**РЕФЕРАТ**

Розробка програмного забезпечення для фільтрації растрових зображень. Дипломний проект з дисципліни «Технології програмування ГКС». Пояснювальна записка - Харків: ХНУРЕ. - 2009. – 62 с.

Предмет дослiдження: растровi зображення.

Мета проекту – систематизацiя, поглиблення i застосування знань по технологiям программування, отриманих на лекцiйних та лабораторних заняттях.

Метод дослiдження – вивчення лiтератури, написання i тестування програми на комп’ютерi.

Програма призначена для фільтрації растрових зображень.

В роботi розглядаються питання розробки програмного забезпечення для фільтрації растрових зображень. ПЗ виконує корекцію кольорів відкритих растрових зображень за допомогою фiльтрiв. Графiчнi фiльтри реалiзованi на основi точечних i просторових(матричних) перетворень.

Програмне забезпечення реалізовано у системі програмування Microsoft Visual С++ 6.0.

Ключові слова:

Растрове зображення, DDB, DIB, BMP, матриці перетворень, MDI-інтерфейс, графічні фільтри

**The abstract**

Development of the program maintenance for filtration of raster’s images. The course project on discipline “The Technology of Programming the FKS”. An explanatory note - Kharkiv:KNURE.-2009.- 62 р.

Object of research: raster images.

The purpose of the course project – the ordering, deepening and application of the knowledge on technologies of the programming, received on lectures and laboratory researches.

The method of research – studying the literature, writing and testing the program on the computer.

The program is assigned for filtration of raster’s images.

In this work, the questions of development of a program maintenance for filtration of raster’s images are considered. The PM realizes the correction of a color of opened raster images, with the help of a filters. Graphic filters are realized on the basis of a dot and spatial (matrix) transformations.

The program maintenance is realized in the system of programming Microsoft Visual C ++ 6.0.

Key words:

The raster image, DDB, DIB, BMP, matrixes of transformations, the MDI-interface, graphic filters.

**ВВЕДЕНИЕ**

Компьютерная графика объединяет довольно широкий круг операций по обработке графической информации с помощью компьютера. Компьютерной графике отводится огромная роль в современном мире.

В компьютерной графике можно выделить несколько основных направлений:

- Визуализация научных (расчетных или экспериментальных) данных. Большинство современных математических программных пакетов (например, Мар1е, Маtlab, МаthCAD) имеют средства для отображения графиков, поверхностей и трехмерных тел, построенных на основе каких-либо расчетов. Кроме того, графическая информация может активно использоваться в самом процессе вычислений.

- Геометрическое проектирование и моделирование. Это направление компьютерной графики связано с решением задач начертательной геометрии - построением чертежей, эскизов, объемных изображений с помощью программных систем, получивших название САD-системы, например АutoCAD. Существует большое количество специализированных САD-систем в машиностроении, архитектуре и т. д.

*-* Распознавание образов. Способность распознавать абстрактные образы считают одним из важнейших факторов, определившим развитие мыслительных способностей человека. Задача распознавания и классификации графической информации является одной из ключевых и при создании искусственного интеллекта. Уже в наши дни компьютеры распознают образы повсеместно (анализ аэро- и космических фотоснимков; системы сортировки, наведения и т. д.). Возможно, самый известный пример распознавания образов - сканирование и перевод "фотографии" текста в набор отдельных символов, формирующих слова. Такую операцию позволяет выполнить программное обеспечение многих современных сканеров.

- Изобразительное искусство. К этому направлению можно отнести разнообразную графическую рекламу: от текстовых транспарантов и фирменных знаков до компьютерных видеофильмов, обработку фотографий, создание рисунков, мультипликацию и т. д. В качестве примера популярных и довольно востребованных программ из этой области компьютерной графики можно назвать АdobePhotoshop (обработка растровых изображений), СогеlDraw (создание векторной графики), 3DS Мах (трехмерное моделирование).

- Виртуальная реальность. Реальность, даже виртуальная, подразумевает воздействия на всю совокупность органов чувств человека, в первую очередь на его зрение. К компьютерной графике можно отнести задачи моделирования внешнего мира в различных приложениях: от компьютерных игр до тренажеров.

- Цифровое видео. Все более широкое распространение получают аними-рованные изображения, записанные в цифровом формате. Это фильмы, передаваемые через компьютерные сети, цифровое, кабельное и спутниковое телевидение.

# **1. АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ**

# В данной работе разрабатывается ПО для фильтрации растровых изображений. Графический формат BMP является аппаратно-независимым (devise-independent bitmap DIB) и наиболее удобным для данного вопроса. Для разработки программ работы с битовыми изображениями необходимо разработать свой собственный класс, поддерживающий файлы \*.bmp. Кроме открытия растровых изображений программа должна выполнять цветокоррекцию открытых изображений. Цветокоррекция выполняется при помощи графических фильтров.

# Для реализации поставленной задачи в проекте целесообразно создать MDI приложение (с многодокументным интерфейсом). В разрабатываемом ПО реализуется возможность масштабирования открытого файла.

# Для обработки изображений при помощи фильтров необходимо предусмотреть, как они будут уживаться между собой и взаимодействовать с остальными модулями программы. Фильтры можно реализовать в виде классов, производных от какого-то одного базового класса. В базовом классе следует определить набор методов, общих для всех фильтров. Процесс преобразования изображения выносится в отдельный поток выполнения программы. Это дает возможность контролировать не только область применения фильтра, но и продолжительность выполнения операции, т.е. возможность остановить выполнение преобразования.

# **2. ОПИСАНИЕ ГРАФИЧЕСКОГО ФОРМАТА**

**2.1 Общая информация о графическом формате**

Графический формат - порядок (структура), согласно которому данные, описывающие изображение, записаны в файле.

Графические данные обычно разделяются на два класса: векторные и растровые. Векторные данныеиспользуются для представления прямых, многоугольников, кривых и т. п., с помощью определенных в числовом виде базовых точек. Растровые данные представляют собой набор числовых значений, определяющих яркость и цвет отдельных пикселов.Пикселами (или пикселями - от английского pixel) называются минимальные элементы (цветные точки), из которых формируется растровое изображение.

Далее под растром будем понимать массив пикселов (массив числовых значений). Для обозначения массива пикселов часто используется термин bitmap (битовая карта). В bitmap каждому пикселу отводится определённое число битов (одинаковое для всех пикселов изображения). Это число называется битовой глубиной пиксела или цветовой глубиной изображения, т.к. от количества битов, отводимых на один пиксел, зависит количество цветов изображения. Наиболее часто используется цветовая глубина 1, 2, 4, 8, 15, 16, 24 и 32 бита.

Растровый форматиспользуется для хранения растровых данных. Файлы такого типа особенно хорошо подходят для хранения изображений реального мира, например оцифрованных фотографий. Растровые файлы содержат битовую карту изображения и ее спецификацию. Наиболее распространенные растровые форматы: ВМР, ТIFF, GIF, РСХ, JРЕС.

В данном проекте будет использоваться ВМР формат.

**2.2 Общее описание формата ВМР**

Microsoft Windows Bitmap (ВМР) - собственный растровый формат операционной системы Windows. Формат основан на внутренних структурах представления растровых данных Windows. Формат совершенствовалсяи развивался по мере появления новых версий Windows. Первоначально был очень простым, содержал лишь растровые данные и не поддерживал сжатие. Растровые данные представляли собой индексы в цветовой палитре, которая была фиксированной и определялась графической платой. Поэтому этот формат называют аппаратно-зависимым (Device Dependent Bitmap, DDB), он был ориентирован на графические платы для IВМ РС (СGА,EGA, НERCULES) и другие.

Развитием формата ВМР стало введение в него поддержки изменяемой цветовой палитры. Это позволило хранить информацию о цветах вместе с растровыми данными. Такое изменение формата позволило сделать хранимые изображения аппаратно-независимыми (Devise Independent Bitmap, DIB). Иногда аббревиатуру DIВ используют как синоним ВМР.

**2.3 Структура файла ВМР**

Файлы ОБВ исходного формата ВМР содержали два раздела: заголовок файла и растровые данные (рис. 2.3.1).

|  |
| --- |
| Заголовок файла |
| Растровые данные |

Рисунок 2.3.1 - Структура файла DDВ исходного формата ВМР

Файлы более поздних версий содержат четыре раздела: заголовок файла, информационный заголовок растра, палитру цветов и растровые данные (рис. 2.3.2).

|  |
| --- |
| Заголовок файла |
| Заголовок растра |
| Палитра цветов |
| Растровые данные |

Рисунок 2.3.2 **-** Структура ВМР-файла

Рассмотрим в деталях структуру данных файла формата ВМР версии 3.x, появившегося с операционной системой Microsoft Windows 3.x. Этот формат поддерживается большинством существующих в настоящее время приложений.

Все версии формата ВМР начинаются с 14-байтового заголовка-структуры.

Листинг 2.3.1 - Структура данных файла формата ВМР версии 3.x

BITMAPFILEHEADER :

Typedef struct tagBITMAPHEADER

{ WORD bfType; //тип файла, должен быть 4d42h («ВМ»)

DWORD bfSise //размер файла в байтах

WORD bfReserved1; //зарезервировано, должен быть 0

WORD bfReserved2; // зарезервировано, должен быть 0

DWORD bfOffBits; //смещение в байтах до начала растровых данных

} BITMAPFILEHEADER;

###### // За заголовком файла следует заголовок растра. Его длина составляет 40 байтов.

Typedef struct tagBITMAPINFOHEADER

{DWORD biSize; //размер этого заголовка в байтах

LONG biWidth ; //ширина изображения в пикселах

LONG biHeight; //высота изображения в пикселах

WORD biplanes; //количество цветовых плоскостей

WORD bibitCount; //количество битов на пиксел

DWORD biCompression; //используемые методы сжатия

DWORD biSizeImage; //размер растра в байтах

LONG biXPelsPerMeter; //вертикальное разрешение

LONG biYPelsPerMeter; //горизонтальное разрешение

DWORD biClrUsed; //количество цветов в изображении

DWORD biClrImportant; //минимальное количество «важных» цветов

} BITMAPINFOHEADER

//За заголовком растра может следовать палитра цветов, состоящая из //последовательности 4-байтовых структур RGBQUARD

Typedef struct \_RGBQUARD

{BYTE rgbBlue; //синяя составляющая

BYTE rgbGreen; // зелёная составляющая

BYTE rgbRed; //красная составляющая

BYTE rgReserved; //заполнитель(всегда 0)

}RGBQUARD;

//Структура BITMAPINFOHEADER и структуры RGBQUARD собираются в структуре BITMAPINFO:

Typedef struct tagBITMAPINFO

{BITMAPINFOHEADER bmiHeader;

RGBQUARD bmiColors[1];

} BITMAPINFO;

После структуры BITMAPINFO на расстоянии bfOffBits (поле структуры BITMAPFILEHEADER) от начала файла начинаются растровые данные. Растровые данные представляют собой индексы в палитре цветов (в случае если bibitCount равно 1, 4, 8) ***или*** реальные значения цветов пикселов (в случае если bibitCount равно 24). Если bibitCount равно 24, то каждый пиксел представляется тремя байтами: первый байт - интенсивность синего цвета, затем по байту на зеленый и красный цвет. Этот формат цвета называется RGB888 или RGB24.

Растровые данные, соответствующие одной строке пикселов изображения, вне зависимости от формата цвета должны быть выровнены на границу двойного слова DWORD, **т**. е. каждая строка пикселов должна описываться целым числом двойных слов. Например, строка из 5 пикселов по 24 бита (3 байта) на пиксел может быть описана 15 байтами, но длина строки растровых данных в формате ВМР должна быть 16 байтов. Последний байт будет служить лишь для целей выравнивания.

Формат ВМР версии 3.xимеет разновидность (для Windows NT), предназначенную для хранения растровых данных с пиксельной глубиной 16 и 32 битов. Этот формат имеет точно такую же структуру заголовка растра BITMAPINFOHEADER. Его длина составляет 40 байтов. Отличие заключается в том, что поле bibitCount может принимать значения 16 и 32.

При пиксельной глубине 16 битов для хранения цвета пиксела отводится два байта (слово - тип WORD), каждому компоненту цвета пиксела отводится по 5 битов (формат цвета RGВ555). Младшие 5 битов задают интенсивность синего цвета, затем по 5 битов на зеленый и красный цвет, старший бит в слове не используется.

При пиксельной глубине 32 бита для хранения цвета пиксела отводится 4 байта (двойное слово - тип DWORD). При этом на каждый компонент цвета отводится по 8 бит, так же как и при 24-битной глубине, а старший байт в DWORD не используется (формат цвета RGВ888).

Дополнительные возможности этой разновидности формата проявляются, если указать значение поля biCompression, равное 3. В этом случае вслед за структурой BITMAPINFOHEADER (на месте палитры цвета) следуют три поля DWORD: RedMask, GreenMask, BlueMask, которые задают битовые маски для компонентов цвета пиксела. Биты в этих масках обязательно должны быть смежными и не содержать перекрывающихся полей.

Для 16-битовых растровых данных часто применяют формат RGВ565, который задается следующей маской.

RedMask = 0хF8000000; // 1111 1000 0000 0000 0000 0000 0000 0000

GгееnМаsk = 0х07Е00000; // 0000 0111 1110 0000 0000 0000 0000 0000

В1uеМаsk = 0х001F0000; // 0000 0000 0001 1111 0000 0000 0000 0000

С помощью этой маски из значения WORD, задающего цвет пиксела, извлекается значение каждого цветового компонента. В формате RGВ565 красному и синему цветам отводится по 5 битов, а зеленому - 6 битов. Такое неравноправие обосновывают тем, что человеческий глаз более восприимчив к зеленому цвету, поэтому тщательная запись его градаций позволяет повысить качество изображения.

Для 32-битовых растровых данных используют формат RGB101010, определяющий по 10 битов на каждый цвет, который задается следующей маской.

RedMask = 0ХFFC00000; // 1111 1111 1100 0000 0000 0000 0000 0000

GгееnМаsk = 0х00ЗFF000; // 0000 0000 0011 1111 1111 0000 0000 0000

В1uеМаsk = 0х00000FFС; // 0000 0000 0000 0000 0000 1111 1111 1100

По сравнению с форматом RGВ888 такое представление позволяет описать большее количество цветов.

**2.4 Преобразования графических файлов**

Рассмотрим два вида преобразований:

*-* точечные- новое значение элемента изображения (пиксела) рассчитывается только на основе его старого значения;

*-* пространственные(матричные) - при расчете нового значения пиксела учитывается не только его старое значение, но также значения некоторой области пикселов вокруг него.

Точечные преобразования удобно выполнять с помощью таблиц преобразования*.* Таблица преобразования- это просто массив, заполненный какими-то значениями. Размер массива равен максимальному значению, которое может принимать преобразуемая величина. С помощью такой таблицы удобно и быстро заменять одно значение другим. Таблицы преобразований применимы, когда новое значение формируется из единственного старого значения, т. е. при точечномпреобразовании.

Рассмотрим, например, как можно инвертировать цвет картинки. Диапазон значений каждого 8-битного компонента цвета находится в пределах от 0 до 255. Создадим таблицу преобразования из 256 элементов и заполним ее значениями от 255 до 0 (рис. 2.4.1).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| индекс | 0 | 1 | 2 | 3 | … | 253 | 254 | 255 |
| значение | 255 | 254 | 253 | 252 | … | 2 | 1 | 0 |

Рисунок 2.4.1 - Таблица преобразования "инверсия"

После преобразования по формуле (V’*=* 255 - *V)* с использованием таблицы интенсивность 255 будет заменена на 0, 254 - на 1 и т. д. В случае такого простого преобразования, как инверсия цвета, использование таблицы может и не дать особого выигрыша по скорости, но если новое значение пиксела должно рассчитываться по более сложной формуле, чем в нашем случае*,* то выигрыш будет весьма заметен. Кроме того, использование таблиц позволяет использовать единообразный подход к осуществлению различных преобразований.

Пространственное преобразование заключается в нахождении свертки значений группы пикселов. Свертка вычисляется как сумма пиксельных значений, попавших в зону преобразования, помноженных на весовые коэффициенты. В качестве весовых коэффициентов выступают элементы матрицы преобразования. Значения элементов матрицы преобразования и определяют тип преобразования. Размер матрицы преобразования соответствует области пикселов, которые будут участвовать в преобразовании. Центральный элемент матрицы - весовой коэффициент преобразуемого пиксела *(х, у).* Поэтому матрицы преобразования, обычно имеют нечетный размер (например, 3x3 или 5x5 элементов). Например, свертка с помощью единичной матрицы соответствует преобразованию "размытие" (понижение четкости) изображения. Достигается такой эффект за счет усреднения значений группы пикселов, охваченной матрицей преобразования. Если значение пиксела *(х, у)* было выше среднего, оно уменьшится, если было ниже среднего, то увеличится. Однако это не означает, что все изображение станет монотонным, так как матрица преобразования движется по изображению вместе с координатами *(х, у),* средний уровень тоже изменяется.

**3. ПРОСМОТР И РЕДАКТИРОВАНИЕ РАСТРОВЫХ ИЗОБРАЖЕНИЙ**

**3.1 Создание многодокументного приложения**

С помощью генератора приложений АррWizard создаём каркас приложения для просмотра и редактирования рисунков.

- С помощью команды File/New/Projects/MFC AppWizard (exe) начнем создание приложения. Назовем проект ВМViewer.

- На первом шаге выберем тип приложения Multiple documents.

- Далее можно принять все установки по умолчанию.

**3.2 Класс CRaster для работы с растровыми изображениями**

Создаем в программе класс CRaster, отвечающий за загрузку и осуществляющий поддержку операций по обработке растрового изображения. Класс CRaster содержит два метода DrawBitmap, выполняющих вывод изображения на контекст устройства. Аргументы одного из методов позволяют задать положение и размеры выводимой области исходного изображения и определить область назначения. По умолчанию изображение выводится полностью в масштабе 1:1, однако с помощью аргументов этой функции можно и изменить масштаб. Второй метод позволяет просто указать позицию начала вывода и масштаб, в котором должно быть нарисовано изображение. Оба метода внутри используют мощную АРI-функцию StretchDIBits().

Режим масштабирования выбирается CDC-методом SetStretchBltMode():

int SetStretchBltMode(int nStretchMode);

Аргумент функции - nStretchMode - режим масштабирования. Поддерживаются следующие режимы масштабирования:

- BLACKONWHITE - выполняет булеву операцию AND между цветом существующих и удаленных пикселов (при уменьшении размера изображения). Этот режим используется, если масштабируется рисунок "черным по белому", т. е. алгоритм масштабирования будет стараться сохранить черные пикселы;

- COLORNCOLOR - этот режим удаляет (добавляет) строки (столбцы) пикселов без каких-либо попыток сохранить содержащуюся в них информацию. Наиболее быстрый режим. Используется, когда необходимо сохранить цвета изображения неизменными;

*-* WHITEONBLACK - выполняет булеву операцию OR. Этот режим используется, если масштабируется рисунок "белым по черному";

- HALFTONE - преобразует изображение к заданному размеру и при этом трансформирует цвета так, чтобы средний цвет полученной картинки приближался к исходному цвету. Наиболее медленный режим. Однако масштабированная картинка выглядит лучше за счет сглаживания "лестничного эффекта". Этот режим не работает в Windows 95/98, и, похоже, заменяется режимом COLORNCOLOR.

При масштабировании фотографий и цветных рисунков в большинстве случаев наиболее подходящим является режимы COLORNCOLOR и HALFTONE .

Чтобы можно было изменять масштаб вывода изображений, добавим в меню View программы команды Zoom In, Zoom Out, а для установки режима масштабирования – команды Stretch HALFTONE, Stretch COLORNCOLOR. Обработчики этих команд добавим в класс облика (листинг 3.2.1). Эти функции изменяют состояние переменных m\_dScale и m\_nStretchModeстановки режимов масштабирования также добавлены методы OnUpdateViewStretchhalftone() и OnUpdateViewStretchcoloroncolor() для обработки сообщения UPDATE\_COMMAND\_UI. В этих функциях можно управлять состоянием соответствующих команд в интерфейсе программы (например, можно делать недоступными команды в зависимости от состояния программы). В данном случае мы просто маркируем соответствующий режим масштабирования.

Листинг 3.2.1 – Обработка команд масштабирования. Файл BMView.cpp

void CBMView::OnViewZoomin()

{// TODO: Add your command handler code here

m\_dScale\*=2;

OnUpdate(NULL, 0, NULL); }

void CBMView::OnViewZoomout()

{// TODO: Add your command handler code here

m\_dScale/=2;

OnUpdate(NULL, 0, NULL); }

void CBMView::OnViewStretchhalftone()

{// TODO: Add your command handler code here

m\_nStretchMode=HALFTONE;

OnUpdate(NULL, 0, NULL); }

void CBMView::OnUpdateViewStretchhalftone(CCmdUI\* pCmdUI)

{// TODO: Add your command update UI handler code here

pCmdUI->SetCheck(m\_nStretchMode==HALFTONE); }

void CBMView::OnViewStretchcoloroncolor()

{// TODO: Add your command handler code here

m\_nStretchMode=COLORONCOLOR;

OnUpdate(NULL, 0, NULL); }

void CBMView::OnUpdateViewStretchcoloroncolor(CCmdUI\* pCmdUI)

{// TODO: Add your command update UI handler code here

pCmdUI->SetCheck(m\_nStretchMode==COLORONCOLOR);}

**3.3 Модификация класса документа для обеспечения работы с изображениями**

Поскольку данными в нашей программе будут изображения, модифицируем класс документа так, чтобы он умел работать с изображениями. В проект приложения добавим файлы Raster.h, Raster.cpp. В классе документа надо определить данные как объекты класса СRaster. В принципе, для целей показа картинки на экране хватит и одного объекта СRaster. Чтобы наделить программу некоторыми возможностями по редактированию изображений потребуется не один, а, как минимум, два объекта: один для хранения исходной картинки, второй - буфер для приема преобразованной картинки.

Порядок работы с двумя объектами СRaster в этом случае будет выглядеть следующим образом:

- Загружаем изображение в первый объект СRaster и показываем его наэкране до тех пор, пока пользователь не даст команду выполнить какие-нибудь изменения изображения.

- Помещаем измененное изображение во второй объект СRaster и начинаем показывать второй объект-картинку.

- Может случиться так, что пользователю не понравится то, как мы изменили его картинку, тогда он отдает команду "Отменить преобразования".При этом меняются объекты местами.

**3.4 Программная схема выполнения преобразований. Графические фильтры**

Поскольку следует реализовать целый ряд процедур преобразования изображений, необходимо предусмотреть, как они будут уживаться между собой и взаимодействовать с остальными модулями программы. В мультимедийном программировании широко распространена концепция фильтров. Фильтр - это некоторая программа, которая, пропуская через себя данные, преобразует их некоторым образом. В нашем случае данными являются значения цветов пикселов изображения. Представим, что у нас имеется набор фильтров, пропуская через которые данные изображения мы можем добиваться различных эффектов (рис. 3.6.1).

Для того чтобы получить нужный эффект, достаточно просто указать программе, какой фильтр считать активным. В программе должна существовать "фильтрация" - процедура, в которой будет выполняться само пропускание данных через фильтр.

Фильтры можно реализовать в виде классов, производных от какого-то одного базового класса. В базовом классе следует определить набор методов, общих для всех фильтров. В программе заведем переменную - указатель на активный фильтр. Используя этот указатель, "фильтрация" будет обращаться к нужному фильтру.



Рисунок 3.6.1 - Схема использования фильтров для преобразования изображений

Саму фильтрацию изображения можно выполнять по-разному. Например, можно передать фильтру всю исходную картинку и ожидать от него уже полностью преобразованного изображения. А можно пропускать через фильтр исходное изображение по одному пикселу. В последнем случае не придется дублировать цикл обработки всего изображения в каждом фильтре, и вызывающая фильтр процедура получит полный контроль над областью изображения, к которой будет применено преобразование.

Реализуем на практике второй способ организации "фильтрации". При этом сам процесс преобразования изображения вынесем в отдельный поток (назовем его "рабочим" потоком) выполнения программы. Это даст нам возможность контролировать не только область применения фильтра, но и продолжительность выполнения операции, т.е. возможность остановить выполнение "фильтрации". Общая схема преобразования в этом случае будет выглядеть следующим образом:

1. Пришла команда выполнить преобразование - создаем рабочий поток.
2. Уведомляем объекты-облики о том, что начали преобразование. Приэтом облик запускает таймер и начинает периодически интересоваться,сколько процентов работы выполнено, показывая пользователю процентвыполнения.
3. В рабочем потоке выполняется преобразование и увеличивается процент выполнения.
4. По окончании преобразования (или если пользователь прервал выполнение) в объекты-облики посылаются сообщения о завершении работы ипоказывается преобразованная картинка.

Поскольку данными в программе ВМViewer заведует класс CBMDoc, именно в него и поместим "фильтрацию". Для создания рабочего потока потребуется добавить в класс CBMDoc несколько методов:

Transform() - создает рабочий поток;

ThreadProc () - функция потока, запускает "фильтрацию" для конкретного объекта-документа;

TransformLoop() - сама "фильтрация";

InformAllViews() - передает сообщения всем обликам документа; Рассмотрим метод TransformLoop() (Листинг 3.6.1).

Листинг 3.6.1 – Метод CBMDoc::TransformLoop(). Файл BMDoc.cpp

void CBMDoc::TransformLoop()

{if(m\_pCurFilter==NULL) return;

if(!CreateCompatibleBuffer()) return;

m\_EventDoTransform.SetEvent();

m\_bEditable=FALSE;

InformAllViews(UM\_STARTTRANSFORM);

CRaster\*pSBM=GetCurrentBMPtr(),//источник

\*pDBM=GetBufferBMPtr();// приёмник

// Установили в фильтр источник и приёмник преобразований

m\_pCurFilter->SetBuffers(pSBM, pDBM);

for(LONG y=0; y<pSBM->GetBMHeight(); y++)

{// Процент выполнения

InterlockedExchange(&m\_lExecutedPercent, 100\*y/pSBM->GetBMHeight());

//Проверка не решили ли прервать преобразование

if(!m\_EventDoTransform.Lock(0))

{InformAllViews(UM\_ENDOFTRANSFORM, FALSE, 0);

m\_bEditable=TRUE;

return; }

LONG x=0;

// Преобразование с использованием текущего фильтра

for(; x<pSBM->GetBMWidth(); x++)

m\_pCurFilter->TransformPix(x, y); }

m\_EventDoTransform.ResetEvent();

m\_bEditable=TRUE;

SwapBM();//Сделать буфер текущим изображением

SetModifiedFlag(); //флаг “данные изменились”

InformAllViews(UM\_ENDOFTRANSFORM, TRUE, 0);

return;

};

В методе TransformLoop() мы сначала "зажигаем" событие "Выполняется преобразование" - объект m\_EventDoTransform класса CEvent. Затем сообщаем текущему фильтру, какое изображение будет исходным, и какое - приемным (адреса объектов CRaster). Далее в цикле прогоняем через фильтр пикселы изображения. На текущий фильтр указывает переменная m\_pCurFilter, которую мы завели в классе CBMDoc специально для этих целей. Тип этой переменной - указатель на объект класса CFilter. Преобразование же данных выполняется с помощью метода Cfilter::TransformPix(), Класс СFilter как раз и является базовым для всех фильтров.

В процессе преобразования перед обработкой очередной строки пикселов вычисляется процент выполнения как процент уже обработанных строк изображения. Вычисленное значение записывается в переменную m\_lExecutedPercent с помощью API-функции InterlockedExchange() - эта функция позволяет предотвратить одновременное обращение к переменной из разных потоков. Далее проверяется, по-прежнему ли установлено событие m\_EventDoTransform. И только затем обрабатываются пикселы строки. Причем в нашей программе в иллюстрационных целях мы позволяем пользователю посмотреть эффект преобразования на половине изображения. Если установлен флаг m\_bEditHalf, первая половина строки копируется в неизменном виде.

После того как все пикселы изображения были обработаны, скидывается флаг m\_EventDoTransform, буферное изображение становится активным и во все облики направляется сообщение UM\_ENDOFTRANSFORM с параметром TRUE, который говорит о том, что преобразование завершилось и надо обновить изображение в окне облика.

Для контроля количества выполненной работы фильтра в класс CBMView с помощью ClassWizard добавим метод OnTimer(). В этом методе будет выполняться запрос процента выполнения операции и обновляться информация о выполнении. Процент выполнения операции отображается в заголовке окна облика.

Приход сообщения UM\_ENDOFTRANSFORM обрабатывается методом OnEndTransform(), который зависит от значения аргумента wParam:

* TRUE - преобразование успешно закончено - выполняет обновление экрана;
* FALSE - пользователь прервал операцию - не выполняет обновление экрана. Далее им вызывается функция OnStopTimer(), которая разрушает таймер.

Выделение операций обработки данных, которые могут выполняться длительный отрезок времени, в отдельный поток позволяет пользователю сохранить контроль над выполнением программы. В нашем приложении пользователь, запустив фильтрацию на одном из открытыхизображений, может переключиться на просмотр и редактирование другого изображения. При необходимости пользователь может остановить выполнение преобразования, для этого в программе предусмотрим команду, которая бы сбрасывала флаг m\_EventDoTransform. При сбросе этого флага цикл выполнения преобразования СВМDос::ТгаnsformLoop() прерывается, потоковая функция завершается и рабочий поток прекращает свое существование.

**3.5 Класс “Фильтр”**

Выполнение задачи подразумевает существование в программе некоторого объекта-фильтра. Фильтры выполняют разные преобразования, но с точки зрения "фильтрации" они все одинаковы и обращаться с ними она будет единообразно. Поэтому нам надо определить базовый класс CFilter для фильтра с минимальным, но основным набором методов, с помощью которых будет происходить общение. Данные класса - два указателя на объекты-картинки класса Craster:

- m\_рSourseBM - адрес объекта "исходная картинка", откуда берутся данные для преобразования;

- m\_рDestBM - адрес объекта "приемная картинка", куда помещаются преобразованные данные.

Методы класса:

- SetBuffers () - сообщает фильтру адреса исходного и приемного изображения;

- TransformPix() – преобразует данные одного пиксела с координатами (x,y).

Переменная-указатель на этот класс m\_pCurFilter заведена в классе CBMDoc. Этой переменной присваивается адрес текущего фильтра.

Для реализации точечных методов преобразования создаём класс CdotFilter (Листинг 3.7.1).

Листинг 3.7.1 – Базовый класс для точечных фильтров CdotFilter. Файл Filter.h

//Базовый класс для точечных фильтров

class CDotFilter: public CFilter

{

protected:

//Таблицы преобразования для компонентов цвета

BYTE BGRTransTable[3][256];

public:

//Метод преобразования пиксела

BOOL TransformPix(LONG x, LONG y);};

Данными этого класса являются три таблицы преобразования компонентов RGB цвета.

Для точечного фильтра переопределён метод . Реализация метода приведена в листинге 3.7.2

Листинг 3.7.2 – Метод CDotFilter:: TransformPix (). Файл Filter.cpp

BOOL CDotFilter::TransformPix(LONG x, LONG y)

{BYTE \*pDPix=NULL, \*pSPix=NULL;

// Источник необходим

if(m\_pSourceBM==NULL)

return FALSE;

//Если приёмник не задан, то преобразование помещаем в источник

if(m\_pDestBM==NULL)

m\_pDestBM=m\_pSourceBM;

// Получаем указатели на пикселы в источнике и приёмнике

if((pDPix=m\_pDestBM->GetPixPtr(x, y))==NULL ||

(pSPix=m\_pSourceBM->GetPixPtr(x, y))==NULL)

return FALSE;

// Преобразование. Порядок BGR

\*pDPix=BGRTransTable[0][\*pSPix];

\*(pDPix+1)=BGRTransTable[1][\*(pSPix+1)];

\*(pDPix+2)=BGRTransTable[2][\*(pSPix+2)];

return TRUE; };

Хотя формат 24-битового цвета называют RGB, в файле формата BMP компоненты цвета хранятся в обратном порядке (Порядок BGR).

В производных от CDotFilter классах останется реализовать инициализацию таблиц преобразования.

Для реализации пространственных (матричных) методов преобразования создаём класс CMatrixFilter. Интерфейс класса приведён в листинге 3.7.3

Листинг 3.7.3 – Интерфейс базового для матричных фильтров класса CmatrixFilter. Файл Filter.h

// Пространственные (матричные фильтры)

// Базовый класс

class CMatrixFilter: public CFilter

{

protected:

int m\_rangX; // размер матрицы по X и Y

int m\_rangY;

const int \*m\_pMatrix; // указатель на матрицу

public:

//Методпреобразования пиксела

BOOL TransformPix(LONG x, LONG y); };

Данными этого класса являются размер матрицы преобразования и указатель на матрицу. Размер мртрицы определяет зону пикселов, окружающих пиксел (x,y), которая будет вовлечена в расчёт нового значения пиксела (x,y). Указателю на матрицу преобразования m\_pMatrix будет присваиваться адрес матрицы, которая будет использована в преобразовании. Реализация метода CmatrixFilter:: TransformPix() приведена в листинге3.7.4

Листинг 3.7.4 – Метод CmatrixFilter:: TransformPix(). Файл Filter.cpp

// Пространственные фильтры

BOOL CMatrixFilter::TransformPix(LONG x, LONG y)

{BYTE \*pDPix=NULL, \*pSPix=NULL;

// Источник и приёмник необходимы

if(m\_pSourceBM==NULL || m\_pDestBM==NULL)

return FALSE;

// Определяем зону перекрытия изображения и матрицы. Это требуется для //обработки пикселов, находящихся на границах изображения

int x\_start=0;

int dx=m\_rangX/2, dy=m\_rangY/2;

if(x-dx<0) x\_start=dx-x;

int y\_start=0;

if(y-dy<0) y\_start=dy-y;

int x\_finish=m\_rangX;

if(x+dx>m\_pSourceBM->GetBMWidth())

x\_finish-=(x+dx-m\_pSourceBM->GetBMWidth());

int y\_finish=m\_rangY;

if(y+dy>m\_pSourceBM->GetBMHeight())

y\_finish-=(y+dy-m\_pSourceBM->GetBMHeight());

// Расчёт новых значений цвета пиксела с учётом соседей, попавших в зону //действия матрицы преобразования

int NewBGR[3];

int count=0;

for(int c=0, mx=0, my=0; c<3; c++)

{NewBGR[c]=0; count=0;

for(my=y\_start; my<y\_finish; my++)

for(mx=x\_start; mx<x\_finish; mx++)

{if((pSPix=m\_pSourceBM->GetPixPtr(x+(mx-dx), y+(my-dy)))!=NULL)

{NewBGR[c]+=(m\_pMatrix[my\*m\_rangX+mx]\*(\*(pSPix+c)));

count+=m\_pMatrix[my\*m\_rangX+mx]; }}}

// Адрес пиксела в изображении-приёмнике

pDPix=m\_pDestBM->GetPixPtr(x, y);

//Установка нового значения в приёмное изображение

for(c=0; c<3; c++)

{

// Приведение значения к допустимому диапазону

if(count!=0)

NewBGR[c]=NewBGR[c]/count;

if(NewBGR[c]<0)

NewBGR[c]=0;

else if(NewBGR[c]>255)

NewBGR[c]=255;

\*(pDPix+c)=NewBGR[c]; }

return TRUE; };

В методе CmatrixFilter:: TransformPix() сначала определяется область перекрытия изображения и матрицы преобразования. Этот шаг необходим в связи с тем, что на границах изображения пиксел может не иметь соседей.

Новое значение пиксела формируется с учетом значений всех пикселов и коэффициентов матрицы преобразования, попавших в область перекрытия изображения и матрицы преобразования.

**3.6 Фильтр “Яркость/Контраст”**

Изменение яркости заключается в изменении интенсивности цвета всех пикселов на заданное значение. Данное преобразование является точечным. Для его реализации добавим в программу класс CBrightCont, производный от класса CDotFilter. Интерфейс класса приведён в листинге 3.6.1

Листинг 3.6.1 – Интерфейс класса CBrightCont. Файл Filter.h

// Яркость/контраст

class CBrightCont: public CDotFilter

{

public:

BOOL Init(int b\_offset, int c\_offset);

};

Переменные b\_offset, c\_offset – это объекты, связанные с ползунками, могут принимать положительные и отрицательные значения, что соответствует увеличению или уменьшению яркости/контрастности изображения.

Реализация метода CBrightCont::Init() приведена в листинге 3.6.2 Этот метод инициализирует таблицы преобразования. Сначала выполняется смещение яркости на заданную величину, а затем либо "сжатие", либо "растяжение" диапазона яркости. Причем при сжатии значения яркости изменяются не равномерно, а пропорционально их удаленности от "серой середины", определенной константой CONTRAST\_MEDIAN. После преобразования яркости работа по коррекции контрастности происходит со значениями таблицы преобразования, полагая при этом, что они являются индексами в таблице, полученной после коррекции яркости.

Листинг 3.6.2 – Метод CBrightCont::Init().Файл Filter.cpp

// "Серая середина" –уровень 159

#define CONTRAST\_MEDIAN 159

BOOL CBrightCont::Init(int b\_offset, int c\_offset)

{int i=0,//Индекс цвета в таблице преобразований

t=0,//Индекс таблицы

//Индекс цвета, соответствующего нижней границе яркости

t\_index=0,

// Индекс цвета, соответствующего верхней границе яркости

b\_index=0,

value\_offset; //Смещение значения цвета

double value=0.; //Новое значение цвета

//Изменяем яркость

for(i, t=0; t<3; t++)

for(i=0; i<256; i++)

{if(i+b\_offset>255) BGRTransTable[t][i]=255;

else if(i+b\_offset<0) BGRTransTable[t][i]=0;

else BGRTransTable[t][i]=i+b\_offset; }

// Изменяем контрастность

if(c\_offset<0)// Уменьшаем контрастность

{for(i=0, t=0; t<3; t++)

for(i=0; i<256; i++)

if(BGRTransTable[t][i]<CONTRAST\_MEDIAN)

{

//Рассчитываем смещение в зависимости от удалённости цвета от “серой середины”

value\_offset=(CONTRAST\_MEDIAN-BGRTransTable[t][i])\*c\_offset/128;

if(BGRTransTable[t][i]-value\_offset>CONTRAST\_MEDIAN) BGRTransTable[t][i]=CONTRAST\_MEDIAN;

else BGRTransTable[t][i]-=value\_offset; }

else

{

// Рассчитываем смещение в зависимости от удалённости цвета от “серой середины”

value\_offset=(BGRTransTable[t][i]-CONTRAST\_MEDIAN)\*c\_offset/128;

if(BGRTransTable[t][i]+value\_offset<CONTRAST\_MEDIAN) BGRTransTable[t][i]=CONTRAST\_MEDIAN;

else BGRTransTable[t][i]+=value\_offset; }

}

elseif(c\_offset>0)

//Увеличиваем контрастность

{ //Расчёт нижней границы цвета

int offset\_b=c\_offset\*CONTRAST\_MEDIAN/128;

//Все значения в таблице ниже нижней границы получат значения 0

for(t=0; t<3; t++)

for(b\_index=0; b\_index<256; b\_index++)

{ if(BGRTransTable[t][b\_index]<offset\_b)

BGRTransTable[t][b\_index]=0;

else break; }

// Расчёт верхней границы цвета

int offset\_t=c\_offset\*128/CONTRAST\_MEDIAN;

// Все значения в таблице ниже нижней границы получат значения 255

for(t=0; t<3; t++)

for(t\_index=255; t\_index>=0; t\_index--)

{ if(BGRTransTable[t][t\_index]+offset\_t>255)

BGRTransTable[t][t\_index]=255;

else break; }

//Расчёт шага изменения интенсивности цвета

double step=256./(256-(offset\_b+offset\_t));

// "Растягиваем" интенсивность цветов между нижней и верхней //границами до диапазона 0-255

for(t=0; t<3; t++)

{ value=0.;

for(i=b\_index; i<=t\_index; i++)

{ if(BGRTransTable[t][i]>=offset\_b || BGRTransTable[t][i]<256- offset\_t)

{value=(int)((BGRTransTable[t][i]-offset\_b)\*step+0.5);

if(value>255) value=255;

BGRTransTable[t][i]=(int)(value); }}}}

return TRUE; };

**3.7 Фильтр “Инверсия”**

Этот фильтр реализуется с помощью таблицы преобразований. Для его реализации добавим в программу класс CInvertColors (листинг 3.7.1), производный от класса CDotFilter.

Листинг 3.7.1 – Интерфейс класса CInvertColors.Файл Filter.h

//Инверсия цветов

class CInvertColors: public CDotFilter

{

public:

CInvertColors(); };

Операция инверсии цветов не требует никаких настроечных параметров, поэтому инициализация таблиц преобразования выполняется в конструкторе класса (листинг 3.7.1).

Листинг 3.7.1 – Конструктор класса CInvertColors .Файл Filter.cpp

CInvertColors::CInvertColors()

{for(int i=0, t=0; t<3; t++)

for(i=0; i<256; i++)

{BGRTransTable[t][i]=255-i; }

};

**3.8 Фильтр “Размытие”**

Фильтр "Размытие" - это уже пространственное преобразование. Применение этого фильтра оказывает эффект сглаживания деталей изображения. Фильтр реализуется классом CВlur (листинг 3.8.1)

Листинг 3.8.1 – Интерфейс класса CBlur. Файл Filter.h

class CBlur: public CMatrixFilter

{public:

CBlur();

};

Листинг 3.8.2 – Конструктор класса CBlur.Файл Filter.cpp

const int BlurMatrix[25]=

{1,1,1, 1,1,

1,1,1, 1,1,

1,1,1, 1,1,

1,1,1, 1,1,

1,1,1, 1,1

};

CBlur::CBlur()

{m\_pMatrix=BlurMatrix;

m\_rangX=5;

m\_rangY=5; };

Матрица BlurMatrix задаёт преобразование “Размытие”, а в конструкторе CBlur() запоминается её адрес и размер.

**3.9 Фильтр “Резкость”**

Для повышения четкости изображения в фильтре используется матрица "Размытие". Задача повышения четкости изображения заключается в том, чтобы выделить высокочастотные детали изображения. Светлые детали сделать ярче, темные - темнее. Для этого изображение сначала размывается, а затем определяется разность между размытым изображением и оригиналом. На величину этой разницы изменяется яркость оригинала. Таким образом, однородные участки изображения не подвергнутся изменениям, а те места картинки, где присутствуют высокочастотные детали, станут конрастнее. Фильтр реализуется классом CSharp (листинг 3.9.1).

Листинг 3.9.1 – Интерфейс класса CSharp. Файл Filter.h

class CSharp: public CMatrixFilter

{ public:

CSharp();

BOOL TransformPix(LONG x, LONG y); };

В классе CSharp переопределён метод TransformPix(), реализация метода приведена в листинге 3.9.1.

Листинге 3.9.1 – Методы класса CSharp. Файл Filter.cpp.

CSharp::CSharp()

{ m\_pMatrix=BlurMatrix;

m\_rangX=5;

m\_rangY=5; };

// коэффициент увеличения резкости

#define SHARP\_COEFF 3

BOOL CSharp::TransformPix(LONG x, LONG y)

{ //Размыли пиксел

if(!CMatrixFilter::TransformPix(x, y))

return FALSE;

BYTE \*pDPix=NULL, \*pSPix=NULL;

pSPix=m\_pSourceBM->GetPixPtr(x,y);

pDPix=m\_pDestBM->GetPixPtr(x, y);

int d=0;

for(int c=0; c<3; c++)

{ // Нашли разницу

d=\*(pSPix+c)-\*(pDPix+c);

// Усилили разницу

d\*=SHARP\_COEFF;

// Присвоили пикселу новое значение

if(\*(pDPix+c)+d <0)

\*(pDPix+c)=0;

else

if(\*(pDPix+c)+d > 255)

\*(pDPix+c)=255;

else

\*(pDPix+c)+=d;}

return TRUE; }

**4. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ**

Запуск программы осуществляется при открытии файла BMViewer.exe. На экране появляется окно, представленное на рисунке 4.1.

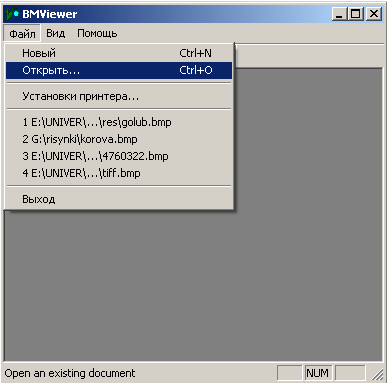


Рисунок 4.1 – Главное окно программы

Выбрав в меню Файл/Открыть… пльзователь выбирает файл, который следует открыть. Одновременно может быть открыто несколько файлов. С помощью пункта меню Файл/Сохранить сохраняется текущий вид изображения. Пункт меню Файл/Сохранить… позволяет сохранить текущий вид изображения под новым именем.

Пункты меню Вид/Масштаб In, Масштаб Out позволяют увеличивать/уменьшать масштаб изображения (Рисунок 4.2).

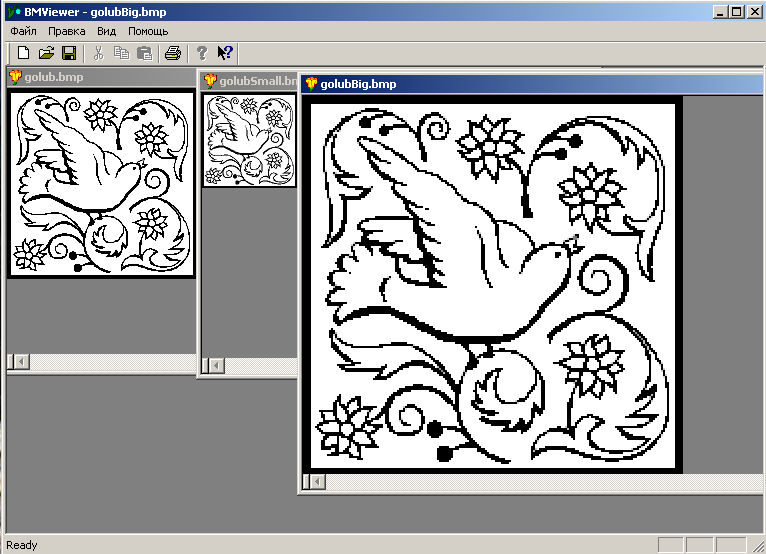


Рисунок 4.2 – Масштабирование изображения.

Пункты меню Правка/… позволяют корректировать открытое изображение при помощи фильтров: Яркость/Контраст (Brightness/Contrast), Резкость (Sharp), Размытие (Blur), Инверсия (Inverse) представлены на рис.4.3.

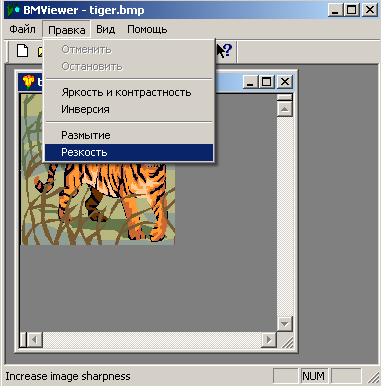


Рисунок 4.3 –Пункты меню Правка

Применение фильтра Размытие оказывает эффект сглаживания деталей изображения. Фильтр целесообразно использовать для повышения качества изображения низкого разрешения. Границы пикселов становятся менее чёткими, сглаживается «шум», что зрительно улучшает качество изображения. Результат применения фильтра представлен на рисунке 4.4.

Фильтр Резкость целесообразно применять при работе с изображением, в котором присутствует большое количество мелких повторяющихся деталей не низкого качества: например, рисунок на ткани или шерсть животных. Применение в этих случаях фильтра Резкость даёт нам в результате изображение, наиболее приближённое к оригиналу. Фильтр Резкость зачастую применяют в рекламной полиграфии для преувеличенного выделения отдельных деталей изображения, на которых следует сделать акцент. Результат применения фильтра представлен на рисунке 4.5.

Фильтр Яркость/Контраст - наиболее часто применяется для цветокоррекции слишком затемнённых изображений. Сначала следует увеличить яркость изображения, затем повысить контрастность. Как правило, контрастность в этом случае следует увеличить немного меньше, чем яркость. Результат применения фильтра представлен на рисунке 4.6.

Фильтр Инверсия инвертирует цвета изображения, что даёт явно заметный, яркий результат. Этот эффект можно использовать для любителей креативного, гипертрофированно яркого дизайна. Применение фильтра представлено на рисунке 4.7.

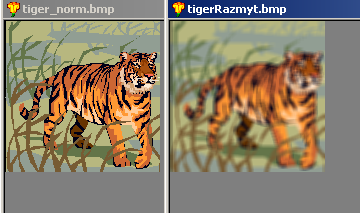


Рисунок 4.4 – результат применения фильтра Размытие



Рисунок 4.5 – результат применения фильтра Резкость (с масштабом)

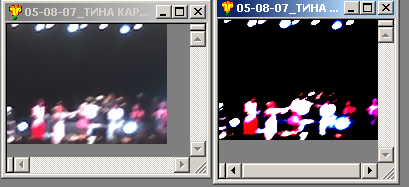


Рисунок 4.6 – результат применения фильтра Яркость/Контраст

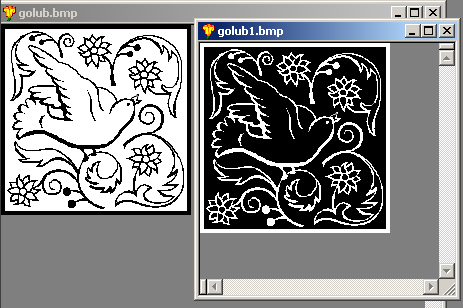


Рисунок 4.7 – результат применения фильтра Инверсия

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения данного дипломного проекта было разработано программное обеспечение для открытия и фильтрации растровых изображений при помощи графических фильтров, основанных на пространственных и точечных преобразованиях.

Систематизированы и закреплены практические навыки использования ЭВМ, программного обеспечения, существующих графических редакторов, работающих с растровой графикой, а также теоретические знания по курсу “Технология программирования гибких компьютеризированных систем” .

**ПЕРЕЧЕНЬ ССЫЛОК**

1. Поляков А.Ю., Брусенцев В.А. Методы и алгоритмы компьютерной графики в примерах на Visual C++, 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2003. – 560 с.
2. Мюррей Д., Райпер У. Ван. Энциклопедия форматов графических файлов/пер. с англ. – К.: BHV, 1997. – 672 с.
3. Холзнер С. Visual C++ 6: учебный курс – СПб: Питер, 2001. – 576 с.
4. Шилдт Г. Справочник программиста по С/С++.: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 448 с.
5. Круглински Д., Уингоу С., Шеферд Дж. Программирование на Microsoft Visual C++ 6.0 для профессионалов/Пер. с англ. - СПб: Питер; М.:Издательско-торговый дом "Русская Редакция", 2003. - 864 с.
6. Глушаков С.В., Коваль А.В., Черепнин С.А. Программирование на Visual C++ 6.0. – Харьков: Фолио, 2002.- 726 с.

**ПРИЛОЖЕНИЕ**

**Листсинг программы**

// BMDoc.cpp : implementation of the CBMDoc class

**//**

#include "stdafx.h"

#include "BMViewer.h"

#include "BMDoc.h"

#include "HistDlg.h"

#include "BrightContDlg.h"

#include "DeNoiseDlg.h"

#include "BMView.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

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

// CBMDoc

IMPLEMENT\_DYNCREATE(CBMDoc, CDocument)

BEGIN\_MESSAGE\_MAP(CBMDoc, CDocument)

//{{AFX\_MSG\_MAP(CBMDoc)

ON\_COMMAND(ID\_EDIT\_UNDO, OnEditUndo)

ON\_COMMAND(ID\_EDIT\_BRIGHTNESSANDCONTRAST, OnEditBrightnessandcontrast)

ON\_COMMAND(ID\_EDIT\_INVERTCOLORS, OnEditInvertcolors)

ON\_COMMAND(ID\_EDIT\_BLUR, OnEditBlur)

ON\_COMMAND(ID\_EDIT\_SHARP, OnEditSharp)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_BLUR, OnUpdateEditBlur)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_SHARP, OnUpdateEditSharp)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_UNDO, OnUpdateEditUndo)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_BRIGHTNESSANDCONTRAST, OnUpdateEditBrightnessandcontrast)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_INVERTCOLORS, OnUpdateEditInvertcolors)

ON\_COMMAND(ID\_EDIT\_STOP, OnEditStop)

ON\_UPDATE\_COMMAND\_UI(ID\_EDIT\_STOP, OnUpdateEditStop)

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

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

// CBMDoc construction/destruction

CBMDoc::CBMDoc():m\_EventDoTransform(FALSE, TRUE)

{

// TODO: add one-time construction code here

m\_pCurBM=NULL;

m\_bEditable=FALSE;

m\_pCurFilter=NULL;

}

CBMDoc::~CBMDoc()

{

}

BOOL CBMDoc::OnNewDocument()

{

if (!CDocument::OnNewDocument())

return FALSE;

// TODO: add reinitialization code here

// (SDI documents will reuse this document)

return TRUE;

}

// Адрес пикселя

pCurPix=m\_pData+y\*DataStrLength+x\*3;

// Яркость рассчитывается как 0.3\*Red+0.59\*Green+0.11\*Blue,

// но пиксельные данные хранятся в файле BMP, в порядке BGR

Brightness=(BYTE)((0.11\*(\*pCurPix) +

0.59\*(\*(pCurPix+1))+

0.3\*(\*(pCurPix+2)))\*Range/256);

pHist[Brightness]+=1;

}

return TRUE;

};

**// stdafx.cpp : source file that includes just the standard includes**

//BMViewer.pch will be the pre-compiled header

//stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

**// BMDoc.h : interface of the CBMDoc class**

**//**

**/////////////////////////////////////////////////////////////////////////////**

#if !defined(AFX\_BMDOC\_H\_\_0CC04D31\_A879\_4674\_AAD5\_292699AE7BA7\_\_INCLUDED\_)

#define AFX\_BMDOC\_H\_\_0CC04D31\_A879\_4674\_AAD5\_292699AE7BA7\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

#include <afxmt.h>

#include "Raster.h"

#include "Filter.h"

#define UM\_STARTTRANSFORMWM\_USER+0x8000

#define UM\_ENDOFTRANSFORMUM\_STARTTRANSFORM+1

class CBMDoc : public CDocument

{

protected: // create from serialization only

CBMDoc();

DECLARE\_DYNCREATE(CBMDoc)

// Attributes

public:

// Флаги

BOOLm\_bEditable;// можем ли редактировать данные

CEventm\_EventDoTransform; // событие - выполняется преобразование

// Статистика

LONGm\_lExecutedPercent;

// Данные

CRaster m\_BM[2];// два буфера для изображений

CRaster \*m\_pCurBM;// указатель на активный буфер

//Фильтры

CFilter\*m\_pCurFilter;

CBrightCont m\_BrightContFilter;

CInvertColors m\_InvertColorsFilter;

CBlurm\_BlurFilter;

CSharpm\_SharpFilter;

// Operations

public:

CSize GetCurrentBMSize();

//Возвращает номер активного буфера

intGetNCurrentBM();

//Возвращает указатель на текущую картинку

CRaster\* GetCurrentBMPtr();

//Возвращает указатель на буфер

CRaster\* GetBufferBMPtr();

//Поменять текущее изображение и буффер местами

void SwapBM();

//Нарисовать текущее изображение

void DrawCurrent();

// Создает буфер заданного размера

// (при вызове без аргументов размер равен текущей картинке)

// совместимый с текущей картинкой

BOOL CreateCompatibleBuffer(LONG width=0, LONG height=0);

//Запускает поток преобразования

void Transform();

protected:

//Функция потока преобразования

static UINT ThreadProc(LPVOID pv);

//Цикл преобразования с использованием активного фильтра

void TransformLoop();

//Посылает сообщение всем окнам

void InformAllViews(UINT msg, WPARAM wPar=0, LPARAM lPar=0);

public:

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CBMDoc)

public:

virtual BOOL OnNewDocument();

virtual void Serialize(CArchive& ar);

virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);

virtual BOOL OnSaveDocument(LPCTSTR lpszPathName);

//}}AFX\_VIRTUAL

// Implementation

public:

void Onio();

virtual ~CBMDoc();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected:

// Generated message map functions

protected:

//{{AFX\_MSG(CBMDoc)

afx\_msg void OnEditUndo();

afx\_msg void OnEditBrightnessandcontrast();

afx\_msg void OnEditInvertcolors();

afx\_msg void OnEditBlur();

afx\_msg void OnEditSharp();

afx\_msg void OnUpdateEditBlur(CCmdUI\* pCmdUI);

afx\_msg void OnUpdateEditSharp(CCmdUI\* pCmdUI);

afx\_msg void OnUpdateEditUndo(CCmdUI\* pCmdUI);

afx\_msg void OnUpdateEditBrightnessandcontrast(CCmdUI\* pCmdUI);

afx\_msg void OnUpdateEditInvertcolors(CCmdUI\* pCmdUI);

afx\_msg void OnEditStop();

afx\_msg void OnUpdateEditStop(CCmdUI\* pCmdUI);

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

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

//{{AFX\_INSERT\_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_BMDOC\_H\_\_0CC04D31\_A879\_4674\_AAD5\_292699AE7BA7\_\_INCLUDED\_)

**// BMView.h : interface of the CBMView class**

**//**

**/////////////////////////////////////////////////////////////////////////////**

#if !defined(AFX\_BMVIEW\_H\_\_A58B1237\_DC72\_4389\_BA43\_93CC54F0F5EA\_\_INCLUDED\_)

#define AFX\_BMVIEW\_H\_\_A58B1237\_DC72\_4389\_BA43\_93CC54F0F5EA\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

class CBMView : public CScrollView

{

// Идентификатор таймера

UINT m\_nTimer;

// Заголовок окна

CString m\_strOrgTitle;

//Виртуальный экран

CBitmap m\_VirtScreenBitmap;// растр

CDCm\_VirtScreenDC;// контекст

// Масштаб вывода картинки

doublem\_dScale;

intm\_nStretchMode;

protected: // create from serialization only

CBMView();

DECLARE\_DYNCREATE(CBMView)

// Attributes

public:

CBMDoc\* GetDocument();

// Operations

public:

BOOL UpdateVirtualScreen();

void OnStartTimer();

void OnStopTimer();

afx\_msg LONG OnStartTransform(UINT wParam, LONG lParam);

afx\_msg LONG OnEndTransform(UINT wParam, LONG lParam);

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CBMView)

public:

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

virtual void OnInitialUpdate();

protected:

virtual BOOL OnPreparePrinting(CPrintInfo\* pInfo);

virtual void OnBeginPrinting(CDC\* pDC, CPrintInfo\* pInfo);

virtual void OnEndPrinting(CDC\* pDC, CPrintInfo\* pInfo);

virtual void OnDraw(CDC\* pDC);

virtual void OnUpdate(CView\* pSender, LPARAM lHint, CObject\* pHint);

virtual void OnPrint(CDC\* pDC, CPrintInfo\* pInfo);

//}}AFX\_VIRTUAL

// Implementation

public:

virtual ~CBMView();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected:

// Generated message map functions

protected:

//{{AFX\_MSG(CBMView)

afx\_msg BOOL OnEraseBkgnd(CDC\* pDC);

afx\_msg void OnTimer(UINT nIDEvent);

afx\_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);

afx\_msg void OnViewZoomin();

afx\_msg void OnViewZoomout();

afx\_msg void OnViewStretchhalftone();

afx\_msg void OnUpdateViewStretchhalftone(CCmdUI\* pCmdUI);

afx\_msg void OnViewStretchcoloroncolor();

afx\_msg void OnUpdateViewStretchcoloroncolor(CCmdUI\* pCmdUI);

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

#ifndef \_DEBUG // debug version in BMView.cpp

inline CBMDoc\* CBMView::GetDocument()

{ return (CBMDoc\*)m\_pDocument; }

#endif

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

//{{AFX\_INSERT\_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_BMVIEW\_H\_\_A58B1237\_DC72\_4389\_BA43\_93CC54F0F5EA\_\_INCLUDED\_)

**// BMViewer.h : main header file for the BMVIEWER application**

**//**

#if !defined(AFX\_BMVIEWER\_H\_\_854A9B8C\_6BD4\_44CA\_B4E4\_F00CEF40C1E7\_\_INCLUDED\_)

#define AFX\_BMVIEWER\_H\_\_854A9B8C\_6BD4\_44CA\_B4E4\_F00CEF40C1E7\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

#ifndef \_\_AFXWIN\_H\_\_

#error include 'stdafx.h' before including this file for PCH

#endif

#include "resource.h" // main symbols

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

// CBMApp:

// See BMViewer.cpp for the implementation of this class

//

class CBMApp : public CWinApp

{

public:

CBMApp();

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CBMApp)

public:

virtual BOOL InitInstance();

//}}AFX\_VIRTUAL

// Implementation

//{{AFX\_MSG(CBMApp)

afx\_msg void OnAppAbout();

afx\_msg void OnFileNew();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

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

//{{AFX\_INSERT\_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_BMVIEWER\_H\_\_854A9B8C\_6BD4\_44CA\_B4E4\_F00CEF40C1E7\_\_INCLUDED\_)

**//BrightContDlg.h**

#if !defined(AFX\_BRIGHTCONTDLG\_H\_\_310F8D78\_02DE\_4055\_A8DD\_609EFC475802\_\_INCLUDED\_)

#define AFX\_BRIGHTCONTDLG\_H\_\_310F8D78\_02DE\_4055\_A8DD\_609EFC475802\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

// BrightContDlg.h : header file

//

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

// CBrightContDlg dialog

class CBrightContDlg : public CDialog

{

// Construction

public:

CBrightContDlg(CWnd\* pParent = NULL); // standard constructor

int m\_iBrightnessOffset;

int m\_iContrastOffset;

// Dialog Data

//{{AFX\_DATA(CBrightContDlg)

enum { IDD = IDD\_BRIGHT\_CONT };

CSliderCtrlm\_ctrlContrast;

CSliderCtrlm\_ctrlBrightness;

CStringm\_strBrightness;

CStringm\_strContrast;

//}}AFX\_DATA

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CBrightContDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

//}}AFX\_VIRTUAL

// Implementation

protected:

// Generated message map functions

//{{AFX\_MSG(CBrightContDlg)

virtual BOOL OnInitDialog();

afx\_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar);

virtual void OnOK();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

//{{AFX\_INSERT\_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_BRIGHTCONTDLG\_H\_\_310F8D78\_02DE\_4055\_A8DD\_609EFC475802\_\_INCLUDED\_)

**// ChildFrm.h : interface of the CChildFrame class**

**//**

**/////////////////////////////////////////////////////////////////////////////**

#if !defined(AFX\_CHILDFRM\_H\_\_D84A743A\_B4B5\_4CE3\_80C2\_DF6B6B8A0860\_\_INCLUDED\_)

#define AFX\_CHILDFRM\_H\_\_D84A743A\_B4B5\_4CE3\_80C2\_DF6B6B8A0860\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

class CChildFrame : public CMDIChildWnd

{

DECLARE\_DYNCREATE(CChildFrame)

public:

CChildFrame();

// Attributes

protected:

CSplitterWnd m\_wndSplitter;

public:

// Operations

public:

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CChildFrame)

public:

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

protected:

virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext\* pContext);

//}}AFX\_VIRTUAL

// Implementation

public:

virtual ~CChildFrame();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

// Generated message map functions

protected:

//{{AFX\_MSG(CChildFrame)

// NOTE - the ClassWizard will add and remove member functions here.

// DO NOT EDIT what you see in these blocks of generated code!

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

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

//{{AFX\_INSERT\_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_CHILDFRM\_H\_\_D84A743A\_B4B5\_4CE3\_80C2\_DF6B6B8A0860\_\_INCLUDED\_)

**// Filter.h : interface of CFilter and derived classes**

**/////////////////////////////////////////////////////////////////////////////**

#ifndef \_FILTER\_INCLUDED

#define \_FILTER\_INCLUDED

class CRaster;

// Базовый виртуальный класс

class CFilter

{

protected:

CRaster \*m\_pSourceBM;

CRaster \*m\_pDestBM;

public:

// Устанавливает исходное и приемное изображения

void SetBuffers(CRaster \*pSource, CRaster \*pDest=NULL)

{m\_pSourceBM=pSource;m\_pDestBM=pDest;};

// Виртуальный метод преобразования пиксела

// будет переопределен в производных классах

virtual BOOL TransformPix(LONG x, LONG y){ return FALSE;};

};

// Базовый класс для точечных фильтров

class CDotFilter: public CFilter

{

protected:

// Таблицы преобразования для компонент цвета

BYTE BGRTransTable[3][256];

public:

// Метод преобразования пиксела

BOOL TransformPix(LONG x, LONG y);

};

// Яркость-контраст

class CBrightCont: public CDotFilter

{

public:

BOOL Init(int b\_offset, int c\_offset);

};

// Инверсия цветов

class CInvertColors: public CDotFilter

{

public:

CInvertColors();

};

// Рельеф

/\*class CEmboss: public CDotFilter

{

public:

BOOL TransformPix(LONG x, LONG y);

};\*/

// Пространственные (матричные) фильтры

// Базовый класс

class CMatrixFilter: public CFilter

{

protected:

int m\_rangX; // размер матрицы по X и Y

int m\_rangY;

const int \*m\_pMatrix; // указатель на матрицу

public:

// Метод преобразования пиксела

BOOL TransformPix(LONG x, LONG y);

};

class CBlur: public CMatrixFilter

{

public:

CBlur();

};

class CSharp: public CMatrixFilter

{

public:

CSharp();

BOOL TransformPix(LONG x, LONG y);

};

#endif

**// MainFrm.h : interface of the CMainFrame class**

**//**

**/////////////////////////////////////////////////////////////////////////////**

#if !defined(AFX\_MAINFRM\_H\_\_0E83271E\_987C\_45D5\_8475\_25E6B2754E48\_\_INCLUDED\_)

#define AFX\_MAINFRM\_H\_\_0E83271E\_987C\_45D5\_8475\_25E6B2754E48\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

class CMainFrame : public CMDIFrameWnd

{

DECLARE\_DYNAMIC(CMainFrame)

public:

CMainFrame();

// Attributes

public:

// Operations

Public:

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CMainFrame)

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

//}}AFX\_VIRTUAL

// Implementation

public:

virtual ~CMainFrame();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected: // control bar embedded members

CStatusBar m\_wndStatusBar;

CToolBar m\_wndToolBar;

// Generated message map functions

protected:

//{{AFX\_MSG(CMainFrame)

afx\_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);

// NOTE - the ClassWizard will add and remove member functions here.

// DO NOT EDIT what you see in these blocks of generated code!

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

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

//{{AFX\_INSERT\_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_MAINFRM\_H\_\_0E83271E\_987C\_45D5\_8475\_25E6B2754E48\_\_INCLUDED\_)

**// Raster.h : interface of CRaster class**

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

#ifndef \_RASTER\_INCLUDED

#define \_RASTER\_INCLUDED

// макрос для определения количества байт в выровненной по DWORD строки пикселов в DIB

// Width - длина строки в пикселах; BPP - бит на пиксел

#define BYTESPERLINE(Width, BPP) ((WORD)((((DWORD)(Width) \* \

(DWORD)(BPP) + 31) >> 5)) << 2)

class CRaster

{

LPBITMAPINFOm\_pBMI; //указатель на описание изображения

PBYTEm\_pData;//указатель на начало растровых данных

public:

CRaster();

~CRaster();

void Clear(); //очистка памяти

// Возвращает

LPBITMAPINFO GetBMInfoPtr(){return m\_pBMI;}//указатель на заголовок растра

RGBQUAD\* GetBMColorTabPtr();//указатель на таблицу цветов

LONG GetBMWidth(); //ширину в пикселах;

LONG GetBMHeight();//высоту в пикселах;

BYTE\* GetBMDataPtr(){return m\_pData;};//указатель на растровые данные

BYTE\* GetPixPtr(LONG x, LONG y);//указатель на пиксел

// Загружает из файла

BOOL LoadBMP(CString FileName);

// Выводит DIB на контекст pDC

// x, y - позиция левого верхнего угла области назначения

// cx, cy - размер области назначения

// x0, y0 - позиция левого верхнего угла выводимой части изображения

// cx0, cy0 - размер выводимой части изображения

// str\_mode - режим масштабирования

// rop - растровая операция определяет способ наложения изображения

void DrawBitmap(CDC \*pDC, LONG x=0, LONG y=0, LONG cx=0, LONG cy=0,

LONG x0=0, LONG y0=0, LONG cx0=0, LONG cy0=0, int str\_mode=COLORONCOLOR, DWORD rop=SRCCOPY);

// Выводит DIB на контекст pDC с позиции (x,y) в масштабе scale

void DrawBitmap(CDC \*pDC, LONG x, LONG y, double scale, int str\_mode=COLORONCOLOR, DWORD rop=SRCCOPY);

// Записывает BMP в файл

BOOL SaveBMP(CString FileName);

// Создает копию

BOOL CreateCopy(CRaster \*pOrg);

// Создает растр заданного размера, совместимый с параметрами BITMAPINFO

BOOL CreateCompatible(LPBITMAPINFO pBMI, LONG width=0, LONG height=0);

// Возвращает гисторамму изображения

BOOL GetHistogham(DWORD \*pHist, int Range);

};

#endif

**//{{NO\_DEPENDENCIES}}**

**// Microsoft Developer Studio generated include file.**

**// Used by BMViewer.rc**

**//**

#define IDD\_ABOUTBOX 100

#define IDR\_MAINFRAME 128

#define IDR\_BMVIEWTYPE 129

#define IDD\_HIST 130

#define IDD\_BRIGHT\_CONT 131

#define IDD\_DENOISE 132

#define IDC\_SLIDER\_BRIGHTNESS 1001

#define IDC\_STATIC\_BRIGHTNESS 1002

#define IDC\_SLIDER\_CONTRAST 1003

#define IDC\_SLIDER\_HIST\_B 1003

#define IDC\_STATIC\_CONTRAST 1004

#define IDC\_SLIDER\_HIST\_T 1004

#define IDC\_OFFSET\_B\_TXT 1005

#define IDC\_OFFSET\_T\_TXT 1006

#define IDC\_NOISE\_SMOOTH 1008

#define IDC\_NOISE\_SHOW 1009

#define IDC\_K 1011

#define IDC\_SPIN\_K 1012

#define IDC\_HIST\_VIEW 1014

#define ID\_VIEW\_HISTOGRAM 32771

#define ID\_EDIT\_HISTOGRAM 32772

#define ID\_EDIT\_BRIGHTNESSANDCONTRAST 32773

#define ID\_EDIT\_INVERTCOLORS 32774

#define ID\_EDIT\_BLUR 32775

#define ID\_EDIT\_SHARP 32776

#define ID\_EDIT\_CONTOUR 32777

#define ID\_EDIT\_EDITHALF 32778

#define ID\_EDIT\_HALF 32779

#define ID\_EDIT\_EMBOSS 32780

#define ID\_EDIT\_DENOISE 32781

#define ID\_EDIT\_STOP 32782

#define ID\_VIEW\_ZOOMIN 32792

#define ID\_VIEW\_ZOOMOUT 32793

#define ID\_VIEW\_STRETCHHALFTONE 32794

#define ID\_VIEW\_STRETCHCOLORONCOLOR 32795

#define ID\_\_DFVGD 32796

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_3D\_CONTROLS 1

#define \_APS\_NEXT\_RESOURCE\_VALUE 133

#define \_APS\_NEXT\_COMMAND\_VALUE 32797

#define \_APS\_NEXT\_CONTROL\_VALUE 1015

#define \_APS\_NEXT\_SYMED\_VALUE 101

#endif

#endif

**// stdafx.h : include file for standard system include files,**

**// or project specific include files that are used frequently, but**

**// are changed infrequently**

**//**

#if !defined(AFX\_STDAFX\_H\_\_3471DB7C\_3910\_44C3\_92EA\_6D934D04BC94\_\_INCLUDED\_)

#define AFX\_STDAFX\_H\_\_3471DB7C\_3910\_44C3\_92EA\_6D934D04BC94\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

#define VC\_EXTRALEAN// Exclude rarely-used stuff from Windows headers

#include <afxwin.h> // MFC core and standard components

#include <afxext.h> // MFC extensions

#include <afxdisp.h> // MFC Automation classes

#include <afxdtctl.h>// MFC support for Internet Explorer 4 Common Controls

#ifndef \_AFX\_NO\_AFXCMN\_SUPPORT

#include <afxcmn.h>// MFC support for Windows Common Controls

#endif // \_AFX\_NO\_AFXCMN\_SUPPORT

//{{AFX\_INSERT\_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX\_STDAFX\_H\_\_3471DB7C\_3910\_44C3\_92EA\_6D934D04BC94\_\_INCLUDED\_)