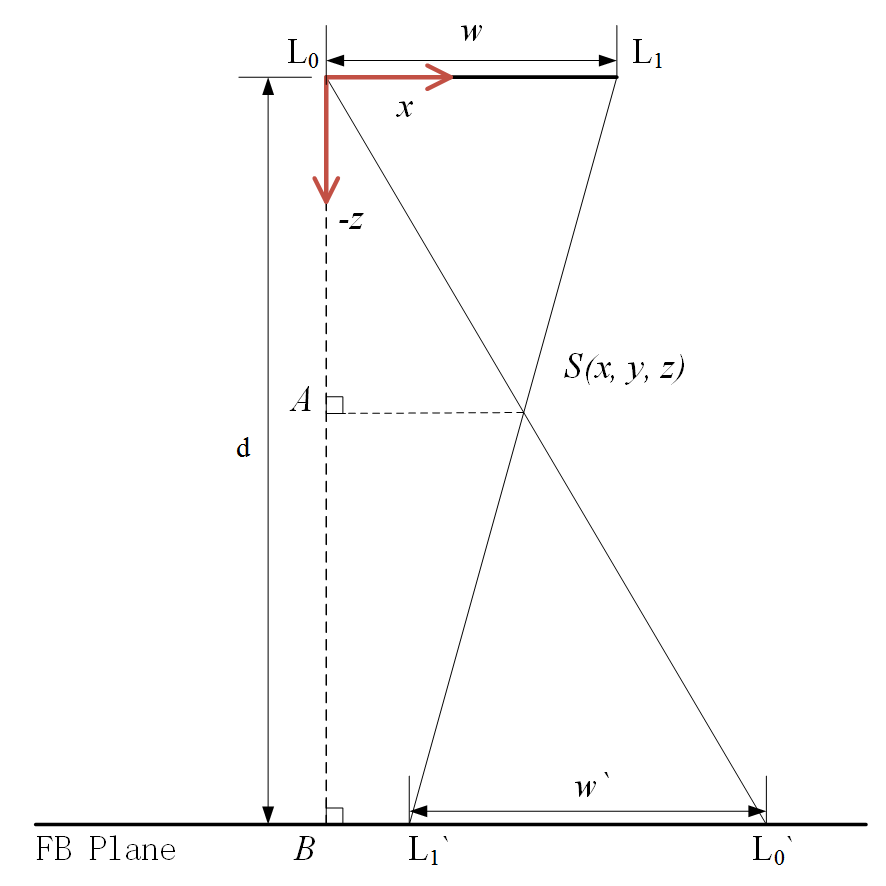


图 4 4D光栅化投影公式推导示意图

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L0在投影平面上对应点L0`的x坐标为：

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

代入数值得到投影矩形的宽度：

类比公式 （2） 和公式（4）可得到：

L0`的y坐标为：

投影矩形的高度为：

投影矩形四个顶点的二维坐标分别为：

L0`

L1`

L2`

L3`

3.5 GPU中建立链表

两种方法： 1 申请一块足够多节点，使用Atomic操作分配节点。

2，使用并行的基数排序算法。

3.6 光照可见性计算及使用

上一节中介绍了本章中简易光栅器的实现，并应用该光栅器计算出了每个像素采样

点对应的位图掩码表。下一步首先需要利用该位图掩码表进一步计算出每个像素采样点

的精确可见性信息，并保存于一张阴影纹理贴图中。再利用延迟着色（deffered shading）

技术，应用基于 Phong 光照模型所保留的屏幕空间场景信息，在屏幕空间中计算出场景

的光照并应用生成的阴影纹理贴图，最终绘制出带有光影的场景。整个绘制阶段只需要

一遍场景绘制和一次延迟着色处理。

3.4.1 获取场景阴影图

在 CUDA 的运算中，并不可以直接的访问图形 API 中使用的缓冲区对象以及纹理

资源，而是以注册的方式将图形 API 生成的阴影图纹理与 CUDA 中的资源句柄进行绑定，以便之后在使用的过程中将二者在存储空间中进行映射。在实现中，我们先用 CUDA提供的运行时 API 在显卡中分配一段线性浮点类型数组，数组的大小与输出图像中的像素采样点个数一致，数组中的元素与采样像素点一一对应，用来保存每个像素采样点的光照可见度。然后将该数组作为参数传入光栅化的核函数中。在 CUDA 函数中，对位图掩码表（图 11 所示是对一个 16x16 分辨率的位图掩码的一个可视化）进行归约操作，即统计该位图掩码表中置 1 的元素个数（图 11 中的白色区域面积）。置 1 的元素个数与掩码表中总元素个数（图 11 总的面积）的比值即为该像素采样点的遮挡比例。最后使用遮挡比例求出该像素点的光照可见度，并存储到浮点数组。

最后，将可见性信息拷贝到与阴影图纹理对应的 cudaResource 资源中，利用其与阴

影图纹理的存储空间映射关系，得到更新的阴影图纹理。这样可以在之后的图形硬件流

水线中使用该阴影纹理贴图。一个阴影图的示例如图 12 所示。



3.4.2 延迟绘制场景的光影

本章中使用 Phong 光照模型完成场景的光照渲染。绘制中使用了 MRT（多渲染对象，Multiple Rendering Targets）技术与延迟着色技术，减少一遍场景绘制。

在经典的 Phong 光照模型中，一个物体的表面颜色是放射（emissive）、环境反射（ambient）、漫反射（diffuse）和镜面反射（specular）等光照作用的总和。每种光照作用均取决于表面材质的性质（例如亮度和材质颜色等）和光源的性质（例如光的颜色和位置等）的共同作用。公式(3.1)表示了该基本光照模型。

3surfaceColor emissive ambient diffuse specular

放射项表示由物体表面所发出的光，它指明了表面所发出的光的颜色，在本章测试场景中，将不涉及此类材质，故忽略该项的影响。环境反射项代表光在一个场景里经过多次反射后看起来像均匀地来自四面八方，环境反射光照项并不依赖于光源的位置。环境反射项依赖于一个材质的环境反射特性，以及照射到材质上的环境光的颜色，是一种固 定 的 颜 色 ， 但 是 其 强 弱 也 受 到 全 局 环 境 光 照 的 影 响 ， 可 以 用 公 式@@表示，其中， a K 是材质的环境反射系数，globalAmbient 是环境光的颜色。漫反射项代表从一个表面相等地向所有方向反射出去的方向光。光的反射量与光到达表面的入射角度成正比，可以用公式 max( ,0) d diffuse K lightColor N L    表示，其中， d K 是材质的漫反射系数，lightColor 是入射漫反射光的颜色，N 是规范化的表面法向量，L 是规范化的指向光源的向量。镜面反射项代表从一个表面主要的反射方向附近被反射的光，通常在光滑的表面上最为显著。镜面反射的作用依赖于观察者的第三章 面光源像素级精确光照可见性判断方法 30 位置，而且受到光源和材质的镜面反射颜色性质以及表面光泽度的影响，因此镜面反射项可以用公式表示，其中 s K 是材

质的镜面反射系数，shininess 表示物体表面的光泽度，光泽度越大，物体的镜面反射能

力超强，H 是 V（指向视点的规范化向量）和 L 的中间向量的规范化向量，facing 取 1如果NL的值大于零，否则取 0。

延迟着色是一种对 3-D 场景进行后期照明的技术。思路是：首先，将 3-D 场景的几何光照信息（位置、法线、材质）绘制到渲染目标（render target）上，把它们从世界的三维空间转变成屏幕的颜色空间，作为光照计算时的输入。然后，对每一个光源，使用这些信息进行计算生成一帧并合成到帧缓存上，最终帧缓存上的图像即为渲染结果。

基于本章算法的常规绘制流程描述如下：

（1）采样点信息采集阶段。从视点绘制一遍场景。

（2）光照可见度计算阶段。计算像素采样点光照可见度。

（3）阴影渲染阶段。从视点方向绘制一遍场景，利用 Phong 光照模型计算每个像素点的漫反射分量和镜面反射分量。然后结合光照可见度计算最终绘制效果。

从以上流程中可以看出，在同一视点下需要对场景绘制两遍。然而，如果应用延迟着色的思想则可以减少一遍场景绘制的开销。改进后的算法的流程（图 13）如下：

1. 场景信息采集阶段。

使用 MRT 技术，先在 FBO（Frame Buffer Object，帧缓冲区对象）中绑定三张纹理作为输出缓冲区，这三张纹理分别用来保存像素采样点的三维坐标、每个像素点对应的颜色信息以及之后在延迟着色阶段需要使用到的若干光照信息。然后在绘制阶段，绑定该 FBO，使用 RTT（Render to Texture，渲染到纹理）技术，从视点方向绘制一遍场景：将每个像素采样点的三维坐标输出到 COLOR0 颜色通道中，保存每个像素采样点的三维 坐标（如图 13 中右上图所示）；将每个像素采样点本身的颜色值输出到 COLOR1 颜色通 道中；将计算 Phong 光照需要的一些信息输出到 COLOR2 颜色通道中。

3.7 实验效果与分析

本节从实验效果，实验数据分析以及参数定量分析等各个方面，对本文算法提供数据论据，并对数据的一些规律进行解释与分析。

3.7.1 实验的软硬件平台

本章实验所采用的硬件平台，软件平台以及开发环境和场景模型如下：

硬件平台环境为：PC平台，Intel® Core™ i7-2600k CPU @3.4GHz 3.70GHz，16G内存，NVIDA GeForce GTX 960 显卡。

软件平台及开发环境为：Windows 10 操作系统，编译环境为Microsoft Visual Studio Community 2015 Update 3 C++ 编译器，OpenGL 4.5 API，NVIDIA CUDA Toolkit v8.0， CUDA SDK Version 8.0， Optix SDK Version 4.0。

实验中所用到的场景信息如表1所示。

|  |  |
| --- | --- |
| 场景（模型）名称 | 三角形面片数 |
| 树 | 367k |
| 铁塔 | 514k |
| 玩具 | 263k |
| 龙 | 81k |
| 过山车 | 1096k |
| 教堂 | 240k |

3.7.2 静态场景软阴影绘制效果

图 5和图 6所示是本文方法在静态场景中的软阴影绘制效果。对于由三角形面片所组成的不同类型，不同面片数的复杂场景而言，使用本文提出的方法均能够获得很好的绘制效果。本文的方法不需要对场景中的阴影遮挡体与阴影接受体进行划分，且对场景中的几何形体无限制。不要求阴影接受体是平面，也不要求阴影遮挡体在空间上具有较强的几何连续性。本文方法不要求对场景进行预处理，而是每一帧都独立运算，因此动态场景和静态场景的绘制过程和效率都是相同的。

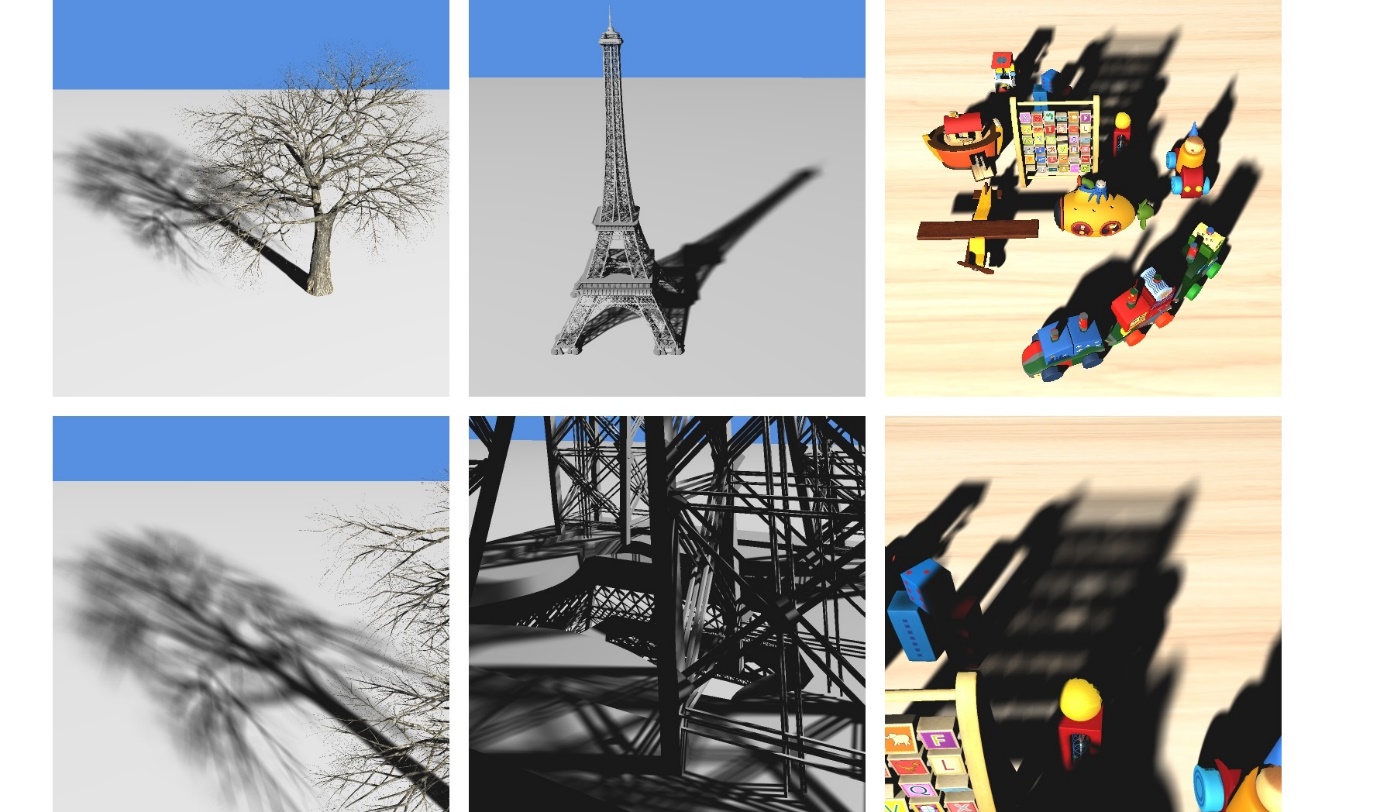


图 5 树，铁塔和玩具绘制效果

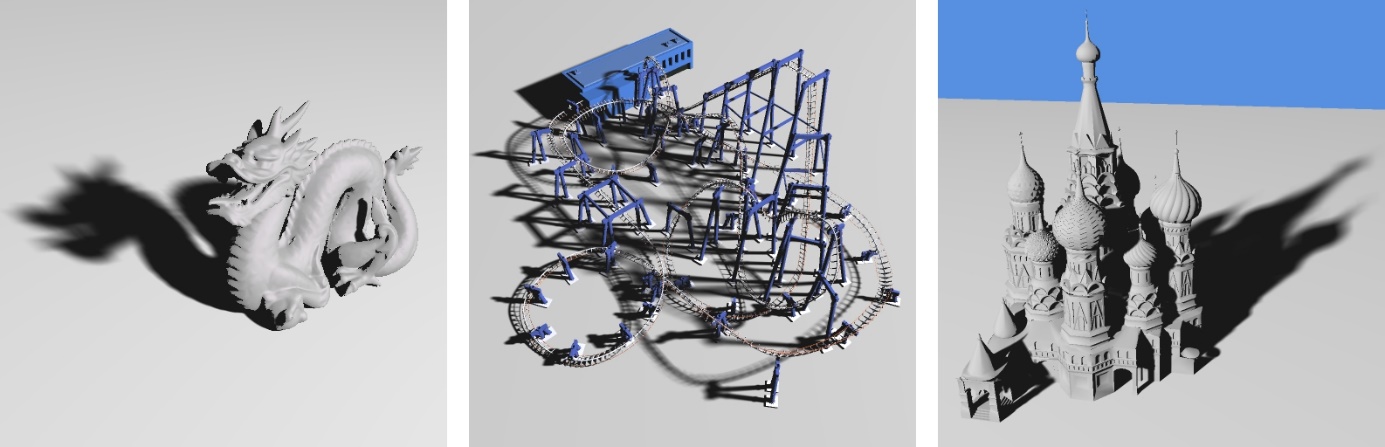


图 6 龙，过山车和教堂的绘制效果

3.7.3 与光线追踪方法的效果对比

本节对本章方法与使用了Optix引擎所实现的光线追踪方法进行实验比较。在相同视点，相同面光源参数（面光源可见性采样分辨率均为32x32，面光源尺寸均为2）的情况下，分别对树，铁塔和玩具场景进行阴影绘制。如图 7和图 8所示，分别是使用了本文算法的绘制结果（图 7左侧图，图 8两组图的左侧图）与使用了光线追踪的算法（图 7右侧图，图 8两组图的右侧图）的绘制结果对比。两种方法生成的图像相减无残差，证明本文方法可以达到像素级光照可见性判断。



图 7 本章方法（左）与光线追踪方法（右）的绘制效果对比1



图 8 本章方法（每组左图）与光线追踪方法（每组右图）的绘制效果对比2

3.8 本章小结

第四章 精确的面光源阴影计算方法的效率优化

4.1 证明全覆盖

4.2 行求交

4.3 GPU 归并

4.4 排除自阴影

第五章 面光源光照计算方法

结论与展望