分类号 密级

中国地质大学（北京）

本 科 毕 业 设 计

**题 目** 紧凑的体素化预制阴影

算法的优化与实现

**英文题目**

**学生姓名** 陈雨航  **学 号** 1004171217

**学 院** 信息工程学院 **专 业** 计算机科学与技术

**指导教师** 季晓慧  **职 称**  副教授

**2020 年 5 月 9 日**

# 摘 要

# ABSTRACT

# 目录

[摘 要 2](#_Toc14277)

[ABSTRACT 3](#_Toc24917)

[目录 4](#_Toc80)

[1 引言 5](#_Toc32200)

[1.1 研究背景和意义 5](#_Toc11730)

[1.2 研究现状和问题分析 5](#_Toc15698)

[1.3 体素阴影算法 6](#_Toc9519)

[1.4 本文工作 8](#_Toc2798)

[2 基于ShadowMap捕捉阴影信息 8](#_Toc8282)

[2.1 阴影产生原理 9](#_Toc28685)

[2.2 ShadowMap实现 9](#_Toc25918)

[2.2.1 坐标系转换 9](#_Toc23612)

[2.2.2 深度贴图 10](#_Toc10557)

[2.2.3 深度测试 10](#_Toc19257)

[3 体素八叉树 10](#_Toc11642)

# 1 引言

## 研究背景和意义

随着计算机与互联网的普及，虚拟仿真和游戏产业正在快速发展，对于大型开放场景的需求正在逐渐增加。而在实际工程中，大型场景的阴影实现方案是一个难以回避的问题。传统的阴影算法面对大场景情景，都会出现性能不佳、效果不好等多种问题。

目前而言，在大型场景的阴影实现方案中，级联阴影算法（Cascaded Shadow Maps ，后文简称CSM）被应用的最为广泛。该方法将按照区域相对相机的距离分配纹理空间，提供不同分辨率的深度纹理。CSM技术将相机的视锥分割成多个部分，分别提供对应的深度纹理。相较于基础的ShadowMap技术，使用CSM在大型场景中模拟太阳等远距离平行光源的阴影，无论是性能还是效果都更为优秀。CSM技术的核心思想便是“因材施教”，即在近距离区域使用高分辨率纹理，在远距离区域使用低精度纹理。由于近距离区域在相机视野中的占比远高于远距离区域，因此该技术比起“一视同仁”的ShadowMap技术更能适应大型场景的阴影需求。

然而，CSM技术也并非与大型场景天生一对。作为基础的ShadowMap技术的改进，使用CSM技术仍旧需要大量高分辨率的阴影贴图，所需的空间成本仍旧很高。除此之外，为了弥补级联之间的接缝，CSM技术需要对不同级联做混合处理，存在重复绘制和多次采样的问题。

由于CSM技术在实际工程中存在局限性，工业界急需占用空间更小、采样时间更少、阴影效果优秀的新阴影技术。而来自于查尔姆斯理工大学的Erik团队提出的采用八叉树数据结构管理体素阴影的方案成为了照亮大型场景黑暗的一盏新的明灯。该方法的核心思想是将场景进行体素化分割，并使用一棵八叉树对所有体素的阴影信息进行自上而下的管理。由于每个体素所需管理的信息十分单一，只有1比特位的阴影信息，并且八叉树的整体结构十分稳定，所以将出现大量具有相同结构、相同信息的子树。该方法通过排序的方法挑选重复子树进行复用，有效压缩了阴影信息的存储空间。结合该方法的其他优化方案，该体素八叉树能够进行十分高比率的压缩，且解压读取信息也十分便利。因此，实现该体素阴影算法进行验证并进行进一步的优化和改进将成为一项十分有意义且具有挑战性的工作。

## 研究现状和问题分析

随着阴影技术的发展，在大型开放场景中生成高质量的阴影仍是一个巨大的挑战。该课题作为图形学领域的重要研究方向受到了广泛的关注。目前在工业界，解决实时阴影生成有两种主要方法，一是阴影贴图方法，二是稀疏的体素八叉树方法（后文简称SVO）。阴影贴图方法已经经过了长期的发展和演变，但随着虚拟场景规模的不断增大，阴影贴图的大小也呈急剧增大，在实际工程中经常出现性能较差和效果不佳的问题；而SVO方法直到2014年才由Erik团队引入到实时阴影问题中，该方法的发展围绕着SVO树的压缩存储进行，工作重点是将SVO无损压缩为稀疏体素有向无环图（后文简称SVDAG），出现了多种不同方向的改良技术并且仍存在提升空间。SVDAG的思想正被不断推广到三维空间中其他问题的解决当中。

现如今多数工程师决定使用Williams于1978年提出的阴影贴图（ShadowMap）方法及其改进方法处理阴影问题，该方法属于一种支持动态的阴影技术，在实际生产中得到了普及，并在某种形式上是得到了所有主要游戏引擎的支持。一种常见的方案是在纹理贴图（通常称为光照贴图或者阴影贴图）中存储预先计算的可见性信息，这些信息可以在阴影处理过程中立即查询。光照贴图可以几乎零成本的在阴影处提供光源可见性信息，但依旧存在一些无法忽视的问题：

1、只能用来评估静态几何体表面的阴影，而动态几何体必须使用一些其他技术被静态环境遮蔽；

2、对于静态几何体，必须经常性的单独设置UV，这既困难又繁琐；

3、即使使用有损图像压缩技术，大量的贴图所占用的内存依然是难以忍受的。

针对于传统ShadowMap方法的不足，ENGEL团队提出了级联阴影贴图算法（Cascaded Shadow Maps，简称CSM）。CSM方法依据对象与观察者之间的距离提供不同分辨率的深度纹理来弥补基础的ShadowMap方法的不足。CSM将相机的视锥体分割成若干部分，并为分割的每一部分生成独立的深度纹理。这为所有视图样本提供了大致统一的阴影贴图分辨率。然而，即使使用最先进的剔除技术，可能还是需要多次重新渲染场景的某些部分，这仍旧对性能有着相当大的负面影响。此外，该方法虽有助于减少重采样误差，但即使阴影贴图的采样率和主视图的采样率匹配良好，也仍然不能解决初始采样误差。CSM减少了因使用单个阴影贴图而引起的欠采样和过采样问题。然而为了完全隐藏分区之间的锯齿和可见性过渡多而引起的瑕疵，仍然需要大量的高分辨率阴影贴图。目前而言，即使使用有损的图像压缩技术，这些贴图所占用的内存空间依旧是庞大而不可忽略的。

随着ShadowMap及其改进技术的推广，该方法的局限性逐渐暴露出来。针对ShadowMap类技术存在的问题，许多研究团队提出了各具特色的改进方案。Rasmusson团队提出了一种专门针对光照贴图的硬件加速压缩方案，并提供了技术标准的概述。Hoppe团队则建议使用空间相干数据（如光照贴图）的层次表示来提高压缩率，而代价是查找变得更昂贵了。然而即使使用有损数据压缩方案，捕获高频率的可见性变化，仍然需要占用大量内存以满足将光照贴图应用在大场景中的需求。Green团队则是建议对表面上的二进制可见度通过距离场进行编码，该方法得到的阴影数据通常可以从低分辨率贴图中很好地重建，但却无法捕捉到细小的阴影或复杂的阴影。

ShadowMap技术都需要对所有几何体进行明确的UV参数化，并且只能准确存储所指定的静态表面的光可见性信息，存在很大的局限性，难以推广。除此之外，兼具性能和阴影效果的ShadowMap技术也迟迟没能出现，这极大拖累了ShadowMap技术在大型场景中的应用。

## 体素阴影算法

在ShadowMap及其衍生技术迅速发展的同时，另一部分研究者关注到：对于绝大多数需要实时构造阴影的场景而言，基本都包含大量的静态几何体与一个或多个的静态平行光源。而选择静态技术预先计算这部分的阴影，相较于使用完全动态的阴影技术，能够更加快速的获得更高质量的阴影效果。

来自于查尔姆斯理工大学的Erik团队决定采用八叉树的数据结构管理体素化的阴影信息。体素概念被广泛应用在三维空间信息的管理当中，而密集网格是体素最简单的表示形式，该形式简单直观，但缺点在于所有体素信息都被一视同仁的存储下来。每当体素网格的分辨率提升到原来分辨率的两倍，密集网格存储形式所需的存储空间便上升到了原先的八倍之多。幸运的是，计算机图形学中的体素数据集往往表现出大量的稀疏性，这为更高效的数据编码提供了可能性。三维体素数据的稀疏八叉树表示法（Sparse Voxel Octree，简称SVO）由Meagher于1982年提出。八叉树的根对应于整个体素网格的体积，八个子节点对应于将每个网格沿三个维度等分成的八个子网格，体素的划分沿树伸展方向递归进行，直到达到所需的分辨率。

Erik团队不仅将SVO引入到了阴影的管理中，还对八叉树的数据结构进行了优化和压缩，将SVO转换成一种紧凑的有向无环图（Directed Acyclic Graph，简称DAG）。该方法的基本算法思想是创建体素化的阴影空间并以八叉树的形式进行数据存储和管理。在创建SVO后，通过合并公共子树来压缩SVO，从而将SVO转为存储上更为紧凑的体素有向无环图（Sparse Voxel Directed Acyclic Graph，简称SVDAG）。由于该方法中SVO仅需管理体素的光可见性，因此SVO会存在大量的公共子树，所以对公共子树的合并能够有效提高最终的压缩率，减少存储空间的占用。经证明，该优化方法使得数据仅需沿着包围阴影空间的表面进行存储，因此SVO的大小很大程度上取决于这些表面的表面积。除此之外该团队还提出对完全封闭空间内无定义体素的可见性依据其在树上合并后对树大小的影响做动态分配。值得一提的是，在文中的测试阶段，该团队发现这种动态分配无定义体素的方法不仅是一种存储优化，也能大幅加快运行时的性能，因为针对具体体素的可见性查询往往因此在树的更高层级结束，这使得信息的查询更为迅速。

Erik团队的研究使SVO成功应用在了三维空间阴影的管理与生成上。由于SVO已被广泛应用于三维空间的多种属性的管理上了，围绕SVO也衍生出了一系列的优化结构。针对于SVO的优化，Erik团队提出了一系列无损压缩方案用于提高SVO的存储性能和运行效率，而Laan团队则是提出了另一种有损压缩方案，以一定量的误差为代价，换取更高的压缩率。Laan团队从概率和实践中进行论断：空间中的绝大多数子树很少被重复引用，通常只被引用一次。在大多数场景中，这些不常被引用的子树又彼此间具有高度的相似性。Laan团队对这些相似的子树进行聚类，并用单个代表代替聚类中的所有子树。该方法在允许一定压缩误差的前提下，进一步对SVDAG进行了压缩。该方法的结果被称为有损系数体素有向无环图（Lossy Sparse Voxel DAG，简称LSVDAG）。为防止误差的无限增长，Laan团队的方法会对相似性的适用层级进行限制，避免对于叶节点基于相似性的修改，因为叶节点总共只有256（8个子空间对应的阴影情况数）种可能类型，对其进行修改会造成整体上的较大误差。经测试，在San Miguel场景中，该方法仅通过修改了0.57%的体素，便将最终场景存储空间从1228MB压缩到了938MB，说明在付出较小的误差代价的情况下，对SVO的存储空间的进一步压缩是切实可行的。

Villanueva团队则通过发现体素分布的反射对称性，提出了一种被称为对称感知的紧凑的体素有向无环图（简称SSVDAG）的新的无损压缩方法，该方法利用体素在空间分布的对称性，进一步对SVDAG进行压缩。Villanueva团队指出：体素在空间中的分布规律可用一部分“原子”表示，这些“原子”的种类是有限的，各种“原子”通过反复的对称变换组合成了复杂的体素分布。因此，Villanueva团队的方法致力于利用相似性变换合并相同“原子”的子树，并利用可变比特位编码存储子指针，实现对SVDAG的无损压缩。Villanueva团队的方法实际上是从寻找对称子树出发，重点寻找潜在的有反射对称性的子树并对其进行合并，设置公共引用，从而压缩原始SVO的存储空间。

无论是Erik团队，还是Laan团队和Villanueva团队的方法都建立在SVO只管理单一属性这一前提之上。失去这一前提，三个团队的SVO压缩优化方法都将大打折扣，因为无论是公共子树、相似子树，对称子树都只有在单一属性前提下会出现高度重复，对其进行公共引用才能产生足够高的压缩率。Dado团队则致力于探索一种新的技术，将多属性SVO的高度压缩变成可能。Dado团队提出了一种基于影响SVDAG数据结构中指针大小的无损压缩方法（简称ESVDAG）。该方法提供了一种压缩任意数据，如颜色、法线或反射信息的方法。它通过一种新的映射方案将几何体数据和体素属性数据解耦，并应用DAG原理进行拓扑编码，同时对体素属性使用基于颜色的压缩方式，从而大大减少了内存消耗，该方法被证明是一种对SVDAG的无损压缩技术。

从目前的研究状况来看，Erik团队引入的SVO方式处理阴影具有很好的发展和应用前景。在SVO的优化使用问题上，Erik团队提出了以合并公共子树为主的无损压缩方法，而Laan团队则是提出了以合并相似子树为主的有损压缩方法，Villanueva团队则是提出了合并对称子树的无损压缩方案，而Dado团队则是关注于多属性树的压缩，提出了一种新的无损压缩方案。这些优化方法都从各自不同的角度挖掘了SVO存储数据的冗余性质，并针对这些冗余做出了不同角度和不同适用范围的优化与改进。而现有的优化技术仍存在一定的优化空间，也依然存在着通过交叉使用不同的优化技术进一步提升SVO性能的可能。

## 本文工作

在回顾了三维空间中的阴影实现技术和体素八叉树的实现原理与优化方案后，本文将重点论述本人基于ShadowMap理论和体素阴影八叉树理论对紧凑的体素化预制阴影算法的Unity实现，并在此基础上结合现有技术对原有算法做出的进一步优化。主要内容如下：

1. 基于ShadowMap技术，将三维场景的点的世界坐标转换到光源空间坐标，并计算该点的深度，与捕捉的场景深度图上对应UV的像素信息做对比，得到点的阴影信息。本人将该方法的计算部分从Unity Shader提取到C#脚本中，从而提取阴影信息进行后续的建树工作。
2. 构建与世界坐标解耦的体素八叉树，用于存储阴影信息。通过将世界坐标与体素节点解耦，保证树上每个体素节点都只保存阴影信息和向子节点的链接。这使得能够对树的结构进行压缩。由于体素节点与世界坐标已经解耦，每个体素节点与世界坐标中的点将不是一一对应的关系，因此将不能直接通过体素节点找到其在世界中对应的位置。为了解决该问题，本人将对八叉树的孩子顺序进行编码，将点在树空间下的坐标信息存储在树的结构中。根据体素阴影八叉树理论，本人实现了自顶向下的八叉树建树过程。由于递归深度将受栈空间大小的限制，本人进一步将递归过程分解成多个步骤转换为迭代过程。
3. 结合现有技术，对体素八叉树的冗余结构进行剔除和复用，实现了对紧凑的体素化预制阴影算法的进一步优化。
4. 提出了一种持久化存储体素有向无环图的方法。该方法依据SVDAG中同一节点不会同时存储阴影信息和子节点信息的事实，使用标记为将结果分类存储在一张32位的位图中。使用时将该位图加载并将信息解码进结构化缓冲区中供Shader使用；
5. 引入了Bias和PCF方法解决ShadowMap引入的阴影表现上的问题。

# 基于ShadowMap捕捉阴影信息

由于体素阴影方法本身无法获取场景的阴影信息，因此需通过ShadowMap从光源视角渲染一张高精度且恰好覆盖整个场景的深度贴图。本章着重解决两个问题：一是如何通过计算自动设置深度相机的坐标和旋转，二是如何通过深度贴图获取场景中指定世界坐标点的阴影信息。

## 2.1 阴影产生原理

在现实世界中，阴影可以被理解为光所照射不到的区域。如图2.1所示，在平行光（太阳光）照射下，因为正方形物体的阻挡，A、D区域均在光线照射范围内，而B、C区域均处于阴影之中。因为光沿直线传播的特性，一旦光线遭到阻挡，其无法到达的地方就会产生阴影。也就是说，从平行光源视角看不到的区域均为阴影区域。

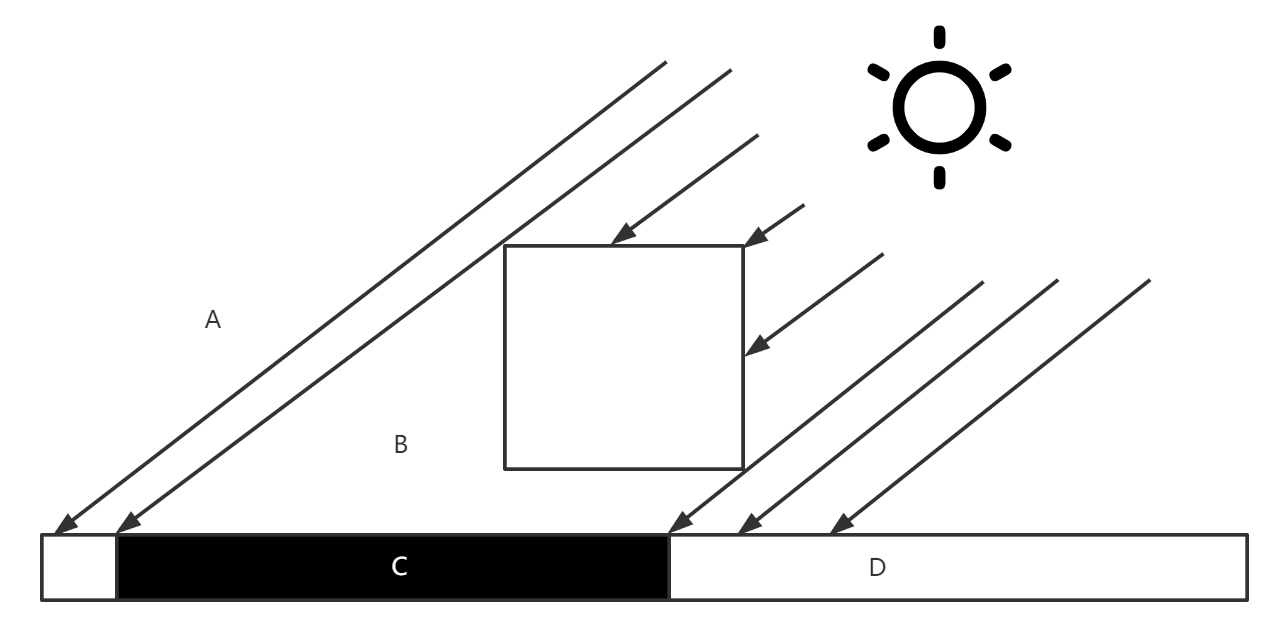


图2-1 阴影产生原理

根据现实世界中阴影的生成原理，可以将阴影生成问题转化成光源视角下的可见性问题。在图形学中，可见性判定通常与深度测试是绑定的。深度测试首先需要计算场景中物体距离光源的距离，并记录各点位置到光源距离的最小值，从而生成一张深度贴图。获取深度贴图后，只需将所需测试的点到光源的距离同深度图上对应点的深度作比较，深度更小的情况下，该点可见，否则该点不可见。

因此只需将场景中的点转换到光源坐标下进行深度测试，即可判断该点的深度信息。

## 2.2 ShadowMap实现

基于ShadowMap的阴影生成原理，本算法中将阴影生成分成三个步骤：

1. 从光源视角渲染场景，生成整个场景的深度贴图并作缓存；
2. 在计算场景中点的阴影信息时进行坐标系转换，将其从世界坐标系变换到光源坐标系，并计算到光源的深度，与深度贴图中的对应点进行对比，得到阴影信息；
3. 结合场景中所有体素中心点的阴影信息生成体素化的八叉树管理阴影。

### 2.2.1 坐标系转换

根据上一节所说，当已知场景中一个点的坐标为x，需首先将其转换到相机坐标系，可设转换矩阵为V。其次需要将p从相机坐标系转换到光源坐标系，转换矩阵设置为P。经过上述两步，原始的世界坐标中xyz分量的取值范围转换到[-1,1]，而希望通过该坐标找到点在深度贴图上的位置则需要进一步将UV范围限制在[0,1]的范围内，可令转换矩阵为C，C通常如公式2-1所示。

（2-1）

因此如公式2-2所示，场景中坐标为x的点在光源空间的坐标为：

（2-2）

通过该p的x分量和y分量可以唯一确定点在深度贴图上对应点的位置，进而完成了场景中任意点的世界坐标与深度贴图上UV坐标的转换，保证了深度测试的顺利进行。

### 2.2.2 深度贴图

深度贴图（Depth Map）被用于存储场景中所有物体位置到光源距离的最小值。深度贴图的尺寸直接影响阴影提取的精度。如果深度贴图尺寸过小，最终的结果难免会存在大量锯齿，出现走样；而如果阴影贴图尺寸过大，则会占用不必要的内存。

由于本算法中深度贴图只在预计算中构建体素八叉树时使用，而在后续使用阴影结果时都已经从内存中释放。因此，本算法可采用超高精度的深度贴图对场景深度进行采样，在避免初始采样误差的同时，而不必担心算法实时运行时的性能问题。

### 2.2.3 深度测试

建立体素化八叉树管理阴影的核心是获取体素中心点的阴影信息。在本算法中，在自顶向下的建树过程中，将对所有叶子节点的体素中心点执行深度测试。

在2.2.1节和2.2.2节的基础上，对场景中世界坐标为x的点执行深度测试，只需要将坐标x带入公式（2-2），得到点在光源空间的坐标。令表示点在深度贴图上的UV坐标，可取uv处深度贴图中RGBA颜色解码出的深度值为d，而此时该点与光源的距离则等于z。比较深度值z与深度贴图上的值d，若则表示该点对光源不可见，即该点处于阴影中，否则该点对光源可见，即不处于阴影当中。

# 3 紧凑的体素化预制阴影算法

有了第2节的铺垫，该算法的数据来源得到了保障。紧凑的体素化预制阴影算法的核心目标是构建一个高效的数据结构对所获取的阴影信息进行管理和压缩。本节将着重描述本人定义与场景解耦的体素化的阴影八叉树的方法和使用迭代方法自顶向下的体素化的阴影八叉树的构建方法。在本节的后半部分，将重点描述本人在紧凑的体素化预制阴影算法上的优化工作，并通过实际场景和数据验证优化效果。

## 3.1 体素化的阴影八叉树

本节将围绕体素化的阴影八叉树的数据结构定义与自顶向下的八叉树构建方案描述本人对体素化的阴影八叉树的实现。

### 3.1.1 数据结构