Лабораторная работа № 4

Изучение возможностей GDI для отображения информации в окнах Windows

Работа выполняется в системе программирования Borland Delphi 2.0 в среде Windows 95. Использование средств визуального программирования и классов VCL не допускается, то есть Delphi используется просто как 32-разрядный компилятор с языка Паскаль для операционной системы Windows.

Теоретические положения

Графический интерфейс устройства (GDI) — это подсистема Windows, отвечающая за отображение графической информации на видеодисплеях и принтерах. Наряду с двумя другими основными подсистемами (KERNEL — ядро, управление памятью, обеспечение многозадачности, файловый ввод/вывод, и USER — организация оконного пользовательского интерфейса) GDI является важнейшей составной частью Windows.

GDI поддерживает аппаратно-независимую графику. Можно рассматривать GDI как высокоуровневый интерфейс для аппаратных средств графики, как своего рода графический язык программирования. Схема прохождения команд управления между программой и аппаратурой отображения в Windows представлена на рис.1. При отображении информации программа обязательно должна пользоваться средствами GDI, попытки осуществить вывод напрямую в видеопамять, минуя GDI, будут пресекаться операционной системой, и даже если ее удастся "обмануть", никто не гарантирует, что такая программа будет работоспособной в будущих версиях Windows.

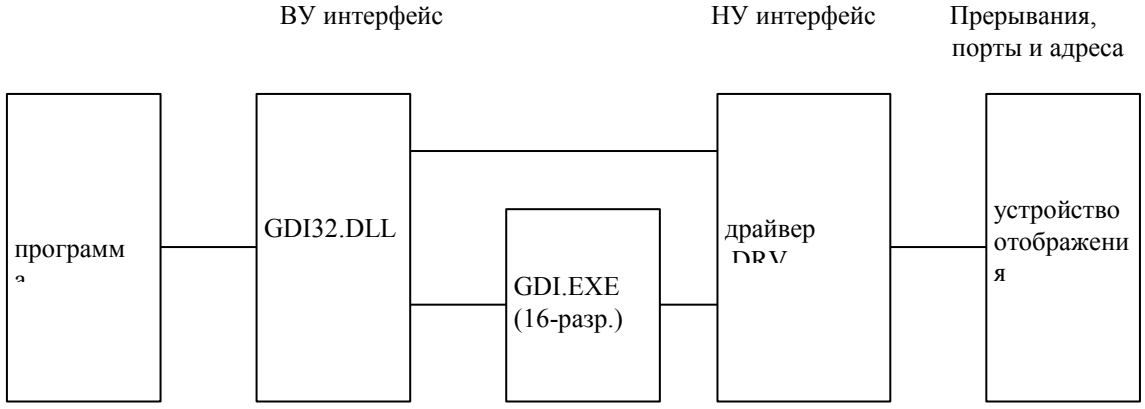

Рис.1 Прохождение команд между программой и устройством отображения

GDI состоит из нескольких сотен функций, описание которых можно получить, воспользовавшись справочной системой по Win32 API. Далее будут освещены лишь ключевые моменты, касающиеся основных вопросов применения функций GDI. Содержащиеся далее утверждения верны "в большинстве случаев", т.е. в некоторых ситуациях они могут не соответствовать действительности. Однако описание таких ситуаций потребовало бы во много раз большего места, чем приводимое описание, поэтому опускается.

Группы функций

Функции GDI можно разделить на следующие крупные группы:

- Функции получения и освобождения контекста устройства, такие как BeginPaint, EndPaint, GetDC, ReleaseDC и др. Понятие контекста устройства рассмотрено в теоретическом материале к ЛР2.

- Функции получения информации о контексте устройства, например GetTextMetrics.
- Функции рисования линий, эллипсов, закрашенных областей и текста.
- Функции управления атрибутами контекста устройства, такие как SetBkMode, SettextAlign, и соответствующие им функции "Get".
- Функции для работы с объектами GDI, такими как перо, кисть, шрифт.

Примитивы GDI

Типы графических объектов, выводимых на устройство вывода, могут быть разделены на несколько категорий, называемых "примитивами". К ним относятся:

- Отрезки прямых и кривые. GDI поддерживает рисование прямых (отрезков), прямоугольников, эллипсов и окружностей, дуг эллипсов, сплайнов Безье, а также ломаных линий, состоящих из множества отрезков. Линии рисуются с использованием объекта GDI "перо" (Pen).
- Закрашенные области. Области закрашиваются с использованием объекта GDI "кисть" (Brush).
- Битовые образы. Битовый образ — это двумерный набор битов, соответствующих пикселам устройства отображения. Для работы с битовыми образами используются соответствующие объекты GDI (Bitmap).
- Текст. GDI поддерживает как растровые шрифты, так и шрифты TrueType, являющиеся векторными и представляющие собой информацию о контурах символов, который закрашивается определенным цветом. GDI поддерживает вывод текста не только по горизонтали, но под любым углом к горизонтали вдоль некоторой прямой. Вывод текста осуществляется с использованием объекта GDI "шрифт" (Font).

Концепция использования объектов GDI

Для управления выводом графических примитивов в контексте устройства должен быть выбран тот или иной объект GDI (шрифт, перо, кисть). Если в программе необходимо изменить, например, начертание шрифта или стиль заливки областей, необходимо в общем случае *создать* новый объект GDI (шрифт или кисть соответственно), *выбрать* его в текущем контексте устройства, произвести требуемые операции рисования, затем *освободить* объект из контекста устройства (при этом обычно снова выбирается объект, являвшийся выбранным до того) и *удалить* (уничтожить) объект.

Режимы масштабирования и преобразования

Вывод в функциях GDI производится в логических координатах. По умолчанию координаты вывода и размеры задаются в пикселах, однако существуют и другие возможности. Могут использоваться доли дюйма, доли миллиметра, типографские пункты, а также любые удобные для программиста единицы.

Другие важные понятия

Регион — это сложная область произвольной формы, возможно, не являющаяся непрерывной. Обычно регион задают как комбинацию простых фигур (например — прямоугольников). Следует понимать, что регион — это не то, что *отображается* на экране, это просто фигура в смысле координат. В то же время, регион можно визуализировать, обведя его контур или выполнив его заливку цветом. Внутреннее представление региона внутри GDI — это набор координат горизонтальных линий сканирования. Обычно регионы используются для задания области отсечения.

Путь (path) — это набор отрезков и кривых. Пути могут быть преобразованы в регионы, а также использованы для рисования, закрашивания и отсечения.

Отсечение (clipping). Рисование может быть ограничено в пределах некоторой области окна, задаваемой регионом или путем. Операции рисования приводят к изменению изображения только в пределах области отсечения; если, например, линия выходит за границы области отсечения, то она прорисовывается только внутри области вплоть до ее границы.

Контекст устройства

Работа с контекстом устройства при обработке сообщения WM_PAINT при помощи функций BeginPaint и EndPaint описана в теоретическом материале к ЛР2.

При необходимости выполнить рисование в других местах программы, т.е. в теле обработчика сообщения, отличного от WM_PAINT, следует пользоваться следующими командами:

```
hdc:=GetDC(hwnd);
{операции рисования}
ReleaseDC(hdc);
```

или

```
hdc:=GetWindowDC(hwnd);
{операции рисования}
ReleaseDC(hdc);
```

Функция GetDC возвращает хэндл контекста устройства для рабочей области окна, функция GetWindowDC — для всего окна (включая заголовок, рамку и полосы прокрутки). Функция GetWindowDC используется редко и полезна при нестандартной обработке сообщения WM_NCPAINT, которое оконная процедура получает при необходимости прорисовки заголовка и рамки, т.е. нерабочей области окна. После использования контекст устройства обязательно должен быть освобожден.

Функция GetDC(0) вернет хэндл контекста устройства для всего экрана.

Для получения контекста произвольного устройства системы используется функция CreateDC (см. Справку по Win32 API); в этом случае контекст устройства должен не освобождаться, а удаляться при помощи функции DeleteDC. Если контекст устройства нужен не для операций рисования, а лишь для получения инйормации о характеристиках устройства, следует использовать функцию CreateIC с параметрами, аналогичными CreateDC. Эта функция создает так называемый информационный контекст устройства, который также впоследствии должен быть удален при помощи вызова DeleteDC.

При работе с битовыми образами (или при необходимости проводить рисование в буфере памяти, что одно и то же) используется контекст памяти, *совместимый* с реальным контекстом устройства вывода. Для работы с ним используются следующие функции:

```
hdcMem:=CreateCompatibleDC(hdc);
{операции рисования в памяти}
DeleteDC(hdcMem);
```

Получение информации из контекста устройства может быть осуществлено различными функциями GDI, название которых как правило начинается с "Get". Самой общеупотребительной из них является

function GetDeviceCaps(hdc:THandle; iIndex:integer):integer;

возвращающая значение параметра, задаваемого константой в iIndex. Перечень возможных параметров можно получить, обратившись к Справке.

Цвет

Функции GDI используют 24-битовую модель цвета TrueColor. Так как трехбайтовых целых типов нет, то цвет задается четырехбайтовым целым числом (longint в 16-разрядных приложениях, а также integer в 32-разрядных) в следующем формате (речь идет о 16-ричных позициях):

00 BB GG RR

Младший байт задает интенсивность красной составляющей, первый байт — зеленой, второй — синей, старший байт нулевой. Для получения такого RGB-цвета из компонент в компиляторах определен макрос RGB(r,g,b) (в Delphi — функция RGB(r,g,b:integer):integer).

Если видеоадаптер работает в режиме с меньшим количеством цветов, то Windows при выводе на экран либо заменяет 24-битовый цвет на ближайший цвет, который может быть отображен видеоадаптером (при рисовании линий, при выводе фона текста), либо (при закрашивании областей в режимах с 16 и 256 цветами) производит растрирование (dithering), т.е. заполняет область не чистым цветом, а набором точек разных цветов, которые визуально воспринимаются как область, имеющая близкий к заданному цвет.

Можно определить ближайший к желаемому чистый цвет, отображаемый устройством, при помощи функции GetNearestColor, например

rgbPureColor:=GetNearestColor(hdc, rgbColor);

Рисование "до, не включая"

Список самых общеупотребительных функций рисования линий, областей и текста приведен в описании ЛР3. Здесь будут рассмотрены лишь принципы их применения.

Прежде всего, чтобы избегать ошибок типа "потерянная единица", надо помнить, что функции GDI рисуют по правилу "от и до, не включая конец". Это означает, например, что функция LineTo(x,y) проводит линию из текущей позиции пера в указанную точку (x,y), при этом текущая позиция пера и все точки линии закрашиваются, а точка (x,y) остается незакрашенной. Это удобно при рисовании ломанных линий в режиме с растровой операцией, отличной от прямого копирования, когда цвет пера объединяется с цветом фона по какому-либо закону — при этом не будет разрывов в точке сочленения отрезков.

Функции, использующие прямоугольные области, получают в качестве параметров координаты левого верхнего и правого нижнего углов прямоугольника. Другим проявлением рассматриваемого принципа является то, что при этом левый верхний угол нарисованного прямоугольника будет иметь указанные координаты, а правый нижний угол нарисованной фигуры всегда будет расположен на единицу *выше и левее* точки, указанной в качестве левого нижнего угла. (Здесь речь идет именно физически о правом верхнем и левом нижнем углах фигуры, а не о парах чисел, переданных в функцию рисования в качестве координат соответствующих углов.)

Наконец, координаты в Windows 95 API хотя и представляются 32-разрядными целыми числами, не могут выходить за границы 16-разрядного знакового диапазона, т.е. от -32768 до 32767. Windows NT такого ограничения не имеет.

Эллипсы и эллиптические дуги

Windows поддерживает только рисование эллипсов с горизонтально-вертикальным расположением осей.

Для рисования эллипсов функции GDI используют непривычный на первый взгляд прием: в качестве параметров функции передается не центр и размеры полуосей эллипса, а координаты левого верхнего и правого нижнего углов прямоугольника, *описанного* вокруг желаемого эллипса.

При рисовании дуг, а также секторов и сегментов (функции Arc, Chord и Pie), эллипс, которому принадлежит дуга, также задается описанным прямоугольником. Кроме того задаются секущие лучи, как бы проводимые из центра эллипса через две точки с указанными

координатами. Эти точки не обязательно должны лежать на линии эллипса. Дуга будет проведена от точки пересечения эллипса с первым лучом до точки пересечения эллипса со вторым лучом; в случае рисования сегмента или сектора будут проведены также хорда или два радиуса в полученные точки пересечения, а получившаяся фигура будет заполнена при помощи текущей кисти.

Этот способ задания эллиптических кривых крайне неудобен при необходимости работы с дугами, заданными в привычной полярной системе координат "угол-радиус".

Использование объектов GDI

Для операций рисования в текущем контексте устройства должны быть *выбраны* некоторые перо, кисть и шрифт. По умолчанию выбраны черное перо толщиной 1 пиксел, рисующее непрерывную линию, белая кисть и шрифт "System". При необходимости изменить эти параметры, необходимо в общем случае создать новый объект GDI, выбрать его в контекст, произвести необходимые операции рисования, а затем освободить и удалить.

В ряде случаев можно обойтись небольшим количеством предопределенных в системе Windows перьев и кистей. Их хэндлы можно получить при помощи функции getStockObject с использованием ряда определенных в модуле WINDOWS.PAS идентификаторов-констант, например

hPen:=GetStockObject(WHITE_PEN); *{белое перо}*

Тогда использование пера в некотором контексте hdc может выглядеть примерно так:

```
oldPen:=SelectObject(hdc, GetStockObject(WHITE_PEN));
 {возвращает хэндл пера, которое было выбрано до того}
{операции рисования}
SelectObject(hdc, oldPen); {выбрать старое перо}
```

Однако в большинстве случаев предопределенных объектов GDI оказывается недостаточно, поэтому программист вынужден создавать свои перья, кисти и шрифты. Так как объекты GDI не являются составляющей частью контекста устройства, то хэндл контекста устройства не используется при их создании. Это означает, во-первых, что эти объекты могут создаваться и уничтожаться один раз, например, при запуске и завершении программы. Во-вторых, объекты могут использоваться (быть выбраны) сразу в нескольких контекстах устройств.

Но это, конечно, не означает, что нельзя порождать и уничтожать объекты GDI во время одного цикла рисования между BeginPaint и EndPaint.

Перья

Для создания пера используются функции:

function CreatePen(iPenStyle, iWidth, rgbColor: integer): THandle;

function CreatePenIndirect(**const** LogPen: TLogPen): THandle;

Обе они возвращают хэндл созданного пера, отличие второй от первой состоит в том, что в качестве параметра передается запись с теми же полями, что и параметры первой функции. iPenStyle определяет стиль рисуемой линии (непрерывная, штриховые, невидимая), iWidth — толщина пера в логических единицах (если указан 0, то всегда 1 пиксел), rgbColor — цвет пера. Подробнее см. Справку по Win32 API.

Тогда работа с зеленым пером будет выглядеть следующим образом:

```
hGreenPen:=CreatePen(PS_SOLID, 1, rgb(0,$FF,0) );
oldPen:=SelectObject(hdc, hGreenPen);
 {возвращает хэндл пера, которое было выбрано до того}
{операции рисования}
```

```
SelectObject(hdc, oldPen); {выбрать старое перо}
DeleteObject(hGreenPen);
```

или, что менее читаемо,

```
oldPen:=SelectObject(hdc, CreatePen(PS_SOLID, 1, rgb(0,$FF,0) ));
 {возвращает хэндл пера, которое было выбрано до того}
{операции рисования}
DeleteObject(SelectObject(hdc, oldPen)) ;
{выбрать старое перо и удалить освободившееся}
```

Объекты, удаляемые с помощью DeleteObject, не должны быть выбранными ни в одном контексте устройства, за этим должен следить программист.

Кисти

Для создания кистей служат функции:

```
function CreateSolidBrush(rgbColor: integer): THandle;
function CreateHatchBrush(iHatchStyle, rgbColor: integer): THandle;
function CreatePatternBrush(hBitmap: THandle): THandle;
function CreateBrushIndirect(const LogBrush: TLogBrush): THandle;
```

Первая создает кисть для закрашивания цветом, вторая — для закрашивания цветной штриховкой, третья — для закрашивания произвольным битовым шаблоном (размер которого, к сожалению, не может в Windows 95 превышать 8x8 пикселов). Последняя функция может быть использована вместо любой из предыдущих. Подробнее см. Справку по Win32 API.

Будучи созданной, кисть может быть выбрана в любой контекст устройства для задания способа закрашивания областей .

Шрифты

Небольшое количество хэндлов шрифтов может быть получено с помощью функции GetStockObject. Но в большинстве случаев программисту приходится создавать объекты-шрифты "вручную". Созданный шрифт должен быть выбран в контекст устройства для использования, а затем, после освобождения из контекста, удален при помощи стандартной функции DeleteObject.

Шрифты создаются функциями CreateFont или CreateFontIndirect, обе функции возвращают хэндл созданного объекта. Эти функции отличаются лишь тем, что в первом случае в функцию передается множество параметров, а во втором — одна большая запись. Рассмотрим последнюю из них.

```
function CreateFontIndirect(const LogFont: TLogFont): THandle;
```

Запись LogFont имеет следующую структуру:

```
TLogFont = packed record
   lfHeight: Longint;
   lfWidth: Longint;
   lfEscapement: Longint;
   lfOrientation: Longint;
   lfWeight: Longint;
   lfItalic: Byte;
   lfUnderline: Byte;
   lfStrikeOut: Byte;
   lfCharSet: Byte;
   lfOutPrecision: Byte;
   lfClipPrecision: Byte;
```

```
    lfQuality: Byte;
    lfPitchAndFamily: Byte;
    lfFaceName: array[0..31] of Char;
  end;
```

Значение полей этой записи таково:

lfHeight - Высота шрифта в логических координатах предполагаемого контекста устройства. Если здесь задано положительное значение, то это высота шрифта с учетом дополнительного межстрочного интервала, если отрицательное — то это минус высота символов шрифта без учета дополнительного межстрочного интервала. Для расчета высоты шрифта, соответствующей заданному значению в типографских пунктах, следует применять формулу

```
lfHeight := - PointSize*GetDeviceCaps(hDC, LOGPIXELSY) div 72;
```

lfWidth - Ширина шрифта в пунктах. Это некоторая абстрактная величина (символы в шрифте могут иметь различную ширину), показывающая, насколько сжат или растянут шрифт по горизонтали относительно своего нормального начертания. Нормальное начертание достигается, когда lfWidth=0 или lfWidth=lfHeight.

lfEscapement - Угол в десятых долях градуса, под которым выводится строка и все символы в ней относительно оси X.

lfOrientation - В Windows NT задает угол в десятых долях градуса, под которым каждый символ строки повернут относительно горизонтали. В Windows 95 не поддерживается и должен быть равен lfEscapement.

lfWeight - Жирность шрифта. 0 или 400 - обычный, 700 - полужирный. Диапазон значений — от 0 до 1000.

lfItalic, lfUnderline, lfStrikeOut - Ненулевое значение делает шрифт наклонным, подчеркнутым и зачеркнутым соответственно.

lfFaceName - Z-строка, содержащая имя шрифта, под которым он зарегистрирован в системе, например 'Times New Roman Cyr'#0.

Остальные параметры могут быть в большинстве случаев заданы нулевыми.

При выводе текста также широко используются следующие функции:

SetBkMode - устанавливает прозрачный или непрозрачный режим вывода.

SetBkColor - устанавливает цвет фона шрифта при непрозрачном выводе.

SetTextAlign - устанавливает способ трактовки координат при выводе текста.

SetTextColor - устанавливает цвет текста.

GetTextExtentPoint - возвращает длину и высоту заданной строки в логических координатах устройства.

Действие этих функций не исчерпывается текстовым выводом. Подробнее см. Справку по Win32 API.

Растровые операции

При рисовании линий и заливке областей цвет пера или кисти может не просто копироваться в закрашиваемые пикселы поверхности рисования, а объединяться с пикселами поверхности рисования побитно в соответствии с одним из 16 заданных режимов ROP2. Режим устанавливается функцией

```
function SetROP2(hdc, iDrawMode: integer): integer;
```

возвращающей предыдущий активный режим ROP2. Прочитать текущий режим можно с помощью функции

```
function GetROP2(hdc: integer): integer;
```

Наиболее употребительные растровые операции — это R2_COPYPEN (обычное рисование, устанавливается по умолчанию) и R2_XORPEN (исключающее или, при этом то, что

нарисовано, может быть удалено повторным рисованием того же самого в том же самом месте).

Операции с битовыми образами

Битовый образ — это цифровое растровое представление изображения. Изображение представляется как двумерный массив чисел, характеризующих цвет точек растра. Практически любой графический редактор позволяет сохранить или экспортировать изображение в формат .BMP, который собственно и является форматом аппаратно-независимого (device-independent) битового образа Windows (DIB). Описание формата DIB приведено в файле BMP.TXT. Также см. Справку по разделам BITMAPFILEHEADER, BITMAPINFOHEADER.

Битовые образы (bitmap) также являются объектами GDI и могут выбираться в контекст устройства. Однако логика работы с битовыми образами отличается от того, что можно делать с перьями, кистями и шрифтами.

Две основные задачи, для которых используются битовые образы в большинстве программ — это:

- показ изображения, хранящегося в файле или ресурсе (сканированной фотографии или изображения на кнопке панели инструментов);
- организация буфера в памяти для построения изображения в нем.

Если битовый образ хранится в ресурсе, то он загружается с созданием объекта bitmap при помощи вызова функции

function LoadBitmap(hInstance: THandle; lpBitmapName: PChar): THandle;

Такие объекты требуют уничтожения с помощью DeleteObject.

При необходимости загрузить изображение из файла возникает гораздо больше проблем. Их была призвана устранить функция LoadImage, появившаяся в API Windows 95, которая позволяет загрузить битовый образ, значок или курсор из ресурса или из файла. К сожалению, эта функция не поддерживает загрузку образа из файла в Windows NT, поэтому программы Windows 95, использующие эту возможность, окажутся нерабочими под NT, что было бы крайне неприятно.

Поэтому для визуализации изображения из файла лучше использовать подход, продемонстрированный в примере LAB4.PAS. В этом случае файл BMP целиком читается в область памяти, после чего с помощью функции StretchDIBits отрисовывается в контексте устройства окна. При этом объект GDI bitmap как таковой не используется.

Для визуализации объекта - битового образа необходимо проделать ряд дополнительных действий. Дело в том, что Windows API не предоставляет средств для рисования объектов битовых образов в контексте устройства — для этого обязательно должен быть задействован контекст памяти, создаваемый функцией CreateCompatibleDC. Упрощенный вариант этих действий представлен в следующей процедуре, визуализирующей битовый образ на контексте устройства. Полный вариант процедуры представлен в тексте примера LAB31.PAS.

```
procedure DrawBitmap(hdc, hBitmap: THandle; xStart,yStart: integer);
  var bm: TBitmap;
      hdcMem: THandle;
begin
  hdcMem:=CreateCompatibleDC(hdc);
  SelectObject(hdcMem, hBitmap);
  GetObject(hBitmap, sizeof(TBitmap), @bm);
  BitBlt(hdc, xStart, yStart, bm.bmWidth, bm.bmHeight,
         hdcMem,0,0,SRCCOPY);
  DeleteDC(hdcMem);
```

```
end;
```

Вначале создается контекст памяти и в него выбирается объект - битовый образ. Выбор битового образа отождествляет контекст памяти с битовым образом в том смысле, что любая операция рисования, проводимая в контексте памяти, приводит к немедленному изменению битового образа, а изменение содержимого битового образа можно рассматривать как рисование в контексте памяти (хотя это никак не отображается на экране).

Функция GetObject позволяет получить информацию о любом объекте GDI по его хэндлу. Нас интересует, в данном случае, ширина и высота объекта - битового образа.

Наконец, выполняется пересылка прямоугольной области из контекста памяти в контекст устройства. Так как в контексте памяти был выбран битовый образ, это приводит наконец к визуализации битового образа в контексте устройства HDC. При необходимости выполнить визуализацию с растяжением или сжатием, следовало бы вместо bitBlt использовать функцию StretchBlt. Подробнее об использованных функциях см. Справку по Win32 API.

> Если стоит задача просто отобразить файл .BMP, то для этого наиболее подходит функция StretchDIBits, рисующая прямоугольную область битового образа, содержащегося в памяти в формате BMP, на контексте устройства с возможным сжатием/растяжением. При этом, если образ масштабируется, то для многоцветных образов желательно установить функцией SetStretchBltMode режим сжатия ColorOnColor. Подробнее см. Справку по Win32 API.

Для организации рисования в буфере также используется похожая технология: создается контекст памяти, создается битовый образ нужного размера с той же организацией цвета, что и в контексте устройства, после чего битовый образ выбирается в контекст памяти. Затем в контексте памяти производятся требуемые операции рисования, и получившаяся картина переносится в контекст устройства при помощи BitBlt или StretchBlt, т.е.

```
hdcMem:=CreateCompatibleDC(hdc);
hBitmap:=CreateCompatibleBitmap(hdc,width,height);
selectObject(hdcMem, hBitmap);
{рисование на hdcMem}
BitBlt(hdc, .... ,hdcMem, ....);
DeleteDC(hdcMem);
DeleteObject(hBitmap);
```

В примере LAB4.PAS такая технология реализована в процедуре PaintInBuffer.

Битовые образы могут использоваться для создания кистей с требуемым рисунком заполнения. При этом используется только та часть битового образа, которая не превышает по размерам квадрата 8x8 пикселов. Кисть на основе битового образа может быть построена, например, следующим образом:

```
hBitmap:=LoadBitmap(hInstance,'PATTERN1'); // загрузить ресурс
hBrush:=CreatePatternBrush(hBitmap); // создать кисть на основе bitmap
DeleteObject(hBitmap); // bitmap больше не нужен
hdc:=getDC(hwnd);      // получить контекст устройства
selectObject(hdc,hBrush);  // выбрать кисть
rectangle(hdc,10,10,20,20); // прямоугольник будет заполнен рисунком кисти
releaseDC(hdc);   // освободить контекст устройства
DeleteObject(hBrush);  // удалить созданную кисть
```

Системы координат устройства

Большинство функций GDI работает в логических координатах устройства. Это означает, что в общем случае координаты исчисляются не в пикселах, а в произвольных единицах, при этом начало координат не обязательно находится в верхнем левом углу окна, оси не

обязательно направлены слева-направо и сверху-вниз, масштаб по осям также может быть различным.

То, как логические координаты будут преобразовываться в физические (пиксельные) координаты устройства, определяется четырьмя парами параметров, которые обозначим как xWinExt, xWinOrg, xViewExt, xViewOrg, yWinExt, yWinOrg, yViewExt и yViewOrg. Физический смысл этих параметров следующий:

- xViewOrg, yViewOrg - положение начала координат в физической (пиксельной) сетке относительно левого верхнего угла области рисования.
- xWinOrg, yWinOrg - положение точки начала координат, заданной (xViewOrg, yViewOrg), в системе логических координат.
- xViewExt / xWinExt - масштабный коэффициент по оси X для перевода логических координат в физические, отрицательный знак означает, что ось X направлена влево.
- yViewExt / yWinExt- масштабный коэффициент по оси Y для перевода логических координат в физические, отрицательный знак означает, что ось Y направлена вверх.

Если обозначить логические координаты как (xWin, yWin), то физические координаты в пикселах относительно левого верхнего угла области рисования вычисляются по формулам:

```
xView = (xWin - xWinOrg) * xViewExt / xWinExt + xViewOrg;
yView = (yWin - yWinOrg) * yViewExt / yWinExt + yViewOrg;
```

Соответственно,

```
xWin = (xView - xViewOrg) * xWinExt / xViewExt + xWinOrg;
yWin = (yView - yViewOrg) * yWinExt / yViewExt + yWinOrg;
```

Для выполнения этих преобразований служат функции

```
function DPtoLP(DC: HDC; var Points; Count: Integer): BOOL;
function LPtoDP(DC: HDC; var Points; Count: Integer): BOOL;
```

производящие преобразование координат из физических в логические и из логических в физические соответственно в контексте DC для Count точек типа TPoint, содержащихся в массиве Points.

В зависимости от режима отображения некоторые из этих параметров заданы жестко, некоторые — можно изменять. Для изменения перечисленных параметров служат функции:

```
function SetViewportExtEx(DC: HDC; XExt, YExt: Integer; Size: PPoint): BOOL;
function SetViewportOrgEx(DC: HDC; X, Y: Integer; Point: PPoint): BOOL;
function SetWindowExtEx(DC: HDC; XExt, YExt: Integer; Size: PPoint): BOOL;
function SetWindowOrgEx(DC: HDC; X, Y: Integer; Point: PPoint): BOOL;
```

Параметры — контекст устройства, значения по осям, а также указатель на структуру типа TPoint, в которую заносятся старые значения по осям. В качестве этого указателя может быть передан NIL. Данные для Viewport задаются в пикселах, для Window — в логических единицах в зависимости от режима отображения.

Выбор режима отображения координат осуществляется функцией

```
function SetMapMode(hdc: THandle; index: integer): integer;
```

Функция возвращает предыдущий установленный режим отображения. Возможные режимы перечислены в следующей таблице:

Режим	Единицы	Напр. X	Напр. Y	WinOrg	ViewOrg	WinExt	ViewExt
MM_TEXT	пикселы	вправо	вниз	(0,0) var	(0,0) var	(1,1) const	(1,1) const
MM_LOENGLISH	0.01"	вправо	вверх	(0,0) var	(0,0) var	(?,?) const	(?,-?) const
MM_LOMETRIC	0.1 мм	вправо	вверх	(0,0) var	(0,0) var	(?,?) const	(?,-?) const
MM_HIENGLISH	0.001"	вправо	вверх	(0,0) var	(0,0) var	(?,?) const	(?,-?) const

MM_TWIPS	1/20 pt	вправо	вверх	(0,0) var	(0,0) var	(?,?) const	(?,-?) const
MM_HIMETRIC	0.01 мм	вправо	вверх	(0,0) var	(0,0) var	(?,?) const	(?,-?) const
MM_ISOTROPIC	произв. dx = dy	произв.	произв.	(0,0) var	(0,0) var	var, kx=ky	var, kx=ky
MM_ANISOTROPIC	произв	произв.	произв.	(0,0) var	(0,0) var	var	var

1" = 2.54 см

1 pt = 1/72" = 0.353 мм

Режим MM_TEXT

Этот режим удобен для работы в пиксельных координатах устройства и выбирается по умолчанию при получении контекста устройства. Ось Y направлена вниз. Шаг логических координат совпадает с шагом физических и составляет 1 пиксел по каждой оси, начало координат по умолчанию расположено в левом верхнем углу области рисования.

WinExt=ViewExt=1 в этом случае означает лишь то, что масштабирование при пересчете координат не производится; эти параметры не связаны с физической шириной и высотой области рисования.

Можно изменять положение начала координат.

Метрические режимы

К метрическим относятся режимы MM_LOENGLISH, MM_HIENGLISH, MM_LOMETRIC, MM_HIMETRIC и MM_TWIPS. Метрическими их называют потому, что логические координаты по обоим осям соответствуют единицам измерения метрической, дюймовой или типографской системы мер.

Windows устанавливает значение WinExt и ViewExt исходя из имеющихся в системе сведений о разрешающей способности устройства отображения, причем yWinExt и yViewExt имеют противоположный знак. Ось Y таким образом оказывается направленной вверх. По умолчанию начало координат находится в левом верхнем углу области рисования.

Можно изменять положение начала координат.

Режим MM_ISOTROPIC

В этом режиме имеется возможность задать не только положение начала координат, но и шаг логической координатной сетки. Особенностью данного режима является то, что Windows принудительно обеспечивает в нем равный шаг по обоим осям координат. Типичная область применения этого режима — отрисовка изображений в виртуальной системе координат, которые должны изменять размеры вместе с изменением размеров окон, но должны сохранять пропорции по горизонтали и вертикали, т.е. не сплющиваться и не вытягиваться. Прямоугольники с равными длинами сторон в таком случае выглядят квадратами, эллипсы с равными полуосями — кругами.

Например, пусть требуется, чтобы изображение с виртуальным размером 512x512 пикселов, всегда помещалось по центру окна, не изменяя соотношения сторон, и всегда занимало максимально возможную площадь. Пусть также необходимо, чтобы координатная ось Y была направлена вверх. Для этого следует воспользоваться следующей последовательностью команд:

```
SetMapMode(hdc, MM_ISOTROPIC);
SetWindowExtEx(hdc, 512, 512, nil);
SetViewportExtEx(hdc, cxClient, -cyClient, nil); {минус -> Y вверх}
SetViewportOrgEx(hdc, cxClient div 2, cyClient div 2, nil);
{Далее рисовать, как будто размер области рисования 512x512}
```

В этом случае cxClient и cyClient — это ширина и высота рабочей области в пикселах. Получить эти величины можно, например, вызвав функцию

function GetClientRect(hWnd: HWND; **var** lpRect: TRect): BOOL;
тогда
cxClient:=lpRect.right; cyClient:=lpRect.bottom;

Режим MM_ANISOTROPIC

Этот режим дает полную свободу в управлении шагом координат по осям. Он, в отличие от MM_ISOTROPIC, подходит тогда, когда нужно, чтобы изображение занимало всю площадь окна независимо от его размеров, при этом сплющиваясь и растягиваясь. Другой вариант использования этого режима — эмуляция текстового терминала с использованием моноширинного шрифта. При этом задается грубая координатная сетка, шаг которой по вертикали равен высоте символа, а по горизонтали — его ширине. После этого для вывода текста с помощью TextOut можно использовать "текстовые" координаты "столбец-строка".

Это реализуется так:

```
SetMapMode(hdc, MM_ANISOTROPIC);
SetWindowExtEx(hdc, 1, 1, nil);
SetViewportExtEx(hdc, CharWidth, CharHeight, nil);
```

где вместо CharWidth и CharHeight должны быть подставлены ширина и высота символов.

Задание

Выполняется при самостоятельной подготовке:

1. Изучить теоретическую часть описания ЛР и материал соответствующих лекций.

2. По материалам Справки (Help) изучить описание следующих функций API и связанных с ними структур данных:

BeginPaint, Endpaint, GetDC, GetWindowDC, ReleaseDC, CreateCompatibleDC, DeleteDC, GetDeviceCaps, RGB, MoveToEx, LineTo, Ellipse, Arc, Pie, Chord, Rectangle, GetStockObject, SelectObject, CreatePen, CreatePenIndirect, DeleteObject, CreateSolidBrush, CreateHatchBrush, CreatePatternBrush, CreateBrushIndirect, CreateFontIndirect, SetROP2, SetBkMode, SetBkColor, TextOut, ExtTextOut, DrawText, SetTextAlign, SetTextColor, GetTextExtentPoint, LoadBitmap, LoadImage, StretchDIBits, BitBlt, StretchBlt, DPtoLP, LPtoDP, SetMapMode, SetViewportExrEx, SetWindowExtEx, SetViewportOrgEx, SetWindowOrgEx.

Выполняется в лаборатории:

1. Включить компьютер. Запустить систему Delphi 2.0.

2. Загрузить исходный текст примера LAB4.PAS, а также модуль WINDOWS.PAS, изучить логику работы с объектами GDI, битовыми образами и логическими координатами.

3. Откомпилировать и запустить пример. Изучить поведение созданного окна.

4. Написать и отладить программу по индивидуальному заданию (см. ниже). Продемонстрировать результаты работы преподавателю.

5. Завершить работу в Delphi. Оставить компьютер включенным.

Варианты индивидуальных заданий

Программа должна создавать окно, в котором фон заполнен рисунком из файла .BMP, а в прямоугольной рамке заданной конфигурации выведен текст. Файл .BMP может быть создан с помощью стандартного редактора Paint или с помощью image Editor из комплекта по-

ставки Delphi. Текст не должен "выезжать" за и на границу рамки. Возможны произвольные дополнения визуального оформления окна по инициативе студента.

№	Заполнение рисунком	Рамка	Текст
1	растянут	10x10 см.	Строки программы, 12pt, Arial
2	растянут	70% по ширине и высоте окна	Надпись по диагонали, 12pt, Arial
3	растянут	максимальный квадрат по центру окна	Строки программы, 14pt, Courier
4	мозаичное	10x10 см.	Надпись по диагонали, 14pt, Courier
5	мозаичное	70% по ширине и высоте окна	Строки программы, 12pt, Times
6	мозаичное	максимальный квадрат по центру окна	Надпись по диагонали, 12pt, Times
7	растянут	10x10 см.	Строки программы, 14pt, Courier
8	растянут	70% по ширине и высоте окна	Надпись по диагонали, 14pt, Courier
9	растянут	максимальный квадрат по центру окна	Строки программы, 12pt, Arial
10	мозаичное	10x10 см.	Надпись по диагонали, 12pt, Arial
11	мозаичное	70% по ширине и высоте окна	Строки программы, 14pt, Times
12	мозаичное	максимальный квадрат по центру окна	Надпись по диагонали, 14pt, Times