分类号　　密级

中国地质大学（北京）

本 科 毕 业 论 文

题　　目 紧凑的体素化预制阴影算法的优化与实现

英文题目 The Optimization and realization of Compact Precomputed Voxelized Shadows Algorithm

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

2021 年 5 月

摘要

虽然计算机技术在快速发展，但是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，一是采用阴影贴图方法，二是采用稀疏的体素八叉树方法。阴影贴图方法已经发展的十分成熟，但在大型场景的阴影生成中仍面临着许多的问题。在这一背景下，稀疏的体素八叉树方法开始受到关注。

基于上述研究背景，本文主要研究如何对紧凑的体素化的预制阴影算法进行实现和优化。具体而言，本人的主要工作如下：

（1） 基于 ShadowMap 技术，将阴影信息的获取同光源空间下的深度测试进行关联。本人将三维场景的点的世界坐标转换到光源空间坐标，并对场景的深度贴图进行捕捉，提供一套能在此背景下高效使用的深度测试方法。该方法将作为后续算法工作的数据基础。

（2） 构建与世界坐标解耦的体素化的阴影八叉树。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，保证了对树的结构进行压缩的可行性。为了确定体素位置的位置信息，本人通过 Morton order 对树节点的八个子节点进行编码，实现了点在树空间下的坐标信息与深度优先搜索顺序的相互转换。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，本人对体素化的阴影八叉树中的重复结构进行剔除和复用，完成了对紧凑的体素化预制阴影算法的实现和优化。

（4） 提出了一种序列化体素化的阴影有向无环图的方法。该方法依据体素化的阴影有向无环图中同一节点不会同时存储阴影信息和子节点链接的事实，使用标记位将结果存储在一张 32 位的位图中。本人将该位图加载并将信息解码进结构化缓冲区中通过 Shader 读取，完成了预置阴影的要求。

（5） 引入了一类解决阴影效果不佳的阴影优化算法，即 Shadow bias 和 Percentage Closer Filtering，保证了算法在实际工程中的可用性。

关键词：阴影，体素化，预计算

Abstract

Although computer technology is developing rapidly, generating high-quality shadows in large open scenes is still a huge challenge. This subject has received extensive attention as an important research direction in Compute Graphics. At present, in the industry, there are two main methods to solve real-time shadow generation, one is to use the shadow map method, and the other is to use the sparse voxel octree method. The shadow map method has been developed very mature, but it still faces many problems when generating the shadow of a large scene. Thus, the sparse voxel octree method began to attract attention.

Based on the above research background, this article mainly studies how to implement and optimize the Compact Precomputed Voxelized Shadows Algorithm. Specifically, my main tasks are as follows:

(1) Based on the ShadowMap technology, the acquisition of shadow information is associated with the depth test in the light source space. I convert the world coordinates of the points of the 3D scene to the light source space coordinates, and capture the depth map of the scene, providing a set of depth testing methods that can be used efficiently in this context. This method will serve as the data basis for subsequent algorithm work.

(2) Construct a voxelized shadow octree decoupled from the world coordinate. By decoupling the world coordinates from the voxel nodes, it is ensured that each voxel node on the tree only relates to its shadow information or links to its child nodes, which ensures the feasibility of compressing the structure of the tree. Since the voxel node has been decoupled from the world coordinate, each voxel node will not have a one-to-one correspondence with the point in the world coordinate, so it will not be possible to find its corresponding position in the world directly through the voxel node. In order to solve this problem, I used Morton order to encode the eight child nodes of the tree node, and realized the mutual conversion between the coordinate information of the point in the tree space and the depth-first search order. According to the voxel shadow octree

theory, I realized the top-down octree building process. Since the depth of recursion will be limited by the size of the stack space, I further decompose the recursive process into multiple steps and convert it into an iterative process.

(3) To the best of my ability, I eliminated and reused the repetitive structure in the voxelized shadow octree, and completed the realization and optimization of the Compact Precomputed Voxelized Shadows Algorithm.

(4) A method of serializing voxelized shadow directed acyclic graphs is proposed. Based on the fact that the same node in the voxelized shadow directed acyclic graph won't store shadow information and child node links at the same time, the result is stored in a 32-bit bitmap using flag bits. I loaded the bitmap and decoded the information into the structured buffer to read it through the Shader, which fulfilled the requirement of precompute shadows.

(5) A type of shadow optimization algorithm that solves the poor shadow effect is introduced, namely Shadow bias and Percentage Closer Filtering, to ensure the usability of the algorithm in actual projects.

Key words: shadows, voxelization, precomputed

目 录

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难以忍受。

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，为后续的建树工作提供了数据基础。

（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。

图 1-1 小型的简单几何体场景

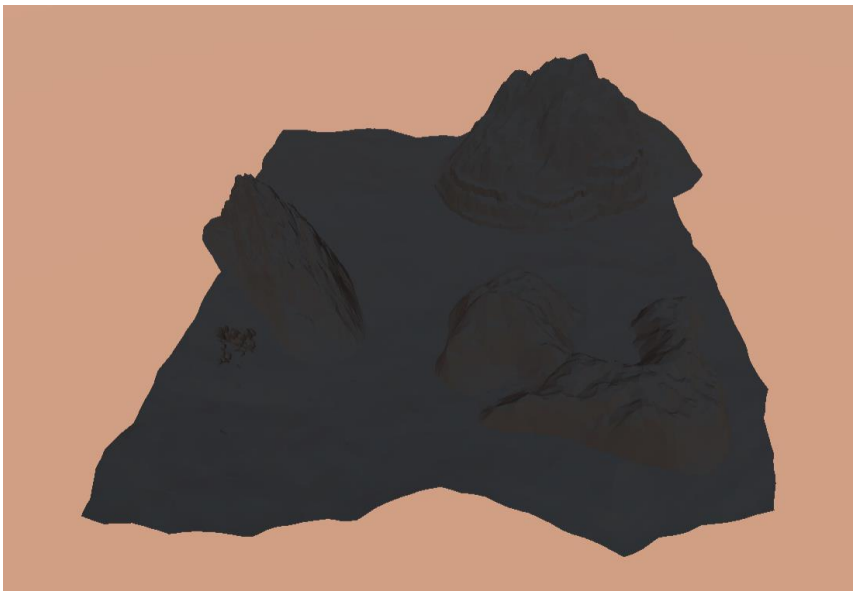

图 1-2 大型的 Low Poly 环境场景

由于本文的算法结果需要有一定的实用价值，因此本文的数据展示将通过两个不同场景进行。如图 1-1 所示第一个场景为小型的简单几何体场景；如图 1-2 所示，第二个场景为大型的 Low Poly 环境场景。

2 基于 ShadowMap 的阴影信息捕捉

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。

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。

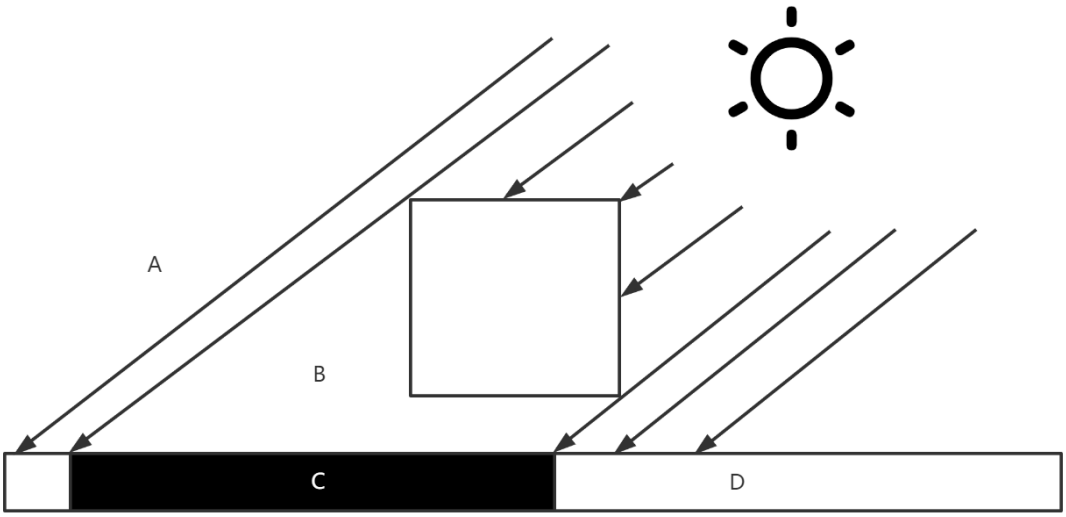

图 2-1 阴影产生原理

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。

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。

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 x、y、z 方向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。

$$C = \begin{bmatrix} 0.5 & & & 0.5 \\ & 0.5 & & 0.5 \\ & & 1 & \\ & & & 1 \end{bmatrix} \tag{2-1}$$

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$\mathrm{p} = (\mathrm{x}, \mathrm{y}, \mathrm{z})$：

$$p = C * P * V * x \tag{2-2}$$

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。

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。

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$p = (x, y, z)$。令$uv = (x, y)$表示点在深度贴图上的UV坐标，可取深度贴图中坐标为uv的像素的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$z > d$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。

小结

本章完成了对场景深度信息的捕捉，并将深度信息写入在深度贴图中。除此之外，本章根据 ShadowMap 算法的基本思想，将场景阴影信息的判断转化成了深度测试，为后文中正式展开的紧凑的体素化预制阴影算法提供了数据保障。

3 紧凑的体素化预制阴影算法

有了第 2 章的铺垫，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。

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。

数据结构

由于本算法的优化目标之一是将体素化的八叉树中的重复子树进行复用，因此该八叉树的树形结构中每个体素节点都应只与阴影信息和八个子节点有关。这要求该数据结构应该与场景完全解耦。

因此本人利用八叉树上的层级结构信息，结合 Morton code[1]将八叉树上的深度优先遍历信息转换成节点在整个树划分的空间中的坐标。先以二维空间的四叉树结构为例，如图 2-2 所示，按照 Morton code 可将空间向下划分的子空间分别编号为 00（0）、01（1）、10（2）、11（3），即可使用两个比特位对四个子空间进行编号。

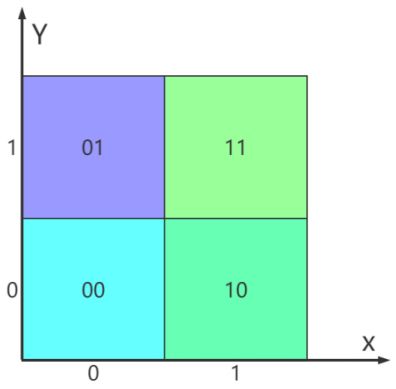

图 3-1 二维空间四叉树的 Morton code

使用 Morton code 对树上所有子节点进行编码后，在对树执行深度优先搜索时，即可通过搜索路径推出节点在树划分的空间中的坐标，反之也可以通过坐标得到搜索路径，快速查找到指定节点。如表 2-1 所示，四叉树空间上的坐标值为(5,3)的点，可以快速推断出对应节点在四叉树上的深度优先搜索路径为：从根节点搜索编码为 2 号的子节点，再搜索该子节点的编号为 1 的子节点，最后搜索新节点编号为 3 的子节点即可找到(5,3)在四叉树上对应的节点。反之在通过深度优先的顺序自顶向下的建立树型结构时，也可以快速判断出当前节点在树所划分的空间中的坐标。

表 3-1 通过点的坐标推导点在四叉树上的深度优先搜索路径

坐标	x = 5	1	0	1
	y = 3	0	1	1
路径	编号	2	1	3
	层级	0	1	2

表 3-2 通过点的坐标推导点在八叉树上的深度优先搜索路径

坐标	x = 5	1	0	1
	y = 3	0	1	1
	z = 5	1	0	1
路径	编号	5	2	7
	层级	0	1	2

八叉树上节点的 Morton code 相对于四叉树增了一个维度。如表 2-2 所示，八叉树空间上的坐标值为(5,3,5)的点，可以快速推断出对应节点在八叉树上的深度优先搜索路径为：从根节点搜索编码为 5 号的子节点，再搜索该子节点的编号为 2 的子节点，最后搜索新节点编号为 7 的子节点即可找到(5,3,5)在八叉树上对应的节点。

根据上述理论，可以推导出在一棵深度为 n 的八叉树上，若令$a_i(i = 0,1,2,\dots,n-1)$表示节点$\mathrm{p} = (\mathrm{x},\mathrm{y},\mathrm{z})$在树上深度优先搜索的路径上第 i 个节点在父节点空间中的编号，则可通过公式 3-1、3-2、3-3 分别计算 p 点的在树空间上坐标的 x、y、z 分量。

$$\mathrm{x} = \sum_{0}^{\mathrm{n}-1}(((\mathrm{a_i} \gg 2)\&1) \ll (\mathrm{n}-\mathrm{i}-1)) \tag{3-1}$$

$$y = \sum_{0}^{n-1} (((a_i \gg 1)\&1) \ll (n - i - 1)) \quad (3\text{-}2)$$

$$z = \sum_{0}^{n-1} ((a_i\&1) \ll (n - i - 1)) \quad (3\text{-}3)$$

反之，当已知点 p 的坐标(x,y,z)与八叉树的深度 n 时，可以唯一确定的推导出八叉树上的搜索路径。

基于上述铺垫，本人构造了一种十分精简的数据结构，即每个节点只保存阴影信息和按照 Morton code 保存八个字节点的指针。该数据结构形式简单且与场景完全解耦，这表明在后续使用优化方法复用重复性信息时，将只需关心数据的重复性，而不必修改除指针外的任何数据。

构建体素化的八叉树

由于本算法所作用的树空间是一个立方体空间，因此构建八叉树的第一步是规定整个树空间的范围。本人在 Unity 中采用一个没有碰撞且只需渲染边缘线框的包围盒确定树空间的范围，令该包围盒中的最小坐标点的坐标向量为p_m，则任意坐标为 p 的点在树空间的坐标p'可由公式 3-4 计算得出。

$$p' = p - p_m \quad (3\text{-}4)$$

在确定了可以向下划分的树空间后，本算法将初始的分辨率设置为与整个树空间的大小相同，即整个树空间共享相同的阴影信息。在算法的每一次递归中，该分辨率将减半，表明在原有空间的基础上按照 Morton order 划分出了新的八个子空间，父节点将八个指针分别指向八个子节点的地址。对于每一个叶子节点，根据 3.1.1 节的公式 3-1、3-2、3-3 计算出对应体素的中心点坐标，并对该坐标点执行深度测试得到该点的阴影信息。

如果基础的体素化八叉树构建止步于此，那么整个树空间中所需存储的节点数量将是庞大而难以忍受的。基于在现实世界的生活经验，由于一些较大体积的物体对光源的遮挡，空间中的阴影区域往往是相互连接的。因此，在这个稀疏的体素空间中，必然存在大量的父节点相同且阴影信息一致的子节点。这表明可以通过父节点直接表示这些子节点的阴影信息，而进一步节省了大量的存储空间，也大量减少了下一步优化的数据量。如图 3-2 所示，逐级向上合并相同阴影信息的节点所减少的数据量是显而易见的，且随着分割深度的增加，压缩率也不断攀

升。

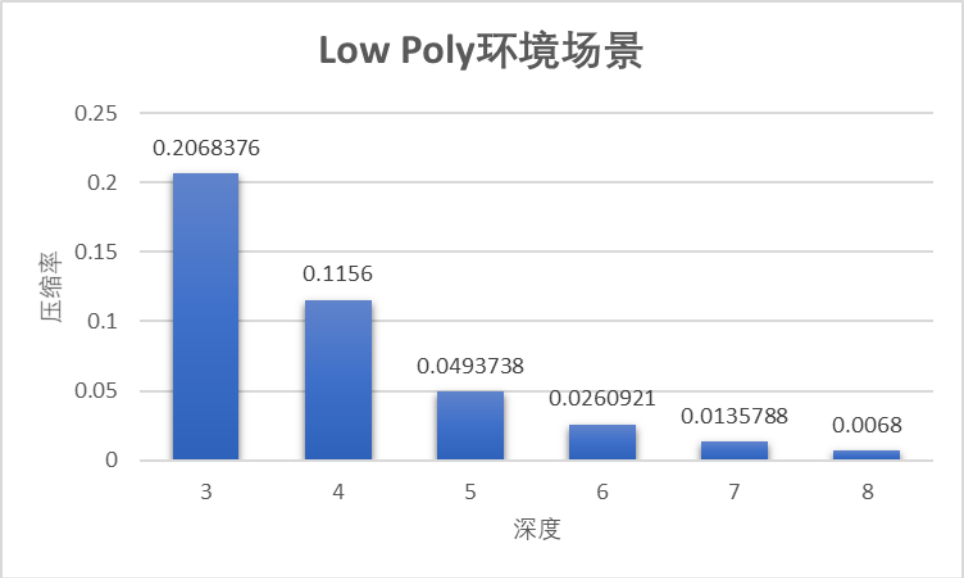

图 3-2 Low Poly 环境场景中向上合并压缩率随深度变化图

闭合区域优化

在 3.1.2 节中介绍了一种逐级向上合并相同阴影信息的节点的方法，该方法所带来的性能提升是十分显著的。本节将描述本算法中所使用的另一种对树节点向上逐级合并的优化方法。

在计算机中，三维场景中的物体都是被分割为一个个面片进行渲染的。而在绝大多数有一定体积的物体内部，必然存在着许多闭合区域。通常情况下，这部分闭合区域是对于观察者不可见的，因此闭合区域内部的阴影状态实际上也不被关心。这所包含的优化意义是显著的，因为它表明对于大部分阴影空间而言，如今只需要详细存储阴影空间表面的阴影信息，而对其内部的阴影状况却可以减少精度和开销。基于这一点，本算法可以进一步优化数据结构，提高存储效率。

如上文所述，位于一个或多个闭合区域内部的节点的阴影信息是未定义的，既可以被认为是处于阴影内部，也可以被认为处于光照中。在本算法的实现中，采取了不符合渲染的方式进行闭合区域的判断。由于在实际工程中，任何物体都将拥有碰撞体积，因此对于每个处于阴影中的体素，对其所代表的空间进行碰撞检测，若发生碰撞则表明该区域处于闭合区域中，即该体素可以向上尝试执行合并。一个节点可以合并其八个子节点的条件被扩展为：所有阴影信息有定义的子节点的阴影信息相同。

闭合区域优化进一步压缩了八叉树的高度，对于原先均被定义为阴影内的大块的闭合区域有着重大的优化效果。

递归展开

在 3.1.2 节中介绍了本算法自顶向下构建体素化的八叉树的递归方式。由于在实际运行环境中，通常需要将树空间进行十次以上的划分，因此递归所需的栈空间往往会超过系统所分配的空间大小，造成栈溢出。针对这一问题，需要将递归算法展开为迭代算法，避免递归过深造成的各种意想不到的问题。

本人首先在堆上开辟了一个栈空间用于在模拟递归过程中暂存数据。在完成存储空间的准备工作后，本人对递归过程中所需要暂存的数据进行细分，通过构造类记录所有这些数据。在这个数据类中还需要记录一个递归的进度值。之所以需要一个进度值，是因为本人是通过将递归按照返回和向下递归划分为多个步骤，步骤与进度值一一对应。当一个步骤执行完成时，则进度值被设置为下一个步骤对应的进度值，随后类被压入栈中。通过这种策略，这种只包含自递归的自顶向下的建树过程可以被正确展开为迭代过程。

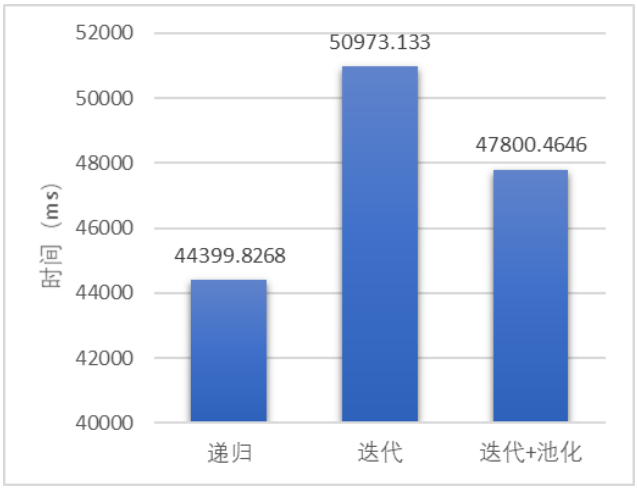

图 3-3 深度 8 时三种建树方式时间消耗对比

由于使用该方式进行递归展开，在执行过程将面临频繁的入栈出栈操作，而入栈时数据类需要重新构造，出栈时数据类会标记销毁等待垃圾回收。这个不起眼的过程将造成严重的性能损耗。针对这一问题，本人进一步实现了一个用于该数据类的内存池。通过池化技术，当执行出栈时，数据类不会被标记销毁，而是继续存在于内存空间中等待复用；当执行入栈操作时并不会总是重新构造一个新

的数据类，而是先查询是否存在可以复用的未释放的数据类。如果存在则直接将数据拷贝到该数据类中，将该数据类标记为入栈；如果不存在再重新构造新的数据类将其入栈。

如图 3-3 所示，池化技术的引入减少了大数据量情况下，系统频繁进行垃圾回收的可能，显著提高了逻辑的执行效率。虽然使用池化技术后的迭代实现相较于递归方式仍有很大的性能差距，但却能够解决栈空间不足问题。

体素化的阴影有向无环图

在 3.1 节中，算法已经完成了体素化的阴影八叉树的构造并进行了部分优化，但距离实现标题中的“紧凑”仍有一段距离。本节将通过介绍 Erik 团队的“紧凑”实现方案[2]和本文的方案，将 3.1 节中得到的体素化的阴影八叉树转换成体素化的阴影有向无环图，实现对重复数据的合理复用。

排序方案

为了实现将体素化的八叉树转换成体素化的有向无环图，Erik 团队提出了先建立完整的八叉树，再自底向上逐层对节点进行排序，从而达到去除和复用重复节点的目的。

在具体排序方案上，Erik 团队提出对于叶子节点，按照八位的阴影信息掩码进行排序，即将八位的阴影信息掩码看作一个 8 位整数；而对于非叶子节点，因为所有子节点已经完成了复用，因此可以将 8 个 32 位的指针看作一个 256 位整数进行排序和去重。自底向上重复上述排序去重方法直到根节点则可以完成将八叉树转换为更为紧凑的有向无环图，而不破坏原有的数据存储，是一种无损压缩方案。

Hash 方案

3.2.1 节介绍了 Erik 团队的排序方案，虽然该方案可以以一种无损方式实现对于八叉树的压缩存储，将八叉树转换为有向无环图，但该方案在性能上却存在着重大的优化空间。

排序方案需逐层进行排序和去重工作，而为了达成去重目的，使用排序方法却着实大材小用。对于排序方案而言，若想自下而上的对同一层的节点进行排序，要么每次进行从根节点开展查询，要么缓存同一层所有的节点信息。无论是哪一

种实现方案，都需要付出时间上或空间上的额外开销。也就是说，只是为了达成简单的去重目的，完全不需要使用复杂的排序方案，本人提出了一种更符合八叉树数据结构特点的去重方案，即树 Hash 方案。

为了实现树 Hash 方案，如公式 3-5 和 3-6 所示，本人定义了一个独特的树 Hash 函数，用于完成使用 Hash 方法去除并复用重复子树的目的。其中 HashVal 用于表示父节点的 Hash 值，Size 表示父节点所对应的树的大小，$hashVal_i$用于 Morton code 为 i 的子节点的 Hash 值，$size_i$表示该子节点对应的子树的大小。而 c_i表示一个子节点对应的一个预先定义的素数，p 表示程序预先生成使用的素数表，max 为该表的大小值，mod 为 HashVal 所能表示的 Hash 量的上限。

$$HashVal = \sum_{i=0}^{7} (c_i * p_{size_i \% max} * hashVal_i) \% mod \tag{3-5}$$

$$Size = \sum_{i=0}^{7} size_i \tag{3-6}$$

如图 3-4 所示，使用树 Hash 方案进一步缩减了存储规模，并且相对复杂的 Low Poly 环境场景中的压缩率更低，因为其存在更多能够复用的子树。采用 Hash 方案的优势不仅在于更有效的发现重复子树并进行复用，而且可以在八叉树的建立过程中进行动态合并。通过 Hash 方案，所有可能存在的重复子树在八叉树构造的早期便被记录下来，不会进入后续的计算，因此当树上节点数量较多时，无论从时间还是空间的占用角度来观察，Hash 方案所带来的性能提升都是十分显著的。

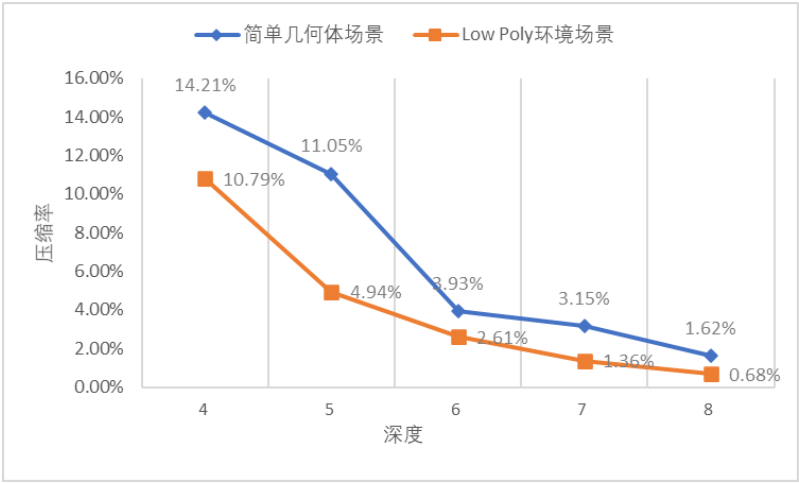

图 3-4 两种场景中 Hash 方法压缩率随深度变化趋势

体素有向无环图的持久化存储

在第 3.2 节中介绍了紧凑的体素化预置阴影算法的核心内容，即通过合并子树将体素化的八叉树转换为体素化的有向无环图，达到压缩存储的目的。直到这一步，算法所管理的一切数据还是存在于内存中，这表明当程序结束，这部分数据将不能继续使用在场景中用于构造阴影。要完成文章标题中的“预置”要求，则需要将程序运行时预先计算的所有可用数据都进行持久化存储，保证即使程序不执行，相关阴影信息也可以继续被使用在场景中用于阴影构造。

数据序列化

由于通过本算法构造的体素化的阴影有向无环图的最终目标是能够在 Shader 中进行数据读取和实时渲染。因此首先需要将体素化的阴影有向无环图中的数据进行序列化，保证数据能够被正确存储下来。

通过第 3 节的前半部分介绍，可以发现体素化的阴影八叉树的数据结构十分稳定。任意选取体素化的阴影八叉树上的一个节点，其中阴影信息和子节点信息必然不会同时有效。这一点很容易理解，因为只有叶子节点才需要进行阴影信息的存储却不需要存储任何子节点信息；对于非叶子节点而言，所有子节点信息都需要被正确存储，而阴影信息是无意义的。通过分析可以得出：对于任意非叶子节点，当其 Morton order 为 0 的子节点存在，则所有八个子节点都必然存在；对于非叶子节点，不存在任何子节点，但阴影信息必须被存储下来。

序列化的目的是进行数据的持久化存储，因此序列化的信息也不应该包括任何节点在内存中的地址。所以本人采用数组对信息进行序列化存储，采用该序列化方法，需要在执行到每个子节点前提前为其分配空间。以根节点作为数组的开始，存储着 Morton order 为 0 的子节点在数组中的下标a_0。如公式 3-7 所示，对于 Morton order 编号非 i（i=0,1,2,..7）的子节点，只需在编号为 0 的子节点下标a_0上加上偏移值 offsets 即可进行位置查询。随后序列化向下递归直到遇到叶子节点。由于叶子节点只需序列化阴影信息，并且阴影信息存在单个布尔值表示形式和八位掩码表示形式。因此需要两个标志位，用于区分存储首子节点下标、存储 1 位阴影信息、存储 8 位

$$a_i = a_0 + offsets \tag{3-7}$$

对于体素化的阴影八叉树，只需执行上述步骤即可完成序列化。但对于体素

化的阴影有向无环图，由于存在节点的复用，为了防止对复用子树进行重复序列化，需要对已经执行序列化的节点进行标记。当发生重复序列化时，则直接将上次序列化的结果下标直接复用。增加这一步判断后，对体素化的阴影有向无环图的序列化即可顺利进行。

纹理存储

在 3.3.1 节中已经完成了对体素化的阴影有向无环图的序列化，现在只需要寻找到一种合适的存储媒介就可以完成体素化的有向无环图的持久化存储。

在序列化过程中，本人通过将同意节点的所有子节点存储在数组中位置相邻的八个空间，并结合偏移量概念，使每个非叶子节点空间内只需要存储首个子节点的下标，减少了单个节点所需要的存储空间大小。

由于在 C#中数组的最大存储大小受 Int32 所表示的最大值的限制，即最大存储 2147483647（即$2^{31}-1$）个元素。因此存储数组下标所占用的位数为 31 位；而对于非叶子节点，存储阴影信息最多需要 8 位，加上用于区分存储种类的标识位，所有节点的数据都可以在 32 位的空间内完成存储。

考虑到有庞大数量的 32 位元素需要进行持久化存储，因此本人选择将数据全部填入一个适合大小的 RGBA32 格式的位图中。虽然 RGBA32 的位图中每个像素所占空间确实是 32 位，但数据是分散在四个通道，即 R（红色）、G（绿色）、B（蓝色）、A（透明度）中进行存储的，所以需要将序列化后的数组中的每一位在存储时切分成四个字节的数据分通道存储。当所有数据存储进纹理像素中后，数据的纹理存储工作便结束了。

纹理读取

在完成纹理存储工作后，只需从纹理中再将数据读取出来便可验证整个流程的正确性。在本人的实现中，Unity 将从资源文件中加载存储的纹理，并以 RGBA32 格式导入。由于在 Shader 中不支持指定像素的读取，所有数据需要通过计算 UV 进行读取，因此本人在导入纹理后将所有像素数据重新恢复为 32 位整数，再存入结构化缓冲区中。存入结构化缓冲区的数据可以按照下标进行读取。

本人在 3.3.1 节中实现的序列化方式保留了体素化的阴影有向无环图的逻辑结构，依然可以按照体素化的阴影八叉树进行层次查询。由于原始的体素化的阴影八叉树的深度是确定的，所以在 Shader 中进行阴影信息查询的最大开销也是

可控的。

在 Shader 中进行指定世界坐标的点的阴影信息的查询时，首先需要将点的坐标转换到树空间的坐标。下一步是根据树空间的坐标利用公式 3-1、3-2、3-3，按照表 3-2 的流程反向推导出搜索路径，按照该路径执行到叶子节点即可读取到该点的阴影信息。

小结

本章完成了紧凑的体素化预制阴影算法中最核心的部分。通过本章介绍的内容，已经可以高效构建一个紧凑的体素化的阴影有向无环图，并可以在纹理中完成数据的序列化存储。更进一步的，通过指定的 Shader 可以在纹理生成的结构化缓冲区中查询阴影信息，从而将预制阴影的作用发挥到最大。

4 阴影效果优化

如图 3-4 所示，通过第 2 节和第 3 节的工作，已经可以成功构建紧凑的体素化预置阴影，但可以发现，阴影效果偏硬，阴影的边缘锯齿很严重出现走样；除此之外非阴影区域也伴随出现很多噪点，这一现象被称为阴影失真（Shadow Acne）。针对上述两种问题，本人将介绍在算法实现中使用的两种优化方案：Shadow bias 和 Percentage Closer Filtering[4]（简称 PCF）。其中 Shadow bias 将被用于缓解 Shadow Acne，而 PCF 将用于缓解阴影边缘的锯齿，减少走样的影响。

Todo:几张不带 PCF 什么的效果图

图 3-4

缓解阴影失真

Shadow Acne

如图 4-1 所示，目前的阴影方案会造成大量的噪点，这被称为阴影失真（Shadow Acne）。在阴影实现中，造成 Shadow Acne 的原因多种多样。在本方案中，Shadow Acne 主要是因为体素分割的精度不足造成的，在同一个体素区域内的所有点都共享着相同的阴影信息。

Todo:不带 bias 的效果图

图 4-1

Shadow bias

为了缓解采样分辨率造成的阴影失真，本人采用了 Shadow bias 方法进行缓解。Shadow bias 的思想很简单，即给所有阴影值增加一个偏移量，使每个采样点都获得比表面深度更大的深度值，保证绝大部分的表面都能被正确的照亮。

引入 Shadow bias 不仅不能一劳永逸的解决阴影失真的问题，还会造成 Peter Panning 问题。Peter Panning 的出现是由于 bias 的值过大，阴影最终明显脱离正确位置。解决 Peter Panning 问题的最常用策略是使用 Slope-Scale Depth Bias[3]，该方法基于物体表面和光源方向的夹角的大小使用不同数值的 bias。该公式定义如公式 4-1 所示，其中 bias 表示施加的 bias 的大小，factorSlope 和 constantBias

则是两个经验值，需要按照具体使用环境调整参数大小。

$$bias = factorSlope * slope + constantBias \quad (4\text{-}1)$$

当引入一个合适大小的 bias 后，如图 4-2 所示，阴影失真问题得到了有效的缓解。

Todo:只用 bias 的效果图

图 4-2

缓解阴影锯齿

阴影锯齿

如图 4-3 所示，本方案实现的阴影边缘会出现明显的锯齿现象，因为本方案的阴影信息被以体素为单位进行存储。当空间划分的深度不足时，阴影边缘的黑色立方体形状的锯齿将十分明显。与此同时，本方案的阴影信息只存储一个布尔位值，这表示阴影将是毫无过度，非黑即白的。

Todo:不使用 PCF 的阴影效果

图 4-3

Percentage Closer Filtering

Percentage Closer Filtering（PCF）的核心思想是在判断一个点是否处于阴影区域的同时需要考虑周围多个点的阴影信息。这表示对于场景中任一点的阴影状况的计算都需要进行多重采样，这是通过付出性能代价换取更好的阴影表现效果。

影响 PCF 性能和效果的最主要因素是滤波核的大小。当采用 3*3*3 的滤波核，则需要同时 27 个体素的阴影信息，由此带来的性能消耗是明显的，但也能很好的缓解阴影锯齿现象。根据滤波中每个点的权重，又能将方法分为平均采样、泊松采样等。

采样 3*3*3 的滤波核进行 PCF 所得到的阴影效果如图 4-4 所示。

Todo:只用 PCF 的效果图

图 4-4

小结

在这一节中，本人引入了 Shadow bias 和 PCF 方法，有效缓解了目前阴影实现效果中阴影失真和存在明显锯齿的问题。通过这两种方法，采用紧凑的体素化的预置阴影算法所生成的阴影效果将显得更加真实，可以在实际应用中有着更好的表现。

5 结果与结论

在本章中，本人将报告并展示通过紧凑的体素化预制阴影算法进行阴影实现的结果，并将结合实际场景对结果进行分析。

Todo:两个场景

图 5-1

参考文献

[1] Morton, Guy M. "A computer oriented geodetic data base and a new technique in file sequencing." 1966.

[2] Kämpe, Viktor, Erik Sintorn, and Ulf Assarsson. "High resolution sparse voxel dags." ACM Transactions on Graphics (TOG) 32.4 (2013): 1-13.

[3] King, Gary. "Shadow mapping algorithms." *GPU Jackpot presentation* (2004): 354-355.

[4] Fernando, Randima. "Percentage-closer soft shadows." ACM SIGGRAPH 2005 Sketches. 2005. 35-es.