第一章　图像和彩色模型

颜色是图像文件的基本组成要素。本章介绍了像素、调色板、彩色模型等计算机图像处理的基本概念和原理。

1.1　彩色模型基本原理

1.1.1　像素和调色板

位图点阵图像(Bitmapped graphics)是最重要的一类图像格式，也是现在使用最广泛的图像形式。它通过定义一个二维数组，即一个点阵方框，用很多小点来描述一幅图像。这些图像的基本组成单位——小点通常称为像素点(pixel)。例如，常见的微机显示器 VGA 的高分辨模式 640×480，其含义为屏幕宽度为 640 个像素点，高度为 480 个像素点；由于显示器的物体尺寸基本相当于 14 英寸电视屏幕，相对固定；像素点数目越多，分辨率越高，图像越细致真实，质量也就越高。

图像有彩色图像和灰度图像之分，就如同彩色照片和黑白照片一样，前者通过定义多种不同的颜色(一般是由红、绿、蓝三种基色组合)组成，后者则是通过黑白的浓淡，用层次反映图像的细节。彩色图像需要定义调色板(pallete)，即一个数组，包含了图像中所有用到的颜色。一般来说，像素值(pixel)就是调色板(pallete)数组的下标索引。

微机环境下，调色板一般有 2 色，16 色和 256 色二种形式，由于数组下标和元素之间是乘方关系，所以相应的像素值由 1 位，4 位和 8 位数据组成，这也称为像素组成位数(bits per pixel)，简称 bpp)。在现有的微机显示器环境下，像素值的数据组织方式分为字节组合法(Pack－Byte)和位平面法(Bit－Plane)。例如同样的 16 色模式，字节组合法由 2 个 4 位像素数据组合为 1 字节数据；而位平面法(即 EGA，VGA 标准模式)，使用 4 个分离的位平面，一个像素值分别对应于 4 个位平面相同位置上的一位(Bit)。

真彩色图像为 16 位或 24 位，这时的像素值(pixel)不再是调色板数组的下标索引，而是直接的颜色数据。一般的图像格式采用 RGB 颜色模型(下节详述)，即用红(R)，绿(G)，蓝(B)三种基色的浓淡组成描述颜色信息。这样，对于 16 位数据像素值，红绿蓝三个分量各用 5 位，可以表示 $2^{15}=32768$ 种颜色；而 24 位数据像素值，红绿蓝三个分量各用 8 位，可以表示 $2^{24}=16277216$ 即 1600 万种颜色，大大超过了人眼所能分辨的颜色数目，是地地道道的真彩色！

1.1.2　彩色模型

(1) RGB 模型

通常的显示器和图像文件都采用 RGB 彩色模型，它反映了光的能量的组合，即红(Red)，绿(Green)和蓝(Blue)三种基色的比例，也称为增强型色彩模型(additile color mode)。

一般情况下，R，G，B 三种颜色分量取值范围为(0，255)，所组合的颜色由一个 RGB 三元

组表示，当红色分量为 255，绿，蓝色分量为 0 时，混合颜色为红色；配色与三元组对应关系如下所示：

黑色：　(0,0,0)
红色：　(255,0,0)
绿色：　(0,255,0)
蓝色：　(0,0,255)
白色：　(255,255,255)

对于红，绿，蓝三种基色取不同值，可以有 1600 多万种颜色选择。

(2) CMY 模型

CMY 模型是 RGB 模型的互补彩色模型，它反映了物体吸收光的能量后反射光的组合，即青(cyan)，洋红(magenta)，黄(yellow)三种基色的比例，也称为减弱型色彩模型(subtractive color model)。

CMY 模型是彩色打印设备的色彩模型，也正好与 RGB 模型互补。我们不难发现，青(cyan)是 RGB 模型中除红(Red)之外另两种颜色绿(Green)和蓝(Blue)的混合色；同理洋红(Magenta)是红和蓝的混合色；黄(Yellow)是绿和红的混合色。用数学矩阵的方式表示出来就是：

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

在实际的彩色打印系统中，为了获得纯黑色，一般不使用青(Cyan)，洋红(Magenta)和黄(Yellow)三种基色的混合色，而是另外定义黑色，这时也称为 CMYK 模型，这也就是为什么有的彩色打印机有四条打印色带或彩色喷头的原因。

由于黑色反映的是一种灰度层次，所以黑色的定义方法也有多种情况，一般由 CMY 模型转化为 CMYK 模型的公式为：

K=min(C,M,Y)　　即 K 为 C,M,Y 三基色的最小取值

C−=K

M−=K

Y−=K

(3) YCbCr 模型

YC_bC_r 模型是彩色图像转化为灰度图像的彩色模型，即把 RGB 三基色的亮度对应为明暗灰度值，一种经典的对应公式如下：

$$Y=0.299\times R+0.587\times G+0.114\times B$$

其中 R,G,B 的灰度总和为 1。

1.1.3 类库函数

COLOR.HPP 和 COLOR.CPP 定义了上一小节所述的三种彩色模型和它们之间的转化函数。

(1) rgb 三原色

rgb 三原色由结构变量定义：　　struct rgb

```
{
    unsigned char red;
    unsigned char grn;
    unsigned char blu;
}
```

由于VGA的硬件机制仅支持6位亮度取值，则red，grn，blu分量在使用时需要右移2位，在程序中使用C++所支持的：

```
*pal >>=2
```

代替了

```
pal->red >>=2;
pal->grn >>=2;
pal->blu >>=2;
```

(2) 彩色模型转化

从RGB模型转化为CMY模型由以下代码实现：

```
rgb R(100,100,100);
......
cmy C (R);
```

CMY模型转化为灰度模型YCbCr时，调用汇编级函数iscale()完成乘除运算，以加快运算速度，由于(grn * 587)/1000会使16位整数发生上溢，所以函数的输入值为16位整数，输出值变为32位整数。

```
// ------------------------------------------------------------ //
//                                                              //
//   File:     COLOR.HPP                                        //
//                                                              //
//   Desc:     Classes for implementation of various            //
//             color models and their conversion.               //
//                                                              //
// ------------------------------------------------------------ //

#ifndef COLOR_HPP
#define COLOR_HPP

struct rgb;
struct cmy;
struct cmyk;

//..............an RGB class

struct rgb
{
  unsigned char red;
```

```
unsigned char grn;
unsigned char blu;

rgb( )
{
    red = grn = blu = 0;
}
rgb( int r, int g, int b )
{
    red = r;
    grn = g;
    blu = b;
}
rgb( rgb& x )
{
    red = x.red;
    grn = x.grn;
    blu = x.blu;
}

rgb( cmy& x );
rgb( cmyk& x );
unsigned char graylevel( void );

rgb& operator = ( rgb& x )
{
    red = x.red;
    grn = x.grn;
    blu = x.blu;
    return *this;
}
rgb operator << ( int i )
{
    rgb x( red<<i, grn<<i, blu<<i );
    return x;
}
rgb& operator <<= ( int i )
{
    red <<= i;
    grn <<= i;
    blu <<= i;
    return *this;
}
```

```
    rgb operator >> ( int i )
    {
        rgb x( red>>i, grn>>i, blu>>i );
        return x;
    }
    rgb& operator >>= ( int i )
    {
        red >>= i;
        grn >>= i;
        blu >>= i;
        return *this;
    }
};

//..............a CMY class

struct cmy
{
    unsigned char cyn;
    unsigned char mag;
    unsigned char yel;

    cmy( )
    {
        cyn = mag = yel = 0;
    }
    cmy( int c, int m, int y )
    {
        cyn = c;
        mag = m;
        yel = y;
    }
    cmy( cmy& x )
    {
        cyn = x.cyn;
        mag = x.mag;
        yel = x.yel;
    }

    cmy( rgb& x );
    cmy( cmyk& x );
    unsigned char graylevel( void );
```

```
    cmy& operator = ( cmy& x )
    {
        cyn = x.cyn;
        mag = x.mag;
        yel = x.yel;
        return *this;
    }
    cmy operator << ( int i )
    {
        cmy x( cyn<<i, mag<<i, yel<<i );
        return x;
    }
    cmy& operator <<= ( int i )
    {
        cyn <<= i;
        mag <<= i;
        yel <<= i;
        return *this;
    }
    cmy operator >> ( int i )
    {
        cmy x( cyn>>i, mag>>i, yel>>i );
        return x;
    }
    cmy& operator >>= ( int i )
    {
        cyn >>= i;
        mag >>= i;
        yel >>= i;
        return *this;
    }
};

//..............a CMYK class

struct cmyk
{
    unsigned char cyn;
    unsigned char mag;
    unsigned char yel;
    unsigned char blk;

    cmyk( )
```

```
{
    cyn = mag = yel = blk = 0;
}
cmyk( int c, int y, int m, int k )
{
    cyn = c;
    mag = m;
    yel = y;
    blk = k;
}
cmyk( cmyk& x )
{
    cyn = x.cyn;
    mag = x.mag;
    yel = x.yel;
    blk = x.blk;
}

cmyk( rgb& x );
cmyk( cmy& x );
unsigned char graylevel( void );

cmyk& operator = ( cmyk& x )
{
    cyn = x.cyn;
    mag = x.mag;
    yel = x.yel;
    blk = x.blk;
    return *this;
}
cmyk operator << ( int i )
{
    cmyk x( cyn<<i, mag<<i, yel<<i, blk<<i );
    return x;
}
cmyk& operator <<= ( int i )
{
    cyn <<= i;
    mag <<= i;
    yel <<= i;
    blk <<= i;
    return *this;
}
```

```
    cmyk operator >> ( int i )
    {
        cmyk x( cyn>>i, mag>>i, yel>>i, blk>>i );
        return x;
    }
    cmyk&. operator >>= ( int i )
    {
        cyn >>= i;
        mag >>= i;
        yel >>= i;
        blk >>= i;
        return * this;
    }
};

//.....an RGB palette class

struct RgbPalette
{
    rgb * colors;
    int   ncolors;

    RgbPalette( )
    {
        colors = 0;
        ncolors = 0;
    }
    RgbPalette( rgb * clrs, int nclrs )
    {
        colors = clrs;
        ncolors = nclrs;
    }
   ~RgbPalette( )
    {
    }
};

#endif


// ------------------------------------------------------ //
//                                                        //
//    File:      COLOR.HPP                                //
```

```
//                                                                    //
//    Desc:    Classes for implementation of various                  //
//             color models and their conversion.                     //
//                                                                    //
// ------------------------------------------------------------------ //

#ifndef COLOR_HPP
#define COLOR_HPP

struct rgb;
struct cmy;
struct cmyk;

//..............an RGB class

struct rgb
{
   unsigned char red;
   unsigned char grn;
   unsigned char blu;

   rgb( )
   {
      red = grn = blu = 0;
   }
   rgb( int r, int g, int b )
   {
      red = r;
      grn = g;
      blu = b;
   }
   rgb( rgb& x )
   {
      red = x.red;
      grn = x.grn;
      blu = x.blu;
   }

   rgb( cmy& x );
   rgb( cmyk& x );
   unsigned char graylevel( void );

   rgb& operator = ( rgb& x )
```

```
    {
        red = x.red;
        grn = x.grn;
        blu = x.blu;
        return *this;
    }
    rgb operator << ( int i )
    {
        rgb x( red<<i, grn<<i, blu<<i );
        return x;
    }
    rgb& operator <<= ( int i )
    {
        red <<= i;
        grn <<= i;
        blu <<= i;
        return *this;
    }
    rgb operator >> ( int i )
    {
        rgb x( red>>i, grn>>i, blu>>i );
        return x;
    }
    rgb& operator >>= ( int i )
    {
        red >>= i;
        grn >>= i;
        blu >>= i;
        return *this;
    }
};

//..............a CMY class

struct cmy
{
    unsigned char cyn;
    unsigned char mag;
    unsigned char yel;

    cmy( )
    {
        cyn = mag = yel = 0;
```

```
}
cmy( int c, int m, int y )
{
    cyn = c;
    mag = m;
    yel = y;
}
cmy( cmy& x )
{
    cyn = x.cyn;
    mag = x.mag;
    yel = x.yel;
}

cmy( rgb& x );
cmy( cmyk& x );
unsigned char graylevel( void );

cmy& operator = ( cmy& x )
{
    cyn = x.cyn;
    mag = x.mag;
    yel = x.yel;
    return *this;
}
cmy operator << ( int i )
{
    cmy x( cyn<<i, mag<<i, yel<<i );
    return x;
}
cmy& operator <<= ( int i )
{
    cyn <<= i;
    mag <<= i;
    yel <<= i;
    return *this;
}
cmy operator >> ( int i )
{
    cmy x( cyn>>i, mag>>i, yel>>i );
    return x;
}
cmy& operator >>= ( int i )
```

```
    {
        cyn >>= i;
        mag >>= i;
        yel >>= i;
        return *this;
    }
};

//..............a CMYK class

struct cmyk
{
    unsigned char cyn;
    unsigned char mag;
    unsigned char yel;
    unsigned char blk;

    cmyk( )
    {
        cyn = mag = yel = blk = 0;
    }
    cmyk( int c, int y, int m, int k )
    {
        cyn = c;
        mag = m;
        yel = y;
        blk = k;
    }
    cmyk( cmyk& x )
    {
        cyn = x.cyn;
        mag = x.mag;
        yel = x.yel;
        blk = x.blk;
    }

    cmyk( rgb& x );
    cmyk( cmy& x );
    unsigned char graylevel( void );

    cmyk& operator = ( cmyk& x )
    {
        cyn = x.cyn;
```

```
        mag = x.mag;
        yel = x.yel;
        blk = x.blk;
        return *this;
    }
    cmyk operator << ( int i )
    {
        cmyk x( cyn<<i, mag<<i, yel<<i, blk<<i );
        return x;
    }
    cmyk& operator <<= ( int i )
    {
        cyn <<= i;
        mag <<= i;
        yel <<= i;
        blk <<= i;
        return *this;
    }
    cmyk operator >> ( int i )
    {
        cmyk x( cyn>>i, mag>>i, yel>>i, blk>>i );
        return x;
    }
    cmyk& operator >>= ( int i )
    {
        cyn >>= i;
        mag >>= i;
        yel >>= i;
        blk >>= i;
        return *this;
    }
};

//.....an RGB palette class

struct RgbPalette
{
    rgb *colors;
    int  ncolors;

    RgbPalette( )
    {
        colors = 0;
```

```
        ncolors = 0;
    }
    RgbPalette( rgb *clrs, int nclrs )
    {
        colors = clrs;
        ncolors = nclrs;
    }
   ~RgbPalette( )
    {
    }
};

#endif


; ---------------------------------------------------------------
;
;  File:   ISCALE.ASM
;
;  Desc:   Function to perform 16-bit integer scaling
;          using a scale define as the fraction n/d
;
; ---------------------------------------------------------------

.model large

; argument addressing -- 16-bit far calls

arg1   equ   [bp+6]
arg2   equ   [bp+8]
arg3   equ   [bp+10]

public iscale

.code

; ---------------------------------------------------------------
;  Desc:   Perform 16-bit integer scaling
;  Use:    int iscale( int i, int n, int d );
;  Ret:    The quantity  (i * n) / d
; ---------------------------------------------------------------

 iscale proc far
```

```
        push   bp
        mov    bp, sp
        push   bx

        mov    ax, arg1
        imul   word ptr arg2
        mov    bx, arg3
        idiv   bx
        shl    dx, 1
        inc    dx
        sub    bx, dx
        adc    ax, 0

        pop    bx
        pop    bp
        ret
_iscale endp

end
```

1.2 应用程序实例:颜色模拟

由于VGA的高分辨率模式只能在256色调色板中选择同时显示其中的16种颜色;如何选取这16种颜色通常称为颜色模拟,或者减色处理。

为了以数学方式严格定义模拟过程,我们把RGB模型定义为一个三维坐标系,其中红色为x轴,绿色为y轴,蓝色为z轴;这样一种特定的颜色具有坐标(r,g,b)。我们近一步定义两个颜色间的距离d_{12}由下式表示:

$$d_{12} = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

如果d_{12}的值只够小时,我们就可以把颜色(r_1, g_1, b_1)和颜色(r_1, g_1, b_2)相互模拟。图1-1给出了这个RGB三维坐标系的示意图。

选取16种颜色的通常方法上根据像素值排序,选择最常出现的前16种颜色组成调色板;这种方法一来耗时多,二来对于某些图像减色效果不够理想。

我们采用另一种方法,即通过计算像素的颜色距离值,取出256色调色板中均匀分布的16种颜色;对于每一种给定的颜色Ci(i从0到255变化),我们定义函数f(i):

$$f(i) = \sum_{c_j \neq c_i} \frac{1}{d^2}$$

f(i)实际上用于计算该颜色到相邻颜色的平均值,如果f(i)取值最小,就是我们所希望的颜色值。

在类库程序中,我们以f(i)值为基础对调色板颜色值排序。第一个颜色取值为RGB坐标原点(即纯黑色),然后按照下列顺序选择下一种颜色值,直到所有调色板表项填满为止。使用

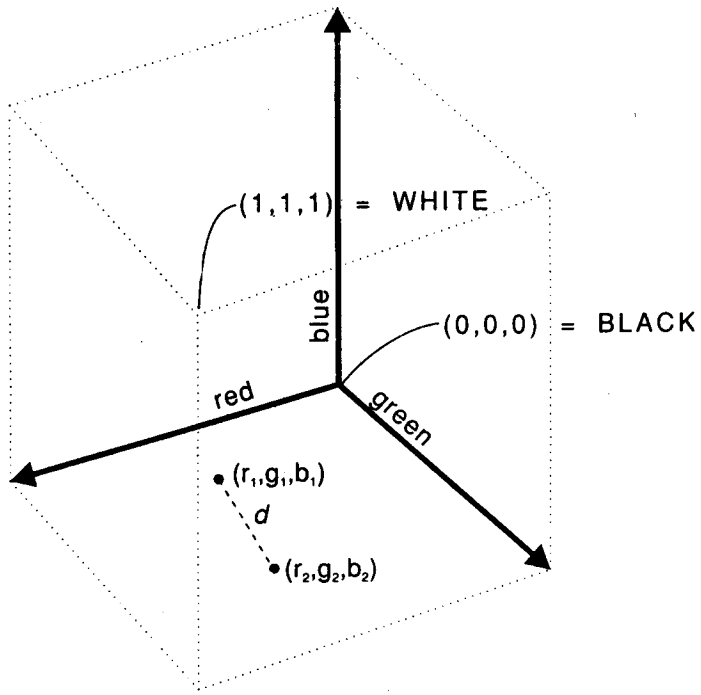

图 1－1　RGB 三维坐标系

RGB 三维坐标系，并用两颜色间的距离值度量调色板，选择未选中颜色中距离值最小者填入调色板表项。程序 1.4 和 1.5 是源程序代码，也许程序的注释更有助于理解这种颜色近似选取方法。这种方法的实质在于尽量均匀地在一张大调色板中抽取若干表项组成一张新的较小的调色板，尽量全面地反映原调色板的所有细节。在类库程序中，对于 256 色调色板选用一个颜色平均过滤函数(averaging filter)，对某一局部的颜色值进行近似平均，并用平均值代替这一范围内的原始颜色值。

在计算两颜色之间距离时使用汇编语言例程可以提高效率，对于 8 位 256 色调色板，需要进行 32 位整数运算；而且由于距离仅用于比较最小间距值，所以使用距离平方而不是其平方根可以在很大程度上提高效率。

```
//-------------------------------------------------------------//
//                                                             //
//   File:    COLORMAP.HPP                                     //
//                                                             //
//   Desc:    Palette modification and color mapping           //
//            functions.                                       //
//                                                             //
//-------------------------------------------------------------//
```