A4: Shadows, Reflection & Refraction

一、概述

**1.目标：**

（1）实现阴影

（2）实现镜面反射材质

（3）实现透明折射材质

**2.知识点介绍：**

实现以上目标使用的主要是光线追踪技术。通过递归调用光线追踪函数，从新的交点处发射新的射线，并检测场景中所有物体寻找交点。

**（1）阴影实现：**得到摄像机和物体的交点A后，从该交点A出发，向每个Light方向发射射线，如果射线在Light的位置和交点A之间和其他物体相交，那么该A点的最终颜色不受该Light的影响。但是要注意射线的起点应该稍微偏离表面，防止与自身相交。

**（2）镜面反射/折射材质实现：**在材质中有reflective Color属性，如果这个值不为0就代表需要反射。实现反射的主要方法是得到摄像机和物体的交点A后，从A点出发，向*镜面反射方向*发射射线并检测与场景中所有物体的交点。根据给出的Bounces值，多次递归调用traceRay函数，可以进行多次反射。透明折射也类似，只是换成了transparent Color属性和向*折射方向*发射射线。

二、实现细节

**1. 阴影实现：**

|  |
| --- |
| if (scene->getGroup ()->intersect (ray, hit, tmin)) {  Vec3f point = hit.getIntersectionPoint ();  Vec3f n = hit.getNormal ();  for (int i = 0; i < scene->getNumLights (); i++) {  Vec3f lightColor, lightDir; float distanceToLight;  scene->getLight (i)->  getIllumination (point, lightDir, lightColor, distanceToLight);  lightDir.Normalize ();  if (shadows) {  Vec3f origin = point + n \* 0.1f;  Ray toLight = Ray (origin, lightDir); Hit m\_hit;  //再从交点处，向光源发射一条射线  bool isShaded = InShadow (toLight, m\_hit, distanceToLight);  RayTree::AddShadowSegment (toLight, 0, m\_hit.getT ());  if (isShaded) continue;//忽略此光源的影响  }  //可计算的光源的影响  color += hit.getMaterial ()->Shade (ray, hit, lightDir, lightColor);  }  color += (ambient \* hit.getMaterial ()->getDiffuseColor ());  } |

**判断是否在阴影中：**

|  |
| --- |
| bool RayTracer::InShadow (Ray &ray,Hit& hit,float dis) const {  hit = Hit (FLT\_MAX, NULL, Vec3f (0, 0, 0));  bool intersect= scene->getGroup ()->intersect (ray, hit, EPSILON);  //对于directional light，只要和别的物体相交就是阴影  //对于point light，到交点的距离超过了到point light的距离，还是没有阴影的  if (hit.getT () > dis)intersect = false;  return intersect;  } |

**2.镜面反射实现：**

|  |
| --- |
| if (bounces < 0) return Vec3f (0, 0, 0);  Vec3f reflectiveColor = hit.getMaterial ()->getReflectiveColor ();  if (bounces && (reflectiveColor != Vec3f (0, 0, 0))){  Vec3f mirror = mirrorDirection (n, ray.getDirection ());  Ray r = Ray (point, mirror); Hit tmp\_h;  //递归，终止条件是bounces<0  Vec3f v =traceRay(r,EPSILON,bounces-1,indexOfRefraction,weight,tmp\_h);  RayTree::AddReflectedSegment (r, 0, tmp\_h.getT ());  Vec3f reflection = reflectiveColor \* v; //反射系数\*得到的反射颜色  color += reflection;  } |

**计算镜面反射的方向：**

|  |
| --- |
| Vec3f RayTracer::mirrorDirection (const Vec3f &normal, const Vec3f &incoming) {  Vec3f v = incoming - 2 \*normal.Dot3(incoming) \* normal;  v.Normalize ();  return v;  } |

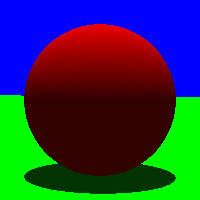
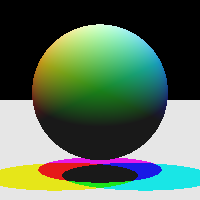
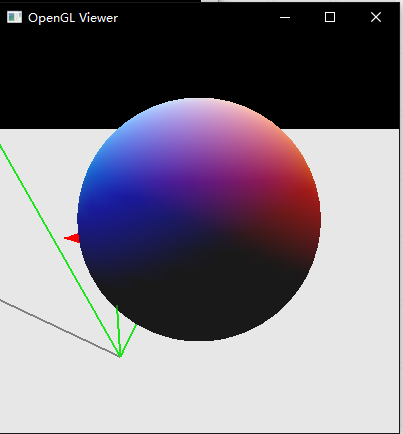
**3.透明折射实现：**

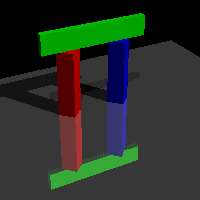
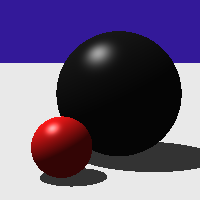
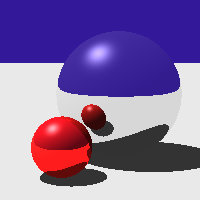
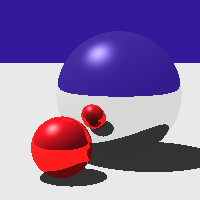
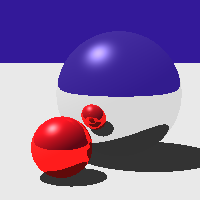
|  |
| --- |
| Vec3f transparentColor = hit.getMaterial ()->getTransparentColor ();  if (indexOfRefraction && (transparentColor != Vec3f (0, 0, 0))){  bool inside = ray.getDirection ().Dot3 (n) > 0;  float new\_index = hit.getMaterial ()->getIndexOfRefraction ();  if (inside == 1) {//如果在内部  new\_index = 1;//indexOfRefraction置为1  n = -1 \* n;//法线取反 }  Vec3f r\_dir;  if (transmittedDirection (n, ray.getDirection (),  indexOfRefraction, new\_index, r\_dir)) {  Ray r = Ray (point, r\_dir); Hit tmp\_h;  //递归，终止条件是bounces<0  Vec3f v = traceRay (r, EPSILON, bounces - 1,  new\_index, weight,tmp\_h);  Vec3f refraction = transparentColor \* v;  color += refraction;  }  } |

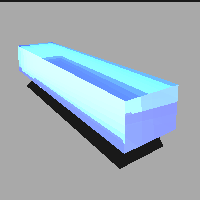
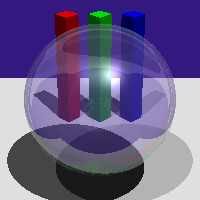
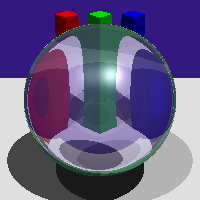
**计算折射方向：**

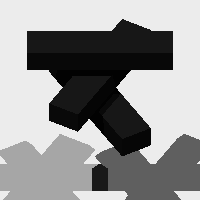
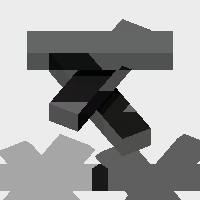
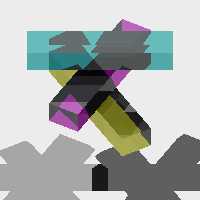
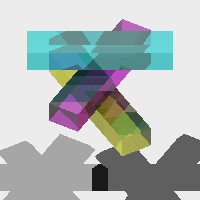
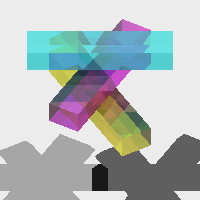
|  |
| --- |
| bool RayTracer::transmittedDirection (const Vec3f &normal, const Vec3f &incoming,  float index\_i, float index\_t, Vec3f &transmitted) {  if (fabs (index\_t) < EPSILON) return false;  float d = normal.Dot3(incoming);  float x = index\_i / index\_t;  float r = 1 - x \* x \* (1 - d \* d);  if (r < 0) return false;  r = sqrt (r);  transmitted = x \* (incoming - d \* normal) - r \* normal;  transmitted.Normalize ();  return true;  } |

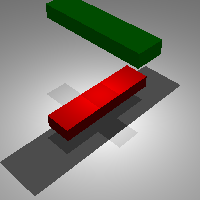
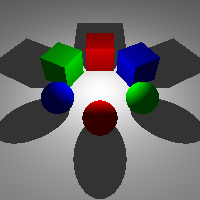
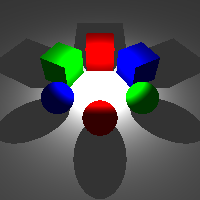
三、结果展示

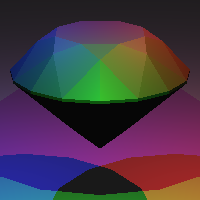
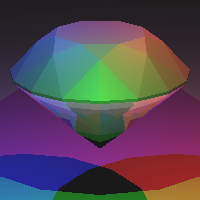
1.阴影测试：

2.反射测试：

3.折射测试：



4.Point Light测试：

5.faceted\_gem测试：

四、心得体会

通过光线追踪的技术可以得到很好的效果，加上点光源、阴影、反射、折射之后整个场景更加接近现实。