華中科技大学

课 程 实 验 报 告

课程名称： 数字图像处理实验

实验时段： 2024 年 03 月 - 2024 年 05 月

指导教师： 金良海

专业班级： 计算机 2109

学　　号： U202111641

姓　　名： 张宸瑞

实验报告成绩：

指导教师签字：

日期：

计算机科学与技术学院

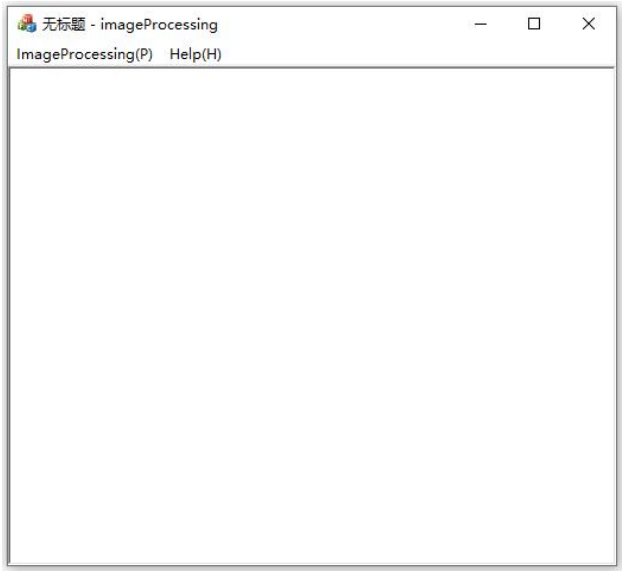
目 录

数 字 图 像 处 理 实 验 报 告

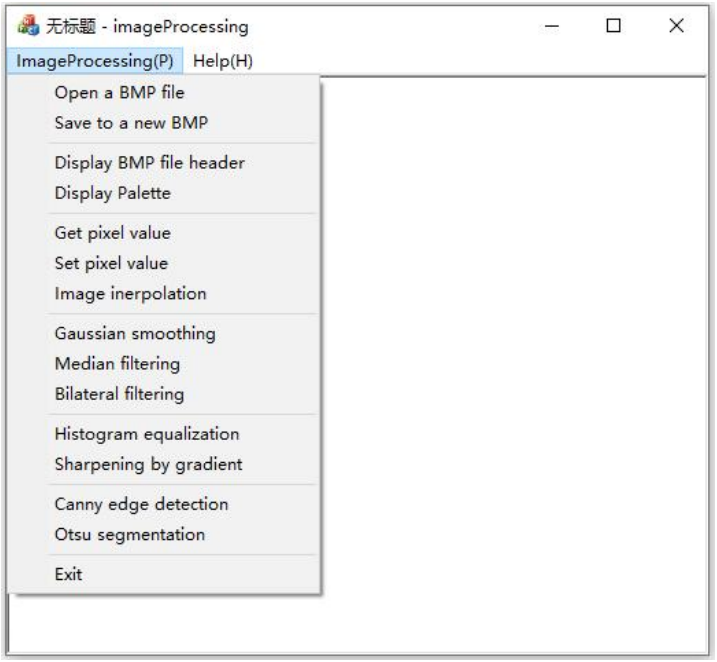
1 实验目的与要求

在 Windows 下用 VC++（VS20XX，不要使用任何第三方库）或 QT 编程实现一个简单的图像处理系统。该图像处理系统的主界面及需要实现的图像处理任务如下所示：

（1）主界面



（2）图像处理任务



2 实验内容

用 VC++或 QT 编程实现 ImageProcessing 菜单下的所有功能，要求不能使用任何第三方图像处理库。需要实现的功能如下所示：

(1) Open a BMP file

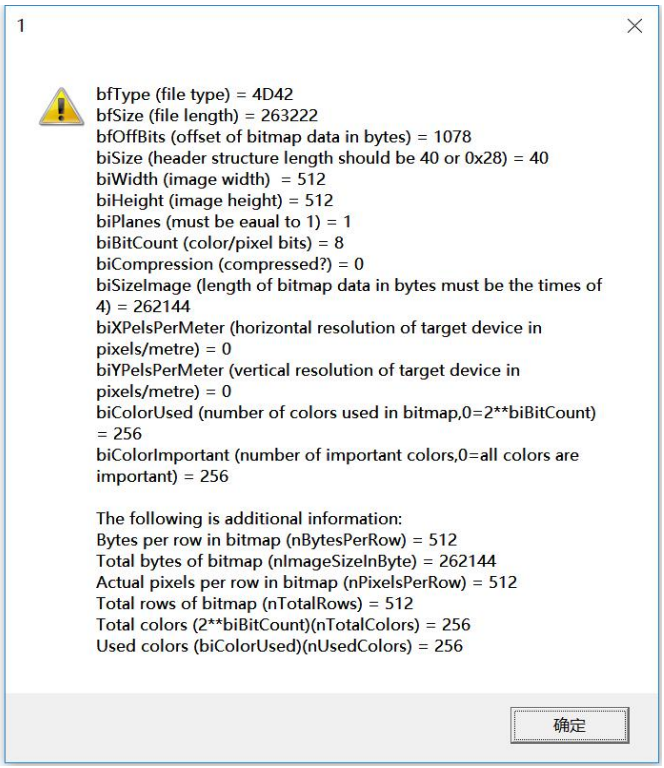
打开一个 BMP 文件，并在窗口中显示出来。

(2) Save to a new BMP file

将当前视图保存为一个新的 BMP 文件(先弹出一个对话框，输入一个 BMP 文件名)。

(3) Display file header

按如下的格式显示文件头信息：



(4) Display Palette

显示调色板信息（如果 BMP 文件包含调色板）。

(5) Get pixel value

取某个位置像素的颜色值，并显示出来（所需的参数从对话框中获取）。

(6) Set pixel value

设置某个位置像素的颜色值，并显示出来（所需的参数从对话框中获取）。

(7) Image interpolation

图像缩放：x 和 y 方向的缩放因子、插值算法选择（最邻近和双线性），从对话框中获取。需要将图像缩放的结果显示出来。

(8) Gaussian smoothing

从对话框中获取高斯函数的均方差，对图像做高斯平滑，并将结果显示出来。

(9) Median filtering

实现中值滤波（滤波器的大小从对话框中获取），并将结果显示出来。

(10) Bilateral filtering

实现双边滤波，参数 sigma_d 和 sigma_R 从对话框中获取，并将结果显示出来。

(11) Histogram equalization

直方图均衡化，并将结果显示出来。

(12) Sharpening by gradient

实现基于梯度的图像锐化，所需参数从对话框中获取，将锐化结果显示出来。

(13) Canny edge detection

实现 Canny 算子边缘检测，所需参数从对话框中获取，并将结果显示出来。

(14) Otsu segmentation

实现 Otsu 阈值分割，并显示分割结果，同时将阈值用对话框（API 函数 AfxMessageBox()）显示出来。

(15) Haze removal

实现图像去雾，所需参数从对话框中获取，并将去雾结果显示出来。

3 实验过程

3.1 开发环境

（1）操作系统：Windows 11 家庭中文版

（2）版本号：21H2 22000.2538

（3）处理器：Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz

（4）内存：16GB

（5）实验平台：Visual Studio Community 2022 17.7.6

3.2 系统设计

首先观察整体代码框架，找到功能实现的切入点。注意到 imageProcessingView.h 文件中的类 CimageProcessingView 中声明了 14 个消息映射函数，这些消息映射函数与我们将要实现的 14 个功能一一对应。因此接下来的实验目标就是修改完善这些消息映射函数在 imageProcessingView.cpp 中的定义（也有可能涉及其它代码文件的修改），使得用户能够使用该图像处理系统中的所有功能。

（1）Open a BMP file

功能：打开一个 BMP 文件并在窗口中显示出来。

实现过程：

该功能在给出的代码框架中已经实现，因此不再阐述其实现。

（2）Save to a new BMP file

功能：根据用户在对话框中输入的 BMP 文件名将当前视图保存为新的 BMP 文件。

实现过程：

首先我尝试了用这个功能保存当前视图，发现当前视图会被命名为@1.bmp 并被保存至 D 盘中。接着我阅读了对应的消息映射函数 OnImageprocessSavetofile。该函数使用的变量与函数的意义如下所示：

1. pFileBuf：代表当前视图的指针；
2. strBmpFile：代表视图将要被保存至的位置的绝对路径；
3. SaveDIB：表示将 pFileBuf 指向的视图保存至 strBmpFile 所描述的位置。

为了将当前视图保存至我们期望的位置，在保存视图时该图像处理系统必须跳

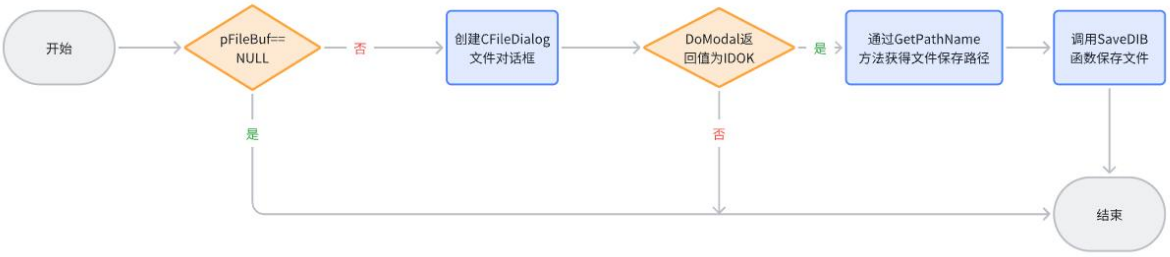
出对话框让用户选择保存的位置以及新文件的名称。注意到在打开 BMP 文件时系统跳出了对话框，因此 OnImageprocessOpenbmpfile 函数中必然调用了与对话框相关的接口。结合网上相关资料，相关类的使用方法如下所示：

1. CFileDialog 类对象的定义。定义 CFileDialog 类的对象时唯一需要注意的是初始化传参时需要将第一个参数（bOpenFileDialog）置为 FALSE，这表示该对话框不是打开文件的对话框，而是保存文件的对话框；

2. CFileDialog 类的 DoModal 方法。该方法只返回两种状态：IDOK 与 IDCANCEL，其中 IDCANCEL 表示关闭对话框。我们需要据此判定是否继续执行文件的保存。

3. CFileDialog 类的 GetPathName 方法。该方法会返回文件保存位置的绝对路径，可以直接赋值给 strBmpFile 并传给 SaveDIB 进行文件保存。

了解了上述内容后实现就可以实现 BMP 文件的保存了，函数流程图如下：



（3）Display file header

功能：按照规定的格式显示 BMP 文件头的信息。

实现过程：

该功能在给出的代码框架中已经实现，因此不再阐述其实现。

（4）Display Palette

功能：若 BMP 文件包含调色板，则显示调色板信息。

实现过程：

首先阅读 OnImageprocessDisplaypalette 函数，发现该函数已经通过调用 GetDIBPaletteData 函数将当前 BMP 文件的调色板信息读入，并返回了 RGBQUAD 类型的指针 palette，同时将调色板中元素的个数记录在了变量 num 中。若 palette 为 NULL 则说明该 BMP 文件没有调色板，否则需要根据 palette 与 num 将调色板信息输出。

需要注意的是这里使用 AfxMessageBox 输出调色板信息并不是一个好的选择，

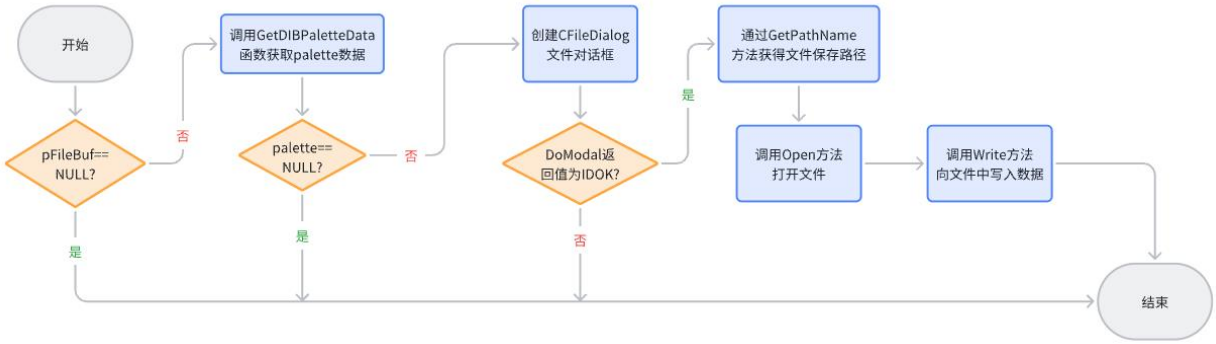
因为有些 BMP 文件的调色板信息可能很大，这会导致在栈上开辟的缓冲区过大以及显示的困难，因此这里选择将调色板信息以文件的形式进行保存。具体来说，我们需要用到以下方法：

1. CFile 类的 Open 方法。该方法用于打开文件，主要接受两个参数 FileName 与 OpenFlags。其中 FileName 代表文件的绝对路径，可以通过 CFileDialog 的 GetPathName 方法获取；OpenFlags 代表文件的打开方式，这里我们要创建一个新文件并写入调色板信息，因此 OpenFlags 应设置为 modeCreate 与 modeWrite。

2. CFile 类的 Write 方法。该方法用于向文件写入，接受两个参数 buff 与 Count。其中 buff 代表将要写入的内容，Count 表示写入内容的长度。

3. sprintf 函数。该方法用于向缓冲区写入。

首先我们调用 Open 方法打开文件，然后定义缓冲区。在每次循环中，利用指针 palette 与当前循环下标 i 获取当前颜色映射关系并将其写入缓冲区，并调用 Write 方法将缓冲区的信息写入文件，函数流程图如下：



（5）Get pixel value

功能：获取对话框中指定位置的像素颜色值并显示出来。

实现过程：

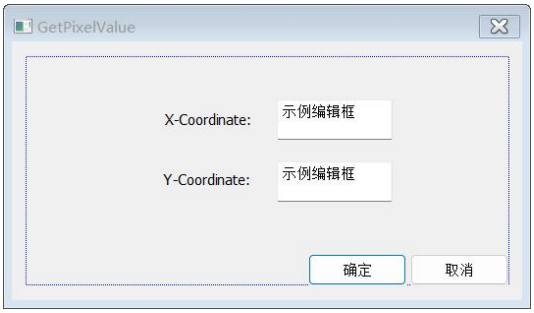
首先阅读 OnImageprocessGetpixelvalue 函数，注意到该函数已经通过调用 GetPixel 函数获取了(x,y)位置上像素的颜色值，因此我们只需要关注如何弹出可以让用户键入(x,y)坐标的对话框即可。

查阅了 MFC 对话框的相关资料后，我将利用 MFC 构建接受用户输入的模态对话框的过程总结如下：

1. 利用 Visual Studio 的类向导添加 MFC 类（继承自 CDialogEx）；

2. 在完成了上一步后，资源视图中的 Dialog 文件夹下会新增一个 Dialog。我们利用工具箱为该 Dialog 添加需要的控件。例如，该任务需要我们允许用户输入

坐标(x,y)，因此我们需要添加两个 Edit Control 控件与两个 Static Text 控件，结果如下图所示：

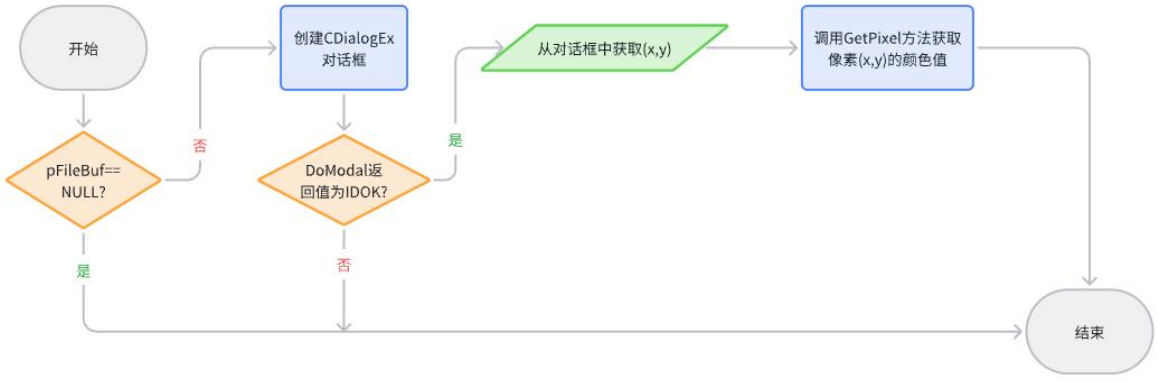


3. 完成对话框的设计后，我们需要将控件与变量绑定。对于 Static Text 类型的控件，我们不需要为其定义变量；对于 Edit Control 类型的控件，我们需要为其定义两个变量，一个变量的类别为“控件”（CEdit 类型），另一个变量的类型为“值”（CString 类型）；

4. 最后用 Visual Studio 的类向导让新建 MFC 类继承基类的 DoModal 虚函数，该方法可以帮助我们判定是否需要执行对话框中的内容。

完成对话框类的构建后，我们就可以在进入 Getpixelvalue 响应函数时弹出对话框了。接着检验对话框 DoModal 方法的返回值，如果为 IDOK 表示用户点击了确认，此时我们就需要从对话框类的 CString 类型的变量中获取用户的输入，从而做进一步的处理。

函数流程图如下：



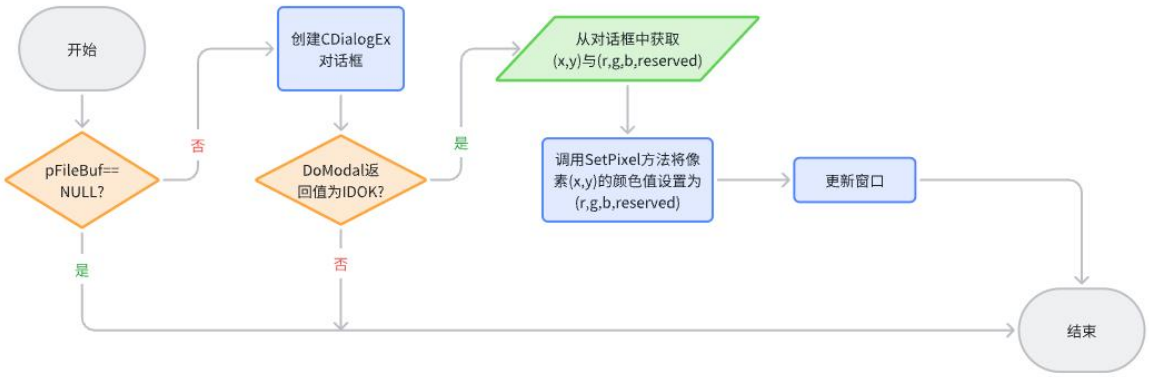
（6）Set pixel value

功能：从对话框中获取位置信息与颜色信息，并将对应位置上的像素颜色值置为从对话框中获取的颜色信息。将修改后的图片显示出来。

实现过程：

响应函数 OnImageprocessSetpixelvalue 功能的实现与（5）**Get pixel value** 中几乎一致。该函数需要完善的部分同样也是构建对话框接受用户输入，采用（5）中列出的四个步骤即可完成该任务。

函数流程图如下：



（7）Image interpolation

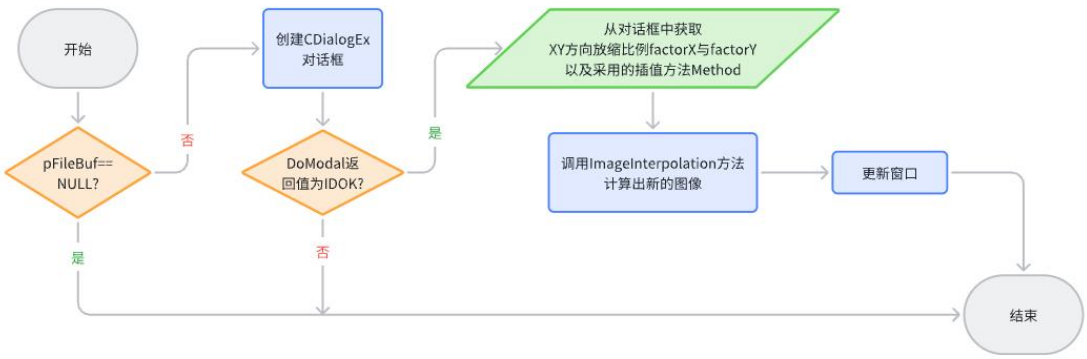
功能：从对话框中获取图像在 XY 方向上的缩放比例以及插值算法类型，并将放缩后的图片显示出来。

实现过程：

阅读函数 OnImageprocessInerpolation，注意到插值函数 ImageInterpolation 已经在_GlobalCommon.cpp 下实现，因此该函数需要完善的部分也仅仅是从用户输入中得到插值方法（最邻近插值或双线性插值）与放缩比例。由于这一部分已经在（5）**Get pixel value** 与（6）**Set pixel value** 中提及，后面就不再赘述。

需要注意的地方在于由于我们读入的是放缩因子，而 ImageInterpolation 函数接受的是图像的新长宽，因此需要先从 BMP 文件信息头中读取图像的长宽，乘上放缩因子后再传递给 ImageInterpolation 函数。另外，由于放缩因子可能不是整数，因此在将 CString 转成数字时应使用 atof 而不是 atoi。

函数流程图如下：



（8）Gaussian Smoothing

功能：根据从对话框中获取的高斯函数均方差对图像做高斯平滑，并将结果输出。

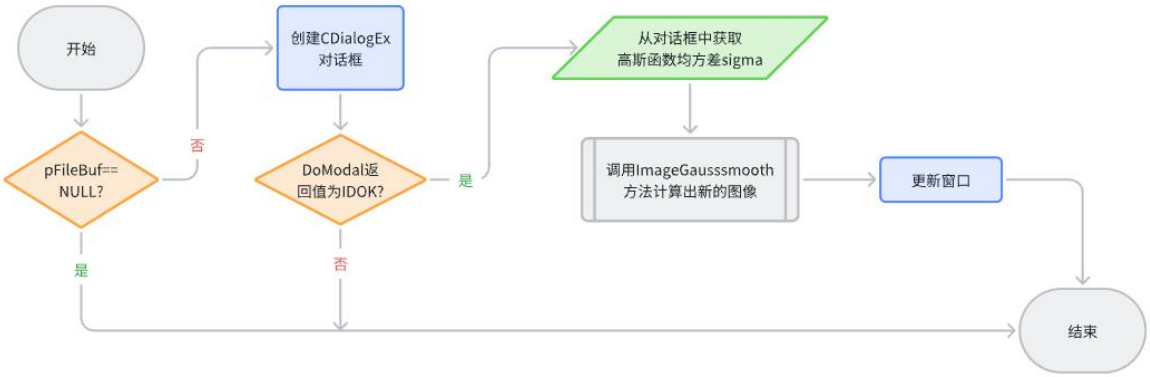
实现过程：

高斯平滑分为高斯核的计算以及图像平滑两步。下面将具体阐述每个部分的实现过程：

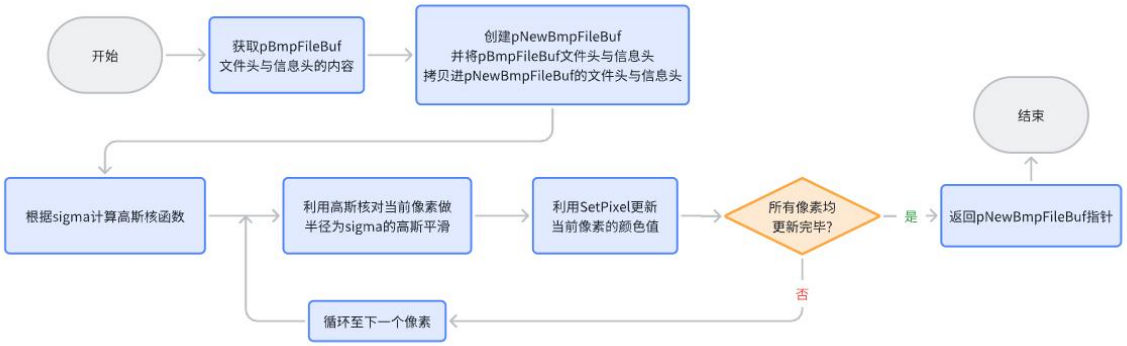
1. 高斯核的计算。高斯核的计算可进一步分为计算与归一化。计算部分首先要得到高斯核的大小，高斯核的大小为 R=sigma*2+1，然后就是利用高斯分布的公式 1/(2*PI*sigma^2)*exp(-(i^2+j^2)/(2*sigma^2))进行计算，注意由于之后会对计算结果进行归一化，而该公式的 1/(2*PI*sigma^2)是所有项的公因子，因此在计算阶段可以忽略。归一化部分就是将高斯核的每一项除以所有项的和，这样做是为了避免对图像进行高斯平滑后对图像整体的亮度造成影响。

2. 图像平滑。对每个像素计算其平滑后的颜色值时，我们需要得到该像素周围像素的颜色值，并将高斯核对应位置上的值作为权重对这些颜色值做加权和，并将最终得到的值作为该像素平滑后的颜色值。图像平滑时有一个细节，即对于边界像素的周围像素的定义方式。我在实现时是将周围像素的选取约束在图像内，即若周围像素的坐标超出图像，则将其坐标修改为距离该越界点最近的图像像素坐标。

OnImageprocessGausssmooth 函数的流程图如下：



ImageGausssmooth 函数的流程图如下：



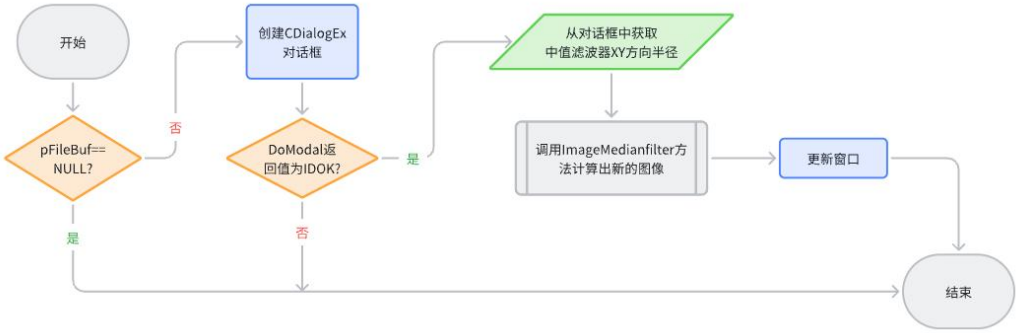
（9）Median filtering

功能：根据从对话框中获取的滤波器大小对图像做中值滤波，并将结果输出。

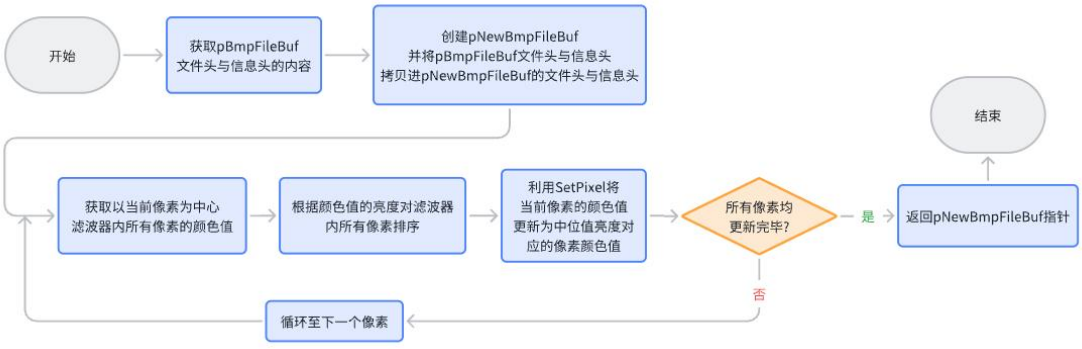
实现过程：

中值滤波在消除冲击噪声方面有着显著效果，其基本实现思路是对于一个待滤波的像素，根据滤波器的宽（2*N1+1）与高（2*N2+1）遍历其周围的所有像素。对于滤波器内的每个像素，利用公式(r*299+g*587+b*114)/1000 计算该像素的亮度，再根据计算出的亮度对滤波器内所有像素进行排序，取中间的像素的颜色值作为滤波的结果。由于冲击噪声的颜色值一般显著高于或低于其周围像素的颜色值，因此在利用中值滤波消除冲击噪声时一般能取得显著效果。

OnImageprocessMedianfilter 函数的流程图如下：



ImageMedianfilter 函数的流程图如下：



（10）Bilateral filtering

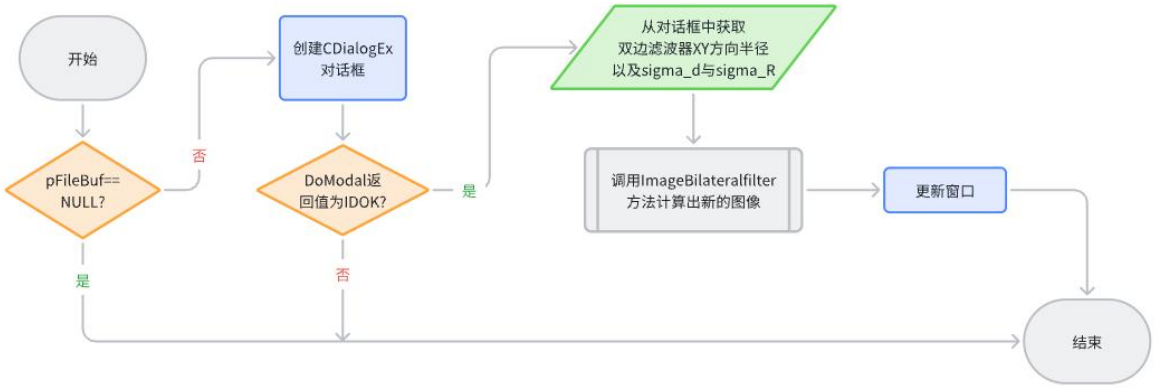
功能：根据从对话框中获取的参数（sigma_d 与 sigma_R）对图像做双边滤波，并将结果显示出来。

实现过程：

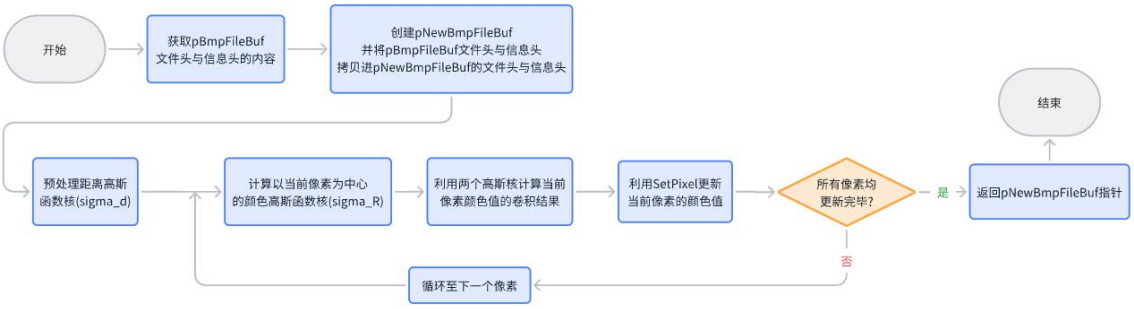
双边滤波能较为有效地消除加性噪声（如高斯噪声）并在一定程度上保护边缘、纹理等细节信息。双边滤波的滤波器与高斯平滑的滤波器类似，区别在于双边滤波除了需要计算关于距离的高斯函数，还需要计算关于颜色的高斯函数。

具体来说，关于距离的高斯函数 Kernel_d 可以在滤波前预处理出来；而关于颜色的高斯函数 Kernel_R 则需要对每个特定位置的滤波器分别计算，计算方法是将每个像素的颜色值视为三维空间的坐标向量，在此基础上计算滤波器中心像素与其周围某个像素的欧氏距离，再将结果代入高斯函数。滤波时需要将滤波器内每个像素的颜色值乘以对应位置的权重（Kernel_d*Kernel_R）并累加，并将最终结果归一化（即除以 Kernel_d 与 Kernel_R 对应位置的乘积和）

OnImageprocessBilateralfilter 函数的流程图如下：



ImageBilateralfilter 函数的流程图如下：



（11）Histogram equalization

功能：对图像做直方图均衡化，并将结果显示出来。

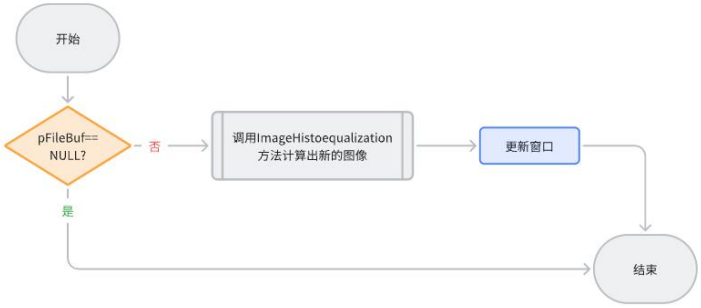
实现过程：

直方图均衡化是一种让图像变清晰的方法。直方图均衡化的过程可分为三步：

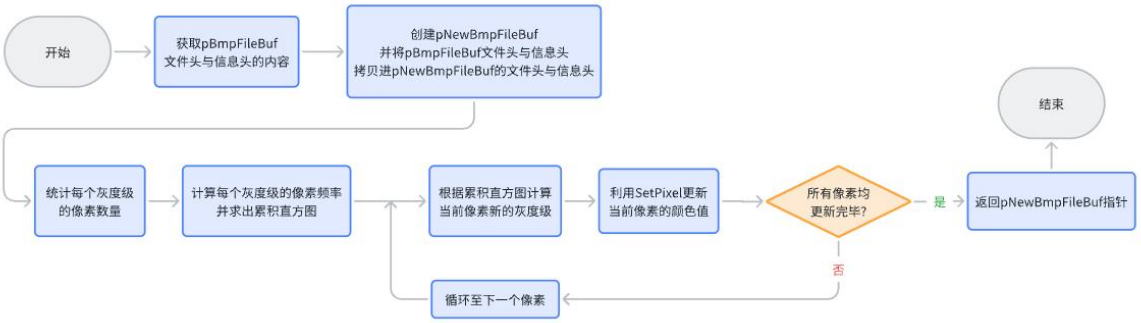
1. 统计图像中每个灰度级的像素个数；
2. 计算每个灰度级的像素频率并得出累积直方图；
3. 根据累计直方图对原图像中的每个像素做灰度变换。

其中第 1 步中的灰度级仍采用(r*299+g*587+b*114)/1000 公式进行计算（对于灰度图像中的像素，该公式计算出来的结果就是该像素的灰度）；第 3 步中的灰度变换公式为 int((L-1)Pj+0.5)，其中 L 为图像的灰度级（默认为 256），Pj 为该像素对应灰度级在累积直方图中的频率。

OnImageprocessHistoequalization 函数的流程图如下：



ImageHistoequalization 函数的流程图如下：



（12）Sharpening by gradient

功能：根据从对话框中获取的参数（基本信息保留因子 k1，细节增强强度系数 k2）实现基于梯度的图像锐化，并显示锐化结果。

实现过程：

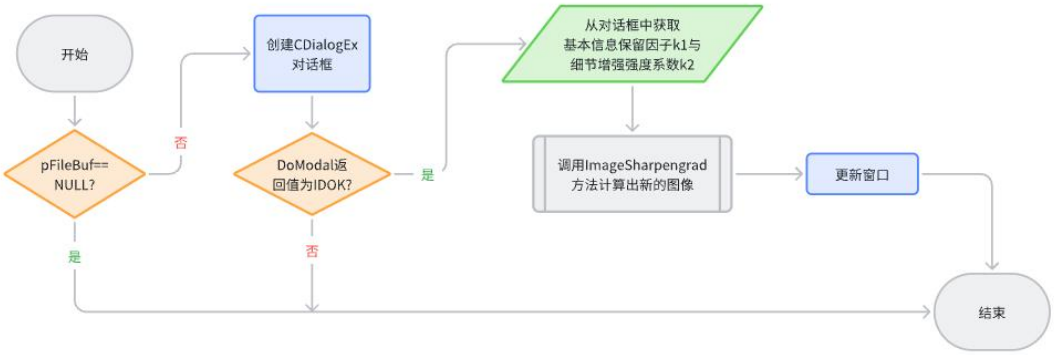
基于梯度的图像锐化可以刻画图像变换显著的区域。图像锐化的过程一般分为两步：

1. 计算当前像素处的梯度模。X 方向梯度 gradX 的计算式是 f(x+1,y)-f(x,y)，

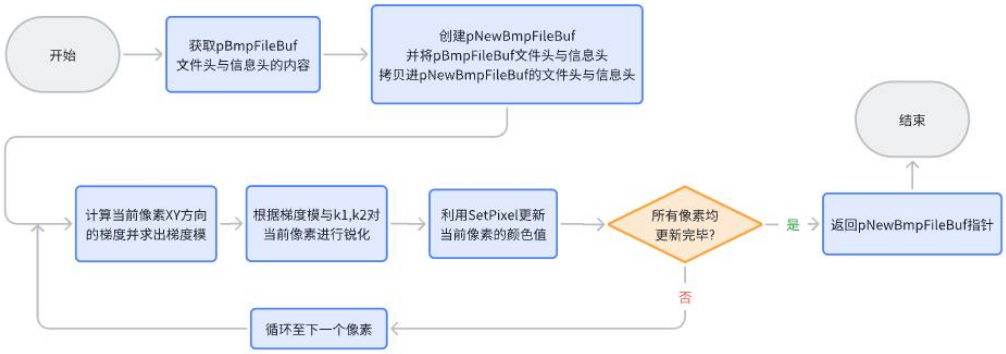
Y 方向梯度 gradY 的计算式是 f(x,y+1)-f(x,y)，这里也可以采用 Sobel 算子、Roberts 算子或 Prewitt 算子计算梯度，但本质上大同小异。计算得到梯度后，就可以利用 grad=sqrt(gradX^2+gradY^2)计算梯度模了；

2. 利用梯度模和给定的系数对图像进行增强。图像增强的计算式为 g(x,y)= k1*f(x,y)+k2*grad，其中 k1 为基本信息保留因子，k2 为细节增强强度系数。

OnImageprocessSharpengrad 函数的流程图如下：



ImageSharpengrad 函数的流程图如下：



（13）Canny edge detection

功能：根据从对话框中获取的参数实现 Canny 算子边缘检测，并将结果显示出来。

实现过程：

Canny 算子边缘检测可用于检测图像中物体的边缘。Canny 算子边缘检测主要分为以下几个步骤：高斯平滑、计算梯度、非极大值抑制、双阈值检测以及边缘链接。在实现过程中，我将双阈值检测和边缘链接放在一起进行。

1. 高斯平滑。由于在边缘检测中，我们不可避免地要去计算梯度，于是图像中的噪声会对边缘检测造成很大的影响，这对 Canny 算子边缘检测也不例外。因此在计算梯度之前首先要对图像做一次高斯平滑，尽可能地消除一些噪声。

2. 计算梯度。计算梯度包括计算梯度模以及梯度角。

实现中采用 Sobel 算子计算 X 方向上的梯度 gradX 以及 Y 方向上的梯度 gradY，接着通过 grad=sqrt(gradX^ 2+gradY^2)计算梯度模。

关于梯度角 theta 的计算，首先需要注意 gradX 接近零的情况，这时需要根据 gradY 的正负情况判断梯度角指向 Y 的正向还是负向。接着计算 theta'=arctan(gradY /gradX)，然后枚举 8 个方向的角度（0,pi/4,pi/2,3pi/4,pi,5pi/4,3pi/2,7pi/4）与 theta' 作差（注意对于某个方向的角度 angle，我们需要计算 fabs(angle-theta')、fabs(angle-theta'+2*pi)、fabs(angle-theta'-2*pi)三种情况并取其中的最小值作为结果），将 theta' 近似为与其差值最小的角，得到 theta。

3. 非极大值抑制。得到每个像素的梯度模后，我们比较该像素梯度模与其梯度角方向（正向和负向）两个相邻像素的梯度模的大小，如果该像素梯度模更大，表示该处是一个极大值（边缘），因此保留该点的梯度模作为其灰度值；否则抑制该点，将其灰度值置为 0。

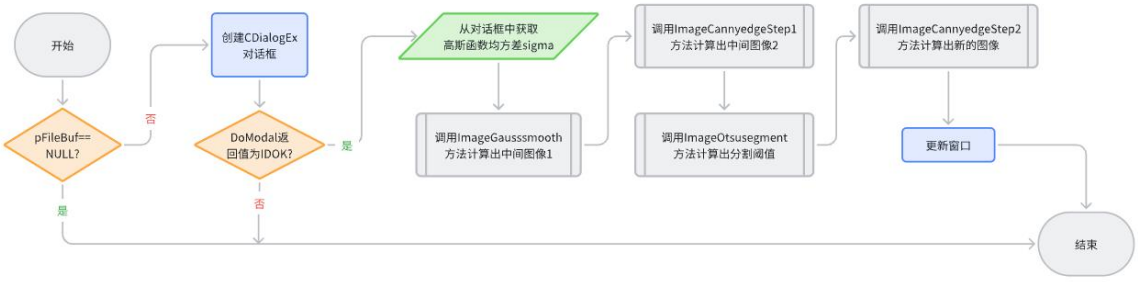
4. 双阈值检测与边缘链接。经非极大值抑制处理过的图像中仍存在一些不明显、不重要的边缘，因此需要利用双阈值检测进一步突出重要边缘，并尽可能保证边缘的连续性。

这里阈值的选取是一个问题，我在实现时是在经非极大值抑制处理过的图像上利用 Otsu 阈值分割得到阈值 threshold，并将其设定为强阈值 maxth，再取强阈值的一半设为弱阈值 minth。

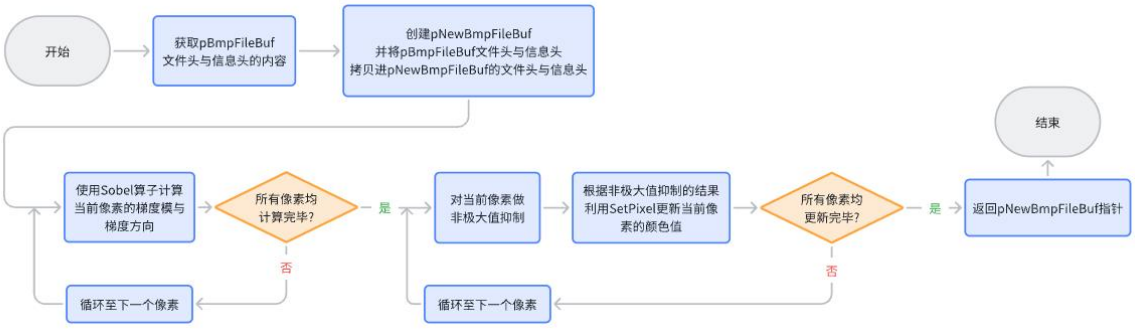
接着我定义了一个队列，将所有梯度模大于强阈值 maxth 的像素放入队列中，并以这些像素作为起始点对整张图像做 BFS。对于遍历过程中遇到的每一个像素，若其梯度模大于弱阈值 minth，则将其加入队列，否则忽视该像素点。

遍历完成后，对于每个像素点，若其在双阈值检测和边缘链接中被遍历过，则在最终图像中保留其灰度值，否则将其灰度值置为 0。

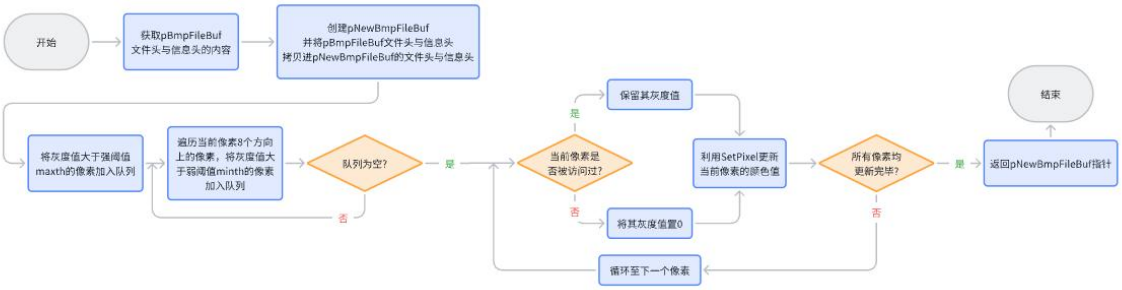
OnImageprocessCannyedge 函数的流程图如下：



ImageCannyedgeStep1 函数的流程图如下：



ImageCannyedgeStep2 函数的流程图如下:



（14）Otsu segmentation

功能：实现 Otsu 阈值分割，显示分割结果，并将阈值输出。

实现过程：

Otsu 阈值分割可以计算出一个阈值并根据该阈值将图像分割成两部分。因此，Otsu 阈值分割可以分为阈值计算与图像分割两个步骤：

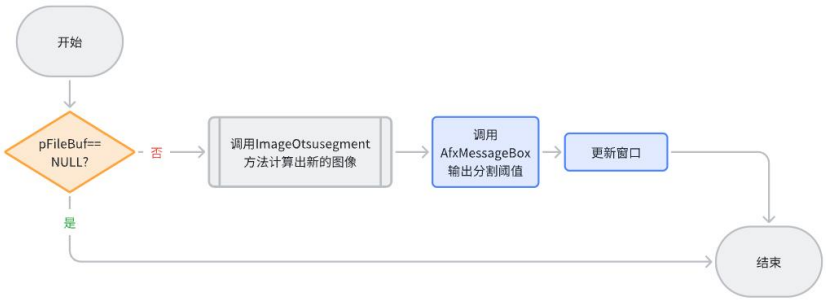
1. 阈值计算。Otsu 阈值分割希望计算得到这样一个阈值 threshold，使得根据 threshold 分割得到的两部分图像灰度值的类间方差最大。对此我们可以考虑枚举所有可能的阈值（从 0 到 255）并依次计算该阈值下的类间方差。

假设我们现在已经计算得出了两个类的类内均值μ1，μ2 以及整体占比 w1，w2（w1+w2=1），那么类间方差σ^2=w1*(μ1-μ)^2+w2*(μ2-μ)^2=w1*w2*(μ2-μ1)^2，其中μ为整体方差。整体占比 w1，w2 可以通过计算图像的累积直方图预处理，类内均值μ1，μ2 也可以通过计算图像的累积均值预处理（即μ(k)代表所有灰度级在 0-k 之间的像素的灰度级均值）；

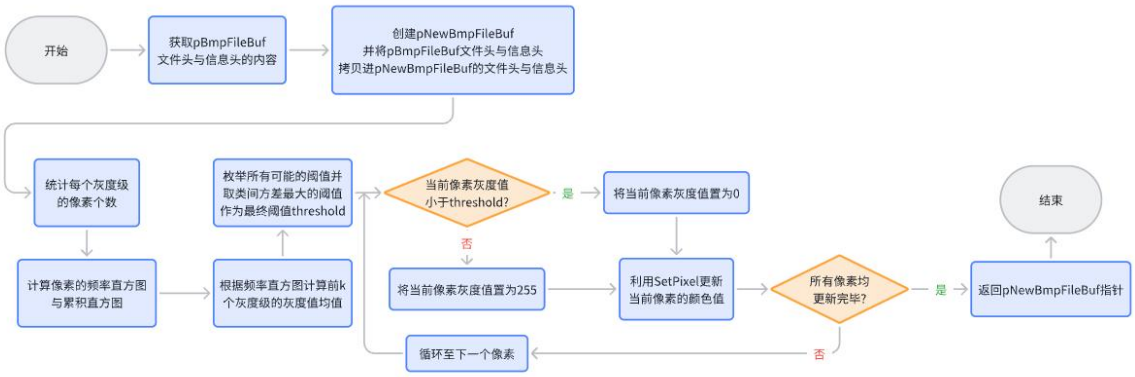
2. 图像分割。得到了最佳阈值后，我们就可以枚举原图像中的每个像素，根据该像素的灰度级与阈值的大小关系决定将该像素的灰度值置为 0 还是 255。

在图像处理系统显示 Otsu 阈值分割结果之前，利用 AfxMessageBox 显示计算得到的最佳分割阈值。

OnImageprocessOtsusegment 函数的流程图如下:



ImageOtsusegment 函数的流程图如下：



（15）Haze removal

功能：根据从对话框中获取的参数对图像进行去雾，并显示去雾结果

实现过程：

图像去雾功能可以去除图像中的雾，从而改善图像品质。实现时采用的算法是基于暗通道先验的去雾算法，该算法有两个重要的理论依据：

1. 雾图像成像模型。该模型给出了物体本来颜色与有雾条件下物体颜色之间的关系，具体公式为 I(x)=J(x)t(x)+A(1-t(x))。其中 I(x)表示有雾条件下 x 处的颜色，J(x)表示无雾条件下 x 处的颜色，t(x)表示 x 方向的透射率，A 表示大气光强度。在有雾条件下，物体自身的颜色会被大气削弱，只有 t(x)比例的颜色值会到达观察窗口，剩下 1-t(x)比例均为大气光的颜色值。

2. 暗通道先验。暗通道先验提出了这样一个假设：对于一张没有雾气的彩色图像，在绝大多数像素所在的局部区域内，总存在部分像素他们至少有一个通道的值会很小。

借助以上两个理论依据，我们便可以着手对图像进行去雾。图像去雾的步骤分为计算暗通道图像、求解大气光矢量、计算每个位置的透射率、求解无雾图像。

1. 计算暗通道图像。对于每个像素我们都要计算其暗通道，方法是遍历以该像素为中心，半径为 R（实现时选择的是 3）范围内的所有像素，取所有这些像素

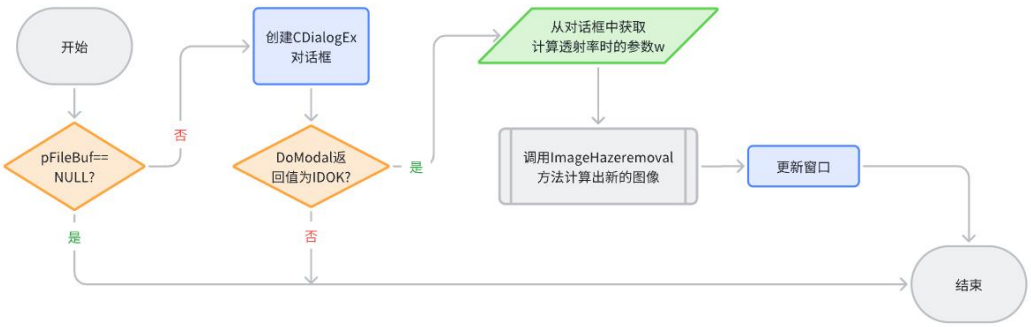
3 个通道的最小值作为当前像素的暗通道值。

2. 求解大气光矢量。借助暗通道先验的假设，如果存在一些像素他们所在的局部区域内所有像素每个通道的值都比较大，说明这些像素所在区域雾气较重，他们的颜色值大部分都由大气光构成，因此可以选取这些像素的颜色值作为大气光矢量。在实现时，我们选择暗通道图像中亮度最大的 0.1%个像素，并在原图像相应位置查找亮度最大的像素，将该像素的颜色值作为大气光矢量 A。

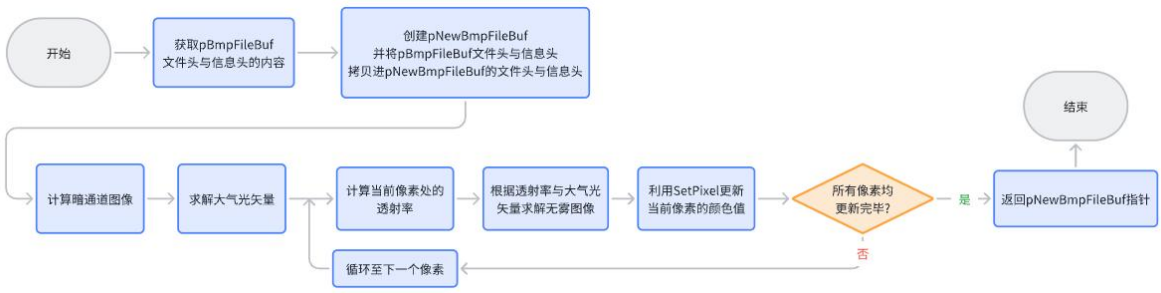
3. 计算每个位置的透射率。对雾图像成像模型进行变形，可以得到 Ic(x)/Ac=(Jc(x)/Ac)t(x)+1-t(x)。基于暗通道先验的假设，在 x 位置的邻域内，极大概率会存在一个位置 y，使得 y 处像素有至少一个通道中的值近似为 0，即存在 y∈Ω(x)，c'∈{r,g,b}，Jc'(y)=0。因此上式变为 Ic'(y)/Ac' =1-t(x)，那么 t(x)=1-Ic'(y)/Ac'。据此计算式，我们只需计算 x 的邻域内 Ic(x)/Ac 的最小值即可。这里可以乘上参数 w 对 t(x)做一定的调整，w 的取值一般为 0.8~0.95。

4. 求解无雾图像。计算出 t(x)值后，就可以直接利用雾图像成像模型求解无雾图像了，变形后的公式为 J(x)=(I(x)-A)/t(x)+A。在求解时，需要注意分母 t(x)不能为零，为此可以将分母改写为 max(t(x),t0)，其中 t0 为设置的值，用于保证分母不为零。

OnImageprocessHazeremoval 函数的流程图如下：



ImageHazeremoval 函数的流程图如下：



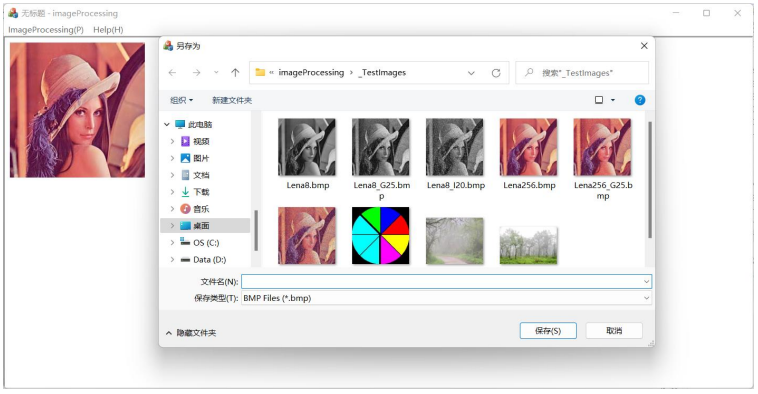
3.3 系统测试

(1) Open a BMP file

该功能在给出的代码框架中已经实现，这里不再展示测试过程。

(2) Save to a new BMP file

选择 Save to a new BMP file 功能后，会弹出以下对话框，给出文件名后当前视图就会被保存至选定的文件夹下。



(3) Display file header

该功能在给出的代码框架中已经实现，这里不再展示测试过程。

(4) Display Palette

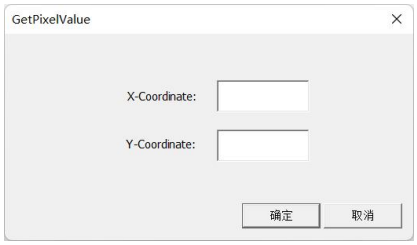
若当前图像文件没有调色板，则选用该功能后会跳出如下窗口：

否则会弹出保存文件的对话框，点击保存后调色板信息会以 txt 格式被保存至对应文件夹下。打开调色板信息文件，其内容显示如下：

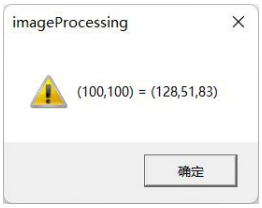
```
Palette[0]:rgbBlue=0,rgbGreen=0,rgbRed=0,rgbReserved=0
Palette[1]:rgbBlue=0,rgbGreen=ff,rgbRed=0,rgbReserved=0
Palette[2]:rgbBlue=ff,rgbGreen=0,rgbRed=0,rgbReserved=0
Palette[3]:rgbBlue=0,rgbGreen=0,rgbRed=ff,rgbReserved=0
Palette[4]:rgbBlue=ff,rgbGreen=ff,rgbRed=0,rgbReserved=0
Palette[5]:rgbBlue=0,rgbGreen=ff,rgbRed=ff,rgbReserved=0
Palette[6]:rgbBlue=ff,rgbGreen=0,rgbRed=ff,rgbReserved=0
```

(5) Get pixel value

选择该功能后会弹出对话框用于输入需要获取颜色值的像素坐标，如下所示：



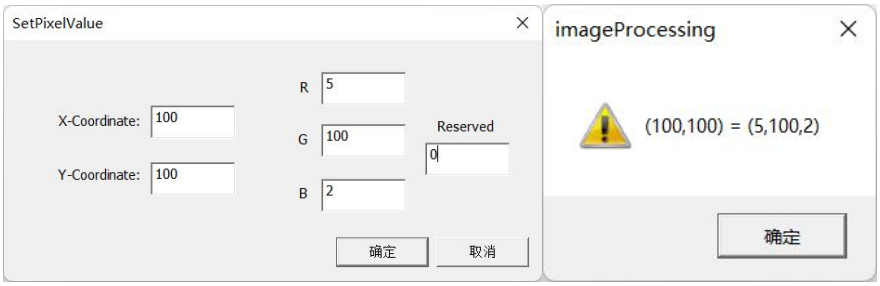
填入坐标后点击确定则会以弹窗形式告知用户该像素的颜色值:



(6) Set pixel value

选择该功能后会弹出对话框用于输入需要设置颜色值的像素坐标与将要设置成的颜色值，如下所示:

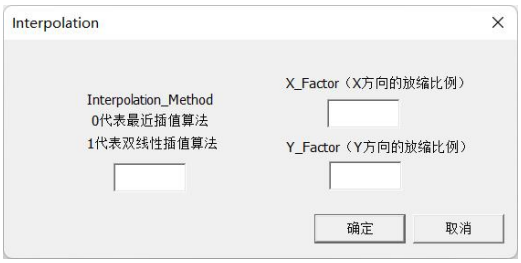
设置完毕后可用 GetPixelValue 功能查看该像素是否已经被修改，如下所示:



可以看到(100,100)处的颜色值已经被修改。

(7) Image interpolation

选择该功能后会弹出对话框用于输入插值算法的选择以及 XY 方向的放缩比例，如下所示:



由于插值算法已经在给出的框架中实现，这里就不再展示插值的效果。

(8) Gaussian smoothing

选择该功能后会弹出对话框用于输入高斯平滑采用的均方差，如下所示:

确定均方差后图像处理系统会对原图像进行平滑，当均方差为 3 时效果对比如下:

(9) Median filtering

选择该功能后会弹出对话框用于输入中值滤波器在 XY 方向的半径:

确定滤波器大小后图像处理系统会对原图像做中值滤波,当 XY 方向的半径均为 2，滤波对象为带有冲击噪声的图像时，效果对比如下:

(10) Bilateral filtering

选择该功能后会弹出对话框用于输入双边滤波器在 XY 方向的半径以及距离高斯函数的均方差 sigma_d、颜色高斯函数的均方差 sigma_R:

确定上述参数后图像处理系统会对原图像做双边滤波，当 XY 方向半径均为 3，sigma_d=3，sigma_R=80，滤波对象为带有高斯噪声的图像时，效果对比如下：

(11) Histogram equalization

选择该功能后图像处理系统会直接对当前视图做直方图均衡化，可以发现均衡化后的图片清晰度显著提升：

(12) Sharpening by gradient

选择该功能后会弹出对话框用于输入基于梯度的锐化时基本信息保留因子 k1 以及细节增强强度系数 k2：

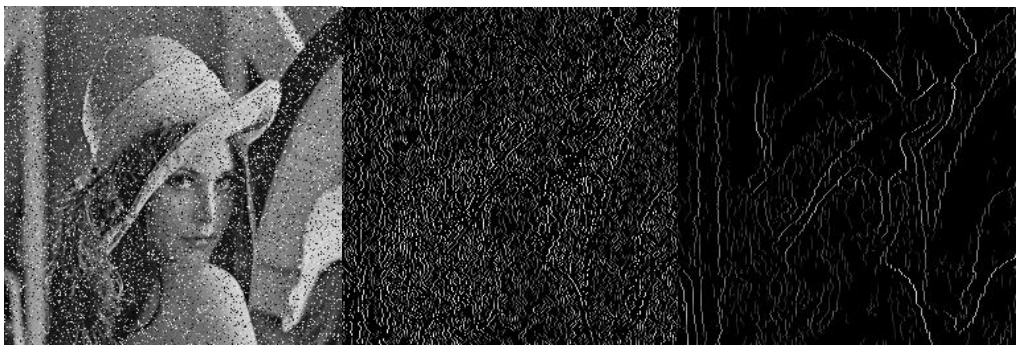
确定上述参数后图像处理系统会对原图像做基于梯度的锐化，当 k1=0.8，k2=3 时效果对比如下：

(13) Canny edge detection

选择该功能后会弹出对话框用于输入 Canny 算子边缘检测中高斯函数的均方差:

确定高斯函数均方差后图像处理系统会对原图像做 Canny 算子边缘检测，当均方差为 1 时，效果对比如下:

当原图像中带有噪声时，需要更大的高斯函数均方差以取得更好的边缘检测效果。例如，当原图像中带有冲击噪声时，均方差为 1 与均方差为 3 时的效果对比如下:



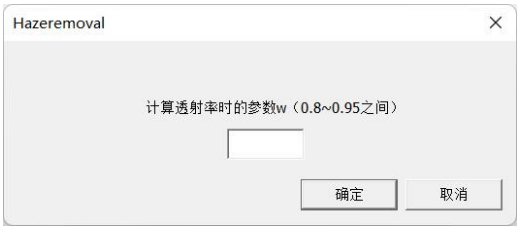
显然当均方差为 3 时取得了更好的边缘检测效果。

(14) Otsu segmentation

选择该功能后图像处理系统会直接对原图像做 Otsu 阈值分割，并在输出分割阈值后显示经阈值分割后的图像：

(15) Haze removal

选择该功能后会弹出对话框用于输入图像去雾算法中的 w 参数：



确定了上述参数后图像处理系统会对原图像做图像去雾操作，当 w=0.95 时，去雾效果对比图如下：

3.4 源程序

全部源程序可在 github 链接 ForwardFuture/imageProcessing (github.com)中找到

（1）_GlobalCommon.h

```
//GlobalCommon.h
#include "afxwin.h"

char       *OpenBMPfile(CString strBmpFile);

BITMAPFILEHEADER *GetDIBHEADER(char *pFileBuffer);
BITMAPINFOHEADER *GetDIBINFO(char *pFileBuffer);
char       *GetDIBImageData(char *pFileBuffer);
RGBQUAD *GetDIBPaletteData(char *pFileBuffer,int nEntryNumber[1]);
int        GetImageWidth(char *pFileBuffer);
int        GetImageHeight(char *pFileBuffer);
int        GetColorBits(char *pFileBuffer);
long       GetUsedColors(char *pFileBuffer);
long       GetWidthBytes(char *pFileBuffer);

void       DisplayHeaderMessage(char *pBmpFileBuf);
void  DisplayImage(CDC *pDC,char *pBmpFileBuf,
              int disp_xL=0,int disp_yL=0,int disp_Width=0,int disp_Height=0,int mode=0 );

long       GetPixel(char *pFileBuffer,int x,int y,RGBQUAD rgb[1],bool bGray[1]=NULL);
void       SetPixel(char *pFileBuffer,int x,int y,RGBQUAD rgb);

BOOL       SaveDIB(char *pFileBuffer,CString strBmpFile);

char       *ImageInterpolation(char *pBmpFileBuf,int newWidth,int newHeight,int nMethod=0);
char       *ImageGausssmooth(char* pBmpFileBuf, int sigma);
char       *ImageMedianfilter(char* pBmpFileBuf, int N1, int N2);
char       *ImageBilateralfilter(char* pBmpFileBuf, int N1, int N2, int sigma_d, int sigma_R);
char       *ImageHistoequalization(char* pBmpFileBuf);
char       *ImageSharpengrad(char* pBmpFileBuf, double k1, double k2);
char       *ImageCannyedgeStep1(char* pBmpFileBuf);
char       *ImageCannyedgeStep2(char* pBmpFileBuf, int maxth, int minth);
char       *ImageOtsusegment(char* pBmpFileBuf, int& threshold);
```

```
char       *ImageHazeremoval(char* pBmpFileBuf, double w);
```

（2）CBilateralfilter.h

```
#pragma once
#include "afxdialogex.h"


// CBilateralfilter 对话框

class CBilateralfilter : public CDialogEx
{
	DECLARE_DYNAMIC(CBilateralfilter)

public:
	CBilateralfilter(CWnd* pParent = nullptr);   // 标准构造函数
	virtual ~CBilateralfilter();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CBilateralfilter };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit N1_Edit;
	CEdit N2_Edit;
	CEdit sigma_d_Edit;
	CEdit sigma_R_Edit;
	CString N1;
	CString N2;
	CString sigma_d;
	CString sigma_R;
	virtual INT_PTR DoModal();
};
```

（3）CCannyedge.h

```
#pragma once
#include "afxdialogex.h"


// CCannyedge 对话框

class CCannyedge : public CDialogEx
{
	DECLARE_DYNAMIC(CCannyedge)

public:
	CCannyedge(CWnd* pParent = nullptr);   // 标准构造函数
	virtual ~CCannyedge();
```

```
// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CCannyedge };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit sigma_Edit;
	CString sigma;
	virtual INT_PTR DoModal();
};
```

（4）CGausssmooth.h

```
#pragma once
#include "afxdialogex.h"


// CGausssmooth 对话框

class CGausssmooth : public CDialogEx
{
	DECLARE_DYNAMIC(CGausssmooth)

public:
	CGausssmooth(CWnd* pParent = nullptr);	// 标准构造函数
	virtual ~CGausssmooth();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CGausssmooth };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit Gauss_Edit;
	CString Gauss;
};
```

（5）CHazeremoval.h

```
#pragma once
#include "afxdialogex.h"


// CHazeremoval 对话框
```

```
class CHazeremoval : public CDialogEx
{
	DECLARE_DYNAMIC(CHazeremoval)

public:
	CHazeremoval(CWnd* pParent = nullptr);	// 标准构造函数
	virtual ~CHazeremoval();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CHazeremoval };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit w_Edit;
	CString w;
	virtual INT_PTR DoModal();
};
```

（6）CInputXY.h

```
#pragma once
#include "afxdialogex.h"


// CInputXY 对话框

class CInputXY : public CDialogEx
{
	DECLARE_DYNAMIC(CInputXY)

public:
	CInputXY(CWnd* pParent = nullptr);	// 标准构造函数
	virtual ~CInputXY();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CInputXY };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit X;
```

```
	CEdit Y;
	CString X_Coord;
	CString Y_Coord;
	virtual INT_PTR DoModal();
};
```

（7）CInputXYRGB.h

```
#pragma once
#include "afxdialogex.h"


// CInputXYRGB 对话框

class CInputXYRGB : public CDialogEx
{
	DECLARE_DYNAMIC(CInputXYRGB)

public:
	CInputXYRGB(CWnd* pParent = nullptr);	// 标准构造函数
	virtual ~CInputXYRGB();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CInputXYRGB };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit X;
	CEdit Y;
	CEdit R_Edit;
	CEdit G_Edit;
	CEdit B_Edit;
	CEdit Reserved_Edit;
	CString X_Coord;
	CString Y_Coord;
	CString R;
	CString G;
	CString B;
	CString Reserved;
	virtual INT_PTR DoModal();
};
```

（8）CInterpolation.h

```
#pragma once
#include "afxdialogex.h"


// CInterpolation 对话框
```

```
class CInterpolation : public CDialogEx
{
	DECLARE_DYNAMIC(CInterpolation)

public:
	CInterpolation(CWnd* pParent = nullptr);	// 标准构造函数
	virtual ~CInterpolation();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CInterpolation };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit Method_Edit;
	CEdit factorX_Edit;
	CEdit factorY_Edit;
	CString Method;
	CString factorX;
	CString factorY;
	virtual INT_PTR DoModal();
};
```

（9）CMedianfilter.h

```
#pragma once
#include "afxdialogex.h"


// CMedianfilter 对话框

class CMedianfilter : public CDialogEx
{
	DECLARE_DYNAMIC(CMedianfilter)

public:
	CMedianfilter(CWnd* pParent = nullptr);	// 标准构造函数
	virtual ~CMedianfilter();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CMedianfilter };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持
```

```
	DECLARE_MESSAGE_MAP()
public:
	CEdit N1_Edit;
	CEdit N2_Edit;
	CString N1;
	CString N2;
	virtual INT_PTR DoModal();
};
```

（10）CSharpengrad.h

```
#pragma once
#include "afxdialogex.h"


// CSharpengrad 对话框

class CSharpengrad : public CDialogEx
{
	DECLARE_DYNAMIC(CSharpengrad)

public:
	CSharpengrad(CWnd* pParent = nullptr);	// 标准构造函数
	virtual ~CSharpengrad();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_CSharpengrad };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

	DECLARE_MESSAGE_MAP()
public:
	CEdit k1_Edit;
	CEdit k2_Edit;
	CString k1;
	CString k2;
	virtual INT_PTR DoModal();
};
```

（11）framework.h

```
#pragma once

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN		// 从 Windows 头中排除极少使用的资料
#endif

#include "targetver.h"

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS	// 某些 CString 构造函数将是显式的
```

```
// 关闭 MFC 的一些常见且经常可放心忽略的隐藏警告消息
#define _AFX_ALL_WARNINGS

#include <afxwin.h>               // MFC 核心组件和标准组件
#include <afxext.h>               // MFC 扩展




#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>                 // MFC 对 Internet Explorer 4 公共控件的支持
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                   // MFC 对 Windows 公共控件的支持
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxcontrolbars.h>       // MFC 支持功能区和控制条
```

（12）imageProcessing.h

```
// imageProcessing.h: imageProcessing 应用程序的主头文件
//
#pragma once

#ifndef __AFXWIN_H__
    #error "在包含此文件之前包含 'pch.h' 以生成 PCH"
#endif

#include "resource.h"           // 主符号

class CimageProcessingApp : public CWinApp
{
public:
    CimageProcessingApp() noexcept;

// 重写
public:
    virtual BOOL InitInstance();

// 实现
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CimageProcessingApp theApp;
```

（13）imageProcessingDoc.h

```
// imageProcessingDoc.h: CimageProcessingDoc 类的接口
//
```

```
#pragma once

class CimageProcessingDoc : public CDocument
{
protected: // 仅从序列化创建
	CimageProcessingDoc() noexcept;
	DECLARE_DYNCREATE(CimageProcessingDoc)

/**/
public:

// 特性
public:

// 操作
public:

// 重写
public:
	virtual BOOL OnNewDocument();
	virtual void Serialize(CArchive& ar);
#ifdef SHARED_HANDLERS
	virtual void InitializeSearchContent();
	virtual void OnDrawThumbnail(CDC& dc, LPRECT lprcBounds);
#endif // SHARED_HANDLERS

// 实现
public:
	virtual ~CimageProcessingDoc();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// 生成的消息映射函数
protected:
	DECLARE_MESSAGE_MAP()

#ifdef SHARED_HANDLERS
	// 用于为搜索处理程序设置搜索内容的 Helper 函数
	void SetSearchContent(const CString& value);
#endif // SHARED_HANDLERS
};
```

（14）imageProcessingView.h

```
// imageProcessingView.h: CimageProcessingView 类的接口
//
#pragma once
```

```
class CimageProcessingView : public CView
{
public:
	char *pFileBuf;
	/**/
protected: // 仅从序列化创建
	CimageProcessingView() noexcept;
	DECLARE_DYNCREATE(CimageProcessingView)

// 特性
public:
	CimageProcessingDoc* GetDocument() const;

// 重写
public:
	virtual void OnDraw(CDC* pDC);  // 重写以绘制该视图
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:

// 实现
public:
	virtual ~CimageProcessingView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// 生成的消息映射函数
protected:
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnImageprocessOpenbmpfile();
	afx_msg void OnImageprocessSavetofile();
	afx_msg void OnImageprocessDisplayfileheader();
	afx_msg void OnImageprocessDisplaypalette();
	afx_msg void OnImageprocessGetpixelvalue();
	afx_msg void OnImageprocessSetpixelvalue();
	afx_msg void OnImageprocessInterpolation();
	afx_msg void OnImageprocessGausssmooth();
	afx_msg void OnImageprocessMedianfilter();
	afx_msg void OnImageprocessBilateralfilter();
	afx_msg void OnImageprocessHistoequalization();
	afx_msg void OnImageprocessSharpengrad();
	afx_msg void OnImageprocessCannyedge();
	afx_msg void OnImageprocessOtsusegment();
	afx_msg void OnImageprocessHazeremoval();
};
```

```
#ifndef _DEBUG  // imageProcessingView.cpp 中的调试版本
inline CimageProcessingDoc* CimageProcessingView::GetDocument() const
   { return reinterpret_cast<CimageProcessingDoc*>(m_pDocument); }
#endif
```

（15）MainFrm.h

```
// MainFrm.h: CMainFrame 类的接口
//

#pragma once

class CMainFrame : public CFrameWnd
{
protected: // 仅从序列化创建
	CMainFrame() noexcept;
	DECLARE_DYNCREATE(CMainFrame)

// 特性
public:

// 操作
public:

// 重写
public:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// 实现
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

// 生成的消息映射函数
protected:
	DECLARE_MESSAGE_MAP()

};
```

（16）Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ 生成的包含文件。
// 供 imageProcessing.rc 使用
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_imageProcessingTYPE         130
#define IDD_CInputXY                    314
```

```
#define IDD_CInputXYRGB 315
#define IDD_CInterpolation 316
#define IDD_CGausssmooth 317
#define IDD_CMedianfilter 318
#define IDD_CBilateralfilter 319
#define IDD_CSharpengrad 320
#define IDD_CCannyedge 322
#define IDD_CHazeremoval 323
#define IDC_X 1008
#define IDC_Y 1009
#define IDC_R 1010
#define IDC_G 1011
#define IDC_Method 1011
#define IDC_B 1012
#define IDC_factorX 1012
#define IDC_Reserved 1013
#define IDC_factorY 1013
#define IDC_Gauss 1016
#define IDC_N1 1017
#define IDC_N2 1018
#define IDC_sigma_d 1021
#define IDC_EDIT4 1022
#define IDC_sigma_R 1022
#define IDC_k1 1023
#define IDC_k2 1024
#define IDC_sigma 1026
#define IDC_EDIT1 1027
#define IDC_w 1027
#define ID_IMAGEPROCESSING_OPENABMPFILE 32771
#define ID_IMAGEPROCESSING_SAVETONEWBMP 32772
#define ID_IMAGEPROCESSING_DISPLAYBMPFILEHEADER 32773
#define ID_IMAGEPROCESSING_DISPLAYPALETTE 32774
#define ID_IMAGEPROCESSING_GETPIXELVALUE 32775
#define ID_IMAGEPROCESSING_SETPIXELVALUE 32776
#define ID_IMAGEPROCESSING_IMAGEINERPOLATION 32777
#define ID_IMAGEPROCESSING_MEDIANFILTERING 32778
#define ID_IMAGEPROCESSING_GAUSSIANSMOOTHING 32779
#define ID_IMAGEPROCESSING_BILATERALFILTERING 32780
#define ID_IMAGEPROCESSING_HISTOGRAMEQUALIZATION 32781
#define ID_IMAGEPROCESSING_SHARPENINGBYGRADIENT 32782
#define ID_IMAGEPROCESSING_CANNYEDGEDETECTION 32783
#define ID_IMAGEPROCESSING_OTSUTHRESHOLDING 32784
#define ID_IMAGEPROCESS_OPENBMPFILE 32785
#define ID_IMAGEPROCESS_SAVETOFILE 32786
#define ID_IMAGEPROCESS_DISPLAYFILEHEADER 32787
#define ID_IMAGEPROCESS_DISPLAYPALETTE 32788
#define ID_IMAGEPROCESS_GETPIXELVALUE 32789
#define ID_IMAGEPROCESS_SETPIXELVALUE 32790
#define ID_IMAGEPROCESS_INERPOLATION 32791
#define ID_IMAGEPROCESS_GAUSSSMOOTH 32792
#define ID_IMAGEPROCESS_MEDIANFILTER 32793
```

```
#define ID_IMAGEPROCESS_BILATERALFILTER 32794
#define ID_IMAGEPROCESS_HISTOEQUALIZATION 32795
#define ID_IMAGEPROCESS_SHARPENGRAD 32796
#define ID_IMAGEPROCESS_CANNYEDGE 32797
#define ID_IMAGEPROCESS_OTSUSEGMENT 32798
#define ID_IMAGEPROCESSING_HAZE 32799
#define ID_IMAGEPROCESSING_HAZEREMOVAL 32800
#define ID_IMAGEPROCESS_HAZEREMOVAL 32801

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 325
#define _APS_NEXT_COMMAND_VALUE 32802
#define _APS_NEXT_CONTROL_VALUE 1028
#define _APS_NEXT_SYMED_VALUE 324
#endif
#endif
```

（17）targetver.h

```
#pragma once

// 包括 SDKDDKVer.h 将定义可用的最高版本的 Windows 平台。

// 如果要为以前的 Windows 平台生成应用程序，请包括 WinSDKVer.h，并将
// 将 _WIN32_WINNT 宏设置为要支持的平台，然后再包括 SDKDDKVer.h。

#include <SDKDDKVer.h>
```

（18）_GlobalCommon.cpp

```
//GlobalCommon.cpp
#define _CRT_SECURE_NO_WARNINGS
#include "_GlobalCommon.h"
#include <algorithm>
#include <cstdlib>
#include <cmath>
#include <queue>

/**
	功能: 从图像文件中建造 DIB 类
	参数: strBmpFile --- 需要打开的 BMP 文件名
	返回: 文件缓冲区指针 (NULL 表示失败)
**/
char *OpenBMPfile(CString strBmpFile)
{
	CFile hFile;
	if( !hFile.Open(strBmpFile,CFile::modeRead|CFile::typeBinary) )
	{
		AfxMessageBox("Failed to open the BMP file");
		return( NULL );
	}
```

```
	/**/
//	if(	hFile.Seek(0L,CFile::begin) != 0L )
//	{
//		return( NULL );
//	}
	/**/
	long lFileSize = (long)hFile.Seek(0L, CFile::end);
	char *pFileBuf = new char [lFileSize+1];
	hFile.Seek(0L, CFile::begin);
	hFile.Read(pFileBuf, lFileSize);
	hFile.Close();
	/**/
	BITMAPFILEHEADER *pBmpHead = (BITMAPFILEHEADER *)pFileBuf;
	BITMAPINFOHEADER *pBmpInfo = (BITMAPINFOHEADER *)(pFileBuf + sizeof(BITMAPFILEHEADER));
	/**/
	if(	pBmpHead->bfType	!= 0x4D42 ||		//"BM"=0x424D
		pBmpInfo->biSize	!= 0x28	||		// 位图信息子结构长度(等于 40,即 0x28)
		pBmpInfo->biPlanes != 0x01 )		// 此域必须等于 1
	{
		AfxMessageBox("It isn't a valid BMP file");
		return( NULL );
	}
	/**/
	if( pBmpInfo->biCompression != BI_RGB )
	{
		AfxMessageBox("It is a compressed BMP file");
		return( NULL );
	}
	/**/
	if( pBmpInfo->biBitCount != 8	&&
		pBmpInfo->biBitCount != 24 )
	{
		AfxMessageBox("Only 8-bit and 24-bit BMP files are supported");
		return( NULL );
	}
	/**/
	return( pFileBuf );
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////

BITMAPFILEHEADER *GetDIBHEADER(char *pFileBuffer)
{
	char *p = pFileBuffer + 0;
	return( (BITMAPFILEHEADER *)p );
}

BITMAPINFOHEADER *GetDIBINFO(char *pFileBuffer)
{
	char *p = pFileBuffer + sizeof(BITMAPFILEHEADER);
```

```
	return( (BITMAPINFOHEADER *)p );
}

char *GetDIBImageData(char *pFileBuffer)
{
	const BITMAPFILEHEADER *pBmpHead = GetDIBHEADER(pFileBuffer);
	char *p = pFileBuffer + pBmpHead->bfOffBits;
	return( p );
}

//return NULL denoting no palette
RGBQUAD *GetDIBPaletteData(char *pFileBuffer,int nEntryNumber[1])
{
	char *pPaletteData = NULL;
	if( GetColorBits(pFileBuffer) <= 8 )
	{
		nEntryNumber[0] = 0;
		char *pDIBImageData = GetDIBImageData(pFileBuffer);
		pPaletteData = pFileBuffer + sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
		int	nNum = (DWORD)(pDIBImageData - pPaletteData) / sizeof(RGBQUAD);
		int	iUsedColors = (int)GetDIBINFO(pFileBuffer)->biClrUsed;
		if( nNum > 0 && (int)iUsedColors > 0 )
			nEntryNumber[0] = min(nNum,(int)iUsedColors);
		else
			pPaletteData = NULL;
	}
	return( (RGBQUAD *)pPaletteData );
}

int GetImageWidth(char *pFileBuffer)
{
	BITMAPINFOHEADER *pInfo = GetDIBINFO(pFileBuffer);
	return( pInfo->biWidth );
}

int GetImageHeight(char *pFileBuffer)
{
	BITMAPINFOHEADER *pInfo = GetDIBINFO(pFileBuffer);
	return( pInfo->biHeight );
}

int GetColorBits(char *pFileBuffer)
{
	BITMAPINFOHEADER *pInfo = GetDIBINFO(pFileBuffer);
	return( pInfo->biBitCount );
}

long GetUsedColors(char *pFileBuffer)
{
	BITMAPINFOHEADER *pInfo = GetDIBINFO(pFileBuffer);
	return( (long)pInfo->biClrUsed );
```

```
}

long GetWidthBytes(char *pFileBuffer)
{
	BITMAPINFOHEADER *pInfo = GetDIBINFO(pFileBuffer);
	long nBytesPerRow = 4 * ((pInfo->biWidth * pInfo->biBitCount + 31) / 32);
	return( nBytesPerRow );
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

void DisplayHeaderMessage(char *pBmpFileBuf)
{
	BITMAPFILEHEADER *pBmpHead = GetDIBHEADER(pBmpFileBuf);
	BITMAPINFOHEADER *pBmpInfo = GetDIBINFO(pBmpFileBuf);
	/**/
	char msg[4096];
	sprintf(msg,
	"bfType (file type) = %4.4X \n"
	"bfSize (file length) = %ld \n"
	"bfOffBits (offset of bitmap data in bytes) = %ld \n"
	"biSize (header structure length should be 40 or 0x28) = %ld \n"
	"biWidth (image width)    = %ld \n"
	"biHeight (image height) = %ld \n"
	"biPlanes (must be eaual to 1) = %u \n"
	"biBitCount (color/pixel bits) = %u \n"
	"biCompression (compressed?) = %ld \n"
	"biSizeImage (length of bitmap data in bytes must be the times of 4) = %ld \n"
	"biXPelsPerMeter (horizontal resolution of target device in pixels/metre) = %ld \n"
	"biYPelsPerMeter (vertical resolution of target device in pixels/metre) = %ld \n"
	"biColorUsed (number of colors used in bitmap,0=2**biBitCount) = %ld \n"
	"biColorImportant (number of important colors,0=all colors are important) = %ld \n\n"
	"The following is additional information: \n"
	"Bytes per row in bitmap (nBytesPerRow) = %ld \n"
	"Total bytes of bitmap (nImageSizeInByte) = %ld \n"
	"Actual pixels per row in bitmap (nPixelsPerRow) = %ld \n"
	"Total rows of bitmap (nTotalRows) = %ld \n"
	"Total colors (2**biBitCount)(nTotalColors) = %ld \n"
	"Used colors (biColorUsed)(nUsedColors) = %ld ",
	pBmpHead->bfType,
	pBmpHead->bfSize,
	pBmpHead->bfOffBits,
	pBmpInfo->biSize,
	pBmpInfo->biWidth,
	pBmpInfo->biHeight,
	pBmpInfo->biPlanes,
	pBmpInfo->biBitCount,
	pBmpInfo->biCompression,
	pBmpInfo->biSizeImage,
	pBmpInfo->biXPelsPerMeter,
```

```
        pBmpInfo->biYPelsPerMeter,
        pBmpInfo->biClrUsed,
        pBmpInfo->biClrImportant,
        GetWidthBytes(pBmpFileBuf),
        GetWidthBytes(pBmpFileBuf) * GetImageHeight(pBmpFileBuf),
        GetImageWidth(pBmpFileBuf),
        GetImageHeight(pBmpFileBuf),
        1 << GetColorBits(pBmpFileBuf),
        GetUsedColors(pBmpFileBuf) );
        AfxMessageBox(msg);
}

//Mode = 0, normal display
//      1,2,3, display grayscale image in red, green, blue colors
void DisplayImage(CDC *pDC,char *pBmpFileBuf,int disp_xL,int disp_yL,int disp_Width,int disp_Height,int mode)
{
        ASSERT( pDC != NULL );
        HDC hDC = pDC->GetSafeHdc();
        ASSERT( hDC != 0 );
        /**/
        int imageWidth    = GetImageWidth(pBmpFileBuf);
        int imageHeight = GetImageHeight(pBmpFileBuf);
        if( disp_Width <= 0 || disp_Height <= 0 )
        {
                disp_Width    = imageWidth;
                disp_Height = imageHeight;
        }
        CRect rect;
        CWnd *pWnd = pDC->GetWindow();
        pWnd->GetClientRect(&rect);
        disp_Width = min(disp_Width, rect.right - disp_xL);
        disp_Height = min(disp_Height, rect.bottom - disp_yL);
        /**/
        BITMAPINFOHEADER *pBitmapInfo = GetDIBINFO(pBmpFileBuf);
        char *pDIBImageData = GetDIBImageData(pBmpFileBuf);
        /**/
        char buf[40+256*4];
        BITMAPINFO *pBitsInfo = (BITMAPINFO *)buf;
        memcpy(&pBitsInfo->bmiHeader,pBitmapInfo,sizeof(BITMAPINFOHEADER));
        /**/
        int palleteNum = 0;
        RGBQUAD *pallete = GetDIBPaletteData(pBmpFileBuf,&palleteNum);
        for(int c = 0; c < 256; c++)
        {
                if( mode == 0 )
                {
                        (pBitsInfo->bmiColors[c]).rgbRed      = (pallete!=NULL && c<palleteNum? pallete[c].rgbRed      : c);
                        (pBitsInfo->bmiColors[c]).rgbGreen = (pallete!=NULL && c<palleteNum? pallete[c].rgbGreen : c);
                        (pBitsInfo->bmiColors[c]).rgbBlue    = (pallete!=NULL && c<palleteNum? pallete[c].rgbBlue    : c);
                }
                else
```

```
            {
                (pBitsInfo->bmiColors[c]).rgbRed   = (mode==1? c : 0);
                (pBitsInfo->bmiColors[c]).rgbGreen = (mode==2? c : 0);
                (pBitsInfo->bmiColors[c]).rgbBlue  = (mode==3? c : 0);
            }
        }
        /**/
        SetStretchBltMode(hDC,COLORONCOLOR);
        StretchDIBits(hDC,disp_xL,disp_yL,disp_Width,disp_Height,
        0,0,imageWidth,imageHeight,pDIBImageData,pBitsInfo,DIB_RGB_COLORS,SRCCOPY );
        /**/
        return;
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//    象素操作

//      读象素颜色值
//      返回: >=0  表示象素在位图数据中的偏移值
//                <0  失败或参数无效
long GetPixel(char *pFileBuffer,int x,int y,RGBQUAD rgb[1],bool bGray[1])
{
        int   nColorBits    = GetColorBits(pFileBuffer);
        int   nImageHeight = GetImageHeight(pFileBuffer);
        int   nBytesPerRow = GetWidthBytes(pFileBuffer);
        /**/
        long nOffInImage   = (nImageHeight-1-y) * nBytesPerRow;
        char *p = GetDIBImageData(pFileBuffer) + nOffInImage;
        /**/
        if( bGray != NULL ) *bGray = true;
        if( nColorBits == 8 )
        {
                nOffInImage += x;
                rgb[0].rgbReserved = p[x];
                rgb[0].rgbRed          = p[x];
                rgb[0].rgbGreen       = p[x];
                rgb[0].rgbBlue         = p[x];
        }
        else if( nColorBits == 24 )
        {
                if( bGray != NULL ) *bGray = false;
                nOffInImage += 3 * x;
                p += (3 * x);
                rgb[0].rgbReserved = 0;
                rgb[0].rgbRed          = p[2];
                rgb[0].rgbGreen       = p[1];
                rgb[0].rgbBlue         = p[0];
        }
        else
```

```
    {
        AfxMessageBox("It is not an 8-bit or 24-bit image");
        return( -1L );
    }
    /**/
    return( nOffInImage );
}

//  设置像素(x,y)的颜色值
void SetPixel(char *pFileBuffer,int x,int y,RGBQUAD rgb)
{
    int   nColorBits     = GetColorBits(pFileBuffer);
    int   nImageHeight = GetImageHeight(pFileBuffer);
    int   nBytesPerRow = GetWidthBytes(pFileBuffer);
    /**/
    long nOffInImage   = (nImageHeight-1-y) * nBytesPerRow;
    char *p = GetDIBImageData(pFileBuffer) + nOffInImage;
    /**/
    if( nColorBits == 8 )
    {
        p[ x ] = rgb.rgbReserved;
    }
    else if( nColorBits == 24 )
    {
        p += (3 * x);
        p[0] = rgb.rgbBlue;
        p[1] = rgb.rgbGreen;
        p[2] = rgb.rgbRed;
    }
    else
    {
        AfxMessageBox("It is not an 8-bit or 24-bit image");
    }
    /**/
    return;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// 保存为 BMP 文件

BOOL SaveDIB(char *pFileBuffer,CString strBmpFile)
{
    CFile hFile;
    if( !hFile.Open(strBmpFile,CFile::modeCreate|CFile::modeWrite|CFile::typeBinary) )
    {
        AfxMessageBox("Failed to create the BMP file");
        return( FALSE );
    }
    /**/
    BITMAPFILEHEADER *pBmpHead = (BITMAPFILEHEADER *)pFileBuffer;
```

```
	long lFileSize = pBmpHead->bfSize;
	hFile.Write(pFileBuffer,lFileSize);
	hFile.Close();
	return( TRUE );
}

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
// 图像插值

/**
	功能: 图像插值
		nMethod	插值算法
			0 = 最临近插值法
			1 = (双)线性插值法
	返回: 新图像的 BMP 文件缓冲区首地址
		NULL 表示失败（内存不足）
**/
char *ImageInterpolation(char *pBmpFileBuf,int newWidth,int newHeight,int nMethod)
{
	BITMAPFILEHEADER *pFileHeader = (BITMAPFILEHEADER *)pBmpFileBuf;
	BITMAPINFOHEADER *pDIBInfo = (BITMAPINFOHEADER *)(pBmpFileBuf + sizeof(BITMAPFILEHEADER));
//	char *pDIBData = pBmpFileBuf + pFileHeader->bfOffBits;
	int	orgWidth  = pDIBInfo->biWidth;
	int	orgHeight = pDIBInfo->biHeight;
	int	colorBits = pDIBInfo->biBitCount;
	/**/
	long bytesPerRow = 4 * ((newWidth * colorBits + 31) / 32);
	long newBmpFileSize = pFileHeader->bfOffBits + bytesPerRow * newHeight;
	char *pNewBmpFileBuf = new char [newBmpFileSize];
	memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);
	BITMAPFILEHEADER *pNewFileHeader = (BITMAPFILEHEADER *)pNewBmpFileBuf;
	BITMAPINFOHEADER *pNewDIBInfo = (BITMAPINFOHEADER *)(pNewBmpFileBuf + sizeof(BITMAPFILEHEADER));
	pNewFileHeader->bfSize = newBmpFileSize;
	pNewDIBInfo->biWidth = newWidth;
	pNewDIBInfo->biHeight = newHeight;
	pNewDIBInfo->biSizeImage = bytesPerRow * newHeight;
//	char *pNewDIBData = pNewBmpFileBuf + pFileHeader->bfOffBits;
	/**/
	/**/
	float xScale  = (float)orgWidth  / (float)newWidth;
	float yScale  = (float)orgHeight / (float)newHeight;
	for(int y = 0; y < newHeight; y++)
	{
		float fy = y * yScale;
		for(int x = 0; x < newWidth; x++)
		{
			RGBQUAD rgb;
```

```
                float fx = x * xScale;
                if( nMethod == 0 )              //最临近插值法
                {
                    int xx = min( (int)(fx+0.5), orgWidth - 1 );
                    int yy = min( (int)(fy+0.5), orgHeight - 1 );
                    GetPixel(pBmpFileBuf, xx, yy, &rgb);
                }
                else
                {                                   //(双)线性插值法
                    RGBQUAD rgbLT,rgbRT,rgbLB,rgbRB;
                    int     x1 = (int)fx;
                    int     x2 = min(x1+1, orgWidth-1);
                    float dx = fx - (float)x1;
                    int     y1 = (int)fy;
                    int     y2 = min(y1+1, orgHeight-1);
                    float dy = fy - (float)y1;
                    GetPixel(pBmpFileBuf, x1, y1, &rgbLT);
                    GetPixel(pBmpFileBuf, x2, y1, &rgbRT);
                    GetPixel(pBmpFileBuf, x1, y2, &rgbLB);
                    GetPixel(pBmpFileBuf, x2, y2, &rgbRB);
                    for(int N = 0; N < 4; N++)
                    {
                        float v1 = ((BYTE *)&rgbLT)[N] + dy * (((BYTE *)&rgbLB)[N] - ((BYTE *)&rgbLT)[N]);
                        float v2 = ((BYTE *)&rgbRT)[N] + dy * (((BYTE *)&rgbRB)[N] - ((BYTE *)&rgbRT)[N]);
                        ((BYTE *)&rgb)[N] = (int)(v1 + dx * (v2 - v1) + 0.5);
                    }
                }
                SetPixel(pNewBmpFileBuf, x, y, rgb);
            }
        }
        /**/
        return( pNewBmpFileBuf );
}

/**
        功能: 高斯平滑
                sigma   高斯函数的均方差
        返回: 新图像的 BMP 文件缓冲区首地址
                NULL 表示失败（内存不足）
**/
char* ImageGausssmooth(char* pBmpFileBuf, int sigma)
{
        BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
        BITMAPINFOHEADER*       pDIBInfo        =       (BITMAPINFOHEADER*)(pBmpFileBuf         +
sizeof(BITMAPFILEHEADER));

        char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
        memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

        // 计算高斯核
```

```
    int R = (sigma << 1) + 1;
    double* Kernel = new double[R * R];
    for (int i = 0; i < R * R; i++)Kernel[i] = 0.0;
    double tot_value = 0.0;
    for (int y = -sigma; y <= sigma; y++) {
        for (int x = -sigma; x <= sigma; x++) {
            int id = (y + sigma) * R + x + sigma;
            Kernel[id] = exp(-1.0 * (x * x + y * y) / (2.0 * sigma * sigma));
            tot_value += Kernel[id];
        }
    }
    for (int i = 0; i < R * R; i++) {
        Kernel[i] /= tot_value;
    }

    int Width = pDIBInfo->biWidth;
    int Height = pDIBInfo->biHeight;

    for (int y = 0; y < Height; y++) {
        for (int x = 0; x < Width; x++) {

            RGBQUAD rgb;
            double r = 0.0, g = 0.0, b = 0.0, res = 0.0;
            int id = 0;
            for (int i = -sigma; i <= sigma; i++) {
                for (int j = -sigma; j <= sigma; j++) {
                    int ny = y + i, nx = x + j;
                    if (ny < 0)ny = 0;
                    if (ny >= Height)ny = Height - 1;
                    if (nx < 0)nx = 0;
                    if (nx >= Width)nx = Width - 1;
                    GetPixel(pBmpFileBuf, nx, ny, &rgb);
                    b += Kernel[id] * ((BYTE*)&rgb)[0];
                    g += Kernel[id] * ((BYTE*)&rgb)[1];
                    r += Kernel[id] * ((BYTE*)&rgb)[2];
                    res += Kernel[id] * ((BYTE*)&rgb)[3];
                    id++;
                }
            }

            ((BYTE*)&rgb)[0] = (int)b;
            ((BYTE*)&rgb)[1] = (int)g;
            ((BYTE*)&rgb)[2] = (int)r;
            ((BYTE*)&rgb)[3] = int(res);

            SetPixel(pNewBmpFileBuf, x, y, rgb);
        }
    }

    return pNewBmpFileBuf;
}
```

```
/**
	功能: 中值滤波
		N1  滤波器 X 方向半径
		N2  滤波器 Y 方向半径
	返回: 新图像的 BMP 文件缓冲区首地址
		NULL 表示失败（内存不足）
**/
char* ImageMedianfilter(char* pBmpFileBuf, int N1, int N2)
{
	BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
	BITMAPINFOHEADER* pDIBInfo = (BITMAPINFOHEADER*)(pBmpFileBuf + sizeof(BITMAPFILEHEADER));

	char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
	memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

	int siz = (N1 * 2 + 1) * (N2 * 2 + 1);
	std::pair<double, RGBQUAD>* elem = new std::pair<double, RGBQUAD>[siz];

	int Width = pDIBInfo->biWidth;
	int Height = pDIBInfo->biHeight;

	for (int y = 0; y < Height; y++) {
		for (int x = 0; x < Width; x++) {

			RGBQUAD rgb;
			double brightness = 0.0;
			int id = 0;
			for (int i = -N2; i <= N2; i++) {
				for (int j = -N1; j <= N1; j++) {
					int ny = y + i, nx = x + j;
					if (ny < 0)ny = 0;
					if (ny >= Height)ny = Height - 1;
					if (nx < 0)nx = 0;
					if (nx >= Width)nx = Width - 1;
					GetPixel(pBmpFileBuf, nx, ny, &rgb);

					// 计算颜色亮度
					brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0] * 114)) / 1000.0;
					elem[id] = std::make_pair(brightness, rgb);
					id++;
				}
			}

			std::sort(elem, elem + siz, [](std::pair<double, RGBQUAD>& a, std::pair<double, RGBQUAD>& b) {return a.first < b.first; });

			SetPixel(pNewBmpFileBuf, x, y, elem[(siz + 1) >> 1].second);
```

```
            }
        }

        return pNewBmpFileBuf;
}

/**
    功能: 双边滤波
            N1          滤波器 X 方向半径
            N2          滤波器 Y 方向半径
            sigma_d     距离高斯函数均方差
            sigma_R     颜色高斯函数均方差
    返回: 新图像的 BMP 文件缓冲区首地址
            NULL 表示失败（内存不足）
**/
char* ImageBilateralfilter(char* pBmpFileBuf, int N1, int N2, int sigma_d, int sigma_R)
{
        BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
        BITMAPINFOHEADER*  pDIBInfo  =  (BITMAPINFOHEADER*)(pBmpFileBuf  +
sizeof(BITMAPFILEHEADER));

        char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
        memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

        int lenX = 2 * N1 + 1;
        int lenY = 2 * N2 + 1;

        // 计算距离高斯函数核
        double* Kernel_d = new double[lenY * lenX];
        for (int i = 0; i < lenY * lenX; i++)Kernel_d[i] = 0.0;
        for (int y = -N2; y <= N2; y++) {
                for (int x = -N1; x <= N1; x++) {
                        int id = (y + N2) * lenX + x + N1;
                        Kernel_d[id] = exp(-1.0 * (x * x + y * y) / (2.0 * sigma_d * sigma_d));
                }
        }

        int Width = pDIBInfo->biWidth;
        int Height = pDIBInfo->biHeight;

        for (int y = 0; y < Height; y++) {
                for (int x = 0; x < Width; x++) {

                        RGBQUAD rgb, crgb;
                        int id = 0;
                        double totvalue = 0.0, weight = 0.0, r = 0.0, g = 0.0, b = 0.0, res = 0.0;
                        GetPixel(pBmpFileBuf, x, y, &crgb);
```

```
                for (int i = -N2; i <= N2; i++) {
                    for (int j = -N1; j <= N1; j++) {
                        int ny = y + i, nx = x + j;
                        if (ny < 0)ny = 0;
                        if (ny >= Height)ny = Height - 1;
                        if (nx < 0)nx = 0;
                        if (nx >= Width)nx = Width - 1;
                        GetPixel(pBmpFileBuf, nx, ny, &rgb);

                        // 计算颜色高斯函数核
                        int d_r = ((BYTE*)&rgb)[2] - ((BYTE*)&crgb)[2];
                        int d_g = ((BYTE*)&rgb)[1] - ((BYTE*)&crgb)[1];
                        int d_b = ((BYTE*)&rgb)[0] - ((BYTE*)&crgb)[0];
                        weight = exp(-1.0 * (d_r * d_r + d_g * d_g + d_b * d_b) / (2.0 * sigma_R * sigma_R));

                        totvalue += weight * Kernel_d[id];
                        r += weight * Kernel_d[id] * ((BYTE*)&rgb)[2];
                        g += weight * Kernel_d[id] * ((BYTE*)&rgb)[1];
                        b += weight * Kernel_d[id] * ((BYTE*)&rgb)[0];
                        res += weight * Kernel_d[id] * ((BYTE*)&rgb)[3];

                        id++;
                    }
                }

                ((BYTE*)&rgb)[0] = (int)(b / totvalue);
                ((BYTE*)&rgb)[1] = (int)(g / totvalue);
                ((BYTE*)&rgb)[2] = (int)(r / totvalue);
                ((BYTE*)&rgb)[3] = (int)(res / totvalue);

                SetPixel(pNewBmpFileBuf, x, y, rgb);
            }
        }

        return pNewBmpFileBuf;
}

/**
        功能: 直方图均衡化
        返回: 新图像的 BMP 文件缓冲区首地址
                NULL 表示失败（内存不足）
**/
char* ImageHistoequalization(char* pBmpFileBuf)
{
        BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
        BITMAPINFOHEADER*       pDIBInfo        =       (BITMAPINFOHEADER*)(pBmpFileBuf         +
sizeof(BITMAPFILEHEADER));

        char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
        memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);
```

```
	int Width = pDIBInfo->biWidth;
	int Height = pDIBInfo->biHeight;

	// 统计图像中每个灰度级的像素数量
	int* num = new int[256];
	for (int i = 0; i < 256; i++)num[i] = 0;
	for (int y = 0; y < Height; y++) {
		for (int x = 0; x < Width; x++) {
			RGBQUAD rgb;
			GetPixel(pBmpFileBuf, x, y, &rgb);
			double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0]
* 114)) / 1000.0;
			num[(int)brightness]++;
		}
	}

	// 计算每个灰度级像素的频率并做前缀和
	double* perc_num = new double[256];
	int maxn = Width * Height;
	perc_num[0] = 1.0 * num[0] / maxn;
	for (int i = 1; i < 256; i++) {
		perc_num[i] = 1.0 * num[i] / maxn;
		perc_num[i] += perc_num[i - 1];
	}

	// 根据统计出的结果重新计算每个像素的灰度级并更新图像
	for (int y = 0; y < Height; y++) {
		for (int x = 0; x < Width; x++) {
			RGBQUAD rgb;
			GetPixel(pBmpFileBuf, x, y, &rgb);
			double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0]
* 114)) / 1000.0;
			int new_brightness = int(perc_num[(int)brightness] * 255.0 + 0.5);

			((BYTE*)&rgb)[3] = min(255, int(1.0 * ((BYTE*)&rgb)[3] / int(brightness) * new_brightness));
			((BYTE*)&rgb)[2] = min(255, int(1.0 * ((BYTE*)&rgb)[2] / int(brightness) * new_brightness));
			((BYTE*)&rgb)[1] = min(255, int(1.0 * ((BYTE*)&rgb)[1] / int(brightness) * new_brightness));
			((BYTE*)&rgb)[0] = min(255, int(1.0 * ((BYTE*)&rgb)[0] / int(brightness) * new_brightness));

			SetPixel(pNewBmpFileBuf, x, y, rgb);
		}
	}

	return pNewBmpFileBuf;
}

/**
	功能: 基于梯度的锐化
		k1	基本信息保留因子
```

```
                k2    细节增强强度系数
        返回: 新图像的 BMP 文件缓冲区首地址
                NULL 表示失败（内存不足）
**/
char* ImageSharpengrad(char* pBmpFileBuf, double k1, double k2)
{
    BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
    BITMAPINFOHEADER* pDIBInfo = (BITMAPINFOHEADER*)(pBmpFileBuf + sizeof(BITMAPFILEHEADER));

    char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
    memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

    int Width = pDIBInfo->biWidth;
    int Height = pDIBInfo->biHeight;

    for (int y = 0; y < Height; y++) {
        for (int x = 0; x < Width; x++) {

            RGBQUAD rgb, crgb;
            // 获得当前像素的颜色值
            GetPixel(pBmpFileBuf, x, y, &crgb);
            double cbrightness = 1.0 * ((((BYTE*)&crgb)[2] * 299) + (((BYTE*)&crgb)[1] * 587) + (((BYTE*)&crgb)[0] * 114)) / 1000.0;

            // 计算 X 方向的梯度
            int ny = y, nx = x + 1;
            if (nx >= Width)nx = Width - 1;
            GetPixel(pBmpFileBuf, nx, ny, &rgb);
            double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0] * 114)) / 1000.0;
            double gradX = brightness - cbrightness;

            // 计算 Y 方向的梯度
            ny = y + 1, nx = x;
            if (ny >= Height)ny = Height - 1;
            GetPixel(pBmpFileBuf, nx, ny, &rgb);
            brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0] * 114)) / 1000.0;
            double gradY = brightness - cbrightness;

            double grad = sqrt(gradX * gradX + gradY * gradY);

            ((BYTE*)&rgb)[3] = min(255, int(k1 * ((BYTE*)&crgb)[3] + k2 * grad));
            ((BYTE*)&rgb)[2] = min(255, int(k1 * ((BYTE*)&crgb)[2] + k2 * grad));
            ((BYTE*)&rgb)[1] = min(255, int(k1 * ((BYTE*)&crgb)[1] + k2 * grad));
            ((BYTE*)&rgb)[0] = min(255, int(k1 * ((BYTE*)&crgb)[0] + k2 * grad));

            SetPixel(pNewBmpFileBuf, x, y, rgb);
        }
```

```
	}

	return pNewBmpFileBuf;
}

/**
	功能: Canny 算子边缘检测步骤 1
	返回: 新图像的 BMP 文件缓冲区首地址
		NULL 表示失败（内存不足）
**/
char* ImageCannyedgeStep1(char* pBmpFileBuf)
{
	BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
	BITMAPINFOHEADER* pDIBInfo = (BITMAPINFOHEADER*)(pBmpFileBuf + sizeof(BITMAPFILEHEADER));

	char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
	memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

	int Width = pDIBInfo->biWidth;
	int Height = pDIBInfo->biHeight;

	int dx[8] = { 1,1,0,-1,-1,-1,0,1 };
	int dy[8] = { 0,-1,-1,-1,0,1,1,1 };
	int sobel_x[3][3] = { {-1,0,1},{-2,0,2},{-1,0,1} };
	int sobel_y[3][3] = { {-1,-2,-1},{0,0,0},{1,2,1} };
	double pi = acos(-1);
	double* grad = new double[Height * Width];
	int* theta = new int[Height * Width];

	// 利用 Sobel 算子计算每个点处的梯度模和方向
	for (int y = 0; y < Height; y++) {
		for (int x = 0; x < Width; x++) {

			RGBQUAD rgb;
			double gradX = 0.0, gradY = 0.0;
			for (int i = -1; i <= 1; i++) {
				for (int j = -1; j <= 1; j++) {

					int ny = y + i, nx = x + j;
					if (ny < 0)ny = 0;
					if (ny >= Height)ny = Height - 1;
					if (nx < 0)nx = 0;
					if (nx >= Width)nx = Width - 1;

					GetPixel(pBmpFileBuf, nx, ny, &rgb);
					double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0] * 114)) / 1000.0;

					gradX += brightness * sobel_x[i + 1][j + 1];
```

```
                    gradY += brightness * sobel_y[i + 1][j + 1];
                }
            }

            int index = y * Width + x;
            grad[index] = sqrt(gradX * gradX + gradY * gradY);
            if (fabs(gradX) < 1e-4) {
                if (gradY > 0)theta[index] = 2;
                else theta[index] = 6;
            }
            else {
                double angle = atan(gradY / gradX);
                if (gradX > 0) {
                    if (gradY < 0)angle += 2.0 * pi;
                }
                else if (gradX < 0)angle += pi;

                double min_delta = 30.0;
                int dir = -1;
                double now_angle = 0.0;
                for (int i = 0; i < 7; i++) {
                    double tmp = angle - now_angle;
                    double now_delta = min(fabs(tmp), min(fabs(tmp + 2 * pi), fabs(tmp - 2 * pi)));
                    if (now_delta < min_delta) {
                        min_delta = now_delta;
                        dir = i;
                    }
                }
                theta[index] = dir;
            }
        }
    }

    // 非极大值抑制
    int id = 0;
    RGBQUAD rgb;
    for (int y = 0; y < Height; y++) {
        for (int x = 0; x < Width; x++) {

            bool check = true;
            int pos = theta[id], neg = (theta[id] + 4) % 8;
            // 检测梯度正方向
            int nx = x + dx[pos], ny = y + dy[pos];
            if (nx < 0)nx = 0;
            if (nx >= Width)nx = Width - 1;
            if (ny < 0)ny = 0;
            if (ny >= Height)ny = Height - 1;
            check &= grad[id] >= grad[ny * Width + nx];
            // 检测梯度负方向
            nx = x + dx[neg], ny = y + dy[neg];
            if (nx < 0)nx = 0;
```

```
                if (nx >= Width)nx = Width - 1;
                if (ny < 0)ny = 0;
                if (ny >= Height)ny = Height - 1;
                check &= grad[id] >= grad[ny * Width + nx];

                if (check) {
                        ((BYTE*)&rgb)[3] = (int)grad[id];
                        ((BYTE*)&rgb)[2] = (int)grad[id];
                        ((BYTE*)&rgb)[1] = (int)grad[id];
                        ((BYTE*)&rgb)[0] = (int)grad[id];
                }
                else {
                        ((BYTE*)&rgb)[3] = 0;
                        ((BYTE*)&rgb)[2] = 0;
                        ((BYTE*)&rgb)[1] = 0;
                        ((BYTE*)&rgb)[0] = 0;
                }

                SetPixel(pNewBmpFileBuf, x, y, rgb);

                id++;
            }
        }

        return pNewBmpFileBuf;
}

/**
        功能: Canny 算子边缘检测步骤 2
                maxth    强阈值
                minth    弱阈值
        返回: 新图像的 BMP 文件缓冲区首地址
                NULL 表示失败（内存不足）
**/
char* ImageCannyedgeStep2(char* pBmpFileBuf, int maxth, int minth)
{
        BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
        BITMAPINFOHEADER*        pDIBInfo        =        (BITMAPINFOHEADER*)(pBmpFileBuf        +
sizeof(BITMAPFILEHEADER));

        char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
        memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

        int Width = pDIBInfo->biWidth;
        int Height = pDIBInfo->biHeight;

        // 从所有灰度值高于强阈值的像素出发进行遍历，标记相邻的灰度值不低于弱阈值的像素
        int dx[8] = { 1,1,0,-1,-1,-1,0,1 };
        int dy[8] = { 0,-1,-1,-1,0,1,1,1 };
        bool* vis = new bool[Height * Width];
```

```
    for (int i = 0; i < Height * Width; i++)vis[i] = false;
    std::queue<std::pair<int, int> >q;
    RGBQUAD rgb;
    for (int y = 0; y < Height; y++) {
        for (int x = 0; x < Width; x++) {
            GetPixel(pBmpFileBuf, x, y, &rgb);
            double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0]
* 114)) / 1000.0;
            if ((int)brightness > maxth) {
                int id = y * Width + x;
                vis[id] = true;
                q.push(std::make_pair(x, y));
            }
        }
    }
    while (!q.empty()) {
        int x = q.front().first, y = q.front().second;
        q.pop();
        for (int i = 0; i < 8; i++) {
            int nx = x + dx[i], ny = y + dy[i];
            if (nx < 0 || nx >= Width || ny < 0 || ny >= Height)continue;
            int index = ny * Width + nx;
            if (vis[index])continue;

            GetPixel(pBmpFileBuf, nx, ny, &rgb);
            double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0]
* 114)) / 1000.0;
            if ((int)brightness > minth) {
                vis[index] = true;
                q.push(std::make_pair(nx, ny));
            }
        }
    }

    int id = 0;
    for (int y = 0; y < Height; y++) {
        for (int x = 0; x < Width; x++) {
            if (vis[id]) {
                GetPixel(pBmpFileBuf, x, y, &rgb);
            }
            else {
                ((BYTE*)&rgb)[3] = 0;
                ((BYTE*)&rgb)[2] = 0;
                ((BYTE*)&rgb)[1] = 0;
                ((BYTE*)&rgb)[0] = 0;
            }
            SetPixel(pNewBmpFileBuf, x, y, rgb);
            id++;
        }
    }
```

```
	return pNewBmpFileBuf;
}

/**
	功能: Otsu 图像分割
		threshold	计算得到的最佳分割阈值
	返回: 新图像的 BMP 文件缓冲区首地址
		NULL 表示失败（内存不足）
**/
char* ImageOtsusegment(char* pBmpFileBuf, int& threshold)
{
	BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
	BITMAPINFOHEADER* pDIBInfo = (BITMAPINFOHEADER*)(pBmpFileBuf + sizeof(BITMAPFILEHEADER));

	char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
	memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

	int Width = pDIBInfo->biWidth;
	int Height = pDIBInfo->biHeight;

	// 计算频率直方图 & 频率累积直方图
	double* histo = new double[256];
	double* acc_histo = new double[256];
	for (int i = 0; i < 256; i++)histo[i] = 0.0;
	for (int i = 0; i < 256; i++)acc_histo[i] = 0.0;
	RGBQUAD rgb;
	for (int y = 0; y < Height; y++) {
		for (int x = 0; x < Width; x++) {
			GetPixel(pBmpFileBuf, x, y, &rgb);
			double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0] * 114)) / 1000.0;
			histo[(int)brightness] += 1.0;
		}
	}
	double fm = 1.0 * Width * Height;
	acc_histo[0] = histo[0] /= fm;
	for (int i = 1; i < 256; i++) {
		histo[i] /= fm;
		acc_histo[i] = histo[i] + acc_histo[i - 1];
	}

	// 计算均值前缀和
	double* mu = new double[256];
	mu[0] = 0.0;
	for (int i = 1; i < 256; i++) {
		mu[i] = 1.0 * i * histo[i];
		mu[i] += mu[i - 1];
	}
```

```
    double max_var = 0.0;

    // 枚举可能的阈值
    for (int now_threshold = 1; now_threshold < 256; now_threshold++) {

        // 计算类间方差
        double w1 = acc_histo[now_threshold - 1];
        double w2 = 1 - w1;
        double mu1 = mu[now_threshold - 1] / w1;
        double mu2 = (mu[255] - mu[now_threshold - 1]) / w2;
        double var = w1 * w2 * (mu2 - mu1) * (mu2 - mu1);

        if (var > max_var) {
            max_var = var;
            threshold = now_threshold;
        }
    }

    // 图像分割
    for (int y = 0; y < Height; y++) {
        for (int x = 0; x < Width; x++) {
            GetPixel(pBmpFileBuf, x, y, &rgb);
            double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0]
* 114)) / 1000.0;
            if ((int)brightness < threshold) {
                ((BYTE*)&rgb)[3] = 0;
                ((BYTE*)&rgb)[2] = 0;
                ((BYTE*)&rgb)[1] = 0;
                ((BYTE*)&rgb)[0] = 0;
            }
            else {
                ((BYTE*)&rgb)[3] = 255;
                ((BYTE*)&rgb)[2] = 255;
                ((BYTE*)&rgb)[1] = 255;
                ((BYTE*)&rgb)[0] = 255;
            }
            SetPixel(pNewBmpFileBuf, x, y, rgb);
        }
    }

    return pNewBmpFileBuf;
}

/**
    功能: 图像去雾
            w    计算透射率时的参数 w（0.8~0.95 之间）
    返回: 新图像的 BMP 文件缓冲区首地址
          NULL 表示失败（内存不足）
**/
char* ImageHazeremoval(char* pBmpFileBuf, double w)
```

```
{
    BITMAPFILEHEADER* pFileHeader = (BITMAPFILEHEADER*)pBmpFileBuf;
    BITMAPINFOHEADER* pDIBInfo = (BITMAPINFOHEADER*)(pBmpFileBuf + sizeof(BITMAPFILEHEADER));

    char* pNewBmpFileBuf = new char[pFileHeader->bfSize];
    memcpy(pNewBmpFileBuf, pBmpFileBuf, pFileHeader->bfOffBits);

    int Width = pDIBInfo->biWidth;
    int Height = pDIBInfo->biHeight;

    // 计算暗通道图像
    std::pair<int, std::pair<int, int>>* darkimage = new std::pair<int, std::pair<int, int>>[Height * Width];
    for (int i = 0; i < Height * Width; i++)darkimage[i].first = 300;
    int id = 0;
    int R = 3;
    for (int y = 0; y < Height; y++) {
        for (int x = 0; x < Width; x++) {

            RGBQUAD rgb;
            for (int i = -R; i <= R; i++) {
                for (int j = -R; j <= R; j++) {
                    int ny = y + i, nx = x + j;
                    if (ny < 0 || ny >= Height || nx < 0 || nx >= Width)continue;
                    GetPixel(pBmpFileBuf, nx, ny, &rgb);
                    int res = min(((BYTE*)&rgb)[2], min((((BYTE*)&rgb)[1]), ((BYTE*)&rgb)[0]));
                    if (res < darkimage[id].first)darkimage[id].first = res;
                }
            }

            darkimage[id].second = std::make_pair(x, y);
            id++;
        }
    }

    std::sort(darkimage, darkimage + Height * Width, [](std::pair<int, std::pair<int, int>>& a, std::pair<int, std::pair<int, int>>& b) {return a.first > b.first; });

    // 求解大气光矢量
    int maxn = int(0.1 * Height * Width);
    int Ar = 0, Ag = 0, Ab = 0;
    int max_brightness = 0;
    RGBQUAD rgb;
    for (int i = 0; i < maxn; i++) {
        GetPixel(pBmpFileBuf, darkimage[i].second.first, darkimage[i].second.second, &rgb);
        double brightness = 1.0 * ((((BYTE*)&rgb)[2] * 299) + (((BYTE*)&rgb)[1] * 587) + (((BYTE*)&rgb)[0] * 114)) / 1000.0;
        if ((int)brightness > max_brightness) {
            max_brightness = (int)brightness;
            Ar = ((BYTE*)&rgb)[2];
            Ag = ((BYTE*)&rgb)[1];
```

```
                Ab = ((BYTE*)&rgb)[0];
            }
        }

        // 计算每个像素处的透射率并求解无雾图像
        std::sort(darkimage, darkimage + Height * Width, [=](std::pair<int, std::pair<int, int>>& a, std::pair<int, std::pair<int, int>>& b)
            {return a.second.second * Width + a.second.first < b.second.second * Width + b.second.first; });
        id = 0;
        double t0 = 1e-6;
        for (int y = 0; y < Height; y++) {
            for (int x = 0; x < Width; x++) {

                // 计算透射率
                double t = 1.0;
                for (int i = -R; i <= R; i++) {
                    for (int j = -R; j <= R; j++) {
                        int ny = y + i, nx = x + j;
                        if (ny < 0 || ny >= Height || nx < 0 || nx >= Width)continue;
                        GetPixel(pBmpFileBuf, nx, ny, &rgb);
                        double res = min(1.0 * ((BYTE*)&rgb)[2] / Ar, min((1.0 * ((BYTE*)&rgb)[1]) / Ag, 1.0 * ((BYTE*)&rgb)[0] / Ab));
                        t = min(t, w * res);
                    }
                }
                t = 1.0 - t;
                GetPixel(pBmpFileBuf, x, y, &rgb);

                // 求解无雾图像
                int r = max(0, (int)(1.0 * (((BYTE*)&rgb)[2] - Ar) / max(t, t0) + Ar));
                r = min(r, 255);
                int g = max(0, (int)(1.0 * (((BYTE*)&rgb)[1] - Ag) / max(t, t0) + Ag));
                g = min(g, 255);
                int b = max(0, (int)(1.0 * (((BYTE*)&rgb)[0] - Ab) / max(t, t0) + Ab));
                b = min(b, 255);

                ((BYTE*)&rgb)[2] = r;
                ((BYTE*)&rgb)[1] = g;
                ((BYTE*)&rgb)[0] = b;
                SetPixel(pNewBmpFileBuf, x, y, rgb);

                id++;
            }
        }

        return pNewBmpFileBuf;
}
```

（19）CBilateralfilter.cpp

```
// CBilateralfilter.cpp: 实现文件
//
```

```
#include "framework.h"
#include "imageProcessing.h"
#include "afxdialogex.h"
#include "CBilateralfilter.h"


// CBilateralfilter 对话框

IMPLEMENT_DYNAMIC(CBilateralfilter, CDialogEx)

CBilateralfilter::CBilateralfilter(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CBilateralfilter, pParent)
	, N1(_T(""))
	, N2(_T(""))
	, sigma_d(_T(""))
	, sigma_R(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CBilateralfilter::~CBilateralfilter()
{
}

void CBilateralfilter::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_N1, N1_Edit);
	DDX_Control(pDX, IDC_N2, N2_Edit);
	DDX_Control(pDX, IDC_sigma_d, sigma_d_Edit);
	DDX_Control(pDX, IDC_sigma_R, sigma_R_Edit);
	DDX_Text(pDX, IDC_N1, N1);
	DDX_Text(pDX, IDC_N2, N2);
	DDX_Text(pDX, IDC_sigma_d, sigma_d);
	DDX_Text(pDX, IDC_sigma_R, sigma_R);
}


BEGIN_MESSAGE_MAP(CBilateralfilter, CDialogEx)
END_MESSAGE_MAP()


// CBilateralfilter 消息处理程序


INT_PTR CBilateralfilter::DoModal()
{
```

```
	// TODO:  在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（20）CCannyedge.cpp

```
// CCannyedge.cpp:  实现文件
//

#include "framework.h"
#include "imageProcessing.h"
#include "afxdialogex.h"
#include "CCannyedge.h"


// CCannyedge  对话框

IMPLEMENT_DYNAMIC(CCannyedge, CDialogEx)

CCannyedge::CCannyedge(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CCannyedge, pParent)
	, sigma(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CCannyedge::~CCannyedge()
{
}

void CCannyedge::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_sigma, sigma_Edit);
	DDX_Text(pDX, IDC_sigma, sigma);
}


BEGIN_MESSAGE_MAP(CCannyedge, CDialogEx)
END_MESSAGE_MAP()


// CCannyedge  消息处理程序


INT_PTR CCannyedge::DoModal()
{
	// TODO:  在此添加专用代码和/或调用基类
```

```
	return CDialogEx::DoModal();
}
```

（21）CGausssmooth.cpp

```
// CGausssmooth.cpp: 实现文件
//

#include "framework.h"
#include "imageProcessing.h"
#include "afxdialogex.h"
#include "CGausssmooth.h"


// CGausssmooth 对话框

IMPLEMENT_DYNAMIC(CGausssmooth, CDialogEx)

CGausssmooth::CGausssmooth(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CGausssmooth, pParent)
	, Gauss(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CGausssmooth::~CGausssmooth()
{
}

void CGausssmooth::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_Gauss, Gauss_Edit);
	DDX_Text(pDX, IDC_Gauss, Gauss);
}


BEGIN_MESSAGE_MAP(CGausssmooth, CDialogEx)
END_MESSAGE_MAP()


// CGausssmooth 消息处理程序
```

（22）CHazeremoval.cpp

```
// CHazeremoval.cpp: 实现文件
//

#include "framework.h"
#include "imageProcessing.h"
```

```
#include "afxdialogex.h"
#include "CHazeremoval.h"


// CHazeremoval 对话框

IMPLEMENT_DYNAMIC(CHazeremoval, CDialogEx)

CHazeremoval::CHazeremoval(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CHazeremoval, pParent)
	, w(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CHazeremoval::~CHazeremoval()
{
}

void CHazeremoval::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_w, w_Edit);
	DDX_Text(pDX, IDC_w, w);
}


BEGIN_MESSAGE_MAP(CHazeremoval, CDialogEx)
END_MESSAGE_MAP()


// CHazeremoval 消息处理程序


INT_PTR CHazeremoval::DoModal()
{
	// TODO: 在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（23）CInputXY.cpp

```
// CInputXY.cpp: 实现文件
//

#include "framework.h"
#include "imageProcessing.h"
#include "afxdialogex.h"
```

```
#include "CInputXY.h"


// CInputXY 对话框

IMPLEMENT_DYNAMIC(CInputXY, CDialogEx)

CInputXY::CInputXY(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CInputXY, pParent)
	, X_Coord(_T(""))
	, Y_Coord(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CInputXY::~CInputXY()
{
}

void CInputXY::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_X, X);
	DDX_Control(pDX, IDC_Y, Y);
	DDX_Text(pDX, IDC_X, X_Coord);
	DDX_Text(pDX, IDC_Y, Y_Coord);
}


BEGIN_MESSAGE_MAP(CInputXY, CDialogEx)
END_MESSAGE_MAP()


// CInputXY 消息处理程序


INT_PTR CInputXY::DoModal()
{
	// TODO: 在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（24）CInputXYRGB.cpp

```
// CInputXYRGB.cpp: 实现文件
//

#include "framework.h"
```

```
#include "imageProcessing.h"
#include "afxdialogex.h"
#include "CInputXYRGB.h"


// CInputXYRGB 对话框

IMPLEMENT_DYNAMIC(CInputXYRGB, CDialogEx)

CInputXYRGB::CInputXYRGB(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CInputXYRGB, pParent)
	, X_Coord(_T(""))
	, Y_Coord(_T(""))
	, R(_T(""))
	, G(_T(""))
	, B(_T(""))
	, Reserved(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CInputXYRGB::~CInputXYRGB()
{
}

void CInputXYRGB::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_X, X);
	DDX_Control(pDX, IDC_Y, Y);
	DDX_Control(pDX, IDC_R, R_Edit);
	DDX_Control(pDX, IDC_G, G_Edit);
	DDX_Control(pDX, IDC_B, B_Edit);
	DDX_Control(pDX, IDC_Reserved, Reserved_Edit);
	DDX_Text(pDX, IDC_X, X_Coord);
	DDX_Text(pDX, IDC_Y, Y_Coord);
	DDX_Text(pDX, IDC_R, R);
	DDX_Text(pDX, IDC_G, G);
	DDX_Text(pDX, IDC_B, B);
	DDX_Text(pDX, IDC_Reserved, Reserved);
}


BEGIN_MESSAGE_MAP(CInputXYRGB, CDialogEx)
END_MESSAGE_MAP()


// CInputXYRGB 消息处理程序
```

```
INT_PTR CInputXYRGB::DoModal()
{
	// TODO: 在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（25）CInterpolation.cpp

```
// CInterpolation.cpp: 实现文件
//

#include "framework.h"
#include "imageProcessing.h"
#include "afxdialogex.h"
#include "CInterpolation.h"


// CInterpolation 对话框

IMPLEMENT_DYNAMIC(CInterpolation, CDialogEx)

CInterpolation::CInterpolation(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CInterpolation, pParent)
	, Method(_T(""))
	, factorX(_T(""))
	, factorY(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CInterpolation::~CInterpolation()
{
}

void CInterpolation::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_Method, Method_Edit);
	DDX_Control(pDX, IDC_factorX, factorX_Edit);
	DDX_Control(pDX, IDC_factorY, factorY_Edit);
	DDX_Text(pDX, IDC_Method, Method);
	DDX_Text(pDX, IDC_factorX, factorX);
	DDX_Text(pDX, IDC_factorY, factorY);
}


BEGIN_MESSAGE_MAP(CInterpolation, CDialogEx)
```

```
END_MESSAGE_MAP()


// CInterpolation 消息处理程序


INT_PTR CInterpolation::DoModal()
{
	// TODO: 在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（26）CMedianfilter.cpp

```
// CMedianfilter.cpp: 实现文件
//

#include "framework.h"
#include "imageProcessing.h"
#include "afxdialogex.h"
#include "CMedianfilter.h"


// CMedianfilter 对话框

IMPLEMENT_DYNAMIC(CMedianfilter, CDialogEx)

CMedianfilter::CMedianfilter(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CMedianfilter, pParent)
	, N1(_T(""))
	, N2(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CMedianfilter::~CMedianfilter()
{
}

void CMedianfilter::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_N1, N1_Edit);
	DDX_Control(pDX, IDC_N2, N2_Edit);
	DDX_Text(pDX, IDC_N1, N1);
	DDX_Text(pDX, IDC_N2, N2);
}
```

```
BEGIN_MESSAGE_MAP(CMedianfilter, CDialogEx)
END_MESSAGE_MAP()


// CMedianfilter 消息处理程序


INT_PTR CMedianfilter::DoModal()
{
	// TODO: 在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（27）CSharpengrad.cpp

```
// CSharpengrad.cpp: 实现文件
//

#include "framework.h"
#include "imageProcessing.h"
#include "afxdialogex.h"
#include "CSharpengrad.h"


// CSharpengrad 对话框

IMPLEMENT_DYNAMIC(CSharpengrad, CDialogEx)

CSharpengrad::CSharpengrad(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_CSharpengrad, pParent)
	, k1(_T(""))
	, k2(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CSharpengrad::~CSharpengrad()
{
}

void CSharpengrad::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_k1, k1_Edit);
	DDX_Control(pDX, IDC_k2, k2_Edit);
	DDX_Text(pDX, IDC_k1, k1);
	DDX_Text(pDX, IDC_k2, k2);
```

```
}


BEGIN_MESSAGE_MAP(CSharpengrad, CDialogEx)
END_MESSAGE_MAP()


// CSharpengrad 消息处理程序


INT_PTR CSharpengrad::DoModal()
{
	// TODO: 在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（28）imageProcessing.cpp

```
// imageProcessing.cpp: 定义应用程序的类行为。
//
#include "framework.h"
#include "imageProcessing.h"
#include "MainFrm.h"
#include "imageProcessingDoc.h"
#include "imageProcessingView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

BEGIN_MESSAGE_MAP(CimageProcessingApp, CWinApp)
	ON_COMMAND(ID_APP_ABOUT, &CimageProcessingApp::OnAppAbout)
	// 基于文件的标准文档命令
	ON_COMMAND(ID_FILE_NEW, &CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, &CWinApp::OnFileOpen)
END_MESSAGE_MAP()

CimageProcessingApp::CimageProcessingApp() noexcept
{
	// TODO: 将以下应用程序 ID 字符串替换为唯一的 ID 字符串；建议的字符串格式
	//为 CompanyName.ProductName.SubProduct.VersionInformation
	SetAppID(_T("imageProcessing.AppID.NoVersion"));

	// TODO: 在此处添加构造代码，
	// 将所有重要的初始化放置在 InitInstance 中
}

CimageProcessingApp theApp;

BOOL CimageProcessingApp::InitInstance()
```

```
{
	AfxOleInit();
	CWinApp::InitInstance();
	EnableTaskbarInteraction(FALSE);
	// 使用 RichEdit 控件需要 AfxInitRichEdit2()
	// AfxInitRichEdit2();
	// 标准初始化
	// 如果未使用这些功能并希望减小
	// 最终可执行文件的大小，则应移除下列
	// 不需要的特定初始化例程
	// 更改用于存储设置的注册表项
	// TODO: 应适当修改该字符串，
	// 例如修改为公司或组织名
	SetRegistryKey(_T("应用程序向导生成的本地应用程序"));
	LoadStdProfileSettings(4);  // 加载标准 INI 文件选项(包括 MRU)
	// 注册应用程序的文档模板。  文档模板
	// 将用作文档、框架窗口和视图之间的连接
	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CimageProcessingDoc),
		RUNTIME_CLASS(CMainFrame),       // 主 SDI 框架窗口
		RUNTIME_CLASS(CimageProcessingView));
	if (!pDocTemplate)
		return FALSE;
	AddDocTemplate(pDocTemplate);
	// 分析标准 shell 命令、DDE、打开文件操作的命令行
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);
	// 调度在命令行中指定的命令。  如果
	// 用 /RegServer、/Register、/Unregserver 或 /Unregister 启动应用程序，则返回 FALSE。
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;
	// 唯一的一个窗口已初始化，因此显示它并对其进行更新
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();
	return TRUE;
}

// 用于应用程序“关于”菜单项的 CAboutDlg 对话框
class CAboutDlg : public CDialogEx
{
public:
	CAboutDlg() noexcept;

// 对话框数据
```

```
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_ABOUTBOX };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

// 实现
protected:
	DECLARE_MESSAGE_MAP()
public:
	virtual INT_PTR DoModal();
};

CAboutDlg::CAboutDlg() noexcept : CDialogEx(IDD_ABOUTBOX)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// 用于运行对话框的应用程序命令
void CimageProcessingApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}


INT_PTR CAboutDlg::DoModal()
{
	// TODO: 在此添加专用代码和/或调用基类

	return CDialogEx::DoModal();
}
```

（29）imageProcessingDoc.cpp

```
// imageProcessingDoc.cpp: CimageProcessingDoc 类的实现
//
#include "framework.h"
#include "imageProcessing.h"
#include "imageProcessingDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
IMPLEMENT_DYNCREATE(CimageProcessingDoc, CDocument)

BEGIN_MESSAGE_MAP(CimageProcessingDoc, CDocument)
END_MESSAGE_MAP()

CimageProcessingDoc::CimageProcessingDoc() noexcept
{
}

CimageProcessingDoc::~CimageProcessingDoc()
{
}

BOOL CimageProcessingDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;

	// TODO: 在此添加重新初始化代码
	// (SDI 文档将重用该文档)

	return TRUE;
}

void CimageProcessingDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO:  在此添加存储代码
	}
	else
	{
		// TODO:  在此添加加载代码
	}
}

#ifdef _DEBUG
void CimageProcessingDoc::AssertValid() const
{
	CDocument::AssertValid();
}

void CimageProcessingDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG
```

（30）imageProcessingView.cpp

```
// imageProcessingView.cpp: CimageProcessingView 类的实现
```

```
//
#define _CRT_SECURE_NO_WARNINGS
#include "framework.h"
#include "imageProcessing.h"
#include "imageProcessingDoc.h"
#include "imageProcessingView.h"
#include "_GlobalCommon.h"

#include "CInputXY.h"
#include "CInputXYRGB.h"
#include "CInterpolation.h"
#include "CGausssmooth.h"
#include "CMedianfilter.h"
#include "CBilateralfilter.h"
#include "CSharpengrad.h"
#include "CCannyedge.h"
#include "CHazeremoval.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

IMPLEMENT_DYNCREATE(CimageProcessingView, CView)

BEGIN_MESSAGE_MAP(CimageProcessingView, CView)
	ON_COMMAND(ID_IMAGEPROCESS_OPENBMPFILE,
&CimageProcessingView::OnImageprocessOpenbmpfile)
	ON_COMMAND(ID_IMAGEPROCESS_SAVETOFILE, &CimageProcessingView::OnImageprocessSavetofile)
	ON_COMMAND(ID_IMAGEPROCESS_DISPLAYFILEHEADER,
&CimageProcessingView::OnImageprocessDisplayfileheader)
	ON_COMMAND(ID_IMAGEPROCESS_DISPLAYPALETTE,
&CimageProcessingView::OnImageprocessDisplaypalette)
	ON_COMMAND(ID_IMAGEPROCESS_GETPIXELVALUE,
&CimageProcessingView::OnImageprocessGetpixelvalue)
	ON_COMMAND(ID_IMAGEPROCESS_SETPIXELVALUE,
&CimageProcessingView::OnImageprocessSetpixelvalue)
	ON_COMMAND(ID_IMAGEPROCESS_INERPOLATION,
&CimageProcessingView::OnImageprocessInterpolation)
	ON_COMMAND(ID_IMAGEPROCESS_GAUSSSMOOTH,
&CimageProcessingView::OnImageprocessGausssmooth)
	ON_COMMAND(ID_IMAGEPROCESS_MEDIANFILTER,
&CimageProcessingView::OnImageprocessMedianfilter)
	ON_COMMAND(ID_IMAGEPROCESS_BILATERALFILTER,
&CimageProcessingView::OnImageprocessBilateralfilter)
	ON_COMMAND(ID_IMAGEPROCESS_HISTOEQUALIZATION,
&CimageProcessingView::OnImageprocessHistoequalization)
	ON_COMMAND(ID_IMAGEPROCESS_SHARPENGRAD,
&CimageProcessingView::OnImageprocessSharpengrad)
	ON_COMMAND(ID_IMAGEPROCESS_CANNYEDGE, &CimageProcessingView::OnImageprocessCannyedge)
	ON_COMMAND(ID_IMAGEPROCESS_OTSUSEGMENT,
&CimageProcessingView::OnImageprocessOtsusegment)
```

```
    ON_COMMAND(ID_IMAGEPROCESS_HAZEREMOVAL,
&CimageProcessingView::OnImageprocessHazeremoval)
END_MESSAGE_MAP()

CimageProcessingView::CimageProcessingView() noexcept
{
    pFileBuf = NULL;
}

CimageProcessingView::~CimageProcessingView()
{
    if( pFileBuf )
    {
        delete [] pFileBuf;
        pFileBuf = 0;
    }
}

BOOL CimageProcessingView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: 在此处通过修改
    //  CREATESTRUCT cs 来修改窗口类或样式

    return CView::PreCreateWindow(cs);
}

#ifdef _DEBUG
void CimageProcessingView::AssertValid() const
{
    CView::AssertValid();
}

void CimageProcessingView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CimageProcessingDoc *CimageProcessingView::GetDocument() const // 非调试版本是内联的
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CimageProcessingDoc)));
    return (CimageProcessingDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////

// CimageProcessingView 绘图
void CimageProcessingView::OnDraw(CDC *pDC)
{
```

```
    CimageProcessingDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: 在此处为本机数据添加绘制代码
    if( pFileBuf != NULL )
    {
        DisplayImage(pDC,pFileBuf,10,10,0,0,0);
    }
}

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
//Here are the functions to be programmed by you!

//Open a BMP file
void CimageProcessingView::OnImageprocessOpenbmpfile()
{
    LPCTSTR lpszFilter = "BMP Files (*.bmp)|*.bmp||";
    CFileDialog dlg(TRUE,NULL,NULL,OFN_NOCHANGEDIR,lpszFilter,NULL);
    if( dlg.DoModal() != IDOK ) return;
    if( pFileBuf != NULL )
    {
        delete [] pFileBuf;
    }
    pFileBuf = OpenBMPfile( dlg.GetPathName() );
    Invalidate();
    UpdateWindow();
}

//Save to a new BMP file
void CimageProcessingView::OnImageprocessSavetofile()
{
    if(pFileBuf == NULL) return;
    LPCTSTR lpszFilter = "BMP Files (*.bmp)|*.bmp||";
    CFileDialog dlg(FALSE, NULL, NULL, OFN_NOCHANGEDIR, lpszFilter, NULL);
    if (dlg.DoModal() != IDOK) return;
    CString strBmpFile = dlg.GetPathName();
    SaveDIB(pFileBuf, strBmpFile);
}

//Display BMP file header
void CimageProcessingView::OnImageprocessDisplayfileheader()
{
    if(pFileBuf == NULL) return;
    DisplayHeaderMessage(pFileBuf);
}

//Display Pallete
void CimageProcessingView::OnImageprocessDisplaypalette()
```

```
{
	if(pFileBuf == NULL) return;
	int num = 0;
	RGBQUAD *palette = GetDIBPaletteData(pFileBuf,&num);
	if( palette == NULL )
	{
		AfxMessageBox("No palette");
	}
	else
	{
		LPCTSTR lpszFilter = "TXT Files (*.txt)|*.txt||";
		CFileDialog dlg(FALSE, NULL, NULL, OFN_NOCHANGEDIR, lpszFilter, NULL);
		if (dlg.DoModal() != IDOK) return;
		CString FilePath = dlg.GetPathName();
		CFile hFile;
		if (!hFile.Open(FilePath, CFile::modeCreate | CFile::modeWrite))
		{
			AfxMessageBox("Failed to create the TXT file");
			return;
		}
		char msg_buff[100];
		for (int i = 0; i < num; i++) {
			memset(msg_buff, 0, sizeof(msg_buff));
			sprintf(msg_buff, "Palette[%d]:rgbBlue=%hhx,rgbGreen=%hhx,rgbRed=%hhx,rgbReserved=%hhx\n"
				, i, (palette + i)->rgbBlue, (palette + i)->rgbGreen, (palette + i)->rgbRed, (palette +
i)->rgbReserved);
			hFile.Write(msg_buff, (UINT)strlen(msg_buff));
		}
		hFile.Close();
	}
}

//Get pixel value
void CimageProcessingView::OnImageprocessGetpixelvalue()
{
	if (pFileBuf == NULL) return;
	CInputXY inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	int x = atoi(inputDlg.X_Coord);
	int y = atoi(inputDlg.Y_Coord);
	RGBQUAD rgb;
	bool bGray;
	GetPixel(pFileBuf,x,y,&rgb,&bGray);
	char buf[100];
	if( bGray )
		sprintf(buf, "(%d,%d) = %d",x,y,rgb.rgbReserved);
	else
		sprintf(buf, "(%d,%d) = (%d,%d,%d)",x,y,rgb.rgbRed,rgb.rgbGreen,rgb.rgbBlue);
	AfxMessageBox( buf );
}
```

```
//Set pixel value
void CimageProcessingView::OnImageprocessSetpixelvalue()
{
	if(pFileBuf == NULL) return;
	CInputXYRGB inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	int x = atoi(inputDlg.X_Coord);
	int y = atoi(inputDlg.Y_Coord);
	RGBQUAD rgb;
	rgb.rgbReserved = atoi(inputDlg.Reserved);
	rgb.rgbRed      = atoi(inputDlg.R);
	rgb.rgbGreen    = atoi(inputDlg.G);
	rgb.rgbBlue     = atoi(inputDlg.B);
	SetPixel(pFileBuf,x,y,rgb);
	Invalidate();
	UpdateWindow();
}

//Image interpolaion
void CimageProcessingView::OnImageprocessInterpolation()
{
	if(pFileBuf == NULL) return;
	CInterpolation inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	BITMAPINFOHEADER* pDIBInfo = (BITMAPINFOHEADER*)(pFileBuf + sizeof(BITMAPFILEHEADER));
	int orgWidth = pDIBInfo->biWidth;
	int orgHeight = pDIBInfo->biHeight;
	int newWidth  = int(atof(inputDlg.factorX) * orgWidth);
	int newHeight = int(atof(inputDlg.factorY) * orgHeight);
	char* pNewImage = ImageInterpolation(pFileBuf, newWidth, newHeight, atoi(inputDlg.Method));
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Gaussian smoothing
void CimageProcessingView::OnImageprocessGausssmooth()
{
	if (pFileBuf == NULL)return;
	CGausssmooth inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	int sigma = atoi(inputDlg.Gauss);
	char* pNewImage = ImageGausssmooth(pFileBuf, sigma);
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Median filtering
```

```
void CimageProcessingView::OnImageprocessMedianfilter()
{
	if (pFileBuf == NULL)return;
	CMedianfilter inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	int N1 = atoi(inputDlg.N1);
	int N2 = atoi(inputDlg.N2);
	char* pNewImage = ImageMedianfilter(pFileBuf, N1, N2);
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Bilateral filtering
void CimageProcessingView::OnImageprocessBilateralfilter()
{
	if (pFileBuf == NULL)return;
	CBilateralfilter inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	int N1 = atoi(inputDlg.N1);
	int N2 = atoi(inputDlg.N2);
	int sigma_d = atoi(inputDlg.sigma_d);
	int sigma_R = atoi(inputDlg.sigma_R);
	char* pNewImage = ImageBilateralfilter(pFileBuf, N1, N2, sigma_d, sigma_R);
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Histogram equalization
void CimageProcessingView::OnImageprocessHistoequalization()
{
	if (pFileBuf == NULL)return;
	char* pNewImage = ImageHistoequalization(pFileBuf);
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Sharpening by gradient
void CimageProcessingView::OnImageprocessSharpengrad()
{
	if (pFileBuf == NULL)return;
	CSharpengrad inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	double k1 = atof(inputDlg.k1);
	double k2 = atof(inputDlg.k2);
	char* pNewImage = ImageSharpengrad(pFileBuf, k1, k2);
```

```
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Canny edge detection
void CimageProcessingView::OnImageprocessCannyedge()
{
	if (pFileBuf == NULL)return;
	CCannyedge inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	int sigma = atoi(inputDlg.sigma);
	char* pTmp1Image = ImageGausssmooth(pFileBuf, sigma);
	delete[] pFileBuf;
	pFileBuf = pTmp1Image;
	char* pTmp2Image = ImageCannyedgeStep1(pFileBuf);
	delete[] pFileBuf;
	pFileBuf = pTmp2Image;
	int threshold = 0;
	char* pTmp3Image = ImageOtsusegment(pFileBuf, threshold);
	delete[] pTmp3Image;
	char* pNewImage = ImageCannyedgeStep2(pFileBuf, threshold, threshold >> 1);
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Otsu segmentation
void CimageProcessingView::OnImageprocessOtsusegment()
{
	if (pFileBuf == NULL)return;
	int threshold = 0;
	char* pNewImage = ImageOtsusegment(pFileBuf, threshold);
	char msg_buff[50];
	sprintf(msg_buff, "threshold = %d\n", threshold);
	AfxMessageBox(msg_buff);
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}

//Haze removal
void CimageProcessingView::OnImageprocessHazeremoval()
{
	if (pFileBuf == NULL)return;
	CHazeremoval inputDlg(NULL);
	if (inputDlg.DoModal() != IDOK)return;
	double w = atof(inputDlg.w);
```

```
	char* pNewImage = ImageHazeremoval(pFileBuf, w);
	delete[] pFileBuf;
	pFileBuf = pNewImage;
	Invalidate();
	UpdateWindow();
}
```

（31）MainFrm.cpp

```
// MainFrm.cpp: CMainFrame 类的实现
//
#include "framework.h"
#include "imageProcessing.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

CMainFrame::CMainFrame() noexcept
{
	// TODO: 在此添加成员初始化代码
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	if( !CFrameWnd::PreCreateWindow(cs) )
		return FALSE;
	// TODO: 在此处通过修改
	//  CREATESTRUCT cs 来修改窗口类或样式

	return TRUE;
}

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CFrameWnd::Dump(dc);
```

```
}
#endif //_DEBUG
```

4 总结与体会

完成了本次数字图像处理实验后,我从编程的角度更加深入理解了数字图像处理中的各种算法以及相关文件结构，体会到了数字图像处理这门学科的乐趣与魅力。在实验过程中我也遇到了许多困难，如接口的调用方法、算法正确性的推导或证明等等。对此，我充分利用了网络上的一些资源，并结合自己的思考克服了这些问题，提升了自己的综合能力。

本次实验中涉及的各种算法在实际应用中也有着广泛的用途。比如，图像缩放和插值在数字摄影和打印技术中是必不可少的；直方图均衡化可以增强图像的对比度和视觉效果，在图像分析和医学影像中有着重要作用；边缘检测和分割则是图像识别和计算机视觉领域的核心技术之一。在完成本次实验后，我对上述这些算法更加熟悉，这为我以后在相关领域做出贡献打下了基础。

总而言之，这次实验让我更全面地了解了数字图像处理领域的知识和技术，并培养了我在图像处理方面的实践能力和创新思维。感谢老师在教学中的指导和支持，我将继续努力学习，探索更多图像处理的应用和前沿技术。