基于体积绘制的光线跟踪算法

骆春桃 [1]

[1](东南大学 计算机科学与工程学院, 南京 210000)

摘 要: 体绘制区别于传统的基于面的绘制，能显示出对象体的丰富的内部细节，观察者不仅可以看到被检物体的表面，而且可看到其内部基本组成，对于科学研究和工程技术具有重要的意义。本文介绍了光线跟踪的基本原理，给出了基于体积绘制的光线跟踪算法，实验结果表明，该方法可以体现物体深层的研究。

关键词: 光线跟踪;体绘制;可视化

Ray tracing algorithm based on volume rendering

Luo Chun-Tao[1]

[1](School of Computer Science and Engineering, Southeast University, Nanjing 210000, China)

Abstract: Volume rendering is different from conventional surface-based rendering, it can display object within the body in rich details. The observer can not only see the surface of the object, and may see its basic form inside.It is of great significance for science and engineering. This article describes the theory of ray tracing and gives the ray tracing algorithm based on volume rendering. The experiments showed that this method can reflect the deep study of the object.

Key words: Ray tracing;volume rendering; visualization

科学计算可视化是研究如何把科学数据，无论是通过计算还是从测量获得的数值，转换成可视的、能帮助科学家理解的信息的计算方法。一幅图像可以把大量的抽象数据有机地组合在一起，从而摆脱直接面对大量抽象数字组合成的复杂形态而很难发现其内在规律的局面。随着虚拟现实（Virtual Reality,VR）技术的流行，真实感图形绘制技术越来越受到重视。真实感图形绘制技术是指，先在计算机中构造出所需场景的几何模型，再根据假定光照条件计算画面上可见景物表面的光亮度，使观者产生如临其境、如观其物的视觉效果[1]。所谓的绘制，就是基于光照模型和光学几何生成一幅真实感图形的过程。光线跟踪算法是绘制真实感图形的基础算法之一，体光线跟踪算法则是在其基础上的改进。

1 光线跟踪基本原理

光线跟踪（Ray tracing）是来自于几何光学的一项通用技术，它通过跟踪与光学表面发生交互作用的光线从而得到光线经过路径的模型。光线跟踪的目的是为了模拟自然现象：能见到各种颜色是因为太阳发射出来的光线经过各种自然物体的反射或折射后，最终进入眼睛。

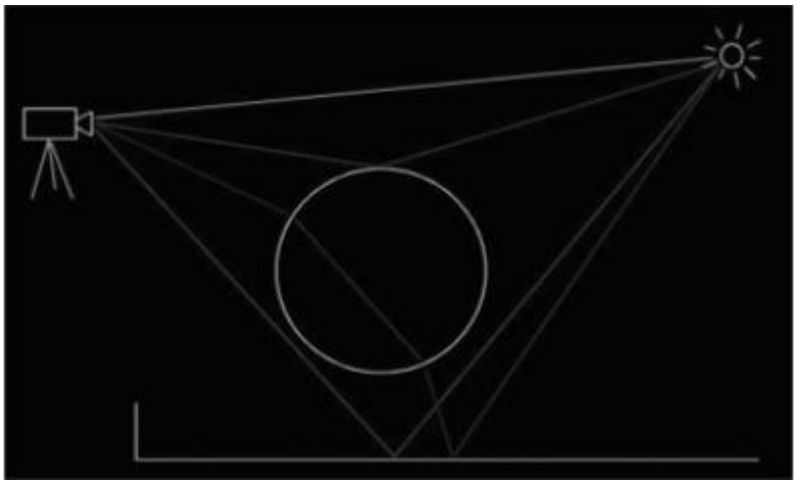

图 1 光线跟踪原理示意图

如图 1 所示，光线有直接从太阳到照相机的，有被场景反射后到达照相机的，也有被玻璃球折射后命中相机的。图中没有画出的是那些无法到达观察者的光线，这些光线也是不从光源往照相机进行跟踪的原因，而是采用相反的路径。由此得到一个启示：与其等待光源发射一条光线穿过一个目前颜色或是黑色的像素，不如自己从照相机发射光线去穿过平面的每个像素，去观察这些光线能击中几何体上的哪些像素[2]。如图 2 所示。

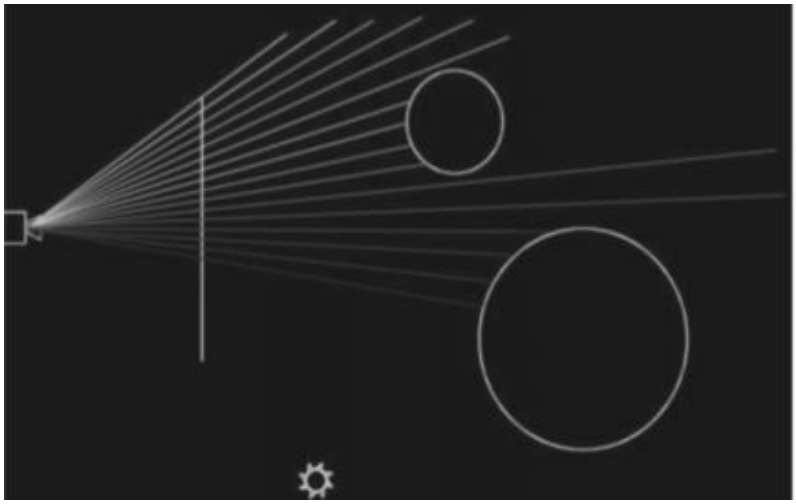

图 2 反向跟踪示意图

2 体光照模型

体绘制的核心是展示体细节而不是表面细节，是依据三维体数据，将所有体细节同时展现在二维图片上的技术。例如，CT 图片中展示的是人体的肌肉和骨骼信息，而不是表面信息。

体素，是组成体数据的最小单元，一个体素表示体数据中三维空间某部分的值，体素相当于二维空间中像素的概念[3]。如图 3 中，每个小方块代表一个体素。通常我们看到的体数据都会有一个体素分布的描述，即，该数据由 n*m*t 个体素组成，表示该体数据在 X、Y、Z 方向上分别有 n、m、t 个体素。在数据表达上，体素代表三维数组中的一个单元。假设一个体数据在三维空间上 256*256*256 个体素组成，则，如果用三维数组表示，就必须在每一维上分配 256 个空间。

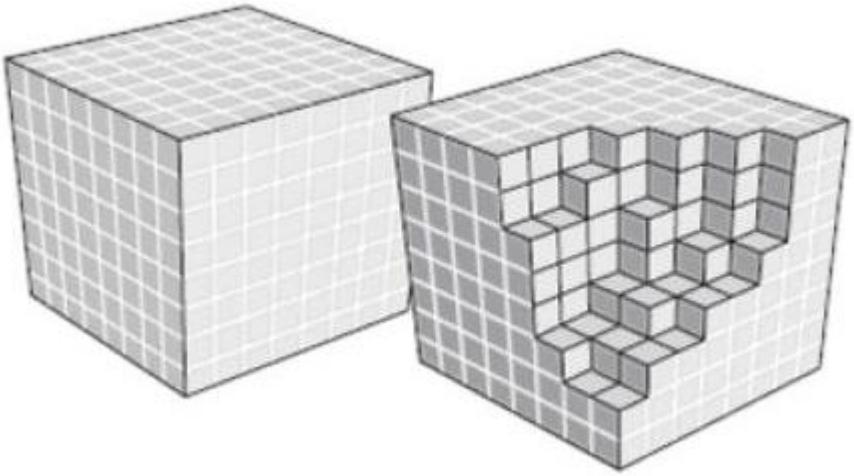

图 3 体数据中的体素

体绘制方法的基本思想是先通过转换函数对每个体素赋予颜色值和不透明度值，然后再根据各体素所在点的光照模型和灰度梯度计算出相应像素的光照强度，最后再计算出全部采样点对屏幕像素的贡献，绘制生成最终图像。由此可见，体光照模型对体绘制的结果的真实性极为重要。

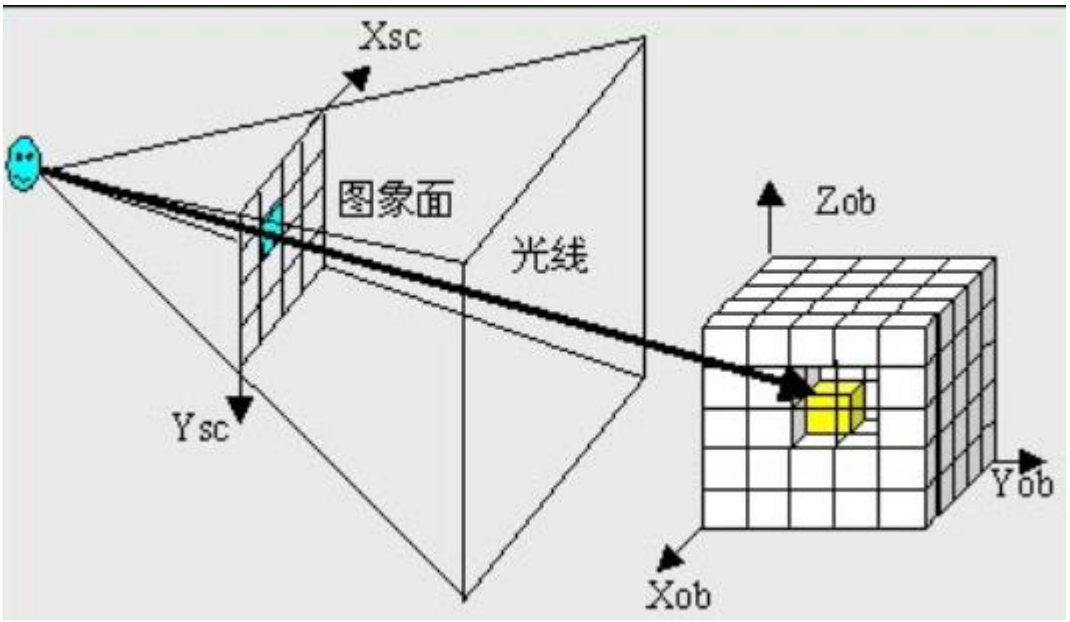

图 4 光线跟踪几何模型

如图 4 所示，源点 $P_{eye}=(x_{eye}, y_{eye}, z_{eye})$，像素阵列 $\{P_{ij}\}$，物体坐标系 $(x_{obj}, y_{obj}, z_{obj})$，图象坐标系 $(x_{scr}, y_{scr}, z_{scr})$，光线方向矢量 d=(dx, dy, dz)。被显示景物处于由源点和象素阵列形成的视锥体内。其中，采用了两种坐标系，即物体坐标系和图像坐标系，可通过旋转和平移等变换联系起来，表示为[4]

$$P_{ob}=[R]\cdot P_{sc}+T$$

其中，[R]为旋转矩阵，T 为平移矢量。

3 算法描述

3.1 光线的定义

光线由图象空间中的一条直线定义，起始点为(x_o, y_o, z_o)，其方向为归一化矢量 d=(d_x, d_y, d_z)，则可由参数形式给出

$$\begin{bmatrix} x_{scr} \\ y_{scr} \\ z_{scr} \end{bmatrix}^T (t) = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix}^T + t \cdot \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}^T$$

通常在图像空间中，“眼”的位置为(0,0,z_{eye})，同时在许多应用中常采用平行投影，上式简化为图像空间中的光线方程

$$x_{scr}(t)=x_0, \quad y_{scr}(t)=y_0, \quad z_{scr}(t)=z_0$$

则在物体空间中的光线方程为

$$\begin{bmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \end{bmatrix}^T (t) = \begin{bmatrix} x_o \\ y_o \\ t \cdot z_o \end{bmatrix}^T \cdot \mathbf{R} + T$$

3.2 确定光线与物体相交情况

如果某一对象的光线跟踪计算开销较大，则可将该对象置入低开销的包围对象中以进一步执行光线跟踪计算。这时，包围体对象将完全封装另一测试对象，由此引入包围盒的概念。如果光线不与包围体对象相交，则该光线也不可能与当前测试对象相交[5]，包围盒一般选正方形或长方形，如图 4 中所示。

在进行光线-物体相交检测时，首先进行光线-包围盒相交检测，如相交，则计算交点，否则计算另一条光线。

3.3 计算像素值

对图像中的每一个像素，创建从视点射向该像素的光线。对场景中的每一个物体，如果物体指针指向空值 NULL，则用背景色填充该像素；如果指针指向光源，则用光源的颜色填充该像素；如果指针既不是 NULL 也不是光源，则从交点向光源发出一条光线，并判断该光线在射向光源的过程中是否被遮挡，如果被遮挡，则该交点对光源不可见；如果没有被遮挡，则从交点到光源做出一条入射光线，并将入射光线单位化，求出物体表面该交点处的法向量之后，再计算出该点的散射光强值、镜面反射光强值，并将它们加到总光强值上。

对于反射光线，先利用视线和交点处的法向向量求出视线的反射光线，并将其单位化。然后以交点为视点，以交点处的反射光线为视线递归地进行跟踪，直至达到最大递归深度，就得到了加上递归反射光强的总光强值。

最后，逐行逐个像素的将颜色值输出到屏幕上。

3.4 光线跟踪伪代码描述[1]

```
TraceRay( VECTOR Start , VECTOR Direction , int Depth , COLOR Color)
{
    VECTOR   IntersectionVoxel , ReflectedDirection , TransmittedDirection ;
    COLOR   LocalColor , ReflectedColor , TransmittedColor;
    if ( Depth >MAXDEPTH)   Color=black ;
    else{
      取 Start 为起点,方向为 Direction 的光线,S 为跟踪光线,用它与场景中的景物进行
      求交测试;
      if (无交)
          Color=Backgroundcolor;
      else{
          计算离起始点 Start 最近的交点 IntersectionVoxel ;
          确定 IntersectionVoxel 所在表面的镜面反射系数 ks 和透射系数 kt;
          if ( IntersectionVoxel 所在的表面为镜面)
          {
            计算跟踪光线 S 在 IntersectionVoxel 处的反射光线方向 ReflectedDirection
            TraceRay ( IntersectionVoxel,ReflectedDirection, Depth +1 , ReflectedColor);
          }
          if( IntersectionVoxel 所在的表面为透明面)
          {
            计算跟踪光线 S 在 IntersectionVoxel 处的规则透射光线方向
            TransmittedDirection;
           TraceRay(IntersectionVoxel,TransmittedDirection,Depth+1,TransmittedColor);
          }
            shadow( LIGHT Light , VECTOR IntersectionVoxel);
        }
    }
}

阴影的生成
shadow (LIGHT Light , VECTOR IntersectionVoxel);
{
   判定交点和光源之间是否有不透明体 ;
   if(有) Color=black ;
   else
      Color = LocalColor + ks × ReflectedColor + kt ×TransmittedColor ;
}

用光照模型计算交点 IntersectionVoxel 处的局部光亮度
LocalColorShade(OBJECT  IntersectionObject,  VECTOR  IntersectionVoxel,  COLOR
LocalColor)
{
```

确定 IntersectionObject 在 IntersectionVoxel 处的单位法向量 N,漫反射系数 k_d,双向漫反射系数,反射系数 k_s ,镜面透射系数 k_t ，及环境反射系数 k_a 和遮挡系数 f ;

LocalColor=f $\times k_a \times I_a$;

{

计算入射光线单位向量 L、虚拟镜面单位法向量 H、虚拟透明面的单位法向量 H′, 以及光源到交点的距离 d ;

$$LocalColor += (I_{PointLight} \times ((k_s \times \frac{F \times D \times G}{\pi (N \circ V) \times (N \circ L)}) + k_d \times R_d) \times (N \circ L) \times (1/d^2)$$

$$+ k_t + R_t \times \left(N \circ H^{'}\right)^m)$$

}

}

4 实验讨论

在 Visual C++ 6.0 平台上简单实现了基于体绘制的光线跟踪算法，程序最终生成一个大正方体，内部包含一个球体，球体中间又包含一个小正方体，如图 5 所示。

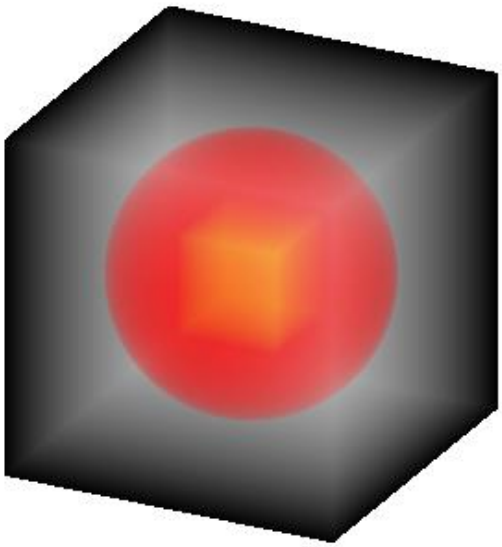

图 5 运行结果

程序主要结构介绍如下：

4.1 生成体数据

使用 GenCube 和 GenSphere 两个函数来生成正方体和球体的体数据，变量 Data 存放体积数据，Dim 存放体数据大小。

4.2 数据分类

将原始体数据的标量值映射为颜色和不透明度，实际中直接将之前生成的数据分三类：

大正方体白色、球体红色、小正方体黄色。

4.3 求取从图像空间到物体空间变换的旋转矩阵

代码如下，其中 R 表示旋转矩阵，eye 为视点位置，center 为物体参考点位置，up 为相机朝上的方向。

```
void RotationMatrix(float *R,float *eye,float *center,float *up)
{
    float XX[3],YY[3],ZZ[3];//图像空间的基向量
    ZZ[0]=eye[0]-center[0];
    ZZ[1]=eye[1]-center[1];
    ZZ[2]=eye[2]-center[2];
    CrossProd(XX,up,ZZ);
    CrossProd(YY,ZZ,XX);
    Normalize(XX,XX);
    Normalize(YY,YY);
    Normalize(ZZ,ZZ);
    //由图像空间基向量构成旋转矩阵
    R[0]=XX[0];R[1]=YY[0];R[2]=ZZ[0];
    R[3]=XX[1];R[4]=YY[1];R[5]=ZZ[1];
    R[6]=XX[2];R[7]=YY[2];R[8]=ZZ[2];
}
```

4.4 合成像素值

代码如下，其中 rgba 为合成颜色值，x_0,y_0 为二维图像像素坐标。

```
void Composite(float *rgba,int x0,int y0,float *CData,int *Dim,float *R,float *T)
{
    int stepsize=1;//采样步长
    float cumcolor[4];//累计颜色值
    cumcolor[0]=cumcolor[1]=cumcolor[2]=cumcolor[3]=0.0;
    float pos[3],dir[3];//投射光线起点、方向
    float startpos[3];//光线与包围盒近视点处的交点坐标
    float samplepos[3];//采样点坐标
    float samplecolor[4];//采样点颜色
    //采用平行投影，故在图像空间中投射光线的方向(0,0,-1),起点(x0,y0,0)
    pos[0]=x0;pos[1]=y0;pos[2]=0;
    //将光线描述转换到物体空间
    //*********************************//
    dir[0]=-R[2];dir[1]=-R[5];dir[2]=-R[8];//光线方向在物体空间的表达
    MatrixmulVec(pos,R,pos);//旋转
    pos[0]+=T[0];//平移
    pos[1]+=T[1];
    pos[2]+=T[2];
    //*********************************//
```

```
    if(Intersection(startpos,pos,dir,Dim))//判断光线与包围盒是否相交
    {
        samplepos[0]=startpos[0];
        samplepos[1]=startpos[1];
        samplepos[2]=startpos[2];
        while(CheckinBox(samplepos,Dim)&&cumcolor[3]<1)//当光线射出包围盒或累计不透明度超过 1 时中止合成
        {
            TrInterpolation(samplecolor,samplepos,CData,Dim);//三线性插值获得采样点处的颜色及不透明度
            //合成颜色及不透明度,采用的是从前到后的合成公式
            cumcolor[0] +=samplecolor[0]*samplecolor[3]*(1-cumcolor[3]);//R
            cumcolor[1] +=samplecolor[1]*samplecolor[3]*(1-cumcolor[3]);//G
            cumcolor[2] +=samplecolor[2]*samplecolor[3]*(1-cumcolor[3]);//B
            cumcolor[3] +=samplecolor[3]*(1-cumcolor[3]);                    //A
            //下一个采样点
            samplepos[0]+=dir[0]*stepsize;
            samplepos[1]+=dir[1]*stepsize;
            samplepos[2]+=dir[2]*stepsize;
        }
        rgba[0]=cumcolor[0];
        rgba[1]=cumcolor[1];
        rgba[2]=cumcolor[2];
        rgba[3]=cumcolor[3];
        return;
    }
    rgba[0]=rgba[1]=rgba[2]=rgba[3]=1.0;//若光线与包围盒不相交，赋白色
}
```

4.5 判断投射光线与包围盒是否相交

将包围盒 6 个面无限扩展，并分成 3 组，即平行于 XOY,YOZ,ZOX 平面的各有 2 个；求光线与 6 个平面的交点，从每组的 2 个交点中选出距离视点较近者，这样得到 3 个候选交点；从这 3 个候选交点中选出距离视点最远的那个。最后判断这个点是否落在包围盒内，若在，即为光线与包围盒的靠近视点处的交点。

4.6 显示函数

使用 OpenGL 的绘图函数显示图像。

```
void Mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glDrawPixels(WIDTH,HEIGTH,GL_RGBA,GL_FLOAT,Image);
    glFlush();
}
```

5 结论

本文介绍了基于体积绘制的光线跟踪算法，给出了基本原理以及具体的算法实现，有较好的效果。用体光线跟踪算法可生成复杂景物的高质量的图形，具有处理速度高、应用范围广等优点。由于时间有限，只实现了一个简单的体积绘制图像，未实现更为复杂图形的绘制，且算法还不够完善，比如对于计算量较大的求交，计算复杂度较高。

参考文献

[1] 曹莹，顾耀林，王骏. 基于体单元的光线跟踪算法[J]. 江南大学学报：自然科学版，2002,1(4):370-373.

[2] 白改朝，杨剑. 光线跟踪技术的原理与实现[J]. 电脑编程技巧与维护，2010(23):59-60.

[3] 康玉之. GPU Programming And Cg Language Primer 1rd Edition[M]. 北京：半山工作室，2009,9：152-159.

[4] 罗立民, Jean Louis Coatrieux. 医学图象中体积数据的光线跟踪法显示[J]. 东南大学学报(自然科学版), 1992, 22(6):7-12.

[5] Kevin Suffern. 光线跟踪算法技术[M]. 北京：清华大学出版社，2011,3:287-291.