МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по компьютерной графике

Учебно-методическое пособие

Часть 1. Графика в Windows

2018 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по компьютерной графике. Учебно-методическое пособие. Часть 1. Графика в Windows. Озерск: ОТИ НИЯУ МИФИ, 2018. — 31 с.

В пособии подробно излагается, как выполнять практические работы по дисциплине «Инженерная и компьютерная графика». Работы первой части изучения дисциплины включают в себя методы и алгоритмы рисования в Windows.

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Общие цели занятий

В ходе практических работ предлагается изучить основы программирования графических примитивов в среде Windows. В этой части работ рассматриваются следующие темы:

1) растровые картинки;
2) метафайлы;
3) шрифты;
4) рисование мышью;
5) графические примитивы;
6) растровые алгоритмы.

Основные сведения по графическим примитивам и графическим функциям Windows приведены в учебно-методическом пособии автора «Компьютерная графика. 2006», а растровые алгоритмы — в пособии «Машинная графика. 2006». Наиболее актуальные версии этих документов доступны в Интернет по адресу http://revol.ponocom.ru.

На выполнение каждой работы предположительно отводится 2 академических часа, однако некоторые, наиболее сложные работы могут потребовать большего времени.

Поэтому, в зависимости от количества учебных часов, выделенных на проведение практических работ, сложности работ и навыков обучающегося, преподаватель может выбирать индивидуальные траектории работ.

Для защиты знаний и навыков, полученных в ходе выполнения практических работ студент должен иметь тетрадь (12-18 листов). Для каждой выполненной работы в тетрадь записывается отчет.

Отчет по работе начинается с заголовка:

1. Фамилия Имя Отчество
2. Группа
3. Дата начала выполнения работы
4. Код работы
5. Название работы
6. Цели работы
7. Задачи работы

При необходимости при выполнении работы в отчет записываются контрольные значения. Во время защиты тетрадь используется для вопросов преподавателя и ответов студента.

Работа КГ-101. Растровые картинки

Цели:
- изучение структур пиксельного набора.

Задачи:
- формирование структур пиксельного набора;
- чтение графического файла типа bmp;
- отображение пиксельного набора в окне;
- обработка события `WM_PAINT`.

1.1. Пиксельный набор

Пиксельным набором здесь называется пиксельное графическое изображение (*bitmap*), хранимое в памяти. Пиксельные наборы используются для работы с растровыми графическими изображениями. Фактически это массив цветов пикселей прямоугольной области размером `BMPX`×`BMPY`, полагая, что `BMPX` — ширина области набора, `BMPY` — высота. Далее вместо слов «пиксельный набор» будем пользоваться словом «картинка».

Различают DIB и DDB картинки.

DIB означает *Device-Independent Bitmap*, набор пикселей, независимый от устройства. В этом случае подразумевается, что цвета, заданные в пиксельном наборе, не зависят от устройства, а при выводе на устройство заданным в наборе цветам сопоставляются наиболее подходящие.

DDB означает *Device-Dependent Bitmap*, набор пикселей, зависимый от устройства. В этом случае подразумевается, что цвета картинки в точности совпадают с цветами устройства.

Например, графический файл формата bmp описывает *dib*-картинку, в то время как вывод ее осуществляется на конкретное устройство, для чего картинка должна быть преобразована в *ddb*-картинку.

Для описания *dib*-картинки используются информационные структуры, без которых невозможно отображение картинки на устройстве. Например, графический файл формата bmp описывает эти структуры в явном виде. Целью данной работы как раз является изучение этих структур, чтение их из файла, чтение из файла пиксельного набора, и отображение.

Важными характеристиками пиксельного набора являются:
- количество плоскостей;
- ширина `BMPX` и высота `BMPY` картинки;
- размер пиксельного набора.
- количество бит на пиксель;

Количество плоскостей в графических файлах типа bmp не используется и всегда равно единице.

Размеры картинки требуются для ее вывода на устройство.

Размер пиксельного набора нужен для того, чтобы знать, сколько байт занимает собственно набор при чтении графического файла.

Количество бит на пиксель (*глубина цвета*) является одной из самых важных характеристик, от этой характеристики зависит не только качество картинки, но и структура графического файла типа bmp.

Возможные значения глубины цвета равны 1, 4, 8, 24.

Значение 1 означает черно-белую (монохромную) картинку, палитра содержит 2 цвета, при этом для задания цвета используется один бит.

Значение 4 означает, что используется палитра из 16 цветов, а цвет пикселя в наборе задается половиной байта.

Значение 8 означает, что используется палитра из 256 цветов, а цвет задается в наборе одним байтом.

Значение 24 означает, что палитра не используется, а цвет задается в наборе непосредственно.

Файл типа bmp, описывающий картинку с глубиной цвета, не равной 24, имеет в своем составе *палитру цветов*, а пиксельный набор содержит *индексы* цветов палитры, а не сами цвета. Файл, описывающий картинку с глубиной цвета 24, *не содержит палитру*, а пиксельный набор состоит из структур, описывающих цвет одного пикселя в виде составляющих RGB.

Вам надлежит выучить наизусть цвета 16-цветной палитры.

1.2. Структура графического файла

Здесь рассматривается структура графического файла типа bmp.

Файл типа bmp состоит из 4-х разделов:

- заголовок файла в виде структуры BITMAPFILEHEADER;
- информация о наборе в виде структуры BITMAPINFOHEADER;
- палитра цветов (если есть);
- массив цветов пикселей (*индексы* в палитре или сами *цвета*).

Размер первой структуры равен 14 байт. Первые два байта имеют значение, отображающееся в буквы "BM", означающие BITMAP. Структура описана в файле wingdi.h и имеет следующий вид:

```
typedef struct tagBITMAPFILEHEADER {
    UINT  bfType;         // признак BM
    DWORD bfSize;         // размер файла
    UINT  bfReserved1;
    UINT  bfReserved2;
    DWORD bfOffBits;      // смещение к массиву пиксельного набора
} BITMAPFILEHEADER;
```

Буквы "BM" задаются полем bfType. Поле bfSize задает размер файла в байтах. Поле bfOffBits задает смещение к массиву пиксельного набора.

С помощью этой структуры определяется, что файл является действительным файлом типа bmp. Для этого сравниваются первые два байта файла, а размер файла сопоставляется с действительным.

Размер второй структуры равен 40 байт.

Структура имеет вид:

```
typedef struct tagBITMAPINFOHEADER {
    DWORD  biSize;
    LONG   biWidth;          // ширина картинки
    LONG   biHeight;         // высота картинки
    WORD   biPlanes;         // для BMP всегда 1
    WORD   biBitCount;       // глубина цвета, 1, 4, 8, 24
    DWORD  biCompression;    // сжатие
    DWORD  biSizeImage;      // размер набора пикселей
    LONG   biXPelsPerMeter;
    LONG   biYPelsPerMeter;
    DWORD  biClrUsed;
    DWORD  biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

В этой структуре описываются характеристики пиксельного набора.

Эта структура входит в состав структуры для считывания из файла не только информации о картинке, но и палитры целиком:

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD          bmiColors[1];
} BITMAPINFO, *PBITMAPINFO;
```

Здесь структура RGBQUAD описывает один элемент палитры:

```
typedef struct tagRGBQUAD {
   BYTE rgbBlue;
   BYTE rgbGreen;
   BYTE rgbRed;
   BYTE rgbReserved;
} RGBQUAD;
```

Если картинка имеет глубину цвета 1, 4 или 8, то с помощью структуры BITMAPINFO из файла считывается структура BITMAPINFOHEADER и палитра целиком. Поскольку в структуре определен всего один элемент RGBQUAD, то при считывании задается размер, соответствующий размеру структуры BITMAPINFO и всей палитры.

Например, если картинка имеет глубину цвета 4 (палитра 16 цветов), считываемый размер определяется следующим образом:

```
int cbmi = sizeof(BITMAPINFO) + sizeof(RGBQUAD) * 15;
```

Считывание производится следующим образом:

```
BITMAPINFO * pbmi = (BITMAPINFO*)LocalAlloc(LPTR, cbmi);
if (!fread((void*)pbmi, cbmi, 1, f)) {
    return 0;
}
```

Здесь подразумевается, что файл открыт с помощью структуры FILE, а переменная f — указатель на файловую структуру.

Если глубина цвета 24, то считывается структура BITMAPINFO без второго поля, считываемый размер при этом можно вычислить так:

```
int cbmi = sizeof(BITMAPINFO) - sizeof(RGBQUAD);
```

1.3. Рабочее пространство

Скачайте с сайта преподавателя архив задания КГ-101. Извлеките их архива каталог bmap в корневой каталог диска C:.

При этом в каталоге C:\bmap\Debug должны находиться четыре файла: ch01.bmp, ch04.bmp, ch08.bmp, ch24.bmp. Название файла указывает на глубину цвета картинки.

Откройте проект и изучите его устройство. При защите работы будет контролироваться знание назначения всех функций и переменных. При необходимости используйте источник [2].

Основные действия производятся в функции bitmap, которая расположена в начале модуля bmap.cpp:

```
#define PATH "ch01.bmp"
// чтение картинки
void bitmap() {
    return 1;
}
```

Константа `PATH` задает путь к файлу. В модуле bmap.h располагаются глобальные константы и переменные.

Кроме того, действия будут также происходить в оконной процедуре в обработке события `WM_PAINT`.

Функция bitmap вызывается в главной функции.

1.4. Чтение файла

Функция bitmap.

Сначала нужно открыть файл для чтения в бинарном режиме.

Будем использовать для операций с файлом структуру FILE, а для открытия файла, путь к которому задается переменной file_name, будем использовать функцию fopen. Пример, показывающий, как открыть файл:

```
FILE * f = 0;
f = fopen(PATH, "rb");
```

После открытия файла проверяем значение переменной f, и если оно равно нулю, завершаем функцию, возвращаем 0.

1.4.1. Чтение заголовка файла

Если файл открыт, считываем первую структуру, из которой состоит файл bmp, а именно структуру BITMAPFILEHEADER.

Для этого ее нужно объявить, переменная bmfh.

Затем с помощью функции fread() считываем заголовок файла:

```
int res = fread((void *) & bmfh, sizeof(bmfh), 1, f);
```

Второй параметр — размер считываемой информации, третий — количество считываемых единиц. После считывания проверяем значение переменной res, и если оно равно нулю, завершаем функцию, возвращаем 0.

В этом месте нужно остановить выполнение программы и изучить считанную информацию. Эту информацию нужно сравнить с дампом исследуемого файла. Чтобы открыть дамп в файловом менеджере FAR, установите указатель на файл и нажмите F3, а затем, если необходимо, F4.

Исследования следует выполнить для всех графических файлов.

При защите этой части будет контролироваться знание расположения элементов структуры в дампе.

В завершении этой части нужно произвести проверку того, что считываемый файл является действительным файлом bmp. Для этого нужно убедиться, что первые два байта файла составляют буквы "BM", и сравнить действительную длину файла со значением поля bfSize. Для простоты можно проверить только значение поля bfType.

1.4.2. Чтение информации о картинке

В этой части сначала нужно считать из файла в динамическую структуру BITMAPINFOHEADER информационную часть графического файла, проанализировать ее и сопоставить с дампом. Сделать это нужно для каждого имеющегося файла.

Поскольку перед чтением информационной части неизвестно, какая глубина цвета используется в картинке, нужен какой-то прием, обходящий это недоразумение.

Например, можно сначала прочитать структуру BITMAPINFOHEADER, выяснить глубину цвета, рассчитать размер структуры BITMAPINFO и палитры вместе, и прочитать их заново, переустановив указатель позиции файла.

Другой вариант — вычислить необходимый размер по имеющимся после чтения заголовочной структуры файла данным, и сразу прочитать информационную часть вместе с палитрой.

BITMAPFILEHEADER
BITMAPINFOHEADER
Палитра
Массив пиксельного набора

В структуре BITMAPFILEHEADER поле bfOffBits — это смещение к началу пиксельного набора, или, иначе, размер первых трех структур файла. Если из этого значения вычесть размер структуры BITMAPFILEHEADER, то получим размер структуры BITMAPINFOHEADER вместе с палитрой.

Пусть переменная cbmi — это требуемый размер:

```
// размер BITMAPINFO
int cbmi = bmfh.bfOffBits - sizeof(BITMAPFILEHEADER);
```

Объявить переменную для считывания BITMAPINFO как статическую нельзя, — размер этой структуры учитывает только один элемент палитры.

Поэтому нужно выделить память динамически:

```
// структура
BITMAPINFO * pbmi = (BITMAPINFO *)LocalAlloc(LPTR, cbmi);
```

После этого можно считывать структуру:

```
res = fread((void*)pbmi, cbmi, 1, f);
```

Значение переменной res опять нужно проверять.

Здесь также нужно остановить выполнение программы для того, чтобы исследовать считанные структуры, сравнить их с дампом файла.

Для анализа элементов палитры можно объявить указатель

```
RGBQUAD * ppal = &pbmi->bmiColors[0];
```

Если установить этот указатель на второй элемент BITMAPINFO, то в окне просмотра переменных Watch можно ввести, например, ppal[1], и проанализировать второй элемент палитры.

При защите этой части будет контролироваться значение всех элементов структуры BITMAPINFOHEADER, в том числе значения цветов палитры.

1.4.3. Чтение набора пикселей

После того, как информационная часть, вместе с палитрой, прочитаны из графического файла, можно считывать пиксельный массив:

```
// массив пиксельного набора
BYTE * dibs = (BYTE*)LocalAlloc(LPTR, pbmi->bmiHeader.biSizeImage);

// читаем пиксельный набор
res = fread((void*)dibs, pbmi->bmiHeader.biSizeImage, 1, f);
```

После этого все необходимые элементы для создания картинки в памяти есть, и картинку можно создать:

```
// картинка
HBMP = CreateDIBitmap(hDCBMP, &pbmi->bmiHeader, CBM_INIT,
    dibs, pbmi, DIB_RGB_COLORS);
```

1.5. Вывод картинки

Теперь мы готовы к тому, чтобы вывести картинку.

Как известно, рисовать можно только в контексте устройства.

На самом деле рисовать картинку будем в совместимом контексте.

Совместимый контекст, — это контекст, совместимый с устройством. Он создается функцией CreateCompatibleDC. Переменная для совместимого контекста CDC объявлена как глобальная в модуле bmap.h.

Переходим в модуль bmap.h, функция InitInstance.

Создаем совместимый контекст. Для создания контекста требуется дескриптор окна hWndBMP, который в этой функции как раз создается.

Возвращаемся в функцию bitmap и выбираем картинку в совместимый контекст с помощью функции SelectObject.

Картинка выводится функций BitBlt. Для вывода нужно иметь:

- контекст окна, в которое выводится картинка hWndBMP;
- размеры картинки `BMPX` и `BMPY`;
- совместимый контекст CDC.

Пример вывода есть в [1].

Масштабировать картинку при выводе можно, если вместо функции BitBlt использовать функцию StretchBlt. Используйте для этой цели коэффициент масштабирования ScaleBlt, определенный в модуле bmap.h.

1.6. Событие WM_PAINT

После того, как картинка выводится в окно, следует убедиться в том, что она исчезает, как только она будет чем-то закрыта. Например, можно просто переместить окно так, чтобы картинка попала за пределы экрана и затем переместить окно обратно.

Нужно понимать, что тот факт, что мы нарисовали что-то в окне, вовсе не означает, что это теперь там находится. Графический вывод тем и отличается, что нарисованное изображение нужно выводить в окно каждый раз, когда часть окна закрывается и открывается вновь.

Для этой цели существует событие `WM_PAINT`.

Оно сообщает окну (*оконной процедуре*), что часть окна или окно целиком требует перерисовки (если что-то было нарисовано).

Перерисовать можно все окно целиком, если алгоритм несложный.

Если алгоритм рисования сложный, можно перерисовывать только ту часть окна, которая очистилась. Эту часть окна всегда можно узнать. На самом деле часто бывает проще перерисовать все окно, чем вычислять его часть. Мы так и поступим, хотя именно в нашем случае часть картинки, которую нужно обновить, вычислить несложно.

На самом деле нужно всего лишь заново нарисовать картинку при помощи функции BitBlt в обработке события `WM_PAINT`.

Чтобы это сработало, все переменные, используемые функцией BitBlt, должны быть глобальными.

1.7. Контрольные вопросы и упражнения

1. Что означает глубина цвета?
2. Какими структурами описывается файл типа .bmp?
3. Какие функции используются для вывода пиксельного набора?
4. Измените один из цветов палитры перед ее выводом.

Работа КГ-102. Метафайлы

Цели:
- изучение структуры метафайла Win32.

Задачи:
- формирование метафайла;
- вывод метафайла;
- запись метафайла в файл.

2.1. Структура метафайла

Метафайл — это запись команд графического ядра Windows, которые проигрываются с помощью функции PlayEnhMetaFile.

Различают метафайлы для платформы Win16 (файлы типа wmf), и улучшенные метафайлы для платформы Win32 (файлы типа emf).

Улучшенный метафайл (далее просто метафайл) состоит из массива записей, образующих следующие разделы:
- заголовок;
- необязательная таблица дескрипторов графических объектов;
- необязательная палитра;
- список команд для воспроизведения — мета-записи.

Структуры для метафайлов определены в файле wingdi.h.

Заголовок метафайла описывается структурой ENHMETAHEADER:

```
typedef struct tagENHMETAHEADER {
    DWORD   iType;          // тип EMR_HEADER
    DWORD   nSize;          // размер структуры
    RECTL   rclBounds;      // границы
    RECTL   rclFrame;       // рамка
    DWORD   dSignature;     // подпись ENHMETA_SIGNATURE
    DWORD   nVersion;       // версия
    DWORD   nBytes;         // размер метафайла в байтах
    DWORD   nRecords;       // число записей
    WORD    nHandles;       // число дескрипторов
    WORD    sReserved;      // 0
    DWORD   nDescription;   // символов Unicode в описании
    DWORD   offDescription; // смещение к описанию или 0
    DWORD   nPalEntries;    // число элементов палитры
    SIZEL   szlDevice;      // разрешение устройства в pels
    SIZEL   szlMillimeters; // разрешение устройства в мм
    DWORD   cbPixelFormat;  // размер PIXELFORMATDESCRIPTOR или 0
    DWORD   offPixelFormat; // смещение к PIXELFORMATDESCRIPTOR или 0
    DWORD   bOpenGL;        // есть команды OpenGL
} ENHMETAHEADER, *PENHMETAHEADER, *LPENHMETAHEADER;
```

Размер структуры в разных версиях операционной системы может быть различным (за счет дополнительных полей). Аналогичная структура для метафайла Win16 раза в два меньше по размеру.

Метафайл может иметь описание, которое в этом случае располагается после заголовка.

Одиночный нулевой символ в описании завершает одну строку описания, два нулевых символа завершают описание в целом.

Мета-записи идентифицируют функции, используемые для рисования изображения, а также параметры этих функций, если они есть. Для этих целей определена структура METARECORD:

```
typedef struct tagMETARECORD {
    DWORD   rdSize;
    WORD    rdFunction;
    WORD    rdParm[1];
  } METARECORD;
```

Каждой функции сопоставлен номер в виде константы вида EMR_XXX, XXX — название функции. Всего определено 120 функций и констант.

Так как разные функции GDI имеют разное количество параметров, размеры мета-записей различны.

2.2. Создание метафайла

Создать метафайла не сложно, и использовать описанные структуры при этом нет необходимости. Сначала создается контекст метафайла с помощью функции CreateEnhMetaFile. Затем с помощью функций GDI рисуется изображение или задаются режимы отображения. В завершение контекст метафайла закрывается функцией CloseEnhMetaFile.

При создании контекста метафайла указывается путь к файлу, в который будет записан метафайл. Если путь к файлу не задан, метафайл создается только в памяти.

Функция CloseEnhMetaFile, закрывающая контекст метафайла, возвращает графический объект типа HENHMETAFILE, который можно использовать в функции PlayEnhMetaFile для воспроизведения. Для воспроизведения нужно задать также прямоугольную область в виде структуры RECT.

2.3. Рабочее пространство

Работа выполняется в рамках предыдущего проекта bmap.

Рабочим модулем является bmap.cpp. Основные действия выполняются в функции metafile, расположенной за функцией bitmap. Эти две функции вызываются друг за другом, но вывод осуществляется в разные окна.

Для получения заданного изображения используются графические функции, такие как MoveToEx, LineTo, Polygon, Rectangle, Ellipse и другие. Описание функций есть, например, в [1].

2.4. Метафайл с автоматическим определением области вывода

Первый метафайл, который нужно создать, не задает область вывода. Нарисуйте графическое изображение государственного флага РФ. Файл метафайла назовите flag.emf.

Изображение должно выводиться в контекст hDCEMF и наблюдаться в правом окне приложения. Созданный в каталоге C:\bmap\Debug метафайл можно просмотреть стандартными средствами Windows.

2.5. Метафайл с заданной областью вывода

Второй метафайл должен задавать область вывода в HIMETRIC.

Код, формирующий первый метафайл, следует закомментировать.

С помощью линий толщиной в 3 мм нарисуйте в этом метафайле рамку, расчерченную на квадраты линиями толщиной 1 мм.

Размер рамки 10×10 см, размер квадратов 5×5 см.

Выведите этот метафайл в файл quad.emf и в окно приложения.

2.6. Проигрывание файла метафайла

В заключение выведите в окно приложения любой из полученных в предыдущих частях работы файлов.

Получить графический объект типа HENHMETAFILE можно с помощью функции GetEnhMetafile. Дополнительно см. [1].

2.7. Контрольные вопросы и упражнения

1. Найдите файл типа wmf и выведите его в окно приложения.
2. Сравните пиксельную графику и метафайл. Укажите достоинства и недостатки растровых картинок и метафайлов.

Работа КГ-103. Работа со шрифтами

Цели:
- изучение функций GDI для работы со шрифтами.

Задачи:
- формирование шрифтов;
- выбор шрифтов;
- вывод надписей.

3.1. Шрифты

Шрифты — важная часть графической подсистемы. С помощью шрифтов выполняются все надписи, видимые на экране компьютера.

Шрифт — это коллекция символов, имеющих одинаковое начертание. Три важнейших элемента шрифта — это гарнитура (*typeface*), начертание (*style*) и размер (*size*).

Гарнитура определяет внешний вид символов разной шириной толстых и тонких основных штрихов, а также наличием или отсутствием засечек. Засечка (*serif*) — небольшой поперечный штрих на конце основного штриха. Шрифт без засечек принято называть *sans-serif*.

Начертание определяет жирность штрихов и наклон символов.

Наклон определяется терминами *roman* (прямые символы), *oblique* (наклонные символы) и *italic* (истинно наклонные символы).

Жирность (*weight*, вес) определяется числовыми значениями от 100 до 900, но в обычных приложениях используется только два типа жирности — нормальная и полужирная, со значениями 400 и 700. Другие значения используются в полиграфии.

Размер шрифта — это высота от верхней части буквы типа À (буквы с надстрочным знаком) до нижней части буквы типа g. Для измерения высоты часто используются *пункты*, принимаемые равными 1/72 дюйма.

Шрифт создается функциями CreateFont или CreateFontIndirect, которые возвращают дескриптор шрифта типа HFONT. Во втором случае используется структура LOGFONT.

Перед использованием шрифта его нужно выбрать в контекст.

Характеристики шрифта описываются структурой TEXTMETRIC.

3.2. Надписи

Для вывода надписей используются функции TextOut и DrawText. Различия между ними заключаются в возможностях форматирования текста при выводе, которых у второй функции больше.

Надписи выводятся выбранным в контекст шрифтом, при этом используется текущие цвета фона и плана.

Цвет надписи задается с помощью функции SetTextColor, а цвет фона — функциями SetBkColor и SetBkMode.

3.3. Рабочее пространство

Скачайте с сайта преподавателя архив работы КГ-102, извлеките из архива каталог fonts в корневой каталог диска C:.

Рабочим модулем является fonts.cpp, программирование выполняется в функции fonts.

Надписи выводятся в окно вывода, но графический вывод следует направлять в совместимый контекст CDC. Для принудительного обновления окна вывода в конце функции fonts должна быть инструкция:

```
// принудительный вывод в окно вывода
InvalidateRect(hWndOut, 0, 1);
```

3.4. Использование стандартных шрифтов

В этой части работы используем стандартные шрифты системы.

Для вывода текста используем функцию TextOut.

Общий порядок вывода текста:

- выбрать шрифт функцией GetStockObject в переменную hf;
- выбрать шрифт hf в контекст CDC функцией SelectObject, результат функции запомнить в переменной exhf;
- установить цвет, режимы вывода;
- вывести текст;
- выбрать шрифт exhf в контекст CDC функцией SelectObject;
- удалить шрифт hf функцией DeleteObject.

Пример см. [1, листинг 16].

Константы системных шрифтов см. [1, приложение «Стандартные объекты Windows»].

Следует вывести за один раз все имеющиеся системные шрифты. Для этого нужно каждый раз выбирать шрифт заново. В отличие от первого выбора, последующие выборы шрифтов не должны изменять значение переменной exhf, которая хранит шрифт по умолчанию.

Выводим надписи вида «Шрифт ANSI_VAR_FONT». Не забываем уточнять длину надписи и корректировать ее при вызове функции TextOut.

В этой части следует использовать разные цвета для разных надписей, для чего перед выводом надписи задайте цвет функцией SetTextColor.

Кроме того, следует использовать функции SetBkColor и SetBkMode. Первая функция задает цвет фона, вторая устанавливает прозрачность.

Еще одна функция, которую нужно применить — SetTextAlign.

С ее помощью задается выравнивание текста относительно указанной точки вывода. Константы, задающие выравнивание, приведены в файле wingdi.h, в разделе /* Text Alignment Options */.

При защите этой части способы выравнивания, указанные в wingdi.h, будут контролироваться.

3.5. Выбор шрифта при помощи стандартного диалога

В этой части работы нужно показать стандартный диалог для выбора шрифта, выбрать в нем шрифт и вывести этим шрифтом надпись.

Для выбора стандартного диалога используем функцию ChooseFont.

Пример см. [1, листинг 10].

Для использования функции ChooseFont сначала нужно сформировать структуру CHOOSEFONT в переменной cf. После выбора шрифта характеристики шрифта формируются в виде структуры LOGFONT. Для разных выбранных шрифтов эти характеристики должны быть изучены.

Должны быть выбраны минимум следующие шрифты: Arial, System, Fixedsys, Courier New, Tahoma, Times New Roman, MS Sans Serif.

Шрифт в этом случае создается с помощью функции CreateFont-Indirect.

Заметим, что функция стандартного диалога ChooseFont возвращает *ложь* в случае, если пользователь отказывается от диалога.

3.6. Создание шрифта

В этой части работы изучаем функцию CreateFont.

Параметры этой функции схожи с полями структуры LOGFONT.

При выполнении этой части работы создаем несколько различных шрифтов. Различия должны быть принципиальными и затрагивать все характеристики шрифтов.

В частности, требуется вывести надписи:

- горизонтально,
- вертикально,
- с поворотом на угол 45°,
- с повернутыми буквами горизонтально и вертикально,
- с широкими и узкими символами.

Для выполнения некоторых частных случаев понадобится установить графический режим GM_ADVANCED при помощи функции SetGraphicsMode.

В качестве гарнитуры используем один шрифт с засечками и один шрифт без засечек.

Во время защиты этой части работы будет контролироваться знание всех характеристик шрифтов.

3.7. Контрольные вопросы и упражнения

1. Опишите характеристики шрифта.
2. Какими функциями задается цвет и фон надписи?
1. В оставшееся время изучите структуру TEXTMETRIC.
2. Получите характеристики шрифта Tahoma, изучите их.

Работа КГ-104. Рисование мышью

Цели:
- изучение сообщений о действиях с мышью;
- изучение основных графических примитивов;
- изучение графических режимов;

Задачи:
- обработка событий мыши;
- рисование следа мыши;
- рисование графических примитивов;
- управление режимом фона.

4.1. Рабочее пространство

Для выполнения работы используется проект Paint, подготовленный преподавателем. Проект следует установить на диск C:.

В проекте три основных модуля.

Модуль Paint.cpp содержит основной код приложения. Никаких изменений в этот модуль вносить не требуется.

Модуль Paint.h содержит константы, переменные, описания функций. Никаких изменений в этот модуль вносить не требуется.

Модуль drawing.h является рабочим модулем, в котором выполняется программирование.

Для рисования используется совместимый контекст CDC. В проекте создается окно для рисования, совместимый контекст, кисть и перья по умолчанию. Кроме того, проект реализует выбор инструментов, режимов и цветов, и устанавливает следующие переменные (почти все типа int):

background_mode — фон: 1 — прозрачный, 2 — не прозрачный;
painting_tool — текущий инструмент, тип TOOLS;
fore_color — текущий цвет плана;
back_color — текущий цвет фона;
pen_type — тип пера, 1 — косметическое, 2 — геометрическое;
pen_size — ширина пера, 1 — 1 px, 2 — 11 px, 3 — 25 px;
line_style — стиль линии, сплошная, пунктирная и т.п.;
pen_endcap — тип концевой точки линии;
pen_jointy — вид точки соединения линий;
g_style — тип кисти геометрического пера;
g_hatch — тип штриховки кисти геометрического пера.

Кроме того, здесь объявлены:

PenFore — перо плана;
PenBack — перо фона;
PenBrush — кисть геометрического пера;
poly_left — признак рисования полилинии пером плана;
poly_right — признак рисования полилинии пером фона.

Предполагается, что рисовать можно как левой, так и правой кнопкой мыши. В первом случае для рисования используется перо плана PenFore, во втором — перо фона PenBack. Если перо геометрическое, тогда оно использует кисть пера PenBrush. Описанные выше переменные нужны для создания этих перьев и этой кисти.

В проекте используется палитра из 16-ти цветов, пронумерованных от нуля до 15. Сами цвета заданы массивом C16 в модуле Paint.h. Заметим, что переменные fore_color и back_color имеют тип COLORREF, то есть они содержат цвет, а не индекс цвета. Для справки, индекс цвета содержат переменные fore_color_index и back_color_index.

В модуле drawing.h есть также несколько функций, которые должны быть описаны в ходе выполнения работы:

ShiftMouse — функция, которая выравнивает координаты мыши для рисования прямых линий в режиме рисования следа мыши, для рисования квадратов в режиме рисования прямоугольника, для рисования кругов в режиме рисования эллипса (овала); эта функция используется, когда пользователь удерживает клавишу Shift во время рисования;

SetBackStyle — функция, устанавливающая режим фона и кисть фона для текущего режима фона, прозрачного или непрозрачного; при рисовании пером плана ей передается текущий цвет фона, и наоборот, при рисовании пером фона ей передается текущий цвет плана;

LeftButtonUp — функция, которая вызывается, если была отпущена левая кнопка мыши;

RightButtonUp — функция, которая вызывается, если была отпущена правая кнопка мыши;

LeftButtonDown — функция, которая вызывается, если была нажата левая кнопка мыши;

RightButtonDown — функция, которая вызывается, если была нажата левая кнопка мыши;

CreateNewPen — функция, которая создает перо. Перо ей передается как параметр, кроме того, функции передается также цвет fore_color или back_color, соответствующий перу.

DrawMouse — функция инструмента «след мыши»; эта функция вызывается, когда выбран инструмент «след мыши», нажата и держится какая-то кнопка мыши, и произошло событие движения мыши;

DrawLine — функция инструмента «линия»; эта функция вызывается, когда выбран инструмент «линия», нажата и держится какая-то кнопка мыши, и произошло событие движения мыши;

DrawRect — функция инструмента «прямоугольник»;

DrawOval — функция инструмента «эллипс»;

DrawBrez — функция инструмента «линия Брезенхейма»;

Задача работы заключается в реализации перечисленных функций.

4.2. Сообщения о действиях с мышью

Операционная система посылает окну графического вывода следующие сообщения, когда пользователь выполняет действие с мышью:

WM_LBUTTONDOWN — левая кнопка мыши нажата (опущена);

WM_LBUTTONUP — левая кнопка мыши отпущена (поднята);

WM_MOUSEMOVE — мышь изменила положение.

Рисование мышью производится в событии WM_MOUSEMOVE. Это событие возникает при любом перемещении мыши, и нужен признак, указывающий на то, что пользователь удерживает левую кнопку мыши нажатой.

В проекте есть переменная mouse_down. Эту переменную нужно устанавливать в функциях вида *XXX*ButtonDown. Когда приходит сообщение WM_MOUSEMOVE, в модуле Paint.cpp вычисляются координаты мыши:

```
case WM_MOUSEMOVE:
    if (!mouse_down) return 0;
    GetMouse(MX2, MY2, wParam, lParam);
    ShiftMouse(MX1, MY1, MX2, MY2);
    switch (painting_tool) {
    case TOOL_MOUSE:
        DrawMouse(MX1, MY1, MX2, MY2);
        break;
    case TOOL_LINE:
        DrawLine(MX1, MY1, MX2, MY2);
        break;
    case TOOL_RECT:
        DrawRect(MX1, MY1, MX2, MY2);
        break;
    case TOOL_OVAL:
        DrawOval(MX1, MY1, MX2, MY2);
        break;
    case TOOL_BREZ:
        DrawBrez(MX1, MY1, MX2, MY2);
        break;
    case TOOL_BWOO:
        DrawBWoo(MX1, MY1, MX2, MY2);
        break;
    case TOOL_POLY:
        if (poly_left == 0 && poly_right == 0) break;
        DrawLine(MX1, MY1, MX2, MY2);
        break;
    }
}
```

Функция GetMouse модуля Paint.cpp извлекает координаты мыши из прикрепленного к сообщению параметра lParam, и возвращает их через первые два аргумента функции.

В программе есть глобальные переменные MX1, MY1, MX2, MY2. MX1 и MY1 соответствуют моменту нажатия кнопки мыши, MX2, MY2 соответствуют моменту движения. Эти переменные поступают в функции, рисующие тот или иной графический примитив.

Есть еще одна важная вещь, касающаяся событий мыши. После того, как кнопка мыши была опущена, нам необходимо, чтобы все дальнейшие события мыши поступали к нам, а не в другие окна, поскольку эти мыши «бегают» по всему экрану. Поэтому мы должны выполнить захват событий мыши при помощи функции SetCapture. Когда кнопка мыши будет отпущена, нам нужно освободить захват при помощи функции ReleaseCapture.

4.3. Рисование следа мыши

Сейчас мы реализуем первый инструмент, назовем его «след мыши». Рисовать будем отрезками прямых при помощи функции LineTo. Последовательность событий такая:

- нажата левая кнопка мыши;
- устанавливаем признак mouse_down;
- захватываем события мыши;
- вычисляем и запоминаем координаты мыши как MX1 и MY1;
- устанавливаем текущую точку рисования при помощи MoveToEx;
- мышь передвинулась;
- координаты мыши передаются в функцию DrawMouse;
- рисуем отрезок прямой при помощи LineTo;
- вызываем событие WM_PAINT при помощи InvalidateRect;
- кнопка мыши отпущена;
- снимаем признак mouse_down;
- освобождаем захват.

Программируем действия.

Функция LeftButtonDown:

```
int LeftButtonDown(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    // нажата левая кнопка
    mouse_down = WM_LBUTTONDOWN;
    // захватываем события мыши
    SetCapture(hWnd);
    // координаты мыши
    GetMouse(MX1, MY1, wParam, lParam);
    // текущая точка рисования
    MoveToEx(CDC, MX1, MY1, 0);
    // событие обработано
    return 0;
}
```

Функция LeftButtonUp:

```
int LeftButtonUp(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    // мышь отпущена
    mouse_down = 0;
    // освобождаем захват мыши
    ReleaseCapture();
    // событие обработано
    return 0;
}
```

Функция DrawMouse:

```
void DrawMouse(short & X1, short & Y1, short X2, short Y2) {
    // рисует отрезок прямой
    LineTo(CDC, X2, Y2);
    // обновляем изображение
    InvalidateRect(hWndGDE, 0, 0);
}
```

Остается только насладиться рисованием. Нажимаем кнопку первого инструмента, нажимаем кнопку мыши над поверхностью окна для рисования, держим кнопку и водим мышью.

Не спешите выбрать другой цвет. Цвет выберется, но перо останется прежним, созданное по умолчанию. Перо мы создадим позднее.

Теперь точно так же можно реализовать рисование правой кнопкой мыши. Для этого нужно описать две функции.

Функция RightButtonDown:

```
int RightButtonDown(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    // нажата правая кнопка
    mouse_down = WM_RBUTTONDOWN;
    // захватываем события мыши
    SetCapture(hWnd);
    // координаты мыши
    GetMouse(MX1, MY1, wParam, lParam);
    // текущая точка рисования
    MoveToEx(CDC, MX1, MY1, 0);
    // событие обработано
    return 0;
}
```

Функция RightButtonUp:

```
int RightButtonUp(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    // мышь отпущена
    mouse_down = 0;
    // освобождаем захват мыши
    ReleaseCapture();
    // событие обработано
    return 0;
}
```

Теперь правая кнопка тоже должна рисовать черным цветом. Чтобы цвет менялся при выборе в палитре, нужно выбирать перо в контекст:

Функция LeftButtonDown:

```
int LeftButtonDown(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    . . .
    // текущая точка рисования
    MoveToEx(CDC, MX1, MY1, 0);
    // выбираем перо плана
    SelectObject(CDC, PenFore);
    // событие обработано
    return 0;
}
```

Функция RightButtonDown:

```
int RightButtonDown(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    . . .
    // текущая точка рисования
    MoveToEx(CDC, MX1, MY1, 0);
    // выбираем перо фона
    SelectObject(CDC, PenBack);
    // событие обработано
    return 0;
}
```

Теперь должно рисоваться черным левой кнопкой и красным правой кнопкой мыши.

4.4. Область обновления

Все хорошо, за исключением одного. Функция InvalidateRect вызывает у нас перерисовку всего окна каждый раз, когда мы двигаем мышь, то есть если даже мы рисуем всего один пиксель.

Как эта функция работает? Если ее второй параметр равен нулю, она посылает сообщение WM_PAINT окну графического вывода, указывая, что перерисовать нужно все окно. Если же второй параметр задать, то в сообщении WM_PAINT будет задана область, которую нужно перерисовать.

Модуль Paint.cpp, функция GDProc:

```
LRESULT CALLBACK GDProc(HWND hWnd, UINT message, WPARAM wParam, . . . {
    . . .
    case WM_PAINT:
        /* перерисовка изображения */
        hdc = BeginPaint(hWnd, &ps);
        /* область обновления */
        L = ps.rcPaint.left;
        T = ps.rcPaint.top;
        W = ps.rcPaint.right - L;
        H = ps.rcPaint.bottom - T;
        /* копируем изображение из CDC в графическое устройство hdc */
        BitBlt(hdc, L, T, W, H, CDC, L, T, SRCCOPY);
        EndPaint(hWnd, &ps);
        return 0;
    }
```

Здесь видно, как вычисляется область обновления и затем из совместимого контекста CDC только эта область копируется в окно вывода.

Следовательно, мы должны вычислить область обновления. Делать это нужно в функции DrawMouse.

Сначала нужно объявить переменную rc типа RECT. Затем нужно вычислить область обновления. Поле rc.left — это минимум из X1 и X2, поле rc.top — это минимум из Y1 и Y2, поле rc.right — это максимум из X1 и X2, поле rc.bottom — это максимум из Y1 и Y2. Кроме того, нужно также учесть ширину пера. Для этого из rc.left и rc.top нужно вычесть pen_size, а к rc.right и rc.bottom прибавить pen_size.

Кроме того, нужно менять координаты X1 и Y1, заменяя их координатами X2 и Y2, иначе координаты X1 и Y1 всегда будут оставаться прежними. Когда все вычисления будут выполнены, изменим вызов InvalidateRect на следующий:

```
InvalidateRect(hWndGDE, &rc, 0);
```

Программируем вычисления и убеждаемся, что рисуется нормально, пиксели не пропадают. Когда мы сделаем перо шириной 25 пикселей, то проверим эти вычисления еще точнее.

4.5. Графические примитивы

Теперь попробуем нарисовать отрезок прямой линии, это наш второй инструмент. Функция DrawLine:

```
void DrawLine(short X1, short Y1, short X2, short Y2) {
    // устанавливаем текущую точку
    MoveToEx(CDC, X1, Y1, 0);
    // рисуем отрезок до нового положения
    LineTo(CDC, X2, Y2);
    // обновляем окно графического вывода
    InvalidateRect(hWndGDE, 0, 0);
}
```

Пробуем рисовать, нажав кнопку второго инструмента. Вид рисунка — прекрасный веер линий. Однако это не совсем то, что предполагалось.

В проекте есть еще один совместимый контекст, названный дополнительным. Перед началом рисования отрезка прямой линии сохраним изображение из CDC в дополнительный контекст. Перед рисованием отрезка восстановим изображение из дополнительного контекста в CDC.

Функции LeftButtonDown и RightButtonDown:

```
int XButtonDown(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    // нажата левая кнопка
    mouse_down = WM_XBUTTONDOWN;
    . . .
    // сохраняем изображение
    ImageSave();
    // событие обработано
    return 0;
}
```

Функция DrawLine:

```
void DrawLine(short X1, short Y1, short X2, short Y2) {
    // восстанавливаем изображение
    ImageRestore();
    // устанавливаем текущую точку
    MoveToEx(CDC, X1, Y1, 0);
    // рисуем отрезок до нового положения
    LineTo(CDC, X2, Y2);
    // обновляем окно графического вывода
    InvalidateRect(hWndGDE, 0, 0);
}
```

Пробуем этот вариант.

Убеждаемся, что интерактивно рисуется одна линия.

Теперь совсем несложно реализовать примитивы Rectangle и Ellipse, в соответствующих функциях. Самостоятельно. Вычислять область обновления для этих примитивов не будем.

Остается примитив «полилиния». Рисуется он так: нажимаем и отпускаем левую кнопку мыши, ведем линию, нажимаем и отпускаем левую кнопку мыши, ведем линию, нажимаем и отпускаем правую кнопку мыши и рисование завершается. На самом деле это не полилиния в том смысле, который определяет графическая подсистема, а просто несколько подряд рисуемых отрезков. Из них можно сделать полилинию, если запоминать точки всех концов отрезков, но это сложно, и мы так делать не будем.

Проблема полилинии в том, что для ее рисования нужно отпускать мышь, и соответственно, захват. Поэтому нужно использовать признаки pole_left и poly_right, которые устанавливаются при выборе инструмента «полилиния» и рисовании левой или правой кнопкой.

Тогда, если нажимается левая кнопка мыши, то если poly_right не установлено, то можно установить poly_left. После этого poly_right нужно очистить. И наоборот, если нажимается правая кнопка мыши и poly_left не установлено, то можно установить poly_right. После этого poly_left нужно очистить.

Для примера, функция LeftButtonDown:

```
int LeftButtonDown(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    // нажата левая кнопка
    mouse_down = WM_RBUTTONDOWN;
    . . .
    // сохраняем изображение
    ImageSave();
    // если полилиния
    if (painting_tool == TOOL_POLY && poly_right == 0) poly_left = 1;
    poly_right = 0;
    // событие обработано
    return 0;
}
```

Теперь нужно запретить освобождение захвата, если кнопка отпускается и установлен режим полилинии.

Для примера, функция LeftButtonUp:

```
int LeftButtonUp(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    // если полилиния, держим захват
    if (poly_left) return 0;
    // мышь отпущена
    mouse_down = 0;
    // освобождаем захват мыши
    ReleaseCapture();
    // событие обработано
    return 0;
}
```

Теперь полилиния должна рисоваться левой и правой кнопкой, и не должно оставаться лишних штрихов.

4.6. Перья

Если все инструменты рисуют, займемся перьями. Перья бывают косметические и геометрические. В приложении тип пера выбирается одной из кнопок, обозначенных буквами «К» и «Г»:

Перо создается в функции CreateNewPen:

Создадим косметическое перо. Сначала имеющееся перо нужно удалить при помощи функции DeleteObject. Затем создаем перо при помощи функции CreatePen, задавая стиль линии, ширину пера и цвет. Перо передается в функцию первым параметром, а цвет — вторым параметром:

```
void CreateNewPen(HPEN & Pen, COLORREF color) {
    // удаляем предыдущее перо
    DeleteObject(Pen);
    if (pen_type == PS_COSMETIC ) {
        // косметическое перо
        Pen = CreatePen(line_style, pen_size, color);
    } else {
        // геометрическое перо
    }
}
```

Тестируем косметическое перо, пробуя различные сочетания стиля линии, ширины пера, инструмента и цвета.

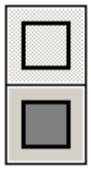
Создадим геометрическое перо. Оно сложнее, в нем есть кисть для закраски линии, кисть может быть сплошной или со штриховкой:

```
        // геометрическое перо
        PenBrush.lbColor = color;
        PenBrush.lbStyle = g_style;
        PenBrush.lbHatch = g_hatch;
        int style = PS_GEOMETRIC | line_style | pen_endcap | pen_jointy;
        Pen = ExtCreatePen(style, pen_size, &PenBrush, 0, 0);
```

Тестируем геометрическое перо. Здесь возможностей гораздо больше, все их нужно выучить. Если перо не включается, у вас в модуле drawing.h есть ограничитель типа пера `MAX_PEN_TYPE`, установите его значение 2.

4.7. Режим фона линий

Перейдем к режиму фона. Он может быть прозрачным и непрозрачным. Кнопки, управляющие режимом, находятся в приложении слева:



Прозрачный

Непрозрачный

Режим фона устанавливается в функции SetBackStyle.

Если режим прозрачный, вызываем функцию SetBkMode со вторым параметром, равным TRANSPARENT, иначе равным OPAQUE. Если режим непрозрачный, нужно также задать цвет фона функцией SetBkColor:

```
void SetBackStyle(COLORREF color) {
    if (backstyle_mode == 1) {
        /* режим фона прозрачный */
        SetBkMode(CDC, TRANSPARENT);
    } else {
        /* режим фона непрозрачный */
        SetBkMode(CDC, OPAQUE);
        // цвет фона
        SetBkColor(CDC, color);
    }
}
```

Режим фона нужно устанавливать всякий раз, когда начинается рисование, для примера, в функции LeftButtonDown:

```
int LeftButtonDown(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    . . .
    poly_right = 0;
    // режим фона
    SetBackStyle(back_color);
    // событие обработано
    return 0;
}
```

Тестируем непрозрачный режим при рисовании линий и фигур с шириной пера, равной 1 пикселю. Имеет значение стиль линии.

4.8. Режим фона фигур

Перейдем к режиму фона для закрашиваемых фигур, таких, как прямоугольник или эллипс. Эти фигуры закрашиваются текущей кистью контекста. По умолчанию в контекст выбрана нулевая кисть из стока. Эту же кисть нужно выбирать в контекст в прозрачном режиме.

В непрозрачном режиме в контекст также будем выбирать стоковую кисть, задавая ей необходимый цвет:

```
void SetBackStyle(COLORREF color) {
    if (backstyle_mode == 1) {
        . . .
        // выбираем в контекст нулевую кисть стока
        SelectObject(CDC, GetStockObject(NULL_BRUSH));
    } else {
        . . .
        // выбираем в контекст стандартную кисть стока
        SelectObject(CDC, GetStockObject(DC_BRUSH));
        // цвет кисти
        SetDCBrushColor(CDC, color);
    }
}
```

Тестируем закрашивание фигур.

4.9. Выравнивание линий

Чтобы рисовать ортогональные линии, нужно фиксировать координату X или Y, если во время движения клавиша Shift нажата.

Для этой цели в проекте определяется состояние клавиши Shift, и устанавливается переменная shift_on. В функции ShiftMouse вычисляется, в каком направлении идет движение, и сравниваются координаты X или Y:

```
void ShiftMouse(short X1, short Y1, short & X2, short & Y2) {
    int DX = X2 - X1;
    int DY = Y2 - Y1;
    int line = painting_tool == TOOL_LINE;
    int rect = painting_tool == TOOL_RECT || painting_tool == TOOL_OVAL
    if (!shift_on) return;
    if (abs(DX) > abs(DY)) {
        if (line) {
            Y2 = Y1;
        } else if (rect) {
        }
    } else {
        if (line) {
            X2 = X1;
        } else if (rect) {
        }
    }
}
```

С рисованием квадратов и кругов можете экспериментировать сами.

4.10. Контрольные вопросы и упражнения

1. Как создаются перья и кисти?
2. Как работает непрозрачный фон для разных примитивов?
3. Когда имеет значение тип концевой точки и точки соединения?

Работа КГ-105. Растровые алгоритмы

Цели:
- изучение алгоритма Брезенхейма;
Задачи:
- реализация алгоритма Брезенхейма.

5.1. Алгоритм Брезенхейма

Алгоритм Брезенхейма был разработан для рисования прямых линий графопостроителем, но нашел широкое применение в компьютерной графике не только для рисования прямых, но и кривых линий. Особенностью алгоритма является использование только целочисленных вычислений.

Вам нужно изучить алгоритм, изложенный в [3], и реализовать его для рисования прямых линий.

Реализуйте следующий алгоритм:

```
Процедура Брезенхэм(Цел X1, Y1, X2, Y2)
    Цел DX, DY, X, Y, D, D1, D2, SX, SY
    DX = Abs(X2 – X1)
    DY = Abs(Y2 – Y1)
    Если X1 < X2 То SX = 1 Иначе SX = –1
    Если Y1 < Y2 То SY = 1 Иначе SY = –1
    X = X1
    Y = Y1
    Точка(X, Y)
    Если DY > DX То -- движение по вертикали Y
        D1 = DX + DX
        D  = D1 – DY
        D2 = D  – DY
        Изменяя Y От Y1 + SY До Y = Y2 Шаг SY
            Если D > 0 То
                D = D + D2
                X = X + SX
            Иначе
                D = D + D1
            Конец Если
            Точка(X, Y)
        Следующее Y
    Иначе -- движение по горизонтали X
        D1 = DY + DY
        D  = D1 – DX
        D2 = D  – DX
        Изменяя X От X1 + SX До X = X2 Шаг SX
            Если D > 0 То
                D = D + D2
                Y = Y + SY
            Иначе
                D = D + D1
            Конец Если
            Точка(X, Y)
        Следующее X
    Конец Если
Конец процедуры
```

Здесь Точка — это вызов функции SetPixel, которая требует, кроме координат, цвет точки. Цвет можно вычислить, зная, что признак нажатия кнопки мыши mouse_down равен WM_LBUTTONDOWN или WM_RBUTTONDOWN. Если mouse_down равно WM_LBUTTONDOWN, то в качестве цвета используем цвет в переменной fore_color, иначе цвет в переменной back_color.

5.3. Рабочее пространство

Данная работа является продолжением предыдущей, и выполняется в проекте Paint. Алгоритм Брезенхейма реализуется в функции DrawBrez.

5.3. Контрольные вопросы и упражнения

1. В чем заключается принцип, заложенный в алгоритм Брезенхейма?

2. Попробуйте реализовать алгоритм Ву для рисования сглаженных линий, функция для этого алгоритма называется DrawBWoo.

Литература

1. Пономарев В.В. Компьютерная графика. Учебное пособие. Часть 1. Вводный курс. Редакция 2. Озерск: ОТИ МИФИ, 2006, 122 с.

2. Пономарев Вл. Microsoft Visual C++ 6.0. Приложение Windows с нуля. Учебное пособие. Озерск: ОТИ МИФИ, 2006. — 59 с., ил.

3. Пономарев В.В. Машинная графика. Методические пособие по дисциплине «Компьютерная графика». Часть 2. Озерск: ОТИ МИФИ, 2005. — 70 с. ил.: 84.