计算机图形学研究进展　实验报告

实现动态场景的阴影场预计算

林涛　3130000064　nblintao@126.com

一、实验目的与要求

- 了解 Precomputed Radiance Transfer 和 Dynamic Soft Shadows 等计算机图像学中有关阴影渲染的知识。
- 了解使用 DirectX 编写计算机图形学程序。
- 了解使用 HLSL 着色器编写非固定管线的渲染程序。
- 了解球谐函数等与计算机图形学相关的数学知识及其运用。

二、实验内容和原理

实现论文 Precomputed Shadow Fields for Dynamic Scenes 的算法。

这篇文章提出了“阴影场”的概念，对于场景中的每一个物体，可以在其周围的若干采样点处进行可见性采样，并将结果存储成遮挡场（Object Occlusion

Field, OOF）。同样，对于光源，也在采样点处进行辐射亮度采样，并将结果存储为辐射亮度场（Source Radius Field, SRF）。绘制时，首先对 OOF 和 SRF 进行排序，并从近到远地利用球谐函数的乘积运算来进行遮挡效果叠加，然后通过三重运算积来计算最终的着色值。[1]

三、实验过程和数据记录

在正式写代码之前，我主要做了两件事。

- 反复通读了要实现的文章，MSDN 上关于 PRT 的介绍以及 Precomputed Radiance Transfer 和 Dynamic Soft Shadows 的课件，争取在基础知识的理解上没有偏差。
- 通过参考 MSDN 和《DIRECTX 9.0 3D 游戏开发编程基础》学习了 DirectX 的基本知识。查找了现有的可以利用的材料，发现最接近的是 DirectX SDK 的样例 PRTDemo 的框架，因为这个框架原先已有 PRT 的实现、测试用的网格和鼠标的控制。为了在这个框架上写，我通读了代码，我理解了其架构和关键部分的计算（尤其是 PRT 的计算）。

下面按照顺序来介绍一下算法实现的主要部分。

采样

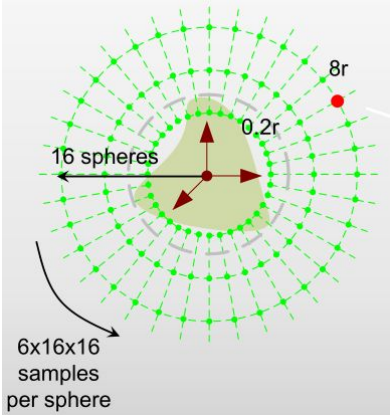
阴影场的预计算，主要在各个采样点上获得全景的光场信息，并使用球谐函数进行压缩。

---

[1] 该段表述引自：任重，基于预计算的交互式全局光照明研究

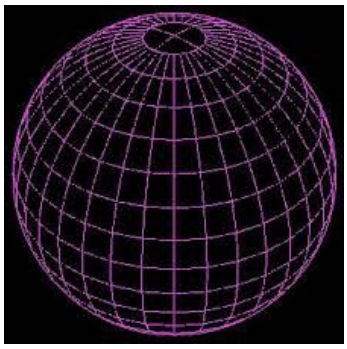
图表 1　Cube Map 采样结果示例

要获取到光场在某一个位置上的信息，可以想象成在那个位置有个相机，朝上下左右前后六个方向拍六张照（如上图所示），即将设定好的场景渲染到这些平面上。这六张照组成的立方体盒子就是 cube map ,使用 DirectX 自带的函数，可以求出相应的球谐函数。这样就完成了一个位置上的采样。



图表 2　原文中给出的采样方式

对一个物体，应该在多个位置进行采样。在原文中，作者在 0.2 倍半径到 8 倍半径距离的圆周均匀选取了 16 层球壳，对于每一层球壳，在 6 个方向上各采了 16 个样。



图表 3　我的采样方法

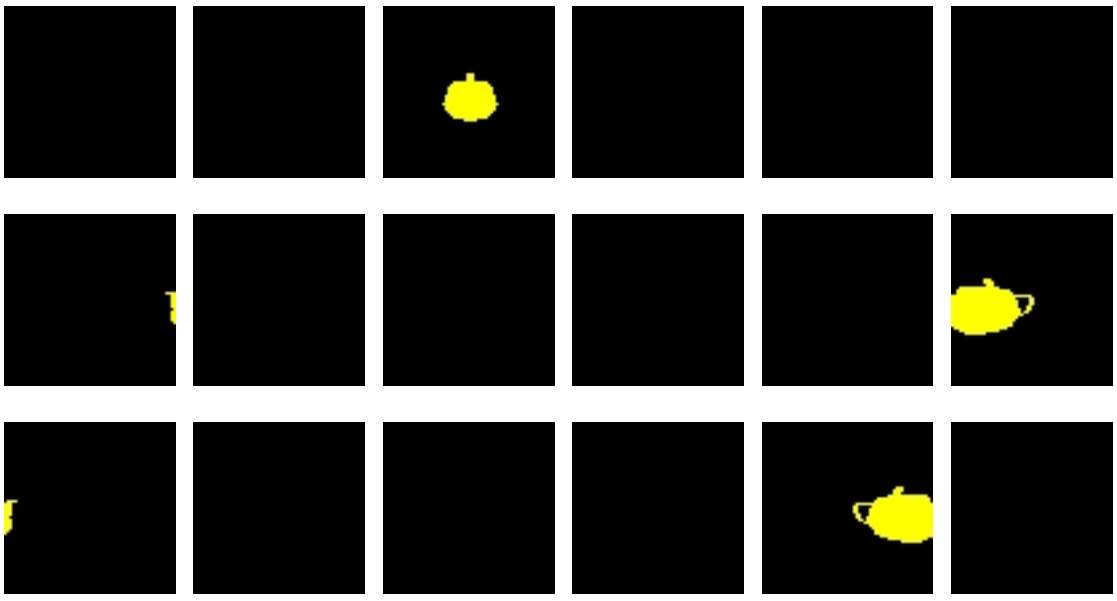
基于原文的思想，我本着方便计算的原则，对采样方法略作了改变。在径向的采样不变，在球面我通过经度和纬度的均分进行采样点的计算。在课堂展示的版本中，我在采样数量是这样的：

参数	采样数量
距离	8
纬度	4
经度	4

通过将采样的 cube map 保存到本地图片，我们可以观察、验证这样采样结果的准确性。

下面各图显示的分别是在不同纬度、经度、距离对同一个茶壶进行渲染得到的图像。

图表 4　不同纬度下采样得到的茶壶图像



图表 5　不同经度下采样得到的茶壶图像（三行图像分别来自于三个相聚 120 度的采样点）

图表 6　不同距离下采样得到的茶壶图像

在采样中，还有一个关键的问题，那就是 SRF 与 OOF 的区别。前期工作是没有注意到这个问题，从而耗费了一些时间。如果把一个物体看做光源去采 SRF，背景色应为黑色，前景是物体的颜色。如果把一个物体看做遮挡物去采 OOF，背景色应为白色（即能反映下一层的所有光），前景可以取为黑色。

图表 7　SRF 的采样示例（左）和 OOF 的采样示例（右）

根据采样得到的 cube map，使用 DirectX 内带的函数，生成对应的球谐函数的参数。

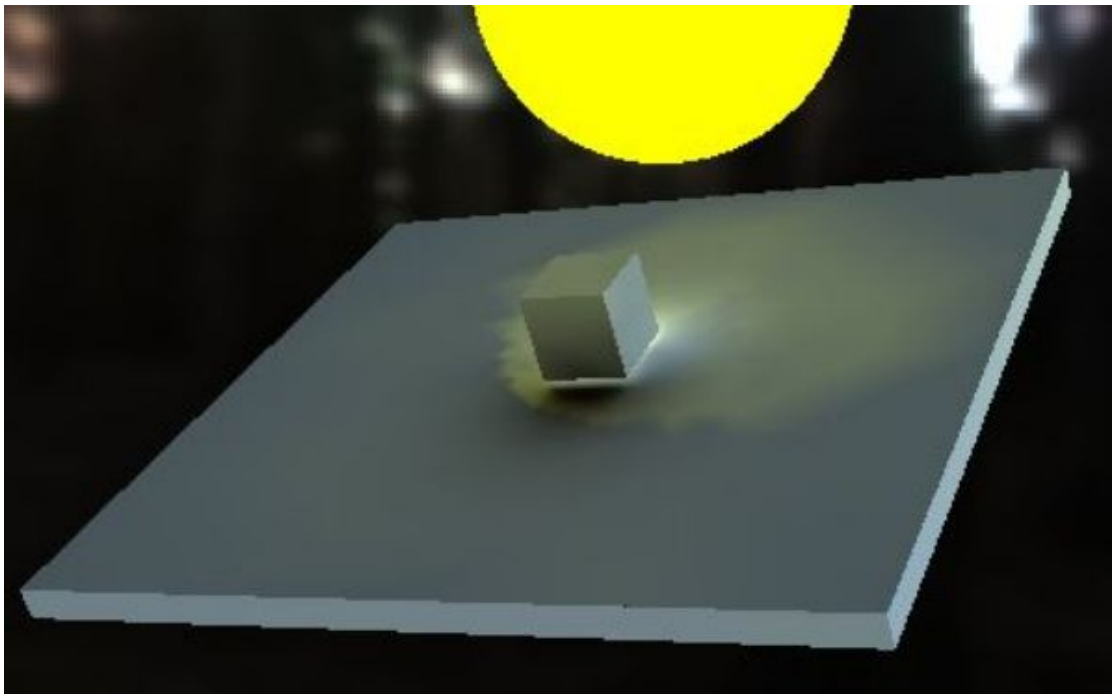
渲染

在原先的 PRTDemo 中，程序并不是直接使用 PRT，而是使用 PCA 算法选取主元做了压缩。在渲染时，每帧在 CPU 中对各个成分算好其与环境球谐函数的点乘的结果，在顶点着色器中只要对其按比例混合即可。

而在我们的程序中，并不能只得到 BRDF 与环境的点乘，而要让顶点着色器获取到这两者的数据。因此我做的第一件事就是将原来的 PCA 版的 PRT 改写进了着色器，运行效率上并未提高（甚至可能降低），但这使后来的工作有了可能。

下一步应该要让每个顶点拿到 SRF 中自己位置对应的值。我将前面的采样结果通过常数寄存器进了着色器中。利用顶点的位置与光源中心和半径的关系，经过一系列的数学换算，确定了其最接近的采样点。进而可以得到这个采样点的值。

将得到的SRF在该位置的球谐函数参数与环境的相加，在与BRDF进行点乘，就可以计算得到这一点的颜色。下图是我在完成到这一步时的效果。



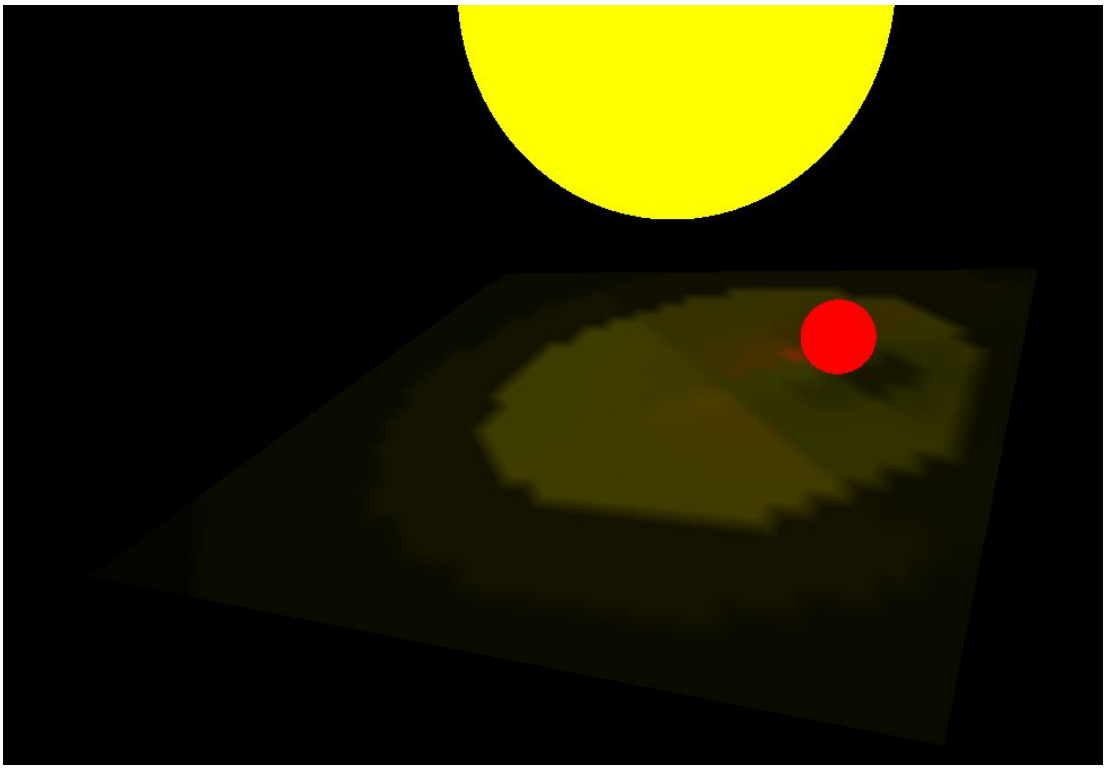
图表 8　BRDF，SRF 与环境光的叠加

下一步就是将 OOF 也加入到计算中。OOF 中最接近采样点的选择与刚才是一样的，可以照搬完成。得到顶点在 OOF 中的球谐函数参数并不难，重要的是将这个值与 BRDF，SRF 和环境光相叠加。

原文中的 Figure 5 给出了这个过程的伪代码，这也是渲染部分的核心。其中关键需要解决的计算问题是球谐函数的三重积。

在 DirectX 中，可以通过调用 SDK 中的库计算 6 阶以下的球谐函数三重积，可在着色器中，并没有类似的工具。我本来计划的是用 6 阶的，在任重老师的建议下，改成了 4 阶的。后来因为着色器资源不够用的缘故，又改成了 3 阶的。我的 3 阶的球谐函数的三重积改自 Code Generation and Factoring for Fast Evaluation of Low-order Spherical Harmonic Products and Squares。

这样，终于做成了如下图所示的结果。



图表 9　加上 OOF 后的渲染结果

黄色的是作为光源的球，红色的是作为遮挡物的球，只对下方的平面进行了上述方式的渲染。在图中可以看到红色小球的右下角能成功看到阴影。

顶点着色器因为版本问题只能以 debug 模式跑在 CPU 上，因此运行的效率不尽人意。上图的平面大概有 1000 个顶点，在 Intel Core i3-3110M 的 2.40GHz CPU 上的帧率是 8.5 FPS。

四、实验结果分析

展示

展示之后，两位老师对我的实现提出了一些意见与建议，主要集中于以下两点：

一，计算的速度太慢，与原文章的帧率相差大；

二，渲染的画面不够美观，与原文章的结果相差大。

对于第一点，可以采取的方法有：

1）使着色器的代码真正在 GPU 上跑起来。我的程序中 Vertex Shader 使用的是 vs_3_0，在我自己的计算机的 GPU 上面运行会有问题。所以在使用中是开启了 DEBUG_VS 模式，利用 CPU 去模拟 GPU 的运行，在运行效率上有数量级的差别。要实现这一点需要研究下我的着色器代码中到底有哪些问题，并找到一台合适的计算机运行。

2）优化着色器代码，利用 GPU 的位操作等技巧加速。

3）改用 CUDA，实现更加底层的优化。

对于第二点，主要原因是 GPU 上的 constant register 空间有限，而实现算法需要让顶点着色器中的每个 vertex 都接触到全部的采样数据。在目前容量下，我的只采了一个 OOF 和一个 SRF，采样的位置数目是 8（距离）*4（精度）*4（纬度）。因而能看到的阴影十分失真。

改进

在展示完后，为了最后提交一个更好的成品，我针对上述的批评进行了评估。我觉得能较快完成的优化是第二点。根据任重老师的建议，可以把采样的数据存

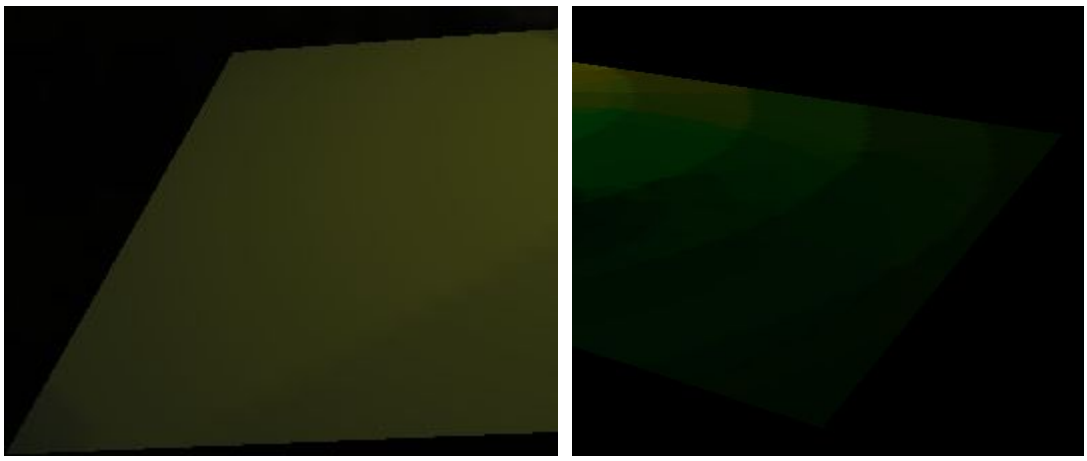
放到纹理中，用顶点着色器调用纹理，读取其中的数据。

展示完之后的一周多时间里，我将主要精力用于摸索这一方法，终于实现了通过纹理从 C++代码中向顶点着色器传送浮点型数据。通过这样的方法，我实现了大批量传输采样数据。优化前后采样数量的对比如下表所示：

参数	采样数量	优化后采样数量
距离	8	≥20
纬度	4	≥16
经度	4	≥16

这个采样数量已经达到了对文章中对低频的例子的采样数，已不再是主要问题。再增大采样不再影响成像质量，反而会成为速度的阻碍。这也成为了展示后的一周多时间里唯一完整解决的一个问题。

在使用纹理传输数据之后，着色器中的资源瞬间就多起来了，因此可以用来做一些以前资源不够无法做成的事，比如对采样点的插值。展示前的版本我是直接取距离最近的采样点的，因此过渡很不均匀。展示后，我根据效果失真的问题，对距离的采样做了插值，当要计算的点在两层球壳之前时，根据距离去混合两个不同的采样点。将这个方法应用于只有 SRF 的情况下，得到了如下的结果：



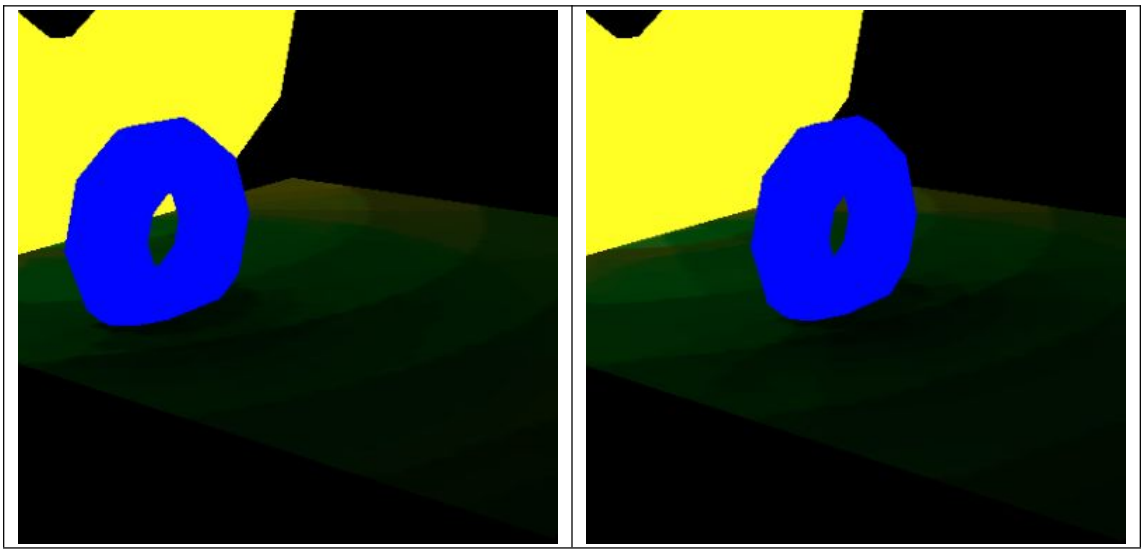
图表 10 径向插值（左）与没插值（右）的对比

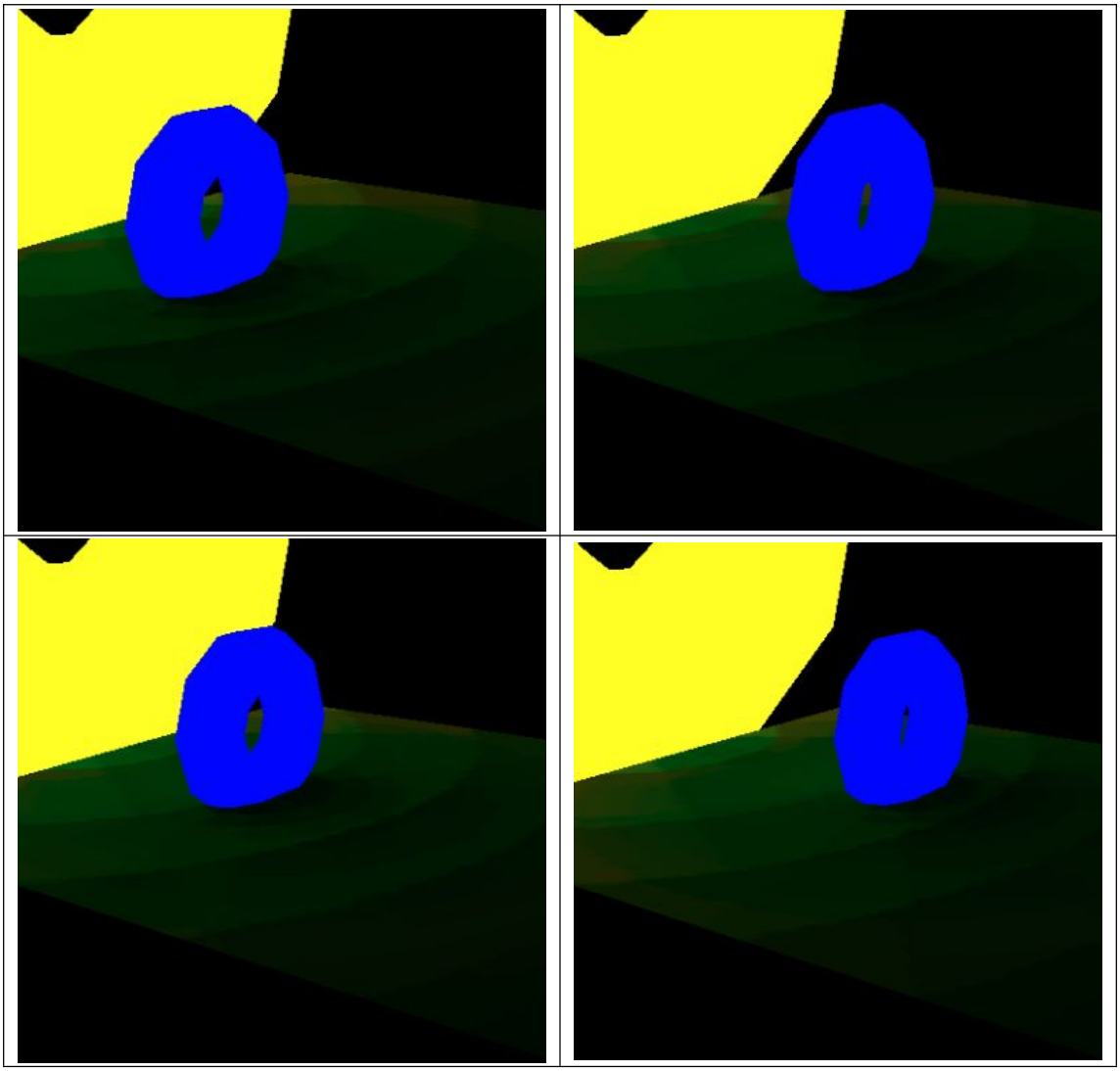
由此可见，插值能让最后效果显得比较光滑，更加自然。受此激励，我对纬度也做了插值。不幸的是，在完成代码之后，看到了熟悉的报错 “maximum temp register index exceeded”，着色器的资源又不够了，而且这次还是临时寄存器。在放弃了对纬度的插值后，我希望将径向插值一样地用到 OOF 上，但依旧是上面的问题。最终，为了两者匹配，我将这部分功能放弃了。

最终效果

下面选取一些实验过程中获得的图片来说明这个算法实现的效果。

下图是圆环在不同位置时产生的阴影，阴影的存在可以明显被察觉到。





图表 11 圆环在不同位置时产生的阴影

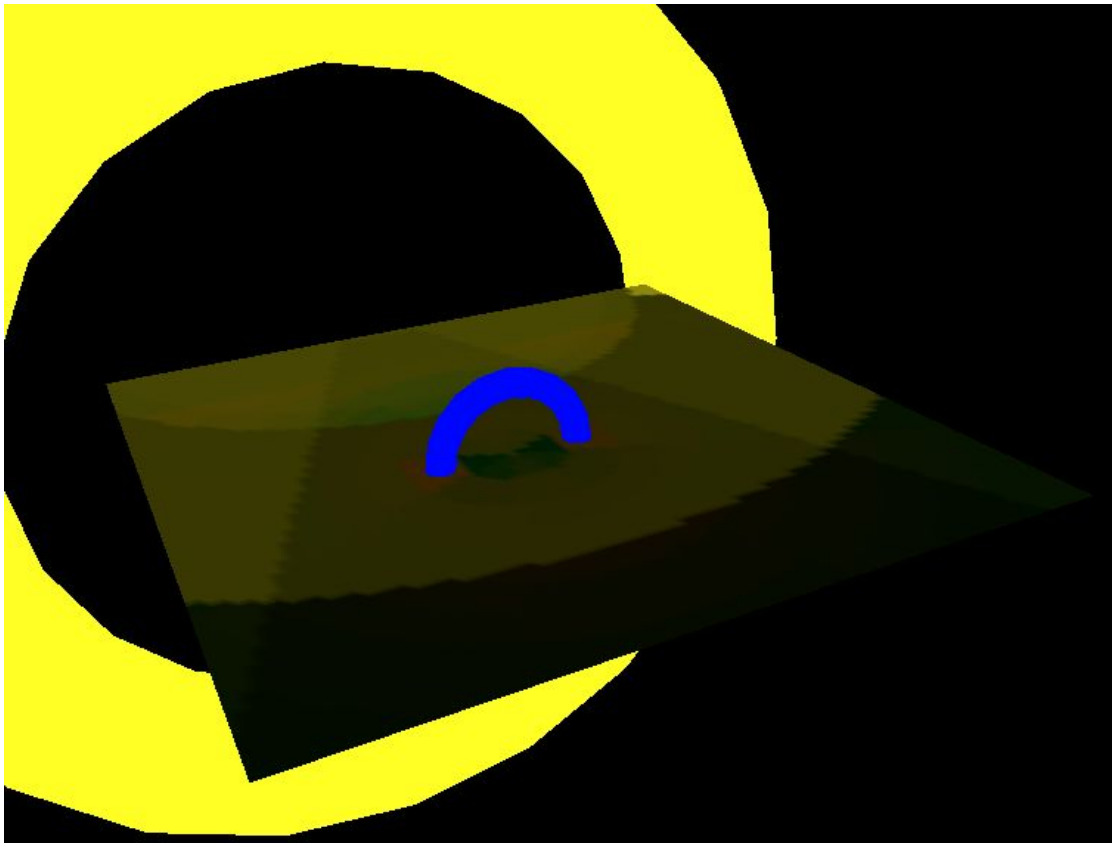
看原文时，我被下面这张图惊艳到了。虽然只是个简单的 utah teapot，但这些多重光源的交叠产生的阴影太让人印象深刻了。

图表 12 原论文中中使用的图

虽然我后来意识到，这是用了 wavelet 做到的全频率的版本。我用这样不精确的采样，球谐函数实现，再加上有限的 GPU 资源配置，是不可能达到这样的效果的，甚至连个茶壶头的阴影也看不到。但为了对照差距，我还是做的下面的这张图。我发现论文中图片采用的位置、角度、裁剪背后可能都充满了心机，当我把我的茶壶摆得和文中几乎一模一样时，似乎变得不那么难看一些了。

图表 13 我实现的黄色光源照在红色茶壶上产生的阴影

此外，在我用圆环测试的时候，得到了这样一个效果，觉得挺有意思。



图表 14 黄色圆环发光，蓝色圆环遮挡，注意蓝色圆环下的阴影

实现结果的展示到这里就结束了。因为我只在底部的平面上使用了阴影场的方法渲染，因此遮挡物和光源的渲染看上去比较简陋。由于着色器上放不下插值版本的代码，平面的渲染上会看到明显的条纹。这都很大程度上影响了最后的效果，在此向读者致以歉意。如果以后我有机会，也许会改正这些问题。

代码和工程说明：

因为是在 DirectX SDK 的样例 PRTDemo 的框架基础上添加的，因此我写的代码与原来的样例混在一起的。在最终提交的代码中，我的代码主要是如下几部分：

- Merging 部分的代码在 SF.fx 效果文件中（全部，第 1 行至第 733 行）。
- Sampling 部分的代码主要在 PRTMesh.cpp 中（大致为第 1 行至第 28 行，第 200 行至第 783 行，第 1188 行至第 1260 行，第 1494 行至 1597 行）。
- 相关的控制的代码主要在 main.cpp 中（大致为第 1387 至 1474 行）。
- 生成网格的脚本在\Media\PRT Demo\gene.py 和 gen.bat。

操作说明：

- 运行工程后，键入 Z，等待一段时间，可以看到两个茶壶（或球）。作为光源的茶壶是黄色的，作为遮挡物的茶壶是红色的，选中的茶壶是蓝色的（选中时的颜色不影响其属性）。
- 除了在界面上提示的 PRTDemo 原有的操作（如用鼠标拖动的旋转）之外，我添加了如下的操作：
- 通过 Q 和 A 可以向前向后依次选择这两个球。
- 通过 J/L/I/K/O/P/可以向左/右/上/下/前/后移动选中的球，如果一个球都没选中，将对所有的球进行操作。
- 通过 N/M 可以放大和缩小选中的茶壶。如果一个球都没选中，将对所有的球进行操作。
- 通过 E 可以将选中的作为光源转化为遮挡物，或将遮挡物转化为光源。

五、讨论与心得

现在是 2015 年 7 月 6 日凌晨 5 点，又是一个“忘了”睡的晚上。从做图形学的大程以来，我已经不止一次在艰辛而又刺激的工作中不知不觉度过一天了。

这个大程或许是我大学期间付出最大也收获最大的一次了。

在前期的准备过程中，我翻来覆去地读这篇文章，争取弄清楚每一环。以前读论文，只是想要了解人家做了什么，翻一翻就知道了。但当我真正面临要实现一篇文章的难题时，我才发现论文中的每一句话都充满了价值。有时候一句话没看到，就不得不花上几天时间摸索，等回过来发现论文中清清楚楚地写着，又不禁拍脑袋叫绝。

这对编程能力也是一种极好的锻炼。虽然以前学习计算机图形学的时候也用 OpenGL 实现过一些东西，但那些只要翻翻什么红书蓝书，或是查查什么博客论坛，基本上就能完成了。而在实现论文的过程中，可能每一步都会遇到难解的问题，有时候在 MSDN 的文档上就能看到解释，有时候或许真相隐藏在某游戏开发论坛一个帖子的回复中，有时候我甚至要尝试几个小时才弄清检索时该如何措辞表述我的问题。

相比之下，DirectX 和 HLSL 知识的掌握，计算机图形学渲染知识的了解，只是锦上添花了。

从确定这篇文章，到动手写第一行代码，从一周多前的那次展示，到现在的完成最后一次 commit，这个过程充满了惊喜，惊喜得可以让我不知疲倦地工作下去。虽然最终的结果还有这样或那样的不足，但我依旧喜爱我的这个小作品，依旧对计算机图形学充满热情。

我要感谢周昆老师和任重老师的耐心解答和倾囊相授。我要感谢两位室友（张闻、周宇恒）他们的队友（荣宇良、王涛）给我带来的灵感和动力，讨论总能迸发出灵感的火花。我还要感谢互联网上无数有名或无名的贡献者们，你们踩过的坑是我们进步的开始。

六、参考文献

- Zhou K, Hu Y, Lin S, et al. Precomputed shadow fields for dynamic scenes[J]. ACM Transactions on Graphics (TOG), 2005, 24(3): 1196-1201.
- Luna, Frank D, DIRECTX 9.0 3D 游戏开发编程基础.
- 任重，基于预计算的交互式全局光照明研究。
- Snyder John, Code Generation and Factoring for Fast Evaluation of Low-order Spherical Harmonic Products and Squares.
- Sloan Peter-Pike, Stupid Spherical Harmonics (SH) Tricks.
- 其他参考站点（不完全）

https://msdn.microsoft.com/en-us/library/windows/desktop/bb153297(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms883586.aspx
http://en.wikipedia.org/wiki/Spherical_harmonics
http://www.cnblogs.com/daniagger/archive/2012/06/02/2532223.html
https://msdn.microsoft.com/en-us/library/windows/desktop/bb147287(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/bb205455(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/bb204871(v=vs.85).aspx
http://blog.csdn.net/poem_qianmo/article/details/8475261
http://www.ownself.org/blog/2010/huan-jing-ying-she-environment-mapping.html
http://developer.178.com/201002/61090665444.html
http://www.wangchao.net.cn/bbsdetail_69543.html
https://github.com/Patapom/GodComplex/blob/master/Resources/Shaders/Inc/SH.hlsl
http://www.gamedev.net/topic/566462-hlsl-large-arrays-of-matrices/
https://github.com/SHTOOLS/SHTOOLS/blob/54c1b7321ba59e58ed0fca5282032e03859383e6/src/SHMultiply.f95
https://msdn.microsoft.com/en-us/library/windows/desktop/bb174363(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/bb205131(v=vs.85).aspx#Filling_Textures_Manually
https://msdn.microsoft.com/en-us/library/windows/desktop/bb205131(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ff476905(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ff476521(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/bb205913(v=vs.85).aspx
http://www.gamedev.net/topic/455511-writing-data-to-a-texture-using-lockrect/
https://msdn.microsoft.com/en-us/library/windows/desktop/bb206339(v=vs.85).aspx

http://xboxforums.create.msdn.com/forums/t/4656.aspx
http://www.gamedev.net/topic/442138-packing-a-float-into-a-a8r8g8b8-texture-shader/
http://www.gamedev.net/topic/585330-packing-a-generic-float-into-a-rgba-texture/
http://www.gamedev.net/topic/333097-hlsl-pixel-shader-pack-32-bit-float-to-32-bit-a8r8g8b8-render-target/