

硕 士 学 位 论 文

题 目 基于物理的体积雾和体积光的研究与实现

作 者 孟蓉蓉 完成日期

培养单位 视觉合成图形图像技术国防重点学科实验室

指导教师 刘健波

专 业 计算机科学与技术

研究方向 计算机图形图像

授予学位日期 年 月 日

基于物理的体积雾和体积光的研究与实现

计算机科学与技术专业

研究生 孟蓉蓉 指导老师 刘健波

为了提高游戏和虚拟现实应用中的场景效果，开发人员经常在场景中增加雾和体积光等自然现象。本文工作的主要内容为使用基于物理的方式实现雾和体积光的实时渲染。基于物理的渲染雾和体积光的算法通常存在效率上的问题，此外相关算法不能兼容多光源场景、透明场景、前向着色渲染管线或延迟着色管线，本文算法主要解决基于物理的模拟雾和体积光算法存在的上述问题。

近年来关注度较高的模拟雾和体积光的算法为基于Ray-marching算法和阴影贴图算法的方法，本文以此方法作为研究对象，详细分析了基于物理的雾和体积光的算法的理论基础以及效果和效率 上的优势和缺点，在充分研究Ray-marching算法和阴影贴图算法的基础上，提出更加高效的Ray-marching算法和渲染效果更好的阴影贴图算法。本文算法的核心思想是将散射现象的光照计算部分从Ray-marching步骤中解耦出来，使用计算着色器加速光照计算的过程，将雾和体积光的结果使用中间存储器三维纹理进行存储，作为后续渲染流程的输入，方便复杂场景的渲染，使用指数阴影贴图技术去优化级联阴影算法的效果。

相比于已有算法，本文工作的优点在于：

1. 使用计算着色器提高散射现象光照计算部分的效率，提高Ray-marching算法的效率；
2. 支持多光源场景的体积雾和体积光的渲染，不需要对光源进行预处理，提高渲染效率；
3. 使用指数阴影贴图技术提高级联阴影贴图的质量，解决低分辨率下边缘走样问题和视点移动时阴影效果抖动导致体积光渲染不稳定的问题，提升体积光实时渲染的效果；
4. 算法兼容性好，兼容前向着色渲染管线、延迟着色管线和存在透明物体的场景；

实验结果表明，本文模拟雾和体积光的算法在不同场景中均可以得到较好的渲染结果，并且本文算法不需要针对多光源场景和透明物体的场景进行额外计算或者渲染步骤。本文算法使用计算着色器加速Ray-marching 算法的过程，在多光源和模型数量较多的场景中也能够进行实时渲染。指数阴影贴图技术能够解决场景中阴影边缘走样和视角变换时阴影抖动的情况，在此基础上体积光的渲染效果更好，本文算法能够满足实际应用的需求。

关键词：Ray-marching；散射现象；实时渲染；三维纹理；阴影贴图

Physical realistic volumetric fog and volumetric light

**Major:** Computer Science and Technology

**Postgraduate:** Rongrong Meng **Tutor:**jianbo liu

In order to improve the effect of scenes in games and virtual reality applications, developers often add natural phenomena such as fog and volumetric light to the scene. The main content of this paper is to use the physics-based way to achieve real-time rendering of fog and volumetric light. Physics-based algorithms for rendering fog and volumetric light usually have problems in efficiency. In addition, related algorithms cannot be compatible with multi-light source scenes, transparent scenes, forward shading rendering pipelines, or deferred shading pipelines. our algorithm mainly solves the above problem of physics-based simulation of fog and volumetric light.

The algorithms of simulated fog and volumetric light with high attention in recent years are based on the Ray-marching algorithm and shadow mapping algorithm. This article takes this method as the research object, and analyzes the theoretical basis of the algorithm based on physical fog and volumetric light in detail. As well as the advantages and disadvantages of effect and efficiency, on the basis of full study of the Ray-Marching algorithm and shadow mapping algorithm, we propose a more efficient Ray-Marching algorithm and a better rendering shadow mapping algorithm. The core idea of this algorithm is to decouple the illumination calculation part of the scattering phenomenon from the Ray-marching step, and use the compute shader to accelerate the lighting calculation process. The results of fog and volumetric light are stored in three-dimensional texture which is a intermediate memory. The results are the input of the rendering process which facilitate the rendering effect of complex scenes, and we uses the exponential shadow mapping technique to optimize the effect of the cascaded shadowing mapping algorithm.

The main works of this paper are summarized as follows:

1. We use compute shader to increase the efficiency of the light calculation part of the scattering phenomenon and improve the efficiency of the Ray-marching algorithm;
2. Our algorithm Supports the rendering of volumetric fog and volumetric light for multi-light source scenes, eliminating the need for preprocessing the light source and improving rendering efficiency;
3. We use exponential shadow map technology to improve the quality of cascaded shadow maps, solve edge aliasing which is the problem of low resolution, and solve the problem of shading effect which caused by the shift of the shadow, This problem results in the instability of volumetric light, and improve the effect of real-time volumetric light rendering;
4. Our algorithm has good compatibility and is compatible with the forward shading pipeline, the deferred shading pipeline, and the scene with transparent objects.

The experimental results show that the algorithm for simulating fog and volumetric light in this paper can achieve better rendering results in different scenes, and the algorithm does not require extra calculation or rendering steps for multi-light source scenes and transparent object scenes. This algorithm uses the compute shader to speed up the Ray-marching algorithm. It also enables real-time rendering in scenes with multi-light source and numerous models. Exponential shadow mapping technology can solve the problem of shadow edge aliasing and shadow jitter when the camera is changed in the scene. On this basis, the rendering effect of volumetric light is better, and this algorithm can meet the needs of practical applications.

**Keywords:** **Ray-marching;** **Real-time rendering; 3D textures; shadow maps**

# 目录

[目录 5](#_Toc510448663)

[图表项目表 7](#_Toc510448664)

[1 绪论 8](#_Toc510448665)

[1.1 课题研究背景及意义 8](#_Toc510448666)

[1.2 国内外研究现状 10](#_Toc510448667)

[1.2.1 体积雾 10](#_Toc510448668)

[1.2.2 体积光 12](#_Toc510448669)

[1.3 本文主要工作 14](#_Toc510448670)

[1.4 本文组织结构 14](#_Toc510448671)

[1.5 本章小结 15](#_Toc510448672)

[2 相关算法概述 16](#_Toc510448673)

[2.1 光线传播概要 16](#_Toc510448674)

[2.1.1 散射现象 17](#_Toc510448675)

[2.1.2 散射模型 20](#_Toc510448676)

[2.2 二维Ray-marching算法生成体积雾和体积光 22](#_Toc510448677)

[2.3 本章小结 24](#_Toc510448678)

[3 体积雾和体积光模拟算法 25](#_Toc510448679)

[3.1 算法基本概念 25](#_Toc510448680)

[3.2 算法步骤 27](#_Toc510448681)

[3.3 单个粒子的内散射光线计算 29](#_Toc510448682)

[3.3.1 介质密度生成 29](#_Toc510448683)

[3.3.2 三维纹理存储结构 30](#_Toc510448684)

[3.3.3 多光源 34](#_Toc510448685)

[3.3.4 单个粒子的散射计算 38](#_Toc510448686)

[3.4 Ray-marching算法求解散射积分方程 41](#_Toc510448687)

[3.5 级联阴影算法及优化 42](#_Toc510448688)

[3.5.1 级联阴影 42](#_Toc510448689)

[3.5.2 阴影贴图滤波 44](#_Toc510448690)

[3.6 场景最终结果的计算 48](#_Toc510448691)

[3.7 本章小节 50](#_Toc510448692)

[4 实验结果及分析 51](#_Toc510448693)

[4.1 实验概述 51](#_Toc510448694)

[4.2 实验环境 51](#_Toc510448695)

[4.3 实验设计及结果分析 51](#_Toc510448696)

[4.3.1 实验参数 51](#_Toc510448697)

[4.3.2 不同密度的体积雾和体积光渲染效果展示 52](#_Toc510448698)

[4.3.3 不同光源下体积雾和体积光的渲染效果展示 52](#_Toc510448699)

[4.3.4 带有透明物体的场景的雾效和体积光的渲染 53](#_Toc510448700)

[4.3.5 使用计算着色器和片元着色器渲染效率对比 53](#_Toc510448701)

[4.3.6 本文算法阴影算法和常见阴影算法效果对比 53](#_Toc510448702)

[4.3.7 本文算法局限性分析 54](#_Toc510448703)

[4.4 本章小节 54](#_Toc510448704)

[参考文献 55](#_Toc510448705)

[作者在读期间的科研成果 57](#_Toc510448706)

[声明 58](#_Toc510448707)

[致谢 59](#_Toc510448708)

# 图表项目表

[图 1.1游戏引擎模拟的体积光和体积雾效果 8](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448709)

[图 1.2 BillBoard生成体积光 12](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448710)

[图 1.3 径向模糊原理图 13](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448711)

[图 2.1 散射现象过程示意图 17](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448712)

[图 2.2 外散射内散射示意图 18](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448713)

[图 2.3 光线传播示意图 19](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448714)

[图 2.4 不对称因子对散射分布的影响示意图 21](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448715)

[图 2.5 Ray-marching示意图 23](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448716)

[图 3.1 算法步骤概要图 28](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448717)

[图 3.2 柏林噪声 30](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448718)

[图 3.3 三维纹理空间映射图 31](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448719)

[图 3.4 二维和三维Ray-marching深度分布对比 33](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448720)

[图 3.5 二维和三维Ray-marching效果图对比 33](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448721)

[图 3.6 点光源阴影 35](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448722)

[图 3.7 聚光灯图示 37](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448723)

[图 3.8 求解散射方程 42](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448724)

[图 3.9 级联阴影图 43](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448725)

[图 3.10 视锥体分割图 44](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448726)

[图 3.11 二维和三维Ray-marching效果图对比 45](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448727)

[图 3.13动态场景中阴影纹理滤波与不滤波对比图 47](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448728)

[图 3.12阴影函数图像对比图 47](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448729)

[图 3.14 渲染结果展示 49](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448730)

[图 3.15 存在透明物体的场景下算法的应用 50](file:///E:\Diploma%20Project\2015级硕士\DPPaper\孟蓉蓉硕士毕业论文word.docx#_Toc510448731)

[表 4.1 实验软硬件环境 51](#_Toc509762385)

[表 4.3 主要参数设定值 52](#_Toc509762386)

# 绪论

## 课题研究背景及意义

近年来，随着电子游戏、电影和虚拟现实等行业的快速发展，虚拟场景的渲染成为热门话题之一，很多游戏追求场景和真实世界的相似性，想做到以假乱真的效果，比如最近的热门游戏《绝地求生》，地图是一个岛屿，渲染的场景真实感较强。逼近真实的场景不仅使用全局光照等算法让光照更加真实，并且会渲染自然界中一些天气现象来让用户的沉浸感更强，感受游戏中的时间流逝或者氛围。雾，雨，云隙光和体积光都是场景中经常出现的自然现象。这些现象除了可以烘托场景的气氛，丰富场景的感情色彩外，还可以增加游戏的趣味性，比如很多热门游戏会推出迷雾关卡。



(a)寒霜引擎体积光



(b) 寒霜引擎体积雾



(c) 虚幻引擎体积光



(d) 虚幻引擎体积雾

图 1.1游戏引擎模拟的体积光和体积雾效果

计算机图形学相关的研究一直致力于正确地重现这些物理现象，目前，市面上主流的游戏引擎都支持这些特效的渲染，比如瑞典DICE 游戏工作室的寒霜引擎和美国EPIC Games游戏制作团队的虚幻引擎。如图 1.1所示，图中四张子图都是使用这两个引擎的游戏渲染出来的特效，第四张子图来源于游戏《神佑》的场景，深夜的薄雾可以给用户一种神秘严肃的直观感受。

上述自然现象产生的主要原因是光的散射。光的散射是一个很重要的物理现象，它是光线与传播介质中的粒子产生相互作用的结果。光线在通过不均匀的介质时，有一部分光线会偏离原有的路径，向多个方向发生散射。光线通过尘埃，云滴和雾时，都会发生散射现象。引起光散射的原因是传播介质中存在其他的物质微粒，或者传播介质本身密度的不均匀性。介质中粒子直径的大小和光线的波长会影响散射的效果。蓝色的天空是大气散射的结果，这种情况下，介质中的粒子直径比较小，波长短，频率高的偏蓝色光被偏折到我们眼睛内的概率大。体积光和体积雾同样是光散射的结果，在这两种现象中，介质中的粒子直径比较大。

因为传播介质中的每一个粒子都会影响光线的传播，所以模拟这些自然现象，通常需要花费很多内存资源和计算资源。开发者需要更多的内存去存储和传播介质相关的信息，并且需要计算传播过程中每个粒子对光线的影响。为了提高效率，很多游戏采用复杂度较低的算法去近似模拟这些现象，或者针对场景做一些特殊的优化，使得场景看上去有雾或者体积光的效果，这些方式生成的效果不够真实或者在动态场景中适应性较差。此外，很多追求场景画面质量的研究者致力于采用基于物理的方式去模拟这些现象，这些方法会在空间中离散地选取一些点作为介质中的粒子，考虑这些粒子的光照信息对相机的影响并得到最终的效果。通常采样点越多，并且光照计算越复杂得到的效果越好。但是这些基于物理的方法存在内存或者效率上的瓶颈。如何实时地，真实地重现这些物理现象成为计算机图形学的难点。

在电影制作中，开发者只考虑场景的画面质量，对渲染实时性没有要求。在渲染场景时，开发者会选用基于物理的方式对光线在介质中的传播过程进行模拟，渲染出每一帧的效果，然后再进行多帧混合得到最终结果。因为这些方法的光照更接近于真实世界，所以模拟的效果可以达到照片级的质量，但是处理这些方法需要巨大的计算量，电影中相对复杂的场景，渲染一帧画面的时间在一个小时左右，效率的限制使得这些方法不能在交互系统中采用。

游戏和基于实时渲染的应用在模拟这些现象时，由于设备性能或者实时性要求的限制，他们会采取简化的处理方式，取得视觉上类似的效果。本文算法主要用来模拟体积雾和体积光，所以国内外现状从这两个方面进行分析。

## 国内外研究现状

体积雾和体积光，以及类似的大气散射现象在游戏和虚拟现实的场景中十分常见,效果和实时性是研究者考虑最多的两大因素。目前，国内外已经有很多与体积雾和体积光相关的研究。这些研究从目标性的不同，可以分为基于物理的渲染和基于屏幕空间的渲染。前者更偏重于渲染画面的质量，后者更偏重于渲染的实时性。共同点是这些算法都想在画面质量和实时性之间求取平衡。

### 体积雾

基于物理的体积雾的渲染算法使用更加接近实际物理理论的计算去模拟雾的形成。雾的颜色是由介质中粒子的散射光线决定，当介质中的粒子直径大于入射光的波长时，散射现象在靠近地面的部分表现为雾和烟，在高空中表现为云。在这些算法中，通常假设空间中存在一些粒子，每个粒子有自己的属性，比如密度，位置等等。通过计算，可以得出这些粒子的散射光线对眼睛的影响。

#### 非实时渲染方法

在电影中，为了追求照片级的画面质量，开发者会使用离线渲染去模拟这些大气散射现象。1984 年，James T. Kajiya和Brian P. Von Herzen[10]提出用光线投射算法计算大气散射相关的现象，并且用体密度去模拟传播介质中的粒子分布情况。1995 年Holly Rushmeier[16]引入蒙特卡洛积分来帮助光线散射步骤的计算。2008年Wojciech Jarosz等人[9]使用体积光子贴图技术来提高算法的效果和效率。但是这些算法由于效率的限制没有办法在交互系统和游戏中使用。

#### 实时渲染方法

近年来，随着GPU(Graphics Processing Unit)性能的快速增长，电脑拥有高速的计算能力，研究人员可以将新的方法应用到模拟散射现象中。光线行进算法(Ray-marching)被很多学者应用到体积雾的模拟中。如2014 年，Egor Tusov 提出极线采样[23]的解决方案来提高效率，2014 年，游戏《杀戮地带：暗影坠落》[19]里面渲染体积雾的方法同样基于Ray-marching算法。目前，基于Ray-marching的模拟算法关注度比较高，这些算法考虑到空间中的每个粒子对光线的影响，并且算法的适应性相对于其他算法较好。但是现有的Ray-marching算法都是在二维纹理中完成，二维纹理中进行Ray-marching算法有以下几个缺点：

* 应用场景是单光源，传播介质密度均匀的场景，应用范围较窄，适应性不广泛。
* GPU性能的极大提升使得开发者不再需要在低分辨率的二维纹理中做Ray-marching，这可以避免锯齿状的走样。但是大部分向上采样的算法，例如双边上采样[13]可能会遗漏一些不明显的几何特征或者产生一些高对比度的伪影。
* 大部分模拟体积雾的算法不能兼容于前向着色渲染管线，或者在透明物体的处理中需要额外的计算。

为了满足实时渲染的要求，游戏和应用会简化体积雾的模拟。很多体积雾的渲染算法在屏幕空间下完成以提高效率。游戏中最普遍的全局雾是基于屏幕空间算法的典型例子，这个算法中雾的效果只和场景的深度和高度有关，在后处理中得到雾的混合因子，与原有场景的颜色进行混合，给人视觉上类似的雾效。2006 年，Carsten Wenzel[20]提出一种指数相关的数学公式来计算雾的混合因子，可以让雾看起来更加贴近真实。2014 年，郭璠等人[7]用柏林噪声去干扰雾的空间分布，在场景中生成各项异性的雾。这个算法使用二维的柏林噪声。在场景深度变化时，雾的分布不会改变。上述这些方法都可以近似地模拟雾的效果，但是在有动态光源和场景发生变化的情况下，算法适应性和雾的真实性较差。

### 体积光

早期游戏由于硬件性能的限制，BillBoard贴片技术和径向模糊是经常使用的两种方式。BillBoard 贴片技术原理如图 1.2所示：用图像处理软件生成一个随机的明暗条纹噪声纹理，然后叠加遮罩纹理，使其看起来像光束。将处理好的纹理放置在场景中可能存在体积光的位置，可以产生简单的体积光效果。这种处理方式只能在某些角度效果较好，动态场景适应性较差。

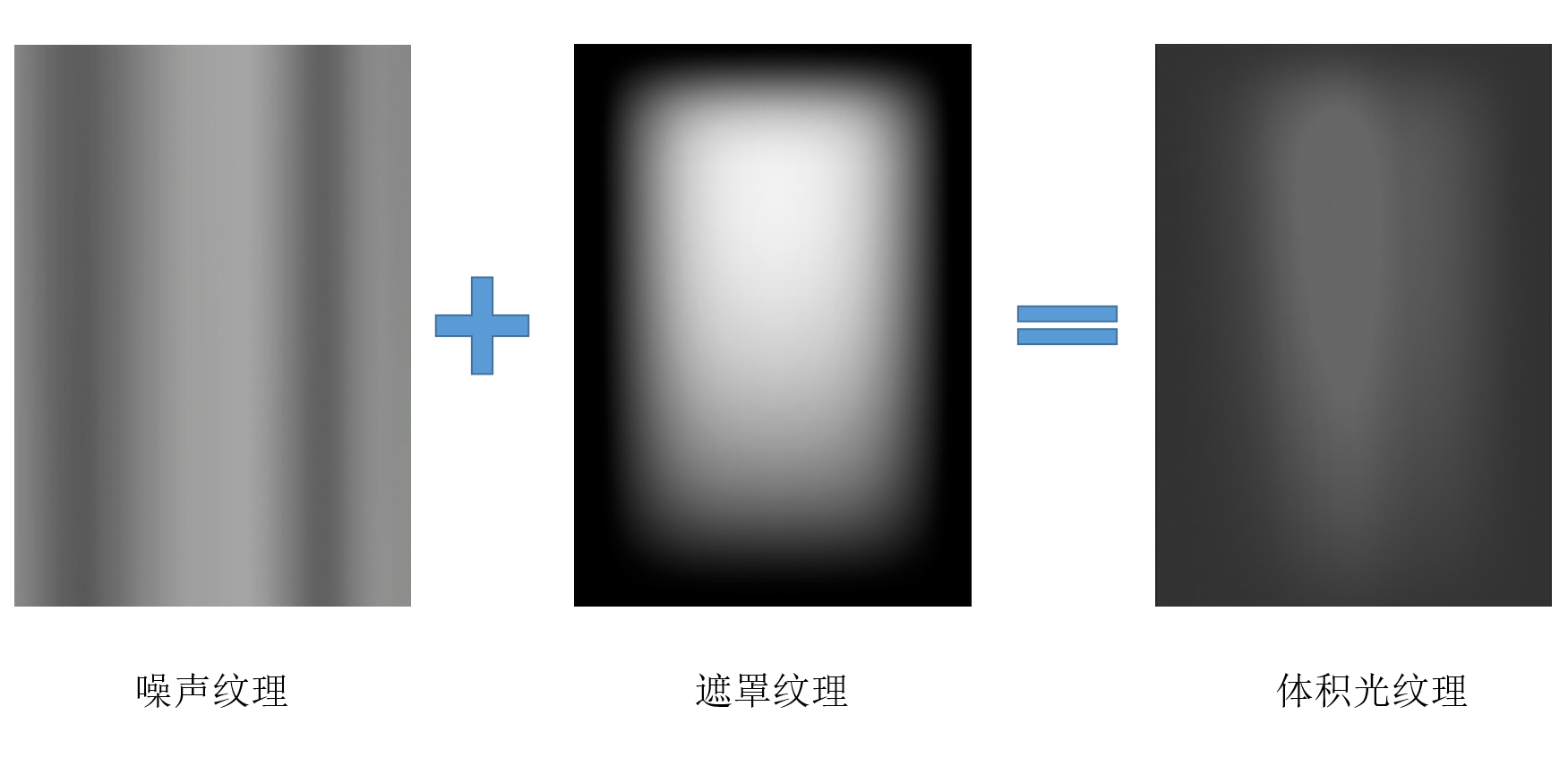


图 1.2 BillBoard生成体积光

径向模糊算法是一种后处理方法，主要用来表现太阳或者月亮的散射效果。径向模糊算法步骤如下：渲染完整个场景以后，提取可见场景中高亮的部分，然后对高亮的区域进行处理，处理步骤如图图 1.3所示，从当前像素点出发，向中心区域移动，每次移动一定的步长，在这个方向上进行采样，将所有的采样点的亮度值相加作为当前像素的亮度值，然后把径向模糊后的高亮层和原场景图进行混合得到最终的效果。径向模糊优点是实现简单，消耗资源小，效果差强人意。缺点也很明显，在光源不可见的情况下无法进行径向模糊。2008 年，Tiago Sousa[17]在孤岛危机游戏中使用另外一种后处理的方式模拟光柱和云隙光。让美工人员提前生成不同场景的光的效果，根据场景不同和时间的不同，选择需要的效果图和原场景进行混合。这种算法自适应能力不强，需要针对不同场景进行优化。

随着硬件性能的不断提升，渲染体积光的算法也在质量上得到很大的改善。最近主流的体积光渲染算法基于Ray-marching算法和阴影贴图技术。2008年Chris Wyman 和Shaun Ramsey[22]提出将影域体算法和Ray-marching算法相结合来模拟体积光的效果。但是这个算法只适用于光源对相机可见的情况，此外，这个算法在二维纹理中进行Ray-marching，会存在边缘产物和走样的情况。2009 年，Balázs Tóth和Tamás Umenhoffer[18]在此基础上提出改进，将阴影贴图技术应用到体积光的模拟中。因为户外大型场景中模型细节丰富，数量众多，所以使用阴影贴图技术存在两个技术难点：第一个是内存消耗和效率问题，第二个是阴影效果走样。近年来，很多阴影算法都致力于滤波低分辨率的阴影贴图来提高阴影的质量。如2007年，Kevin Myers[15]提出的方差阴影贴图和Thomas Annen[1]等人提出的卷积阴影图。2008 年Thomas Annen等人[2]又提出指数阴影贴图。2010 年，Jon Jansen等人[8]将傅里叶不透明贴图应用到阴影技术中。 2011 年，Cyril Delalandre[3]等人提出了透射率函数贴图。这些算法都是在效率和效果中寻找平衡点，通过滤波技术去解决阴影走样问题。2014 年，游戏《杀戮地带：暗影坠落》里面渲染体积光的算法使用的阴影技术是级联阴影[4]，由于级联阴影的贴图中存储的阴影细节更加丰富，所以这个算法在大型户外场景渲染中有一定的优势。级联阴影的缺点是需要更多的内存去存储中间结果，本文算法将级联阴影算法和指数阴影贴图技术相结合，生成低分辨率的阴影贴图缓解内存压力。

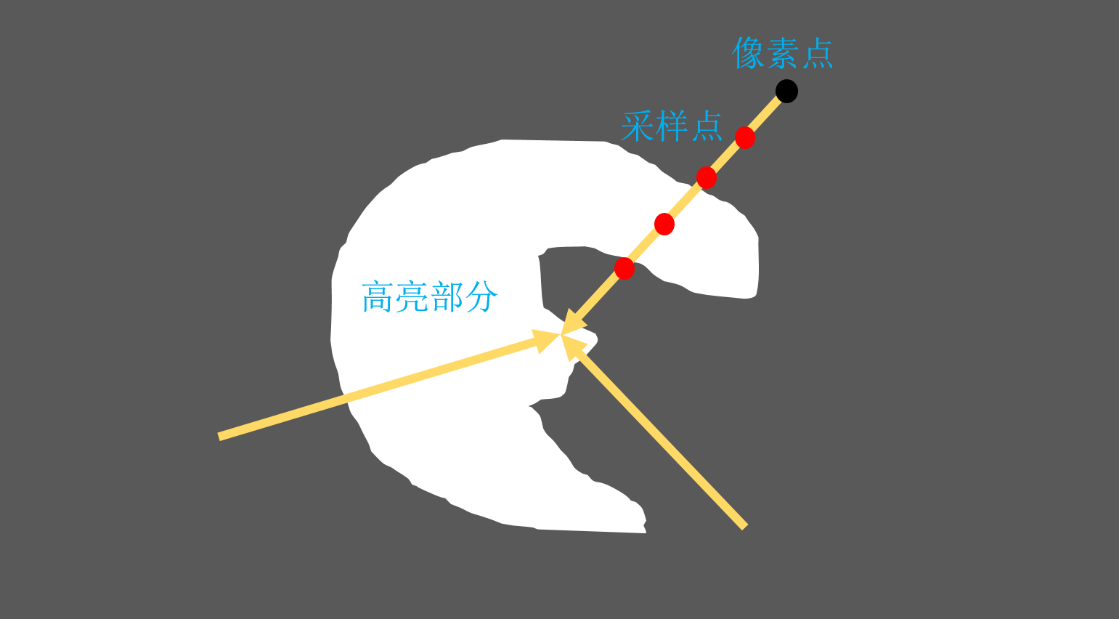


图 1.3 径向模糊原理图

## 本文主要工作

Ray-marching算法和阴影贴图算法相结合来模拟体积雾和体积光是近年来认可度比较高的算法，本文算法在此基础上进行改进。本文工作可以从三个部分进行描述，首先针对Ray-marching算法的效率与阴影贴图算法的内存和效果进行改进，其次将粒子的光照计算部分和其他渲染步骤解耦，提高算法的灵活性和兼容性，最后将原始的场景和本文算法生成的特效相结合，证明本文算法对复杂场景适应性较好。本文算法的改进点和使用技术如下：

1. 使用计算着色器完成Ray-marching算法中的光照计算和求解散射方程步骤，计算着色器可以提高效率，并且我们将中间结果存放在三维纹理中，在后续步骤中可以多次使用，避免重复计算；
2. 运用指数阴影技术去滤波级联阴影的贴图，这个做法可以不必使用过大的阴影贴图去增加内存的负担，并且可以处理阴影边缘的走样；
3. 本文算法可以兼容多种渲染管线并且支持复杂场景的特效应用，比如多光源场景和存在透明物体的场景；

## 本文组织结构

本文共分为五章，具体的章节安排如下：

第一章，绪论。首先介绍体积雾和体积光课题的研究背景、研究意义；然后介绍了体积雾和体积光的国内外研究现状，包括：基于物理的体积光和体积雾、基于屏幕空间的体积光和体积雾等，最后介绍了本文的内容分布和组织结构。

第二章，相关技术研究。介绍与本文算法实现相关的理论知识和技术，包括光线传播概要，散射现象，散射模型，二维Ray-marching算法等。

第三章，算法实现细节。详细介绍本文方法的实现细节和关键技术，包括级联阴影算法的实现和优化，单个粒子的散射计算，多光源下内散射光线的预处理，求解散射积分方程等。

第四章，实验结果与分析。针对本文所提出的改进点进行实验，给出效率和效果上的对比实验，并分析原因，证明本文算法的正确性和可行性。

第五章，总结与展望。对本文提出的方法进行归纳、总结，并分析不足之处，提出未来工作的改进方法和研究方向。

## 本章小结

本章首先概述了本文工作的研究背景与意义，体积雾和体积光在游戏和应用中有广泛的应用，其次介绍了体积雾和体积光的国内外现状，包括基于物理的渲染方法和基于屏幕的渲染方法等，分析现有算法的优点与不足，然后概括了本文的主要工作，主要工作包括本文算法对原有算法改进和本文算法在实际应用中的正确性和灵活性。最后介绍了本文的组织结构。

# 相关算法概述

自然界中各种现象的出现其背后都有一定的物理规则为依据，基于物理的渲染就是利用这些规则重现这些现象，比如众所周知的全局光照算法[5]。本文的工作是用基于物理的方法渲染体积雾和体积光，我们需要知道雾和体积光产生的物理原因，并且根据这个原因构建渲染模型去重现这个现象，因此这个章节会着重介绍大气散射现象相关的物理知识和数学模型，并且介绍一下二维Ray-marching 算法生成体积雾和体积光的算法，本文的算法以此为基础进行改进。

## 光线传播概要

在计算机图形学中，呈现给用户的最终画面由像素组成，这些像素是视点近平面上的离散点。这些像素的颜色由达到像素的光线决定，因此我们需要计算出每个光源发射的光线在场景中的传播路径，求出每根光线在场景中传播后到达视点近平面上每个像素点的辐射强度。

光线在场景中的传播本质上是能量的传播，遵循能量守恒定律。当光线到达物体表面时，因为物体材质不同，会发生反射或者折射等多种光学现象，物体的颜色是这些现象共同作用的结果，对于不透明物体来说，其颜色受入射光线和出射光线影响，入射光线来源于光源或者周围物体的反射光线，出射光线与物体的材质和入射光线的角度有关，用公式来表达这个过程就是我们所熟悉的渲染方程公式。



在公式中， 表示空间中的点在时间时，波长为的光线在点往方向上发出的光线辐射强度总和。表示 点的自发光强度，是以 点为中心点的半球积分公式，表示点处的各个方向的入射光线对出射方向的贡献值总和。为了简化公式，我们假设光线具有特定波长并且光线的出射强度不受时间的影响。此时，点在方向的光线出射强度只与位置和出射方向有关。公式简化如所示：



上述渲染方程描述的是进入我们眼睛的光线辐射强度的来源：自发光和周边光线反射值。这个方程很难求其解析解，大部分基于物理的渲染都是求其近似解。以这个值作为人眼所看到的最终结果有一个前提条件：光线的传播介质是真空。光在传播过程中没有任何能量的损耗，所见即所得。

### 散射现象

现实生活中，绝大部分场景都处于介质中，介质由很多粒子组成。虽然这些粒子不能被肉眼所见，但是它们却对光线的传播过程影响甚大，它们会像肉眼能见的物体一样对光能有吸收和散射作用，因此在光线传播过程中我们需要考虑这些不可见的物质。

由于传播介质中粒子的存在，光线在传播过程会与介质中的粒子进行碰撞，碰撞以后光能会被散射和吸收，在光学里，这个过程用统计学进行数学模型分析。对于单个介质粒子来说，光的传播能量分布如图 2.1所示。

在光线传播过程中，光子的能量被分为下面三个部分：

* 光能在传播过程中进入人眼的部分；

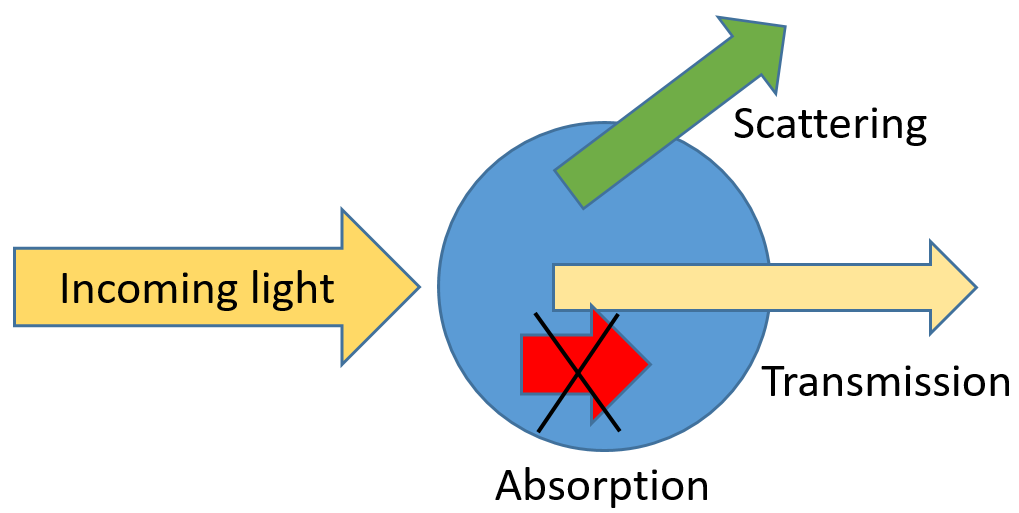


图 2.1 散射现象过程示意图

* 光能在传播过程中被吸收的部分（通常转化为热能）；
* 光能在传播过程中被散射的部分；

根据能量守恒定律，我们可以得到公式：



散射又包含两种类型，一种是内散射，一种是外散射。内散射是指外部光线进入介质中时，经过散射后依然能够汇聚到视点的散射；外散射是指外部光线被散射以后偏离视点方向而不能进入人眼的散射，如图 2.2所示。因为吸收和外散射导致的结果都是光线部分辐射值不能到达人眼，所以消光系数() 为吸收系数()和外散射系数() 之和。

在公式中,代表光线在传播过程中没有受到吸收或者散射的部分，能够透过介质到达人的眼睛。这部分的光线与入射光线的比值被称为透射比。比尔- 朗伯特定律(Beer-Lambert Law)可以计算光在传播过程中的透射比。根据比尔- 朗伯特定律，光在传播过程中，被介质吸收的比例与入射光的强度无关，在光程上每等厚层介质吸收相同比例值的光，我们可以通过消光系数计算出单位厚度下介质吸收光的比例值。这个定律表明， 和光在介质中传播的距离和消光系数有关。比尔- 朗伯特定律计算透射比的数学表示形式为公式：

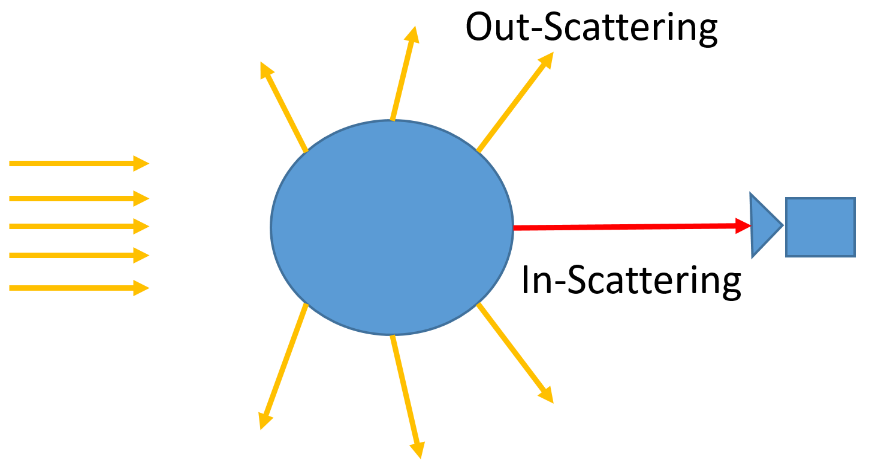


图 2.2 外散射内散射示意图



代表透射光的强度，代表入射光的总强度。透射比为这两个强度的比值。为消光系数，为光线在传播过程中的距离。我们将点和点的透射比用表示。公式中的表示入射光的光强，那么进入眼睛的透射光强度为公式：



真实世界里，进入人眼的除了透射光线，也有一部分光线是散射光线，也就是上述提到的内散射光线。当散射的方向和视线方向一致时，散射光线也会进入人眼。理论上，介质中粒子的散射光线碰到其他粒子之后还会进行散射，进入介质中的光线进行多次散射以后都有可能进入人眼，变成内散射光线。因为粒子数量的庞大和散射过程的各项异性，所以对散射过程进行建模是个复杂的任务。为了方便我们进行建模和提高渲染效率，我们只考虑粒子的单次散射，并且只考虑视线方向上的点的散射光线对人眼的影响。简化模型以后，光的传播过程如图 2.3所示。

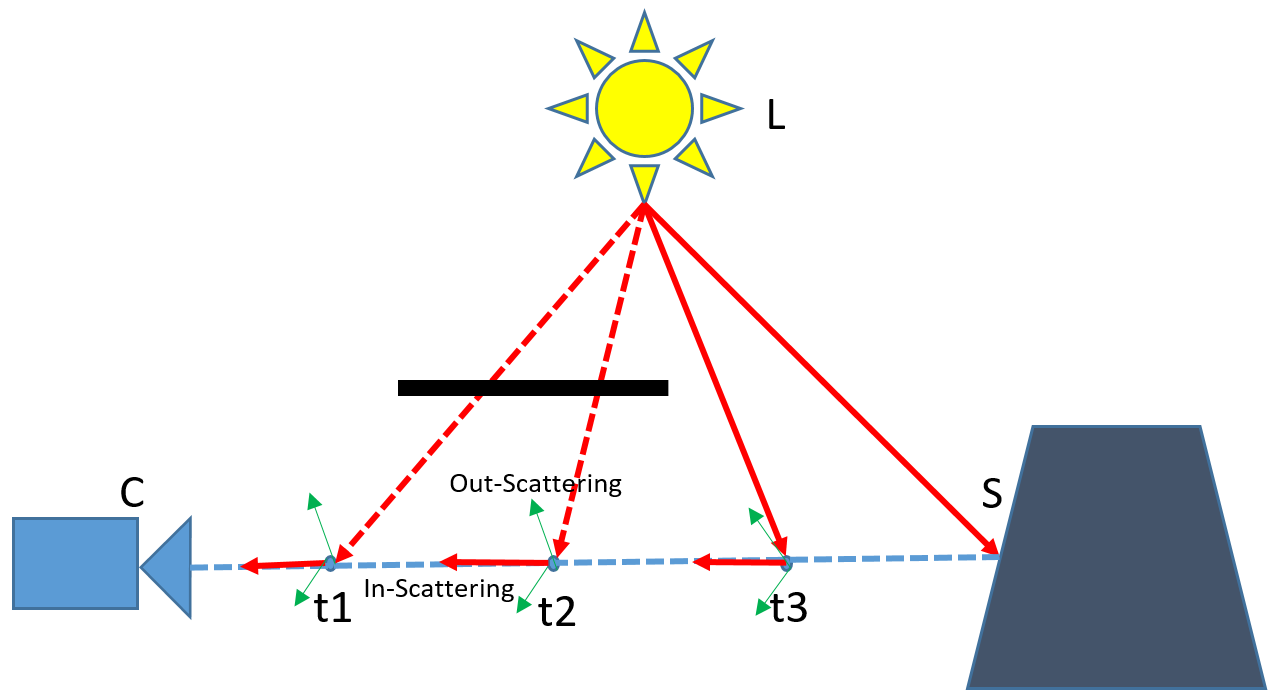


图 2.3 光线传播示意图

在图 2.3中，点是物体所在的位置。点是光源所在的位置，点是人眼所在的位置。点处的光线经过传播介质到达人的眼睛，传播方向上有三个粒子、、，这三个粒子的内散射光线最终也会到达人的眼睛。如图，、粒子被遮光板挡住，处于阴影里，所以他们的内散射光线对最终结果的贡献受阴影的影响，我们计算光线的时候需要考虑阴影因子。在不考虑三个粒子的内散射光线时，点处的反射光线进入人眼的透射光值为，计算公式为\ref{eq:LTE-6}，其中为光源的强度，为点处的阴影因子，为点处视线方向的反射率。



接下来我们需要考虑的是介质中每个点的内散射光线对眼睛的影响。以点为例，点处的内散射光线 计算公式如下，为相位方程，用来表示光线经过粒子时，散射进人眼的光线占总入射光线强度的比例如公式所示：



结合公式和公式，图 2.3中进入人眼的光线强度为公式：



现实生活中传播介质中的粒子数量庞大，将公式进行推广，计算光线传播过程中视线方向上所有粒子-的散射光线，则得到公式:



上述公式考虑的是单光源场景，如果场景中有多光源，那么进入人眼的光线是场景中所有光源的反射光线和粒子内散射光线的总和。如公式所示：



### 散射模型

上一小节主要介绍与光的传播过程相关的物理知识和数学知识。由公式可知，在渲染过程中模拟散射现象，我们需要计算透射光线和内散射光线。透射光与消光系数和传播距离有关，内散射光线由光线在视线方向上的能量传输率决定。根据传播介质中粒子直径的大小，散射可以分为两种模型：米氏散射模型(Mie Scattering) 和瑞利散射模型(Rayleigh Scattering)。

**米氏散射：**

米氏散射模型中，光线传播介质中粒子的直径较大。在大气中，引起米氏散射的主要物质有烟，雾，灰尘，微水滴等。所以雾对光线的散射主要是米氏散射。米氏散射模型中，散射强度和光线的波长几乎无关，不同波长的入射光线会被粒子均匀地散射，因此米氏散射最终的结果呈现白色，如云，雾等。米氏散射具有明显的各向异性，米氏散射中各方向的散射强度由相位方程决定，相位方程依赖于介质中粒子的直径和入射光的波长。在实时应用中，为了减少计算量，采用Henyey-Greenstein方程近似模拟雾的米氏散射相位方程。方程如公式所示：



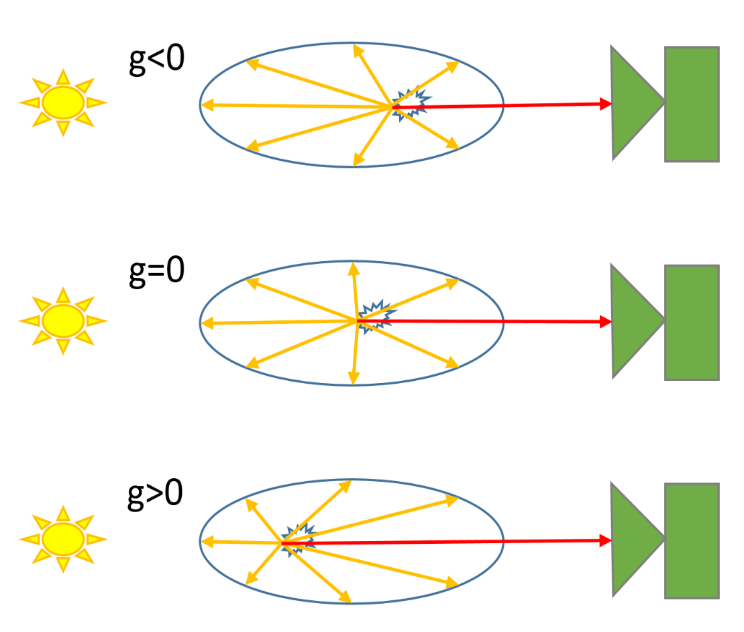


图 2.4 不对称因子对散射分布的影响示意图

其中，，是入射光线与出射光线的夹角。是不对称因子，，对散射光线分布的影响如图 2.4所示，越大，前向散射特征越明显，越小，后向散射特征越明显。

**瑞利散射：**光线在传播过程中遇到直径比自身波长更小的粒子时会发生瑞利散射，比如光线在穿过大气层4.5千米以上的区域。此时传播介质中的微粒直径较小，发生瑞利散射。瑞利散射的计算与同样由散射系数和相位方程决定。散射系数计算公式如所示，相位方程公式如所示。



其中，是可见光波长，是光线的大气折射率，是大气单位体积中的微粒数，表示去极化因子，用于校正大气分子的各向异性。



其中，是入射光线与出射光线的夹角。当夹角为时，计算结果为出射光强度占入射光强度的比重。

由公式可知，瑞利散射的散射系数与波长的四次方成反比。所以在瑞利散射的作用下，波长较短的蓝光进入人眼的强度较大，天空呈蓝色。由公式可知，瑞利散射具有各向同性的特点。

## 二维Ray-marching算法生成体积雾和体积光

在上述光线散射模型中，我们对光线的传播过程进行建模。在模拟光线的散射步骤时，假设光线在传播方向会遇到很多的微粒，这些微粒会对光线有散射作用。介质中的粒子差异性很大，模拟和计算需要消耗的资源较多。为了方便渲染，我们需要简化模型，只考虑视线上的微粒对眼睛的影响，在每条视线上取个离散的点，计算这些点的内散射光线对人眼的影响。

在计算内散射光线的步骤中，我们采用Ray-marching算法。Ray-marching算法的思想如图 2.5所示：从人眼(相机)的位置发射条光线，在每条光线上找到Ray-marching 步骤的起点和终点。在起点和终点之间，每隔一定的步长采样一次，计算我们渲染所需要的信息，步长可以是定长，也可以动态调整。图中，在长方形的包围盒中进行Ray-marching，所以起始点和终点是光线与包围盒的两个交点。因为本文算法以Ray-marching 算法为基础，所以本小节讲一下使用Ray-marching 算法计算内散射光线的基本步骤。

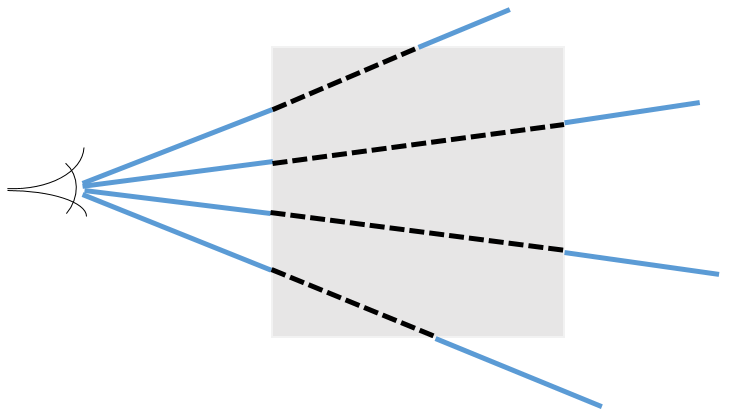


图 2.5 Ray-marching示意图

Ray-marching算法中并行处理每条射线的计算，因此这种耦合度较低的计算很适合在GPU 上实现。通常Ray-marching 算法需要考虑两个问题：Ray-marching 的步长，Ray-marching的方向和起点、终点。Ray-marching 的步长可以取固定长度，这种做法是不需要额外计算与权衡。缺点也很明显，步长太大，会遗漏一些重要的信息，步长太小，Ray-marching的次数会成倍增大。Ray-marching 的步长也可以动态计算，但是可能每个场景对步长的要求不一样，所以动态计算步长要考虑的因素也不一样。在光线传播路径上采用Ray-marching 算法，Ray-marching的起点是人眼(相机)，终点是这条射线与场景的交点。

二维Ray-marching算法模拟体积雾和体积光的算法中，雾和体积光中介质的密度分布由二维纹理决定，在Ray-marching步骤中，通过采样点的位置信息求得二维纹理中的采样坐标，根据采样坐标得到当前采样点的密度。

由公式可知，根据光线传播路径中每个采样点的位置信息、密度信息和散射模型的相位方程等，可以计算出每个采样点内散射光线的强度。从视点出发，到Ray-marching 的终点，我们可以近似计算出在这条传播路径上，有多少散射光线进入人眼。为了考虑算法的效率，对于近平面上的每个像素点，我们只投射一根光线，Ray-marching 算法求得每根光线上的内散射值后，与透射光的强度进行叠加，得到每个像素点最终的颜色。

在不考虑阴影的情况下生成的效果为雾效，体积光的渲染需要在二维Ray-marching算法中增加阴影对粒子的影响。通常情况下为了提高效率，会采用阴影贴图技术，基本的阴影贴图技术在场景较为复杂的情况下不能很好处理阴影，因此很多研究者会考虑使用滤波等操作来优化阴影。

在二维Ray-marching算法中，三维空间中粒子的分布只由二维纹理控制，缺点是密度分布的同质性较强，视觉效果欠佳，因此，二维Ray-marching算法会增加一个随机变量来干扰雾密度的分布，使其看起来呈现各项异性的特点。二维纹理中进行Ray-marching算法的另一个缺点就是二维纹理需要较大的分辨率，较小的分辨率走样的情况比较严重。此外在存在透明场景的物体中，雾效和体积光的渲染需要额外的计算成本。

## 本章小结

本章首先简单地介绍了雾和体积光形成的原因，详细介绍了和光线传播相关的物理知识和数学模型，然后简要讲述了散射现象和两种常见的散射模型。最后本章介绍了如何在二维纹理中应用Ray-marching算法生成体积雾和体积光，这个方法是本文算法的基础，本文算法在此方法上进行改进，改进点和改进方法在第三章中作详细说明。

# 体积雾和体积光模拟算法

用Ray-marching算法和阴影贴图算法生成体积雾和体积光是近年来比较流行的方法，本文算法在原有算法[18][19]的基础上进行改进，改进点包括四点：

* 提高Ray-marching算法的效率；
* 支持多光源场景的体积雾和体积光的渲染；
* 通过提高阴影的质量来提升体积光渲染的效果；
* 兼容性好，兼容前向着色渲染管线、延迟着色管线和存在透明物体的场景；

本文算法主要包括，介质密度的生成、Ray-marching算法的改进，级联阴影的实现与改进，多光源的预处理步骤。本章将对整个渲染流程以及涉及的算法进行详细地描述与分析。首先概述算法的核心思想，然后对算法的基本原理和实现细节进行了叙述，最后对本文所采用的方法进行了分析与总结。

## 算法基本概念

本文算法将采样点的光照计算部分和Ray-marching积分计算部分解耦，提高原有算法的效率。在章节2.2中详细描述了二维Ray-marching 算法的步骤，在二维Ray-marching 算法中，在片元着色器进行Ray-marching 步骤，从视点出发在视线上每隔一定的步长采样一次，计算出当前点的光照值和阴影值，每次得到当前点的光照计算值时，与前面所有点的光照结果做积分运算，求得当前点的消光系数和累计光照值，Ray-marching的终点为视线与场景的交点。虽然光线与光线是并行计算关系，但是光线上每个采样点的计算存在依赖关系，当前采样点的累计光照值依赖于前面所有采样点的计算值。假设每根光线上采样点数量为，每个点自身的光照计算所需时间为，与前面所有采样点进行积分求和的计算时间为，屏幕分辨率为，在片元着色器中Ray-marching步骤所需的总时间为。虽然Ray-marching步骤中的累计光照值的运算依赖于每个采样点的光照计算值，但是每个采样点的光照计算步骤互相独立，所以本文算法将每个采样点的光照计算从Ray-marching 步骤中解耦，将这个计算量大但是相互之间独立的步骤使用三维计算器着色器完成，这个步骤的结果存储在三维纹理中，传入后续的Ray-marching步骤中供光照积分运算使用，在Ray-marching 步骤，使用二维计算着色器进行计算，和片元着色器中的计算方式一致，本文算法中这两个步骤的总时间为，本文算法与原有算法相比在Ray-marching步骤缩短的时间理论值为，在章节4.3.5中会有实际的实验数据表明这个步骤效率的提升对整个渲染流程效率提升的影响。

本文算法可以支持多光源场景，透明场景，兼容前向着色渲染管线和延迟着色管线，不需要额外的渲染绘制遍进行处理。在极线采样算法渲染体积光[23]中，需要对Ray-marching 算法进行预处理，使用光源的相机矩阵和透视矩阵对Ray-marching 投射的光线进行预筛选，通过减少光线数量来提高效率，这个算法只支持单光源处理，此外由于光线投射数量的减少，这个算法只能适用于户外场景，对于近处的体积光模拟效果并不理想。游戏《杀戮地带：暗影坠落》[19]中模拟体积光的算法为了提高效率，只在光源范围内进行Ray-marching，需要提前渲染光源几何体并且记录该几何体的深度影响范围，该步骤使得这个算法只能在光源可见的情况下生效，并且如果一个像素点如果同时受几个光源影响，这个算法需要改变存储光源几何体深度的数据结构，二维纹理没有办法满足其需求。光源对粒子的影响本质上是影响粒子的内散射光线，本文算法在三维计算着色器中处理所有光源对粒子的光照影响，根据每个光源的位置信息、颜色信息、衰减函数、阴影贴图和相位方程等可以求得其对粒子的内散射光线的贡献值，在光照计算步骤不需要预处理，且每个光源的计算过程与其他光源没有耦合，同一个粒子受多个光源影响只要分别计算出每个光源的影响值然后进行叠加，这个步骤在章节2.2中有详细说明。计算出光源对粒子的影响后，我们在积分过程中使用每个点的内散射光线和消光系数计算出当前采样点的累积光线值，并将累积光线值存储到对应的三维纹理中。将雾效和体积光应用到场景中时，我们使用场景的深度值作为参考值去采样三维纹理，得到当前深度对应的雾效或体积光的颜色。在本文算法中，计算出每个粒子最终的累积光线值以后，在后续处理过程中，唯一的传入参数是场景的深度，在前向着色渲染管线和延迟着色管线中场景深度很容易获得。此外对透明物体的处理，由于其渲染流程的特殊性，比较主流的透明物体渲染算法[14][6]都保存了每个透明片元的深度，得到透明物体片元的深度值以后，我们可以像处理不透明物体的片元一样得到当前透明片元雾的颜色值或体积光的影响值，本文算法在没有改变透明物体的渲染流程和方式的情况下可以完成透明物体的雾效和体积光的渲染。

本文算法对阴影算法部分进行改进，使用ESM技术对级联阴影进行滤波。原有算法中使用级联阴影算法生成场景的阴影，级联阴影的思想和LOD[24]的思想很相近，在靠近相机的地方使用更精细的阴影贴图纹理，远处的地方使用粗糙的阴影贴图纹理，将纹理的分辨率进行合理划分，提高场景的阴影质量。级联阴影因为需要生成多张阴影纹理，所以会给内存增加负担，此外当相机很接近物体时，总会存在因为阴影纹理分辨率不够而产生走样的效果，所以本文算法使用指数阴影贴图技术去解决这两个问题，指数阴影贴图技术对原有的级联阴影贴图进行滤波处理，这个操作解决了分辨率较低的阴影贴图的走样问题，在章节4.3.6中会对这部分内容做详细的分析和描述。

本文算法的主要思想是将Ray-marching步骤和粒子的光照计算等步骤从原有算法中解耦出来，将这些步骤计算得到的结果暂时保存下来，供后续步骤使用。本文算法使用计算着色器(Compute Shader) 来处理和粒子相关的光照计算，提高渲染的效率。介质中的每个粒子需要单独计算内散射光线并保存内散射的结果，本文算法用三维纹理来存储每个粒子的内散射光线和消光系数。2009 年，Kaplanyan 等人[12]提出将三维纹理应用到游戏和电影的制作中，作为渲染过程的中间存储器。在2010年，Magnus Wrenninge 等人[21]将这个想法应用到特效行业，计算单次和多次散射效应，本文算法受其影响使用三维纹理的数据结构。此外，使用指数阴影贴图技术优化级联阴影相对于其他阴影滤波技术来说有效率、效果和内存上的优势，这部分会在章节3.5.2中做详细描述。

## 算法步骤

本文算法分为以下几步：

* 生成级联阴影贴图并使用指数阴影贴图技术滤波；
* 计算每个粒子的内散射光线；
* Ray-marching求解体积雾或者体积光的颜色；
* 将雾效与渲染场景相结合；

本文算法详细的算法步骤描述如下：本文算法所有步骤在视锥体中完成，Ray-marching是在视锥体中进行，Ray-marching步骤的作用是对视线上所有采样点的内散射光线值进行积分，求得雾和体积光的颜色值。所以本文算法第一步是求每个粒子的内散射值，计算内散射值所需的输入参数为：粒子的密度，相位方程，粒子的位置，场景中光源的信息。计算出每个粒子的内散射值以后再用Ray-marching算法做积分运算，求得雾效，其中单个粒子的内散射值存放在三维纹理中，供Ray-marching步骤使用，Ray-marching 步骤的结果同样存放在三维纹理中，供后续效果叠加步骤使用。本文算法在GPU上的主要渲染绘制遍和使用的数据结构如图 3.1所示。



图 3.1 算法步骤概要图

图 3.1中灰色箭头代表数据的传递过程，红色箭头代表渲染过程。绿色长方形代表二维纹理贴图，蓝色长方体代表三维纹理，这些数据结构都用来存放计算步骤的中间结果。首先，生成多张级联阴影贴图，将这些贴图经过FS(Fragment Shader)的滤波处理变成指数阴影贴图，指数阴影贴图在计算密度和光照部分的CS(compute Shader)中用到，这个CS 的处理结果是单个粒子的散射值，将单个粒子的散射值传进第三个CS 中求解散射积分方程，得到此方程的近似解。最后在FS 中，使用散射方程的近似解、场景的深度纹理和颜色纹理求得最终输出到屏幕上的像素值。下面小节将会对本文算法每个步骤进行详细描述。

## 单个粒子的内散射光线计算

在视线上积分求取雾效的颜色首先需要知道每个粒子的内散射光线值，所以这一小节首先介绍如何计算单个粒子的内散射光线。计算单个粒子的内散射值需要计算出粒子的密度，粒子的位置信息，场景中每个光源对粒子内散射光线的贡献值，因此首先我们需要预计算粒子内散射光线所需要的信息，然后介绍如何利用这些信息计算粒子的内散射值。

### 介质密度生成

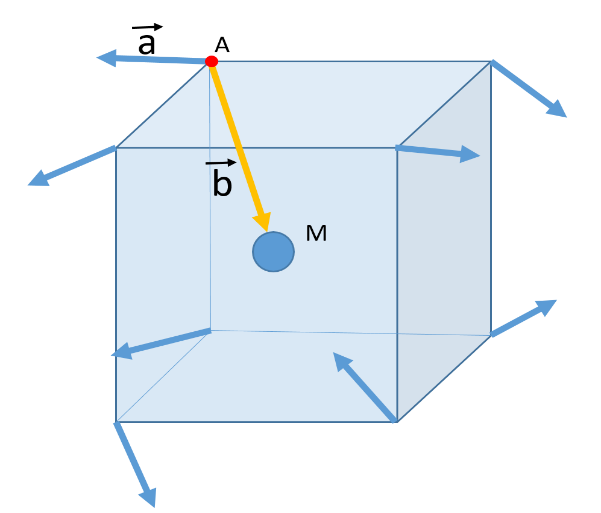
因为我们需要用密度来计算散射需要的消光系数，所以介质密度生成是求解散射方程的第一步。介质密度的生成有三种，第一种是函数模拟，第二种是用柏林噪声生成雾的密度分布纹理，第三种由美术人员使用美术工具生成。

函数模拟的方式如下。以雾为例子，雾的密度比空气大，雾相对来说会聚集在地面附近，雾的密度是在高度上呈指数分布趋势，雾的密度分布函数如公式所示。

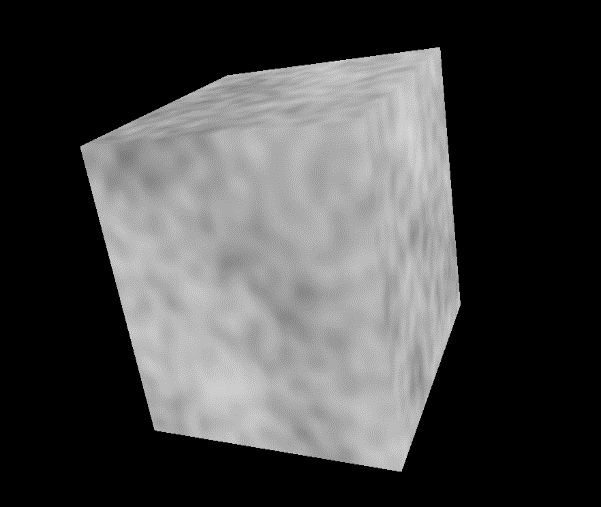


其中，表示高度为时粒子的密度，代表密度的基准值，通常情况下为地面处的密度。系数用来表示粒子密度的类型，越大，随着高度的变化，密度下降速度越快。

实际场景中，雾的密度分布不均匀的因素有很多种，只用高度作为输入参数去干扰雾的分布不够真实，并且相机只有在远景视角才能体现。为了使密度的分布更加自然，需要用噪声纹理去干扰雾的分布，但是使用普通随机算法生成的噪声纹理与自然界中物体的随机分布有很大差别，模拟出来的效果不够真实。本文算法采用柏林噪声[25]去干扰雾的分布，柏林噪声在图形学中的应用广泛，常用于自然现象的模拟，可以得到很好的效果。介质的粒子在三维空间中分布，所以我们需要三维的柏林噪声纹理，柏林噪声需要一个种子作为输入参数，我们使用位置信息作为柏林噪声的输入参数。柏林噪声的生成如图3.2(b)所示。假设点是三维空间中的一个粒子，坐标为，这个粒子的噪声值由其周围的八个顶点决定，每个顶点有一个伪随机的梯度向量，三维空间中这些梯度向量不是完全随机的，是单位正方体的中点指向每条边中点的12个向量，每个顶点从这十二条向量中随机选取一条。2002 年，ken Perlin[26]解释了选用这12 个向量的原因。顶点的梯度向量为，点到点的向量为，点处的噪声值为，点对点的、、三个方向上的影响由向量 的三个分量决定，，使用插值函数计算三个分量上的影响因子、、，插值函数为，因为这个函数的一阶导数和二阶导数在0和1处都为0，所以生成的噪声纹理更加平滑。计算点的噪声值时，首先按照上述步骤计算出每个顶点的噪声值，然后使用向量计算出、、三个方向上点的影响因子，最后计算出点的噪声值。本文算法中三维柏林噪声纹理的渲染结果如图3.2(b)所示。



b 柏林噪声生成示意图



b 三维柏林噪声渲染图

图 3.2 柏林噪声

美术工具预先生成的方式不属于算法的范畴，不做过多赘述，美术生成雾和体积光的体密度纹理，在程序中密度可以通过采样直接得到，渲染效果和体密度纹理的分布有关。

### 三维纹理存储结构

因为本文算法在三维空间中进行Ray-marching步骤，介质中的微粒在三维空间中存在，所以我们可以将介质所在的空间假设为场景中的一个长方体，长方体被分割成很多个微型立方体，每个立方体中心是粒子的坐标，计算粒子的密度值时，函数的输入参数为粒子坐标位置方向的值，柏林噪声的输入参数为粒子的坐标位置。三维纹理可以作为存储结构使用，能够满足上述所有需求。

在计算散射的过程中，需要预计算每个粒子的内散射光线值，每个粒子的内散射光线值存在三维纹理中，供后续Ray-marching步骤计算累计散射光线值使用，累计散射光线值也需要用一个三维纹理进行存储，作为场景添加雾效和体积光的步骤中的输入参数，因此本文算法需要两个三维纹理作为中间存储器。纹理有四个通道(RGBA)，每个通道存储16 位的浮点数类型。考虑到内存的开销，我们选用分辨率较低的三维纹理，本文算法中使用的分辨率为128×128×64。

因为需要计算出三维纹理中每个纹理代表的粒子的位置，所以将三维纹理坐标系映射到屏幕坐标系是一个关键的步骤。三维纹理与屏幕坐标系的映射如图 3.3所示。经过透视投影矩阵的处理，场景中的点会被投射到二维的屏幕中，三维纹理中每个纹理对应的点也会被映射到屏幕上，三维纹理的，轴代表屏幕的，方向，轴代表屏幕方向上的深度。根据透视投影近大远小的原则，靠近相机的地方需要更多的细节描述。基于这个原理，我们在用三维纹理描述屏幕空间下方向的分布时不采用线性映射，采用指数映射。在经过透视投影矩阵处理以后，相机近平面的屏幕深度值为0，远平面为1，所以三维纹理中 的采样点所代表的深度为0，(为方向的最大采样深度，本文算法中)的采样点所代表的深度为1，在本文算法中，我们对0-1之间的深度进行分割，分成64份，越靠近0 的地方，分布越密，越靠近1 的地方越稀疏，所以三维纹理中的深度分布如下，的采样点深度为 ，的采样点深度为 ，的采样点深度为 。 这种选取步长的方式其实是将计算资源分配更加合理，靠近相机的场景细节部分会被更加充分的考虑。

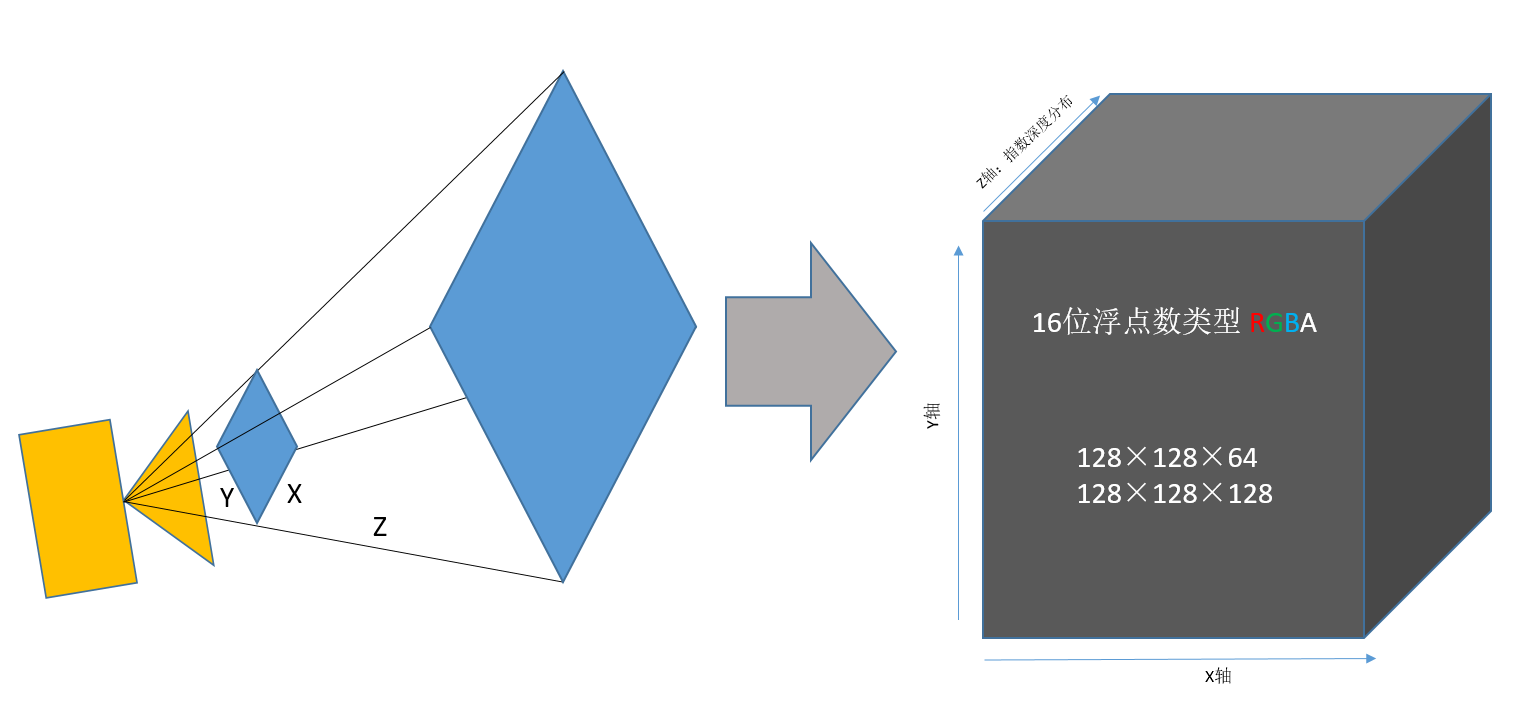
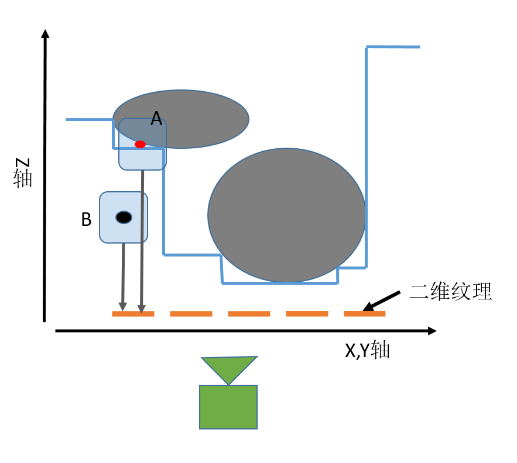


图 3.3 三维纹理空间映射图

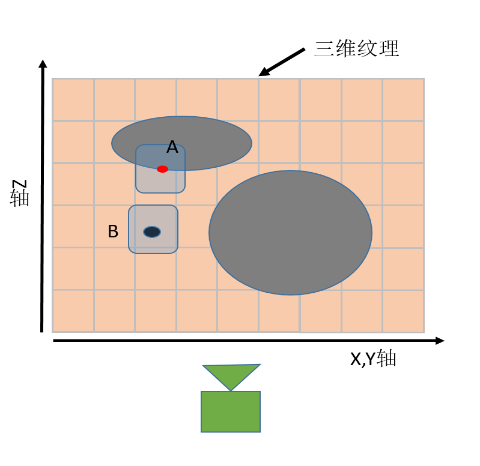
得出三维纹理坐标系和屏幕空间坐标系的映射关系以后，通过计算得到每个粒子的坐标位置（这个步骤在章节（。。。）中会详细描述），将坐标位置作为上一小节密度公式的输入参数和柏林噪声的输入参数可以得到不同的密度分布，使用坐标信息和密度信息就可以进行后续的光照计算。

为了节约内存开销，本文算法使用的纹理分辨率较低，为128×128×64，对于、轴而言，因为屏幕上一个区域内的像素点采样得到的密度值为同一个值，所以低分辨率下雾和体积光的效果会不够精致，雾会产生小正方行状的块状分布。相对于二维Ray-marching 算法来说，三维Ray-marching 算法得到的结果会好很多。因为二维Ray-marching 算法步骤中，密度纹理值只和、 方向有关，为了体现深度上的差异，通常会随机选取一个值来干扰密度纹理的分布，使其在深度上有差异，但是实际上这个值和深度没有任何关系。考虑到这个问题，有些算法会采用双边上采样来修正结果，但是当空间中存在细小几何体或者空间比较复杂的情况下这种处理方式仍会失效。本文算法不会受这个问题影响，因为对屏幕上每个像素而言，其采样的纹理值和其空间位置有关，空间位置包括这个像素点的深度值。

在实际操作过程中，二维Ray-marching算法和三维算法的密度纹理采样过程比较如图图 3.4所示。在二维Ray-marching 算法中，点和点对应的采样点为二维纹理中相同的位置，但是实际过程中，他们在空间中的密度分布并不不一样。在三维纹理中，采样坐标除了受、方向的值影响，还受深度影响，所以点和点可以分别得到精确的密度值。二维Ray-marching算法和三维Ray-marching算法相同分辨率下得到的效果如图 3.5所示。



二维Ray-marching深度图分布



三维Ray-marching深度图分布

图 3.4 二维和三维Ray-marching深度分布对比



b 二维Ray-marching雾的效果图



b三维Ray-marching雾的效果图

图 3.5 二维和三维Ray-marching效果图对比

此外为了提高场景的效果，更好地解决低分辨率下产生的问题，本文算法还做了额外优化工作，这会在章节3.5.2中详细描述。

### 多光源

计算内散射光线首先得知道当前粒子的入射光线，米氏散射和光的波长无关，所以我们计算入射光线时可以将多个光源的入射光线强度叠加，在计算散射的时候对总入射强度进行处理。在实际渲染过程中，有三种光源，平行光、平行光和聚光灯。因为三种光源计算光线的方式不一样，所以本章节中会分开描述。

#### 平行光

平行光通常用来模拟太阳光，从固定方向照射到物体上，光线的方向互相平行，平行光的强度为固定值，没有距离的衰减。平行光的阴影可以用级联阴影生成，级联阴影的生成在章节3.5.1中有详细的描述。对于平行光的光照处理比较简单，介质中微粒接受平行光的入射光线计算公式如所示，平行光的入射光线为入射强度与颜色的乘积。



其中，为入射光线的值，为平行光的强度，为平行光的颜色值，为阴影因子。

#### 点光源

点光源没有固定方向，从一点出发，向各个方向均匀地发射光线的光源为点光源，点光源的衰减和距离有关，当物体离光源越远时衰减效果更加明显。在一般情况下，会给定一个简单的假设，光线强度和距离的平方成反比。光强计算如公式所示。



这个公式基本能满足衰减的要求，但是产生的视觉效果不太理想，比如距离很近的时候，因为分母很小，所以光线的强度会趋近于无穷大，而且除了初始光强，没有其他系数来控制衰减程度。所以我们倾向于在这个公式的基础上加另外一些控制系数。如公式所示。



这个公式中，加了三个光的衰减系数（常量系数，线性系数，指数系数）。当常量系数和线性系数值为0，指数系数值为1时，便是公式。当常量系数为1 时，在距离为0时，光强是程序设置的最大光强。通过调整线性系数和指数系数可以控制衰减的快慢，得到不同的灯光效果。点光源的入射光线值计算如公式所示。



因为点光源向各个方向发射光线，在计算点光源阴影时，不能使用基本的阴影贴图。在点光源的阴影贴图中，需要保存各个方向的阴影贴图。我们使用立方体贴图来保存六个面的深度值。如图 3.6所示，以点光源为原点，上下左右前后六个面都需要生成阴影贴图，每个面有单独的光源变换矩阵，经过变换矩阵的处理，得到阴影贴图，存放在立方体贴图对应的面。

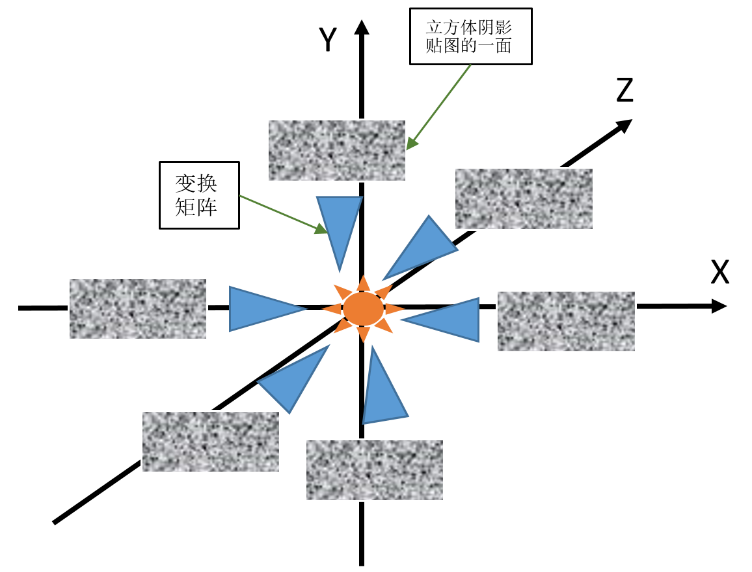


图 3.6 点光源阴影

用这个方法创建点光源的深度值贴图需要将场景重复渲染6次，每次可以得到一个面的阴影深度值。多次进行渲染调用会降低场景的效率，在场景特别大的时候这个弊端会更加明显。其实渲染场景的六次过程中，只是将场景中的点或者三角形做了6次不同的矩阵变换，矩阵中变化的只有相机矩阵。通常来说矩阵变换是在顶点着色器中完成，为了方便处理，我们可以借助几何着色器来帮助渲染立方体贴图。具体步骤如下：

* 创建一个立方体贴图，为立方体每个面绑定一个二维的纹理图像用来保存点光源6 个面的深度值，为每个二维的纹理设置纹理参数；
* 使用函数将立方体贴图绑定到到帧缓冲区的深度区域，因为立方体纹理贴图只关心深度值，不需要往颜色缓冲区输出颜色，所以将和函数的参数都设置为;
* 为点光源的6个面计算其单独的变换矩阵。变换矩阵包括投影矩阵和相机矩阵。在设置投影矩阵时，将视野参数设置为，确保视野能够填满立方体贴图的每个面;
* 在渲染过程中启用几何着色器，在顶点着色器阶段，我们对顶点只做模型矩阵变换，几何着色器阶段的输入是以三角形为单位处理三角形的三个顶点，每个顶点需要使用到步骤三中计算的变换矩阵进行6次变换生成6个不同的顶点，所以几何着色器阶段输出为18个顶点，为了确保几何着色器可以帮助我们将这6个面的三角形传送到立方体贴图对应的面上，我们需要使用几何着色器中的内建变量，为同一组三角形的顶点设置相同的值，几何着色器就会把这组顶点传送到对应的立方体贴图中。在片元着色器阶段我们计算阴影的深度值并将深度值写到对应的纹理中；
* 在计算光照阶段，我们将片元的世界坐标系和点光源的世界坐标的差值作为采样坐标对立方体纹理进行采样得到正确的深度值，将这个值与片元在光空间下的深度值进行比较，计算出阴影因子；

阴影因子求出以后，根据公式便可以计算出点光源的入射光线值。

#### 聚光灯

聚光灯是三种灯光里比较复杂的一种灯光。聚光灯可以理解为有范围的点光源，聚光灯有位置信息，我们需要给它指定一个固定的方向，它的光线只分布在圆锥形的空间中，并且光线也会像点光源的光线一样随着距离而不断减弱。手电筒是典型的聚光灯。聚光灯有三个参数：位置信息、聚光灯的方向、光线分布的圆锥形的锥体角。下面我们要讨论的是怎么判断物体是否受聚光灯影响，聚光灯的衰减因子计算和聚光灯的阴影计算，通过这三个系数的计算我们才能最终得到聚光灯的入射值。

如图 3.7所示，聚光灯的位置为，锥体角为，方向为。物体和物体与聚光灯位置所形成的射线与聚光灯的方向形成的夹角分别为和，我们通过判断夹角与角的大小，就可以得出物体是否受聚光灯的影响。影响因子计算如公式所示。

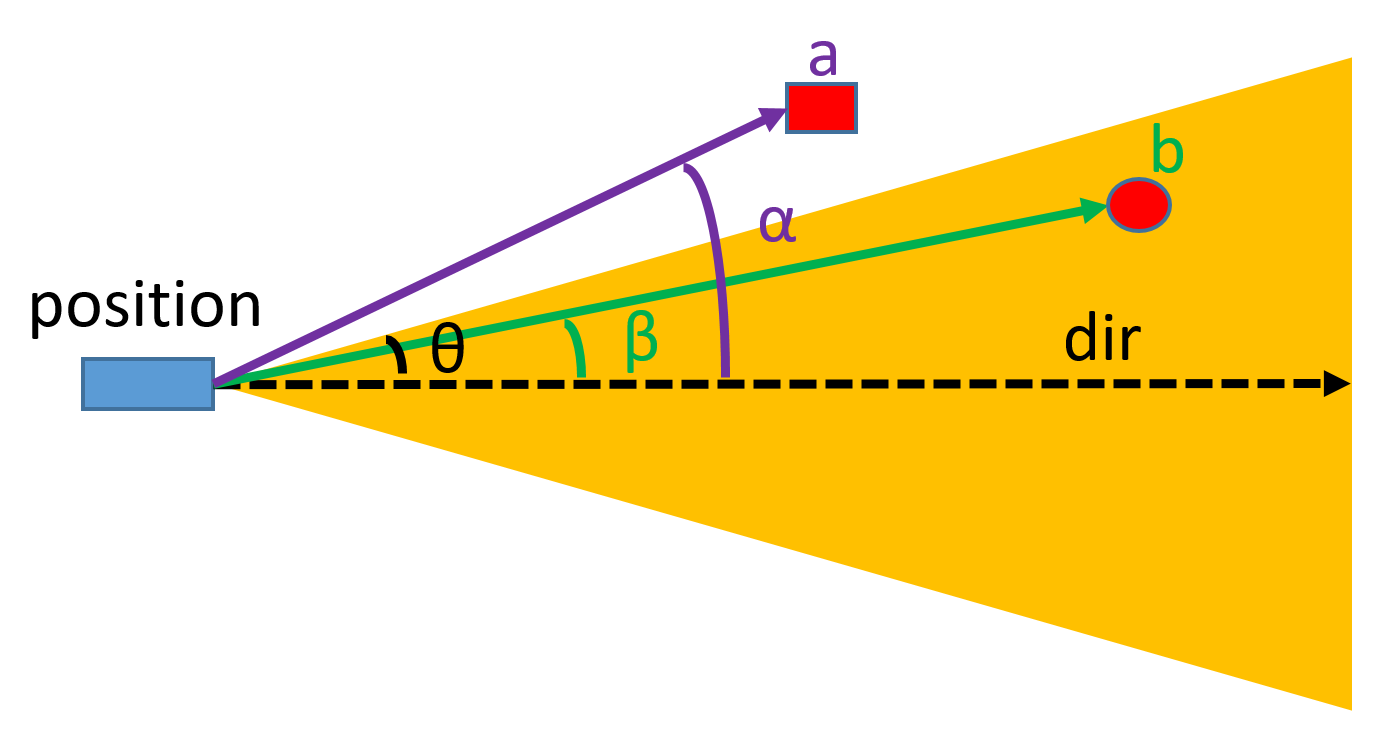
 

图 3.7 聚光灯图示

聚光灯的衰减因子和点光源有部分的相同，聚光灯的强度也会随着距离衰减，计算公式为，将聚光灯的这种衰减系数记为。 现实场景中，聚光灯照亮的区域，中心部分最亮，亮度从中心向照亮区域的边缘慢慢衰减。因为聚光灯的夹角范围一般不超过，所以边缘区域的距离和中心距离相差不大，没有办法用距离的衰减去模拟这个效果。所以在距离衰减的基础上，算法还需要考虑角度对光线强度的影响，角度的余弦值在之间是单调递减，可以满足光线的衰减分布，但是 值为1， 值为0.7071，两个角度之间余弦值的差距不大，当聚光灯角度较小时，差距更小，人眼对这种小范围的差距不太敏感，所以需要将余弦值的差距变大，对其做一个简单的线性映射，将的值映射到 ，表示聚光灯最大角度的余弦值。映射关系如公式所示。



这个公式表示，当角度的余弦值为时，经过映射得到的角度衰减因子记为，当 时，衰减值为0，当时，衰减值为1，满足要求。

聚光灯的阴影使用阴影贴图[27]生成，在这里不做赘述。由公式、公式和公式可知，聚光灯的入射光线值计算如公式所示。



#### 小结

本小结讲述了本文算法对场景中三种光源的处理方式，三种光源的处理方式各不干扰，在这个步骤中，只需要计算出每个光源对粒子所贡献的入射光线的总和供后续计算使用。这种计算量很大但是与其他步骤耦合度不高的部分可以使用计算着色器进行处理，提高渲染效率，这点会在后续章节中做详细描述。

### 单个粒子的散射计算

单个粒子的散射计算的准备工作包括阴影因子计算、密度计算、三维纹理的应用、光源入射光线的计算，这些都在前面章节有过详细的描述，这个小节主要描述怎么在计算着色器中计算单个粒子的散射，包括内散射值和消光系数，并将最终结果存放到三维纹理中。

为了数据存储的方便，本文算法将单个粒子的散射计算放在相机空间下完成，三维纹理中的每个采样点表示视线上的粒子。以128×128×64的三维为例，在相机空间下，一共投射128×128 根光线，每根光线上有64个微粒。每个微粒计算出密度、阴影值、消光系数的指数值和入射光线值。这些系数的计算流程已经在前面章节做了描述，但是密度计算使用柏林噪声进行处理时的输入参数为粒子的世界坐标系下的位置，计算阴影值时，也需要粒子的位置信息，在散射计算之前需要得到每个粒子相应的世界坐标系下的位置。

本例中使用一个128×128×64的计算着色器来计算每个粒子所需要的值。可以理解为计算着色器的每个单元和三维纹理的采样点一一对应。三维纹理每个采样点的世界坐标系的位置需要在计算着色器中计算并使用。由于从三维纹理直接映射到世界坐标系比较麻烦，所以需要通过NDC(Normalized Device Coordinate) 坐标系作为中间桥梁，将三维纹理的纹理坐标先映射到NDC 坐标系，再由NDC 坐标系转化到世界坐标系中。由图 3.3可知，三维纹理的，，轴与相机空间下的，，轴保持一致。经过投影矩阵，，轴的分量在NDC 坐标系中的映射为，轴映射为。 这个步骤是简单的线性映射，所以反映射的步骤如公式所示，公式(1) 为，轴的反映射，公式(2)为轴的反映射， 为或轴上三维纹理的大小，为轴上三维纹理的大小，为或轴上的纹理坐标，为轴上的纹理坐标。



得到屏幕空间下的坐标以后，根据屏幕坐标计算世界坐标系下的位置。屏幕坐标系下的点 的位置为，点用屏幕空间坐标系下的三个基向量、和表示,则表达式为公式。



坐标系的转换可以理解为基向量的变化，根据这个思想，我们需要找到这个点世界坐标系下基向量的三个系数，由于没有直观的映射关系，这三个系数比较难计算，可以先从相机坐标系入手。在渲染过程中，相机的输入参数有三个：相机位置信息、相机的目标位置、相机的朝向，这三个信息都是世界坐标系下的值。相机的三个基向量在世界坐标系下的表示如公式所示。

其中为轴方向，为轴方向，为轴方向，将屏幕坐标系轴的分量和轴的分量与 和相乘，可以得到世界坐标系下，当前点相对于相机位置的，轴方向的偏移和。在描述三维纹理的小节中提到，方向是指数级的变化，所以方向需要做另外的变换，变换如公式所示。



其中为屏幕坐标系下的深度值，为世界坐标系方向的分量，为三维纹理在世界坐标系下分量下的最大值。由、和的值可知，点在世界坐标系下以相机的位置为参考点，偏移量为。根据相机的位置信息可知点在世界坐标系下的位置为。得到之后，计算器着色器将计算写入三维纹理中的值：内散射值和消光系数的指数值。消光系数的指数值和介质的密度和厚度有关，将作为输入参数传入柏林噪声函数得到密度，三维纹理把介质沿方向切成块薄片，当前例子下，当前薄片的厚度为前一个粒子的值减去当前粒子的值，由公式可以求得和，和的差值即为厚度。消光系数的指数值计算如公式所示，为控制系数，控制消光的强弱。



内散射光线的值与入射光线和相位方程有关，入射光线由场景中的光源决定，各个光源对粒子的影响在章节3.3.3中有详细的描述，将场景中所有光源的入射光线值的和记为。雾的散射模型为米氏散射，散射方程为公式。为视线和入射光线夹角的余弦值，所以内散射值计算如公式所示。



计算得到内散射光线值存放在纹理的通道中，消光系数存放在通道中，每个通道的数据类型为浮点数，然后将这些值存放在三维纹理对应的坐标中。计算着色器的大小和三维纹理一致，因此计算着色器每个单元在全局工作组中的位置便为三维纹理对应的存储坐标。

## Ray-marching算法求解散射积分方程

本文算法的最后一步是求解散射方程，这是一个迭代的过程，不同视线之间的计算互不干扰，但是每一根视线线上点的计算存在数据依赖关系，所以只能使用大小的计算着色器。

在计算着色器中处理光线行进步骤的迭代计算，需要得到两个值，并将结果存放在第二个三维纹理中，第一个值是粒子内散射值的总和，第二个值为介质在不同深度下的消光系数的指数值。在前一小节，已经计算了粒子的内散射光线，但是这个光线同样要在介质中经过一段距离的传播才能到达相机，我们需要计算每个粒子内散射光线最终进入相机的内散射光线的总和。此外，上一小节计算的消光系数的指数值是单个粒子的值，这个章节需要不同厚度的介质总的消光系数指数值，这个值用来计算处于场景中不同深度的物体的反射光线对相机的贡献。

每个计算着色器单元处理一根光线上的计算，纹理每个纹素包含四个浮点型的数据，通道用来保存最终内散射光线的颜色值，通道用来保存当前深度下的消光系数指数值。具体计算步骤如下：

* 读取当前视线下第个粒子的内散射光线和消光系数指数值，从0开始，最大值不超过，为三维纹理方向的最大分量；
* 将当前点的消光系数指数值增加到一个用来存放总的消光系数的累加寄存器中，用比尔— 朗伯定律计算点的光线透射比；
* 根据透射比计算点的内散光线的透射光线，即最终达到相机的光线值；
* 将透射光线的值和消光系数累加寄存器中的值存放到三维纹理中，对应的纹理坐标，轴的分量为计算着色器的，方向的坐标值为;
* 将值增加一，并且继续从步骤一开始执行；

用图来更加清晰地阐明这个步骤，如图 3.8所示。当前步骤按图中从左向右的顺序执行，最左边的纹理是单个粒子散射计算的结果，其为输入参数。从这个三维纹理的方向对这些散射值使用计算着色器进行迭代计算。中间的图为计算着色器在方向从层到层的运算。计算的结果储存到右边的三维纹理中，三维纹理的颜色值随着的增大而增大，在视觉上的感受是雾越厚，雾的颜色越浓。

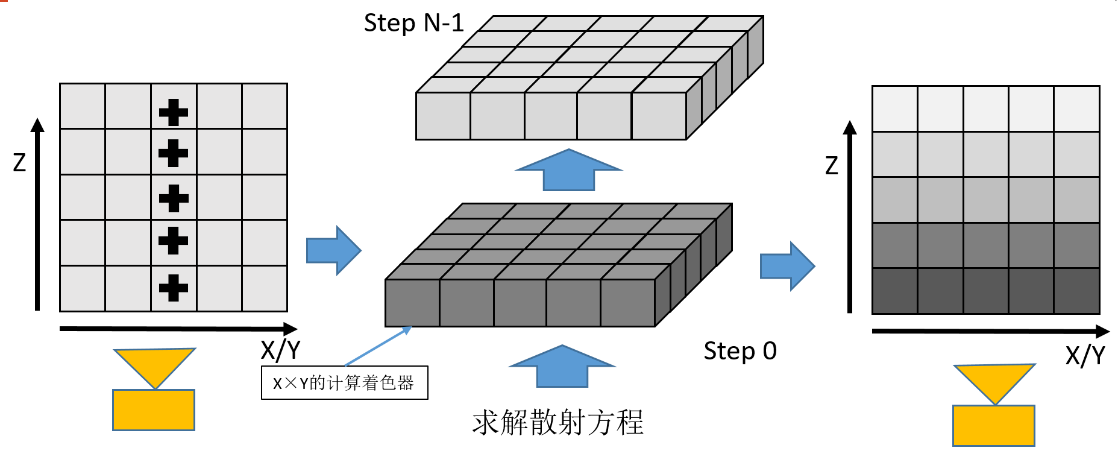


图 3.8 求解散射方程

## 级联阴影算法及优化

### 级联阴影

在计算单个粒子的内散射光线值时需要使用阴影因子，我们可以使用阴影贴图技术[28]计算阴影因子，但是普通的阴影贴图技术对分辨率要求较高，在低分辨率下走样严重，场景越大需要的阴影贴图分辨率越高。为了得到更好的雾效和体积光，我们使用级联阴影技术作为本文算法中计算阴影因子的方法。级联阴影的思想如下，将场景划分为几个不同的部分，对每个部分分别生成不同分辨率的阴影贴图，划分的依据为场景到相机的距离。级联阴影生成阴影贴图的方式如图图 3.9所示，图中红色梯形为视锥体的范围，将视锥体中的场景按照远近分为三个部分，分别生成不同分辨率的纹理，越靠近眼睛的区域生成的纹理分辨率越高。根据透视投影的原则，相同大小的物体在近处映射到屏幕上的面积比远处的映射面积大得多。距离眼睛较近的地方需要高分辨率的阴影纹理来防止阴影效果走样，离相机较远的地方并不需要高分辨率的阴影纹理，高分辨的阴影纹理反而会加重内存的负担。当屏幕上多个像素对应阴影贴图中一个像素或者屏幕上一个像素对应阴影贴图中多个像素时都会存在走样的情况，最理想的情况是屏幕上的像素与阴影贴图中的像素为1:1 。

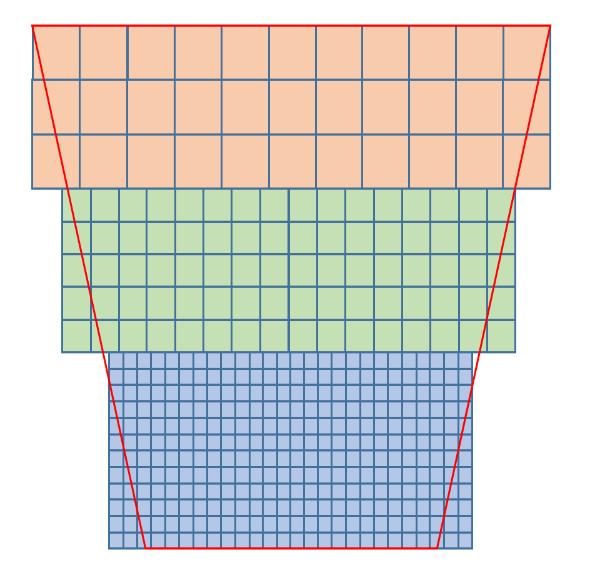


图 3.9 级联阴影图

使用级联阴影算法，渲染时每帧需要执行以下步骤：

* 将视锥体分块；
* 分别计算每个子视锥体的投影矩阵；
* 为每个子视锥体生成一张阴影贴图；
* 计算光照时，将片元投影到不同的子视锥体中，确定当前片元应该使用的阴影贴图。找到合适的阴影贴图以后，采样阴影贴图中的深度值，计算阴影因子，计算当前片元的光照值；

在大型场景中，使用级联阴影可以更加合理地分配阴影贴图的资源，让近处的物体可以有更好的阴影效果。

#### 分割视锥体

在级联阴影中最重要的部分就是如何去划分视锥体，怎样将视锥体的子块分布更加合理。因为级联阴影根据远近来处理场景中的物体，所以最常想到的一个划分办法是按视锥体中的轴方向对视锥体进行划分。视锥体的近平面的深度值为0，远平面为1，划分过程如图 3.10所示，图中最左边的图为视锥体，中间和右边两幅图分别为将视锥体分为两个部分和三个部分的示意图，红色虚线为分割线，中间的图的分割线为方向的二等分线，右边的图的分割线为方向的三等分线。在实际应用中，动态场景中每一帧都需要重新分割视锥体，渲染时分割部分的边缘会产生闪烁现象。一般做法是为每个场景生成静态级联间隔集。沿轴方向根据间隔集划分视锥体。间隔集由以下几个因素决定：

* 当前场景的深度范围。当场景的深度范围较大，则间隔集里的元素也会增多从而分割出更多的子视锥体；
* 相机与物体的距离。当相机非常靠近地面或者物体时，间隔集需要能够在近处分割更多的子视锥体以保证渲染效果；相机与物体较远时，比如航拍，则需要的更少的级联间隔集；

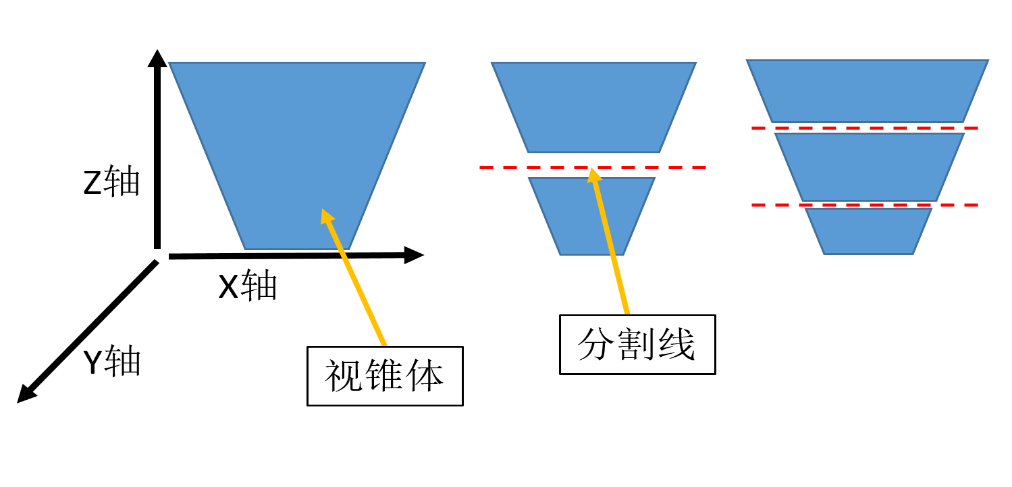
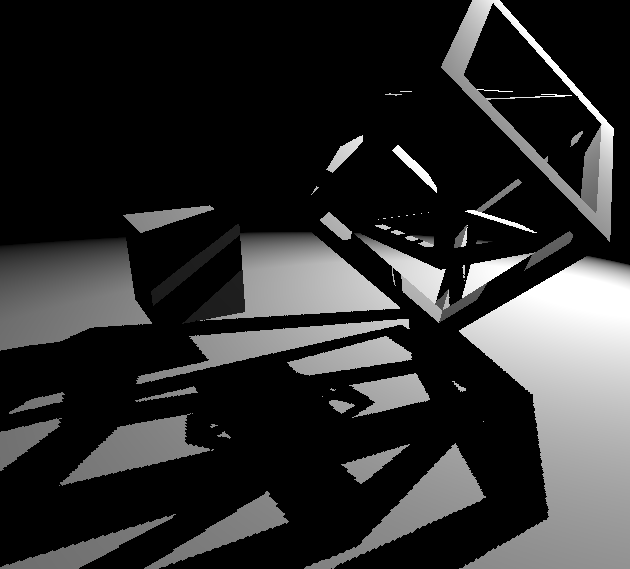


图 3.10 视锥体分割图

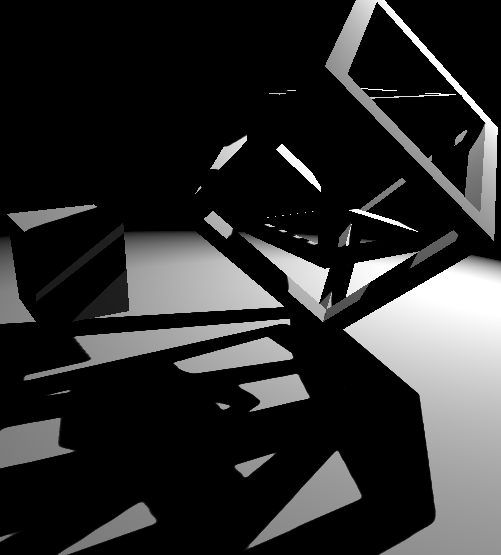
### 阴影贴图滤波

由上述算法描述可知，我们使用级联阴影算法时来生成阴影时，需要多张阴影纹理，会加大内存的负担，场景越大，这个弊端会越明显，此外，阴影贴图技术存在两个问题，第一个问题是低分辨下阴影边缘存在严重的锯齿，第二个问题是硬阴影，遮挡物离阴影很远时，阴影边缘依旧很明显。为了解决这些问题，我们使用阴影贴图滤波技术，本文算法生成低分辨率的阴影贴图，再对阴影贴图进行滤波以得到和使用高分辨率阴影贴图相似质量的渲染效果，降低了阴影贴图分辨率的纹理，可以减少渲染内存的开销，滤波以后抗锯齿问题也得到解决，并且在滤波时，阴影到遮挡物的距离是参考因素之一，所以硬阴影的问题也得到改善。

本文在国内外现状中提到很多处理阴影滤波的方式，包括VSM[15]，CSM[1]，ESM[2]，Transmittance Function Mapping[3]和Fourier Opacity Mapping[8]等。这些滤波算法解决两个方面的问题，第一个是对阴影进行滤波，减少边缘锯齿，第二个是利用距离控制阴影的软硬程度。PCF滤波技术使用多重采样和插值函数对原有的阴影贴图进行处理，多重采样会增加性能的负担，PCF不支持提前滤波，需要在阴影贴图生成以后再进行滤波操作，这两个缺点都会影响算法的效率，此外，基于采样的阴影滤波方式都会有二次走样的问题。CSM支持提前滤波，CSM在遮挡体靠近阴影时阴影贴图会出现抖动情况，阴影的渲染结果出现条纹状的波纹，VSM比CSM稳定，但是VSM需要消耗额外的内存来存储阴影深度的平方值。ESM不需要多重采样，支持提前滤波，并不需要额外的内存消耗，阴影效果也没有抖动现象，相对来说，指数阴影贴图是本文算法中最优的解决方案。



a 普通阴影贴图渲染效果



b 指数阴影贴图渲染效果

图 3.11 二维和三维Ray-marching效果图对比

下面介绍ESM的原理以及在本文算法中的应用。在阴影贴图技术中，我们计算阴影时，在阴影里时阴影因子为0，不在阴影里是阴影因子为1，图3.11(a)为普通阴影贴图技术生成的阴影效果，这种阴影通常会容易产生锯齿，边界生硬。如果我们再0 和1 之间不是梯度的过渡，而是平滑过渡，那么阴影效果会更加柔和。指数阴影贴图的产生基于这个思想，选用合适的数学函数去模拟阴影边界的过渡过程，下面内容会介绍ESM 的数学的理论基础。

我们对阴影因子计算可以表示为公式。



其中，表示阴影因子的值， 表示相机空间下的片元 在光源空间下的深度值，为映射到光源空间下的纹理坐标在阴影纹理中采样得到的深度值，为片元映射到光源空间下的屏幕坐标位置。基本阴影贴图的函数如公式所示，这是一个阶梯函数。



图中，红色表示普通阴影贴图的阶梯函数，绿色为我们需要模拟的平滑的阴影函数。在指数阴影贴图中，我们用指数函数来模拟函数，表达式如公式所示，为片元在光源空间下的深度，为阴影贴图中的值，为控制系数，控制滤波的强弱，这个函数可以近似模拟图 3.12中绿色的线，得到更加平滑的阴影，指数阴影贴图生成的阴影效果如3.11(b)所示。 

本文算法生成多张级联阴影贴图以后会使用指数阴影贴图技术对阴影贴图进行滤波，用指数阴影贴图计算阴影因子，解决走样的问题。此外，使用滤波技术还可以减少相机在漫游过程中纹理出现抖动的情况，原理如图 3.13所示。图中锯齿状的物体为产生阴影的物体，物体在阴影纹理中沿左下角往右上角移动，上面两张图是没有纹理滤波的情况，第帧阴影纹理值和第帧阴影纹理值存在明显的跳变过程。下面两张图是有纹理滤波的情况，两帧之间的阴影值平滑过度，能够缓解因为相机移动导致的阴影效果不稳定现象。

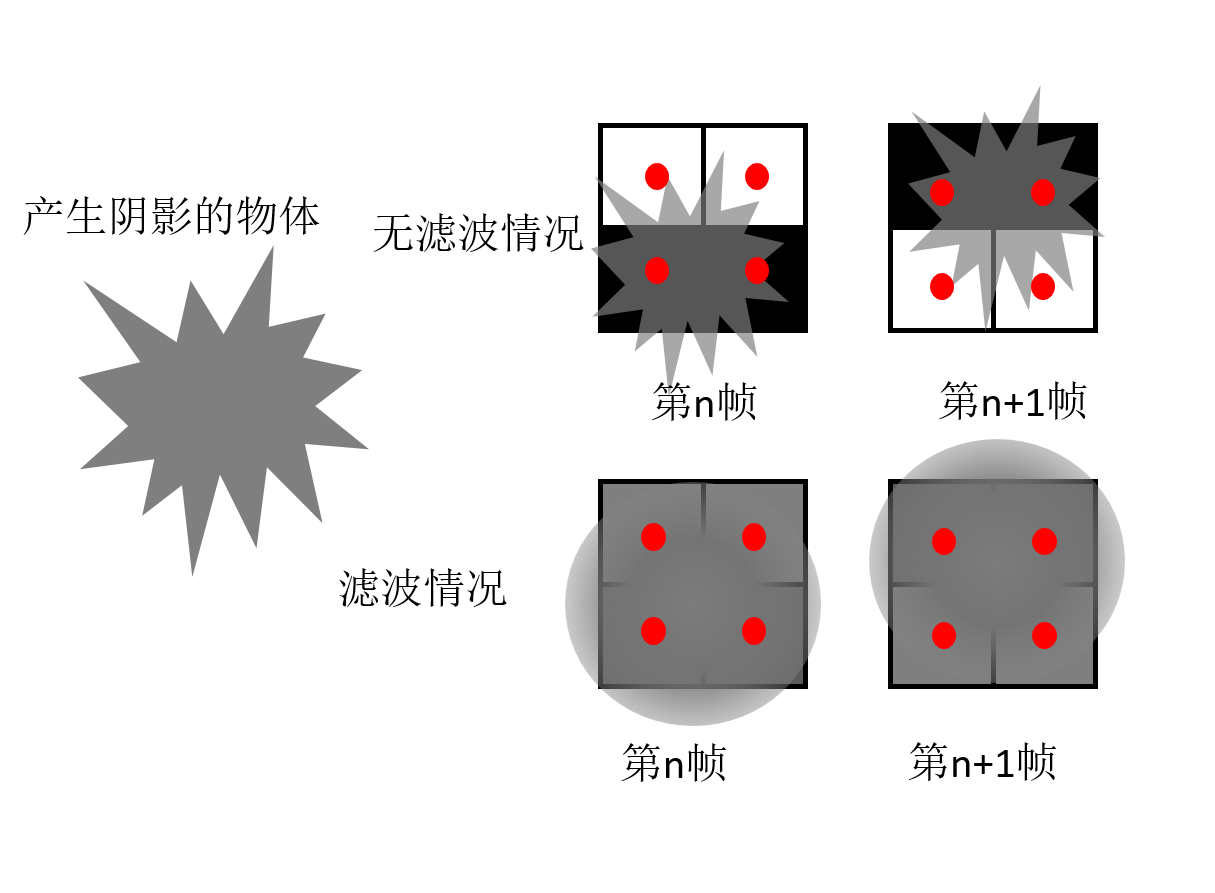


图 3.13动态场景中阴影纹理滤波与不滤波对比图

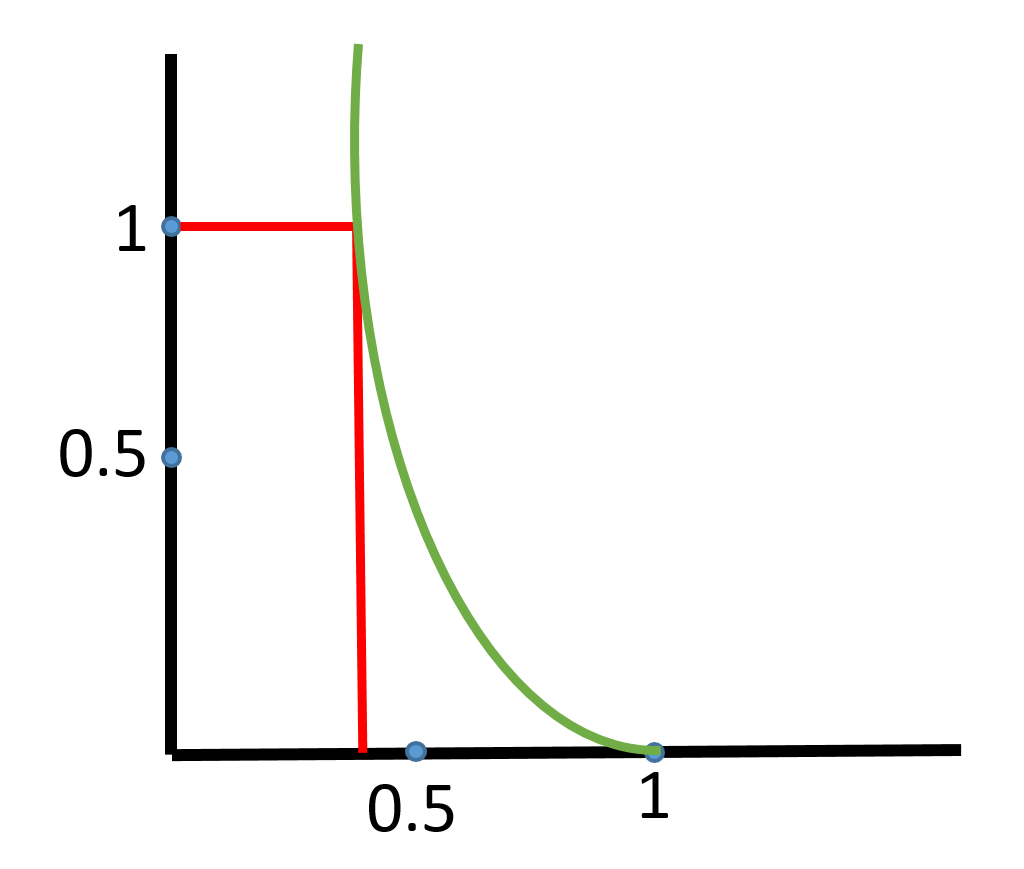


图 3.12阴影函数图像对比图

## 场景最终结果的计算

得到场景中不同深度下的内散射值和消光系数的指数值以后，后续渲染步骤就可以将散射效果应用到原始场景中形成雾效。总体的计算如公式所示。

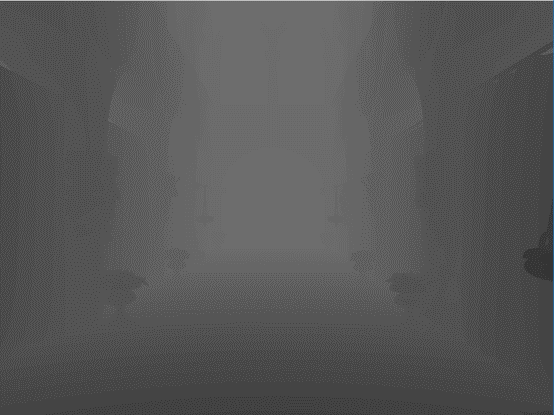


其中为场景中最终的颜色值，为场景原始的颜色值，为透射比，为内散射光线的总和。因为在三维纹理中存储的是每条视线上离散点的计算值，所以我们将效果应用到场景中时只需要考虑场景的深度在三维纹理中对应的位置，在前向着色渲染管线和延迟着色管线中都可以方便地取得场景的深度信息，所以本文算法可以兼容这两种渲染流程。

效果应用步骤在片元着色器中完成，得到场景深度之后，根据公式可以求得当前片元在三维纹理中的方向的坐标值，因为屏幕的，方向和三维纹理的，方向一致，所以只需要对屏幕坐标进行线性变换就可以得到当前片元在三维纹理中，方向的坐标值。得到当前片元的三维纹理采样坐标之后，从纹理中取出当前深度对应的内散射值和消光系数指数值，我们根据比尔—朗伯定律得到透射比，计算当前片元到达眼睛的透射光线。

如图 3.14 渲染结果展示所示，图(a)为当前场景深度下三维纹理中的散射光线的值，图(b) 为当前场景深度下每个片元的透射比，图(c)为场景的原始颜色值，图(d)为在场景中添加雾效的效果图。

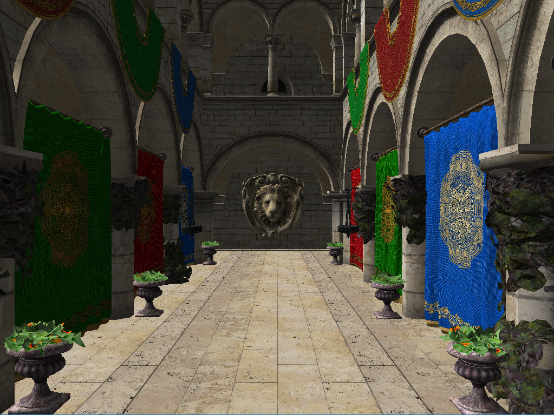
本文算法在前向渲染和延迟着色渲染中的应用比较灵活，此外对透明物体的处理也一样灵活，如所示。其中，视线的起点为不透明物体的位置，终点为相机的位置，在视线传播过程中的小长方形代表视线上的离散粒子，长方形有两部分组成，上边部分表示每个粒子的内散射光线，下边部分表示每个粒子的消光系数，这些数据存放在三维纹理中。视线的路径上有两个透明物体，本文算法在这种情况下也可以很好地应用雾效。现有的处理透明物体较为普遍的方法有Depth Peeling[14]，OIT[6]等，这些算法都是将透明物体的片元进行存储并按照深度排序，在最后渲染时有序进行混合。我们可以得到这些透明物体多层片元的深度值，然后按深度值对三维纹理进行采样，得到当前片元的雾效。图中透明物体上的灰色短箭头代表当前透明物体前后片元应该采样的三维纹理值，不透明物体的灰色短箭头代表不透明物体片元应该采样的三维纹理值，对五个片元的值进行混合，得到最终的颜色。



(a)



(b)



(c)



(d)

图 3.14 渲染结果展示

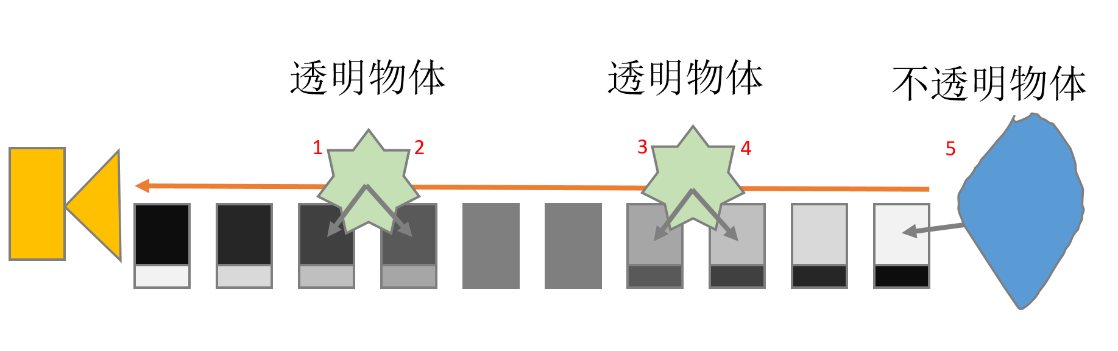


图 3.15 存在透明物体的场景下算法的应用

## 本章小节

本章首先对本文算法的核心思想进行了概述，对算法的具体步骤进行描述，然后对算法中阴影处理，散射计算和主要数据结构做了详细的介绍，此外还介绍了多光源场景下散射计算的预处理步骤，最后介绍如何将本文算法生成的效果与现有的渲染管线和特殊的场景相结合。

# 实验结果及分析

## 实验概述

本文采用改进的Ray-marching算法和级联阴影算法实现体积光和体积雾的模拟。通过模拟光在介质中传播的过程，进行相关的光照计算得到雾和体积光的效果。本文使用计算着色器提高算法的效率，并使用ESM算法优化级联阴影算法提高场景的雾效和体积光效果。本文从不同的粒子密度，不同复杂度的场景作为本文算法的实验数据，从效率和效果两个方面进行对比分析，全面验证本文算法渲染体积雾和体积光的真实性和高效性。

## 实验环境

本章实验中，对体积光和体积雾的模拟，包括对每个实验场景的运行和数据采集都在如表 4.1所示的软硬件环境上完成。

表 4.1 实验软硬件环境

|  |  |  |
| --- | --- | --- |
| 硬件环境 | CPU | Intel(R) Core(TM) i3-4160 @3.60GHz |
| 内存 | 8GB |
| 显卡 | NVIDIA GeForce GTX 960 |
| 软件环境 | 操作系统 | Window 10 Pro |
| 编译器 | Visual Studio 2015 |
| 开发库 | Boost-1.56；glfw-3.1.2；glm-0.9.9 |
| 辅助工具 | 3ds Max 2014 |

## 实验设计及结果分析

### 实验参数

本文体积雾和体积光模拟算法的主要参数设定，若本章实验无特别说明，均采用如表 4.2所示的默认参考值。

表 4.2 主要参数设定值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ScatCoeff | AbsorCoeff | FogDepth | FogPhaseAniso | ScreenRes |
| 0.03 | 0.01 | 10000 | 0.2 | 1280\*720 |

### 不同密度的体积雾和体积光渲染效果展示

实验目的：本文算法将密度作为散射光线计算的输入参数，不同的密度分布可以模拟出不同形态的体积光和体积雾。本文支持的密度输入在章节3.3.1中有详细的介绍，通过渲染不同密度分布下的雾效和体积光，证明本文算法可以支持多种形态的雾和体积光的处理。

实验输入数据：输入三组不同密度，第一组，均匀密度的雾，本次实验生成两种均匀密度的雾，第一种密度为0.8， 第二章密度为1.0；第二组，密度分布由数学函数模拟，数学函数如所示，参数。。控制雾的衰减强度，密度分布的主要影响因素为高度；第三组，生成三维柏林噪声纹理干扰雾的分布，纹理的输入参数如表所示，纹理图像如图所示。在本次实验中，其他默认参数如表 4.2所示。

结果输出：均匀密度的雾如图\ref{pic-avgDenRe}所示，函数模拟的密度分布生成的雾如图\ref{pic-funexpRe} 所示，由不同三维柏林噪声生成的雾如图\ref{pic-perlinRe}所示。

实验结果分析：本文实现了三种不同密度分布的雾和体积光的渲染，第一种为均匀密度，雾和体积光的渲染效果各项同性，雾的浓度只和深度有关，相同密度下，每等厚层的雾的浓度相同；第二种雾的效果和公式\ref{eq:Density}有关，总体分布为随着高度的上升，雾的密度越小，浓度越低。系数$D$为400时，雾的浓度随着高度的衰减更加缓和，系数$D$为200时，雾的浓度随着高度的衰减更加剧烈；第三种雾的密度分布受柏林噪声纹理的干扰，柏林噪声纹理的分布主要受三个参数影响，频率参数控制柏林噪声的波动情况，频率越大，波动越剧烈，振幅参数控制柏林噪声数值的波动范围，为了让密度分布看起来更加自然，我们会使用几个柏林噪声函数混合得到最终的结果，倍频参数控制柏林噪声函数计算的次数，倍频越大，计算次数越多，频率和振幅因为多个函数的叠加也会有所上升，密度分布更加无序。通过不同密度分布的场景渲染效果对比可知，用柏林噪声纹理干扰雾的分布后，雾的效果看上去更加自然。

### 不同光源下体积雾和体积光的渲染效果展示

实验目的：

实验输入数据：

结果输出：

实验结果分析：

### 带有透明物体的场景的雾效和体积光的渲染

实验目的：

实验输入数据：

结果输出：

实验结果分析：

### 使用计算着色器和片元着色器渲染效率对比

实验目的：

实验输入数据：

结果输出：

实验结果分析：

### 本文算法阴影算法和常见阴影算法效果对比

实验目的：

实验输入数据：

结果输出：

实验结果分析：

### 本文算法局限性分析

## 本章小节

# 总结与展望

* 1. 总结

本文主要针对雾和体积光的实时渲染这一研究课题进行了深入的研究，通过对比分析近几年关于雾和体积光渲染的国内外研究成果的优缺点，总结并归纳出基于物理的雾和体积光的实时渲染的重难点，将散射光照计算部分从Ray-marching算法中解耦，并且使用计算着色器去加速光照的计算部分，使用指数阴影贴图算法去优化级联阴影贴图的效果，从而提高体积光的渲染效果。本文实现了一种基于物理的雾和体积光的实时渲染算法。本文研究成果如下：

1. 对雾和体积光的物理现象进行建模，散射现象的计算量非常大，本文使用单次散射物理模型作为雾和体积光模拟的基础模型。使用Ray-marching算法求解散射方程，将单个粒子的散射光照计算和求解散射方程步骤解耦，使用计算着色器进行单个粒子散射光照的计算，这个步骤可以提高本文算法的效率。
2. 多光源场景中，每个光源对粒子的影响同样在单个粒子散射计算步骤中进行，将光源的参数传入计算着色器，并行处理每个粒子的多光源光照，这个操作使得本文算法在多光源场景中可以提高效率，减少光源的预处理步骤。
3. 本文算法在将雾效和体积光应用到场景的步骤中，根据透明物体或者不透明物体的片元深度，在三维纹理中采样得到当前片元的效果颜色值。因为在前向着色渲染管线、延迟着色管线以及处理透明物体的算法中，片元的深度都方便得到，所以这个特点提高了本文算法的兼容性。
4. 在效果方面，因为纹理分辨率低而产生的走样结果在雾和体积光的渲染中比较常见，考虑到内存消耗，本文使用指数阴影贴图技术对阴影贴图进行滤波，减少走样问题，并且针对大规模场景的渲染，本文选用级联阴影技术提高大规模场景的效果。
   1. 展望

本文基于物理的体积雾和体积光的实时渲染算法，对原有的Ray-marching和阴影贴图算法进行改进，使得本文算法满足实时渲染的效率要求并且改善了雾和体积光的效果，但从这两方面来看，本文算法仍存在一定的不足之处，其改进的方向是未来研究工作的重点。未来研究方向主要包含以下几个方面：

1. 采用多次散射模型模拟雾和体积光，并且能够保证实时渲染效率。
2. 使用纹理压缩技术处理纹理，减少算法内存。
3. 支持更多密度分布的雾和体积光的渲染。
4. 统一处理场景中光源的阴影贴图，提高渲染效率。

# 参考文献

1. Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution Shadow maps. In Proceedings of the 18th Eurographics conference on Rendering Technique, pages 51-60. Eurographics Association, 2007.
2. Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In Proceedings of graphics interface 2008, pages 155–161. Canadian Information Processing Society, 2008.
3. Cyril Delalandre, Pascal Gautron, Jean-Eudes Marvie, and Guillaume François. Transmittance function mapping. In Symposium on Interactive 3D Graphics and Games, pages 31–38. ACM, 2011.
4. Rouslan Dimitrov. Cascaded shadow maps. Developer Documentation, NVIDIA Corp, 2007.
5. Julie Dorsey. Radiosity and global illumination. The Visual Computer, 11(7):397–398, 1995.
6. C. Everitt. Interactive order-independent transparency. Technical Report, 2002.
7. Fan Guo, Jin Tang, and Xiaoming Xiao. Foggy scene rendering based on transmission map estimation. International Journal of Computer Games Technology, 2014:10, 2014.
8. Jon Jansen and Louis Bavoil. Fourier opacity mapping. In Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, pages 165–172. ACM, 2010.
9. Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The beam radiance estimate for volumetric photon mapping. In ACM SIGGRAPH 2008 classes, page 3. ACM, 2008.
10. James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. In ACM Siggraph Computer Graphics, volume 18, pages 165–174. ACM, 1984.
11. J.T. Kajiya. The rendering equation. ACM SIGGRAPH Computer Graphics, 20(4):143–150, 1986.
12. Anton Kaplanyan. Light propagation volumes in cryengine 3. ACM SIGGRAPH Courses, 7:2, 2009.
13. J. Kopf, M.F. Cohen, D. Lischinski, and M. Uyttendaele. Joint bilateral upsampling. ACM Transactions on Graphics, 26(3):96, 2007.
14. A. Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Computer Graphics & Applications, 9(4):43–55, July 1989.
15. Kevin Myers. Variance shadow mapping. Retrieved May, 11:2011, 2007.
16. Holly Rushmeier. Rendering participating media: Problems and solutions from application areas. In Photorealistic Rendering Techniques, pages 37–59. Springer, 1995.
17. Tiago Sousa. Crysis next gen effects. In Game Developers Conference, 2008.
18. Balàzs Tôth and Tamás Umenhoffer. Real-time volumetric lighting in participating media. In Eurographics (Short Papers), pages 57–60. Citeseer, 2009.
19. Nathan Vos. Volumetric light effects in killzone: Shadow fall. GPU Pro, 5:127–147, 2014.
20. Carsten Wenzel. Real-time atmospheric effects in games. In ACM SIGGRAPH 2006 Courses, pages 113–128. ACM, 2006.
21. Magnus Wrenninge, N Bin Zafar, J Clifford, G Graham, D Penney, J Kontkanen, J Tessendorf, and A Clinton. Volumetric methods in visual effects. ACM SIGGRAPH 2010 Courses, 2010.
22. Chris Wyman and Shaun Ramsey. Interactive volumetric shadows in participating media with single-scattering. In Interactive Ray Tracing, 2008.
23. Egor Yusov. High performance outdoor light scattering using epipolar sampling. GPU Pro, 5:101–126, 2014.
24. David P Luebke. Level of detail for 3D graphics. Morgan Kaufmann, 2003.
25. K.Perlin. An image synthesizer. ACM Computer Graphics, 19(3):287{296, 1985.
26. Ken Perlin. Improving noise. In ACM Transactions on Graphics (TOG), volume 21, pages 681{682. ACM, 2002.
27. Tom Lokovic and Eric Veach. Deep shadow maps. In Proceedings of SIGGRAPH '00, Computer Graphics Proceedings, Annual Conference Series, pages 385{392. ACM SIGGRAPH, 2000.
28. Marc Stamminger and George Drettakis. Perspective shadow maps. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002), pages 557-562, 2002.

# 作者在读期间的科研成果

【攻读硕士学位期间发表的论文】

[1] 孟蓉蓉. 基于链表的实时屏幕空间反射技术[J]. 现代计算机上旬刊, 2018(2):40-43.

# 声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得四川大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

本学位论文成果是本人在四川大学读书期间在导师指导下取得的，论文成果归四川大学所有，特此声明。

研 究 生：

指导教师：

日 期： 年 月 日

# 致谢

时光荏苒，岁月如梭，本人学生生涯已然接近尾声。回顾三年来在硕士研究生阶段的学习生活，有欢乐，也有苦闷，有得意之时，也有失落不得志。这篇论文从最初选题，经过资料搜集、算法研究、系统实现、实验验证等过程，最终能够顺利完成文章，离不开众人给予的无私帮助与热切关怀。在这里，我要向这些一路陪伴的人道一声感谢。

首先，我要感谢我的导师刘健波老师在研究生期间对我学习和生活上关心和帮助，让我能够快速适应研究生阶段学习生活节奏，消除迷茫，老师的不断敦促和指导使我得以顺利完成研究生阶段的学业课程和学术研究工作。同时要感谢实验室张严辞老师，不仅在我论文选题的时候能够站在专业的角度帮助我进行研究难点的分析，最终顺利定下题目，并且能够仔细、耐心地帮助我修改论文，使得我能够及时改正思考上和行文上的缺陷之处。本人能够顺利完成毕业设计离不开两位老师的付出与关怀，老师踏实严谨的学术作风，以及乐观积极的处事态度都使我受益匪浅。

还要感谢计算机学院的所有老师的教导与帮助，特别是实验室吴志红老师和李梅老师，为我们提供了舒适的科研环境和坚实的后勤保障。毕业设计的完成也要感谢实验室伙伴们的无私帮助，陆应通，余瀚游、谢瑞麟、任文婕，谢凯丽，全伟衡在论文的结果渲染和实验DEMO制作环节提供了宝贵的建议和帮助。

此外，感谢我的父母在三年期间对我学习和生活上的关心。父母不仅为我提供了能够舒适生活、安心学习的经济基础，并且每当我情绪失落时，总能为我排解负面情绪，让我重新恢复斗志。

最后，感谢在百忙之中对本文进行审阅和指正的各位专家学者！

铭刻恩情，勿忘初心。前路漫漫，虽万千里吾亦往矣！