Ray Tracer---光线跟踪 实验报告

711064XX XXX

一、 实验目的

在计算机图形学课程作业中，题目要求是做 Ray Tracing 或 碰撞检测， 其中对 Ray Tracing 的要求是：

(1) 多种形状物体，Ball, box 等

(2) 包含多种材质物体：纯镜面反射、透明物体、纯漫反射、半透明物体等

(3) Moving in a 3D world

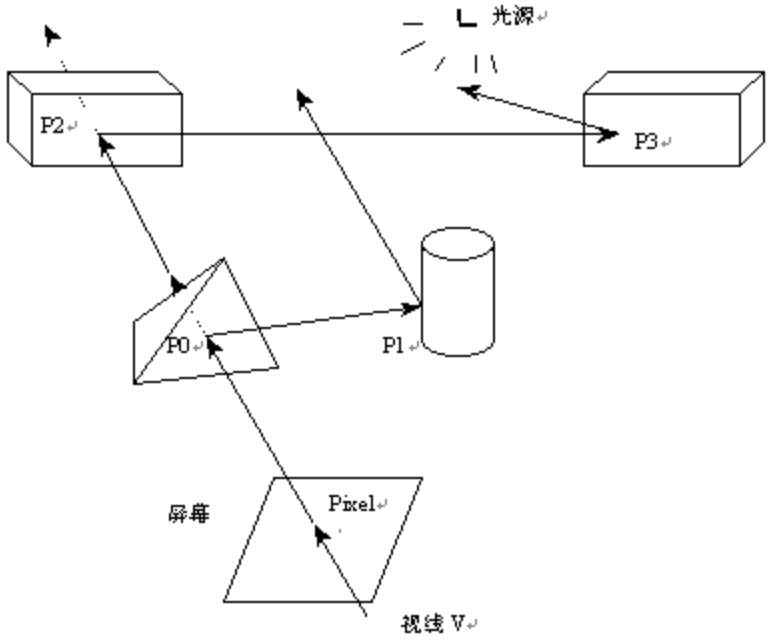
(4) environment texture

二、 实验原理

在这次实验中，使用了真正的光线跟踪算法，而不是采用环境纹理来反映周围环境。

1、 光线跟踪简介

光线跟踪是一种真实地显示物体的方法，该方法由 Appel 在 1968 年提出为了生成在三维计算机图形环境中的可见图像，光线跟踪是一个比光线投射或者扫描线渲染更加逼真的实现方法。这种方法通过逆向跟踪与假象的照相机镜头相交的光路进行工作，由于大量的类似光线横穿场景，所以从照相机角度看到的场景可见信息以及软件特定的光照条件，就可以构建起来。当光线与场景中的物体或者媒介相交的时候计算光线的反射、折射以及吸收。由于一个光源发射出的光线的绝大部分不会在观察者看到的光线中占很大比例，这些光线大部分经过多次反射逐渐消失或者至无限小，所以对于构建可见信息来说，逆向跟踪光线要比真实地模拟光线相互作用的效率要高很多倍。计算机模拟程序从光源发出的光线开始查询与观察点相交的光线从执行与获得正确的图像来说是不现实的。



2、经典光线跟踪算法

```
对图像中的每一个像素 {
  创建从视点通过该像素的光线
  初始化 最近 T 为 无限大，最近物体 为 空值
```

```
对场景中的每一个物体 {
    如果光线与物体相交 {
        如果交点处的 t 比 最近T 小 {
            设置 最近T 为焦点的 t 值
            设置 最近物体 为该物体
        }
    }
}

如果 最近物体 为 空值{
    用背景色填充该像素
} 否则 {
    对每个光源射出一条光线来检测是否处在阴影中
    如果表面是反射面，生成反射光；递归
    如果表面透明，生成折射光；递归
    使用 最近物体 和 最近T 来计算着色函数
    以着色函数的结果填充该像素
}
}
```

由以上经典的光线追踪算法可以发现，在此算法中，环境中的物体等模型，并不是一次性的画好的，而是对整个场景一个像素一个像素的画上去的，光线跟踪算法中的每一根光线要与场景中的每一个物体所含的每一个面求交。

三、 光线跟踪算法实现

1、 计算观察光线

首先需要确定光线的数学表达式。一条光线实际上只是一个起点和一个传播方向，假设起点为 O（x1，y1，z1），屏幕上一点为 D（x2，y2，z2），则光线的方向 dir（x3，y3，z3）为：

dir*=*O* – *D；

即

x3 = x1 – x2; y3 = y1 – y2 ; z3 = z1 – z2;

在程序中，光线的起点定义为：

vector3 o(0, 0, -5);

方向为：

vector3 dir = vector3(m_SX, m_SY, 0) - o;

由此可以确定一条光线

Ray r(o, dir);

然后就需要求出与该光线相交的物体中的最近的交点

2、 光线与球体相交

球体由方程$(x-a)^2+(y-b)^2+(z-c)^2=r^2$ 确定，求光线是否与方程相交，只需计算方程组

$$(x-x_1)^2+(y-y_1)^2+(z-z_1)^2=R^2$$

$$\boldsymbol{e}+\boldsymbol{d}\,t=0$$

有无实数解即可。

若令 $c(x_1,y_1,z_1)$为圆心，将二式带入一式整理可得，

$$(\boldsymbol{d}\cdot\boldsymbol{d})t^2+2\boldsymbol{d}\cdot(\boldsymbol{e}-\boldsymbol{c})t+(\boldsymbol{e}-\boldsymbol{c})\cdot(\boldsymbol{e}-\boldsymbol{c})-R^2=0$$

这里，除了参数 t 外所有的都是已知的，所以也就是标准的一元二次方程，即

$$At^2+Bt+C=0$$

二次解下中根号下的项 $\mathbf{B^2-4AC}$ 为判别式，它可以说明有多少实数解。如果判别式为负，球和直线没有交点。如果判别式为正，则有两个解；一个解是光线进入球的位置，另一个是离开的位置。如果判别式为零，光线与球相切并只有一个交点。代入球的方程中，并消除公共因子得

$$\boldsymbol{t}=\frac{-\boldsymbol{d}\cdot(\boldsymbol{e}-\boldsymbol{c})\pm\sqrt{(\boldsymbol{d}\cdot(\boldsymbol{e}-\boldsymbol{c}))^2-(\boldsymbol{d}\cdot\boldsymbol{d})((\boldsymbol{e}-\boldsymbol{c})\cdot(\boldsymbol{e}-\boldsymbol{c})-\boldsymbol{R2})}}{(\boldsymbol{d}\cdot\boldsymbol{d})}$$

在实际实现中，在计算其他项之前应该首先检查判别式的值。

在程序中的具体实现如下：

```
int Sphere::Intersect( Ray& a_Ray, float& a_Dist )
{
	vector3 v = a_Ray.GetOrigin() - m_Centre;
	float b = -DOT( v, a_Ray.GetDirection() );
	float det = (b * b) - DOT( v, v ) + m_SqRadius;
	int retval = MISS;
	if (det > 0)
	{
		det = sqrtf( det );
		float i1 = b - det;
		float i2 = b + det;
		if (i2 > 0)
		{
			if (i1 < 0)
			{
				if (i2 < a_Dist)
				{
					a_Dist = i2;
					retval = INPRIM;
				}
			}
			else
```

```
                {
                    if (i1 < a_Dist)
                    {
                        a_Dist = i1;
                        retval = HIT;
                    }
                }
            }
        }
        return retval;
}
```

3、 与平面相交

假设平面方程为 Ax + By + Cz +d=0 ，同理，将平面的法向量与直线的方向向量做差乘运算，若结果为零，则说明光线平行于平面没有交点。

在程序中的实现如下：

```
int PlanePrim::Intersect( Ray& a_Ray, float& a_Dist )
{
    float d = DOT( m_Plane.N, a_Ray.GetDirection() );
    if (d != 0)
    {
        float dist = -(DOT( m_Plane.N, a_Ray.GetOrigin() ) + m_Plane.D) / d;
        if (dist > 0)
        {
            if (dist < a_Dist)
            {
                a_Dist = dist;
                return HIT;
            }
        }
    }
    return MISS;
}
```

4、 颜色的确定

一旦我们知道了光线在传播的过程中与哪些物体相交了，我们就可以这一点的颜色，而单纯的使用物体颜色，会使图像看起来很不自然，所以我们将会计算由两个光源产生的散射的阴影效果，两个光源都会对物体的颜色起作用

下面这段代码还计算了从交点（'pi'）到光源('L')的向量，并且通过计算这个向量和物体表面的法向量的点乘积来确定该点的光照强度。这样的计算会产生这样的效果：那些面对光源的点要比其他的点要更加明亮，背对光源的点将不会被照亮。

```
// calculate diffuse shading
vector3 L = ((Sphere*)light)->GetCentre() - pi;
NORMALIZE( L );
vector3 N = prim->GetNormal( pi );
if (prim->GetMaterial()->GetDiffuse() > 0)
{
    float dot = DOT( L, N );
    if (dot > 0)
    {
        float diff = dot * prim->GetMaterial()->GetDiffuse() * shade;
        // add diffuse component to ray color
        a_Acc += diff * light->GetMaterial()->GetColor() *
prim->GetMaterial()->GetColor();
    }
}
```

5、 反射的计算

计算一个已知法向量的点的反射光线，可以使用以下公式

$$\boldsymbol{R} = \boldsymbol{V} - 2(\boldsymbol{V} \cdot \boldsymbol{N})\boldsymbol{N}$$

其中 ***R*** 是反射光线的方向向量，***V*** 是入射光线的方向向量，***N*** 是该点的法向量

计算反射光线并递归跟踪的关键代码如下：

```
float refl = prim->GetMaterial()->GetReflection();
if (refl > 0.0f)
{
    vector3 N = prim->GetNormal( pi );
    vector3 R = a_Ray.GetDirection() - 2.0f * DOT( a_Ray.GetDirection(), N ) * N;
    if (a_Depth < TRACEDEPTH)
    {
        Color rcol( 0, 0, 0 );
        float dist;
        Raytrace( Ray( pi + R * EPSILON, R ), rcol, a_Depth + 1, a_RIndex, dist );
        a_Acc += refl * rcol * prim->GetMaterial()->GetColor();
    }
}
```

6、 折射的计算

光线跟踪同样可以构造经折射的光线并继续递归跟踪，并最后将遇到的颜色信息加到最近的交点上。折射时可根据折射定律进行计算，可以使用以下式子

$$\boldsymbol{T} = (n * \boldsymbol{V}) + (n * (-\boldsymbol{N} \cdot \boldsymbol{V}) - \mathbf{SQRT}(1.0 - (n * (\boldsymbol{N} \cdot \boldsymbol{V})^2))) * \boldsymbol{N};$$

其中，***T*** 是折射光线的方向向量，***V*** 是入射光线的方向向量，***N*** 是该点的法向量，n 是两种材料的折射率的比值

计算折射光线并递归跟踪的关键代码如下：

```
// calculate refraction
float refr = prim->GetMaterial()->GetRefraction();
if ((refr > 0) && (a_Depth < TRACEDEPTH))
{
	float rindex = prim->GetMaterial()->GetRefrIndex();
	float n = a_RIndex / rindex;
	vector3 N = prim->GetNormal( pi ) * (float)result;
	float cosI = -DOT( N, a_Ray.GetDirection() );
	float cosT2 = 1.0f - n * n * (1.0f - cosI * cosI);
	if (cosT2 > 0.0f)
	{
		vector3 T = (n * a_Ray.GetDirection()) + (n * cosI - sqrtf( cosT2 )) * N;
		Color rcol( 0, 0, 0 );
		float dist;
		Raytrace( Ray( pi + T * EPSILON, T ), rcol, a_Depth + 1, rindex, dist );
		Color absorbance = prim->GetMaterial()->GetColor() * 0.15f * -dist;
		Color transparency = Color( expf( absorbance.r ), expf( absorbance.g ),
expf( absorbance.b ) );
		a_Acc += rcol * transparency;
	}
}
```

7、 Phong 明暗处理

当物体表面被光源照亮时，会形成一个高亮的光斑。这个光斑会随着视点的移动而移动，而不是固定的。Phong 提出的光照模型，实际上就是把反射光的方向向量考虑了进来。

```
intensity = diffuse * (L.N) + specular * (V.R)n
```

上式中的 L 是从交点到光源的向量，N 是平面的法向量，V 是观察方向，R 是 L 在平面上的反射向量，可以发现这个公式同时包含了漫反射和镜面反射光，实现的代码如下：

```
vector3 V = a_Ray.GetDirection();
vector3 R = L - 2.0f * DOT( L, N ) * N;
float dot = DOT( V, R );
if (dot > 0)
{
	float spec = powf( dot, 20 ) *
prim->GetMaterial()->GetSpecular() * shade;
// add specular component to ray color
a_Acc += spec * light->GetMaterial()->GetColor();
}
```

8、 阴影效果

为了计算阴影，需要在算法中判断点是否在阴影中，对于场景中的每个光源都创建一条阴影光线，再判断对场景中的所有物体是否相交，若相交，则说明物体在阴影中而光源是不可见的，并置 shade = 0；否则，说明物体不再阴影中，光线可以照射到该点，置 shade = 1；而 shade 最终的值在 0 和 1 之间的话表示该点对部分光源是可见的，其他的是不可见的。

```
// handle point light source
float shade = 1.0f;
if (light->GetType() == Primitive::SPHERE)
{
	vector3 L = ((Sphere*)light)->GetCentre() - pi;
	float tdist = LENGTH( L );
	L *= (1.0f / tdist);
	Ray r = Ray( pi + L * EPSILON, L );
	for ( int s = 0; s < m_Scene->GetNrPrimitives(); s++ )
	{
		Primitive* pr = m_Scene->GetPrimitive( s );
		if ((pr != light) && (pr->Intersect( r, tdist )))
		{
			shade = 0;
			break;
		}
	}
}
```

四、 程序效果截图