Science Discovery
2017; 5(1): 40-47
http://www.sciencepublishinggroup.com/j/sd
doi: 10.11648/j.sd.20170501.17
ISSN: 2331-0642 (Print); ISSN: 2331-0650 (Online)

Research on Fast Soft Shadow Algorithm Based on 4D Rasterization

Zhao Qi*, Wang Lili

State Key Laboratory of Virtual Reality Technology and Systems, Bei Hang University, Beijing, China

Email address:
zhaoqi606@buaa.edu.cn (Zhao Qi), wanglily@buaa.edu.cn (Wang Lili)
*Corresponding author

To cite this article:
Zhao Qi, Wang Lili. Research on Fast Soft Shadow Algorithm Based on 4D Rasterization. *Science Discovery*. Vol. 5, No. 1, 2017, pp. 40-47.
doi: 10.11648/j.sd.20170501.17

Received: March 27, 2017; **Accepted**: April 6, 2017; **Published**: April 12, 2017

Abstract: This paper introduces an algorithm that extends traditional 2D projection and rasterization to 4D space for fast soft shadow rendering. First, the rectangular area light source is seen as a point light source that translates with two degrees of freedom. As the point light source moving, the projections of the triangles and the output image samples on the projection plane are also moving. The locus of the projection is called the 4D projection. 4D projections are rasterized, and a conservative set of output image sample/triangle pairs can be obtained. The set is then examined to derive light mask for each sample. Since all potentially blocking triangles are considered, the algorithm is an accurate algorithm. And, the algorithm does not require any type of precomputation, so it supports fully dynamic scene. We have tested our algorithm on several scenes to render complex soft shadows accurately at interactive rates.

Keywords: Photorealistic Rendering, Soft Shadow, Three-Dimensional Graphics

基于4D光栅化的快速软阴影算法的研究

赵琦*，王莉莉

虚拟现实技术与系统国家重点实验室，北京航空航天大学，北京，中国

邮箱
zhaoqi606@buaa.edu.cn（赵琦），wanglily@buaa.edu.cn（王莉莉）

摘要：本文介绍了一种将传统的2D投影和光栅化方法推广到4D空间，从而进行高效软阴影渲染的算法。首先，矩形面光源被看作带有两个自由度的点光源。随着点光源的移动，三角形和输出图像采样点在投影面上的投影也在移动。投影移动中经过的区域被称为4D投影。随后，对4D投影进行光栅化，可以得到采样点/三角形对的一个保守集合。之后对这个集合进行运算，可以得到每个采样点处的光源掩码。由于在采样点的光源掩码计算中，考虑了所有可能产生遮挡的三角形，因此该算法是精确的算法。并且，该算法不需要任何预计算，支持全动态场景。在多个场景中进行测试，本文算法能够以可交互帧率，精确地绘制出复杂的软阴影效果。

关键词：真实感绘制，软阴影，三维图形学

1. 引言

在计算机图形学应用中，许多光源都具有一定的面积。这些光源投射出软阴影，精确计算得到的软阴影能够在很大程度上提高渲染图像的质量。然而，渲染软阴影是十分耗时的，因为必须为数以百万计的图像采样点估计出光源可见性的比率。一种常见的方法是将区域光源离散为成几百个，甚至几千个点光源，这导致在每一帧中必须计算数十亿光线的点对点可见性。

本文提出了一种以交互式速率，渲染复杂软阴影的算法。该算法将矩形面光源看作具有两个平移自由度的点光源。通过从光源处对输出图像采样点和场景三角形进行投影和光栅化，从而估计出每个采样点处的面光源可见性。首先，随着点光源在矩形面光源上进行两个自由度的平移，生成的投影也会在两个自由度上移动。之后，对三角形和输出图像采样点的生成投影进行光栅化，从而定义出一个即保守又严格的输出图像采样点/三角形对的组合。最后，对这些组合进行检测从而为每个输出图像采样点生成一个面光源遮挡掩码。面光源遮挡掩码是一个的2D位图，如果光源采样点被遮挡则为1，否则为0。由固定点光源投射的硬阴影是利用常规2D光栅化来完成的；而我们的4D光栅化算法在2D光栅化基础上，又考虑了点光源在矩形面光源上的两个自由度的平移。

本文的算法可以精确地计算出每个输出图像采样点和每个光源采样点之间的可见性。 并且不需要任何预计算，因此它支持完全动态场景，移动光源、更改光源大小以及移动物体、对物体变形等。本文算法已经在多个具有复杂软阴影的场景上进行验证。图1中的软阴影是通过以32×32均匀采样率对矩形面光源进行采样，得到的渲染结果。本文算法产生的图像与通过光线跟踪以相同的面光源采样分辨率渲染的图像相同，但是帧速率为15至35倍。 其中光线追踪算法使用NVIDIA的具有BVH（bounding volume hierarchy）加速度的Optix光线跟踪器。玩具场景是动态的，所以光线跟踪必须重新计算每帧的BVH。 树和塔的场景是静态的，因此图1中的光线跟踪性能不包括计算BVH的时间。

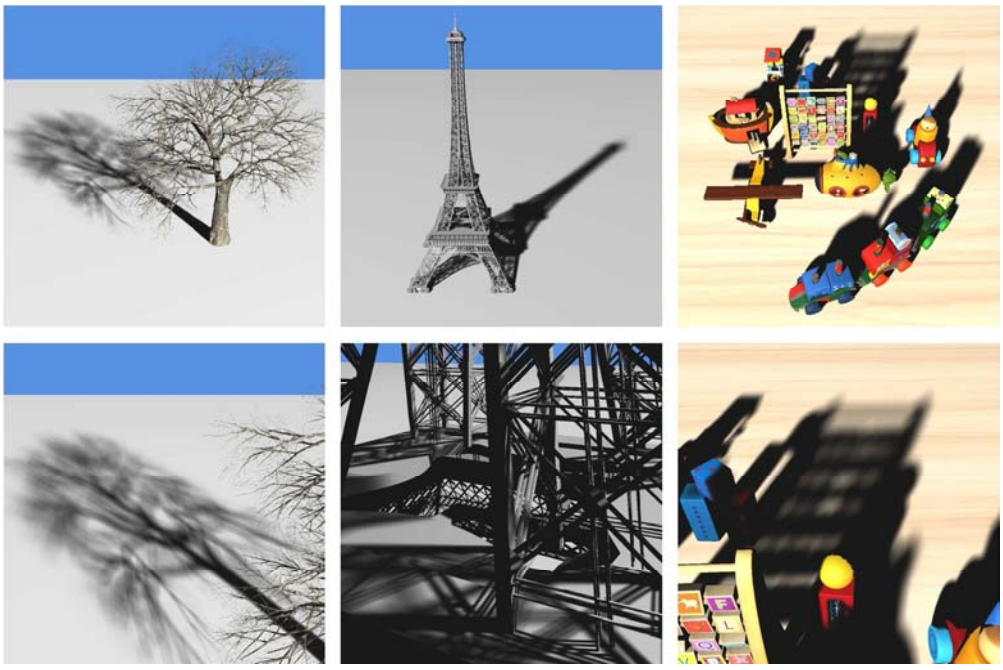

图1 使用本文方法绘制的软阴影效果。在相同的光源采样率下，本文方法与光线追踪方法生成的图像逐像素对比无差别。本文方法和光线追踪方法的平均帧率对比分别为：树5.2fps vs 0.34fps；铁塔10.0fps vs 0.29fps；玩具15.0fps vs 0.60fps。对应的加速比分别为15倍，35倍和25倍。

2. 相关工作

业界已经对软阴影渲染进行了广泛的研究。 除了这个简短的概述，还可以参考[1，2]中关于软阴影的综述。

阴影模拟方法为硬阴影匹配合适的半影区域，从而获得软阴影[3，4，5]。比如，软阴影可以通过对阴影边缘进行模糊的得到[6，7]。

阴影近似法简化了遮挡的几何体，以加速可见性查询。例如，反投影方法使用阴影图近似遮挡的几何体[8]。阴影图方法可以通过平滑轮廓检测、径向面积积分[9]、微小四边形和微三角形[10]等方法得到效果的改进。同时，使用多张阴影图可以减少从单个视点[11]采样产生的阴影失真现象。最后，还可以使用指数滤波[12，13]和随机采样[14]等方法对效果作进一步提升。

准确的软阴影方法能够正确地计算输出图像样本和光样本之间的可见性。本文的方法属于这个类别。光线跟踪[15]通过在输出图像采样点和光源采样点之间跟踪光线来提供对软阴影的自然支持，但是其性能低下。一种方法对每个像素投射稀疏和可变的光线集合以获得粗糙的可见性估计，然后自适应地滤波[16，17]。该方法通过减少射线的数量来获得性能，但这是以降低质量为代价的。

分层遮挡体方法用于估计由三角形投射的软阴影[18]，该方法的缺点是对光源尺寸的变化比较敏感。软阴影体算法使用物体轮廓，而不是所有三角形边缘进行阴影计算[19]，通过忽略不影响软阴影的内部三角形来提高效

率。与本文的方法类似，软阴影体方法将可能存在遮挡的三角形绑定在输出图像采样点中。该类方法的性能受到场景中物体复杂度和碎片化的网格影响。Forest等人[20]加速了该方法，但是仍然不能很好地处理碎片网格。

惰性可见性评估方法[21]通过在Plucker空间中对光线进行分组来计算准确的软阴影。由于需要为每个遮挡的三角形建立一个BSP树，因此在复杂场景中，可扩展性比较差。一种用于估计两个矩形片之间的可见性的方法可以应用于软阴影，但是该方法中接收物体的几何形状的存在限制[22]。另一种方法使用点光源和遮挡物轮廓来估计半暗带区域，但首先需要稳定和有效得进行出轮廓检测[23]。

最近，几种方法在第一步骤中通过将三角形阴影体和输出图像采样点投影到网格上，来计算每个输出图像采样点的一组潜在的遮挡三角形，然后在第二步骤中，从输出图像采样点视点估计光源可见性，通过光线跟踪[24，25]或通过光栅化集合[26，27，28，29]。本文的方法使用的4D光栅化可以看作是三角形阴影体的有效投影。在第二步中，本文的方法通过直接计算一行的遮挡掩模来加速。

本文采用更高维度光栅化的方法来估计来矩形面光源的可见性。在运动模糊的渲染中也已经注意到较高维度光栅化的益处，其中通过将时间维度添加到常规2D光栅化[30]中，使得快速移动的三角形在3D中被光栅化。

3．4D光栅化算法

4D光栅化算法将传统的2D投影和光栅化方法推广到4D空间，从而进行高效软阴影渲染。首先，矩形面光源被看作带有两个自由度的点光源。随着点光源的移动，三角形和输出图像采样点在投影面上的投影也在移动。投影移动中经过的区域被称为4D投影。随后，对4D投影进行光栅化，可以得到采样点/三角形对的一个保守集合。之后对这个集合进行运算，从而得到每个采样点处的光源掩码。下面从基本的4D光栅化算法开始，介绍具体的算法流程以及效率优化方法。

3.1．基本的4D光栅化算法

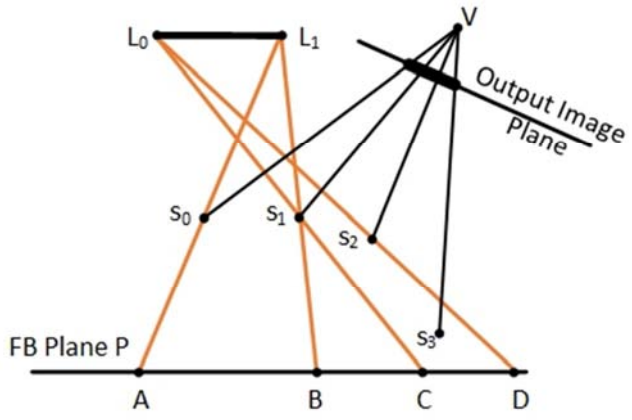

图2 帧缓存平面P平行于矩形光源L_0L_1。输出图像采样点s_1在P上的投影为BC。从A到D为所有采样点4D投影的范围。

首先，考虑计算由单个点光源L投射的硬阴影的问题。该问题可以通过从L处将场景三角形和输出图像采样点投影到一个使用均匀网格划分的图像平面上来解决，这个图像平面被称为帧缓存。如果输出图像采样点s的投影落在帧缓存的像素p处，并且三角形T的投影也覆盖了p，则检查三角形T是否遮挡了从光源L到像素s之间的光线。原始的算法需要考虑所有采样点/三角形对，而大部分采样点和三角形对之间是不存在遮挡关系的，帧缓存算法提供了一个针对光源可见性进行检查的采样点/三角形对的保守集合。这里需要特别说明，帧缓存在这里用于将输出图像采样点与潜在的遮挡三角形配对，而不是生成可见性查询的阴影图。在获得帧缓存后，会精确地计算输出图像采样点和三角形之间的可见性关系。

本文将上面的方法进行扩展，用来渲染柔和的软阴影。首先将矩形面光源用包含两个自由度的点光源进行模拟。随着点光源的移动，三角形和输出图像采样点的投影也会随之移动。投影移动中产生的轨迹称为4D投影。使用4D投影方式对输出图像采样点和三角形投影可以生成帧缓存，从而提供了针对光源可见性进行检查的采样点和三角形对的保守集合。

```
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 * n, 4D
rasterization framebuffer resolution w * 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.v0, L, FB) ∪
15.     Proj4(t.v1, L, FB) ∪ Proj4(t.v2, L, FB)
16.     for each FB pixel p in t.aabb do
17.     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.maskij |= Occlusion(Lij, t, s.xyz)
```

Alogrithm 1用来绘制软阴影，共分为5个主要步骤。第一步，从相机所在位置，对场景进行不带阴影的绘制（第2行）。这一步骤定义了需要计算光照遮挡的输出图像采样点。

第二步为4D光栅化定义帧缓存的参数。帧缓存FB所在平面p被定义为一个与光源平面平行，且距离光源足够远的位置，以保证场景中的物体都位于光源和帧缓存平面之

间(第4行)。实际中平面p的位置是由场景对角线决定的。第5~6行用于定义帧缓存的范围，首先，划分帧缓存的均匀网格与矩形面光源的边平行。帧缓存网格的大小和位置是由输出像素采样点投影的轴向包围盒(AABB)决定的(第5~6行）。

一个三维点A在平面P上，由矩形面光源$L_0L_1L_2L_3$生成的4D投影是由光线AL_0、AL_1、AL_2、AL_3、与平面P的4个交点定义的矩形。帧缓存FB是使用w*h 的均匀网格进行划分。图2所示的2D图，展示了帧缓存FB的构建过程。在这幅图中，输出图像采样点s1的投影是线段BC，帧缓存网格范围是A到D。

步骤3和步骤4用来将输出图像采样点和场景三角形加入帧缓存的像素中，这也隐含地将输出图像采样点和三角形之间进行绑定。输出图像采样点将被加入到其4D投影覆盖的所有FB像素中（第8-11行）。一个三角形将被加入到三个顶点的4D投影的AABB（轴向包围盒）所覆盖的所有FB像素中（第13-17行）。图3中描述了将采样点和三角形加入到FB像素的过程。三角形T的三个顶点对应的4D投影在图中用红色，绿色和蓝色矩形表示。T被加入到这三个矩形的轴向包围盒覆盖的所有FB像素中。矩形Sa是输出图像采样点sa的4D投影。sa会被添加到Sa覆盖的所有像素中。像素p中既有t也有sa，因此(sa,t)将在第5步中进行光照遮挡的计算。

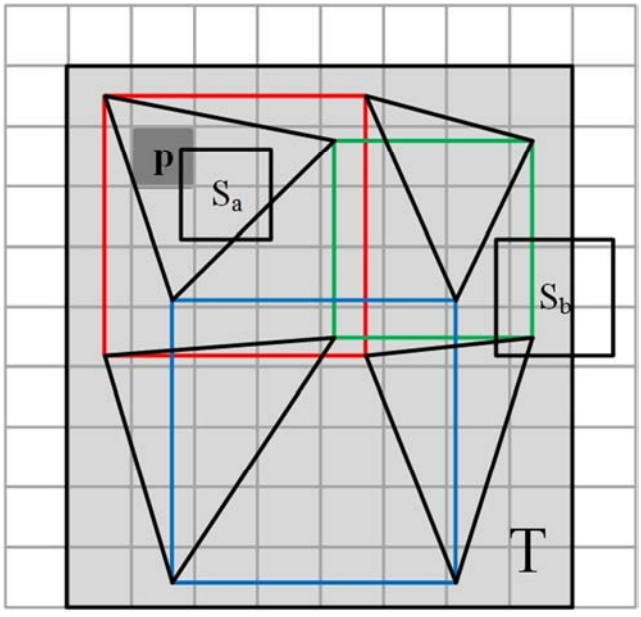

图3 三角形和输出图像采样点的4D投影。三角形t的4D投影为图中灰色矩形T。T是三角形t的三个顶点的4D投影（使用红色，绿色和蓝色框表示）的二维轴向包围盒。输出图像采样点s_a和s_b的投影分别为矩形S_a和S_b。因为S_a和T都覆盖了像素P,因此会产生（s_a,t），在计算s_a的遮挡掩码时，就会考虑三角形t。因为S_b没有完全遮挡住T，因此在计算s_b的遮挡掩码时，不需要考虑t。

步骤5通过处理帧缓存的每个像素，为每个输出图像采样点计算一个遮挡掩码。在像素p中的所有三角形和所有输出图像采样点之间都需要进行一次遮挡判断（第20-21行）。对于每个输出像素采样点和三角形对(s，t)，算法会依次检查m*n个点光源采样点，判断每个点光源采样点L_{ij}到s的光线是否被三角形t所遮挡（第22-24行）。

基本的4D光栅化算法可以从去除重复的采样点\三角形对（3.2节）和使用基于行的光源遮挡判断方法（3.3节）两个方面进行效率优化。

3.2. 去除冗余的输出采样点/三角形对

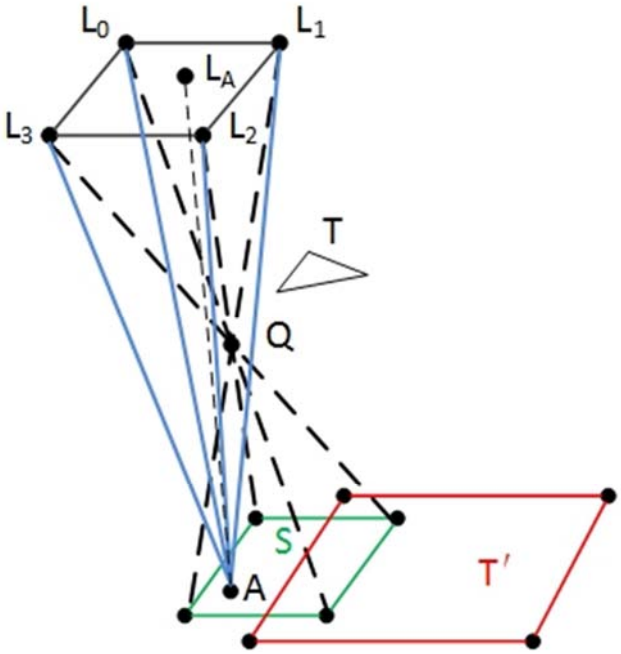

图4 输出图像采样点Q的4D投影S没有完全被三角形t的4D投影T遮挡。因此三角形t不会遮挡从矩形面光源射向输出图像采样点Q的光线。所以在绘制软阴影时，不需要考虑（Q，t）。

在使用基本的4D光栅化进行软阴影绘制的过程中，本文发现只有当采样点的4D投影被三角形的4D投影完全遮挡时，三角形才有可能遮挡住从光源射向输出图像采样点的光线。反过来，如果三角形的4D投影不能完全遮挡住采样点的4D投影，则该采样点/三角形对不需要被考虑。如图3中所示，采样点s_b的投影矩形S_b没有完全被三角形t的4D投影T遮挡，因此(s_b，t)不需要进行遮挡判断。本文使用全覆盖性质对算法1进行效率优化。

全覆盖性质：s为一个输出图像采样点，t是场景中的一个三角形. S是s通过矩形面光源$L_0L_1L_2L_3$在平面P上的4D投影, T是三角形t三个顶点4D投影的AABB(轴向包围盒)。如果 $S \notin T$, 那么三角形t不会遮挡从光源$L_0L_1L_2L_3$ 到s的光线。

证明：

因为：$S \not\subset T$，$S-T \neq \emptyset$（如**图4**所示）。

取S中一点A，且 $A \notin S-T$。

因为$A \notin T$，所以任取$L \in L_0L_1L_2L_3$，有$AL \cap t = \emptyset$。

继而可得：$T \cap$四棱锥$AL_0L_1L_2L_3 = \emptyset$。 (1)

令$Q = s.xyz$， $LA = AQ \cap L_0L_1L_2L_3$。

因为$A \in S$，$LA \in L_0L_1L_2L_3$。

所以$Q \in$四棱锥$AL_0L_1L_2L_3$。

所以四棱锥$QL_0L_1L_2L_3$∈四棱锥$AL_0L_1L_2L_3$。 (2)

根据(1)和(2)可以得出 t∩四棱锥$QL_0L_1L_2L_3 = \emptyset$。

即如果S⊄T，那么t不会遮挡从矩形光源$L_0L_1L_2L_3$射向s的任意一条光线。

证明结束。

```
Algorithm 2 STEP 3 optimization
1:   STEP3: assign output image samples to FB pixels
2:   for each output image sample s in I do
3:   p = Proj(s.xyz, L0, FB)
4:   p.sampleSet = p.sampleSet ∪ {s}
```

首先，将算法1中的步骤3修改为算法2。通过全覆盖性质，可知在计算遮挡时，只需要考虑三角形投影完全覆盖采样点投影的采样点/三角形对。因此，将采样点加入到帧缓存时，只需要在1个帧缓存像素中加入采样点即可。实际中，使用传统的投影方式，从L_0处对采样点进行投影。使用此优化后，每个采样点/三角形对只需要至多被计算一次，从而消除了算法1中同一采样点/三角形对被多次计算遮挡的冗余情况。

其次，使用全覆盖性质还可以对算法1进一步优化。算法1中的步骤5被修改为算法3。在算法3的5-6行中，加入一个判断采样点/三角形对是否满足全覆盖性质的判断，如果不满足，则直接跳过。第一种使用全覆盖性质的优化方法避免了同一个采样点/三角形对被运算多次的情况；而第二种方式减少了一个FB像素中需要计算的采样点/三角形对的个数。第7-17行使用基于移位的整行遮挡掩码计算方式，将在3.3节具体展开。

3.3. 基于行的光源遮挡计算方法

```
Algorithm 3 STEP 5 optimization
1:    STEP 5: computation of light occlusion masks
2:    for each FB pixel p do
3:    for each triangle t in p.triangleSet do
4:    for each output sample s in p.sampleSet do
5:    if Proj4(s.xyz, L, P) ⊄ t.aabb then
6:    continue
7:    P0 = Plane(s.xyz, t.v0, t.v1)
8:    P1 = Plane(s.xyz, t.v1, t.v2)
9:    P2 = Plane(s.xyz, t.v2, t.v0)
10:   for i =1 to m do
11:   if ((Li1Lin ∩ P0) = Lik) = ∅ then
12:   M0 = P0(Li1) > 0 ? 0xFFFF: 0x0000
13:   else
14:   k = Li1Lik / Li1Lin;
15:   temp = (1<<k)-1
16:   M0 = P0(Li1)> 0 ? temp: ~temp
17:   s.maski |= M0 & M1 & M2
```

算法3是针对算法1的第5步进行优化。其中，第7-9行计算由三角形的三条边所决定的三个平面方程。第10-16行用来逐行更新遮挡掩码。

如果当前行的光源采样所在线段$L_{i1}L_{in}$没有与平面P_0相交。那么该行的光源采样点要么全部被遮挡，要么都不被遮挡，这取决于平面P_0的判别式（第13行）。这里，本文以16*16光源采样分辨率为例进行说明，因此可以使用4个16进制的数字表示一行的掩码M_0。如果$L_{i1}L_{in}$和平面P_0（第14行）相交，那么该行的光源采样点将被分成$L_{i1}L_{ik}$和$L_{ik}L_{in}$两部分。一部分是被遮挡的，另外一部分是没有被遮挡的（第15-16行）。图5中例子说明了$L_{ik}L_{in}$被遮挡，而$L_{i1}L_{ik}$没有被遮挡的情况。可以对应到算法中第16行。同样的方式可以为其他两个平面P_1，P_2计算出掩码M_1，M_2。最终将M_0，M_1，M_2作与操作得到当前三角形对当前行的遮挡掩码，并更新采样点对应行的掩码。

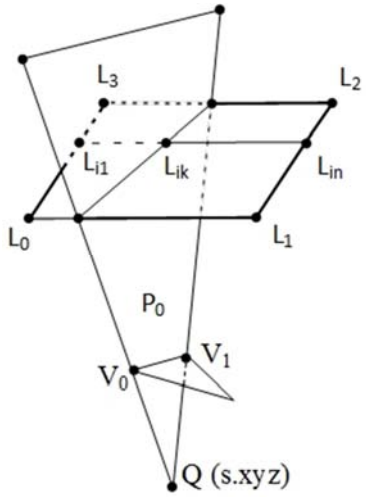

图5 直接为输出像素采样点Q计算第i行的光照遮挡掩码。行线段$L_{i1}L_{in}$和三角形边平面QV_0V_1相交于L_{ik}，因此在k处发生遮挡到不遮挡的转变。

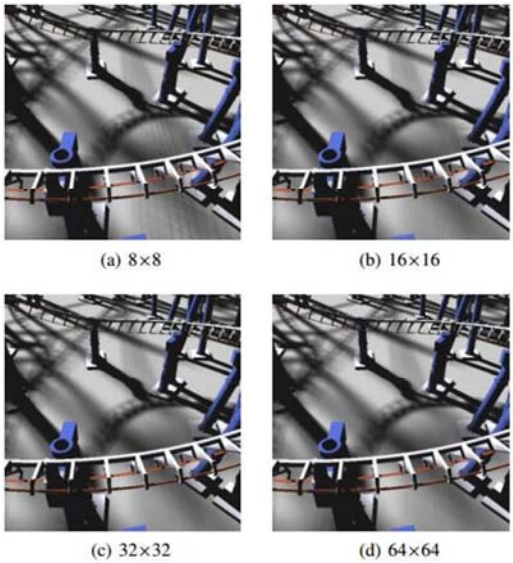

图6 不同光源采样率下的绘制效果。

4. 实验结果与分析

这一节中将从算法的渲染结果，渲染效率以及算法局限性三个方面进行说明。在实验中，使用了多个场景对算

法进行验证：树(367k三角面片)，塔(514k三角片面)，玩具(263k三角片面)，龙(81k三角片面)，过山车(1,096k三角片面)，和教堂(240k三角片面)。渲染结果如图7和图1所示。本文中所有效率统计都是在一台个人电脑上完成，具体配置如下：3.5GHz Intel(R) Core(TM) i7-4770 CPU, 8GB 内存，NVIDIA GeForce GTX 780 显卡。同时本文还使用NVIDIA 提供的Optix 光线追踪引擎[31]作为对比程序。

4.1. 渲染效果

本文的方法能够准确地估计出光源采样点和输出图像采样点之间的可见性。当使用相同的面光源采样分辨率时，使用本文渲染出的图片与使用Optix渲染出的图片逐像素作比较无残差。本文的方法在计算面光源软阴影时，唯一的近似存在于面光源采样分辨率。如图6所示，8*8的面光源分辨率明显不足，16*16的分辨率下存在一定的阴影不连续现象，32*32的分辨率下能够取得较好的阴影效果。当继续提高分辨率时，阴影效果没有明显提升。

图7 本文方法在龙，过山车和教堂场景中的软阴影效果。

4.2. 绘制效率

本文在算法实现中综合使用了可编程渲染管线（步骤一）和CUDA（步骤2-5）。下面将给出不同参数下的绘制效率，如果没有特殊说明，默认参数为：光源采样分辨率32*32、帧缓存的分辨率128*128、输出图像分辨率512*512。

表1中给出了不同输出图像分辨率下所有场景的平均绘制帧率以及相对于光线追踪方法的加速比。本文提出的方法在512*512的输出分辨率下，可以达到实时的绘制帧率，随着输出分辨率的增大，绘制帧率以线性比率下降。所有情况下，本文提出的方法都比光线追踪方法绘制效率更高。步骤1-3花费的时间可以忽略，表2中给出步骤4将三角形加入到帧缓存像素和步骤5为每个输出像素采样点计算遮挡掩码所花费的时间。可以看出，算法中最耗时的步骤是第5步。

表1 不同输出分辨率下的绘制帧率（fps）以及相对于光线追踪方法的加速比。

输出分辨率	绘制帧率（帧/秒）		
	512*512	1024*1024	1280*1280
树	5.2 (18×)	1.4 (15×)	0.8 (11×)
铁塔	10 (35×)	3.4 (21×)	1.8 (20×)
玩具	15 (25×)	3.8 (18×)	2.2 (18×)
龙	20 (15×)	5.5 (9×)	3.5 (9×)
过山车	6.7 (14×)	1.9 (11×)	1.2 (11×)
教堂	13 (10×)	3.2 (6×)	2.2 (6×)

表2 算法主要步骤耗时。

	树	铁塔	玩具	龙	过山车	教堂
STEP 4	27ms	16ms	5ms	5ms	15ms	18ms
STEP 5	160ms	72ms	60ms	36ms	121ms	52ms

表3 优化策略的效果。

场景	冗余的相机采样点数	剔除策略加速比	Bit 移位算法加速比
树	100	3.3x	13x
铁塔	88	4.4x	12x
玩具	40	3.8x	12x
龙	206	4.8x	10x
过山车	50	2.5x	13x
教堂	183	12x	11x

表3中说明了本文提出的3个优化方法对算法效率的影响。算法2是对算法1的步骤3进行优化，避免了将同一个采样点加入到多个帧缓存像素的情况，从而避免了重复运算（第2列所示）。从第3列中可以看出，当在算法第5步中加入提前终止的判断（算法3的第5-6行）后，绘制帧率有了明显的提升。第四列给出使用按行求解遮挡掩码所产生的加速比。

表4给出了本文算法的不同光源分辨率下的平均绘制帧率。算法花费的时间与光源分辨率的行数成线性关系。当光源分辨率增加4倍时，绘制帧率只会减少2倍。而逐个光源采样点进行遮挡掩码计算的方法，当光源分辨率增加4倍时，绘制帧率会减少4倍。

表4 不同光源采样率下的绘制帧率。

面光源分辨率	绘制帧率（帧/秒）			
	8*8	16*16	32*32	64*64
树	12	8.2	5.2	2.2
铁塔	20	15	10	5.4
玩具	30	24	15	6.3
龙	45	30	20	9.1
过山车	15	11	6.7	3.2
教堂	24	18	13	6.7

表5中给出了在不同帧缓存分辨率下的平均帧率。最好的帧缓存分辨率为128*128。当帧缓存的分辨率过低时，有大量的不存在遮挡的采样点/三角形对进入第5步进行遮挡判断，从而导致步骤5的效率下降；当帧缓存的分辨率过高时，将导致步骤4中每个三角形需要加入过多的帧缓存像素，从而导致步骤4的效率下降。

表5 不同帧缓存分辨率下的绘制帧率。

帧缓存分辨率	绘制帧率（帧/秒）			
	256*256	128*128	64*64	32*32
树	4.5	5.2	4.6	3.1
铁塔	8.4	10	9.9	7.8
玩具	15	15	9.2	6.1
龙	18	20	17	12
过山车	6.9	6.7	6.4	5.2
教堂	9.4	13	12	8.7

本文算法最核心的方面是能够高效地将输出图像采样点与对其产生遮挡的三角形之间建立联系，在此我们对这一点进行具体分析。一种理想的建立联系的方式是：当

三角形和输出图像采样点之间存在遮挡关系时，才将三角形赋给输出图像采样点。本文所提出的4D帧缓存方法是一种保守的方法。在算法中为每个帧缓存像素都保存了一个三角形集合和一个输出图像采样点集合。在遮挡判断阶段，需要考虑由两个集合组合出的所有采样点/三角形对（算法3中2-4行）。这些采样点/三角形对中，大部分因为不符合全覆盖性质而提前返回（第5-6行）。剩下的采样点/三角形对不是都存在遮挡关系，因为在计算三角形的4D投影时使用的是三个顶点4D投影的二维AABB。表6中给出了将三角形与输出采样点进行绑定的有效率。对每个场景，分别给出了使用全覆盖性质剔除前和剔除后的有效率。使用全覆盖性质，有效率明显提升，达到了28%-65%之间。

表6 使用全覆盖性质剔除前和剔除后，有效的采样点/三角形对占所有采样点对的比率。

	树	铁塔	玩具	龙	过山车	教堂
Before	15%	14%	23%	26%	8%	10%
After	44%	45%	53%	65%	28%	37%

表1中给出了不同分辨率下本文方法相对于光线追踪方法的加速比，同时，本文还测试了相同帧率下，与光线追踪的效果对比（表7）。为了达到相同的帧率，光线追踪方法只能降低光源采样分辨率，最终导致绘制效果中存在明显的软阴影不连续现象（图8）。

表7 相同帧率下本文方法与光线追踪方法的效果对比。

场景	帧率（帧/秒）	本文方法 光源分辨率	光线跟踪 光源采样数
树	5.2	32x32 = 1,024	72
铁塔	10.0	32x32 = 1,024	36
玩具	15.0	32x32 = 1,024	42
龙	20.0	32x32 = 1,024	72
过山车	6.7	32x32 = 1,024	81
教堂	21.4	32x32 = 1,024	100

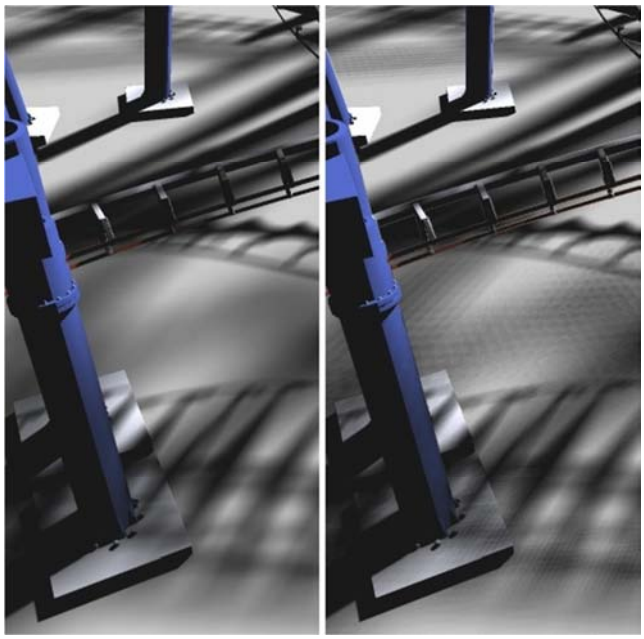

图8 本文方法（左）和光线追踪方法（右）在相同帧率下的效果对比。为了达到相同帧率，光线追踪方法降低了光源采样率。

4.3. 算法的局限性

与所有的精确软阴影算法类似，本文算法的耗时是与输出图像中的像素个数成正比的。当输出图像的宽和高各增加1倍时，帧率会下降4倍。虽然较快的帧率只有输出分辨率在512*512时可以取得，但是本文的算法在进行遮挡运算时，大约有一半的采样点/三角形对是有效的。换句话说，如果使用理想的算法，在进行遮挡运算时，所有的采样点/三角形对都是有效的，那么帧率只会是目前情况的2倍。

本文提出的算法每次可以求解一整行的光源采样点，由于使用了硬件支持的移位操作，因此在64-bit的硬件上，最大支持64*64的光源采样分辨率。 另外一个限制是，帧率是与当前视角下的软阴影边缘有关，这会导致帧率的不稳定。与光线追踪方法类似，本文的方法首先使用投影然后进行光栅化操作计算软阴影，要保证帧率稳定是一项非常大的挑战。使用投影和光栅化操作计算软阴影，可以将多根光线合并计算，从而提高效率，但是使用光线追踪方法逐根光线计算可以更灵活地调整光线个数，以适应不同帧率。

5. 结论与展望

本文提出一种能够以可交互效率,进行复杂场景下精确软阴影计算的方法。本文算法使用4D投影和遮挡三角形的光栅化操作求解光源处的可见性。并且该算法简单、通用，既可以在未来更加灵活的GPU硬件中实现，也可以用于大部分应用场景。

本文的算法使用成百上千的点光源对面光源进行采样，从而解决面光源软阴影问题。然而，该算法中光源采样点的位置和排布都是规则的，将来可以设计一种带有扰动的采样方案。同时，未来需要考虑将该算法应用在拥有多个面光源和点光源的场景中。

致谢

本文为国家自然科学基金《面向实时逼真绘制的非传统相机模型研究》（61272349）的阶段性成果之一。

参考文献

[1] Haines E. Real-time shadows [J]. Silicon Graphics Inc, 2011.

[2] Hasenfratz J M, Lapierre M, Holzschuch N, et al. A Survey of Real-time Soft Shadows Algorithms [J]. Computer Graphics Forum, 2003, 22(4):753－774.

[3] Akenine-Möller T, Assarsson U. Approximate soft shadows on arbitrary surfaces using penumbra wedges [C]// Eurographics Workshop on Rendering. Eurographics Association, 2002:297-306.

[4] Brabec S, Seidel H P, Mccool M, et al. Single Sample Soft Shadows Using Depth Maps [J]. Graphics Interface, 2002(2002).

[5] Arvo J, Hirvikorpi M, Tyystjärvi J. Approximate Soft Shadows with an Image-Space Flood-Fill Algorithm [J]. Computer Graphics Forum, 2004, 23(3):271-279.

[6] Fernando R. Percentage-closer soft shadows[C]// ACM SIGGRAPH 2005 Sketches. ACM, 2005:35.

[7] Mohammadbagher M, Kautz J, Holzschuch N, et al. Screen-Space Percentage-Closer Soft Shadows [J]. Acm Siggraph Posters, 2010:1-1.

[8] Gaël Guennebaud, Paulin M. Real-time soft shadow mapping by backprojection [C]// Eurographics Symposium on Rendering Techniques, Nicosia, Cyprus. DBLP, 2006:227-234.

[9] Guennebaud G, Barthe L, Paulin M. High-Quality Adaptive Soft Shadow Mapping[C]// Computer Graphics Forum. 2007:525 - 533.

[10] Schwarz M, Stamminger M. Microquad Soft Shadow Mapping Revisited[C]// Eurographics 2008 Annex to the Conference Proceedings Short Papers. 2008:295 - 298.

[11] Yang B, Feng J, Guennebaud G, et al. Packet-based hierarchal soft shadow mapping[J]. Computer Graphics Forum, 2009, 28(4):1121-1130.

[12] Shen L, Feng J, Yang B. Exponential Soft Shadow Mapping[J]. Computer Graphics Forum, 2013, 32(4):107-116.

[13] Selgrad K, Dachsbacher C, Meyer Q, et al. Filtering Multi-Layer Shadow Maps for Accurate Soft Shadows[J]. Computer Graphics Forum, 2014, 34(1):205 - 215.

[14] Liktor G, Spassov S, Ckl G, et al. Stochastic Soft Shadow Mapping [J]. Computer Graphics Forum, 2015, 34(4):1-11.

[15] Whitted T. An improved illumination model for shaded display [J]. Communications of the ACM, 1980, 23(6):343-349.

[16] Mehta S U, Wang B, Ramamoorthi R. Axis-aligned filtering for interactive sampled soft shadows[J]. Acm Transactions on Graphics, 2012, 31(6):163.

[17] Yan L Q, Mehta S U, Ramamoorthi R, et al. Fast 4D Sheared Filtering for Interactive Rendering of Distribution Effects[J]. Acm Transactions on Graphics, 2015, 35(1):1-13.

[18] Samuli L, Timo A. Hierarchical Penumbra Casting[C]// Computer Graphics Forum. 2005:313 - 322.

[19] Laine S, Aila T, Assarsson U, et al. Soft Shadow Volumes for Ray Tracing[J]. Acm Transactions on Graphics, 2005, 24(3):1156-1165.

[20] Forest V, Barthe L, Paulin M. Accurate Shadows by Depth Complexity Sampling [J]. Computer Graphics Forum, 2008, 27(27):663-674.

[21] Mora F, Aveneau L, Apostu O, et al. Lazy Visibility Evaluation for Exact Soft Shadows[J]. Computer Graphics Forum, 2012, 31(1):132-145.

[22] Eisemann E, Décoret X. Visibility Sampling on GPU and Applications [J]. Computer Graphics Forum, 2007, 26(3):535-544.

[23] Johnson G S, Hunt W A, Hux A, et al. Soft irregular shadow mapping: fast, high-quality, and robust soft shadows[C]// Symposium on Interactive 3d Graphics, Si3d 2009, February 27 - March 1, 2009, Boston, Massachusetts, Usa. DBLP, 2009:57-66.

[24] Benthin C, Wald I. Efficient ray traced soft shadows using multi-frusta tracing[C]// ACM Siggraph/eurographics Conference on High PERFORMANCE Graphics 2009, New Orleans, Louisiana, Usa, August. DBLP, 2009:135-144.

[25] Nabata K, Iwasaki K, Dobashi Y, et al. Efficient divide-and-conquer ray tracing using ray sampling[C]// High-Performance Graphics Conference. 2013:129-135.

[26] Aila T, Laine S. Alias-Free Shadow Maps.[C]// Eurographics Workshop on Rendering Techniques, Norköping, Sweden, June. DBLP, 2004:161-166.

[27] Sintorn E, Eisemann E, Assarsson U. Sample Based Visibility for Soft Shadows using Alias-free Shadow Maps[J]. Computer Graphics Forum, 2008, 27(4):1285-1292.

[28] Wang L, Zhou S, Ke W, et al. GEARS: A General and Efficient Algorithm for Rendering Shadows[C]// 2014:264-275.

[29] Lecocq P, Gautron P, Marvie J E, et al. Sub-pixel shadow mapping[C]// ACM SIGGRAPH. ACM, 2013:1.

[30] Akenine-M, Ller T, Munkberg J, et al. Stochastic rasterization using time-continuous triangles[C]// ACM Siggraph/eurographics Conference on Graphics Hardware 2007, San Diego, California, Usa, August. DBLP, 2007:7-16.

[31] NVIDIA: Nvidia optix ray tracing engine.http://developer.nvidia.com/optix (2016)