**Графический интерфейс GDI в Microsoft Windows**

*© Александр Фролов, Григорий Фролов  
Том 4, М.: Диалог-МИФИ, 1993, 288 стр.*

|  |  |  |
| --- | --- | --- |
| [[Назад]](http://frolov-lib.ru/books/bsp/v14/ch1.htm) | [[Содеожание]](http://frolov-lib.ru/books/bsp/v14/index.html) | [[Дальше]](http://frolov-lib.ru/books/bsp/v14/ch3.htm) |

**2.3. Рисование геометрических фигур**

В этом разделе мы расскажем вам об использовании функций, предназначенных для рисования точек, линий, окружностей и других геометрических фигур.

Несмотря на то, что в программном интерфейсе GDI имеется функция SetPixel , позволяющая нарисовать один пиксел, не следует думать, что рисование линии или окружности сводится к многократному вызову этой функции. Если бы это было так, процесс рисования занимал бы очень много времени. На самом деле многие из функций рисования выполняются драйвером или даже аппаратурой видеоконтроллера, что значительно ускоряет вывод.

С помощью функции GetDeviceCaps приложение может определить, поддерживает ли драйвер ту или иную функцию рисования.

Первый параметр функции hdc задает контекст устройства, для которого необходимо получить информацию о его возможностях.

Второй параметр iCapability определяет параметр устройства, значение которого необходимо получить.

Приведем список значений для второго параметра функции GetDeviceCaps, с помощью которых можно определить, какие операции рисования выполняет драйвер устройства вывода.

|  |  |
| --- | --- |
| Имя константы | Описание |
| LINECAPS | Способности устройства рисовать линии. Возвращаемое значение представляет собой набор битовых масок, установленных в 1, если устройство может само рисовать линии различного типа:LC\_INTERIORS устройство может закрашивать внутреннюю область;LC\_MARKER маркеры;LC\_NONE устройство не может рисовать линии;LC\_POLYLINE ломаные линии;LC\_POLYMARKER линии polymarker;LC\_STYLED устройство может рисовать линии с использованием различных стилей (штриховые, пунктирные, штрих пунктирные и т.д.);LC\_WIDE широкие линии;LC\_WIDESTILED устройство может рисовать широкие линии с использованием различных стилей (штриховые, пунктирные, штрих-пунктирные и т. д.) |
| CURVECAPS | Способность устройства рисовать различные кривые линии и геометрические фигуры. Возвращаемое значение представляет собой набор битовых масок, установленных в 1, если устройство может само рисовать различные фигуры:CC\_CIRCLES окружности;CC\_CHORD сегмент эллипса;CC\_ELLIPSES эллипсы;CC\_INTERIORS устройство может закрашивать внутреннюю область геометрических фигур;CC\_NONE устройство не может рисовать кривые линии и геометрические фигуры;CC\_PIE секторы эллипса;CC\_ROUNDRECT прямоугольники со скругленными углами;CC\_STYLED устройство может рисовать рамки с использованием различных стилей (штриховые, пунктирные, штрих-пунктирные и т.д.);CC\_WIDE широкие рамки;CC\_WIDESTYLED устройство может рисовать широкие рамки с использованием различных стилей (штриховые, пунктирные, штрих-пунктирные и т. д.) |
| POLYGONALCAPS | Способности устройства рисовать многоугольники. Возвращаемое значение представляет собой набор битовых масок, установленных в 1, если устройство может само рисовать многоугольники различного типа:PC\_INTERIORS устройство может закрашивать внутреннюю область;PC\_NONE устройство не может рисовать многоугольники;PC\_RECTANGLE прямоугольники;PC\_SCANLINES устройство может выполнять сканирование линий растра;PC\_STYLED устройство может рисовать рамки с использованием различных стилей (штриховые, пунктирные, штрих-пунктирные и т. д.);PC\_WIDE широкие рамки;PC\_WIDESTILED устройство может рисовать широкие рамки с использованием различных стилей (штриховые, пунктирные, штрих-пунктирные и т. д.)PC\_WINDPOLYGON многоугольники с заполнением в режиме WINDING |

Для приложения не имеет особого значения, кто именно будет рисовать - видеоконтроллер, драйвер или GDI. Запрос на рисование, например, эллипса, будет выполнен, даже если соответствующая операция не поддерживается драйвером. В последнем случае эллипс будет нарисован самим GDI с использованием более примитивных операций, но процесс рисования займет больше времени.

Учитывая сказанное выше, не следует строить работу приложений таким образом, чтобы периодичность вывода или скорость работы приложения зависела от скорости рисования (подобная практика не приветствуется и при создании программ для MS-DOS, вспомните, как ведут себя старые игры, разработанные для процессора 8088, на компьютерах с процессорами i80386 или i486). Современные видеоадаптеры сконструированы таким образом, что большинство основных операций рисования, используемых в операционной системе Windows, выполняются аппаратно. Эти видеоадаптеры иногда называются ускорителями Windows. Скорость рисования для ускорителя Windows может превышать в десятки раз скорость рисования для обычного адаптера VGA или SVGA.

Результат рисования геометрических фигур зависит от установки таких атрибутов контекста, как ширина, цвет и стиль линии (определяются выбранным в контекст отображения пером), способ закраски замкнутых фигур (определяется выбранной в контекст отображения кистью), цвета фона, прозрачностью фона (прозрачный режим TRANSPARENT и непрозрачный режим OPAQUE ), режимом рисования, режимом закрашивания, областью ограничения, режимом отображения, т. е. практически от всех атрибутов контекста отображения. Поэтому при описании функций мы будем попутно описывать способы изменения атрибутов контекста отображения, влияющих на результат их выполнения.

Работа с цветовыми палитрами и битовыми изображениями будут рассмотрены позже в отдельных разделах, так как эти вопросы далеко не тривиальны и поэтому заслуживают отдельного обсуждения.

Итак, перейдем непосредственно к описанию функций рисования геометрических фигур.

**Рисование точки**

Функция рисования точки SetPixel устанавливает цвет точки с заданными координатами:

COLORREF WINAPI SetPixel(

HDC hdc, // контекст отображения

int nXPos, // x-координата точки

int nYPos, // y-координата точки

COLORREF clrref); // цвет точки

Установка первых трех параметров этой функции не должна вызывать у вас никаких затруднений. Параметр hdc определяет контекст отображения, для которого необходимо изменить цвет точки. Параметры nXPos и nYPos определяют координаты точки в системе координат, которая зависит от установленного для контекста hdc режима отображения.

Последний параметр clrref определяет новый цвет точки. О том, как "раскрасить" окно приложения, вы узнаете из третьей главы нашей книги. Тем не менее мы опишем самый простой способ для функции SetPixel.

В файле windows.h есть описание макрокоманды RGB , позволяющей сконструировать цвет в формате COLORREF из отдельных компонент красного (r), зеленого (g) и голубого (b) цвета:

#define RGB(r,g,b) \

((COLORREF)(((BYTE)(r)|((WORD)(g)<<8)) | \

(((DWORD)(BYTE)(b))<<16)))

Вы можете использовать эту макрокоманду совместно с функцией SetPixel для установки, например, красного цвета точки, расположенной в начале системы координат (0,0), следующим образом:

SetPixel(hdc, 0, 0, RGB(0xff, 0, 0));

Три параметра макрокоманды RGB позволяют задать любой из примерно 16 млн. цветов и оттенков, однако это не означает, что вы получите на экране точно такой цвет, какой был задан при помощи этой макрокоманды. Скорее всего вы сможете воспользоваться одним из 20 системных цветов. Причины этого вы узнаете в третьей главе. Там же мы расскажем о том, как в некоторых случаях можно расширить используемую гамму цветов.

Функция SetPixel возвращает цвет, который фактически был использован для рисования точки. Как мы только что заметили, возвращенное значение может отличаться от заданного параметром clrref. В случае ошибки оно будет равно -1.

Функция GetPixel позволяет узнать цвет точки, заданной идентификатором контекста отображения и координатами:

COLORREF WINAPI GetPixel(HDC hdc, int nXPos, int nYPos);

С помощью следующих трех макрокоманд, определенных в файле windows.h, вы можете определить отдельные цветовые компоненты для значения, возвращаемого функциями SetPixel и GetPixel:

#define GetRValue (rgb) ((BYTE)(rgb))

#define GetGValue (rgb) ((BYTE)(((WORD)(rgb)) >> 8))

#define GetBValue (rgb) ((BYTE)((rgb)>>16))

Функции SetPixel и GetPixel используются достаточно редко, так как для построения графических изображений есть более мощные функции.

**Рисование линий**

Приложения Windows могут рисовать прямые и ломаные линии, а также дуги эллипса (и окружности, как частного случая эллипса). Параметры этих линий определяются несколькими атрибутами контекста отображения. Это режим отображения, влияющий на используемую систему координат, цвет фона, режим фона (прозрачный или непрозрачный), режим рисования, цветовая палитра (в режимах, использующих цветовую палитру), перо (может иметь различный цвет, толщину и стиль).

**Текущая позиция пера**

Для рисования прямых линий (и только для этого) в контексте отображения хранятся координаты текущей позиции пера . Для изменения текущей позиции пера в Windows версии 3.1 есть две функции с именами MoveTo и MoveToEx . Для совместимости с 32-разрядными версиями Windows, такими, как Windows NT, в новых приложениях следует использовать функцию MoveToEx:

BOOL WINAPI MoveToEx(

HDC hdc, // идентификатор контекста отображения

int x, // x-координата

int y, // y-координата

POINT FAR\* lppt); // указатель на структуру POINT

Для контекста отображения hdc эта функция устанавливает текущую позицию пера, равную (x,y). В структуру типа POINT, на которую указывает параметр lppt, после возврата из функции будут записаны старые координаты пера.

Функция MoveToEx возвращает TRUE при нормальном завершении или FALSE при ошибке.

Чтобы узнать текущую позицию пера, приложение может использовать функцию GetCurrentPositionEx :

BOOL WINAPI GetCurrentPositionEx(HDC hdc, POINT FAR\* lppt);

После вызова этой функции текущая позиция пера будет записана в структуру типа POINT, на которую указывает параметр lppt. Функция GetCurrentPositionEx возвращает TRUE при нормальном завершении или FALSE при ошибке.

**Рисование прямой линии**

Для того чтобы нарисовать прямую линию, приложение должно воспользоваться функцией LineTo :

BOOL WINAPI LineTo(HDC hdc, int xEnd, int yEnd);

Эта функция рисует линию из текущей позиции пера, установленной ранее функцией MoveTo или MoveToEx, в точку с координатами (xEnd,yEnd). После того как линия будет нарисована, текущая позиция пера станет равной (xEnd,yEnd).

Функция LineTo возвращает TRUE при нормальном завершении или FALSE при ошибке.

Таким образом, для того чтобы нарисовать прямую линию, приложение должно сначала с помощью функции MoveToEx установить текущую позицию пера в точку, которая будет началом линии, а затем вызвать функцию LineTo, передав ей через параметры xEnd и yEnd координаты конца линии.

Особенностью функции LineTo является то, что она немного не дорисовывает линию - эта функция рисует всю линию, не включая ее конец, т. е. точку (xEnd,yEnd). Это иллюстрируется на рис. 2.11.

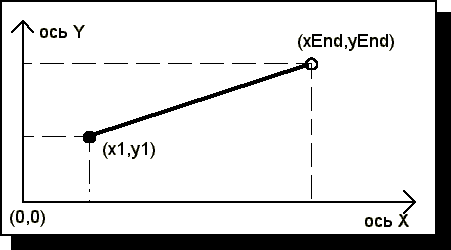


Рис. 2.11. Рисование прямой линии

Если вас не устраивает необходимость пользоваться двумя функциями для рисования линии, вы можете создать свою собственную, например такую:

BOOL DrawLine(HDC hdc, int x1, int y1, int x2, int y1)

{

POINT pt;

MoveToEx(hdc, x1, y1, &pt);

return LineTo(hdc, x2, y2);

}

Преимущества использования отдельных функций для установки текущей позиции и для рисования линии из текущей позиции в заданную точку с последующим изменением текущей позиции проявляются при рисовании ломаных линий. В этом случае вы можете только один раз установить текущую позицию пера на начало ломаной линии и в дальнейшем вызывать только функцию LineTo, передавая ей координаты точек излома линии. Однако для рисования ломаных линий (если известны координаты всех точек излома) больше подходит функция Polyline, которую мы рассмотрим в следующем разделе.

**Рисование ломаной линии**

Функции Polyline , предназначенной для рисования ломаных линий, следует передать идентификатор контекста отображения hdc, указатель lppt на массив структур POINT, в котором должны находится координаты начала ломаной линии, координаты точек излома и координаты конца ломаной линии, а также размер этого массива cPoints:

BOOL WINAPI Polyline(

HDC hdc, // идентификатор контекста отображения

const POINT FAR\* lppt,// указатель на массив структур POINT

int cPoints); // размер массива

Функция Polyline возвращает TRUE при нормальном завершении или FALSE при ошибке. Она не использует текущую позицию пера и не изменяет ее.

Если ломаная линия не замкнута, ее последняя точка не рисуется (рис. 2.12).

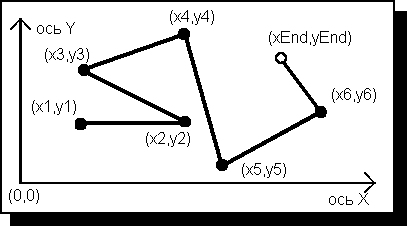


Рис. 2.12. Рисование ломаной линии

**Рисование дуги эллипса**

К сожалению, возможности рисования кривых линий при помощи функций GDI ограничены - единственная функция Arc позволяет нарисовать дугу эллипса или окружности:

BOOL WINAPI Arc(

HDC hdc, // идентификатор контекста отображения

int nxLeft, int nyTop, // верхий левый угол

int nxRight, int nyBottom, // правый нижний угол

int nxStart, int nyStart, // начало дуги

int nxEnd, int nyEnd); // конец дуги

Первый параметр этой функции определяет контекст отображения, в котором будет нарисована дуга. Для объяснения назначения остальных параметров обратимся к рис. 2.13.

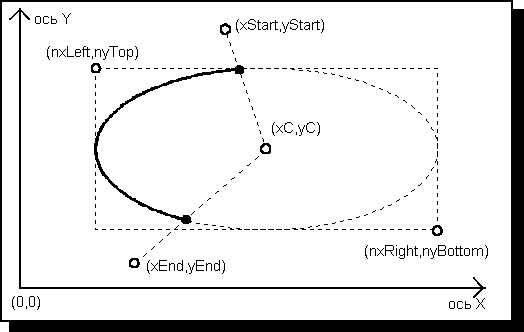


Рис. 2.13. Рисование дуги эллипса

Параметры (nxLeft,nyTop) и (nxRight,nyBottom) задают координаты, соответственно, верхнего левого и правого нижнего углов воображаемого прямоугольника, в который вписан эллипс.

Начало дуги эллипса определяется пересечением эллипса с воображаемой прямой линией, проведенной из центра эллипса (xC,yC) в точку (xStart,yStart).

Конец дуги определяется аналогично - как пересечение эллипса с воображаемой прямой линии, проведенной из центра эллипса в точку (xEnd,yEnd).

Дуга рисуется в направлении против часовой стрелки.

Координаты центра эллипса (если это потребуется) можно вычислить следующим образом:

xC = (nxLeft + nxRight) / 2;

yC = (nyTop + nyBottom) / 2;

**Настройка атрибутов контекста отображения для рисования линий**

Как это нетрудно заметить, функции, предназначенные для рисования линий, не имеют никаких параметров, определяющих толщину, цвет и стиль линии. Эти, а также другие характеристики (например, режим прозрачности), выбираются при установке соответствующих атрибутов контекста отображения.

**Выбор пера**

Для рисования линий приложения Windows могут выбрать одно из трех встроенных перьев, либо создать собственное перо.

Для выбора встроенного пера лучше всего воспользоваться макрокомандами GetStockPen и SelectPen , определенными в файле windowsx.h:

#define GetStockPen(i) ((HPEN)GetStockObject(i))

#define SelectPen(hdc, hpen) \

((HPEN)SelectObject((hdc), (HGDIOBJ)(HPEN)(hpen)))

Макрокоманда GetStockPen возвращает идентификатор встроенного пера, заданного параметром i. Вы можете выбрать для этого параметра одно из следующих значений:

|  |  |
| --- | --- |
| Значение | Описание |
| BLACK\_PEN | Перо, рисующее черную линию толщиной в один пиксел (для любого режима отображения). Это перо выбрано в контекст отображения по умолчанию |
| WHITE\_PEN | Перо белого цвета. Толщина пера также равна одному пикселу и не зависит от режима отображения |
| NULL\_PEN | Невидимое перо толщиной в один пиксел. Используется для рисования замкнутых закрашенных фигур (таких, как прямоугольник или эллипс) в тех случаях, когда контур фигуры должен быть невидимым |

После получения идентификатора пера его необходимо выбрать в контекст отображения при помощи макрокоманды SelectPen. Первый параметр этой макрокоманды используется для указания идентификатора контекста отображения, в который нужно выбрать перо, второй - для передачи идентификатора пера.

Макрокоманда SelectPen возвращает идентификатор пера, который был выбран в контекст отображения раньше. Вы можете сохранить этот идентификатор и использовать его для восстановления старого пера.

Однако при помощи встроенных перьев вы не можете нарисовать цветные, широкие, штриховые и штрих-пунктирные линии.

Если вас не устраивают встроенные перья, вы можете легко создать собственные. Для этого нужно воспользоваться функциями CreatePen или CreatePenIndirect.

Функция CreatePen позволяет определить стиль, ширину и цвет пера:

HPEN WINAPI CreatePen(

int fnPenStyle, // стиль пера

int nWidth, // ширина пера

COLORREF clrref); // цвет пера

Параметр fnPenStyle определяет стиль линии и может принимать одно из следующих значений, определенных в файле windows.h:

|  |  |  |
| --- | --- | --- |
| Стиль линии | Внешний вид | Описание |
| PS\_SOLID | **http://frolov-lib.ru/books/bsp/v14/img00021.gif** | Сплошная |
| PS\_DASH | **http://frolov-lib.ru/books/bsp/v14/img00022.gif** | Штриховая |
| PS\_DOT | **http://frolov-lib.ru/books/bsp/v14/img00023.gif** | Пунктирная |
| PS\_DASHDOT | **http://frolov-lib.ru/books/bsp/v14/img00024.gif** | Штрих-пунктирная, одна точка на одну линию |
| PS\_DASHDOTDOT | **http://frolov-lib.ru/books/bsp/v14/img00025.gif** | Штрих-пунктирная, две точки на одну линию |
| PS\_NULL |  | Невидимая |
| PS\_INSIDEFRAME | **http://frolov-lib.ru/books/bsp/v14/img00026.gif** | Линия, предназначенная для обводки замкнутых фигур |

Параметр nWidth определяет ширину пера. Используемая при этом единица длины зависит от режима отображения, поэтому вы можете задавать ширину пера не только в пикселах, но и в долях миллиметра или дюйма. Учтите, что для линий PS\_DASH, PS\_DOT, PS\_DASHDOT, PS\_DASHDOTDOT можно использовать только единичную или нулевую ширину, в обоих случаях ширина линии будет равна одному пикселу. Поэтому вы не можете создать перо для рисования, например, пунктирной линии шириной 5 миллиметров.

Параметр clrref задает цвет пера.

На первый взгляд линии PS\_SOLID и PS\_INSIDEFRAME похожи, однако между ними имеются различия, особенно заметные для широких линий. Широкая линия, имеющая стиль PS\_SOLID, располагается по обе стороны оси, заданной координатами линии. Линии, имеющие стиль PS\_INSIDEFRAME, располагаются внутри контура, определяющего размеры замкнутой фигуры (рис. 2.14).

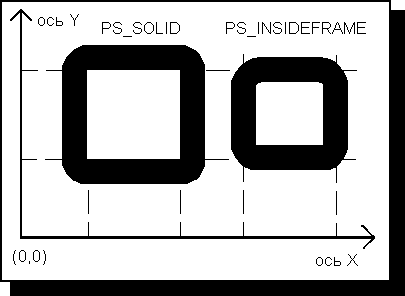


Рис. 2.14. Использование стилей PS\_SOLID и PS\_INSIDEFRAME

Еще одно отличие связано с использованием смешанных цветов (dithered color). Когда Windows не может в точности подобрать цвет, указанный для толстой линии стиля PS\_INSIDEFRAME, он раскрашивает эту линию с использованием смешанного цвета, полученного из других цветов. В этом случае изображение линии формируется из отдельных точек разных цветов. Техника смешанных цветов может применяться и при закрашивании замкнутых фигур, о чем мы еще будем говорить.

При рисовании тонких линий, а также линий, имеющих другой стиль, используются только чистые цвета.

Небольшое замечание относительно концов толстых линий. Концы толстых линий закруглены (рис. 2.15).

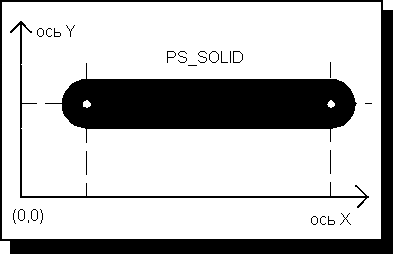


Рис. 2.15. Закругленные концы толстой линии

Для изображения толстой линии с прямыми концами следует задать прямоугольную область ограничения (см. ниже раздел, посвященный области ограничения). Можно также нарисовать толстую линию как закрашенный прямоугольник с использованием тонкого пера.

Другая возможность создать перо - вызвать функцию CreatePenIndirect :

HPEN WINAPI CreatePenIndirect(LOGPEN FAR\* lplgpn);

Эта функция работает аналогично функции CreatePen, однако в качестве параметра ей необходимо передать указатель на структуру типа LOGPEN, в которой должны находиться характеристики создаваемого пера.

Структура LOGPEN и различные указатели на нее определены в файле windows.h:

typedef struct tagLOGPEN

{

UINT lopnStyle; // стиль пера

POINT lopnWidth; // ширина пера

COLORREF lopnColor; // цвет пера

} LOGPEN;

typedef LOGPEN\* PLOGPEN;

typedef LOGPEN NEAR\* NPLOGPEN;

typedef LOGPEN FAR\* LPLOGPEN;

Заметим, что ширина пера в данном случае находится в поле x структуры POINT. Поле y не используется.

Если вы создали перо, его можно выбрать в контекст отображения при помощи макрокоманды SelectPen. После этого можно рисовать линии обычным образом, вызывая функции MoveToEx и LineTo.

Созданные вашим приложением перья принадлежат GDI, соответствующие структуры данных располагаются в его сегменте данных. Поэтому если перо больше не нужно, его нужно удалить для освобождения памяти.

Прежде чем удалять созданное вами перо, следует выбрать в контекст отображения одно из встроенных перьев (например то, которое использовалось раньше). После этого для удаления вашего пера нужно вызвать макрокоманду DeleletePen , определенную в файле windowsx.h:

#define DeletePen(hpen) DeleteObject((HGDIOBJ)(HPEN)(hpen))

В качестве параметра этой макрокоманде необходимо передать идентификатор удаляемого пера.

Нельзя удалять перо, если оно выбрано в контекст отображения. Нет никакого смысла в удалении встроенных перьев.

**Выбор режима фона**

Режим фона влияет на заполнение промежутков между штрихами и точками в штрих-пунктирных, штриховых и пунктирных линиях.

Напомним, что по умолчанию в контексте отображения установлен непрозрачный режим фона OPAQUE . В этом режиме промежутки закрашиваются цветом фона, определенным как атрибут контекста отображения. Приложение может установить прозрачный режим фона TRANSPARENT , в этом случае промежутки в линиях не будут закрашиваться (рис. 2.16).

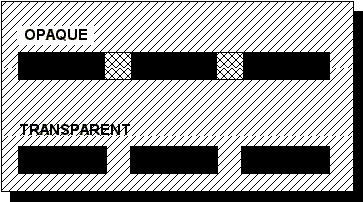


Рис. 2.16. Режимы фона OPAQUE и TRANSPARENT

Для установки режима фона предназначена функция SetBkMode :

int WINAPI SetBkMode(HDC hdc, int fnBkMode);

Эта функция устанавливает новый режим фона fnBkMode для контекста отображения hdc, возвращая в случае успеха код старого режима фона.

Для параметра fnBkMode вы можете использовать значения OPAQUE или TRANSPARENT, определенные в файле windows.h.

Приложение может определить текущий режим фона, вызвав функцию GetBkMode :

int WINAPI GetBkMode(HDC hdc);

С помощью функций SetBkColor и GetBkColor вы можете, соответственно, установить и определить текущий цвет фона, который используется для закраски промежутков между штрихами и точками линий:

COLORREF WINAPI SetBkColor(HDC hdc, COLORREF clrref);

COLORREF WINAPI GetBkColor(HDC hdc);

**Выбор режима рисования**

Возвращаясь к обычной бумаге и карандашу, отметим, что в процессе рисования графит (или иной материал) переносится с острия карандаша на поверхность бумаги. Цвет полученной линии полностью соответствует цвету карандаша и не зависит от цвета бумаги. По умолчанию в контексте отображения выбран именно такой режим рисования , т. е. цвет рисуемой линии не зависит от цвета изображения, поверх которого рисуется линия.

Однако это не единственная возможность. При выборе соответствующего режима рисования цвет линии (на растровых устройствах вывода, таких, как экран монитора) может зависеть от цвета подложки, причем зависимость может быть достаточно сложная.

Для выбора режима рисования предназначена функция SetROP2 :

int WINAPI SetROP2(HDC hdc, int fnDrawMode);

Параметр hdc предназначен для указания контекста отображения, в котором необходимо установить новый режим рисования, определяемый параметром fnDrawMode.

Функция SetROP2 возвращает код предыдущего режима рисования.

Процесс рисования на экране монитора заключается в выполнении логической операции над цветами точек экрана и цветами изображения. Ниже в таблице мы привели возможные значения для параметра fnDrawMode. Для каждого режима рисования в этой таблице есть формула, с использованием которой вычисляется результат, и краткое описание режима рисования. В формулах цвет пера обозначается буквой P, цвет подложки - D.

|  |  |  |
| --- | --- | --- |
| Режим рисования | Формула | Цвет пиксела |
| R2\_COPYPEN | P | Соответствует (равен) цвету пера |
| R2\_BLACK | 0 | Черный |
| R2\_WHITE | 1 | Белый |
| R2\_NOP | D | Не меняется, т. е. перо ничего не рисует |
| R2\_NOT | ~D | Получается инвертированием цвета подложки, т. е. цвета пиксела до рисования |
| R2\_NOTCOPYPEN | ~P | Получается инвертированием цвета пера |
| R2\_MASKPEN | P&D | Комбинация компонент цветов, имеющихся как в цвете подложки, так и в цвете пера |
| R2\_NOTMASKPEN | ~(P&D) | Инверсия предыдущего значения |
| R2\_MERGEPEN | P|D | Комбинация компонент цветов подложки и пера |
| R2\_NOTMERGEPEN | ~(P|D) | Инверсия предыдущего значения |
| R2\_XORPEN | P^D | При определении цвета пиксела выполняется операция "ИСКЛЮЧАЮЩЕЕ ИЛИ" между компонентами цвета подложки и пера |
| R2\_NOTXORPEN | ~(P^D) | Инверсия предыдущего значения |
| R2\_MASKNOTPEN | ~P & D | Комбинация цвета подложки и инверсии цвета пера |
| R2\_MASKPENNOT | P & ~D | Комбинация двух цветов: инверсии цвета подложки и цвета пера |
| R2\_MERGENOTPEN | ~P | D | Комбинация компонент цветов подложки и инверсии цвета пера |
| R2\_MERGEPENNOT | P | ~D | Комбинация инверсии цвета подложки и цвета пера |

Если изображение и перо черно-белые, результат выполнения описанных выше операций (которые, кстати, называются растровыми операциями ) можно легко предсказать.

В режиме R2\_COPYPEN, который установлен в контексте отображения по умолчанию, цвет нарисованной линии будет такой же, как и цвет пера. Для режимов R2\_BLACK и R2\_WHITE цвет линии будет, соответственно, черный и белый. В режиме R2\_NOP вы не увидите нарисованную линию, так как цвет вдоль нее вообще не изменится. Более интересен режим R2\_NOT, при использовании которого на черном фоне будет нарисована белая линия, а на белом фоне - черная.

Для цветных изображений перечисленные выше формулы применяются по отдельности к каждой компоненте цвета (всего в Windows используется три компоненты цвета - красная, зеленая и голубая), поэтому для некоторых режимов рисования цвет линии предсказать достаточно трудно. Использование цветовых палитр, которые мы рассмотрим в третьей главе нашей книги, дополнительно усложняет эту задачу.

С помощью функции GetROP2 приложение может определить режим рисования, установленный для контекста отображения hdc:

int WINAPI GetROP2(HDC hdc);

**Рисование линий произвольного стиля**

Как мы уже говорили, вы не можете создать перо для рисования пунктирных, штрих-пунктирных или штриховых линий толщиной больше одного пиксела. Однако в некоторых случаях у вас может возникнуть необходимость в рисовании таких линий.

В программном интерфейсе GDI есть функция с именем LineDDA, которая позволяет рисовать любые линии (правда, основная работа по рисованию линий при этом будет возложена на программиста).

Функция LineDDA имеет следующий прототип:

void WINAPI LineDDA(

int nxStart, int nyStart, // начальная точка

int nxEnd, int nyEnd, // конечная точка

LINEDDAPROC lnddaprc, // адрес функции для рисования

LPARAM lParam); // дополнительные параметры

Первые четыре параметра этой функции определяют координаты начальной и конечной точки, между которыми надо нарисовать линию.

Через параметр lnddaprc передается указатель на функцию рисования, которая является функцией обратного вызова, определяемой программистом. Эта функция получает управление много раз, она вызывается для каждой точки рисуемой линии.

Для режима STRICT тип LINEDDAPROC определен в файле windows.h следующим образом:

typedef void (CALLBACK\* LINEDDAPROC)(int, int, LPARAM);

Последний параметр предназначен для передачи дополнительных данных в функцию рисования.

Приведем прототип функции рисования (для функции можно использовать любое имя):

void CALLBACK \_export

LineProc(int xPos, int yPos, LPARAM lParam);

Первые два параметра представляют собой координаты точки, для рисования которых вызвана функция. Последний параметр соответствует последнему параметру функции LineDDA и содержит передаваемое этой функции значение.

Пример использования функции LineDDA вы можете найти ниже в разделе "Приложение DASHLINE".

**Рисование замкнутых фигур**

Помимо линий, приложения Windows могут использовать функции GDI для рисования замкнутых закрашенных или незакрашенных фигур, таких как прямоугольники, эллипсы, многоугольники с прямыми и скругленными углами и т. д.

Для закрашивания внутренней области замкнутых фигур используется кисть, задаваемая как атрибут контекста отображения. Внешний контур фигуры обводится пером, которое также выбирается в контекст отображения. Учитываются и остальные атрибуты, установку которых мы рассмотрели для функций рисования линий, такие, как режим отображения, режим фона, код растровой операции.

Мы начнем изучение функций GDI, предназначенных для рисования замкнутых фигур, с функций рисования прямоугольников.

**Рисование прямоугольника**

Простейшая функция, с помощью которой можно нарисовать прямоугольник, называется Rectangle :

BOOL WINAPI Rectangle(

HDC hdc, // идентификатор контекста отображения

int nxTL, // координата x верхнего левого угла

int nyTL, // координата y верхнего левого угла

int nxBR, // координата x правого нижнего угла

int nyBR); // координата y правого нижнего угла

Функция Rectangle рисует прямоугольник для контекста отображения hdc, возвращая значение TRUE в случае успеха или FALSE при ошибке.

Назначение остальных параметров иллюстрируется рис. 2.17.

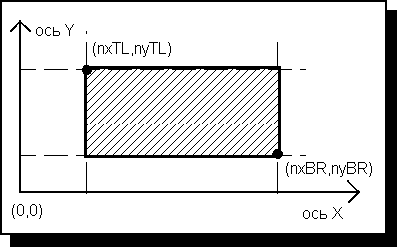


Рис. 2.17. Рисование прямоугольника

Как видно из этого рисунка, последние четыре параметра функции задают координаты верхнего левого и нижнего правого угла прямоугольника.

В зависимости от стиля пера граница фигуры может находится полностью внутри прямоугольника, заданного координатами (nxTL, nyTL), (nxBR,nyBR) или выходить за его пределы (см. рис. 2.14). Если выбрать стиль пера PS\_NULL, граница фигуры станет невидимой.

В зависимости от кисти, выбранной в контекст отображения, внутренность прямоугольника может быть закрашенной в тот или иной цвет, заштрихована одним из нескольких способов (как показано на рис. 2.16) или закрашена с помощью любого битового изображения размером 8х8 пикселов.

С помощью функции RoundRect можно нарисовать прямоугольник со скругленными углами (рис. 2.18).

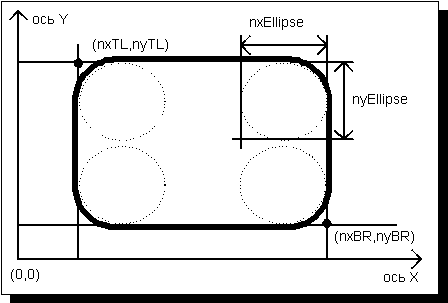


Рис. 2.18. Прямоугольник со скругленными углами

По сравнению с функцией Rectangle функция RoundRect имеет два дополнительных параметра nxEllipse и nyEllipse, определяющих форму и радиус закругления:

BOOL WINAPI RoundRect(

HDC hdc, // идентификатор контекста отображения

int nxTL, // координата x верхнего левого угла

int nyTL, // координата y верхнего левого угла

int nxBR, // координата x правого нижнего угла

int nyBR, // координата y правого нижнего угла

int nxEllipse, // ширина эллипса

int nyEllipse); // высота эллипса

Есть и другие функции, которые можно использовать для рисования прямоугольников.

Функция FillRect закрашивает прямоугольную область окна заданной кистью:

int WINAPI FillRect(

HDC hdc, // идентификатор контекста отображения

const RECT FAR\* lprc, // указатель на структуру RECT

HBRUSH hbrush); // идентификатор кисти для закрашивания

Параметр lprc должен указывать на структуру типа RECT, в которую следует записать координаты закрашиваемой прямоугольной области. Правая и нижняя граница указанной области не закрашивается.

Независимо от того, какая кисть выбрана в контекст отображения, функция FillRect будет использовать для закраски кисть, указанную параметром hbrush.

Учтите, что правильная работа функции FillRect гарантируется только в том случае, когда значение поля bottom структуры RECT больше значения поля top, а значение поля right больше значения поля left.

Для закрашивания границы прямоугольной области (т. е. для рисования прямоугольной рамки) можно использовать функцию FrameRect :

int WINAPI FrameRect(

HDC hdc, // идентификатор контекста отображения

const RECT FAR\* lprc, // указатель на структуру RECT

HBRUSH hbrush); // идентификатор кисти для закрашивания

Параметры этой функции аналогичны параметрам функции FillRect.

Ширина пера, используемого для рисования рамки, всегда равна одной логической единице. Структура RECT должна быть подготовлена таким же образом, что и для функции FillRect, т. е. значение поля bottom структуры RECT должно быть больше значения поля top, а значение поля right - больше значения поля left.

Значение, возвращаемое функциями FillRect и FrameRect не используется, приложения должны его игнорировать.

Используя функцию InvertRect , вы можете инвертировать содержимое прямоугольной области, заданной параметром lprc:

void WINAPI InvertRect(HDC hdc, const RECT FAR\* lprc);

Есть еще одна интересная функция, предназначенная для рисования прямоугольников. Она имеет имя DrawFocusRect :

void WINAPI DrawFocusRect(HDC hdc, const RECT FAR\* lprc);

Эта функция рисует прямоугольную рамку, предназначенную для выделения окна, имеющего фокус ввода.

Функция DrawFocusRect имеет три интересные особенности.

Во-первых, для рисования используется растровая операция "ИСКЛЮЧАЮЩЕЕ ИЛИ". Это приводит к тому, что для удаления нарисованной таким образом рамки ее достаточно нарисовать еще раз на том же месте.

Вторая особенность заключается в том, что для использования этой функции не нужно выбирать перо, рисующее пунктирную линию. Функция DrawFocusRect рисует пунктирную линию с нестандартным, очень близким расположением точек.

Третья особенность заключается в том, что перед использованием этой функции необходимо установить режим отображения MM\_TEXT.

Первые две особенности позволяют использовать ее для рисования рамки выделения произвольных участков изображения на экране монитора (при помощи мыши).

В заключение отметим, что в программном интерфейсе Windows нет функции для рисования квадрата и круга. Эти фигуры являются частными случаями, соответственно, прямоугольника и эллипса, поэтому для рисования, например, квадрата, вы должны использовать одну из только что описанных функций. Для сохранения пропорций проще всего использовать одну из метрических систем координат.

**Рисование эллипса**

Для рисования эллипса вы можете использовать функцию Ellipse :

BOOL WINAPI Ellipse(

HDC hdc, // идентификатор контекста отображения

int nxTL, // координата x верхнего левого угла

int nyTL, // координата y верхнего левого угла

int nxBR, // координата x правого нижнего угла

int nyBR); // координата y правого нижнего угла

Первый параметр этой функции указывает идентификатор контекста отображения, остальные - координаты верхнего левого и правого нижнего углов прямоугольника, в который должен быть вписан эллипс (рис. 2.19).

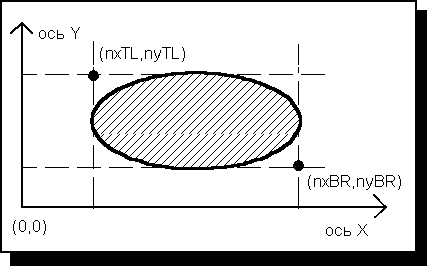


Рис. 2.19. Рисование эллипса

**Рисование сегмента эллипса**

Сегмент эллипса (рис. 2.20) можно нарисовать при помощи функции Chord :

BOOL WINAPI Chord(

HDC hdc, // идентификатор контекста отображения

int nxLeft, int nyTop, // верхий левый угол

int nxRight, int nyBottom, // правый нижний угол

int nxStart, int nyStart, // начало дуги

int nxEnd, int nyEnd); // конец дуги

Параметры этой функции аналогичны параметрам рассмотренной нами ранее функции Arc.

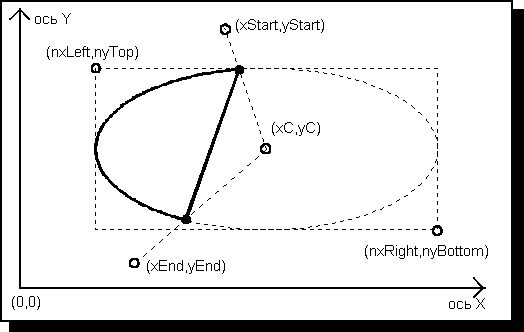


Рис. 2.20. Рисование сегмента эллипса

**Рисование сектора эллипса**

Для рисования сектора эллипса (рис. 2.21) следует использовать функцию Pie , аналогичную по своим параметрам функциям Arc и Chord:

BOOL WINAPI Pie(

HDC hdc, // идентификатор контекста отображения

int nxLeft, int nyTop, // верхний левый угол

int nxRight, int nyBottom, // правый нижний угол

int nxStart, int nyStart, // начало дуги

int nxEnd, int nyEnd); // конец дуги

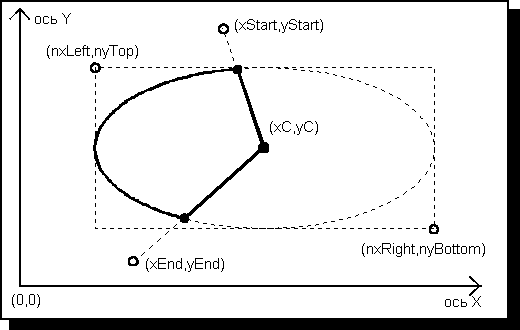


Рис. 2.21. Рисование сектора эллипса

**Рисование многоугольников**

Рисование многоугольников (рис. 2.22) выполняется функцией Polygon , аналогичной по своим параметрам функции Polyline, с помощью которой рисуются ломаные линии:

BOOL WINAPI Polygon(

HDC hdc, // идентификатор контекста отображения

const POINT FAR\* lppt,// указатель на массив структур POINT

int cPoints); // размер массива

Через параметр hdc передается идентификатор контекста отображения.

Параметр lppt указывает на массив структур POINT, в котором должны находится координаты вершин многоугольника. Параметр cPoints определяет размер этого массива.

Функция Polygon возвращает TRUE при нормальном завершении или FALSE при ошибке. Она не использует текущую позицию пера и не изменяет ее.

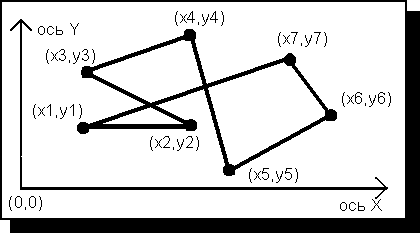


Рис. 2.22. Рисование многоугольника

В массиве структур POINT, определяющих вершины многоугольника, каждая вершина должна быть указана один раз. Функция Polygon автоматически замыкает ломаную линию, образующую многоугольник.

С помощью функции PolyPolygon можно нарисовать одновременно несколько многоугольников:

BOOL WINAPI PolyPolygon(

HDC hdc, // идентификатор контекста отображения

const POINT FAR\*lppt, // указатель на массив структур POINT

int FAR\* lpnPolyCounts, // адрес массива количества точек

// в многоугольниках

int cPolygons); // количество многоугольников

Первый параметр hdc, как обычно, задает контекст отображения.

Параметр cPolygons определяет количество многоугольников, которые нужно нарисовать.

Параметр lppt должен содержать указатель на массив структур типа POINT, содержащий координаты вершин всех многоугольников.

И, наконец, через параметр lpnPolyCounts передается указатель на массив целых чисел. Каждое число в этом массиве определяет количество точек в соответствующем многоугольнике.

В отличие от функции Polygon, функция PolyPolygon не замыкает автоматически ломаную линию, образующую многоугольник.

В контексте отображения имеется атрибут, влияющий на способ закрашивания для самопересекающихся многоугольников. По умолчанию выбран режим ALTERNATE , в котором эти области не закрашиваются (закрашиваются только те области, которые расположены между нечетными и четными сторонами многоугольника).

С помощью функции SetPolyFillMode вы можете изменить значение этого атрибута на WINDING . В этом режиме для того чтобы определить, надо ли закрашивать область многоугольника, учитывается направление, в котором был нарисован этот многоугольник. Каждая сторона многоугольника может быть нарисована в направлении либо по часовой стрелке, либо против часовой стрелки. Если воображаемая линия, нарисованная в направлении из внутренней области многоугольника в наружную, пересекает сегмент, нарисованный в направлении по часовой стрелке, содержимое некоторого внутреннего счетчика увеличивается на единицу. Если же эта линия пересекает сегмент, нарисованный против часовой стрелки, содержимое счетчика уменьшается на единицу. Область закрашивается только в том случае, если содержимое счетчика не равно нулю.

Немного позже вы сможете изучить этот алгоритм с помощью приложения LINER.

Приведем прототип функции SetPolyFillMode:

int WINAPI SetPolyFillMode(HDC hdc, int fnMode);

Параметр fnMode, определяющий режим закрашивания многоугольников, может принимать значения ALTERNATE или WINDING. Функция возвращает код старого режима закрашивания.

Вы можете определить используемый в данный момент режим закрашивания многоугольников с помощью функции GetPolyFillMode :

int WINAPI GetPolyFillMode(HDC hdc);

**Выбор кисти**

Для закрашивания внутренней области замкнутых фигур вы можете использовать встроенные кисти, или кисти, созданные вашим приложением. Последние необходимо удалять после использования.

**Использование встроенной кисти**

Для выбора одной из встроенной кисти вы можете воспользоваться макрокомандой GetStockBrush , определенной в файле windowsx.h:

#define GetStockBrush(i) ((HBRUSH)GetStockObject(i))

В качестве параметра для этой макрокоманды можно использовать следующие значения:

|  |  |
| --- | --- |
| Значение | Описание |
| BLACK\_BRUSH | Кисть черного цвета |
| WHITE\_BRUSH | Кисть белого цвета |
| GRAY\_BRUSH | Серая кисть |
| LTGRAY\_BRUSH | Светло-серая кисть |
| DKGRAY\_BRUSH | Темно-серая кисть |
| NULL\_BRUSH | Бесцветная кисть, которая ничего не закрашивает |
| HOLLOW\_BRUSH | Синоним для NULL\_BRUSH |

Как видно из только что приведенной таблицы, в Windows есть только монохромные встроенные кисти.

Макрокоманда GetStockBrush возвращает идентификатор встроенной кисти.

Прежде чем использовать полученную таким образом кисть, ее надо выбрать в контекст отображения (так же, как и перо). Для этого проще всего воспользоваться макрокомандой SelectBrush :

#define SelectBrush(hdc, hbr) \

((HBRUSH)SelectObject((hdc), (HGDIOBJ)(HBRUSH)(hbr)))

Макрокоманда SelectBrush возвращает идентификатор старой кисти, выбранной в контекст отображения раньше.

**Создание кисти**

Если вам нужна цветная кисть, ее следует создать с помощью функции CreateSolidBrush :

HBRUSH WINAPI CreateSolidBrush(COLORREF clrref);

В качестве параметра для этой функции необходимо указать цвет кисти. Для выбора цвета вы можете воспользоваться, например, макрокомандой RGB, позволяющей указать содержание отдельных цветовых компонент.

Windows может выбрать для кисти чистые или смешанные цвета, что зависит от текущего цветового разрешения. Подробности вы сможете узнать позже из главы, посвященной работе с цветами и цветовыми палитрами.

После использования созданной вами кисти ее следует удалить, не забыв перед этим выбрать в контекст отображения старую кисть. Для удаления кисти следует использовать макрокоманду DeleteBrush :

#define DeleteBrush(hbr) DeleteObject((HGDIOBJ)(HBRUSH)(hbr))

Приложение может заштриховать внутреннюю область замкнутой фигуры, создав одну из шести кистей штриховки функцией CreateHatchBrush :

HBRUSH WINAPI CreateHatchBrush(int fnStyle, COLORREF clrref);

С помощью параметра clrref вы можете определить цвет линий штриховки.

Параметр fnStyle задает стиль штриховки:

|  |  |
| --- | --- |
| Стиль штриховки | Внешний вид |
| HS\_BDIAGONAL | **http://frolov-lib.ru/books/bsp/v14/img00036.gif** |
| HS\_CROSS | **http://frolov-lib.ru/books/bsp/v14/img00037.gif** |
| HS\_DIAGCROSS | **http://frolov-lib.ru/books/bsp/v14/img00038.gif** |
| HS\_FDIAGONAL | **http://frolov-lib.ru/books/bsp/v14/img00039.gif** |
| HS\_HORIZONTAL | **http://frolov-lib.ru/books/bsp/v14/img00040.gif** |
| HS\_VERTICAL | **http://frolov-lib.ru/books/bsp/v14/img00041.gif** |

Вы можете использовать свой собственный стиль штриховки, создав кисть из битового изображения размером 8х8 пикселов (можно использовать только такой размер).

Если битовое изображение кисти определено в ресурсах приложения, его следует загрузить при помощи функции LoadBitmap . Эта функция возвратит идентификатор битового изображения. Затем для создания кисти этот идентификатор следует передать в качестве параметра функции CreatePatternBrush :

HBRUSH WINAPI CreatePatternBrush(HBITMAP hBitmap);

Забегая вперед, скажем, что битовые изображения делятся на те, которые хранятся в формате, зависящем от аппаратных особенностей устройства отображения, и на те, которые хранятся в аппаратно-независимом формате. Последние более универсальны, однако труднее в использовании. С помощью функции CreateDIBPatternBrush вы можете использовать для кисти битовое изображение в аппаратно-независимом формате:

HBRUSH WINAPI CreateDIBPatternBrush(

HGLOBAL hglbDibPacked, UINT fnColorSpec);

Первый параметр указывает на область глобальной памяти, в которой содержится аппаратно-независимое битовое изображение в упакованном формате. Второй параметр определяет содержимое таблицы цветов, используемое этим битовым изображением, и может принимать два значения: DIB\_PAL\_COLORS (таблица цветов содержит ссылки на цветовую палитру) DIB\_RGB\_COLORS (таблица цветов содержит отдельные компоненты цвета).

Более подробное обсуждение таких понятий, как аппаратно-независимые битовые изображения и таблица цветов мы отложим до главы, посвященной битовым изображениям.

**Закрашивание внутренней области окна**

Напомним, что кисть можно использовать еще и для закрашивания внутренней области окна . Для этого идентификатор кисти следует записать в поле hbrBackground структуры типа WNDCLASS перед регистрацией класса окна:

wc.hbrBackground = (HBRUSH)GetStockObject(LTGRAY\_BRUSH);

**Установка начальных координат кисти**

Начальные координаты кисти (brush origin ) - это атрибут контекста отображения. Он используются для определения координат точки внутри кисти, которая будет служить начальной при закраске внутренней области фигуры или окна. По умолчанию используются координаты (0,0), соответствующие верхнему левому углу кисти (в системе координат, выбранной в контекст отображения по умолчанию).

Если кисть используется для закраски внутренней области окна, верхний левый угол изображения кисти совмещается с верхним левым углом этой области. Затем изображение кисти многократно повторяется с шагом 8 пикселов.

При закраске фигур начальное расположение кисти привязывается не к фигуре, а по-прежнему к верхнему левому углу внутренней области окна. Поэтому при закраске, например, прямоугольника, верхний левый угол кисти может не совпадать с верхним левым углом прямоугольника.

Приложение может изменить начальные координаты кисти (сдвинуть кисть) при помощи функций SetBrushOrg и UnrealizeObject.

Прежде всего нужно вызвать функцию UnrealizeObject , передав ей в качестве параметра идентификатор сдвигаемой кисти (только если это не встроенная кисть):

BOOL WINAPI UnrealizeObject(HGDIOBJ hbrush);

В этом случае система сбросит координаты кисти после выбора ее в контекст отображения. После сброса надо установить новые значения координат кисти, вызвав функцию SetBrushOrg :

DWORD WINAPI SetBrushOrg(HDC hdc, int nx, int ny);

Параметры nx и ny определяют новые значения для начальных координат кисти пикселах (от 0 до 7).

В завершении следует снова выбрать кисть в контекст отображения при помощи макрокоманды SelectBrush.

|  |  |  |
| --- | --- | --- |
| [[Назад]](http://frolov-lib.ru/books/bsp/v14/ch1.htm) | [[Содеожание]](http://frolov-lib.ru/books/bsp/v14/index.html) | [[Дальше]](http://frolov-lib.ru/books/bsp/v14/ch3.htm) |