矢量矩阵类库与计算机图形程序设计

陈国华

(广东民族学院计算机科学系)

摘　要　本文利用 C++面向对象的新特征,推广并极大地丰富了文[1]和[2]中介绍的矢量矩阵类库,从而使之成为计算机图形程序设计的有力工具.

关键词　面向对象;类;模板;矢量矩阵;计算机图形

分类号　TP311.52

0　引　言

正如文[1]所言,在各种工程及科学计算中,经常要用到矢量及矩阵的各种运算,如加法、减法和乘法等,用以往的程序设计语言如 BASIC 语言,FORTRAN 语言,普通 C 语言等进行这类程序设计时,往往是将这些运算操作编成一些子程序或函数,然后显式地调用这些函数来处理矢量矩阵间的各种运算.这种方法一般都涉及复杂的参数传递问题,除了系统开发者外,普通用户并不易方便地使用,广大的工程计算人员和科学研究人员都很希望所使用的程序设计语言有这样的功能,即对矢量和矩阵间的运算能象写数学公式一样,如矩阵 $\boldsymbol{A}$ 等于矩阵 $\boldsymbol{B}$ 乘于矩阵 $\boldsymbol{C}$ 可以写成 $\boldsymbol{A}=\boldsymbol{B}*\boldsymbol{C}$ 等.为此有不少科技工作者(见文[1]和[2])都对这个问题进行一些探索.但文[1]由于语言工具的限制,所以分别设计了行向量类 RowVector 和列向量类 ColVector,并且定义了一个自定义数据类型 NORM-VAR,从而使问题趋于复杂化.其实无论是行向量还是列向量,它们在计算机中的表示都只是一个数组而已,之所以人为的区分它们只不过是因为用它们左乘一个矩阵或右乘一个矩阵时运算方法有所不同而已,但通过运算符的重载,这个问题可以很容易地加以解决.此外,通过使用 Borland C++的模板功能,自定义数据类型 NORM-VAR 也是不必要的,文献[2]很好地解决了这个问题.本文从计算机图形学的观点出发,将向量和矩阵类的定义加以推广,从而使它们成为计算机图形程序设计的锐利武器.

1　向量模板的定义

众所周知,在计算机图形软件的开发中,最基本的对象是点,由两点可以决定一条线段,然后再加上一些别的图形基元如圆或椭圆等,便可构成各类复杂的对象.尽管最终表现在计算机

收稿日期:1997-02-31.　陈国华,男,1961 年生,讲师.

屏幕上的点只能是二维整型数组,但在进行初始数学计算时一个点可以是三维的,并且可以是任何精度的[19].显然我们首先必须能够方便地使用计算机来处理点对象,但仅仅用一个二维或三维数组来存贮一个点对象常常是不方便的,因为在实际构图的过程中我们还需要对点或由它们生成的折线进行各种仿射变换[19].为方便地处理这个问题[19],在计算机图形学中,每个点一般都采用齐次坐标来表示[19].这样一来,一个二维点实际上是用三维数组来表示的;而一个三维点实际是用四维数组来表示的[19].下面构造的向量模板的构造函数之一恰如其分地表现了这一点,在这个类的定义中,我们还封装了二个向量的加、减、数乘,乘以及一个标量与向量的乘积等运算,这一切自然都是为了使图形变换容易进行[19].

```
template<class T>
class vectorTemplate {
private:
    int numElements;
    T * elements;
protected:
public:
    VectorTemplate(void);
    VectorTemplate(int);
    VectorTemplate(VectorTemplate<T> &vSrc);
    ~VectorTemplate(void);
    T &operator[](unsigned int index)    {return elements[index];};
    int size(void)    {return numElements;};
    void resize(int sizeIn);
    VecotorTemplate<T>& operator=(VectorTemplate<T> &);
    VecotorTemplate<T>& operator=(T);

    float length(void);
    VectorTemplate<T> &normalize(void);

    friend T operator *(VectorTemplate<T> &v1, VectorTemplate<T> &v2);
    friend VectorTemplate<T> operator+(VectorTemplate<T> &v1, VectorTemplate<T> &v2);
    friend VectorTemplate<T> operator-(VectorTemplate<T> &v1, VectorTemplate<T> &v2);
    friend VectorTemplate<T> operator *(VectorTemplate<T> &v1, T scalar);
    friend VectorTemplate<T> operator *(T scalar, VectorTemplate<T> &v1);
    VecotrTemplate<T> & operator += (VectorTemplate<T> &v);
    VecotrTemplate<T> & operator *= (T scalar);
    VecotrTemplate<T> & operator -= (VectorTemplate<T> &v);
    VecotrTemplate<T> cross(VectorTemplate<T> &v);
};
typedef VectorTemplate<float> Vector;
```

在向量模板的定义中,通过运算符的重载,使我们能象对待普遍的数学运算一样来引述两个向量或向量与标量之间的四则运算,例如:对于两个向量 $V1, V2$ 的内积,只须简单地表示为 $V1 * V2$,而对于一个标量 s 与向量 $V1$ 的乘积亦只须记着 $s * V1$,而丝毫不会引起混淆[13].

一般情况下,我们用 Vector 表示一个 float 型的向量[19].

2 矩阵模板的定义

正如上文所述，在计算机图形学中，点是用齐次坐标表示的，因此为了运算方便，一个 n × n 矩阵实际是用(n + 1) 行乘(n + 1) 列的形式存贮的，并且一个矩阵的每一行正是一个不折不扣的向量[19]所以在下面矩阵类的私有成员中我们封装了一个矩阵的行向量指针，由于我们的目的是将向量和矩阵对象用于图形程序设计，故而我们不但需要矩阵与向量的乘法运算以及由于连续变换而导致的矩阵相乘运算，我们还专门封装了平移，旋转和缩放等成员函数，以便在一个图形对象需要进行这类变换时，能够方便地调用与之相应的这类成员[19].

```
template<class T>
class MatrixTemplate {
private:
    int numRows;
    int numColumns;
    VectorTemplate<T> *elements;

    void create(int,int);

protected;

public;
    MatrixTemplate(void) {
        numRows=0;
        numColumns=0;
        elements=NULL;
     };
    MatrixTemplate(int size)    {create(size,size); };
    MatrixTemplate(int,int);
    MatrixTemplate(MatrixTemplate<T> &mSrc);
    ~MatrixTemplate(void);
    VectorTemplate<T>& operator[](int row)    {return elements[row]; };
    int nrows(void)    {return numRows; };
    int ncols(void)    {return numColumns; };

    void resize(int new_ncols, int new_nrows);

    friend MatrixTemplate<T> operator * (MatrixTemplate<T>&, MatrixTemplate<T>&);
    friend VectorTemplate<T> operator * (MatrixTemplate<T>&, VectorTemplate<T>&);

    MatrixTemplate<T>& operator *= (MatrixTemplate<T> &m)
        {*this=m*(*this); return *this; };
    MatrixTemplate<T>& operator=(MatrixTemplate<T> &);

    float determinant(void);

    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);
    MatrixTemplate<T> & setIdentity(void);
    MatrixTemplate<T> & identity(void)      {return setIdentity();};
};
typedef MatrixTemplate<float> Matrix;
```

一般我们用 Matrix 来表示一个 float 型矩阵[19].

3 应　用

我们知道,在计算机图形学中,一个复杂图形常常由一些简单图形基元组合而成,虽然这些图形基元各有特点,但它们都有色彩、类型、位置、参考点等共同特性,因而我们可以构造这些图形基元的一个基类,比如叫做 GraphBase,使各个基元都由 GraphBase 派生,尤其当用几个基元组合一个图形时,因为涉及各个元素的摆放位置,因此各个元素都应相对参考点作一些变换[19].我们自然想到应使每个元素与一个矩阵相联系,所以我们可以如下构造图形基类

```
typedef enum {
    UNKNOWNGRAPH,     //未知图形类型
    POLYLINEGRAPH,    //折线类型
    CIRCLEGRAPH,      //圆类型
    TABLEGRAPH,       //由系列基元构成的图形
}GraphType;
class GraphBase:public Matrix {
private:
    GraphType type;
    int color;
    Vector ref_point;
public:
    ......
    void setMatrix(Matrix &mIn);
    Matrix &getMatrix();
    void setColor(int colorIn);
    int getColor();
    void setReferencePoint(Vector &refpointIn);
    Vector getReferencePoint(void);
    void setType(GraphType typeIn);
    GraphType getType(void);
    void move(float xoff, float yoff);
```

```
    void move(float xoff, float yoff, float zoff);
    virtual void draw(void) { };
    virtual void fill(void) { };
    virtual void draw(class Perspective &) { };
    ......
}
```

在这个图形基类中,我们可以将每个图形归入一个特定的种类,并为之指定颜色,参考点及相关矩阵19.并且封装了相对于这个参考点进行平移的方法,以及纯虚的绘制函数和填充函数,如果给定适当的透视参数(由 class Perspective 定义),则可以按透视要求来绘制19.当然,具体的绘制方法必须在每个派生类中重载 draw()来实现19.然后我们可以由此基类派生出线段类,折线类,圆类等19这样每个派生类便都有了一个相关矩阵,比如我们现在要绘制一条线段 linesegment,则我们只须确定它的两上顶点向量 first,end 然后将它们相连即可,现在如果要把它移动一个位置或者要确定另外一条线段,则我们只须为现有的线段 linesegment 设定一变换矩阵 Mat,如:

```
setMatrix(Mat);
```

然后将 linesegment 的两个顶点 first,end 变换到新的位置

```
first=getMatrix() * first;
end=getMatrix() * end;
```

再重新连接 first 和 end 即可,在使用一系列图形基元构造复杂图形时,这种方法尤其显得方便19.

4 结束语

以上介绍的只是矢量矩阵类在计算机图形学中的一点应用思想,矢量矩阵类在计算机图形学中的应用是丰富多彩的,尤其在分形图片的制作中,其威力更是令人赞叹,详情可以参见文[5]和文[6]19.

参 考 文 献

1 赖晓平,朱桂华等,方昌钦19.一种实用的矢量矩阵类库19.计算机应用与软件,1996,13(6):38～48

2 易忠亮19.C++矩阵运算类与 Windows 应用软件19.北京:清华大学出版社,1995.1～380

3 罗振东,廖兴裕19.计算机图示学原理和方法19.上海:复旦大学出版社,1993.120～140

4 蔡士杰等19.三维图形系统 PHIGS 的原理与技术19.南京:南京大学出版社,1993.62～89

5 Marc Berger· Computer Graphics with Pascal· The Benjamin/cummings Pub· Inc, 1986.1～340

6 M Finlay, K A Blanton·用 C++设计二维,三维分形图形程序19.北京:科学出版社,1995.1～56

Vector, Matrix Classes and Computer Graphics Programming

Chen Guohua

(Dept of Computer Science Guangdong Institute for Nationalities)

Abstract　In this paper, the C++ OOP features are used to generalize and enrich the vector, and matrix classes introduced in reference [1] & [2]. And then they are utilized as powerful tools in computer graphics programming.

Key words　OOP; template; class; vector matrix; computer graphics

(上接第27页)

参 考 文 献

1　孙元和,郑远中,孙锡炎.超精研圆柱滚子凸度送料辊设计及制造工艺研究.见:滚动轴承学术委员会学术论文,1983

2　李煜伟,郑远中.MZ6220圆柱凸度滚子超精研机研究报告.机械工业部轴承研究所,1986

3　センタレス スルーフィード超仕上盤におけるクラウニング用キヤリアロ-Ⅳ.日本东洋工业株式会社.昭和54年.

A Method for Superfinishing the Convex Cylindrical Rollers

Zhou Ermin

(Industrial Office)

Abstract　This paper focuses on the building up of a mathematical model for the calculation of the shape design of guide rolls. By comparsion, the roll shape derived from this model and SK636 made by the SUPFINA corporation of German have much in common. By super finishing the guide roll, the roller's convexity can be formed, which modifies its roundness and enhances its roughness.

Key words　convex roller; superfinishing; guide roll; shape design of guide rolls