# 三维连续图形变换的一类算法与实现

陈国华
(广东职业技术师范学院计算机科学系，广州 510633)

**摘 要** 概括地将当前主要的几种连续图形变换方法加以分类，并对其中非常重要的一类，即连续图形变换方法——隐式矩阵计算法在构造图形对象时的应用，提出了一种新的解决方案. 具体说，主要解决了如下问题：假如一个任意点 $P$ 绕对称轴 $R_1$ 旋转角度 $\theta_1$，得到点 $P_1$，再绕对称轴 $R_2$ 旋转角度 $\theta_2$，得到点 $P_2$，那么能否合并这两次旋转，即找到一个对称轴 $R_3$，使得 $P$ 绕 $R_3$ 旋转一个角度 $\theta_3$ 即可得到点 $P_2$. 这一问题的解决，致使最终设计出了一个类似于 3D MAX 的产品——凯迪电子教具.

**关键词** 连续图形变换 四元数
**中图法分类号**：TP391.41 **文献标识码**：B **文章编号**：1006-8961(2001)12-1240-04

# On Three-Dimensional Continuous Graphic Transformations ——Calculation and Implementation

CHEN Guo-hua
(*E-Commerce Group, Computer Science Department Guangdong Normal Polytechnic, Guangzhou* 510063)

**Abstract** In this paper, we briefly classified the several major continuous graphic transformations current in use in the area of computer graphics. Long till now, many graphics organizations or companies have their own unique graphics transformation methods. It seems they do not care for the goodness of other people's implementing methods. But in fact, many of their methods are similar in a sense. So, it is valuable to classify these various different graphics transformations into several major categories and make each one know what category his method belongs to and what goodness the other transformation category has. Besides, we had proposed a new solution to the application of implied matrix method, a most important continuous graphic transformation method, in the construction of graphics objects. In the concrete, we have solved the following problems: For a giving point $P$, let it rotate an angle $\theta_1$ round the spindle $R_1$ and reaches point $P_1$, then $P_1$ in turn rotate an angle $\theta_2$ round the spindle $R_2$ and reaches point $P_2$. Can we combine these two rotations? In other words, Can we get a spindle $R_3$ and an angle $\theta_3$ so that $P$ just rotate the angle $\theta_3$ round $R_3$ and it will reach $P_2$, the resolution to this problem result in a software's generation which is much like the famous 3D Max.

**Keywords** Continuous graphic transformation, Quaternion

## 1 问题的提出

在各类图形应用软件的开发实践中，不可避免地需要经常处理一些典型的图形变换，如：图形的平移、旋转和缩放等. 这些图形变换的标准算法在各类计算机图形学的教材中一般都有较详细的论述，但具体实现方法则是见仁见智，各有特点. 归纳起来，主要有两大类：

(1) 基于函数调用的实现方法

这是经典的实现方法，这种方法的特点是：将与变换有关的参数传给对应的函数，然后由该函数计算出相应的变换矩阵. 将其细分，又可分为两类：

① 显式矩阵计算法

收稿日期：2000-10-30；改回日期：2001-04-26

所谓显式矩阵计算法指的是：在调用函数的参数中显式地包含结果矩阵. 例如，可以定义一个矩阵类型

typedef float Matx[4][4]

然后将各类变换产生的结果显式地存入已知一个矩阵 $M$ 中，即各类调用函数定义如下：

void Translate3f(float $T_x$, float $T_y$, float $T_z$, Matx $M$)/*产生沿 $X$ 方向平移 $T_x$，沿 $Y$ 方向平移 $T_y$，沿 $Z$ 方向平移 $T_z$ 的矩阵 $M$*/

void Scale3f(float $S_x$, float $S_y$, float $S_z$, Matx $M$)/*产生沿 $X$ 方向缩放 $S_x$，沿 $Y$ 方向缩放 $S_y$，沿 $Z$ 方向缩放 $S_z$ 的矩阵 $M$*/

void Rotate3f(float $v_x$, float $v_y$, float $v_z$, float $\theta$, Matx $M$)/*产生绕 $(v_x, v_y, v_z)$ 方向旋转 $\theta$ 角的矩阵 $M$*/

② 隐式矩阵计算法

隐式矩阵计算法指的是：在调用函数的参数中不包含结果矩阵，也不明显地返回结果矩阵. 作为工业标准的 OpenGL 采用的即是隐式矩阵计算法. 它的各类变换函数定义如下：

void glTranslate3f(float $T_x$, float $T_y$, float $T_z$)/*产生沿 $X$ 方向平移 $T_x$，沿 $Y$ 方向平移 $T_y$，沿 $Z$ 方向平移 $T_z$ 的平移变换*/

void glScale3f(float $S_x$, float $S_y$, float $S_z$)/*产生沿 $X$ 方向缩放 $S_x$，沿 $Y$ 方向缩放 $S_y$，沿 $Z$ 方向缩放 $S_z$ 的缩放变换*/

void glRotate3f(float $v_x$, float $v_y$, float $v_z$, float $\theta$)/*产生绕 $(v_x, v_y, v_z)$ 方向旋转 $\theta$ 角的旋转变换*/

隐式矩阵计算法主要是借助于矩阵堆栈来发挥作用. 此时，不便明确地访问当前的变换矩阵，所有进入堆栈的矩阵都由系统控制顺序相乘，以产生当前变换矩阵，然后作用在堆栈中的各个顶点之上.

(2) 面向对象的实现方法

面向对象的实现方法将平移，旋转和缩放等方法封装到一个矩阵类之中，并为每个图形对象附着一个变换(矩阵)对象，以便在一个图形对象需要进行某类变换时，能够方便地调用附属对象(即它的成员对象)的相应方法. 矩阵类大致可以定义为：

```
template <class T>
class MatrixTemplate
{
    private:
        int numRows;
        int numColumns;
        VectorTemplate<T> * elements;
    public:
        MatrixTemplate<T>& translate(VectorTemplate<T>&);
        MatrixTemplate<T>& scale(VectorTemplate<T>&);
        MatrixTemplate<T>& translate(float,float);
        MatrixTemplate<T>& scale(float,float);
        MatrixTemplate<T>& rotate(float);
        MatrixTemplate<T>& translate(float, float, float);
        MatrixTemplate<T>& scale(float,float,float);
        MatrixTemplate<T>& rotate(float, float, float, float);
};
typedef MatrixTemplate<float> Matrix;
```

无论采用那种方式，当要对一个图形对象施加连续变换时，都必须将它与一个变换矩阵相关连. 此时图形对象本身的信息可以相对独立，例如：对于一个圆 Circle，只需要知道它的半径，但为了交互的方便，也可以给定初始圆心位置. 可以假定这个圆最初是平行于 $x$-$y$ 平面的，至于这个圆最终显示在空间的哪个位置，则由与之相关的变换矩阵决定. 倘若采用显式矩阵计算法或面向对象的实现方法，则最终每个顶点的显示输出位置都必须自己加以控制，而不能由图形引擎(如 OpenGL)加以控制，此时，两次连续的图形变换只需将前后两个矩阵相乘即可. 这种变换处理方式的开销非常大，而且不便于保存变换前一刻的状态信息. 现在的问题是：倘若采用隐式矩阵计算法，则由于不便访问当前的变换矩阵，因此存在如何实现前后两个变换矩阵的乘法的问题. 事实上，不但难于做到这一点，即便是能够做到，也不应该这样做. 人们真正需要的是一个图形对象的变换信息，具体地说，需要知道的是：这个图形对象相对于原始位置发生了多少位置偏移(TranslateX，TranslateY，TranslateZ)，在原来的基础上发生的比例变化(ScaleX，ScaleY，ScaleZ)以及绕某个对称轴旋转了多少角度(RotAngle；RotX，RotY，RotZ). 在这 3 种形式的变换中，前面两种形式的连续变换比较容易处理，问题出在第 3 种形式的变换上. 这里的问题是：

假如一个任意点 $P$ 绕对称轴 $R_1$ 旋转角度 $\theta_1$，得到点 $P_1$，再绕对称轴 $R_2$ 旋转角度 $\theta_2$，得到点 $P_2$，

那么能否合并这两次旋转,即找到一个对称轴 $R_3$,使得 $P$ 绕 $R_3$ 旋转一个角度 $\theta_3$,即可得到点 $P_2$.

## 2 四元数在连续旋转变换中的应用

一个四元数可以看作是由一个标量和一个向量组成的序偶

$$q=(s,\boldsymbol{v})$$

其中,$s$ 是一个标量,$v=(a,b,c)$是一个向量.也可以将 $q$ 表示为

$$q=s-ai+bj+ck$$

其中,i,j,k 为四元数的虚数据单位满足以下条件

$$i^2=j^2=k^2=-1 \quad i\cdot j=-j\cdot i=k$$

$$j\cdot k=-k\cdot j=i \quad k\cdot i=-i\cdot k=j$$

两个四元数 $q_1=(s_1,v_1)$,$q_2=(s_2,v_2)$的加法定义为:

$$q_1+q_2=(s_1+s_2,v_1+v_2)$$

两个四元数的乘法定义为:

$$q_1\cdot q_2=(s_1s_2-\boldsymbol{v}_1\cdot\boldsymbol{v}_2,s_1\boldsymbol{v}_2+s_2\boldsymbol{v}_1+\boldsymbol{v}_1\times\boldsymbol{v}_2)$$

1 个四元数的模定义为:

$$|q|^2=s^2+\boldsymbol{v}\cdot\boldsymbol{v}$$

1 个四元数的逆可以表示为:

$$q^{-1}=1/|q|^2(s,-\boldsymbol{v})$$

易知:对任意 2 个四元数 $q_1$ 和 $q_2$,如下关系式成立:

$$(q_1\cdot q_2)^{-1}=q_2^{-1}\cdot q_1^{-1}$$

现在,对三维空间的任意一个位置向量 $\boldsymbol{p}$,可用 1 个四元数 $P=(0,\boldsymbol{p})$来表示它.

假如点 $\boldsymbol{p}$ 绕对称轴 u(单位向量)旋转角度 $\theta$ 到达点 $\boldsymbol{p}_1$,令

$$q=(s,\boldsymbol{v})$$

其中 $s=\cos(\theta/2)$,$\boldsymbol{v}=\mathrm{u}\cdot\sin(\theta/2)$,则 $P=(0,\boldsymbol{p})$和 $P_1=(0,\boldsymbol{p}_1)$,满足如下关系

$$P_1=q\cdot P\cdot q^{-1}$$

其中,$q^{-1}=(s,-\boldsymbol{v})$.

进一步,如果 $\boldsymbol{p}_1$ 再绕对称轴 $\boldsymbol{u}_1$ 旋转角度 $\varphi$(从而确定另一个四元数 $q_1$)到达 $\boldsymbol{p}_2$,记 $P_2=(0,\boldsymbol{p})$,则有

$$P_2=q_2\cdot P\cdot q_2^{-1}$$

其中,$q_2=(s,\boldsymbol{v}_2)$

现在只要确定 $q_2=(s,\boldsymbol{v}_2)$.

更进一步的实际目标是:已知点 $\boldsymbol{p}$ 绕对称轴 u(单位向量)旋转角度 $\theta$ 到达点 $\boldsymbol{p}_1$,再绕对称轴 $\boldsymbol{u}_1$ 旋转角度 $\varphi$ 到达 $\boldsymbol{p}_2$,欲求对称轴 $\boldsymbol{u}_2$ 及旋转角 $\theta_2$,使得 $\boldsymbol{p}$ 绕 $\boldsymbol{u}_2$ 转 $\theta_2$ 角到达 $\boldsymbol{p}_2$.以下是实现这一目标的主要算法:

```
double s, s1, s2, theta, theta1, thata2;
s= cos(theta/2);
s1=cos(theta1/2);
s2=0.0;
theta2=0.0;
VERTEX V=U*sin(theta/2);
VERTEX V1=U1*sin(theta1/2);
VERTEX V2=(0.0, 0.0, 0.0);
//calculate starting
s2=s*s1-V*V1;
theta2=2*acos(s2);
V2.x=s*V1.x+s1*V.x+Wedge(V, V1).x;
V2.y=s*V1.y+s1*V.y-Wedge(V, V1).y;
V2.z=s*V1.z+s1*V.z+Wedge(V, V1).z;
//其中 Wedge(V,V1) 是一个计算 V 和 V1 的外积的函数
V2=V2/length(V2);
//calculation end, (theta2,V2) is what we need.
```

## 3 算法应用与示例

有了上述的算法基础,就可以将它运用于使用隐式矩阵计算法构造的图形对象之中.这种方法尤其适用于作为工业标准的 OpenGL 图库进行图形对象构造的场合,因为它使人们可以对平移、旋转和缩放等变换有一个统一的表述.现仍然以画圆为例,可以将圆类定义为:

```
class CKDglCircle : public CKDglObject
{
public:
    CKDglCircle();
    virtual ~CKDglCircle();
    CKDglCircle ( LPCOLOR STRUCT colorIn,
LPVERTEX center, GLdouble radius);
    //一些典型的为变换因子赋值的方法
    virtual void SetTransX(GLdouble TrX);
    virtual void SetTransY(GLdouble TrY);
    virtual void SetTransZ(GLdouble TrZ);
    virtual void SetRotX(GLdouble RtX);
    virtual void SetRotY(GLdouble RtY);
    virtual void SetRotZ(GLdouble RtZ);
    virtual void SetRotAng(GLdouble RtAng);
    virtual void SetScaleX(GLdouble ScX);
    virtual void SetScaleY(GLdouble ScY);
    virtual void SetScaleZ(GLdouble ScZ);
    //一些典型的读取变换因子值的方法
```

```
    GLdouble GetTransX();
    GLdouble GetTransY();
    GLdouble GetTransZ();
    GLdouble GetRotX();
    GLdouble GetRotY();
    GLdouble GetRotZ();
    GLdouble GetRotAng();
    GLdouble GetScaleX();
    GLdouble GetScaleY();
    GLdouble GetScaleZ();
    GLdouble GetCenterX();
    GLdouble GetCenterY();
    GLdouble GetCenterZ();
      旋转算法
    virtual void CalcRotX(double theta);
    virtual void CalcRotY(double theta);
    virtual void CalcRotZ(double theta);
      辅助函数,求向量的内积、外积和数量积
    GLdouble InnerProd(VERTEX u,VERTEX v);
    VERTEX WedgeProd(VERTEX u,VERTEX v);
    VERTEX ScalarProd(double d,VERTEX v);
      各有关变换因子
    GLdouble TranslateX;
    GLdouble TranslateY;
    GLdouble TranslateZ;
    GLdouble RotAngle;
    GLdouble RotX;
    GLdouble RotY;
    GLdouble RotZ;
    GLdouble ScaleX;
    GLdouble ScaleY;
    GLdouble ScaleZ;
    GLdouble CenterX;
    GLdouble CenterY;
    GLdouble CenterZ;
    void draw();//圆的绘制方法
protected:
    GLdouble m_Radius;//圆的半径
    LPVERTEX m_Center;//圆心
};
```

有了上述圆类对象，就可以随时在与用户的交互过程中，动态地绘制圆形．具体绘制方法如下：

```
void CKDglCircle::draw()
{
  glPushMatrix();
    glTranslated(TranslateX,TranslateY,TranslateZ);
    GLdouble alpha=0.0;
    glPushMatrix();
      glTranslated(CenterX,CenterY,CenterZ);
      glRotated(RotAngle,RotX,RotY,RotZ);
      glScaled(ScaleX,ScaleY,ScaleZ);
      glTranslated(-CenterX,-CenterY,-CenterZ);
      glBegin(GL_LINE_LOOP);
        while(alpha<2*PI)
        {
          GLdouble x=m_Center->x+
                     ((GLdouble)sin(alpha)*
                     m_Radius);
          GLdouble y=m_Center->y-
                     ((GLdouble)cos(alpha)*
                     m_Radius);
          ::glVertex3d(x,y,m_Center->z);
          alpha-=PI/128;
        }
      glEnd();
    glPopMatrix();
  glPopMatrix();
}
```

## 4　结　语

上面所介绍的方法是在具体的开发实践中对所遇到的问题提出的解决办法，它主要适用于使用隐式矩阵计算法构造图形对象、尤其是使用作为工业标准的 OpenGL 图库进行图形对象构造的场合．这种方法已成功地在广州凯迪软件公司的商业软件《凯迪电子教具》中加以运用．

**陈国华**　1961 年生，1987 年获杭州大学数学系研究生毕业证书，现为广东职业技术师范学院计算机科学系副教授，兼任广州迎新丰科讯有限公司技术部主管．主要从事计算机图形学、计算机辅助几何设计以及 ERP 等方面的研究．发表论文 10 余篇．