17. Основы графического вывода

Главным принципом организации графического вывода в системах Windows является принцип **независимости** разрабатываемых программ от используемых устройств вывода. В идеале, прикладной программе должно быть безразлично, какое устройство вывода она использует в настоящий момент времени. Реализация принципа независимости основывается на трех фундаментальных понятиях:

- использование на уровне ОС набора низкоуровневых **драйверов** графических устройств вывода;
- использование приложением набора высокоуровневых **системных вызовов** (API-функций), реализующих основные графические операции;
- использование приложением и ОС специальной структуры данных, в которой хранится информация об активном в данный момент устройстве вывода; эта важнейшая структура данных называется **контекстом устройства** (DC, Device Context).

Контекст устройства относится к системным ресурсам, поэтому любая программа, желающая выполнить графический вывод, должна делать три следующих шага:

- запрос контекста у операционной системы;
- использование полученного контекста для графического вывода;
- освобождение контекста после завершения вывода.

Для запроса и освобождения контекста существует 4 пары функций. Наиболее часто используются следующие 2 пары функций:

- пара GetDC/ReleaseDC позволяет получить и освободить контекст устройства при необходимости вывода во всей клиентской части окна;
- пара BeginPaint/EndPaint используется в случае перерисовки лишь части клиентской области окна.

Поскольку контекст устройства является системным ресурсом, его использование отслеживается системой с помощью специального

дескриптора. Поэтому перед запросом контекста надо объявить переменную-дескриптор системного типа HDC:

```
var   MyDC : HDC;
```

Значение для этой переменной устанавливается в результате отработки запроса на получение контекста функцией GetDC или BeginPaint:

```
MyDC := GetDC (дескриптор_окна);
```

Дескриптор контекста используется как входной параметр в большинстве функций графического вывода. При освобождении контекста соответствующий дескриптор задается в качестве параметра вызовов ReleaseDC или EndPaint.

После получения контекста графический вывод реализуется с помощью какого-либо инструмента. Основными инструментами вывода являются:

- шрифт для вывода текста;
- перо (pen) для рисования линий;
- кисть (brush) как инструмент заполнения замкнутых областей.

В качестве примера рассмотрим рисование с помощью основных графических примитивов.

После получения контекста устройства приложение может с помощью вызова соответствующих API-функций выполнить прорисовку необходимого изображения. Один из наиболее популярных способов построения изображений состоит в использовании элементарных графических фигур, так называемых графических примитивов. При этом основным инструментом рисования является **перо** (Pen), а для заполнения замкнутых областей используется инструмент **кисть** (Brush). В системе существует несколько стандартных перьев и кистей, но при необходимости легко можно создать собственный инструмент рисования с нужными свойствами.

Установленными по умолчанию стандартными инструментами рисования являются перо черного цвета толщиной в 1 пиксель и белая кисть. Для использования таких инструментов после получения контекста не требуются никакие дополнительные вызовы.

Для изменения стандартных инструментов рисования можно использовать системную функцию GetStockObject, единственный параметр которой определяет новый стандартный инструмент. Набор этих инструментов очень небольшой и задается с помощью системных констант Black_Pen (черное перо), White_Pen (белое перо), Null_Pen (пустое перо). В последних версиях Windows добавлено еще цветное перо (константа DC_Pen). Для кистей можно использовать константы White_Brush и Black_Brush.

Если стандартного набора инструментов недостаточно, то можно создать собственное перо и собственную кисть, причем любое их количество, но в каждый момент времени рисование выполняется только одним конкретным инструментом. Создание и использование новых инструментов рисования надо выполнять в соответствии со следующей общей схемой (необходимые для этого функции рассматриваются чуть ниже):

- создать новый инструмент;
- связать созданный инструмент с контекстом устройства; данная связь позволяет заменить в контексте устройства существовавший до этого инструмент новым инструментом; при этом рекомендуется запомнить во вспомогательной переменной заменяемый инструмент для последующего его восстановления;
- использовать инструмент для рисования;
- восстановить в контексте старый инструмент рисования;
- уничтожить ненужный инструмент рисования.

Рассмотрим реализацию этой схемы для инструмента-пера.

Создание пера выполняется функцией CreatePen, принимающей 3 параметра:

- тип пера, определяемый с помощью системных констант, из которых чаще всего используется константа ps_Solid (сплошное перо);
- толщина пера в пикселах (только для сплошных перьев);

- цвет пера (системные цветовые константы или стандартная функция RGB).

Вызов CreatePen возвращает дескриптор пера – переменную системного типа HPen. Если предполагается использовать несколько перьев, то надо объявить соответствующее число дескрипторных переменных или даже массив таких переменных.

Связь нового инструмента с контекстом выполняется с помощью функции SelectObject, имеющей два параметра - дескриптор контекста и дескриптор нового инструмента:

OldPen := SelectObject (MyDC, NewPen);

Восстановление старого инструмента также выполняется с помощью вызова SelectObject, но уже не в функциональной, а в процедурной форме:

SelectObject (MyDC, OldPen);

Для уничтожения ненужного инструмента используется вызов DeleteObject с единственным параметром - дескриптором инструмента.

Аналогично, для создания новой кисти можно использовать вызовы CreateSolidBrush (сплошная кисть) и CreatePatternBrush (создание кисти по заранее заданному образцу). Каждая кисть описывается своим дескриптором – переменной типа HBrush.

Теперь кратко рассмотрим основные графические примитивы, применяемые для графического вывода.

Для рисования отрезка используется функция LineTo, которая проводит отрезок от начальной точки, определяемой с помощью специального невидимого указателя текущей позиции (УТП), до конечной точки, координаты которой задаются как параметры вызова. При запуске приложения начальное положение УТП устанавливается в (0,0) относительно левого верхнего угла клиентской области окна. Для перемещения УТП без рисования существует функция MoveToEx. Отрезок, как и все остальные примитивы, рисуется с помощью активного в данный момент пера.

Для прорисовки прямоугольников используется вызов Rectangle, параметрами которого являются дескриптор контекста и координаты левого верхнего и правого нижнего углов. С помощью специальной функции можно заполнить прямоугольник активной кистью.

Для прорисовки эллипсов и окружностей можно использовать функцию Ellipse, параметры которой полностью совпадают с параметрами вызова Rectangle, т.е. эллипс задается с помощью охватывающего прямоугольника.

Прорисовка дуги выполняется функцией Arc. Кроме перечисленных, существуют функции для прорисовки ломаных линий и замкнутых многоугольников.

Кроме примитивов, графические изображения можно получить с помощью растровых изображений и с помощью так называемых метафайлов.

Корректная прорисовка изображения требует соблюдения определенных правил. Предположим, что приложение в клиентской области окна вывело некоторое изображение по рассмотренной ранее схеме (запрос контекста, создание инструментов, вывод изображения, уничтожение инструментов, освобождение контекста). При перемещении окна по экрану система сама отвечает за перерисовку изображения в новом месте. Но подобная перерисовка не выполняется при других манипуляциях с окном (изменение размеров, перекрытие другим окном и т.д.). Система не отвечает за перерисовку графического изображения, автоматически перерисовываются только все стандартные управляющие элементы. Перерисовка графического изображения возлагается не на систему, а на приложение.

При этом система оказывает определенную помощь приложению, сообщая ему о том, что изображение в его окне испорчено и требует обновления. Операционная система следит за взаимным расположением окон на экране. В тех случаях, когда какое-либо окно должно быть перерисовано, система генерирует специальное сообщение wm_Paint. Это сообщение

должно обрабатываться оконной функцией приложения с выполнением необходимого графического вывода. Отсюда следует, что весь графический вывод должен быть собран внутри обработчика сообщения wm_Paint, выполняющего следующие основные действия:

- запрос контекста устройства с помощью вызова BeginPaint (а не GetDC !!!);
- создание всех необходимых инструментов, активизация нужных инструментов и использование их для рисования так, как было описано выше;
- уничтожение инструментов;
- освобождение контекста с помощью функции EndPaint.

Важное отличие от ранее рассмотренной схемы состоит в том, что вместо пары GetDC/ReleaseDC используется пара функций BeginPaint и EndPaint. Здесь функция BeginPaint имеет два входных параметра: дескриптор контекста и переменная специального системного типа **PaintStruct,** которая предварительно должна быть объявлена, хотя явно она обычно нигде не используется.

Довольно часто возникает необходимость программной генерации сообщения wm_Paint для принудительной перерисовки изображения в окне. Необходимость в принудительной перерисовке возникает тогда, когда приложение хочет динамически изменить уже существующее изображение, т.е. добавить или удалить какой-нибудь примитив, изменить цвет текста или примитива, изменить стиль заполнения фона какой-либо фигуры и т.д. Чаще всего для этих целей используется системный вызов InvalidateRect. Данная функция позволяет перерисовывать только часть окна, размеры которой можно получить с помощью специальной функции, хотя практически всегда выполняется полная перерисовка клиентской части окна. Вызов InvalidateRect должен включаться в обработчик какого-либо события (меню, таймер и т.д.), при этом в обработчике сначала должны выполняться какие-либо изменения в структуре или параметрах изображения.