TALLINNA TEHNIKA ÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut
Tarkvaratehnika õppetool

OS Windows graafilise redaktori funktsionaalsuse arendus

Bakalaureusetöö

Üliõpilane: Sergei Malõšev
Üliõpilaskood: 121079
Juhendaja: Kaarel Allik

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

(*kuupäev*) (*allkiri*)

Annotatsioon

Antud töö eesmärgiks on rakendusliidese kasutamine (Win32 API) Windows NT süsteemide jaoks ülesannete lahendamiseks, mis ei saa lahendada ainult C standardteeke (msvcrt.dll) kasutades, Windows akna liidese uurimine, Windows GDI võimaluste uurimine ja objektorienteeritud API GDI+ võimaluste uurimine. Aga samuti joonistamise ja lihtsa piltide töötlemise algoritmide uurimine ja loomine.

Töö käigus projekteeritakse seadme rasterkujutiste avamise, salvestamise ja töötlemise jaoks. Lahendatakse lihtsate tööriistade projekteerimise probleemid, mis saab leida enamikus graafilistes redaktorites. Luuakse lihtsad bitmapi pikselite töötlemise filtrid. Aga samuti tehakse pildi informatsiooni ja statistikat kogumise ja näitamise võimalus ja resultaadi kuvamine.

Tulemusena on loodud prototüüp, mis vastab püstitatud eesmärkidele.

Lõputöö on kirjutatud vene keeles ning sisaldab 31 teksti lehekülge, 10 peatükke, 20 joonist.

Разработка функционала простого графического редактора для ОС Windows

Аннотация

Основными целями данной дипломной работы являются использование интерфейса прикладного программирования (Win32 API) для систем семейства Windows NT для решения задач, которые нельзя решить, используя только стандартную библиотеку C (msvcrt.dll), изучение оконного интерфейса Windows, возможностей интерфейса графического устройства (GDI), возможностей объектно-ориентированного API GDI+. А также изучение и разработка алгоритмов рисования и простой обработки растровых изображений.

В ходе работы проектируется средство открытия, сохранения и обработки растровых графических изображений. Решаются проблемы проектирования простых инструментов рисования, характерных для большинства графических редакторов. Пишутся простые фильтры попиксельной обработки битовых карт. А также делается возможность сбора и отображения информации, статистики об обрабатываемом изображении и отображение результата.

В итоге получен прототип, удовлетворяющий поставленным в начале целям.

Работа написана на русском языке, одержит 31 страницу текста, 10 глав, 20 рисунков.

Development of functionality of the simple graphics editor for OS Windows

Abstract

The main aims of this thesis is application programming interface (Win32 API) for Windows NT operating systems using for problems which can't be solved only with standard C library (msvcrt.dll), studying Windows windowing interface, capabilities of the graphics device interface (GDI), capabilities of the object oriented API GDI+. Studying and developing simple drawing and raster images processing algorithms.

During the work, were designed tools for opening, saving and processing raster images. Solved design problems of the drawing tools, which are common for the majority of the graphics editors. Written simple pixel filters for bitmaps processing. Developed possibility of statistics and information collecting and displaying the result.

As a result was created working prototype, which satisfies of the initial goals.

The thesis is in Russian and contains 31 pages of text, 10 chapters, 20 figures.

Словарь терминов и аббревиатур

Нативное приложение	***Native application*** Приложение, поддерживаемое определённой системой с минимальными вычислительными затратами и дополнительными компонентами.(1)
Интерфейс прикладного программирования	***Application programming interface*** Набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением или системой для использования во внешних программных продуктах.(2)
Дескриптор Windows API	***Windows API handler*** Уникальное число, которое Windows использует для идентификации. Большинство индексов являются значениями внутренних индексов таблиц, в которых описываются реальные объекты операционной системы. (3)
Интерфейс графического устройства (GDI)	***Graphics device interface (GDI)*** Интерфейс Windows для представления графических объектов и передачи их на устройства отображения, такие, как мониторы и принтеры. В Windows XP было введено объектно-ориентированное расширение GDI+.(4)
Контекст устройства	***Device context (DC)*** Структура данных, содержащая сведения об атрибутах рисования на устройстве. (5)

CLSID ***Class identifier (CLSID)***

Один из подтипов глобальных идентификаторов (GUID), идентифицирующий COM класс. Имеет длину 128 битов, записывается 16-ричным числом в фигурных скобках.(6)

Блиттинг ***Block image transfer (blitting)***

Аппаратно-ускоренная операция копирования и наложения части изображения. В GDI используется для переноса изображения из одного контекста рисования на другой.(7)

Аффинное преобразование ***Affine transformation***

Отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные, пересекающиеся в пересекающиеся, скрещивающиеся в скрещивающиеся.(8)

Список изображений

Оглавление

1. Вступление

В наше время приложения с графическим интерфейсом очень сильно распространены, причём на Windows написано приложений больше, чем на какую-либо другую систему. Приложения, использующие средства операционной системы напрямую, а не промежуточную виртуальную машину вроде Java, CLR всегда ценятся больше. (9) К тому же написание "родных" для системы приложений даёт более подробные знания о системе, помогает понять её устройство и даже в каком-то смысле заглянуть под капот. К явным преимуществам таких приложений можно отнести скорость работы. Как правило, чем меньше уровней абстракции, тем приложение работает быстрее. Однако это влечёт за собой усложнение кода, что требует от программиста больше времени и опыта для написания приложения. Ещё одним минусом "нативного" программирования является привязанность к конкретной платформе и невозможность переноса кода на другие платформы. К счастью Windows поддерживает обратную совместимость приложений, поэтому приложения, написанные для старых версий Windows будут без проблем работать на более новых версиях этой операционной системы.

Windows—самая популярная ОС среди настольных компьютеров и ноутбуков, то есть её больше всего используют для простого пользования вроде веб-сёрфинга, игр, проигрывания мультимедиа простые пользователи. Редактирование растровых изображений также является распространённой задачей. Существует множество растровых графических редакторов: бесплатных и очень дорогих, простых, вроде Microsoft Paint и сложных, вроде Adobe Photoshop. Последний обладает очень широкими возможностями: множество инструментов с большим количеством настроек, поддержка слоёв и режимов смешивания, различные фильтры, трансформации, анализ изображения. Кроме этих возможностей работы с растровыми изображениями имеются средства векторной графики, вроде путей, поддержка 3-мерных моделей, видео и анимации. Однако цель данной работы не создавать редактор, с огромным количеством возможностей, выводящими его за рамки растровой графики, а создания простого редактора с базовыми возможностями редактирования изображений.

Данная работа демонстрирует написание небольшого оконного Windows приложения на примере графического редактора и решение проблем, встречающихся в ходе его написания. Для написания используются только стандартные Windows библиотеки. Приложение должно обеспечивать возможность открытия, сохранения и простой обработки изображений в наиболее популярных форматах и должно работать в Windows начиная с XP.

Работа может оказаться полезной для C/C++ программистов, желающих изучать программирование под Windows. В работе так же описываются алгоритмы, с помощью которых можно запрограммировать простейшие возможности рисования и обработки растров, следовательно, она имеет теоретическую ценность.

1.1 Методика

Для написания предложения используются стандартные функции Windows, для удобства использования, завёрнутые в классы. Используется язык программирования C/C++, однако для уменьшения размера стандартная библиотека C++ не используется. Для компиляции используется MinGW-w64—версия компилятора GCC для Windows, вместе с его библиотеками импорта, заголовочными файлами и прочими утилитами из того же пакета с возможностью компиляции x86 и x86-64 кода. (10) Вкачестве редактора исходного кода используется Notepad++, для генерации ресурсов используется ResEdit. (11) (12) Для создания рисунков и диаграмм используется MS Paint и средство UML моделирования—Rational Rose.(13)

1.2 Краткий обзор работы

В начале работы даётся краткое введение в используемый интерфейс прикладного программирования—Windows API.(14) После этого начинается описание процесса разработки графического редактора. Описывается способ открытия и сохранения изображений, а также формат их хранения в оперативной памяти для более быстрого вывода на экран, что требуется для подготовки изображения к редактированию. Далее идёт описание манипуляции с пикселями: получение информации о пикселях и описание возможных представлений цвета. Следующим этапом является разработка инструментов редактирования. Заключительный этап—написание фильтров.

2. Введение в Windows API

Windows API проектируются таким образом, чтобы быть независимыми от языка, хотя основным языком реализации является C/C++. Поэтому их можно использовать из разных языков программирования, начиная от низкоуровневых языков ассемблера и кончая такими высокоуровневыми языками как, например, Python. Однако основным языком написания Windows API приложений по-прежнему остаётся C/C++. Windows API не является объектно-ориентированным, чтобы не привязываться к ООП и расширить список поддерживаемых языков. Функции Windows являются довольно громоздкими, поэтому при программировании программисту наверняка захочется обернуть их в классы для большей структурированности кода. А это является хорошей практикой для начинающего программиста. Многие бывалые программисты советуют начинать программирование с более низкоуровневых вещей, и далее двигаться наверх, а не хвататься за более высокоуровневые вещи вроде COM и .NET.(15)

2.1 Основные положения

Как было сказано выше, Windows API не должны быть зависимы от конкретного языка программирования. Поэтому типы данных имеют своё название, общее для всех языков. Для C/C++ эти типы представляют синонимы стандартных типов данных, определённые через ключевое слово *typedef*. Таких типов очень много, причём большая их часть обозначает одно и то же. В качестве примера можно привести определение синонима 16 битного UNICODE-символа*: typedef wchar_t WCHAR;*. В названиях типов есть определённые правила. Названия типов пишутся прописными буквами, буква C обозначает неизменяемое значение, P или LP обозначают указатель (в новых версиях Windows между ними нет разницы).

Функции, принимающие строку, обычно имеют 2 версии: ANSI и UNICODE. Однако реализация написана с использованием UNICODE, а ANSI версии просто используют UNICODE версии и преобразование. Чтобы указать, какую версию использовать существует макрос UNICODE, без него используются ANSI версии, что не рекомендуется из-за необходимости неявного преобразования в UNICODE. Также имеются макросы для определения соглашения о вызове, например, WINAPI,

CALLBACK, которые обозначают стандартное для Windows API соглашение о вызове stdcall.

Для манипуляции с объектами Windows API используются специальные идентификаторы—дескрипторы, которые совсем не обязательно обозначают адрес объекта, хотя в некотором случае это так. Многие функции принимают структуры через указатель. Имена полям структур даются таким образом, что сначала идёт последовательность символов (префикс), указывающая на тип поля, а затем описание, то есть используется венгерская нотация. (16) В отличие от полей структур, имена функций используют CamelCase. (17)Windows API насчитывают огромное количество функций, структур, макросовв в разных заголовочных файлах, но к счастью многие из них включены в заголовочном файле *windows.h*, поэтому его включения уже достаточно для разработки.

Основной библиотекой, с которой компонуетсяwindows-приложение, является kernel32.dll, в которой реализуется работа с файлами, процессами, потоками, памятью, динамически-подключаемыми библиотеками, обработка ошибок. Для использования графического интерфейса необходима библиотека user32.dll. Для вывода 2-мерной графики gdi32.dll. Стандартные элементы управления и стандартные диалоговые окна реализованы в comctl32.dll и comdlg32.dll соответственно. Также есть библиотеки для работы с сетью, 3-мерной графикой, мультимедиа, взаимодействия с другими технологиями и оболочкой, однако они не требуются для разработки простого графического редактора.

2.2 Принцип взаимодействия окон

Все окна в Windows взаимодействуют посредством сообщений. Сообщения генерируются пользователем при использовании клавиатуры и мыши, системой или другими приложениями. Сообщения попадают в системную очередь сообщений, после которой доставляются окну-получателю, и программа может на них реагировать. Можно выделить несколько групп сообщений. Названия идентификаторов сообщений начинаются с *WM_*. Например, для перерисовки окна требуется обработка *WM_PAINT*, для реакции на нажатие кнопок клавиатуры *WM_KEYDOWN, WM_KEYUP, WM_CHAR*, для реакции на мышь имеются свои сообщения, меню и элементы управления генерируют *WM_COMMAND*. В Windows можно выделить окна верхнего уровня и

дочерние окна этих окон. Элементы управления также являются окнами, для которых оконный класс и оконная процедура определены системой. Это придаёт стандартный вид разным приложениям. Взаимодействие с элементами управления также осуществляется посредством сообщений, что является несколько необычным для тех, кто привык их рассматривать с точки зрения ООП, то есть вызывая методы для их настройки. В Windows API же придётся использовать функцию *SendMessage* и посылать определённые сообщения. Однако неплохо завернуть этот функционал в методы класса для представления элемента ввиде объекта, который сам определяет своё поведение и хранит данные с ним связанные.

2.3 Общая структура оконного приложения

Работу графического (оконного) приложения можно описать следующим образом:

1. Пользователь запускает приложение, и система вызывает функцию—точку входа. При написании приложения с использованием стандартной библиотеки C эту функцию предоставляет библиотека, и при её выполнении происходит инициализация, необходимая для работы библиотеки, вызываются конструкторы глобальных переменных, разбор входящей командной строки. После происходит вызов представленной программистом функции main (для консольных приложений) или *WinMain* (для графических приложений).

```
int WINAPI WinMain(HINSTANCE instance, HINSTANCE
previousinstance, LPSTR cmdline, intcmdshow)
//Аргументами принимает дескриптор экземпляра
//приложения(загруженного в память exe-модуля),2 аргумент не
//используется в новых Windows, 3 аргумент — командная строка,
//4 аргумент предпочтительные флаги показа окна.
```

2. Внутри *WinMain* создаётся оконный класс (заполняется структура *WNDCLASS* или *WNDCLASSEX*). Созданный оконный класс регистрируется в системе вызовом *RegisterClass* или *RegisterClassEx*. Создаётся и показывается окно вызовом *CreateWindow* или *CreateWindowEx*.

3. Далее создаётся цикл, в котором приложение извлекает сообщение из очереди сообщений окна с помощью *GetMessage* и отправляет его на обработку вызовом

DispatchMessage. Перед *DispatchMessage* обычно вызывается *TranslateMessage* для генерации сообщений, содержащих UNICODE коды на основе виртуальных кодов от сообщений клавиатуры и выбранной пользователем раскладки.

```
MSGmsg;   //структура с информацией сообщения
while(GetMessage(&msg,NULL, 0, 0)){
     TranslateMessage(&msg);
     DispatchMessage(&msg);
}
```

4. *DispatchMessage* приводит к тому, что система вызывает так называемую оконную процедуру, которая указывается в оконном классе. В этой процедуре программист определяет, как реагировать на полученные сообщения или использовать стандартную обработку через *DefWindowProc*.

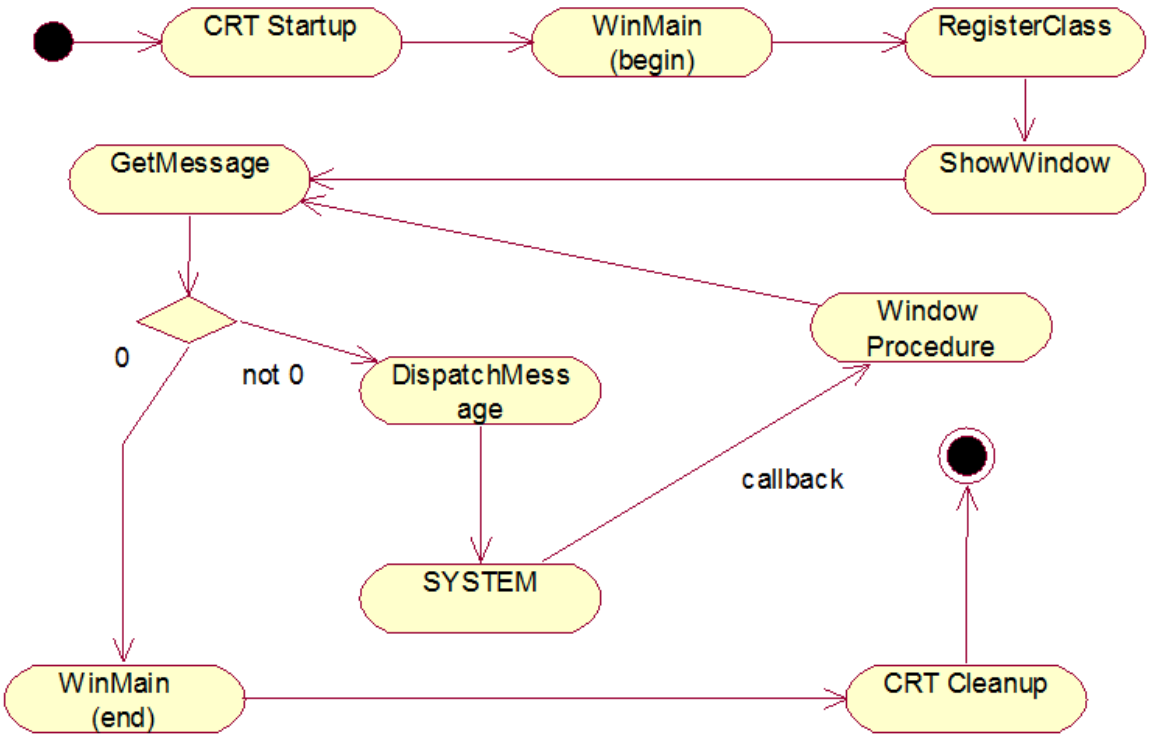
```
LRESULT CALLBACK WndProc(HWND window, UINT msg, WPARAM wparam,
LPARAM lparam){
     switch(msg){
          case WM_CLOSE:
               DestroyWindow(window);
               break;
          case WM_DESTROY:
               PostQuitMessage(0);
               break;
           ...
          default:
          returnDefWindowProc(window, msg, wparam, lparam);
     }
}
//Аргументами принимает дескриптор окна, которому
//предназначается сообщение, идентификатор сообщения, 2 32-
//битных параметра сообщения.
```

4. При вызове *PostQuitMessage* посылается сообщение, при выборе которого из очереди *GetMessage* возвращает 0, что приводит к выходу из цикла и завершению *WinMain*. Если используется C библиотека, то происходит вызов её завершающей функции.

2.4 Ресурсы и стандартные диалоговые окна

Ресурсы представляют удобный способ для определения таких элементов графического интерфейса как меню, элементы управления, диалоговые окна, акселераторы. Также через ресурсы часто определяются курсоры, изображения, шрифты, манифест, информация о версии, таблицы символов. Вообще ресурсом можно сделать любой файл. Ресурсы представляют собой информацию в одной из секций исполняемого файла или динамической библиотеки. Объектный файл с ресурсами компилируется компилятором ресурсов на основе текстового файла описания ресурсов, после чего компонуется вместе с кодом в исполняемый файл или динамическую библиотеку. Другим средством, облегчающим разработку, являются стандартные диалоговые окна. Windows представляет стандартные диалоговые окна, например, для выбора файла, шрифта, цвета.

Рис1. Простейшее оконное приложение (activity diagram).



3. Открытие и сохранение файла изображения

Наиболее популярными форматами для хранения однокадровых изображений являются такие форматы как JPEG, PNG, BMP. BMP является родным для Windows и часто вообще не использует алгоритм сжатия, хотя поддерживает сжатие без потерь. JPEG и PNG форматы популярны для WEB, так как поддерживают сжатие, что позволяет значительно уменьшить размер файла. BMP может использовать алгоритм RLE (run-length encoding), то есть заменять повторяющиеся последовательности байтов на его значение и количество повторений. (18) Также можно использовать алгоритм Хаффмана, суть которого в том, что каждому символу исходного алфавита ставится во взаимно-однозначное соответствие некая последовательность битов, причём ни одна такая последовательность не является префиксом другой, и длина последовательности тем меньше, чем чаще символ встречается в кодируемых данных. (19) PNG использует более сложный алгоритм сжатия без потерь DEFLATE, который основан на том, что входные данные разбиваются на блоки, и для каждого блока применяется статический алгоритм Хаффмана или динамический алгоритм Хаффмана или не применяется кодирование вообще. (20) JPEG позволяет добиться наименьшего размера, однако использует алгоритм сжатия с потерями на основе дискретного косинусного преобразования. (21)

3.1 Открытие

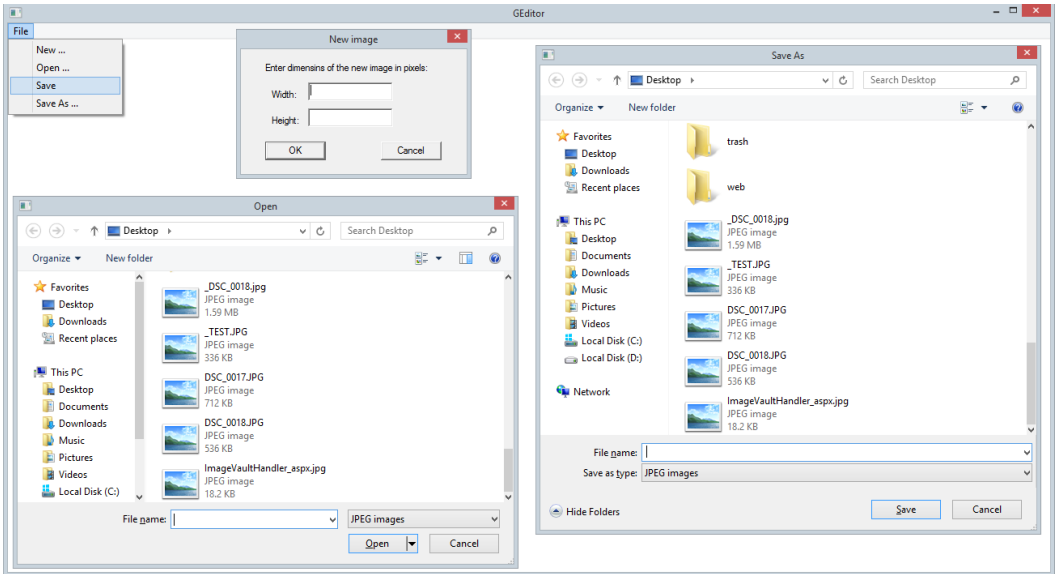
GDI+ поддерживает набор декодировщиков для открытия форматов BMP. PNG, JPEG, TIFF, GIF. Последние 2 формата поддерживают хранение нескольких кадров в одном файле, но наш редактор должен работать с однокадровыми изображениями, поэтому их поддержка необязательна. GDI+ предоставляет класс *Bitmap*, один из конструкторов которого принимает путь к файлу и загружает изображение на основе этого пути. Выбор пути удобно сделать через стандартное диалоговое окно выбора файла вызовом *GetOpenFileName*, которая позволяет ограничитьсписок подлежащих открытию файлов по их расширению. Команду открытия принято обычно помещать на панель меню в крайнее левое подменю "Файл". Проблемой является то, что при открытии изображения GDI+ блокирует файл, и сохранить в то же место его уже не удастся. Для решения этой проблемы можно создать ещё один объект *Bitmap* с такими же

размерами, как и у первого, получить его контекст, используя конструктор *Graphics*, принимающий *Bitmap*, и нарисовать первый *Bitmap* на новом. После этого можно удалить старый, что приведёт к разблокировке файла. Новое изображение создаётся с глубиной цвета 32 бит, что позволяет использовать дополнительные 8 бит в качестве альфа-канала. Также необходимо добавить пункт меню для подменю "Файл", который создаёт изображение заданного размера. Для этого можно использовать диалоговое окно, определённое в ресурсах, содержащее поля для ввода ширины и высоты изображения.

3.2 Сохранени

GDI+ также поддерживает кодировщики для BMP, PNG, JPEG, TIFF, GIF, что позволяет сохранять изображения также легко, как и открывать. Для этого достаточно вызвать метод *Save* изображения, передав ему идентификатор требуемого формата (CLSID). Команды для сохранения помещаются в то же подменю, что и команда открытия. Делается 2 стандартные команды: "Сохранить", которая перезаписывает ранее открытый файл, и команда "Сохранить как", которая позволяет, используя диалоговое окно выбора файла, указать место и имя файла для сохранения. Для создания диалогового окна сохранения можно использовать функцию *GetSaveFileName*, которая также позволяет выбрать предпочтительное расширение файла, на основе которого создаётся глобальный идентификатор *CLSID*, который передаётся методу *Image::Save*.

Рис2. Диалоговые окна создания, открытия и сохранения.



4. Работа с изображением.

В GDI существует 2 типа растровых изображений: зависящие от устройства растры (DDB) и не зависящие от устройства растры (DIB). Не зависящие от устройства растры—это тоже самое, что хранится в BMP файлах. Определяются структурой *BITMAPINFO*, которая содержит информацию о изображении вроде высоты, ширины, байтов на пиксель и массивом структур *RGBQUAD*, которые содержат значения четырёх каналов. Позволяют получить доступ к пикселям через массив структур *RGBQUAD*. Зависящие от устройства растры определяются структурой *BITMAP*, которая хранит размеры, глубину цвета, массив данных, представляющий пиксели в формате устройства, прямого доступа к пикселям не получить. Для вывода DIB на определённое устройство можно, например, использовать функцию *StretchDIBits*, однако вывод таким способом относительно медленный, поэтому для производительного вывода лучше использовать DDB. Для вывода DDB создаётся контекст в памяти, в который выбирается DDB, то есть он используется как поверхность для рисования. Для вывода на экран используется метод, называемый блиттингом, суть которого в быстром копировании части изображения в видеопамять. Есть ещё один тип, называемый DIB-секциями, который обладает преимуществами предыдущих, а именно он позволяет организовать прямой доступ к пикселям и позволяет использовать себя как поверхность для рисования, что даёт возможность использовать блиттинг.

4.1 Получение доступа к пикселям

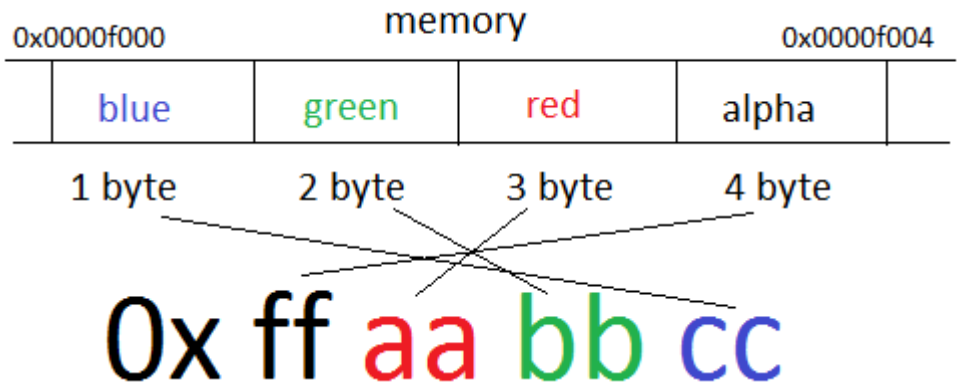
На предыдущем шаге изображение было загружено и сохранено в виде GDI+ класса *Bitmap*. Он поддерживает несколько способов для доступа к пикселям. Можно использовать методы *GetPixel* и *SetPixel*, однако этот метод чрезвычайно неэффективный и совсем не подходит для интенсивной манипуляции пикселями. Использовать метод *LockBits*, что позволяет получить доступ к массиву пикселей и затем *UnlockBits*, чтобы применить изменения. Второй метод намного быстрее первого, однако основная проблема в этом случае связана не с получением доступа к пикселям, а со скоростью вывода изображения на экран. Для графического редактора нужна

максимальная скорость, поэтому лучше всего использовать DIB-секцию. GDI+Bitmap на основе DIB-секции создать легко, для этого служит один из его конструкторов. Чтобы создать DIB-секцию из GDI+ Bitmap, используем структуру *BITMAPINFOHEADER*, инициализированную на основе параметров изображения, после чего скопируем пиксели изображения в массив DIB-секции. Теперь мы можем использовать этот массив для максимально быстрого чтения и редактирования значений пикселей.

4.2 Вывод изображения на экран

Выводить изображение лучше всего при получении события *WM_PAINT* окном. Чтобы вывести изображение на экран надо создать предварительно совместимый контекст в памяти вызовом *CreateCompatibleDC* и выбрать в него DIB-секцию. Это лучше всего делать при загрузке или создании изображения. Теперь при получении события перерисовки окна можно использовать функцию *StretchBlt*, которая осуществляет блиттинг из заданного прямоугольника источника в заданный прямоугольник приёмника, используя трансформацию и выбранную растровую операцию.

Рис3. Порядок расположения цветов в памяти.



5. Расчёт статистики и показ информации о пикселе

Графический редактор должен показывать информацию о пикселях. Обычно анализируется пиксель, находящийся под курсором. Например, в программе Abode Photoshop в окне "Info" можно посмотреть размеры, позицию пикселя на изображении, значения красного, зелёного и синего каналов, эквивалентные значения в цветовой модели CMYK. В создаваемом редакторе также будет добавлен данный функционал. Кроме этого добавится показ эквивалентных значений в цветовой модели HSV. Также показывается статистика для красного, зелёного, синего каналов ввиде гистограммы. При движении курсора информация о пикселе обновляется, гистограммы пересчитываются при изменении. Окно, выводящее информацию и статистику, обрабатывается в отдельном потоке с низким приоритетом.

5.1 Создание гистограмм для RGB каналов

Требуется сгенерировать гистограмму, которая показывает частоту встречи каждого из возможных значений для красного, зелёного и синего каналов. Значения частот нормированы относительно максимальной частоты, что позволяет корректно масштабировать гистограмму по вертикали.

Рис4. Внешний вид диаграмм для RGB каналов.

Чтобы нарисовать гистограммы, необходим массив из 256 элементов, содержащий частоту встречи каждого из возможных значений канала, причём для каждого канала необходим свой массив. Чтобы заполнить массивы необходимо пройтись по всем пикселям изображения. В теле цикла для получения значения каждого из каналов

используются битовая маска, операция побитового "И" и побитовый сдвиг вправо. Полученные значения используются в качестве индексов в созданных массивов для инкрементирования элемента массива на единицу. Так же в этом же цикле ищется максимальное значения частоты для каждого из каналов: переменная, созданная перед циклом, становится равной значению частоты для значения канала текущего пикселя, если это значение больше значения переменной.

```
redIndex=(data[i]&0x00ff0000)>>16; //позиция для красного
greenIndex=(data[i]&0x0000ff00)>>8;//позиция для зелёного
blueIndex=(data[i]&0x000000ff);    //позиция для синего
```

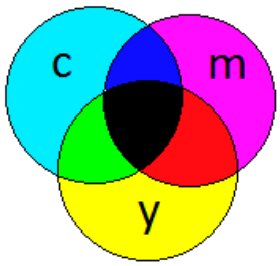
Теперь, после выполнения цикла для всех пикселей изображения можно нарисовать вертикальную линию стандартными средствами GDI вызывая *MoveToEx* и *LineTo* для каждого элемента массива с посчитанной статистикой. Длину линии легко определить, зная частоту для оттенка, максимальную частоту и высоту гистограммы.

```
lineLength=stats[i]/max*histogramHeight; //высота столбика
```

5.2 Преобразование RGB в CMYK

CMYK (Cyan, Magenta, Yellow, Keycolor)—цветовая схема, используемая преимущественно в полиграфии. (22) Значения данной цветовой схемы соответствуют голубому, пурпурному и жёлтому цветам. В отличие от RGB, где все цвета в сумме образуют белый, в CMYK комбинация CMY даёт чёрный цвет. Голубой цвет в CMYK даёт комбинация зелёного и синего в RGB, пурпурный—комбинация синего и красного, жёлтый—комбинация зелёного и красного. Наличие четвёртое значение соответствует дополнительному чёрному, что связано с тем, что в полиграфии при комбинации голубой, пурпурной и жёлтой краски получается не чёрный, а грязно-тёмный цвет. Значения CMYK часто выводятся в процентах.

Рис5. Формула преобразования в CMYK.(22)



$K = 1-\max(R, G, B)$ K=1-min(C',M',Y')

$C = (1-R-K) / (1-K)$ C = 1-R/max(R,G,B)

$M = (1-G-K) / (1-K)$ M = 1-G/max(R,G,B)

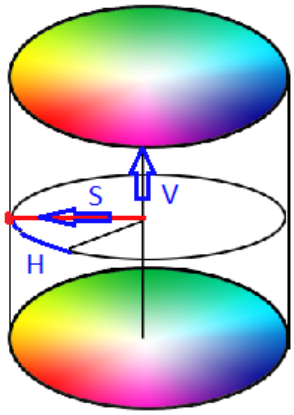
$Y = (1-B-K) / (1-K)$ Y = 1-B/max(R,G,B)

В формулах используем действительные значения от 0 до 1.0. При отсутствии недостатка, связанного с полиграфией (при рассмотрении только 3 значений), чтобы получить голубой, достаточно из 1.0 вычесть красный, чтобы получить пурпурный—вычесть зелёный, чтобы получить жёлтый—вычесть синий. При введении 4 значения, формулы становятся как на рисунке выше. Видно, что K соответствует дополнению минимального значения 3-значного CMY, а при расчёте значений 4-значного CMYK красный, зелёный и синий нормируются относительно максимального из них.

5.3 Преобразование RGB в HSV

Ещё одной распространённой моделью является HSV (Hue, Saturation, Value). Цвет в данной модели изображается в цилиндрических координатах. (23) Длиной радиус-вектора является насыщенность (saturation), которая характеризует "сочность цвета". Полярным углом является значение тона (hue), который плавно проходит по всем цветам радуги и промежуточным цветам. Значением высоты является яркость (value). Тон часто выражается в градусах, насыщенность и яркость в процентах.

Рис6. Формула преобразования в HSV.(23)



$$H = \begin{cases} 0, \text{если } MAX = MIN \\ 60 \times \dfrac{G-B}{MAX-MIN} + 0, \text{если } MAX = R \text{ и } G \geq B \\ 60 \times \dfrac{G-B}{MAX-MIN} + 360, \text{если } MAX = R \text{ и } G < B \\ 60 \times \dfrac{B-R}{MAX-MIN} + 120, \text{если } MAX = G \\ 60 \times \dfrac{R-G}{MAX-MIN} + 240, \text{если } MAX = B \end{cases}$$

$$S = \begin{cases} 0, \text{если } MAX = 0; \\ 1 - \dfrac{MIN}{MAX}, \text{иначе} \end{cases}$$

$$V = MAX$$

В приведённой на рисунке формуле min и max это минимальное и максимальное значение среди значений красного, зелёного, синего. Значению яркости соответствует максимальное значение каналов. Насыщенность рассчитывается как относительная разность наибольшего и наименьшего значений. Значение угла тона рассчитывается по-разному для своего 120-градусного сегмента, добавлением смещения относительно середины сегмента.

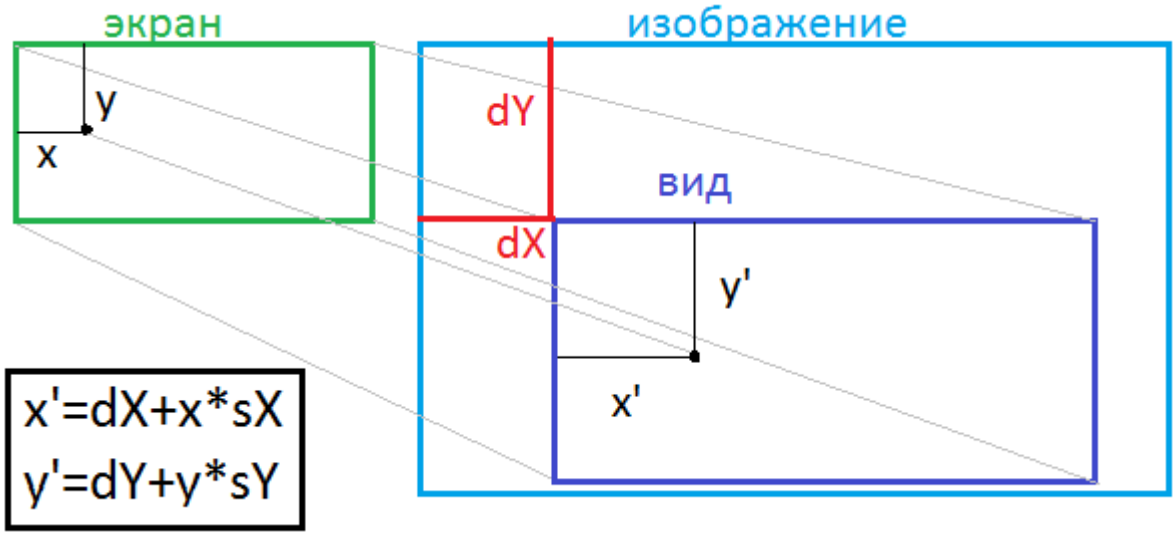
6. Разработка инструмента манипуляции видом.

Для графического редактора важно наличие инструмента, позволяющего менять масштаб и перемещать вид по изображению, чтобы можно было рассматривать разные его части. В Photoshop, например, такими инструментами являются "рука" и "масштаб". В разрабатываемом редакторе масштабирование и перемещение вида будет осуществляться одним и тем же инструментом. Перемещение мыши при нажатой левой кнопке приводит к перемещению вида. Перемещение мыши по вертикали, при нажатой правой кнопке, приводит к масштабированию.

6.1 Общая идея

Вывод изображения на экран осуществляется функцией *StretchBlt*, которая позволяет задать прямоугольник на источнике и спроецировать его на заданный прямоугольник в приёмнике. При уменьшении масштаба поведение функции по умолчанию приводит к появлению артефактов на изображении. Для устранения этого можно установить режим обработки ситуации, когда одному пикселю в источнике соответствует несколько в приёмнике. Для этого можно вызвать *StretchBltMode*, указав флаг режима. Лучшее качества даст флаг *HALFTONE*, что приведёт к усреднению значений конфликтующих пикселей, однако в разрабатываемом приложении с целью производительности используется *COLORONCOLOR*, который оставляет только одно значение. Наличие возможности масштабирования и перемещения вида приводит к тому, что координаты клиентской области и координаты на изображении больше не совпадают. Введено 4 переменные, хранящие смещение вида по осям и отношение вида на изображении к прямоугольнику клиентской области: dX, dY, sX, sY. Теперь координаты на изображении можно узнать по координатам на экране, умножив их на коэффициент масштаба (sX и sY) и добавив смещение (dX и dY). Это даёт возможность проецировать позицию мыши на рисунок, что необходимо, например, для получения информации о пикселе. Также это позволяет найти прямоугольник-источник для блиттинга.

Рис 7. Проецирование клиентских координат на изображение.



6.2 Изменение параметров вида

Чтобы теперь настроить перемещение вида и масштабирование достаточно настроить обработку событий мыши, чтобы изменялись dX, dY, sX, sY. При нажатии левой или правой кнопки мыши значение позиции курсора заносится в oldX, oldY и позже при его перемещении с нажатой кнопкой эти значения обновляются. При движении мыши, например, вправо, чтобы показываемое изображение смещалось вправо, вид должен смещаться влево, с учётом масштаба и наоборот. Пусть x, y—текущие координаты курсора, тогда выражение для смещения будут следующие:

```
dX+=sX*(oldX-x);
dY+=sY*(oldY-y);
//Теперь необходимо обновить предыдущее значение позиции.
oldX=x;
oldY=y;
```

Масштабирование требуется реализовать таким образом, чтобы изображение центрировалось относительно места нажатия. Для правильного центрирования необходимо помнить место нажатия, поэтому оно сохраняется в переменных initialX, initialY. При движении мыши вниз с нажатой правой кнопкой требуется увеличить масштаб, то есть уменьшить коэффициенты sX и sY.Это можно реализовать следующим образом:

```
doubledsY=y-oldY;                    //смещение мыши по верт.
doublemult=1.0+dsY/100.0;            //множитель, который будет
     //использоваться для рассчёта масштабирования. Множитель
     //увеличится на 100% если мышь сместить вниз на 100
     //пикселей.
if(sX*mult<=5.0 &&sX*mult>=0.1){     //ограничение масштаба
     //между 20% и 1000%
     dX-=(1.0-sX)*initialX;          //отменяем предыдущее
          //центрирование
     dY-=(1.0-sY)*initialY;
     sY*=mult;                       //обновляеммасштаб
     sX*=mult;
     dX+=(1.0-sX)*initialX;          //устанавливаем новое
          //центрирование
     dY+=(1.0-sY)*initialY;
}
```

Как видно из кода, центрирование заключается в дополнительном смещении вида, чтобы пиксель, соответствующий точке нажатия, оставался на месте. Эта точка смещается на величину равную разности её конечной координаты sX*x, sY*y и начальной координаты x, y. Чтобы это скомпенсировать и добавляется смещение, помноженное на -1.

7. Рисование линий

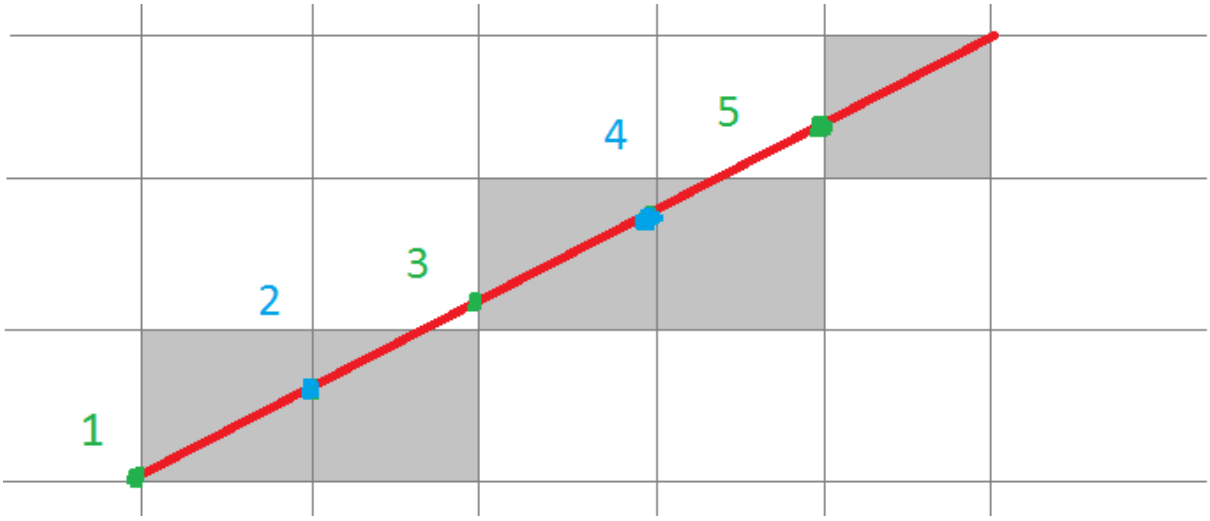
Рисование линий является неотъемлемой частью любого растрового графического редактора. Запрограммировав рисование прямой линии можно рисовать прочие фигуры, из них состоящие, как например прямоугольники. Линии можно проводить сглаженные и неслаженные. Линии без сглаживания рисуются быстрее, чем линии со сглаживанием, так как используют более простой алгоритм. Можно получить сглаженную линию из неслаженной, используя мультисэмплинг. Линия рисуется на растре большего разрешения, после чего трансформируется до нужного размера. В результате этого пиксели на границах имеют промежуточную прозрачность, что создаёт эффект сглаженной линии. В данной работе используется рисование не сглаженной линии и на основе его созданы алгоритмы рисования толстой линии и рисования прямоугольника. Перед проведением окончательного варианта линии, при удержании кнопки мыши и её перемещении рисуется линия, единичной толщины, демонстрирующая, где будет проходить окончательный вариант. Чтобы линию было всегда видно используется инверсия её пикселей. Это также позволяет сделать "отмену" этой вспомогательной линии, инвертировав пиксели ещё раз.

7.1 Алгоритм Брезенхэма

Для рисования неслаженных линий часто используется один из старейших алгоритмов—алгоритм Брезенхэма. (24) Данный алгоритм является очень быстрым, так как количество итераций его цикла равно количеству закрашенных точек. В полученной линии число закрашенных точек равно длине большей проекции линии. Поэтому для каждого значения x или y, в зависимости от того, что больше, определяется другая координата и точка закрашивается. На следующем рисунке наглядно показано условие закрашивания точки. Если значение смещения линии по меньшей координате превысит 0.5, то следующий пиксель имеет значение меньшей координаты на 1 больше. В точках 1, 3, 5 этого не происходит, и следующий пиксель выводится на том же уровне по вертикали. В точках 2, 4 это происходит, и следующий пиксель рисуется на уровень выше. Когда происходит смещение, то значение счётчика смешения уменьшается на 1 и, следовательно, для следующего смешения меньшей

координате линии надо покрыть уже больше, чем 0.5. Например, на рисунке в точке 2 смещение превысило 0.5 (в точке 1 начальное смещение равно 0) и стало, например, 0.6, после чего счётчик смещения стал -0.4 и полученная разница в 0.9 (до 0.5) покрылась в точке 4.

Рис8. Работа алгоритма Брезенхэма.



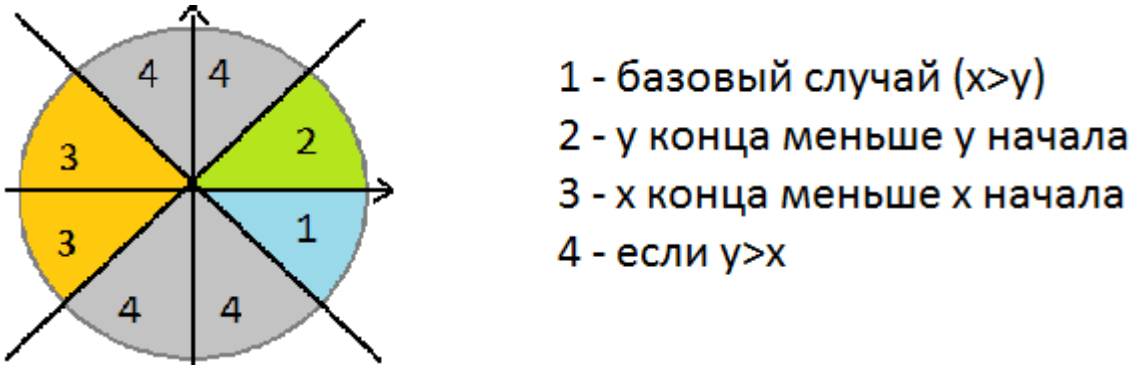
Алгоритм принимает значения координат начала и конца линии и устанавливает значение цвета color в массиве пикселей pixels изображения шириной width. Работа данного алгоритма немного отличается для различных значений координат начала и конца. Всего можно выделить 8 случаев. За базовый случай возьмём такие значения координат, что проекция линии на ось X больше, чем на ось Y и у конца больше чем у начала и x конца больше чем x начала. Тогда цикл алгоритма выглядит следующим образом (x1, y1, x2, y2—координаты начала и конца):

```
double threshold=0;int y=y1;
double delta=fabs((y2-y1)/(x2-x1));
for(int x=x1;x<=x2;x++,threshold+=delta){
     pixels[y*width+x]=color; //закрашиваемпиксель
     if(threshold>=0.5){y++; threshold-=1;}
}
```

Следующий рисунок демонстрирует оставшиеся варианты. Во 2 секторе вместо инкрементирования y на 1 надо его декрементировать на 1. Для построения линии в 3 секторе достаточно поменять начало с концом. Для построения в 4 секторе делается то

же самое, но вместо поиска позиции y для каждого x делается всё в точности наоборот—для каждого y ищется x.

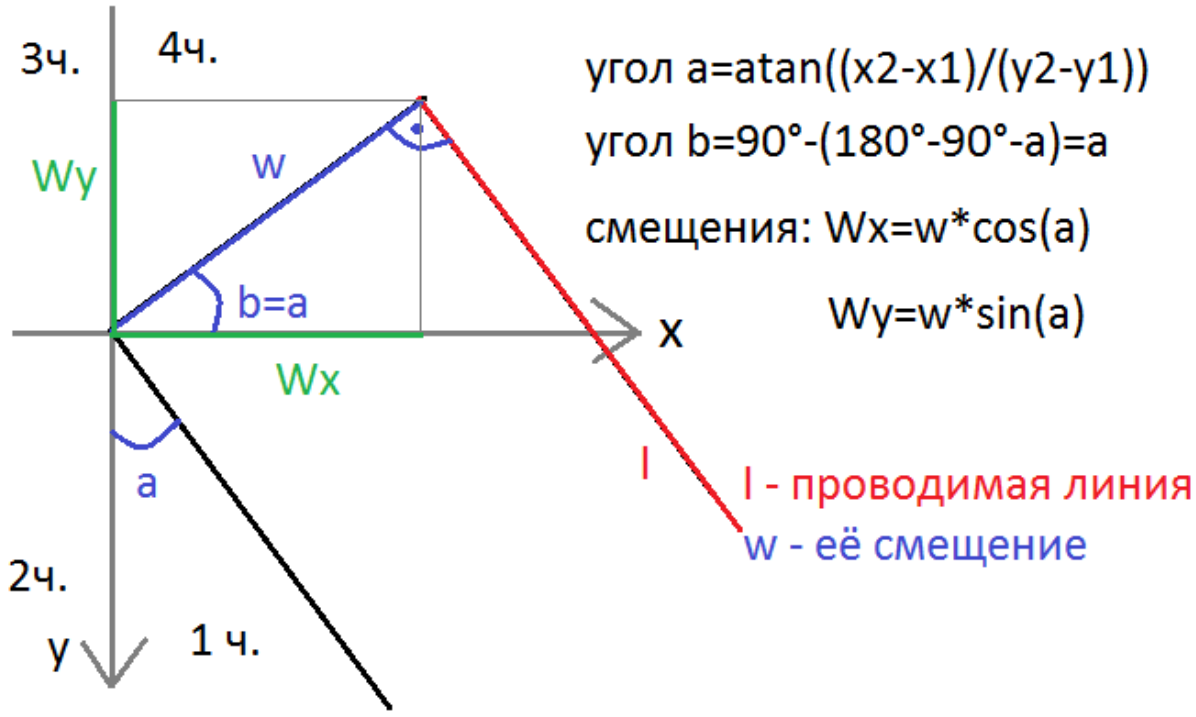
Рис9. Возможные варианты координат конца и начала.



7.2 Рисование толстой линии

Для того, чтобы толщина линии была больше, чем 1 пиксель, можно проводить несколько параллельных линий. Однако при таком подходе появляется 2 проблемы: нужно рассчитать координаты конца и начала линий, чтобы край линии был ей перпендикулярен, не все пиксели внутри толстой линии закрашиваются. Для решения второй проблемы можно проводить для каждой линии, кроме последней ещё одну, но на 1 выше. Это закрасит оставшиеся незакрашенными пиксели внутри линии. Следовательно, количество проведённых линий меньше удвоенной ширины линии на 1. Линии проводятся в цикле, смещение конца равно смещению началу, так как линии параллельны. Чтобы найти смещение по оси X надо косинус угла наклона линии умножить на текущее значение счётчика итераций цикла, который проходит от 0 до ширины линии не включительно. Чтобы заданным координатам соответствовала середина линии, от значения счётчика отнимается половина ширины линии. Для нахождения смещения по оси Y разность счётчика и половины ширины умножается на синус угла наклона. Таким образом линия проводится во 2 и 4 четвертях (где проекции линии на координатные оси имеют разные знаки). Чтобы провести линию в 1 и 3 четвертях (где проекции имеют одинаковые знаки) нужно смещения по оси X умножить на -1.

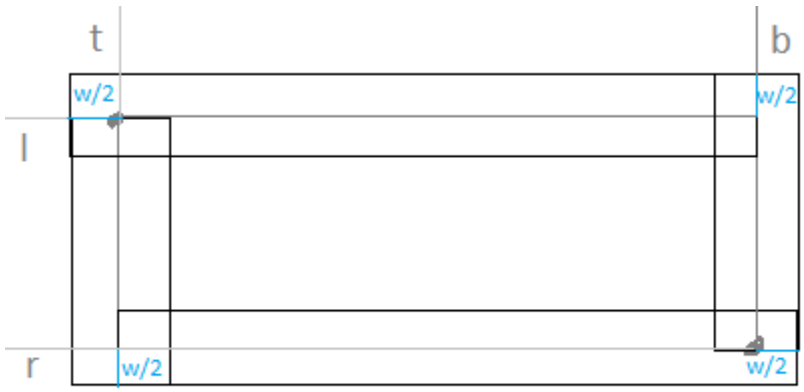
Рис10. Расчёт координат для рисования толстой линии (1 четверть).



7.3 Рисование прямоугольника

Часто прямоугольник задаётся координатами верхнего левого и правого нижнего угла. Уже имеется алгоритм рисования толстой линии, чтобы нарисовать прямоугольник достаточно провести 4 линии. Если обозначить координаты, которыми задаётся прямоугольник как l, t, r, b, то линии, его образующие, задаются следующими координатами: (l-w/2, t, r, t), (r, t-w/2, r, b), (r+w/2, b, l, b), (l, b+w/2, l, t). Если l>r или t>b, то меняем их местами.

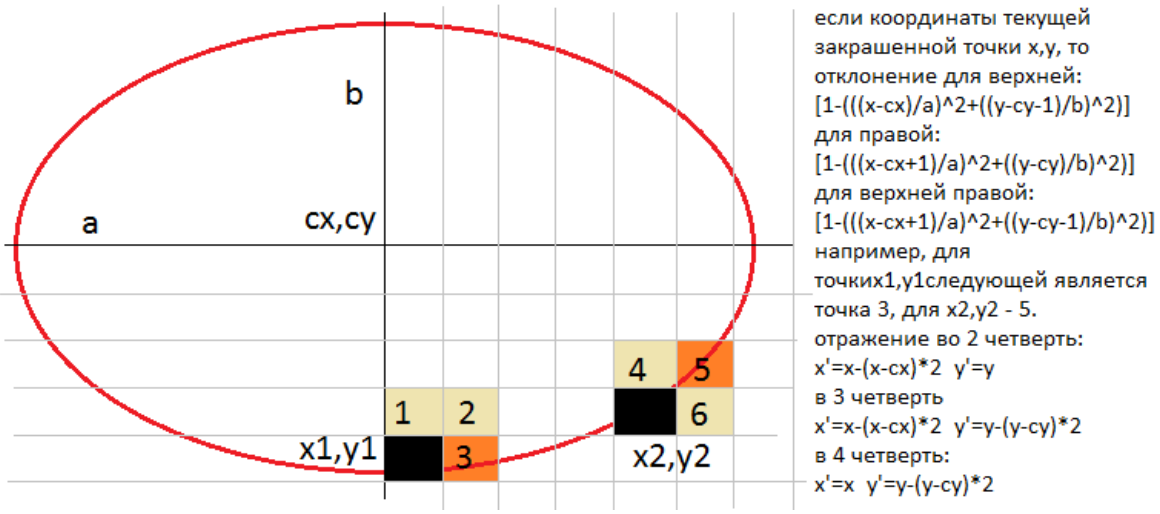
Рис11.Формирование прямоугольника.



7.4 Рисование эллипса

Для задания Эллипса используется такой же способ, как и для задания прямоугольника—по 2 углам, задающим прямоугольник, в который вписан эллипс. Полуоси можно найти, поделив разницу координат пополам. Прибавив длину полуосей к координатам начала можно найти координаты центра. Для построения эллипса за начальную точку берётся самая нижняя его точка, после чего строится дуга 1 четверти. В первой четверти к эллипсу может принадлежать одна из 3 точек: правее, выше, правее и выше. Чтобы определить, какая точка подходит, её координаты подставляются в уравнение эллипса, и считается величина отклонения от 1. После того как точка найдена, её координаты симметрично отображаются относительно полуосей, чтобы закрасить точки в остальных четвертях.

Рис12. Построение эллипса.



7.5 Рисование толстого эллипса

Для того чтобы нарисовать эллипс не единичной толщины, можно использовать приём, аналогичный рисованию линии. Если координаты начала больше, чем координаты конца, то они меняются местами. В цикле значения координат конца увеличиваются, а координаты конца уменьшаются, после чего проводится эллипс. Таким образом, эллипс как бы утолщается внутрь.

8. Заливка

Наряду с рисованием линий и фигур, их заливка также является важной возможностью растрового графического редактора. Редактор должен содержать инструмент, который позволяет проводить линию вслед за движением мыши. Подобный инструмент обычно называется кистью или карандашом. Также редакторы содержат инструмент заливки, который позволяет заливать указанный пиксель и все с ним смежные пиксели указанным цветом. В готовых редакторах есть возможность заливки со сглаживанием краёв, однако в данной работе используются простейшие приёмы заливки прямоугольной и эллиптической областей без сглаживания. Для демонстрации алгоритмов рисования линий и заливок создано 3 инструмента. Первый при нажатии на левую кнопку рисует линию указанной толщины (используя алгоритм из предыдущей главы), при нажатии на правую рисует прямоугольник и заливает его. Второй инструмент при нажатии на левую кнопку представляет собой кисть, при нажатии на правуюрисует эллипс и заливает его. Третий инструмент при нажатии на точку изображения заливает её и все ей смежные указанным цветом.

8.1 Заливка прямоугольника

Прямоугольник, как и раньше, задаётся по двум точкам, и если координаты конечной точки меньше, то точки обмениваются местами. Чтобы залить прямоугольник достаточно изменить цвет всех пикселей внутри него на заданный цвет. Для этого во внешнем цикле перебираются все строки прямоугольника, а во внутреннем все пиксели строки.

```
for(int y=t;y<b;y++){
      for(int x=l;x<r;x++){
            pixels[y*width+x]=color;
      }
}
```

8.2 Заливка эллипса

Для заливки эллиптической области используется метод, похожий на заливку прямоугольника. Однако в отличие от прямоугольника, эллипс не полностью покрывает просматриваемую область. Чтобы пиксель был закрашен, требуется также чтобы пиксель лежал не за пределами эллипса. Чтобы проверить это условие можно представить другой эллипс, которому принадлежит данная точка и который имеет такой же центр и такой же эксцентриситет, как и заданный эллипс. И если полуось меньше соответствующей полуоси заданного эллипса, то точку нужно закрасить.

Допустим, эллипс задан в координатах, исходящих из его центра. x, y—координаты исследуемой точки, a, b—горизонтальная и вертикальная полуоси заданного эллипса, A—искомая горизонтальная полуось,тогда используя равенство эксцентриситетов и совпадение центров можно записать уравнение и найти полуось:

Рис13. Нахождение полуоси вспомогательного эллипса

$$\frac{x^2}{A^2} + \frac{y^2}{\left(A\frac{b}{a}\right)} = 1 \Rightarrow A = \sqrt{x^2 + \left(\frac{y\,a}{b}\right)^2}$$

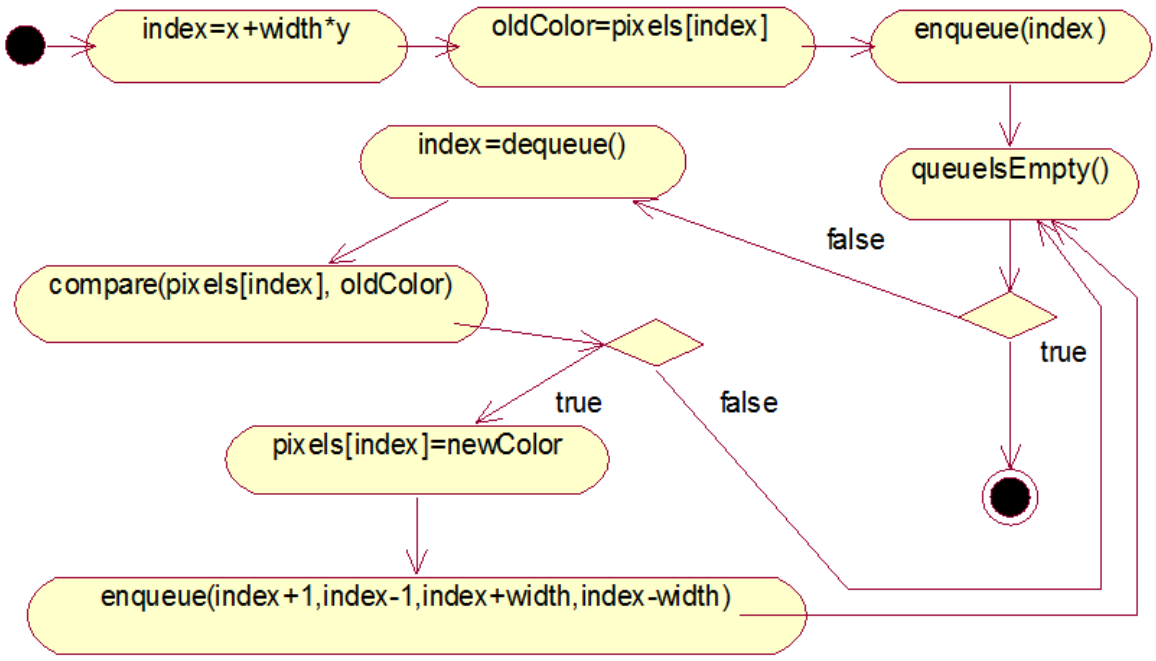
8.3 Кисть

Для создания простейшего поведения кисти достаточно рисовать круги для каждого сообщения мыши при нажатой кнопки. Однако расстояние между положениями мыши при достаточно быстром её перемещении может быть довольно большим, и поэтому также необходимо рисовать линию, соединяющую эти точки. То есть для каждой точки вызывается функция заливки эллипса, для которой координатами начала служит положение мыши с вычтенным значением радиуса, а конца—с прибавленным значением радиуса. После этого из текущей точки в предыдущую проводится линия с толщиной равной удвоенному значению радиуса.

8.4 Заливка смежных пикселей

Инструмент заливки обычно изображается в виде ведра, что отображает его поведению. При применении его к пикселю изображения его цвет, а также цвет смежных с ним пикселей того же цвета меняется на указанный. Также реализована возможность указания допуска—значения характеризующего, будет ли текущий проверяемый пиксель приравнен по цвету к первому пикселю или нет. Здесь это значение равно максимальной разнице значений каждого из каналов сравниваемых пикселей. Для определения пикселей, сохраняется цвет первого пикселя для дальнейшего сравнения с ним, а далее выполняется основной цикл, в котором из очереди выбирается один пиксель и сравнивается с цветом первого, если условие выполнено, то пиксель закрашивается, а затем правый, левый, нижний, верхний пиксели помещаются в очередь. Цикл работает, пока очередь не пуста. Чтобы не допустить прохода по уже закрашенным пикселям, дополнительным условием ставится то, чтобы пиксель не имел указанный цвет.

Рис14. Работа алгоритма заливки.



9. Попиксельная обработка

Почти все графические редакторы содержат функционал, позволяющий проходить по всем пикселям изображения (или указанной области) и определять их цвет на основе позиции пикселя и значений остальных пикселей. Тоесть на основе старого изображения получается новое, в котором для каждого пикселя выполняется некоторая функция f(x, y, pixels), возвращающая новое значение. Данный подход даёт огромные возможности по преобразованию изображений. Например, можно реализовать трансформацию изображения, можно изменять параметры цвета, например, яркость, контрастность, насыщенность и так далее. Можно также реализовать менее тривиальные функции, которые в графических редакторах часто называются фильтрами. В Adobe Photoshop в качестве примеров фильтров можно привести фильтры размытия, искажения, резкости, а также фильтры, эмулирующие рисование разными средствами на разных поверхностях. Можно также организовать обработку изображения по частям. Для этого изображение делится на прямоугольники одинаковой высоты, а высота последнего прямоугольника может быть выше, чтобы компенсировать оставшиеся строки. Далее запускается обработка каждой части в своём потоке. Количество потоков для максимальной эффективности равно количеству логических процессоров. Это число можно узнать из переменной окружения *NUMBER_OF_PROCESSORS*.

9.1 Трансформация

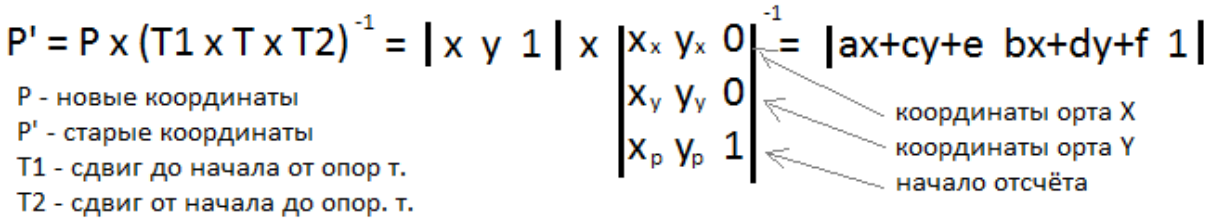
Для реализации таких базовых трансформаций, как сдвиг, масштабирование, поворот, наклон удобно использовать линейные преобразования с помощью умножения на матрицу. Подобные преобразования координат часто используются в линейной алгебре. Для того, чтобы найти координаты точки в новой системе, они умножаются на матрицу, обратную так называемую матрице перехода справа, элементы которой представляют координаты ортов новой системы в старой. Координаты рассматриваются как строки матрицы, то есть координаты радиус-вектора. Однако такое преобразование не позволяет реализовать сдвиг. Для реализации любого аффинного преобразования вместо матицы 2x2 используется матрица 3x3, и вектор

имеет 3 координаты. Третья координата точки всегда 1, в матрице в последнем столбике во всех строках кроме последней 0, а в последней 1. За сдвиг теперь отвечает 1 и 2 элементы последней строчки матрицы. В GDI+ имеется класс *Matrix*, представляющий собой матрицу аффинных преобразований для 2x-мерного пространства. В нём реализованы методы для умножения точек, матриц, нахождения обратных матриц, которые используются в работе.

Чтобы значения матрицы перемещали, масштабировали, наклоняли, поворачивали изображение, чтобы это выглядело будто координаты точек умножаются на заданную матрицу, требуется найти значения координат точки в экранных координатах, при условии, что даны координаты в преобразованной системе координат. Следовательно, чтобы найти новые координаты нужно умножить старые координаты на матрицу аффинных преобразований справа. Однако, фильтр рассчитывает значения для каждой точки нового изображения, отображая их на старое, а не координаты в новом изображения для каждой точки старого. Поэтому нужно умножать на обратную матрицу.

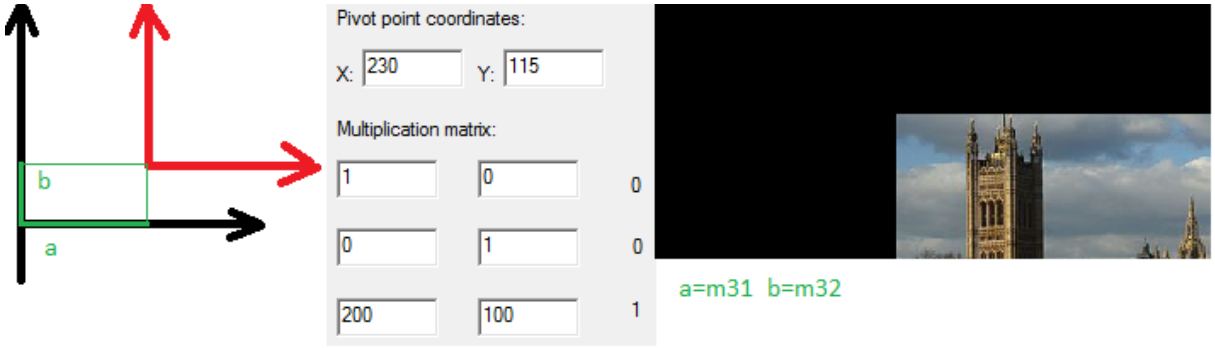
Чтобы реализовать возможность выбора опорной точки— точки на изображении, относительно которой производятся видимые преобразования, необходимо сместить изображение, чтобы эта точка оказалась в начале координат, выполнить основное преобразование и сместить изображение обратно. На уровне вычислений это выглядит как получение конечной матрицы путём умножения единичной матрицы на первую матрицу сдвига, затем на матрицу основного преобразования и на вторую матрицу сдвига. Инвертировав конечную матрицу и умножив на неё все точки конечного изображения, будут рассчитаны координаты пикселей на старом изображении, которые станут пикселями нового.

Рис15. Аффинные преобразования.



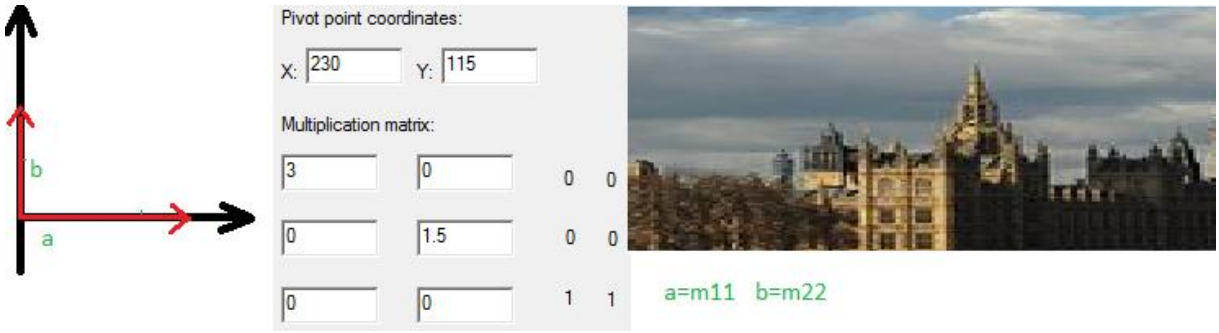
Для реализации смещения используются значения последней строки матрицы, которые отвечают за положение начала отсчёта новой системы координат. Значение в первом столбике представляет горизонтальное смещение, во втором—вертикальное.

Рис16. Смещение.



Для эффекта масштабирования нужно увеличить или уменьшить координаты ортов новой системы. То есть умножить их на коэффициент, пропорциональный старому значению. Следовательно, коэффициент масштабирования по горизонтали находится в первой строке, в первом столбике, а по вертикали во второй строке, втором столбике.

Рис17. Масштабирование.

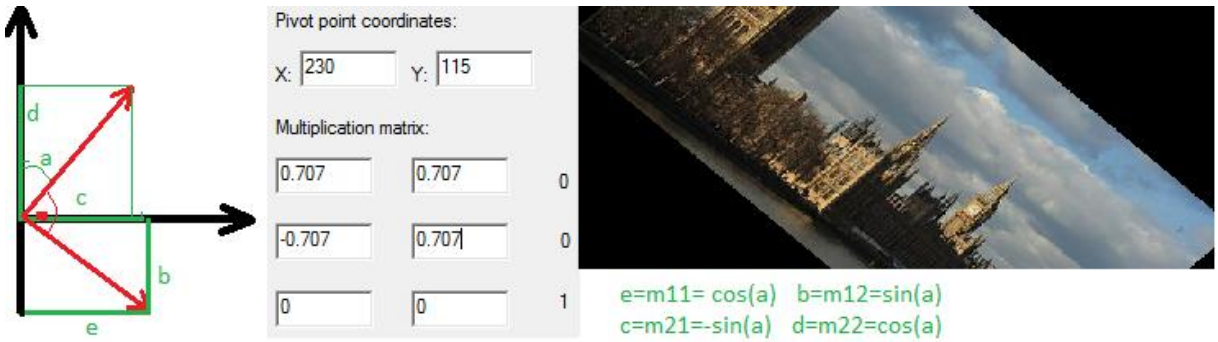


При наклоне по горизонтали точка смещается горизонтально на величину, пропорциональную её Y-координате, а при вертикальном наклоне, пропорциональную X, следовательно, значения углового коэффициента угла наклона по горизонтали и по вертикали находятся в первом столбике, второй строчке и первой строчке, втором столбике.

Рис18. Наклон.

При повороте новые координаты ортов зависят и от X и от Y. Элементами матрицы перехода являются проекции ортов на координатные оси, зависящие от угла поворотаа. Новые координаты орта X: (cos(a), sin(a)), для Y: (-sin(a), cos(a)).

Рис19. Поворот.



9.2 Изменение параметров цвета

Иногда приходится менять значения яркости, оттенка и насыщенности для всего изображения. Это можно сделать, преобразовав значение пикселя из цветовой модели RGB в цветовую модель HSV, произвести изменения и преобразовать обратно. Для задания изменённых значений, исходные значения суммируются со значениями, заданными пользователем.

Формулы преобразования RGB в HSV были представлены ранее. для обратного преобразования проводятся вычисления, представленные ниже вместе с результатом.

Рис20. Изменения оттенка, насыщенности, яркости путём преобразования в HSV и обратно.(25)

$$C = V \times S$$

$$X = C \times (1 - |(H / 60°) \bmod 2 - 1|)$$

$$m = V - C$$

$$(R,G,B) = (R'+m, G'+m, B'+m)$$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0° \le H < 60° \\ (X, C, 0) & , 60° \le H < 120° \\ (0, C, X) & , 120° \le H < 180° \\ (0, X, C) & , 180° \le H < 240° \\ (X, 0, C) & , 240° \le H < 300° \\ (C, 0, X) & , 300° \le H < 360° \end{cases}$$

.

10. Заключение

Целью данной работы было создание средства открытия, сохранения и обработки растровых изображений, используя только Windows API и стандартную библиотеку C, то есть разработка простого функционала графического редактора для Windows. Полученный прототип удовлетворяет поставленным целям. Была создана основа, позволяющая представлять открытые изображения в виде, удобном для обработки и анализа. На основе этого создано несколько простых инструментов рисования и 2 фильтра. Также создан функционал, позволяющий показывать статистическую информацию об изображении. На основе этого полученную программу уже можно назвать графическим редактором.

Полученный прототип можно расширить добавлением новых команд меню, новых инструментов, алгоритмы рисования можно усложнить, например, добавив возможность сглаживания, также можно написать новые фильтры. То есть полученный прототип можно использовать как основу для более совершенной версии графического редактора. Другим возможным применением полученного прототипа может быть тестирование новых алгоритмов по обработке битовых карт.

Хоть полученный прототип и удовлетворяет начальным целям, однако остались нереализованными некоторые важные свойства, характерные полноценному редактору. Например, реализация отмены/повтора действий, возможность выделения области, изменение размеров изображения, поддержка буфера обмена, настройка параметров кодировщиков при сохранении. Однако всё это можно реализовать, используя текущий прототип, что даёт простор для дальнейших усовершенствований.

Kokkuvõte

Sellise töö eesmärgiks oli avamise, salvestamise ja töötlemise vahendi loomine ainult Windows API ja C standardteeke kasutades, s.t lihtsa graafilise redaktori funktsionaalsuse arendus Windows jaoks. Loodud prototüüp vastab püstitatud eesmärkidele. On loodud vundament, mis aitab avada pildid vormis, mis on mugav töötlemise ja analüüsimise jaoks. Selle abiga oli loodud mitu lihtsat joonistamise tööriista ja 2 filtrit. Ka on loodud funktsionaalsus, mis võimaldab näidata statistilist infot pildist. Loodud programmi võibnimetada graafilist redaktoriks.

Loodud redaktori prototüüpi saab laiendada lisades uued menüükäsud, tööriistad, joonistamise algoritmid saab teha keerulisem ja anda silumise võimalust, ka saab kirjutada uued filtrid. Saanud prototüüp saab kasutada nagu platvorm graafilise redaktori uue versiooni jaoks. Teine võimalik kasutamise viis saab olla uute pikseli töötlemise algoritmide testimine.

Vaatamata sellele, et tehtud prototüüp vastab eesmärkidele, on jäänud realiseerimata mõned olulised omadused, mis on iseloomulikud täieliku graafilise editori jaoks. Näiteks undo/redo realiseerimine, piirkonna valimine, suuruse muutumine, lõikelaua toetus, kodeerimise seadistamine. Aga kõik seda saab realiseerida praeguse prototüüpi kasutades, mis annab võimalusi edasi arendamise jaoks.

Summary

The aim of this work was to create mean for raster images opening, saving and processing using only Windows API and C standard library, other words development of functionality of the simple graphics editor for Windows. Written prototype satisfies specified aims. Was created base, which presents image in form convenient for next manipulations: processing and analysis. Based on this was created some drawing tools and 2 filters. In addition to this was created functionality for displaying of statistic information about image. Therefore created prototype can be called graphics editor.

Prototype can be extended by new menu commands, new tools, drawing algorithms can be improved for example, by anti-aliasing, can be added new filters. Other words, current prototype can be used for new versions ofthe graphics editor. The other possibility of using is platform for new bitmap processing algorithms testing.

Despite of current prototype satisfies initial requirements, but some important features, which are also common for real graphics editors are not implemented. For example, undo/redo implementation, region selection, image dimensions change, clipboard support, encoder parameters tuning. However, all of this can be implemented using current prototype.

Список литературы

1. **Native (computing).** ***Wikipedia.*** **[ВИнтернете] http://en.wikipedia.org/wiki/Native_(computing).**
2. Application programming interface. ***Wikipedia.*** **[ВИнтернете] https://en.wikipedia.org/wiki/Application_programming_interface.**
3. Шаг 58 - Понятие дескриптора. ***ПЕРВЫЕ ШАГИ.ru.*** **[В Интернете] http://www.firststeps.ru/mfc/winapi/r.php?58.**
4. Graphics Device Interface. ***Wikipedia.*** **[ВИнтернете] https://en.wikipedia.org/wiki/Graphics_Device_Interface.**
5. Device Context. ***microsoft develper network.*** **[В Интернете] https://msdn.microsoft.com/en-us/library/windows/desktop/dd183553(v=vs.85).aspx.**
6. CLSID Key. ***microsoft developer network.*** **[В Интернете] https://msdn.microsoft.com/en-us/library/windows/desktop/ms691424(v=vs.85).aspx.**
7. Blitter. ***Wikipedia.*** **[ВИнтернете] https://en.wikipedia.org/wiki/Blitter.**
8. Affine transformation. ***Wikipedia.*** **[В Интернете] https://en.wikipedia.org/wiki/Affine_transformation.**
9. Virtual machine. ***Wikipedia.*** **[ВИнтернете] http://en.wikipedia.org/wiki/Virtual_machine.**
10. ***mingw-w64, GCC for Windows 64 & 32 bits.*** **[В Интернете] http://mingw-w64.org.**
11. ***Notepad++ Home.*** **[В Интернете] https://notepad-plus-plus.org/.**
12. ***ResEdit Home Page.*** **[ВИнтернете] http://www.resedit.net/.**
13. Rational Rose Family. ***IBM.*** **[ВИнтернете] http://www-03.ibm.com/software/products/en/ratirosefami.**
14. Windows API Index. ***microsoft developer network.*** **[В Интернете] https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx.**
15. Component Object Model. ***Wikipedia.*** **[ВИнтернете] http://en.wikipedia.org/wiki/Component_Object_Model.**
16. Hungarian notation. ***Wikipedia.*** **[ВИнтернете] https://en.wikipedia.org/wiki/Hungarian_notation.**
17. CamelCase. ***Wikipedia.*** **[ВИнтернете] https://en.wikipedia.org/wiki/CamelCase.**
18. Run-lenght encodin. ***Wikipedia.*** **[В Интернете] http://en.wikipedia.org/wiki/Run-length_encoding.**
19. Huffman coding. ***Wikipedia.*** **[ВИнтернете] http://en.wikipedia.org/wiki/Huffman_coding.**
20. DEFLATE. ***Wikipedia.*** **[В Интернете] http://en.wikipedia.org/wiki/DEFLATE.**
21. JPEG. ***Wikipedia.*** **[В Интернете] http://en.wikipedia.org/wiki/JPEG.**
22. RGB to CMYK color conversion. ***RapidTables.*** **[В Интернете] http://www.rapidtables.com/convert/color/rgb-to-cmyk.htm.**
23. HSL and HSV. ***Wikipedia.*** **[ВИнтернете] http://en.wikipedia.org/wiki/HSL_and_HSV.**
24. Bresenham's line algorithm. ***Wikipedia.*** **[В Интернете] https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm.**
25. HSV to RGB color conversion. ***RapidTables.*** **[В Интернете] http://www.rapidtables.com/convert/color/hsv-to-rgb.htm.**