| 学号 | 姓名 | 论文规范性（10） | 问题分析与调研（30） | 方案创新性（20） | 实验结果分析与讨论（40） | 结课论文总成绩（100） |
|---|---|---|---|---|---|---|
| 21301152 | 贺昱欣 | 8 | 27 | 18 | 35 | 88 |

# 关于游戏中体积光的四种实现方法的研究

Research on Four Implementation Methods of God Rays in Video Games

学　　院：软件学院

专　　业：软件工程

学生姓名：贺昱欣

学　　号：21301152

北京交通大学

2024 年 6 月

# 中文摘要

体积光，俗称“光柱”或“神光”，是现代游戏视觉效果中的重要组成部分。这种效果通过模拟光线在空气、雾气、烟尘等介质中的传播和散射，使光线的路径可见，从而增强场景的氛围和真实感。本文对四种常见的体积光实现方法进行了研究和实验，分别是：BillBoard 特效贴片、径向模糊、Volume Shadow（光方向挤出）和 Ray-Marching（光线追踪）。BillBoard 贴片技术简单高效，但缺乏真实感；径向模糊通过后期处理生成体积光效果，适用于光源在视野内的场景；Volume Shadow 方法通过沿光方向挤出顶点来计算阴影体积，能较好地模拟光的散射；Ray-Marching 技术通过逐步计算光线在介质中的传播，生成高度逼真的体积光效果，尽管计算开销大，但能够处理复杂的光照场景。本文详细介绍了每种方法的原理，并通过实验对它们进行了对比分析，展示了不同方法在性能和视觉效果上的差异。最终，本文讨论了这些方法的适用性和在实际游戏开发中的应用场景。

**关键词：** 体积光，光柱，BillBoard 特效贴片，径向模糊，Volume Shadow，Ray-Marching，光线追踪

# ABSTRACT

**Abstract:**

God rays, commonly referred to as "volume light" or "light shafts," are a crucial visual effect in modern video games. These effects simulate the visible paths of light as it travels through mediums like air, fog, and smoke, enhancing the atmosphere and realism of scenes. This paper explores and experiments with four common methods for implementing god rays: BillBoard Sprite Technique, Radial Blur, Volume Shadow Extrusion, and Ray-Marching based on ray tracing. The BillBoard Sprite Technique is straightforward and efficient but lacks realism; Radial Blur applies post-processing to create volume light effects suitable for scenarios where the light source is within the view; Volume Shadow Extrusion calculates shadow volumes by extruding vertices along the light direction, effectively simulating light scattering; Ray-Marching, through incremental calculation of light propagation in a medium, produces highly realistic god ray effects, albeit with high computational costs, making it suitable for complex lighting scenarios. This paper elaborates on the principles of each method and compares them through experiments, demonstrating their differences in performance and visual quality. Finally, the paper discusses the applicability of these methods and their use in practical game development scenarios.

**KEYWORDS:** God Rays, Volume Light, BillBoard Sprite, Radial Blur, Volume Shadow, Ray-Marching, Ray Tracing

# 目　　录

# 1 引言

体积光通俗来说是我们能看见的”光路“，它是光照到大气中粒子散射后得到的效果（丁达尔效应）。如果我们用体素的思想来考虑体积光，某一点处的体积光颜色是眼睛到当前点的射线上，光路中所有粒子散射光的叠加。所以，通过物理方法模拟光线透过空气中的尘埃，雾气，烟尘等粒子形成光柱效果的计算十分复杂。

要在游戏中模拟这种现象，不太可能完全按照现实世界中的方式去做，需要非常非常大量的粒子，对于设备的计算压力太大了。所以，游戏中实现体积光，就是在需要的地方，能显示出一道光线就好了。

体积光特效在现今的中高端游戏中非常常见，优秀的体积光特效在烘托游戏氛围，提高画面质感方面发挥了很大的作用。

早期游戏中由于机能限制经常使用的是 BillBoard 贴片和径向模糊这两种方式，这些方法实现简单，但缺乏真实感，难以支撑复杂的情况。所以，伴随着渲染技术的进步，业界已经开始使用基于光线追踪、阴影贴图等更为精细的渲染技术来实现体积光的效果，这些方法能够实现高精度渲染，处理复杂的光照和多光源场景，从而呈现更加优质的场景表现。

本文主要介绍并通过实验复现改良了以下几种实现方式，BillBoard 特效贴片，径向模糊，Volume Shadow 沿光方向挤出顶点，Ray-Marching 基于光线追踪。几种方式殊途同归，都是尽可能用最省的消耗来近似模拟这一酷炫的现象。

# 2 实现原理

## 2.1 BillBoard 贴片

BillBoard 贴片就是生成一个随机的明暗条文，加上遮罩，让它看起来有光条的感觉，如图 2-1。然后将贴片放置在场景中光线会泄露出来的区域，这就是最简单的体积光效果。做得精细一点的会再加上 UV 动画，粒子和远景透明的效果。这种技术不需要复杂的几何计算或光照模型，对处理器和内存的要求低，非常适合资源受限的环境。易于实现和集成。但缺乏真实感：无法准确模拟复杂的光散射和阴影效果，视觉上不如更复杂的方法自然。

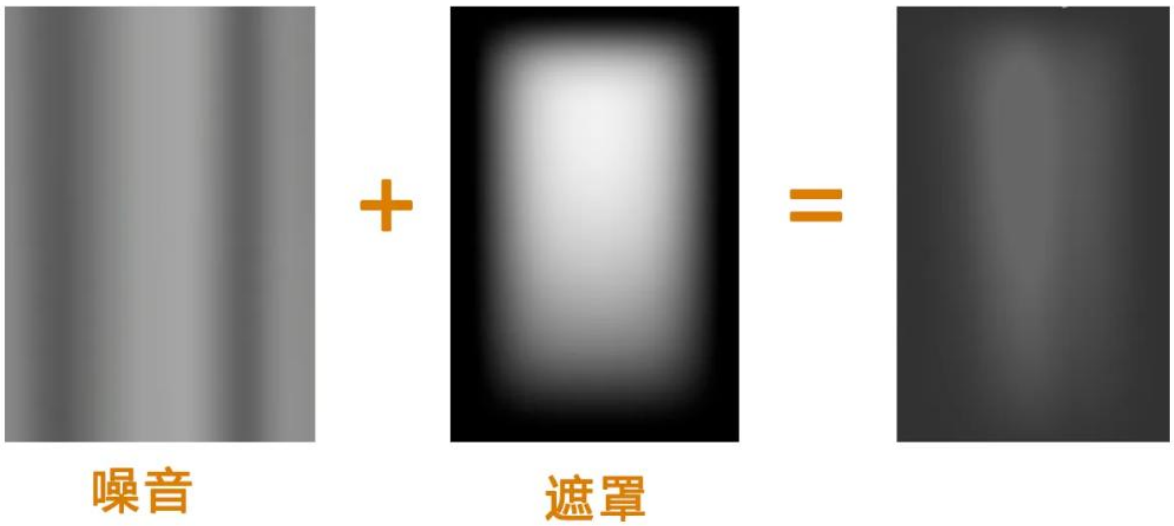



图 2-1 BillBoard 贴片

## 2.2 径向模糊

径向模糊是一种后处理的方法。主要用来表现天空中日月星光散射的效果。所谓后期处理就是在游戏画面渲染完毕之后，另外加一次渲染，类似于 PHOTOSHOP，但处理的对象是每一帧游戏画面，因为速度要求多使用 GPU 计算。

径向模糊实现体积光主要流程如下：

1：渲染出整个画面

2：抽取出画面中高亮的部分

3：对高亮部分进行径向模糊

4：将径向模糊后的高亮层和原图合并

径向模糊的 GPU 实现就是从原本像素的位置开始，向画面中心移动坐标，每移动一次就采样一次，将所有采样叠加在一起，如图 2-2。

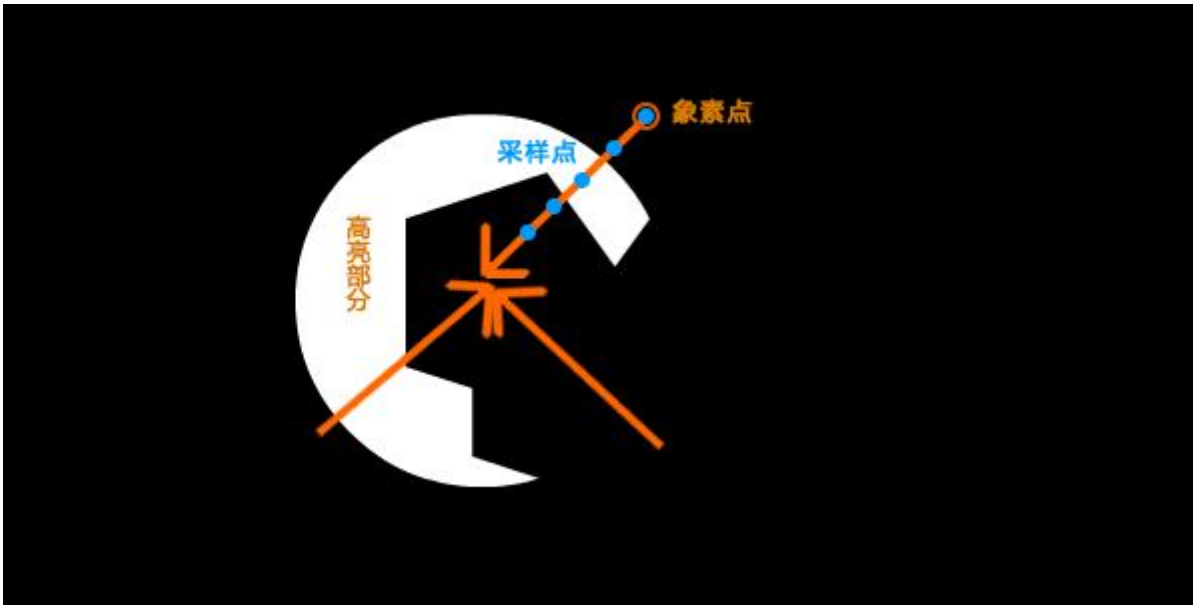



图 2-2 径向模糊采样示意图

因为实现简单，消耗小，效果也不错，径向模糊在很多游戏中都有广泛应用。但径向模糊要求光源在视野里，不然就无法进行采样，所以他无法支持复杂的场景。

## 2.3 Volume Shadow 光方向挤出

Volume Shadow 光方向挤出（也称为阴影体积挤出）这个方法主要的实现思想是阴影的一种实现-体积阴影的扩展，是一种在三维计算机图形学中用于渲染光线与物体相互作用的技术，特别是用于模拟体积光和阴影效果。这个方法通过将物体的几何体沿光源方向挤出，生成一个阴影体积，这个体积可以用来计算光线在这些体积内的传播和散射，进而模拟出真实的体积光效果。

Volume Shadow 光方向挤出实现体积光主要流程如下：

1. 检测物体的几何边缘，这些边缘是在光线传播过程中产生阴影的部分。

2. 将这些边缘沿光源方向拉伸（挤出），形成一个阴影体积。挤出的距离通常取决于场景中的光源位置和视图的远端范围。

3. 生成阴影体积，阴影体积由挤出的边缘构成，包围了光线不能直接照射的区域。

4. 使用光线步进算法，沿着光线方向逐步计算光的强度和散射效果，累加光线的散

射和吸收，最终得到体积光效果。

5. 最终使用阴影体积来决定渲染时哪些区域被光线照亮，哪些区域被阴影覆盖。

## 2.4 Ray-Marching 光线追踪

Ray-Marching 是一种基于光线追踪技术的体积渲染方法，它通常用于渲染复杂的体积光效果，如雾气、烟雾和云等场景中的光柱。

Ray-Marching 技术的核心思想是从眼睛向场景发出许多条射线，对于每条射线而言，我们截取落在体积光中的线段，并每次推进一定步长在线段中进行位置采样，并做散射光强的计算，最终把所有结果加在一起，得到当前位置的光强，如图 2-3（c）所示。

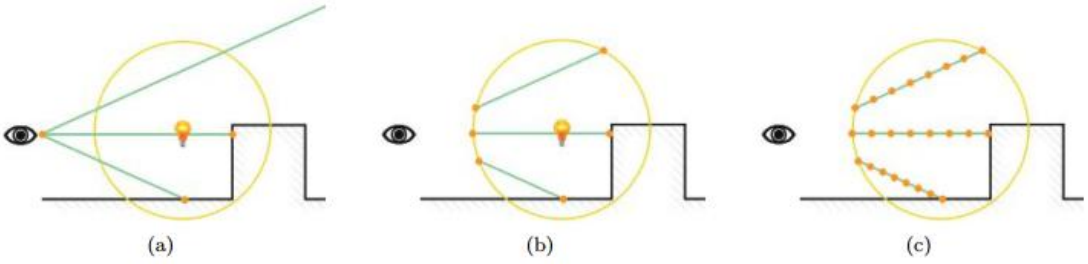



图 2-3 Ray-Marching 的实现原理

Ray-Marching 实现体积光主要流程如下：

1. 从摄像机视点出发，沿每个像素的方向投射一条光线。对于每条光线，沿其方向进行步进，计算光线在每个步进点的光照和散射情况。

2. 在光线方向上，以固定的步长（步进距离）进行移动，每一步记录光线在该点的光强度和散射效果。

3. 在每个步进点，计算光线的吸收和散射情况，并将其累加到光线的最终光照值中。

4. 建立光散射模型，使用适当的光散射模型来模拟光线与介质粒子（如尘埃、雾气等）的交互。

5. 最终，将累积的光照值应用到像素上，生成最终的图像。

Ray-Marching 基于光线追踪，可以精确模拟光线在体积介质中的传播和散射。但计算开销大，尤其在高分辨率和高密度体积介质的情况下，对实时性能要求高。

# 3 实验步骤

## 3.1 实验平台

| | |
|---|---|
| Unity 版本 | Unity 2021.3.36f1 (LTS),2018.3.0b5(光线追踪使用) |
| 硬件配置 | 处理器：Intel Core i7-9700K<br>显卡：NVIDIA GeForce RTX 2070<br>内存：16GB DDR4<br>操作系统：Windows 10 64-bit |
| 开发工具 | Unity Editor<br>Visual Studio 2019<br>Shader Graph<br>Post-Processing Stack v2 |
| 场景设置 | 场景包含一个光源、几个简单的几何体（如立方体、球体）和一些模拟的雾气或尘埃粒子。 |

## 3.2 BillBoard 贴片实现

这是实现体积光最简单的方法，直接在需要有体积光的地方，放一个特效片，模拟一个光效，就可以实现初步的体积光，如图 3-1。

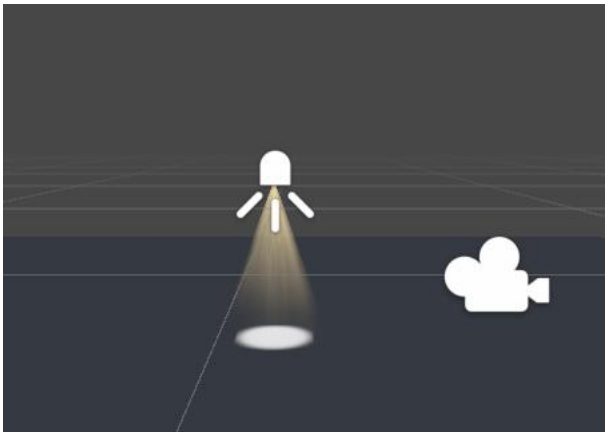


图 3-1 贴图的体积光

还可以通过 Unity 自带的粒子系统，控制粒子贴图采样 uv 变换，以及颜色的 alpha 变换，模拟灯光摇曳的状态，效果如图 3-2.

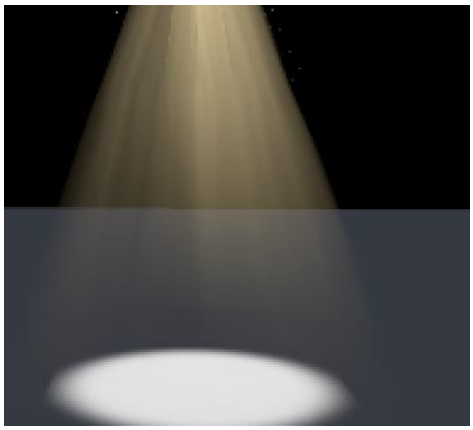


图 3-2 加了粒子效果的体积光

这是实现体积光最简单的方法，unity 只需要贴找好的图然后操作一下默认设置就可以了，实现起来十分快速。

## 3.3 径向模糊

径向模糊说到底就是对贴图进行操作，一个原始的图像将光源进行径向模糊后，有了如丁达尔效应一般的光束，这个处理比较方便，并且不占用太多性能，就是在贴图的时候对图片光源（亮的地方）都加了如同光束般的模糊，如图 3-3。

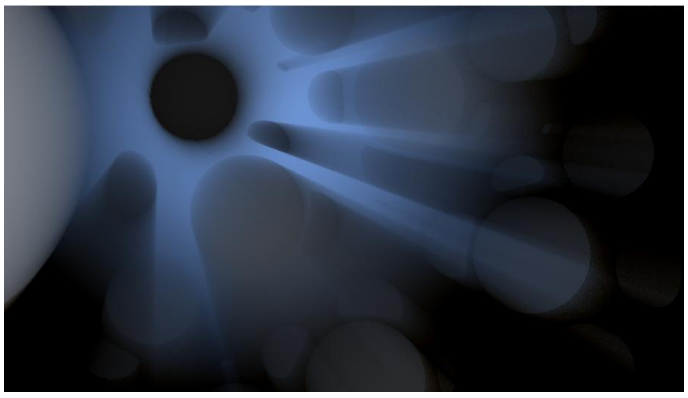


图 3-3 径向模糊示意图

所以我在 unity 里径向模糊了一张图片并贴在天空盒内，代码使用 threshold 过滤亮度值，获取高亮像素后进行径向模糊，效果如图 3-4。

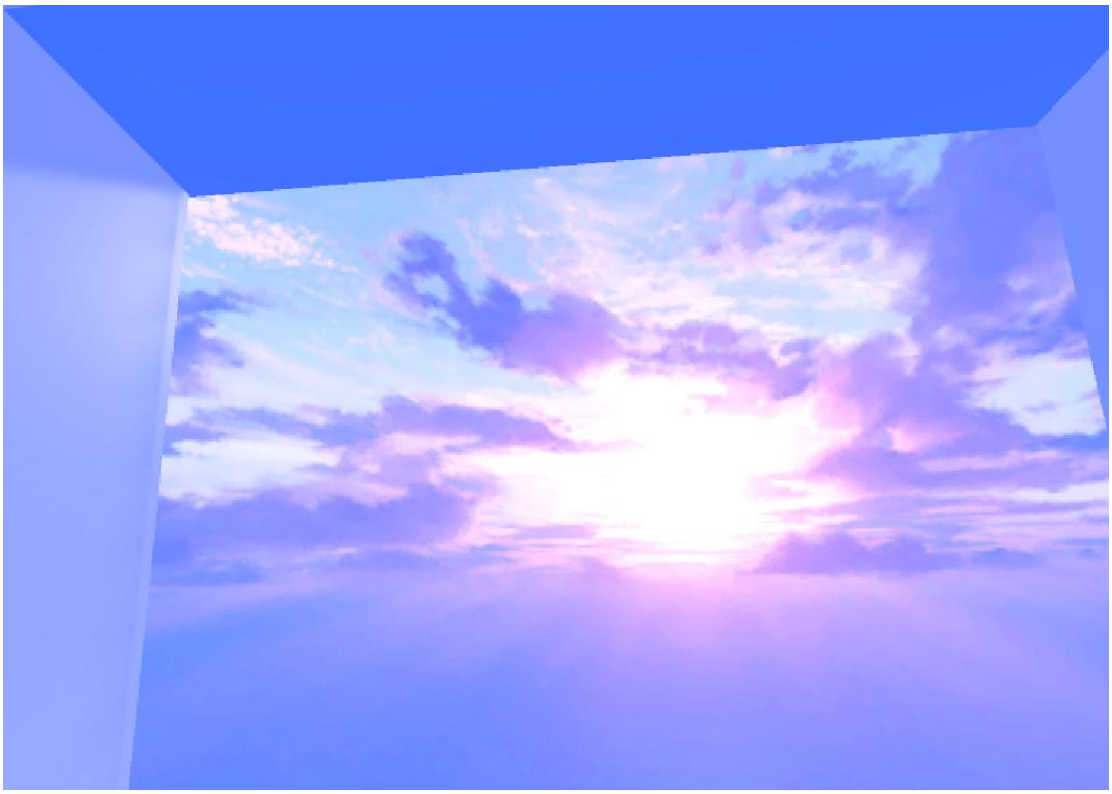


图 3-4 径向模糊结果展示

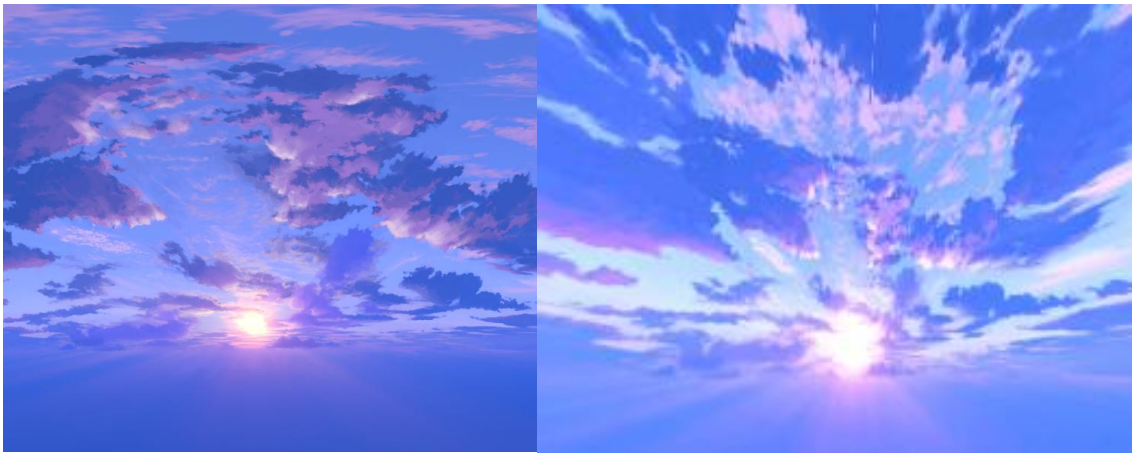


图 3-5 径向模糊实现效果对比

图 3-5 是照片对比，对比实现径向模糊后的效果，可以明显感觉光感更加强烈了，光更加有质感了。

## 3.4 Volume Shadow 沿光方向挤出顶点

这次我们实现的是平行光。

首先，我们需要确定只有受光面才沿着光方向挤出，所以这个时候就要想起 diffuse 的计算方式，也就是 Lambertian 模型中的漫反射计算，如下公式，直接用法线方向点

$$I_d = \mathbf{N} \cdot \mathbf{L}$$

乘光线方向。

这里我们直接把世界空间光位置转到模型空间进而计算了模型空间的光方向。点乘的结果就代表了光方向与法线方向的贴合程度，我们通过 sign 函数直接把这个值变成一个-1,1 的控制值，然后再进行一个最常见的*0.5+0.5 变换，-1,1 变化为 0,1。这样这个点乘结果就可以作为我们判断是受光面还是背光面的控制值。因为只用挤出受光面，所以我们将获得的物体受光面的每个顶点沿着光的反方向增加一个偏移值，就达到了“挤出”的效果。

关于顶点偏移，在描边效果以及溶解效果也都有使用。上面的操作都是在顶点阶段进行，在片元阶段，我们只需要采样一下贴图，并且正确渲染就好了，实现结果如图 3-6 所示。

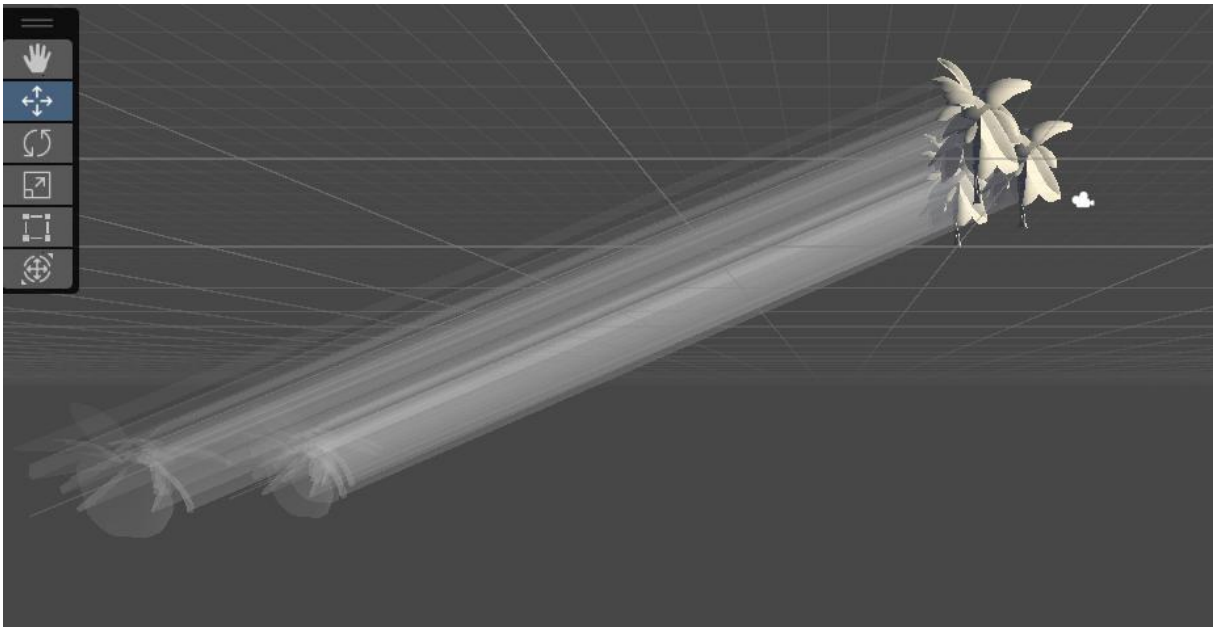


图 3-6 挤出体积光效果

## 3.5 RayMarching 光线追踪

本实验实现了一个比较简单的 Point-Light 的体积光效果。

我们使用延迟渲染进行实验，在延迟渲染的框架中，我们可以在 g-buffer 的后处理过程中完成计算散射光强这一过程。在不考虑任何优化的情况下，我们逐像素做这件事（一般情况下，我们会做降采样）。

在具体实现中，我们通过相机视锥体边角反推全屏幕像素点对应的世界位置再进行 Ray-Marching 来实现这一处理。

点光源本身是有一个衰减的，而这个衰减计算并没这么简单，Unity 的点光源衰减计算，实际上是采样了一张衰减贴图，首先计算当前位置距离点光源的距离平方，然后除以光源范围的平方，结果去采样衰减图，所以最基本的 Ray-Marching 实现如图 3-7。

```
//点光源体积光RayMarching
float4 RayMarching(float3 rayOri, float3 rayDir, float rayLength)
{
    //ori：相机位置
    //raydir：从相机到当前像素点对应世界坐标值的方向
    //rayLength：长度

    //步进值
    float delta = rayLength / RAYMARCHING_STEP_COUNT;
    float3 step = rayDir * delta;
    float3 curPos = rayOri + step;

    float totalAtten = 0;
    for (int t = 0; t < RAYMARCHING_STEP_COUNT; t++)
    {
        float3 tolight = (curPos - _VolumeLightPos.xyz);
        float att = dot(tolight, tolight) * _MieScatteringFactor.w;
        float atten = tex2D(_LightTextureB0, att.rr).UNITY_ATTEN_CHANNEL;
        //每次步进增加该点光强
        totalAtten += atten;
        curPos += step;
    }

    float4 color = float4(totalAtten, totalAtten, totalAtten, totalAtten);
    return color * _TintColor;
}
```

图 3-7 点光源步进采样

但此时的点光源采样是一个发光的圆球，如图 3-8，因为 Ray-Marching 是一个偏向物理的体积光实现方法，所以我们要考虑光的散射，光的散射应该是向四面八方的，在一个以尘埃为球心的球体中几乎所有方向都有可能反射到光线，而且每个方向散射出去的光线亮度应该是不一样的，而这些散射出去的光线亮度总合应该和射到尘埃上的那束光线亮度一样，也就是能量守恒。具体公式如下：

$$f_{HG}(\theta) = \frac{(1-g)^2}{4\pi\cdot(1+g^2-2g\cdot\cos(\theta))^{3/2}}$$

g 所表示的就是光的散射系数，g 越大，光束越集中，散射越少，g 越小，散射越强，θ 为光线方向和视线方向的夹角。

同样我们还要根据 Beer-Lambert 法则，表现为入射光强和透光强度的比，具体公式如下

$$OutLight = InLight \times \exp(-c \times d)$$

c 为物质密度，d 为距离。我们可以使用 3D 噪声纹理+uv 流动来动态采样每个点的密度值，这样就可以模拟灰尘在灯光下流动的效果，如图 3-9。

图 3-8 未散射的点光源

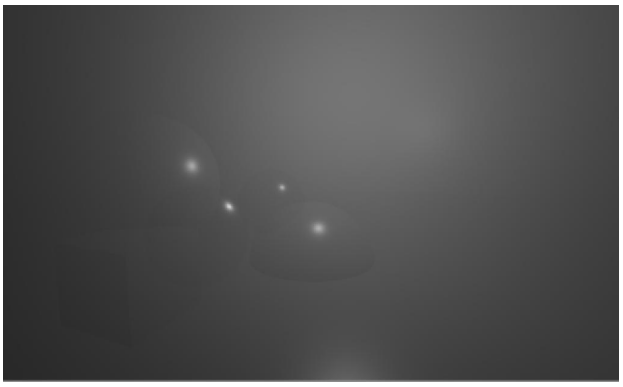


图 3-9 散射后的点光源

我们实验的时候实现了调节灰尘的浓密程度和范围的操作，所以可以实现像布满浓雾一样的状态，如图 3-10 这个也被称为体积雾，很漂亮，也是游戏里经常出现的技术。

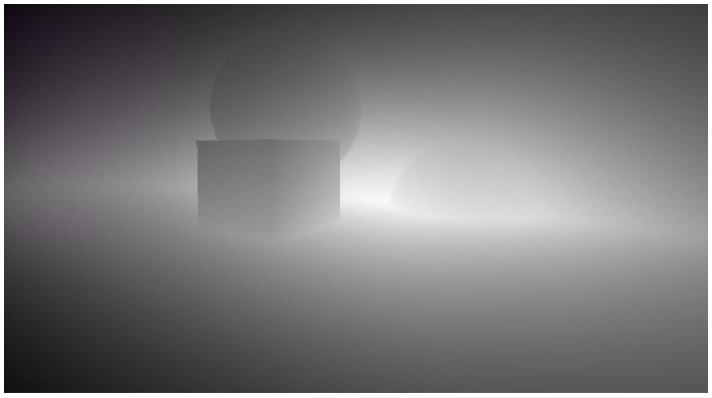
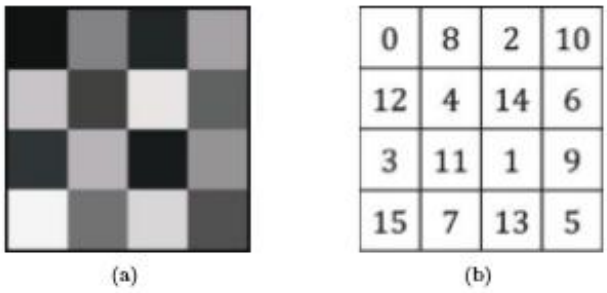


图 3-10 体积雾

虽然实现了光线追踪，但是还是没有出现很明显的丁达尔效应，因为我们没有处理阴影，没有体积光的阴影，体积光就变成了雾，所以此时我们需要给他添加上遮挡效果。本实验采用的是基于 shadowmap 的方法，它的基本思路是，对于每一个采样点，通过阴影图采样，来判断它是否落在阴影中。shadowmap 是指为了得到光源视角下的深度值，将视点设置在光源位置，在光源坐标系下，对整个场景进行渲染到一张纹理图像，记录到 cube texture 中，此时我们便获得所有的遮挡信息，然后二次渲染场景，从而获得如图 3-11 的效果，同时我们还可以对比不同光强下的体积光。

图 3-11 加入遮挡效果的不同光强下的体积光

但明显能看到分层的情况，这是因为当步进次数不够大的时候，等距采样会出现刚好漏过洞口的情况，造成某一段地方的光强累计不够，从而出现分层。从而我查阅资料，参考该文章[1]的实现，尝试了随机采样，也就是 Dither，一个噪声格子，如图 3-12，我们可以定义一个 4*4 的贴图，然后加入高斯模糊处理边缘，提高真实性，图 3-13 是我们的最终实现效果图，明显更加清晰了。



(a)

| | | | |
|---|---|---|---|
| 0 | 8 | 2 | 10 |
| 12 | 4 | 14 | 6 |
| 3 | 11 | 1 | 9 |
| 15 | 7 | 13 | 5 |

(b)

图 3-12 噪声格子

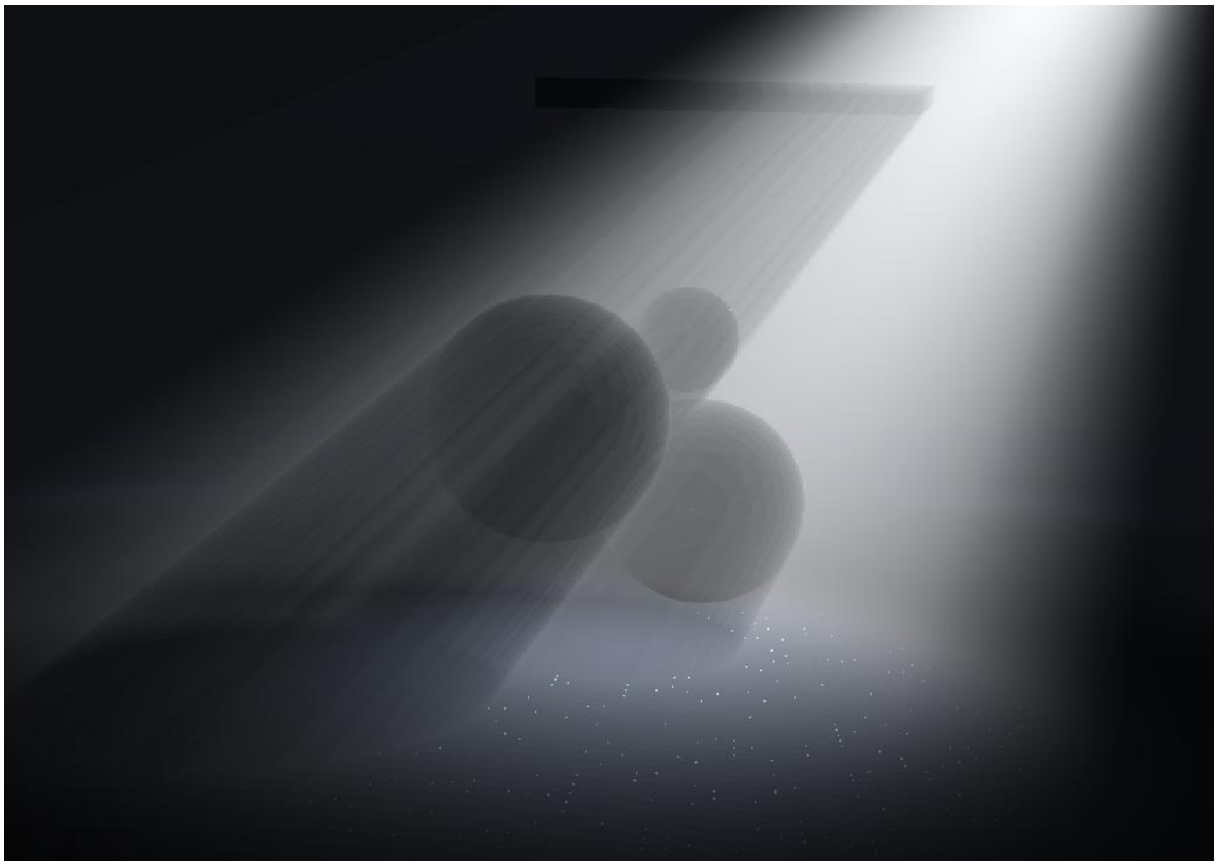


图 3-12 光线追踪最终效果图

# 4 实验总结

本次实验使我受益匪浅，我选择体积光作为实验内容的原因是作为一个游戏爱好者，我着实十分好奇游戏里是如何实现如此逼真的光效的（丁达尔效应），经过本次的实验探究，我可以了解到如下图 4-1 所示的游戏图片（我在游戏里截取的），太阳光大概率是使用径向模糊生成的光效，而清晨的晨雾的实现则依赖与光线追踪实现的体积雾也就是物理实现的 God-Ray（体积光的别称，因为这个效果看起来十分的酷炫）。

图 4-1 游戏中的体积光实现

我对此的判断基本没错，我查询了米哈游（上图游戏发行商）在 unity 技术大会上对游戏画面技术的宣讲记录[2]，”God Ray 也是通过 Ray marching 的方式去生成的，我们会去采样 shadow map，但是最多会采样 5 级的 cascades,God Ray 生成完之后，我们会提供美术一些可以调整的参数，然后将 God Ray 的结果叠加到体积雾上面去。它在使用上面，并不是一个物理上正确的东西，但是它的效果是能够让美术满意的”

图 4-2 游戏中的体积光实现

所以对于游戏，实现体积光不仅要考虑性能，而且还要考虑美感，所以在游戏里，通过美术效果和 Ray marching 实现是十分普遍的，所以我研究的四种方法目前在游戏里都是普遍使用的，比如《耻辱-外魔之死》的窗缝中透光的效果用的就是 BillBoard 贴片，但他们多数用的是远景，而《Shadow Gun》对于 BillBoard 贴片使用出神入化[3]，通过动态调整体积光的颜色及透明度，来达到更加真实的体积光的效果，从而可以真实的用于近景，相比之下游戏界对于 Volume Shadow 挤出的方法就没有这么普遍了，《黑魂》这个游戏最初是使用这个方法处理体积光的，但是后期更新的时候修改掉了，《Inside》这个游戏则在大部分有光源的场景使用了径向模糊，后处理对于场景渲染要求高的游戏

确实是在能实现最高效果下最小的花销了，所以大部分游戏都会在这个方向上偷懒，比如常见的运动场景[4]，如图 4-3, 我在实现的时候也发现这个方案就是处理了一下图片合成，


图 4-3 径向模糊实现运动效果

但是却有着惊人的效果，性价比远高 Volume Shadow 挤出方法，就是只能处理光源，这算是一个很大的限制了，而对于 Ray marching，感觉性能上没太大的问题，就是太吃计算能力了，因为使用了 shadow map，所以大部分游戏会压缩 texture 来提高性能。

在进行实验的时候，我体验了四种方案的实现方法，可以明显的感觉到前三种实现确实快，简单，但是很难变化，也无法实现体积雾这种效果，但是使用 Ray marching，只是实现一个简单的点光源，我可以通过控制灰尘粒度，密度，和光影大小，灵活实现不同场景，并且可以实时移动计算当前光影情况，我实现了脚本可以自动调节光影效果，但是真的很难写，还有各种优化算法，跟 BillBoard 贴片实现真的是两码事（就用了 10 分钟）。

关于这个方向的实现还有很多优化方法和实现方案，我只是研究尝试了最基础的四种，只算是了解了一点皮毛，关于体积光的实现，我觉得还值得更深入的研究。

正文到此结束，谢谢老师。

# 参考文献

[1] Volumetric Light Effects in Killzone: Shadow Fall
https://blog.csdn.net/ZJU_fish1996/article/details/87352770
[2] 详解《原神》画面效果的技术实现 https://www.stmbuy.com/news/item-2jw000004
[3] 游戏开发相关实时渲染技术之体积光
https://zhuanlan.zhihu.com/p/21425792?utm_source=tuicool&utm_medium=referral
[4] 后处理，径向模糊 https://blog.csdn.net/puppet_master/article/details/54566397

# 附　　录

## 附录 A 程序代码

github 关于实验实现的 unity 工程代码,实现详情可以看 md

https://github.com/hyxzjbnb/GodRay-Experiment.git