# 研究学习文档

**大作业选题：MFC 图像处理编程题***
**选题人数：1**
**选题者：王阳**
**学号：2017013580**
**邮箱：wangyang3434@gmail.com**
**微信号：wy1198486241**
**手机号码：18810756772**

1> 预备知识

1.MFC 编程(单文档分割窗口，使用 Ribbon 功能区)

使用 Ribbon 功能区进行编程与以往的经典菜单存在不同：

很多功能控件在不添加 *Update_Command_UI(...)*函数时会调用默认函数从而不能正常显示，且如果按钮点击事件的响应函数 *OnCommand(...)*设置在子视图类中，将不视为添加了响应函数而仍为 disable 状态。因此很多消息响应考虑存放在 Doc 类中。

Ribbon 功能区的各种控件封装在以 CMFCRibbon 开头的类中，均为从原有对话框控件类继承而来，使用方式不尽相同。详见微软开发者文档。同时，这些控件也可以添加在状态栏、对话框等其他位置中，可以获取后通过成员函数修改其状态。

另外，Ribbon 功能区有自己单独的设计器，可以在资源视图打开 ribbon 菜单后，在工具箱面板中查看。根据范围从大到小，分为类别、面板、控件。在每个类别通过属性面板指定使用的图标集之后，所有的控件可以设置图片集中的图标作为标识。控件可以右击添加响应函数。

静态分割窗口[1]：(整理自[数字图像处理]系列博文，见参考文献一)

单文档程序的客户区可以通过 CSplitterWnd 类进行分割，操作为在窗口框架类 CMainFrame 的 *OnCreateClient(...)*函数中调用 CSplitterWnd 成员函数 *CreateStatic(...)*后再调用 *CreateView(...)*，具体参数见相关文档。当然，也可以在建立工程的导航过程中选择分割窗口，默认左右分割为两个视图。

MFC 各窗口类及 Doc 类互相调用：

在 MFC 工程中，doc 类一般用于存储数据，视图类一般用于显示及交互，因此 doc 类与视图类的信息交换十分重要。两边互相调用的方式为：

从 doc 类中调用窗口类：利用 GetNextView(…)函数，返回值为视图类的指针

```
POSITION pos = GetFirstViewPosition();    //这里是获取视图类在内存中的位置
GetNextView(pos);   //这里获取了第一个子视图位置
CLKIView *view_tmp = (CLKIView*)GetNextView(pos);   //这里获取第二个子视图位置，并将视图类指针保存起来
```

从各个窗口类调用 doc 类：`CLKIDoc* pDoc = GetDocument();  //直接获取 doc 类指针`

2.图像基础知识

各种格式的图像在计算机中储存都会分为文件头、信息头、调色板以及具体像素内容四个部分，文件头中包含文件大小、文件类型等信息，信息头中包含了图片的宽高、分辨率、

像素所占字节数等重要信息，需要时都可以直接读取。

在之前的版本中，曾经尝试过多种图像读取方式，比如利用 *LoadImage(...)*函数加载位图文件，利用 *CBitmap* 装载位图，绘制时使用当前视图 DC 的 pDC 指针的 *StretchBlt(...)*函数；或是直接构建自己的图像类，使用最底层的数据指针直接操作。多种方式各有利弊，在尝试了各种方式后，选择了一种较为简洁且兼容性好的方式，利用基于 GDI+的 *CImage* 类进行各项操作。

应注意的是，在 bmp 格式的文件中，像素数据是倒储存的，即从下到上，从左到右，因此自己重写底层函数时注意坐标的转换。另外，图像宽度(以像素个数为返回值)并不一定等于位图宽度(以字节为返回值)，因为 Windows 规定一个扫描行所占的字节数必须是 4 的倍数（即以 long 为单位），不足的以 0 填充。直接使用指针操作需注意。

*CImage* 类成员函数[2]：(整理自毛星云博文，华丽的 CImage 类，见参考文献二)

**<1>功能为创建与连接，释放的函数**

*Attach*--附加一个 HBITMAP 到 CImage 对象，位图类型 DIB 与否都可以
*Create*--创建一个 DIB 部分的位图，并将其附加到之前创建的 CImage 对象
*CreateEX*--创建一个 DIB 部分的位图（拥有额外的参数），并将其附加到之前　　创建的 CImage 对象
*Destroy*--从 CImage 类上分离该位图并进行删除
*Detach*--从 CImage 类里分离该位图
*ReleaseDC*--释放设备描述表中的数据
*ReleaseGDIPlus*--释放 GDI+使用的源

**<2>功能为输入与输出的函数**

*GetExporterFilterString*--返回系统支持的输入文件格式类型及其描述
*GetImporterFilterString*--返回系统支持的输出文件格式类型及其描述
*LoadFromResource*--从指定的源处加载一个图像资源
*Load*--从指定文件处加载一个图像资源
*IsIndexed*--判断一个位图颜色映射到了一个索引调色盘
*IsNull*--判断一个源位图是否被当前载入
*Save*--以指定的类型来保存图像

**<3>关于位图类型与参数的函数**

*GetBits*--返回一个指向该位图实际像素值指针
*GetBPP*--返回该位图每个像素的位
*GetColorTable*--返回颜色表中 RGB 值的范围条目
*GetDC*--返回目前被选择的设备描述表
*GetExporterFilterString*--返回系统支持的输入文件格式类型及其描述
*GetImporterFilterString*--返回系统支持的输出文件格式类型及其描述
*GetHeight*--返回当前图像的像素高度
*GetMaxColorTableEntries*--返回颜色表条目中的最大值
*GetPitch*--返回当前图片的间距（单位为字节），用来决定像素格式的
*GetTransparentColor*--返回颜色表中透明色的位置
*GetWidth*--返回当前图片的宽度（单位为像素）

**<4>功能为图形绘制与位图块传输相关的函数**

*AlphaBlend*--显示一个半透明或者透明像素的位图
*BitBlt*--从源设备描述表复制一个位图文件到当前设备描述表
*Draw*--从源矩形复制一个位图到目的矩形，该函数伸缩或者拉伸位图来适应目标矩形的尺寸，如果有必要，会处理 Alpha 值和透明颜色。
*MaskBlt*--用指定的掩码和光栅操作来结合颜色数据和目的位图
*PlgBlt*--执行一个从源设备描述表的矩形到目标设备描述表的平行　　四边形的块状位图转换
*SetColorTabel*--在 DIB 的颜色表中设定一系列条目的 RGB 颜色的值
*SetPixelIndexed*--设置在指定坐标处的像素（使用索调色板的索引值）。
*SetPixelRGB*--设置在指定坐标处的像素（使用 RGB 值）
*SetPixel*--在指定坐标处设置像素的颜色

*SetTransparentColor*--设置将被视为透明色的颜色的索引值（只能选取调色板中的一种颜色）

*StretchBlt*--从源矩形复制一个位图到目的矩形，如果有必要，该函数会伸缩或者拉伸位图来适应目标矩形的尺寸

*TransparentBlt*--从源设备描述表中复制一个带有透明色的位图到当前设备描述表

*CImage* 类注意事项:

在使用时首先要调用判断函数 *IsNull()*，之后 *Destroy()*之前加载的图像；

注意格式的匹配，尤其是 *create(...)*时位深度的匹配；

类中带有获取单点像素 RGB 值的 *GetPixel(...)*函数以及设置单点像素 RGB 值的 *SetPixel(...)*函数，但不适合大量调用时，因为会由于频繁的栈空间操作使得效率较慢；

*Draw(...)*函数有多种重载，涵盖了其他绘制函数的所有功能；

每个图像拥有自己的 DC，因而可以通过在现有图像的 DC 上绘制实现快速覆盖而不必使用 *SetPixel(...)* 与 *GetPixel(...)*，但是需注意的是 *GetDC()*后需要及时 *releaseDC()*，否则 *destroy()*加载的图像时会抛出异常；

## 2> 操作说明及实现

### 1.设置及读取颜色值:

由于 *setpixel(...)*处理速度慢，获取及填充像素值的部分均采用地址操作以加速处理。但由于 RGB 在内存中是倒储存的，因此设置某个位置颜色值的语句如下：

```
*(position+ 2) = GetRValue(Color);
*(position + 1) = GetGValue(Color);
*(position) = GetBValue(Color);
```

### 2.自适应显示及缩放:

坐标比例变换公式为：

$$(X,Y,Z)=(x,y,z)\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & Sz \end{bmatrix}$$

为了简便，原图像与坐标系各维护一个变量用于记录缩放比例，且锁定宽高比。使用多重重载的类成员 *draw(...)*函数可以进行图像缩放，开发过程中就采用了这种方式。通过 *draw(...)*函数提供的缩放功能，并将当前客户区大小与图像大小对比获得缩放比例，即可实现简单的自适应显示。本工程中的自适应是锁定宽高比的，且要求图像全部内容能在客户区内显示出来。图像自适应部分代码如下：

```
if (ImageClient.Height() / (float)ImageClient.Width() < image_show.GetHeight() /
(float)image_show.GetWidth()) {
    ZoomSize = ImageClient.Height() / (float)image_show.GetHeight();
}
else {
    ZoomSize = ImageClient.Width() / (float)image_show.GetWidth();
}
```

对于底层，真正的缩放实现一般采用插值算法取附近像素的平均值插入。

### 3.平移:

平移坐标变换公式如下：

$$(X,Y,1)=(x,y,1)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix}$$

仍是为简便与运算速度考虑，直接通过位移量修改，不通过矩阵运算。

平移量的获取：通过鼠标左键抬起按下的事件获得，同时考虑到平滑性，平移量大小与缩放比例有关。

4.坐标系旋转：

在三维图像变换与显示时，涉及到多次矩阵变换：

坐标变换：两组标准正交基间的过渡，矩阵为正交矩阵，且涉及到矩阵求逆

视角变换：在所维护的三维系外部套一个球，在球上设置一个摄像机用于观察，涉及到球坐标变换

**投影变换：通过观测摄像机将三维系投影在投影面上，这也是本工程中实现旋转使用的方式。**

投影变换有很多方式[3]，包括透视投影以及平行投影，透视投影符合实际生活，而平行投影则是投影中心到投影面距离无限时的特殊情况。

由于底层操作(包括绘制坐标系、视角变换等)复杂性以及工程量原因，工程仅给出了绕竖直轴实现的旋转。且为简化操作，将投影面固定为 RG 平面。采用斜二测投影的方式实现了对旋转的支持。其公式如下：设转角为 a，则有，

$$(X,Y,Z,1)=(x,y,z,1)\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ \cos(a)*1/2 & \cos(a)*1/2 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$$

坐标变换通过投影坐标变换函数实现，仍是为了简便与速度没有使用矩阵：

```
void G_3DView::Transform3Dto2D(float &x, float &y, float z)
{
    x = x - (z * cos(slant)) / 2;     //slant为转角
    y = y - (z * sin(slant)) / 2;
}
```

5.坐标系数据的预储存与读取：

为了加速从坐标图选取原图像数据的过程，起初准备通过 STL 库中的 set 嵌套 vector 来实现，但是进过测试，预处理操作要经过 10s 以上，并未起到理想的加速效果，考虑到 set 对于查询操作实现太慢，因此考虑维护有序的树形结构以加速查询。

四叉树与八叉树分别在平面二维碰撞检测以及三维包围集的产生中有着重要应用。考虑到四叉树与八叉树的查询操作均达到 log(n)级别(实际上更快)，故决定采用四叉树维护从绘制平面上一点到对应颜色值集合的映射，利用八叉树维护从颜色值到原图像对应像素所在位置集合的映射(具体实现在 *Octree* 以及 *Quadtree* 类中，不在此赘述，请自行查看)。通过两者的配合，实现了相同颜色点去重、选择集快速设置等功能。经测试，速度较快，性能较好。

且由于每个树节点成员变量点集的维护仍然采用的是 vector，速度仍有进一步提升空间。

6.选择颜色值：

对于按钮事件进行响应，使用 MFC 封装好的弹出对话框类 *CColorDialog* 进行实现，文件的打开以及关闭类似，通过 *CFileDialog* 类实现。此部分代码如下：

```
CColorDialog dlg;
dlg.m_cc.Flags |= CC_RGBINIT | CC_FULLOPEN;
if (IDOK == dlg.DoModal())
{
    C1 = dlg.m_cc.rgbResult;      //将dlg.m_cc.rgbResult获取到的颜色对话框中的颜色保存到变量C1中
}
```

7.选择集相关操作:

为逻辑简便起见，相关操作(选择集显示、填充等)均使用遍历进行。通过 switch 开关语句判断不同情况并进行操作，还有可封装的空间。代码冗长，不在此贴出。

8.[扩展功能]滤镜：灰度图像显示

利用 24 位深度的 RGB 图像模拟 8 位灰度图像的简单方法即将 R、G、B 均设为三者平均值即可。但其实根据光的亮度特性，其实正确的灰度公式应当是加权平均值算法：R=G=B=R*0.299+G*0.587+B*0.144，不过为了运算速度，简单平均值也可以接受。

真正的 24 位真彩图与 8 位的灰度图的区别就在于，真彩图文件中没有调色板，灰度图有调色板，真彩图中的像素矩阵是 RGB 值，灰度图中的像素矩阵是调色板索引值。源代码只简单的改变像素矩阵的 RGB 值，来达到彩色图转为灰度图，并没有添加调色板。因此说只是简单模拟灰度图像显示。

9.[扩展功能]滤镜：冷暖色调

暖色调：通过增加绿色及红色通道的值来模拟增加黄色通道的值，从而实现色调偏暖。

冷色调：通过增加蓝色通道的值，从而实现色调偏冷。

10.[扩展功能]滤镜：亮度调节

通过同时增加三个通道的数值使图片颜色更接近白色，从而提高图片亮度；反之变暗。

## 3> 总结

通过本次大作业的开发学习，提高了工程开发与组织能力，学习了一些图像处理的基本方法与操作，加深了对 MFC 编程的理解，受益颇多。

## 4> 参考文献、图书及博文


[1] [数字图像处理]系列博文 一.MFC 详解显示 BMP 格式图片 二.MFC 单文档分割窗口显示图片
https://blog.csdn.net/eastmount/article/details/18238863?spm=a2c4e.11153940.blogcont26096.3.74bb1fb952V6sv

[2]【Visual C++】游戏开发笔记十四 游戏画面绘图(四) 华丽的 CImage 类
https://blog.csdn.net/poem_qianmo/article/details/7422922

[3] 和青芳. 计算机图形学原理及算法教程[M]. 清华大学出版社, 2006.

[4] 任哲. MFC Windows 应用程序设计[M]. 清华大学出版社, 2004.