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）
21301133	牛梦玥	9	27	17	37	90

计 算 机 图 形 学 课 程 论 文

基于 OpenGL 的阴影映射渲染技术的研究与实现

Research and Implementation of Shadow Mapping Rendering Technique Based on OpenGL

学　　院：软件学院

专　　业：软件工程

学生姓名：牛梦玥

学　　号：21301133

指导教师：吴雨婷

北京交通大学

2024 年 6 月 21 日

中文摘要

本文对基于 OpenGL 的阴影映射渲染技术进行了研究与应用实践，了解了什么是阴影渲染以及其研究现状，重点介绍了阴影映射算法的基本原理、常见问题以及其解决方法。首先介绍并实践了阴影映射算法，包括通过光源视角渲染深度信息、使用深度图进行阴影检测等步骤。随后详细分析了阴影映射中常见的问题，如 Shadow Acne、Peter Panning、阴影锯齿现象，并提出了相应的解决方案。进一步探讨并应用了提升阴影质量的技术手段，如引入偏置、采用滤波技术 PCF（Percentage-Closer Filtering）、Poisson 采样和层级阴影图等方法，最终得到了理想的软阴影效果。

关键词：阴影映射；Shadow Acne；Peter Panning；阴影锯齿；PCF

目　　录

1 引言

1.1 研究的意义

在虚拟场景中，阴影的作用也不可忽视，它能增强虚拟场景中的真实感，能使场景显得更加逼真。同时，在 3D 场景中，观察者理解物体的空间位置是多么的重要，真实、准确的阴影效果就能很好的帮助观察者。因为在三维场景中，阴影是不可或缺的重要因素，观察者通过准确的阴影效果可以获得重要的场景空间几何信息[1]。

1.2 国内外研究现状

目前已有多种阴影渲染算法，这些算法主要分为三类，即全局光照算法、阴影映射算法和阴影体算法[2]。其中以光线追踪为代表的全局光照算法虽然效果逼真，但计算量巨大，很难进行实时绘制[3]。阴影映射算法和阴影体算法可视为全局光照方法的近似方法。阴影体算法由于对场景中的物体几何复杂程度有严重的依赖，且对场景中物体形状的构成有严格限制，因此适用性较差。而阴影映射算法[4]对场景复杂度几乎没有任何限制，并且原理简单，绘制速度快，已逐渐成为该领域的研究热点。研究表明，在深度纹理中储存除深度外的更多信息可有效提升阴影渲染的质量[5]，并且通过滤波技术过滤阴影的边缘，可得到软阴影效果[7]。传统阴影映射算法在每帧中均对场景中的所有物体进行两遍绘制，第一遍绘制计算场景中物体的深度并生成一张深度纹理，第二遍绘制渲染场景并计算阴影。

1.3 本文主要工作内容

本实验采用传统阴影映射算法，第一遍绘制计算场景中物体的深度并生成一张深度纹理，第二遍绘制渲染场景并计算阴影，初步得到阴影映射效果。分析该效果的缺陷后，将其通过引入偏置、采用滤波技术等方法进行了改进，得到了理想的软阴影效果。

1.4 本文组织结构

第一章引言，介绍基于时空序列模型的新馆疫情下地铁客运特征分析与预测的背景和意义，以及其同类研究的发展现状，同时阐述该项目的主要内容。

第二章理论基础，介绍了本文涉及到的一些算法、现象的名词解释与理论知识。

第三章实验设置，描述了本文实验使用的环境及工具。

第四章实验方法描述，介绍了本次实验的过程与方法。

第五章实验结果分析与优化，展示了通过第四章的实验方法的实践得到的实验结果，并对实验结果的不足做出了分析，找到解决方法并付诸实践，在三次改进中最终得到较为满意的阴影效果。

第六章结论，对本文的实验工作进行总结，并总结了学到的知识。

2 理论基础

本章主要介绍了本文涉及到的一些算法、现象的名词解释与理论知识。

2.1 Shadow Mapping 基本原理

Shadow Mapping 算法[8]包括对场景的两遍绘制（2-pass）。首先，从灯光的角度渲染场景，仅计算每个点的深度，将深度信息储存在一幅深度纹理（Shadow map）中。接下来，以常规方式渲染场景，对于每一个绘制点,计算其在光源裁剪空间中的深度值 D2,并将此深度值与第一遍绘制后生成的纹理图中相应的深度值 D1 做比较。若 D2>D1,则表示该点与光源之间还有其他物体,该点位于阴影中;否则,当前点不在阴影中，图解如图 1-1 所示。

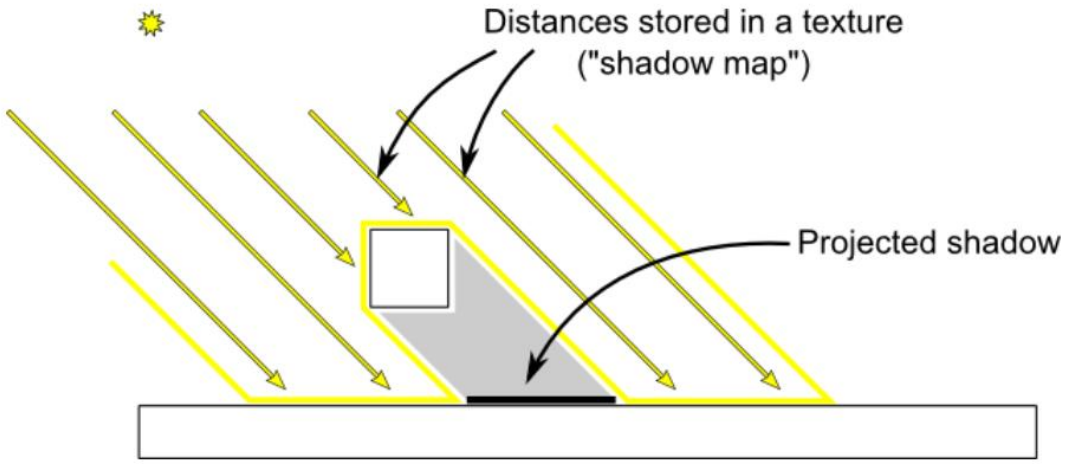

图 2-1 判断采样点是否在阴影中的方法图解

2.2 Shadow Acne

Shadow Acne 是由于深度缓冲精度不足或深度值比较过程中出现微小误差所导致的一种视觉伪影，表现为阴影边界上出现的条纹或斑点状的噪声。Shadow Acne 产生的原因根本原因就是阴影贴图的分辨率不够，因此多个像素

会对应图上的同一个点，形如下图所示。

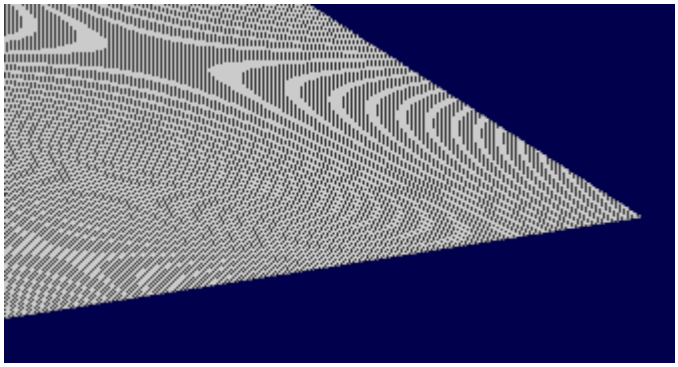

图 2-2 Shadow Acne 图示

解决方法：直接给采样阴影深度加一个阴影偏移（bias），相当于把阴影深度往远处加，从而更不容易产生遮挡。

2.3 Peter Panning

Peter Panning 是由于阴影偏移（bias）设置不当导致的伪影，表现为物体的阴影与物体实际位置之间出现不合理的偏移，在物体缝隙间发生漏光现象。由于阴影偏移过大，阴影的计算位置和物体之间不太贴合，产生中间的缝隙，就像影子脱离了物体（像飞起来的小飞侠一样）被称为 Peter Panning，形如下图所示。

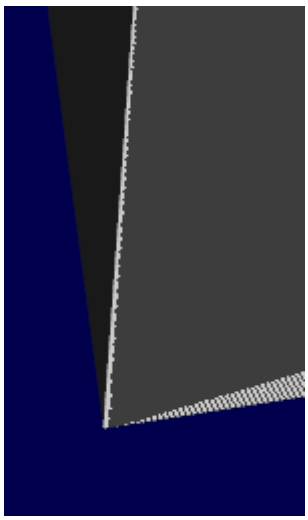

图 2-3 Peter Panning 图示

解决方法：

1. 不使用过大的阴影偏移。

2. 避免使用单薄的几何体（例如薄墙、薄地面）；只要几何体厚度大于阴影

偏移，影子边界便会产生在几何体内部，从而不易看见影子与几何体的分离现象。

3.不采用阴影偏移，生成阴影贴图时设置成仅渲染背面，将正面剔除。

2.4 阴影锯齿（Shadow Aliasing）与 PCF（Percentage Closer Filtering）

阴影锯齿效应（Shadow Aliasing）是指在阴影映射过程中，由于分辨率不够高导致的阴影边缘出现锯齿状不平滑的现象。该效应会使阴影看起来不自然，并且降低渲染图像的视觉质量，形如下图所示。

图 2-4 Shadow Aliasing 图示

解决这一问题的方法包括增加阴影贴图的分辨率、使用滤波技术（如 PCF，即 Percentage Closer Filtering）以及其他高级技术（如 VSM，Variance Shadow Mapping，或 CSM，Cascaded Shadow Mapping），本次实验主要应用 PCF 来解决阴影锯齿现象。

PCF 的核心想法是计算阴影时不是考虑单个采样点，而是在一定范围内进行多重采样，这样可以让阴影的边缘不那么锯齿，因为 Visibility 不再是非 0 即 1，而是带有渐变的取值。在对周围一定范围内若干个坐标进行采样的时候，可以通过分布采样函数来确定 NUM_SAMPLES 个采样位置，为了让阴影边缘更加柔和，可以用一些较好的分布采样函数：

1.均匀圆盘分布采样（Uniform-Disk Sample）：圆范围内随机取一系列坐标作为采样点；看上去比较杂乱无章，采样效果的噪声比较严重。

2.泊松圆盘分布采样（Poisson-Disk Sample）：圆范围内随机取一系列坐标作为采样点，但是这些坐标还需要满足一定约束，即坐标与坐标之间至少有一定距离间隔。

3 实验设置

本次实验以 Windows11 为操作系统，以 Microsoft visual studio 为开发工具，采用 OpenGL 作为图形渲染接口，并基于现有的阴影映射算法进行实现。实验环境包括使用 GLSL（OpenGL Shading Language）编写的着色器程序，用于在 GPU 上进行深度图的生成和阴影检测。

4 实验方法描述

4.1 渲染阴影贴图

在本实验中，只考虑定向光，即距离很远的光，以至于所有光线都可以被认为是平行的。因此，渲染阴影贴图是使用正交投影矩阵完成的。正交矩阵与通常的透视投影矩阵一样，只是不考虑透视，无论物体在相机的远处还是近处，它看起来都是一样的。

本实验使用 1024x1024，16 位深度纹理来放阴影贴图。之所以使用的是深度纹理，而不是深度渲染缓冲区，因为需要对其进行采样。

```
// The framebuffer, which regroups 0, 1, or more textures, and 0 or 1 depth buffer.
GLuint FramebufferName = 0;
glGenFramebuffers(1, &FramebufferName);
glBindFramebuffer(GL_FRAMEBUFFER, FramebufferName);

// Depth texture. Slower than a depth buffer, but you can sample it later in your shader
GLuint depthTexture;
glGenTextures(1, &depthTexture);
glBindTexture(GL_TEXTURE_2D, depthTexture);
glTexImage2D(GL_TEXTURE_2D, 0,GL_DEPTH_COMPONENT16, 1024, 1024, 0,GL_DEPTH_COMPONENT, GL_FLOAT, 0);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_R_TO_TEXTURE);

glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, depthTexture, 0);

// No color output in the bound framebuffer, only depth.
glDrawBuffer(GL_NONE);

// Always check that our framebuffer is ok
if(glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
    return false;
```

图 4-1 深度纹理代码

4.2 设置 MVP 矩阵

用于从灯光角度渲染场景的 MVP 矩阵计算如下：

1. 投影矩阵是一个正交矩阵，它将分别包含 X、Y 和 Z 轴上轴对齐框(-10,10)、(-10,10)、(-10,20) 中的所有内容。这些值的创建使我们的整个可见场景始终可见。

2. 视图矩阵旋转世界，以便在相机空间中，光方向为-Z。

3. 模型矩阵的内容是是模型参数。

```
glm::vec3 lightInvDir = glm::vec3(0.5f,2,2);

// Compute the MVP matrix from the light's point of view
glm::mat4 depthProjectionMatrix = glm::ortho<float>(-10,10,-10,10,-10,20);
glm::mat4 depthViewMatrix = glm::lookAt(lightInvDir, glm::vec3(0,0,0), glm::vec3(0,1,0));
glm::mat4 depthModelMatrix = glm::mat4(1.0);
glm::mat4 depthMVP = depthProjectionMatrix * depthViewMatrix * depthModelMatrix;

// Send our transformation to the currently bound shader,
// in the "MVP" uniform
glUniformMatrix4fv(depthMatrixID, 1, GL_FALSE, &depthMVP[0][0]);
```

图 4-2 设置 MVP 矩阵代码

4.3 着色器

顶点着色器是一种直通着色器，它简单地计算顶点在齐次坐标中的位置：

```
// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;

// Values that stay constant for the whole mesh.
uniform mat4 depthMVP;

void main(){
    gl_Position =  depthMVP * vec4(vertexPosition_modelspace,1);
}
```

图 4-3 顶点着色器代码

片段着色器也很简单：它只是在位置 0（即在深度纹理中）写入片段的深度。

```
layout(location = 0) out float fragmentdepth;
void main(){
    fragmentdepth = gl_FragCoord.z;
}
```

图 4-4 片段着色器代码

4.4 使用阴影贴图

在 main.cpp 中，对于计算的每个片段，必须要知道它是否在阴影贴图的后面。为此，需要计算当前片段在创建阴影贴图时使用的相同空间中的位置。具体做法是用通常的 MVP 矩阵转换一次，然后用深度 MVP 矩阵转换一次。这里将顶点的位置乘以 depth MVP 将得到齐次坐标，其位于[-1,1]，而纹理采样必须在[0,1]中完成。

```
glm::mat4 biasMatrix(
    0.5, 0.0, 0.0, 0.0,
    0.0, 0.5, 0.0, 0.0,
    0.0, 0.0, 0.5, 0.0,
    0.5, 0.5, 0.5, 1.0
);
glm::mat4 depthBiasMVP = biasMatrix*depthMVP;
```

图 4-5 计算阴影贴图代码

编写顶点着色器，输出 2 个位置：gl_Position 是从当前相机看到的顶点的位置，ShadowCoord 是从最后一个摄像机（光源）看到的顶点位置。

```
gl_Position =  MVP * vec4(vertexPosition_modelspace,1);
ShadowCoord = DepthBiasMVP * vec4(vertexPosition_modelspace,1);
```

图 4-6 编写顶点着色器代码

编写片段着色器：texture（shadowMap，ShadowCoord.xy）.z 是光线与最近的遮挡器之间的距离，ShadowCoord.z 是光源与当前片段之间的距离。如果当前片段比最近的遮挡器更远，这意味着我们处于（所述最近的遮挡器）的阴影中，代码中做如下设置：

```
float visibility = 1.0;
if ( texture( shadowMap, ShadowCoord.xy ).z  <  ShadowCoord.z){
    visibility = 0.5;
}
```

图 4-7 编写片段着色器代码

需要利用 visibility 来修改阴影，目的是伪造一些入射光，因为在阴影中

一切都是纯黑色的，但环境颜色不会被修改。

```
color =
 // Ambient : simulates indirect lighting
 MaterialAmbientColor +
 // Diffuse : "color" of the object
 visibility * MaterialDiffuseColor * LightColor * LightPower * cosTheta+
 // Specular : reflective highlight, like a mirror
 visibility * MaterialSpecularColor * LightColor * LightPower * pow(cosAlpha,5);
```

图 4-8 修改阴影颜色代码

5 实验结果分析与优化

5.1 实验结果分析

在第四章实验方法的实践中，最终得到了初步的实验结果，如下图所示。

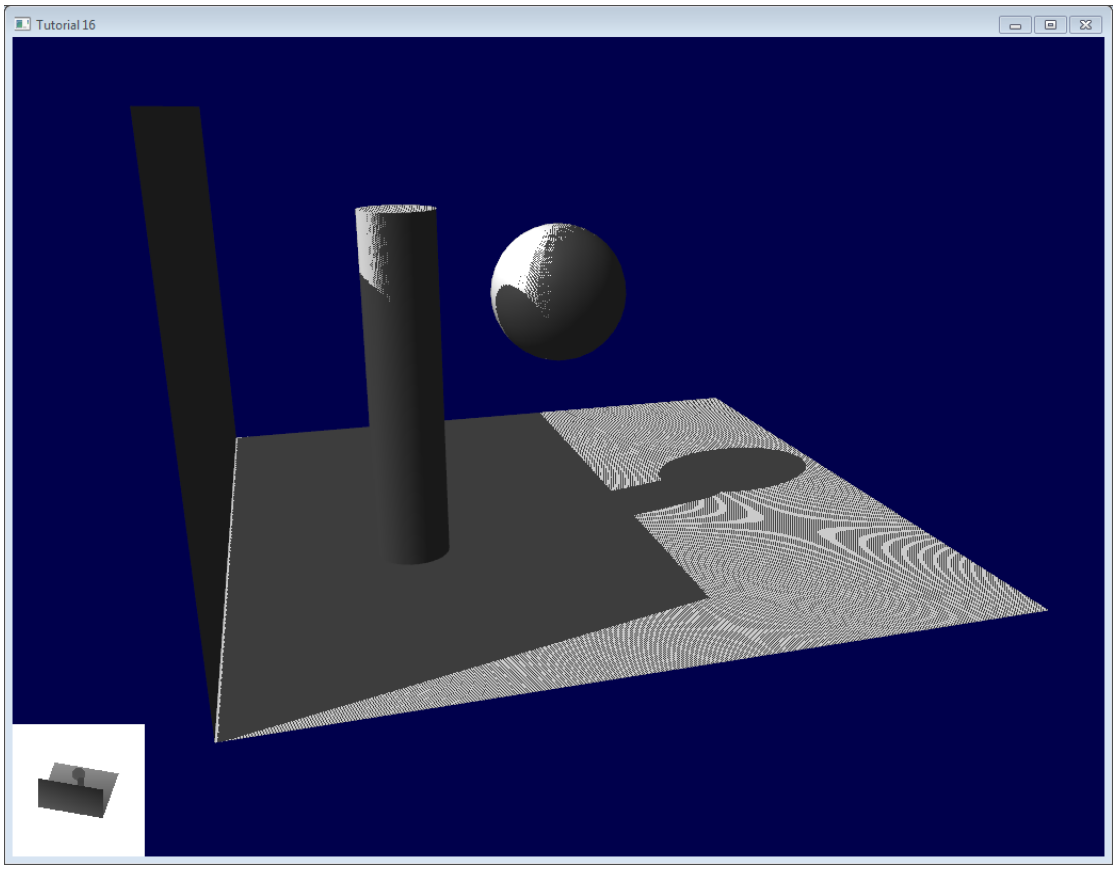

图 5-1 初步阴影映射效果

在基本的阴影映射算法下，虽然能够生成初步的阴影效果，但存在严重的 Shadow Acne 和 Peter Panning 现象。通过引入偏置调整、PCF 和 Poisson 采样等技术手段，能够显著改善阴影质量，使阴影边缘更加平滑，并减少混叠现象的出现。

5.2 解决 Shadow Acne 问题

通过引入偏置（bias），来消除 Shadow Acne。只有在当前片段的深度（同样

地，在光照空间中）与光照贴图值相差甚远时，我们才会着色。

```
float bias = 0.005;
float visibility = 1.0;
if ( texture( shadowMap, ShadowCoord.xy ).z  <  ShadowCoord.z-bias){
    visibility = 0.5;
}
```

图 5-2 引入偏置代码

结果如下图所示，Shadow Acne 基本消失。

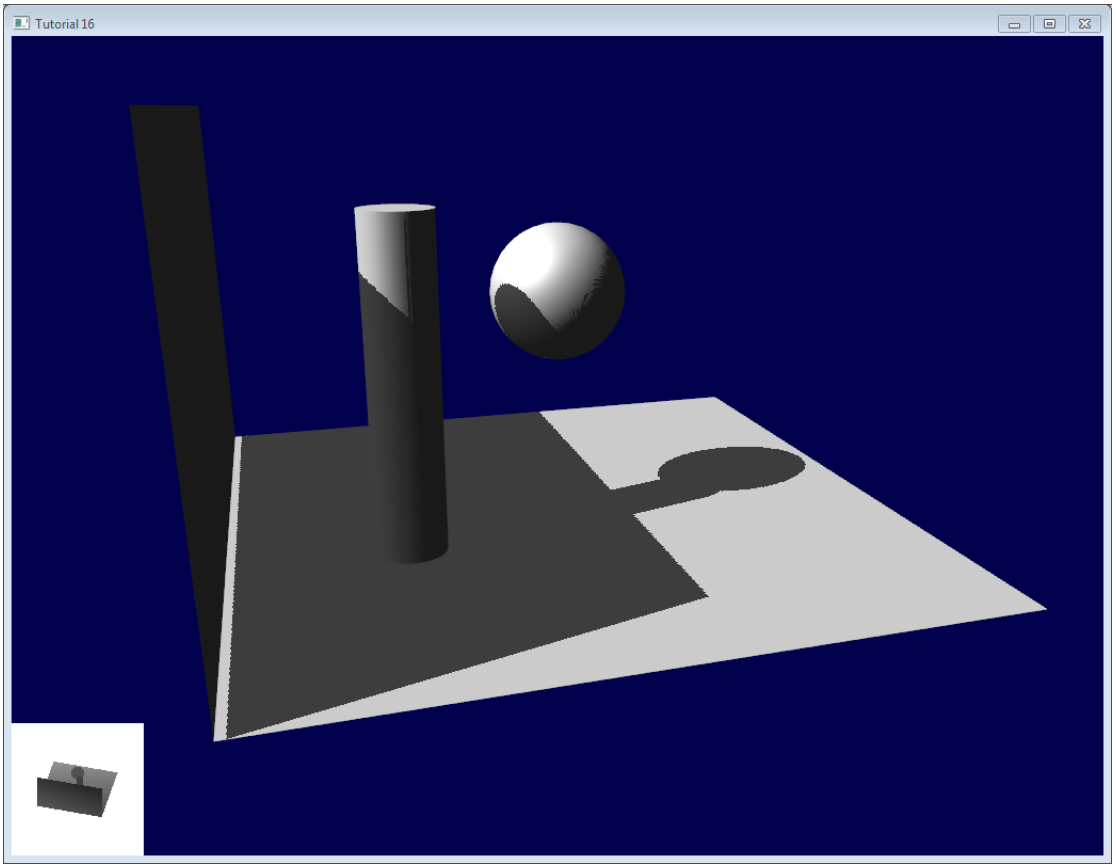

图 5-3 解决 Shadow Acne 问题之后的阴影效果图

5.3 解决 Peter Panning 问题

由于偏置 bias 的引入，导致了一个新的问题：Peter Panning 现象加重。想要解决此问题，只需避免几何形状过薄即可，于是加厚墙壁和地面的厚度，使其厚度大于偏置 bias。其次，在渲染光照贴图时将背面剔除，因为现在，墙的多

边形面向光线，这将遮挡另一侧，而背面剔除不会渲染另一侧，此做法的缺点是要渲染更多的三角形。效果如下图所示。

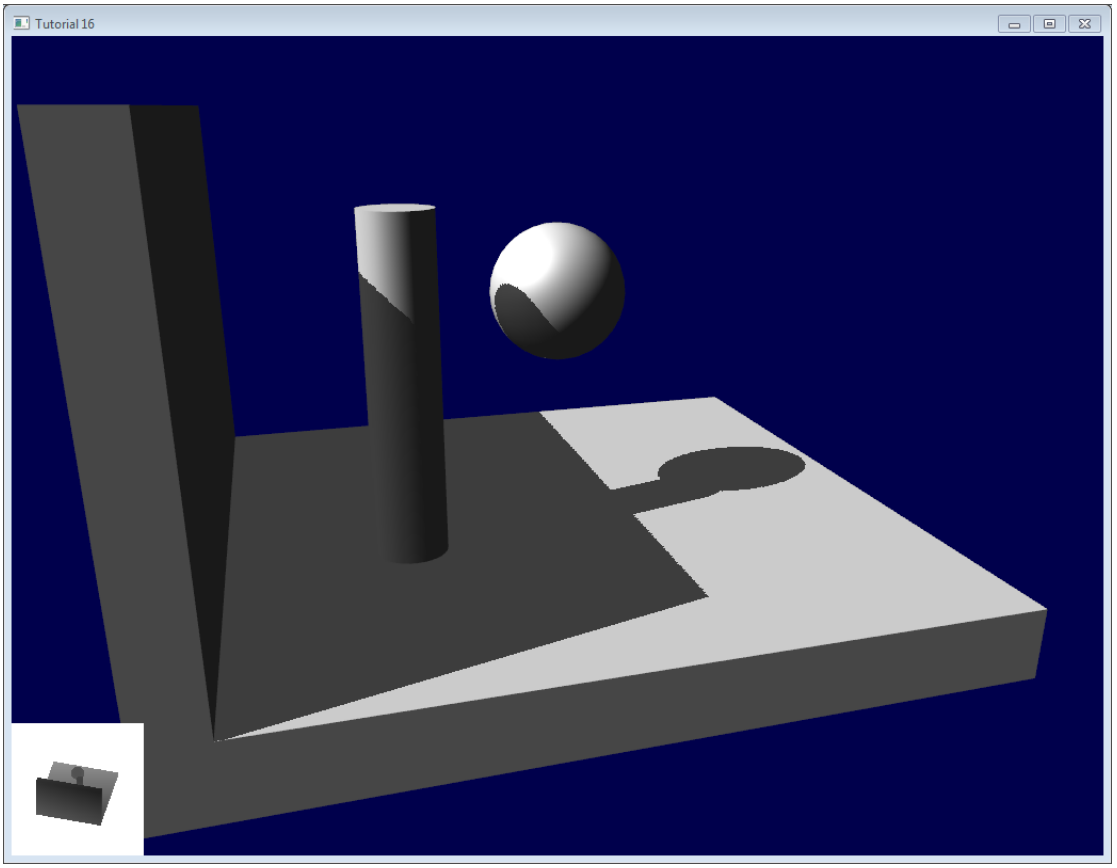

图 5-4 解决 Peter Panning 问题之后的阴影效果图

5.4 解决阴影锯齿问题

解决了上述两个问题之后，阴影的边界上仍存在阴影锯齿的现象。为了解决此问题，采用 PCF 技术，以及泊松圆盘分布采样，对阴影贴图进行 N 次采样，而不是一次，即使 N 很小，也可以有非常好的结果。

```
for (int i=0;i<4;i++){
  if ( texture( shadowMap, ShadowCoord.xy + poissonDisk[i]/700.0 ).z  <  ShadowCoord.z-bias ){
    visibility-=0.2;
  }
}
```

图 5-5 PCF 代码

PoissonDisk 是一个常量数组，定义如下：

```
vec2 poissonDisk[4] = vec2[](
  vec2( -0.94201624, -0.39906216 ),
  vec2( 0.94558609, -0.76890725 ),
  vec2( -0.094184101, -0.92938870 ),
  vec2( 0.34495938, 0.29387760 )
);
```

图 5-6 PoissonDisk 代码

常数“700.0”定义了样本的“扩散”程度。如果样本扩散得太少，就会再次出现阴影锯齿现象;扩散太多了，会得到多层重叠的阴影条带，如下图所示。

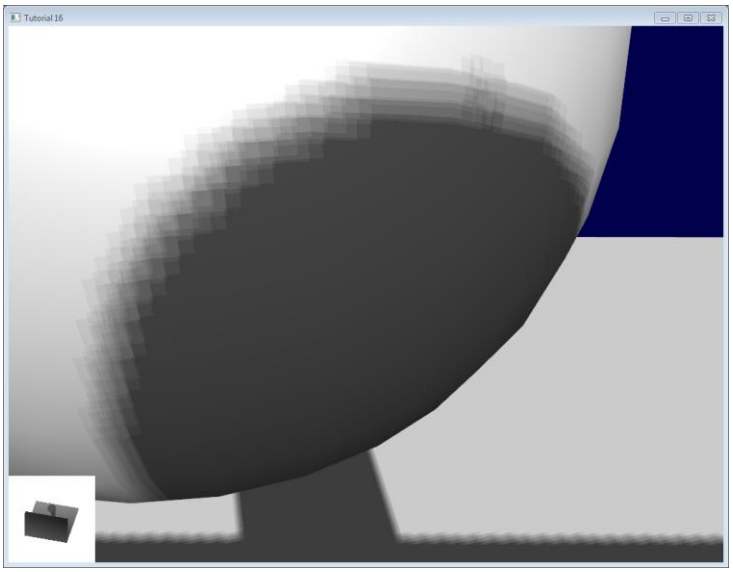

图 5-7 出现多层重叠条带的阴影效果图

继续改进 PCF 的写法，通过为每个像素选择不同的样本来消除这种条带。采用分层泊松采样，用随机索引索引 PossionDisk：

```
for (int i=0;i<4;i++){
        int index = int(16.0*random(gl_FragCoord.xyy, i))%16;
        visibility -= 0.2*(1.0-texture( shadowMap, vec3(ShadowCoord.xy + poissonDisk[index]/700.0,  (ShadowCoord.z-bias)/ShadowCoord.w) ));
    }
```

图 5-8 加入随机索引代码

最终达成了相对逼真的阴影效果，如下图所示。

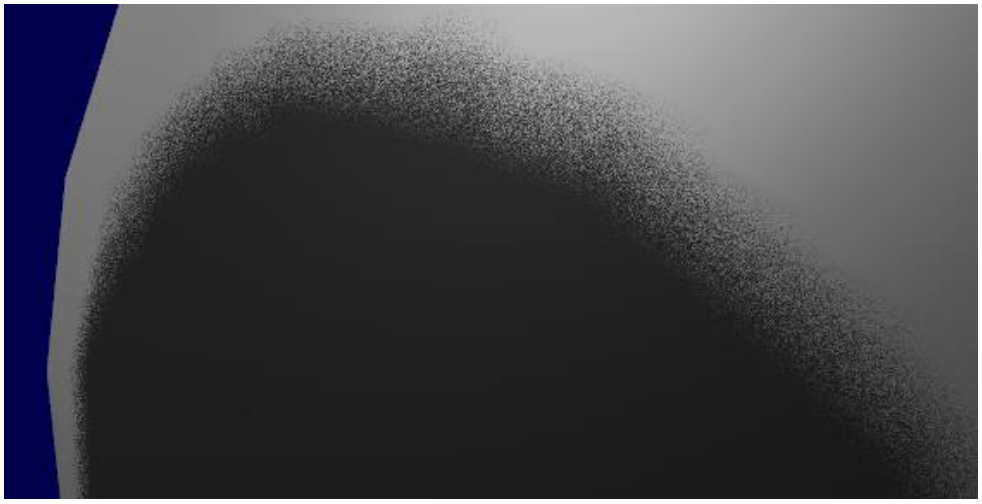

图 5-9 解决阴影锯齿问题之后的阴影效果图

6 结论

本文采用了阴影映射（Shadow Mapping）算法，来实现场景中的阴影效果。第一遍绘制中，计算了场景中物体的深度并生成了一张深度纹理；第二遍绘制中，渲染了整个场景并计算了阴影，初步得到了阴影映射效果。在此过程中，发现了生成阴影效果的一些缺陷，如 Shadow Acne、Peter Panning、阴影锯齿等。

为了改进阴影效果，引入了偏置来解决 Shadow Acne 问题，修改了几何物体的厚度来解决 Peter Panning 问题，并采用了滤波技术（PCF，Percentage Closer Filtering）来平滑阴影边缘，从而减轻阴影锯齿效应。经过这些改进，成功生成了理想的软阴影效果，显著提升了渲染图像的视觉质量。

通过本次实验，我不仅掌握了阴影映射算法的基本原理和实现方法，还深入了解了这种算法效果的一些缺陷以及其成因，还实现了通过技术改进来优化阴影效果。特别是对偏置和滤波技术的应用和理解，使我在计算机图形学领域的阴影渲染方面有了更深入的认识。这些知识和经验将为我今后的研究和实际应用打下坚实的基础。

参考文献

[1] 赵丹.实时阴影渲染技术的研究与应用[D].四川:电子科技大学,2012.DOI:10.7666/d.D765673.

[2] 过洁,徐晓旸,潘金贵.基于阴影图的阴影生成算法研究现状[J].计算机辅助设计与图形学学报, 2010(4):13.DOI:CNKI:SUN:JSJF.0.2010-04-004.

[3] Crow F C .Shadow algorithms for computer graphics[C]//Conference on Computer Graphics & Interactive Techniques. ACM, 1977.DOI:10.1145/563858.563901..

[4] Williams L .Casting Curved Shadows on Curved Surfaces[J].ACM SIGGRAPH Computer Graphics, 1978, 12(3):270-274.DOI:10.1145/965139.807402.AhmedMS,CookAR.ANALYSISOFFREEWAYTRAFFICTIME-SERIESDATABYUSINGBOX-JENKINSTECHNIQUES[J].1979.

[5] Peters C , Klein R .Moment shadow mapping[J].ACM, 2015.DOI:10.1145/2699276.2699277.

[6] SchwaRzler M , Luksch C , Scherzer D ,et al. Fast Percentage Closer Soft Shadows using Temporal Coherence[C]//Proceedings of ACM Symposium on Interactive 3D Graphics and Games 2013.ACM, 2013.DOI:10.1145/2448196.2448209.

[7] 曹家乐,冯月萍.一种改进的阴影映射算法[J].吉林大学学报（理学版）,2018,56(1):89-94.DOI:10.13413/j.cnki.jdxblxb.2018.01.15.

[8] 谭同德.一种基于 Shadow Mapping 的阴影生成改进算法[J].计算机工程与应用, 2008, 44(32):4.DOI:10.3778/j.issn.1002-8331.2008.32.049.