《计算机图形学基础》大作业报告

——光线追踪与网格简化

清华大学计算机科学与技术系

计 34 班

杨志灿 2013011377

目录

1. 简介

在这次计算机图形学基础的大作业中，我使用了 Phong 光照模型实现光线追踪算法。在完成核心功能后，还实现了抗锯齿、两种 k-d tree 求交加速，基于蒙特卡罗算法的软阴影和景深效果、平滑法向量的生成与渲染、纹理贴图、凹凸贴图、bmp 文件输入输出等功能。同时，我的程序还支持复杂 obj 模型的处理，可以处理 mtl 材质文件，支持多贴图，还可方便地对模型进行放缩、平移、旋转等变换。

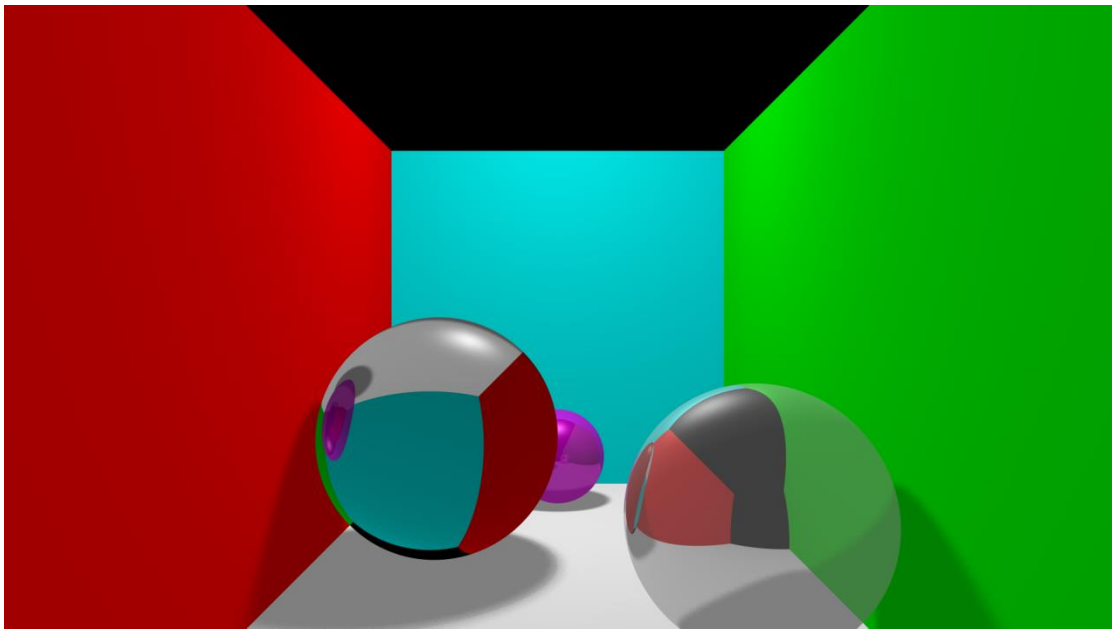
我实现了基于最短边和基于二次误差的网格简化算法，并通过手写堆、人工栈、链表、并查集等数据结构加速，提高算法效率。

本程序未使用除 STL 外的任何库文件，可以很方便地在其他平台运行。

2. 光线追踪

2.1 基本功能

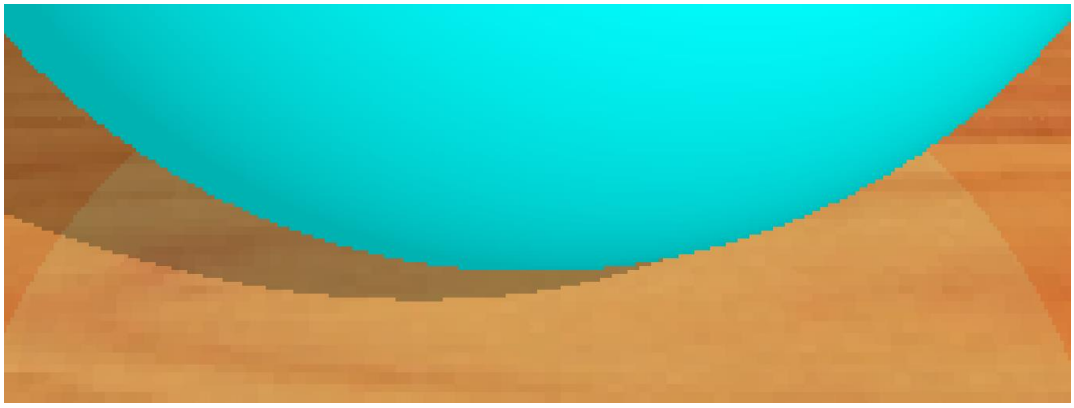
光线追踪的基本功能包括漫反射、高光、镜面反射、折射等。如下图所示，三个球体从前往后分别以反射、折射、漫反射效果为主。



2.2 抗锯齿

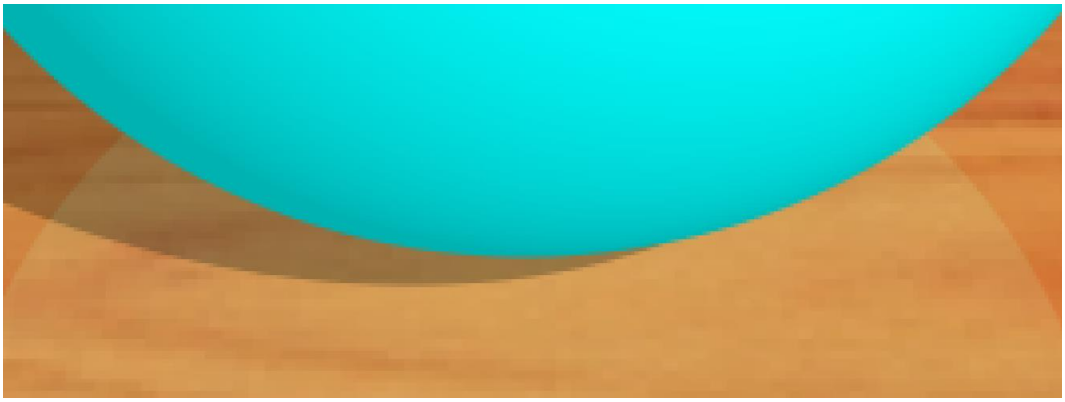
光线追踪算法对每个像素点发出一条射线，根据相交物体的属性计算该像素点的颜色。所以在物体的边缘处，射线的求交结果分为有交和无交两种，就会产生明显的锯齿效果。

下图所示是一个摆放在反射地板上的球体的底部，图片放大后锯齿感明显。



我使用了基于蒙特卡罗算法的抗锯齿方法。具体方法是将每个像素切分成 n*n 的小矩形，在每个小矩形中随机选取一点作为采样点，最后取均值即可得到该像素点的结果。下图为将每个像素点分为 4*4 个采样点的渲染结果，物体的边缘的锯齿感减轻。

同时，程序支持自适应抗锯齿，对于多次采样结果相同或相近[1]的情况下，不再细分采样点。这样既保证了效果，也兼顾了效率。



[1] 可以在配置文件中设置参数，判断两种颜色的差距为多大时算为“相近”。

2.3 k-d tree 求交加速

若场景中有许多物体，O(n)枚举判断光线与哪个物体相交将会耗费大量时间。利用轴平行包围盒技术，使用 k-d tree 将三维空间不断切分，可以使求交检测的时间复杂度降为期望$O(\log n)$。

我实现了两种 k-d tree，可在配置文件中选择使用哪种建树方法：

一是面片均分，k-d tree 的每个非叶节点将所有面片沿某个坐标轴依据重心坐标均匀地分为两份，这样做能保证最少的节点数和树的平衡。我使用快速中位数算法实现均分操作，使得建树的时间复杂度为期望$O(n \log n)$。

二是 SAH 启发式建树，启发函数为轴平行包围盒的表面积与面片数的乘积，可以简单理解为：遍历面片数多的节点所需时间更多，而遍历表面积更大的节点的概率更大，所以最小化启发函数能极大提高光线求交的效率。不过这样建树的时间复杂度为$O(n^2 \log n)$，树的节点数也会更多，这意味着空间占用也更多。

以下为本地的测试结果，图片分辨率均为 1920×1080,总时间为包括输入、建树、渲染、输出在内的程序运行时间。

模型	面片数	面片数均分			SAH 启发		
		建树	节点数	总时间	建树	节点数	总时间
龙	209227	0.259s	418453	55.36s	3.831s	722691	50.98s
龙(4x 自适应 AA)	209227	0.277s	418453	226.83s	3.752s	722801	207.26s
法拉利	681893	0.861s	1363785	136.20s	13.99s	2341143	133.16s

运行环境：

操作系统： Windows 8.1 64 位
处理器：Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
内存：8.00GB

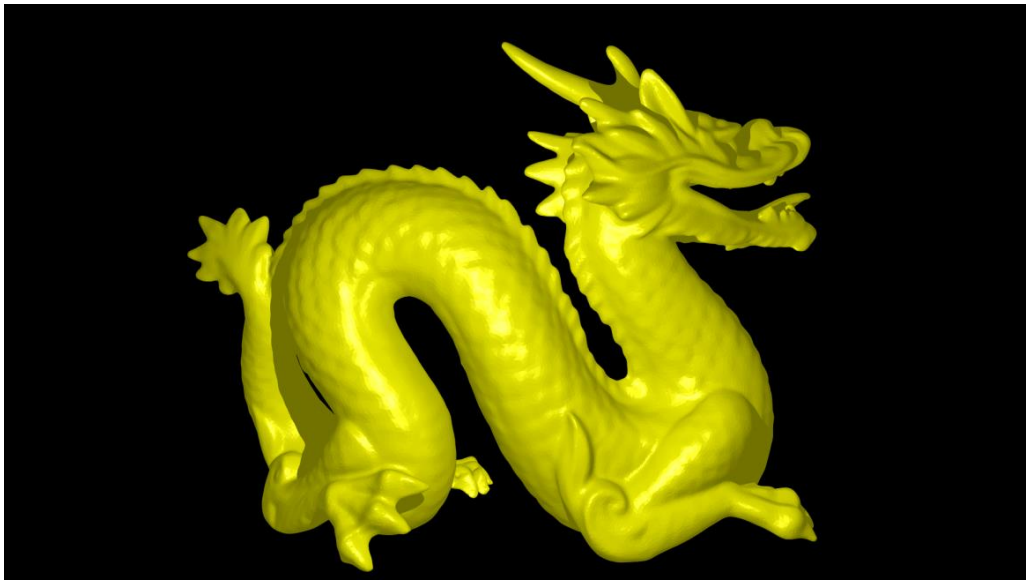
从表中可以看出，面片数均分方法在建树方面有着绝对的时间与空间优势，但在求交时效率不如 SAH 启发式方法。综合来看，当渲染图片分辨率不高，模型复杂度较低时，使用面片数均分方法；当渲染图片分辨率较高，模型较复杂时，使用 SAH 启发式方法。

此外，k-d tree 的叶节点可以改为包含不止一个面片。这样在求交检测时，可以更早遍历到叶节点。这样虽增加了枚举叶节点中面片的时间开销，但也减少了包围盒求交与遍历的时间开销。从经验来看，在处理复杂模型时将叶节点包含面片数上限设为 2 或 3 较佳。叶节点面片数上限为 3 的运行结果如下表所示，可看出渲染法拉利模型时效率有明显提升。

模型	面片数	面片数均分			SAH 启发		
		建树	节点数	总时间	建树	节点数	总时间
龙	209227	0.226s	156309	57.09s	3.753s	307437	52.82s
龙(4x 自适应 AA)	209227	0.250s	156309	233.97s	3.778s	307241	203.65s
法拉利	681893	0.621s	524287	128.14s	12.79s	1022889	109.66s

2.4 obj 模型

程序可以渲染较为复杂的 obj 模型，可以有效处理模型的法向量信息、贴图信息，并根据 mtl 材质文件给模型染色、贴图、改变透明度等。



2.5 软阴影

在光线追踪中，使用点光源会使得阴影的边缘分明。下图中模型头发的阴影就显得过于清晰锐利。

每次判断阴影处的亮度时，使用蒙特卡罗算法将每个点光源在其附近打散成若干个均分亮度的点光源（或使用面光源），取阴影的均值就能产生软阴影的效果。均分的点光源数越多，软阴影的效果越好，但随着点光源数目的增加，渲染时间也线性增加。我采用了两次判断方法，对于某个光源，若其打散的前几个[2]点光源均能照到物体或均被遮挡，则不再计算其他打散的点光源。这么做相当于只

[2] 与打散的点光源数目均为配置文件中的可调参数

在阴影的边缘处进行软阴影效果处理，兼顾了效果和效率。以下几张图展示了不同参数的软阴影效果。

2.6 平滑法向

模型多由三角面片和四边形面片构成，若直接将其当作互相独立的面片渲染，就会使得模型的表面不够光滑细致，如下左图所示。

在光线追踪算法中，物体表面的法向量与光源的位置关系影响到该点的亮度。所以通过平滑法向量，就可以做到面片与面片之间的平滑过渡。具体来说，就是在求面片上某点的法向量时，不再是统一取该面片的固有法向量，而是对该面片的顶点的法向量插值计算而得。而顶点的法向量有两种计算方法，一是根据该点所属的面片的法向量依面片的面积为权重插值而得，二是模型自带的法向量信息 vt[3]。如下右图所示，模型的脸蛋和帽子都变得光滑。

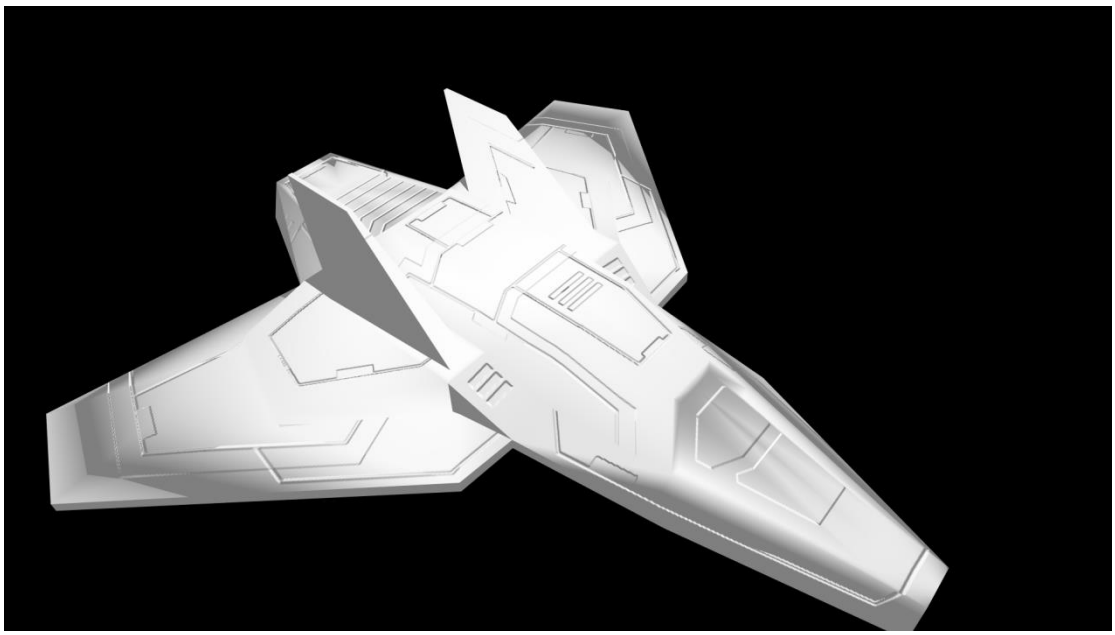
2.7 凹凸贴图

如前文所述，改变面片上各点法向量的朝向即可获得不同的光影效果。凹凸贴图正是通过一张图片[4]来表示面片上各位置的深度，通过计算凹凸贴图对应的点的在 X 方向和 Y 方向的导数，并将图片的 XY 向量映射到三维空间中的 UV 向量，对法向量进行 UV 方向的扰动，即可实现凹凸效果。

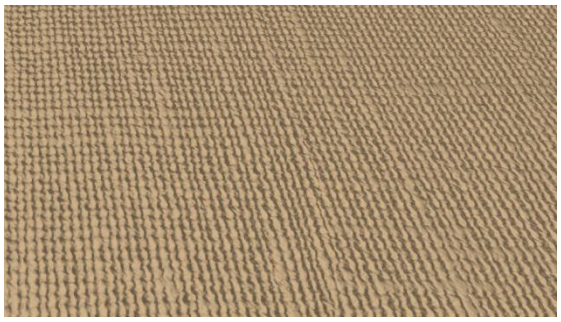
下图中所示的飞机模型，使用凹凸贴图技术比起仅使用贴图产生的光影错觉相比，层次感更加分明。

[3] 使用 obj 文件的 vt 法向量信息的好处是在不该平滑的地方不会平滑，此外模型文件提供的法向量一般比使用通用插值方法而得的法向量更好。

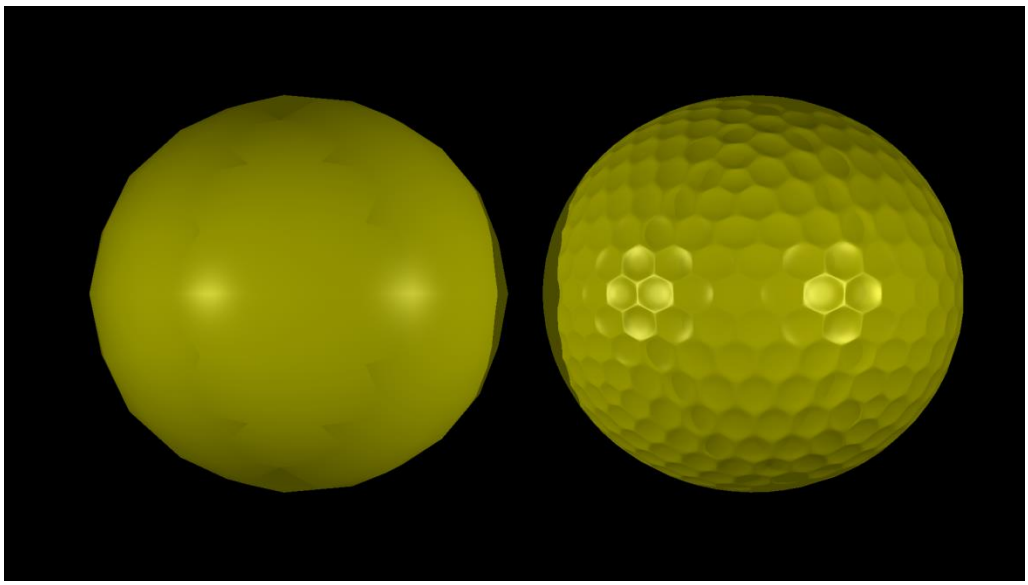
[4] Bump 文件用颜色的 rgb 值表示深度，向量场文件用 rgb 三个通道表示法向量。



仅使用一个面片与凹凸贴图技术实现的两种材质的地毯：



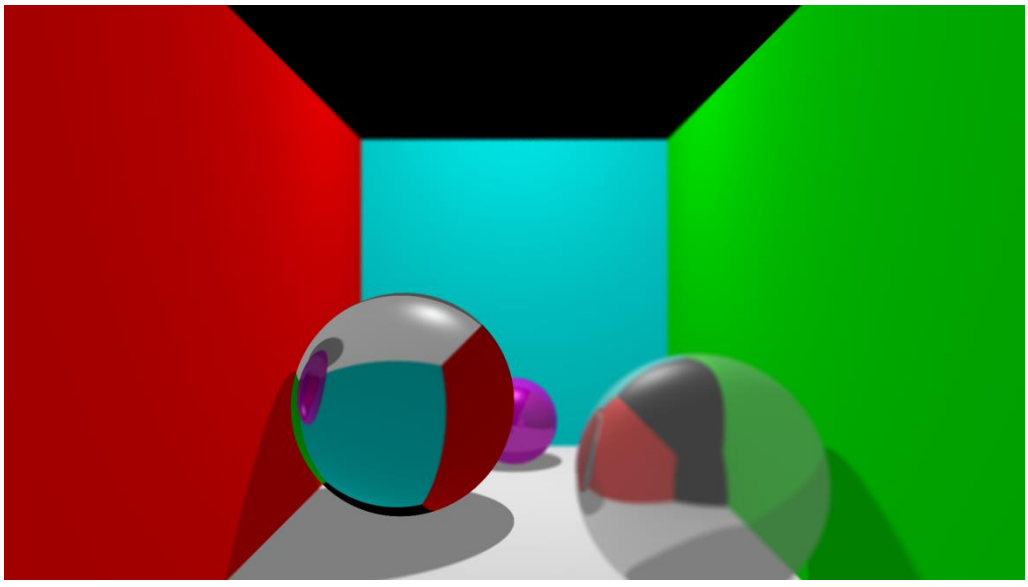
高尔夫球：凹凸贴图的一个不明显的缺陷是，在光源照不到的地方法向量的扰动是没有作用的。



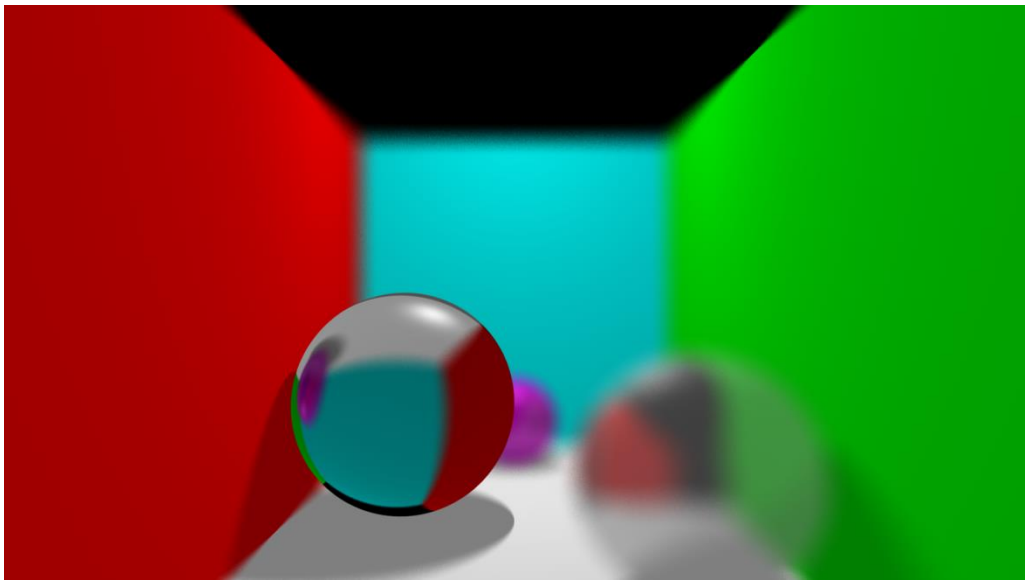
2.8 景深

通过设置镜头光圈和对焦距离，在光圈内多次随机取点，即可模拟人眼与相机的景深效果。与软阴影相同，景深效果好坏与采样次数密切相关，而渲染效率也随采样次数线性增长。故我同样采取二次判断方法，若前几次采样结果相同或相近，则不再重复采样。

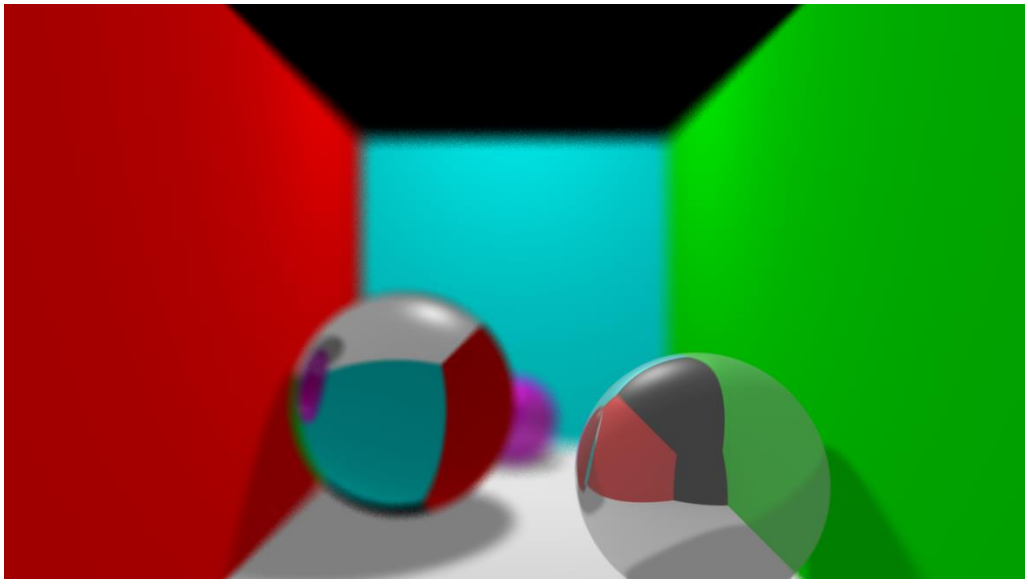
小光圈：



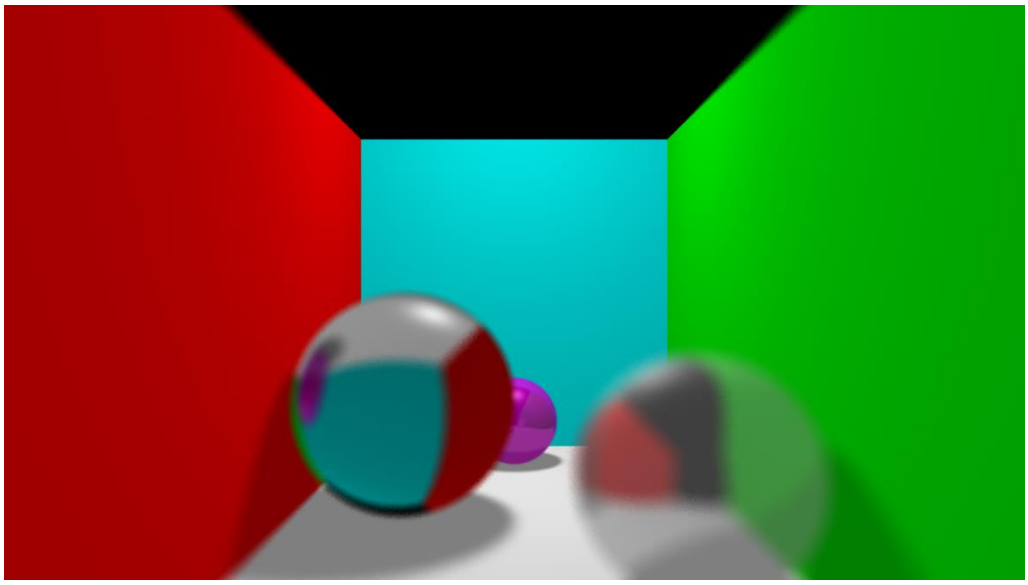
大光圈：



近处对焦：



远处对焦：



3. 网格简化

3.1 基于最短边的网格简化

基于最短边的网格简化算法即为将网格中当前最短的边的两个顶点，缩为它们的中点，直到面片数或顶点数满足数量要求或所有边的长度都大于设定的阈值时为止。显然，缩最短边的目标就是尽量维护模型的整体外观不变，而损失的细节就会比较多。我的程序能够保持面片的拓扑关系不变，这为平滑法向量的计算提供了便利。

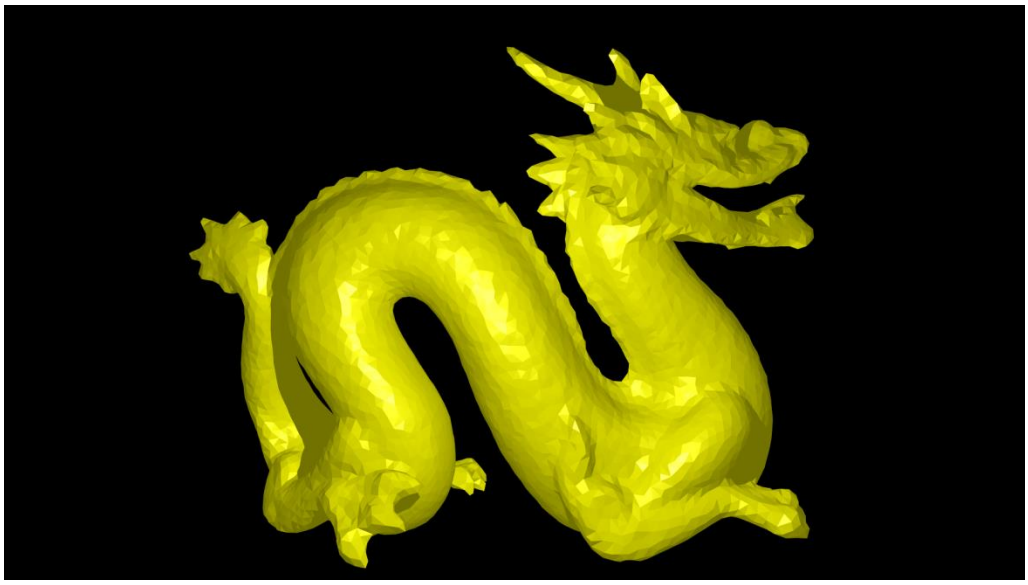
此外，该算法易于实现，效率较高，故当模型的距离较远时，使用最短边网格简化过的模型与平滑法向技术能取到不错的效果和极大的效率提升。

原模型：

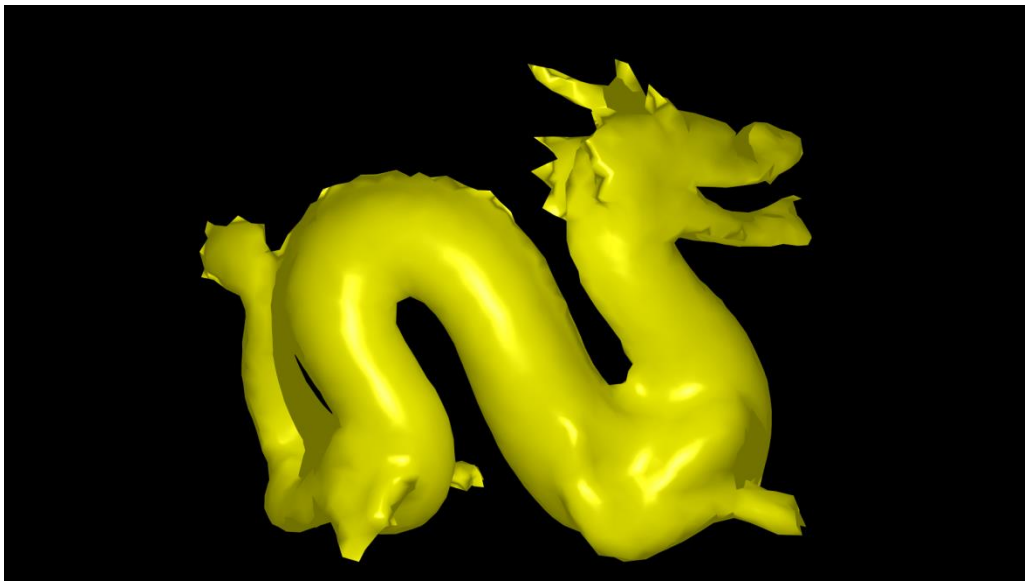
10%面片数的马：

10%面片数（平滑法向）：在平滑法向技术的支持下，简化模型的表现良好

10%面片数的龙：与原模型对比，背上的鳞片变得破损。



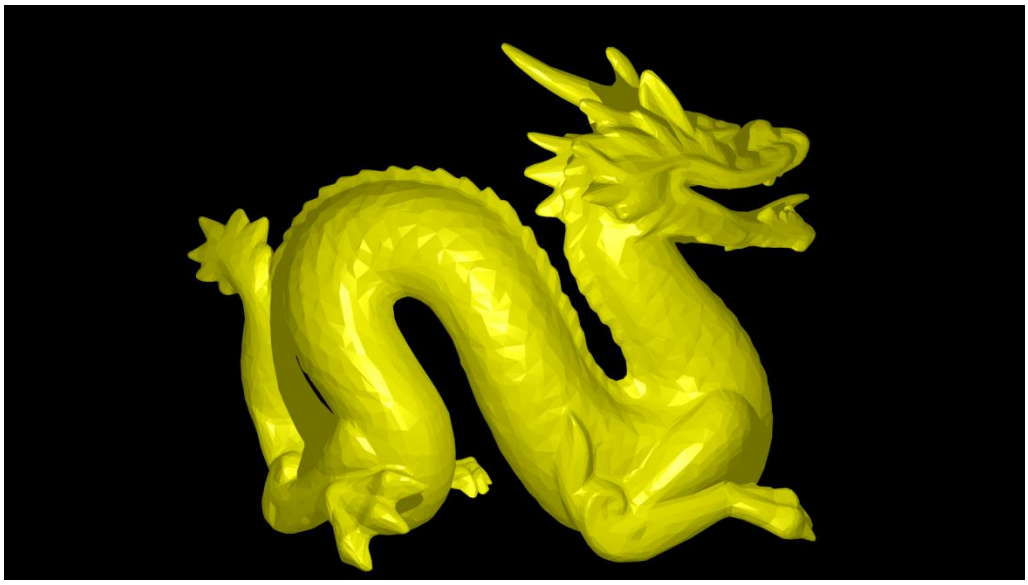
3%面片数的龙（平滑法向）：爪子和脊背的细节信息基本损失殆尽。



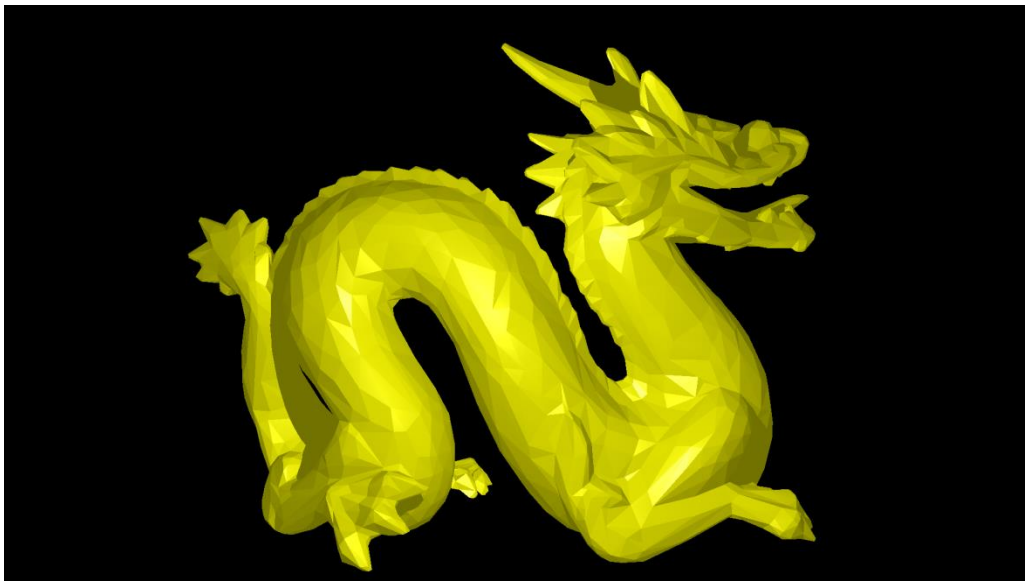
3.2 基于二次误差的网格简化

基于二次误差的网格简化算法同样属于边坍塌算法，它的目标函数是使因缩边而生成的顶点到原平面的几何距离之和最短。具体的计算公式涉及简单的线性优化与矩阵运算，在习题课与其他参考文献中均有提及，在此不再赘述。基于二次误差的网格简化在保留模型的细节方面要比最短边方法杰出许多。

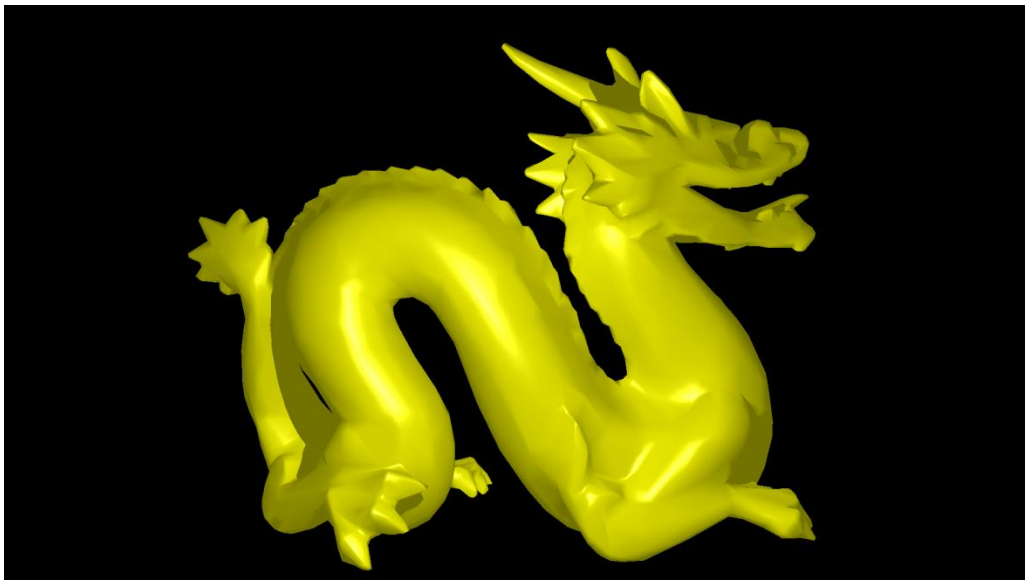
10%面片的龙：细节清晰完整



3%面片的龙：爪子锐利、背脊完整



3%面片的龙（平滑向量）：即便近看也几无瑕疵



3.3 效率对比

基于最短边的网格简化算法仅需维护点与点之间的相连关系，我使用链表来维护。我通过手写映射堆来寻找和维护最短边，单步操作$O(\log n)$。此外，边坍塌相当于将两个节点合并为一个新的节点，我使用并查集来维护这一信息，单次操作均摊$O(1)$。用人工栈来手动分配内存，避免频繁 new 和 delete 操作。

基于二次误差的网格简化算法还需维护每个点所属的三角形面片。除此之外，还需要求矩阵的逆，我的做法是使用高斯-约当消元法求解线性方程组。当方程组无解，也即矩阵不可逆时，枚举两定点间的等距点[5]，选损失函数最小的为合并点。

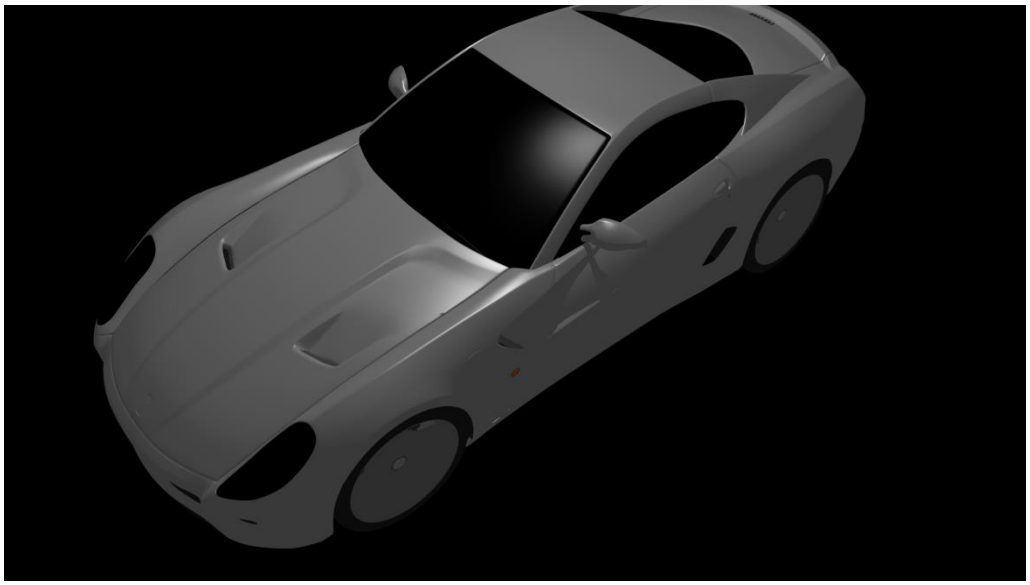
以下是对 fixed.perfect.dragon.100K.0.07.obj 进行简化的本地测试结果，其中运行时间包括文件读入输出的时间，二次误差算法在矩阵不可逆时取 4 个等距点。可以发现，最短边算法的运行时间随面片数的变化不大，由此可见其瓶颈主要在于预处理与 I/O 上，体现出其算法效率之高。

二次误差算法因每次操作需要重新维护矩阵与误差值，需要解方程组或枚举等距点，故运行时间与面片数相关性较大。但其效率仍然足够优秀，可以接受。

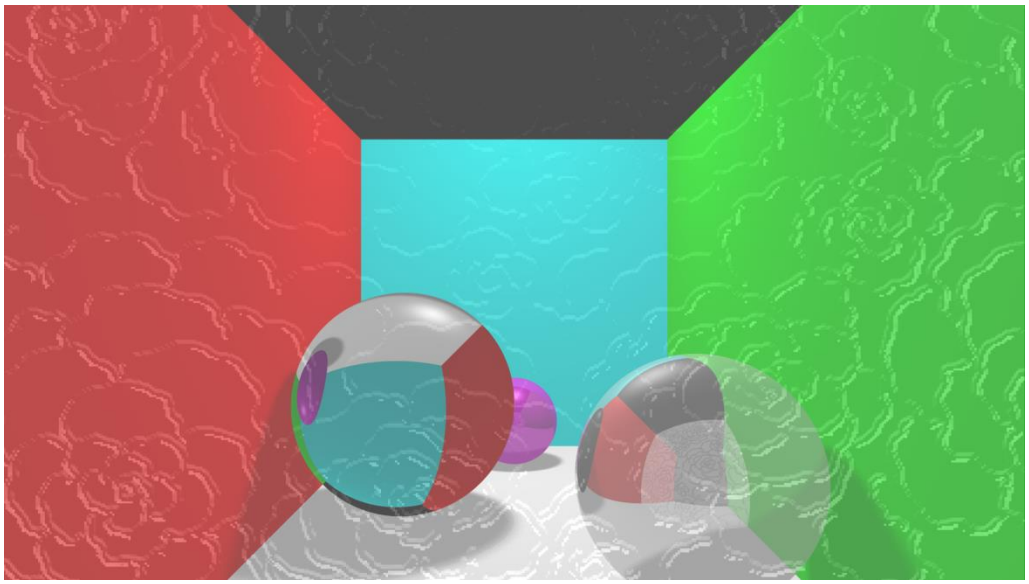
	50%面片		10%面片		3%面片	
	运行时间	面片数	运行时间	面片数	运行时间	面片数
最短边	0.941s	104628	0.983s	20940	0.990s	6295
二次误差	2.985s	104620	4.406s	20929	4.626s	6287

[5] 等距点的数目同样可在配置文件中调整.

4. 其他成果展示



使用凹凸贴图实现的简易雕花玻璃：



5. 鸣谢

感谢胡事民教授开的这门《计算机图形学基础》课，使我接触到了许多有趣的研究成果与图形学未来发展方向。

感谢陈康助教提供的 obj 文件读入框架与习题课上的慷慨帮助。

感谢古裔正同学的中厅讲座，bmp 文件 I/O 与配置文件的创意来自于他。