**Лабораторная работа №2**

Тема: Чтение и запись графических изображений в формате **BMP**.

**1. Растровая и векторная графика**

**Визуализация изображений**

Наиболее известны два способа визуализации: **растровый** и **векторный**. Первый способ ассоциируется с такими графическими устройствами как дисплей, телевизор, принтер. Второй используется в векторных дисплеях, плоттерах.

Наиболее удобно, когда способ описания графического изображения соответствует способу визуализации. Иначе нужна конвертация. Например, изображение может храниться в растровом виде, а его необходимо вывести (визуализировать) на векторном устройстве. Для этого нужна предварительная **векторизация** – преобразование из растрового в векторное описание. Или, наоборот, описание изображения может быть в векторном виде, а нужно визуализировать на растровом устройстве – необходима **растеризация**.

Растровая визуализация основывается на представлении изображения на экране или бумаге в виде совокупности (массива) отдельных точек (пикселов). Вместе пикселы образуют **растр**.

Для обозначения массива пикселов часто используется термин **bitmap** (битовая карта). В bitmap каждому пикселу отводится определенное число битов (одинаковое для всех пикселов изображения). Это число называется битовой глубиной пиксела или цветовой глубиной изображения, т. к. от количества битов, отводимых на один пиксел, зависит количество цветов изображения.

Построение векторного изображения основано на так называемых опорных точках, которые соединены между собой кривыми, определяемыми соответствующими математическими алгоритмами. Работая с векторным изображением, пользователь задаёт его опорные точки и характер векторных кривых между ними.

Доминирующим сейчас является растровый способ визуализации. Это обусловлено большей распространенностью растровых дисплеев и принтеров. Недостаток растровых устройств – дискретность изображения. Недостатки векторных устройств – проблемы при сплошном заполнении фигур, меньшее количество цветов, меньшая скорость (в сравнении с растровыми устройствами).

**Растровые изображения**

Плюсы: четко и максимально правдоподобно отображает оттенки цветов, их перетекание из одного в другой, а также тени.

Минусы: При увеличении заметно теряет в четкости и выглядит некачественно.

Применение: Применяются, как правило, при работе с фотографиями и другими изображениями с насыщенной цветовой гаммой и плавными переходами цвета. Активно применялась при дизайне сайтов, иконок приложений. Правда сейчас, когда flat и material дизайн стали так популярны, дизайнеры все чаще использую векторные программы для своих творений.



**Векторные изображения**

Плюсы: масштабирование без потери четкости изображения. Малый размер изображений.

Минусы: очень сложно передать плавные переходы цвета и добиться фотографического качества

Применение: Применяется при создании логотипов компаний, визиток, буклетов и иной печатной продукции. Также редакторы векторной графики незаменимы при создании новых, оригинальных шрифтов. Но и это еще не все. В редакторах векторной графики можно создавать прекрасные иллюстрации.



**2. bmp-файл, его структура, реализация этой структуры на C++**

**Формат bmp**

Формат bmp (от слов BitMap Picture - битовая карта, или, говоря по-русски, битовый массив) представляет из себя несжатое (в основном) изображение, которое довольно легко читается и выводится в ОС Windows, в которой есть специальные функции API, которые в этом помогают. Структура файла BMP используется Windows для хранения растровых изображений.

**Формат bmp-файлов** представлен на рисунке ниже.



Файл, содержащий битовое изображение, начинается со структуры **BITMAPFILEHEADER**. Эта структура описывает тип файла и его размер, а также смещение области битов изображения.

typedef struct **tagBITMAPFILEHEADER**

{

WORD bfType; //признак bmp-файла, 42 4D (коды букв ВМ)

DWORD bfSize; //размер файла

WORD bfReserved1; //первое резервное поле = ноль

WORD bfReserved2; //второе резервное поле = ноль

DWORD bfOffBits; //смещение от начала файла до первого байта графических данных

} BITMAPFILEHEADER, \*PBITMAPFILEHEADER;

Сразу после структуры BITMAPFILEHEADER в файле следует структура **BITMAPINFO**, которая содержит описание изображения и таблицу цветов.

typedef struct tagBITMAPINFO {

BITMAPINFOHEADER bmiHeader;

RGBQUAD bmiColors[1];

} BITMAPINFO, \*PBITMAPINFO;

Описание изображения (размеры изображения, метод компрессии, размер таблицы цветов и т. д.) находится в структуре BITMAPINFOHEADER.

typedef struct **tagBITMAPINFOHEADER**{

DWORD biSize; //размер самой структуры BITMAPINFOHEADER в байтах.

LONG biWidth; //ширина картинки в пикселах

LONG biHeight; //высота картинки в пикселах

WORD biPlanes; //количество битовых плоскостей (=1)

WORD biBitCount; //количество бит на пиксел (определяет максимальное число цветов в bitmap)

DWORD biCompression; //способ сжатия (0 – данные не сжимаются).

DWORD biSizeImage; //размер изображения в байтах (без заголовков)

LONG biXPelsPerMeter; //число пикселов на метр по горизонтали

LONG biYPelsPerMeter; //число пикселов на метр по вертикали

DWORD biClrUsed; //количество элементов палитры, хранящихся после заголовка

DWORD biClrImportant; //количество разных цветов, действительно используемых в рисунке

} BITMAPINFOHEADER, \*PBITMAPINFOHEADER;

В некоторых случаях (не всегда) в файле может присутствовать палитра–таблица цветов (как массив структур RGBQUAD), присутствующих в изображении.

typedef struct **tagRGBQUAD**

{

BYTE rgbBlue;

BYTE rgbGreen;

BYTE rgbRed;

BYTE rgbReserved;

} RGBQUAD;

После палитры (если она есть) в файле BMP записывается растр в виде би-тового (а точнее, байтового массива). В битовом массиве последовательно запи-сываются байты строк растра. Количество байт в строке должно быть кратно че-тырем, поэтому если количество пикселов по горизонтали не соответствует та-кому условию, то справа в каждую строку дописывается некоторое число битов (выравнивание строк на границу двойного слова).

**Реализация структуры bmp-файла на C++** находится в файле TestBMP.cpp в функции ClientToBmp.

**3. Загрузка, сохранение изображений, выделение области и т.д.**

**Общий алгоритм работы загрузки изображения:**

1. Получение параметров переданного изображения.

2. Выделение памяти для отображения переданного изображения.

3. Передача изображение в окно.

**Общий алгоритм работы записи выделенной области изображения:**

1. Получение координат точек левой верхней и правой нижней для захвата области выделения (при нажатии левой кнопки мыши мы захватываем точку левой вехней области изображения, при отпускании мы получаем правую нижнюю точку области изображения).

2. Получение информации хранимой в пределах данных двух точек.

3. Формирование файла для сохранения.

4. Сохранения файла в директорию.

**Загрузка изображения**

void CLab01App::LoadImageBMP(CPoint point) // загрузка изображения

{

CFileDialog fileDialog(TRUE); //объект класса выбора файла

int result = fileDialog.DoModal(); //запустить диалоговое окно

if (result == IDOK) //если файл выбран

{

AfxMessageBox(fileDialog.GetPathName()); // показать полный путь

CWnd \*cwnd = AfxGetMainWnd(); получить указатель на активное главное окно приложения

hwnd = cwnd->GetSafeHwnd(); // Возвращает дескриптор окна m\_hWnd или NULL если указатель thisNULL.

hBit = (HBITMAP)LoadImage(NULL, fileDialog.GetPathName(), IMAGE\_BITMAP,

0, 0, LR\_LOADFROMFILE | LR\_CREATEDIBSECTION); //загрузка изображения

ShowBitMap(hwnd, hBit, point.x, point.y); //вывод изображения

}

}

В MFC есть стандартный класс для открытия и сохранения файлов **CFileDialog**. Смотрите, в функции LoadImageBMP(CPoint point) объявляем класс, первый параметр простой TRUE - диалог открытия, FALSE - диалог сохранения. Следующий параметр устанавливаем в NULL. И последний параметр позволяет задать расширение файлов, используемое по умолчанию. Если пользователь не укажет расширение файла явным образом, ему автоматически присваивается расширение, принятое по умолчанию.

Создание объекта класса CFileDialog еще не вызывает отображения соответствующей диалоговой панели. Для этого необходимо воспользоваться специальным методом **DoModal** класса CFileDialog.

При вызове метода DoModal для ранее созданного объекта класса CFileDialog на экране открывается соответствующая диалоговая панель. После того, как вы закончите работать с диалоговой панелью и закроете ее, метод DoModal вернет значения IDOK или IDCANCEL в случае успешного завершения и 0 - в случае возникновения ошибок.

Если вы выберите имя файла и нажмете кнопку Open (для панели Open) или кнопку Save (для панели Save As), метод DoModal вернет значение IDOK. Если вы решите отменить выбор файла и нажмете кнопку Cancel или выберите из меню диалоговой панели строку Close, метод DoModal вернет значение IDCANCEL.

После того, как вы закроете диалоговую панель и метод DoModal вернет управление, можно воспользоваться другими методами класса CFileDialog, чтобы определить имена выбранных файлов. В нашем случае:

|  |  |
| --- | --- |
| Метод | Описание |
| **GetPathName** | Определяет полный путь выбранного файла |

HWND **GetSafeHwnd**( ) const;

Возвращает дескриптор окна для окна. Возвращает NULL, если CWnd не присоединен к окну или если указатель на объект CWnd - NULL.

**AfxGetMainWnd** Возвращает указатель на текущее «главное» окно приложения, отличного от OLE, или окно фрейма на месте серверного приложения.

**LoadImage** function загружает (читает) значок, курсор, анимированный курсор или растровое изображение.

HANDLE LoadImageA(

[in, optional] HINSTANCE hInst,

[in] LPCSTR name,

[in] UINT type,

[in] int cx,

[in] int cy,

[in] UINT fuLoad

);

Если функция завершается успешно, возвращаемое значение является дескриптором только что загруженного изображения.

И последняя вызванная функция пользовательская:

int ShowBitMap(HWND hWnd, HANDLE hBit, int x, int y)

{

// Функция отображает рисунок в заданной позиции окна

// hWnd - дескриптор окна, куда выводится изображение

// hBit - дескриптор рисунка

// (x,y) - координаты левого верхнего угла изображения в окне вывода

BITMAP BitMap; // BITMAP - структура, которая определяет параметры растрового изображения

GetObjectW(hBit, sizeof(BITMAP), &BitMap); // получаем параметры изображения в структуру BitMap

int Height = BitMap.bmHeight; // получаем высоту изображения

int Width = BitMap.bmWidth; // получаем ширину изображения

HDC hdc = GetDC(hWnd); // извлечение контекста изображения

HDC hdcMem = CreateCompatibleDC(hdc); // создание контекста памяти

HBITMAP OldBitmap = (HBITMAP)SelectObject(hdcMem, hBit); // в созданный контекст памяти заносим дескриптор битовой карты

BitBlt(hdc, x, y, Width, Height, hdcMem, 0, 0, SRCCOPY); // в окно, с которым связан контекст изображения, картинка переносится копированием

SelectObject(hdcMem, OldBitmap); // после копирования уничтожаются контексты памяти и изображения

ReleaseDC(hWnd, hdc);

return 0;

}

Функция **BitBlt** выполняет передачу битовых блоков данных о цвете, соответствующих прямоугольнику пикселей из заданного исходного контекста устройства в целевой контекст устройства.

BOOL BitBlt(

HDC hdcDest, // дескриптор целевого DC

int nXDest, // x-коорд. левого верхнего угла целевого прямоугольника

int nYDest, // y-коорд. левого верхнего угла целевого прямоугольника

int nWidth, // ширина целевого прямоугольника

int nHeight, // высота целевого прямоугольника

HDC hdcSrc, // дескриптор исходного DC

int nXSrc, // x-коорд. левого верхнего угла исходного прямоугольника

int nYSrc, // y-коорд. левого верхнего угла исходного прямоугольника

DWORD dwRop // код растровой операции

);

Функция **SelectObject** выбирает объект в заданный контекст устройства (DC). Новый объект заменяет предыдущий объект того же самого типа.