山东大学 计算机 学院

计算机图形学 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201400301202 | 姓名： 王瑶 | | 班级： 2014级菁英班 |
| 实验题目：[ray tracing具体实现与opengl库函数对比](https://wangyao-sdu.github.io/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/opengl/2017/06/09/raytracing%E5%AF%B9%E6%AF%94) | | | |
| 实验学时：4 | | 实验日期：2017.6.9 | |
| 实验目的：[ray tracing具体实现与opengl库函数对比](https://wangyao-sdu.github.io/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/opengl/2017/06/09/raytracing%E5%AF%B9%E6%AF%94) | | | |
| 硬件环境：  联想笔记本一台 | | | |
| 软件环境：  Windows10+VS2015 | | | |
| 实验步骤与内容：  一. 实验要求  (1) 自己编写代码实现光线的追踪，画出光照模型。  (2) 调用opengl自带的函数，通过环境光、漫反射和镜面反射参数的设置，实现实验要求一同样场景的绘制。  (3) 在同一幅图中实现两种做法的绘制，但是在实现中实现较为困难，我就绘制了两幅图来进行对比。  二．完成情况  (1) 场景的绘制：由三个小球组成，三个小球分别是蓝色，红色和黑色。这里图形的绘制前面的实验已经讲解的很清楚，不再累赘。  (2) 定义点光源：光源所在的位置坐标和光源方向。  (3) 编写程序，描绘光线照射小球时，人眼所看到的场景：物体的颜色，形状，光线直射物体时它所发生的镜面反射，以及没有直射到的部分所受到的光的漫反射。  (4) 自己实现的ray tracing代码中需要递归地进行光线追踪，因此还需要设置追踪深度。不断的以新交点为起始点，以反射光方向为方向进行在一个的求交点的过程中，直到到达我们设定的深度。  (5) 直接在opengl中调用opengl自带的函数调整设定参数，完成相同场景的绘制。  三．具体实验过程  （一）自己实现ray tracing  1. 绘制场景  一个蓝色的球，一个红色的球，一个黑色的球。  2．定义点光源  设置光源颜色为白色的点光源，设置它的位置为(0,5,-5)  m\_Primitive[4] = new Sphere(vector3(0, 5, -5), 0.1f);  m\_Primitive[4]->Light(true);  m\_Primitive[4]->GetMaterial()->SetColor(Color(1.0f,1.0f, 1.0f));  3. 根据ray tracing算法编写实现代码  光线追踪，简单地说，就是从摄影机的位置，通过影像平面上的像素位置(比较正确的说法是取样(sampling)位置)，发射一束光线到场景，求光线和几何图形间最近的交点，再求该交点的著色。如果该交点的材质是反射性的，可以在该交点向反射方向继续追踪。光线追踪除了容易支持一些全局光照效果外，亦不局限于三角形作为几何图形的单位。任何几何图形，能与一束光线计算交点(intersection point)，就能支持。光线追踪示意图如下：    4. 利用Phong模型处理光照场景  利用法向量得到球体一点的漫反射；假设人眼观察的地方在点光源的地方，由此根据观察向量和反射向量得到镜面反射；环境光是一个常数，根据点光源强度可以求出。然后将三者相加我们就可以得到球体在点光源下的光照情况。  （二）利用opengl自带函数实现场景绘制  这里使用opengl自带的函数定义它的光源位置在坐标系的右上角。  //光源的位置在世界坐标系右上角  GLfloat sun\_light\_position[] = { 1.0f, 1.0f, 0.0f, 0.0f };  利用opengl自带的函数调节参数，绘制与ray tracing一致的场景。以下是详细的代码：  GLfloat earth\_mat\_ambient[] = { 0.0f, 0.0f, 1.0f, 1.0f }; //定义材质的环境光颜色  GLfloat earth\_mat\_diffuse[] = { 0.0f, 0.0f, 1.0f, 1.0f }; //定义材质的漫反射光颜色  GLfloat earth\_mat\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //定义材质的镜面反射光颜色  GLfloat earth\_mat\_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f }; //定义材质的辐射光颜色  GLfloat earth\_mat\_shininess = 30.0f;  glMaterialfv(GL\_FRONT, GL\_AMBIENT, earth\_mat\_ambient);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, earth\_mat\_diffuse);  glMaterialfv(GL\_FRONT, GL\_SPECULAR, earth\_mat\_specular);  glMaterialfv(GL\_FRONT, GL\_EMISSION, earth\_mat\_emission);  glMaterialf(GL\_FRONT, GL\_SHININESS, earth\_mat\_shininess);  四．自己实现ray tracing算法和利用opengl库函数实现对比  虽然两种绘制场景的方法没有在同一幅图中绘制，但是把他们当作两个程序，通过两幅图的对比，我们可以发现两种实现方法还是将同一场景绘制的很相近的。具体实验结果截图可以见下面的实验结果。这说明我自己实现的这个ray tracing算法还是比较好的。 | | | |
| 分析与结论：  实验结果: 自己实现ray tracing代码和利用opengl库函数实现的实验结果截图分别如下：  自己实现ray tracing的实验结果截图：    Opengl库函数实现的结果截图：    我们可以发现两种实现方法还是将同一场景绘制的很相近的。具体实验结果截图可以见下面的实验结果。这说明我自己实现的这个ray tracing算法还是比较好的。 | | | |