Тема 4.4 Графические возможности системы программирования

В Windows на окне можно рисовать средствами GDI, консольное окно не исключение. Естественно такое решение будет непереносимым. Пример:

```
#include <windows.h>
#include <iostream>
#include <cmath>
int main(){
   HWND hwnd = GetConsoleWindow();
   HDC hdc = GetDC(hwnd);

   int x = 0;
   for (float i = 0; i < 3.14 * 10; i += 0.05)
   {
      SetPixel(hdc, x, 50 + 25 * cos(i), RGB(255, 255, 255));
      x += 1;
   }

   ReleaseDC(hwnd, hdc);
   std::cin.ignore();
   return 0;
}
```

Windows GDI

Интерфейс графических устройств Microsoft Windows (GDI) позволяет приложениям использовать графику и отформатированный текст как на видео-дисплее, так и на принтере. Приложения на основе Windows не обращаются к графическому оборудованию напрямую. Вместо этого GDI взаимодействует с драйверами устройств от имени приложений.

GDI+

Windows GDI+ — это API на основе классов для программистов C/C++. Это позволяет приложениям использовать графику и форматированный текст как на видео-дисплее, так и на принтере. Приложения на основе API-интерфейса Microsoft Win32 не обращаются к графическому оборудованию напрямую. Вместо этого GDI+ взаимодействует с драйверами устройств от имени приложений. GDI+ также поддерживается Microsoft Win64.

Растровое изображение — это графический объект, используемый для создания, управления (масштабирования, прокрутки, поворота и рисования) и хранения изображений в виде файлов на диске. В этом обзоре описываются классы растровых изображений и операции с растровыми рисунками.

Растровое изображение — это один из объектов GDI, которые можно выбрать в контексте устройства (DC).

Контексты устройств — это структуры, определяющие набор графических объектов и связанные с ними атрибуты, а также графические режимы, влияющие на выходные данные. В приведенной ниже таблице описаны объекты GDI, которые можно выбрать в контексте устройства.

Графический объект	Описание
Растровые изображения	Создает, управляет (масштабирование, прокрутка, поворот и рисование) и сохраняет изображения в виде файлов на диске.
Кисти	Рисует внутреннюю часть многоугольников, многоточия и контуров.
Шрифты	Рисует текст на видеотрансляции и других устройствах вывода.
Логическая палитра	Цветовая палитра, созданная приложением и связанная с заданным контекстом устройства.
Пути	Одна или несколько фигур (или фигур), которые заполнены и (или) контуром.
Перья	Графический инструмент, который приложение использует для рисования линий и кривых.
Регионы	Прямоугольник, многоугольник или эллипс (или сочетание двух или более этих фигур), которые можно заполнять, красить, инвертировать, обрамлять и использовать для проверки попадания (проверка расположения курсора).

С точки зрения разработчика растровое изображение состоит из коллекции структур, которые указывают или содержат следующие элементы:

- ✓ Заголовок, описывающий разрешение устройства, на котором был создан прямоугольник пикселей, размеры прямоугольника, размер массива битов и т. д.
- ✓ Логическая палитра.
- ✓ Массив битов, определяющий связь между пикселями в растровом изображении и записями в логической палитре.

Размер растрового изображения связан с типом изображения, содержащегося в нем. Точечные изображения могут быть монохромными или цветными. На изображении каждый пиксель соответствует одному или нескольким битам в растровом изображении. Монохромные изображения имеют соотношение 1 бит на пиксель (bpp). Цветообразующее изображение является более сложным. Число цветов, которые могут отображаться на растровом рисунке, равно двум, повышаемым до числа битов на пиксель. Таким образом, для растрового изображения с 256 цветом требуется 8 бит/с (2^8 = 256).

Панель управления приложения — это примеры приложений, использующих растровые изображения. При выборе фона (или обои) для рабочего стола фактически выбирается растровое изображение, которое система использует для рисования фона рабочего стола. Система создает выбранный фоновый узор, многократно рисуя шаблон 32 на 32 пикселя на рабочем столе.

Кисть — это графический инструмент, который приложения используют для рисования внутренней части многоугольников, эллипсов и контуров. Приложения для рисования используют кисти для рисования фигур; текстовые приложения используют кисти для рисования правил; Приложения автоматизированного проектирования (САПР) используют кисти для рисования интерьеров представлений сечения; Приложения и для электронных таблиц используют кисти для рисования разделов круговых диаграмм и линейчатых диаграмм.

Существует два типа кистей: ***логические и физические.***

Логическая кисть — это описание идеального растрового изображения, которое приложение использует для рисования фигур.

Физическая кисть — это фактическое растровое изображение, создаваемое драйвером устройства на основе определения логической кисти приложения.

Когда приложение вызывает одну из функций, создающих кисть, оно получает дескриптор, идентифицирующий логическую кисть. Когда приложение передает этот

дескриптор функции SelectObject , драйвер устройства для соответствующего дисплея или принтера создает физическую кисть.

Приложения используют координатные пространства и преобразования для масштабирования, поворота, преобразования, сдвига и отражения выходных данных графики.

Координатное пространство — это плоское пространство, которое находит двумерные объекты с помощью двух опорных осей, перпендикулярных друг другу. Существует четыре координатных пространства: мир, страница, устройство и физическое устройство (клиентская область, рабочий стол или страница бумаги принтера).

Преобразование — это алгоритм, который изменяет (преобразует) размер, ориентацию и форму объектов. Преобразования также переносят графический объект из одного пространства координат в другое. В конечном итоге объект отображается на физическом устройстве, которое обычно является экраном или принтером.

Заполненные фигуры — это геометрические формы, которые обрисовываются с помощью текущего пера и заполняются с помощью текущей кисти. Существует пять заполненных фигур:

- ✓ Ellipse
- ✓ Хорда
- ✓ Pie
- ✓ Многоугольник
- ✓ Прямоугольник

Шрифты используются для рисования текста на видеотрансляции и других устройствах вывода. Функции шрифта и текста позволяют устанавливать, выбирать и запрашивать различные шрифты.

Линии и кривые используются для рисования выходных данных графики на растровых устройствах.

Линия — это набор выделенных пикселей на растровом дисплее (или набор точек на печатной странице), определяемый двумя точками: начальной и конечной точкой.

Обычная кривая — это набор выделенных пикселей на растровом дисплее (или точки на печатной странице), который определяет периметр (или часть периметра) конического сечения.

Нерегулярная кривая — это набор пикселей, определяющий кривую, которая не соответствует периметру конического сечения.

Перо — это графический инструмент, который приложение может использовать для рисования линий и кривых. В приложениях для рисования используются ручки для рисования линий, прямых и кривых. Приложения с автоматизированным дизайном (САПР) используют ручки для рисования видимых линий, скрытых линий, линий разделов, центральных линий и т. д. Word приложения для обработки и публикации классических приложений используют ручки для рисования границ и правил. Приложения для электронных таблиц используют ручки для обозначения тенденций в графах, а также для контура линейчатых и круговых диаграмм.

Opengl

В качестве программного интерфейса для графического оборудования OpenGL отрисовывает многомерные объекты в framebuffer. Microsoft реализация OpenGL для операционной системы Windows — это стандартное графическое программное обеспечение, с помощью которого программисты могут создавать высококачественные анимированные трехмерные цветные изображения.

OpenGL предназначен для обеспечения совместимости между оборудованием и операционными системами. Эта архитектура упрощает перенос программ OpenGL из одной системы в другую. Хотя каждая операционная система имеет уникальные требования, код OpenGL во многих программах можно использовать как есть.

Сведения о рисовании

Почти все приложения используют экран для отображения данных, с помощью которые они управляют. Приложение рисует изображения, рисует рисунки и записывает текст, чтобы пользователь смог просматривать данные по мере их создания, редактирования и печати. Microsoft Windows предоставляет широкие возможности для рисования и рисования, но из-за особенностей многозадачных операционных систем приложения должны взаимодействовать друг с другом при доступе к экрану.

Чтобы обеспечить бесперебойную и совместную работу всех приложений, система управляет всеми выходными данными на экране. Приложения используют окна в качестве основного устройства вывода, а не самого экрана. Система предоставляет контексты устройств отображения, которые однозначно соответствуют окнам. Приложения используют контексты устройства отображения для направления выходных данных в указанные окна. Рисование в окне (направление к нему выходных данных) не позволяет приложению вмешиваться в выходные данные других приложений и позволяет приложениям сосуществовать друг с другом, используя при этом все преимущества графических возможностей системы.

Приложение выполняет рисование в окне в разное время: при первом создании окна, при изменении размера окна, при перемещении окна из-за другого окна, при минимизации или максимальном увеличении окна, при отображении данных из открытого файла, при прокрутке, изменении или выборе части отображаемых данных.

Система управляет такими действиями, как перемещение и изменение размера окна. Если действие влияет на содержимое окна, система помечает затронутую часть окна как готовую к обновлению и при следующей возможности отправляет **WM_PAINT** сообщение в процедуру окна. Сообщение является сигналом для приложения, чтобы определить, что необходимо обновить, и выполнить необходимый рисунок.

Некоторые действия управляются приложением, например отображение открытых файлов и выбор отображаемых данных. Для этих действий приложение может пометить для обновления части окна, затронутой действием, что приводит к отправке WM_PAINT сообщения при следующей возможности. Если действие требует немедленной обратной связи, приложение может рисовать во время выполнения действия, не дожидаясь WM_PAINT. Например, типичное приложение выделяет область, выбранную пользователем, а не ожидает следующего сообщения WM_PAINT для обновления области.

Во всех случаях приложение может рисовать в окне сразу после его создания. Для рисования в окне приложение сначала должно получить дескриптор контекста устройства отображения для окна. В идеале приложение выполняет большую часть операций рисования во время обработки WM_PAINT сообщений. В этом случае приложение получает контекст устройства отображения, вызывая функцию BeginPaint . Если приложение выполняет рисование в любое другое время, например из WinMain или во время обработки сообщений клавиатуры или мыши, оно вызывает функцию GetDC или GetDCEx для получения контроллера домена дисплея.

Как правило, приложение рисует в окне в ответ на WM_PAINT сообщение. Система отправляет это сообщение в процедуру окна, когда изменения в окне изменили содержимое клиентской области. Система отправляет сообщение только в том случае, если в очереди сообщений приложения нет других сообщений.

Получив сообщение WM_PAINT , приложение может вызвать BeginPaint , чтобы получить контекст устройства отображения для клиентской области и использовать его в вызовах функций GDI для выполнения любых операций рисования, необходимых для обновления клиентской области. После завершения операций рисования приложение вызывает функцию EndPaint , чтобы освободить контекст устройства отображения.

Прежде чем BeginPaint возвращает контекст устройства отображения, система подготавливает контекст устройства для указанного окна. Сначала она задает область обрезки для контекста устройства, равную пересечению части окна, которая требует обновления, и части, видимой пользователю. Перерисовываются только те части окна, которые были изменены. Попытки рисования за пределами этой области обрезаются и не отображаются на экране.

Система также может отправлять WM_NCPAINT и WM_ERASEBKGND сообщения в процедуру окна до возврата BeginPaint . Эти сообщения направляют приложение на рисование неклиентской области и фона окна. Неклиентская область — это часть окна, которая находится за пределами клиентской области. Область включает такие функции, как заголовок окна, меню окна (также известное как системное меню) и полосы прокрутки. Большинство приложений используют оконную функцию по умолчанию DefWindowProc для рисования этой области и, следовательно, передачи сообщения WM_NCPAINT в эту функцию. Фон окна — это цвет или узор, которым окно заполняется перед началом других операций рисования. Фон охватывает все изображения, ранее размещенные в окне или на экране под окном. Если окно принадлежит классу окна с фоновой кистью класса, функция DefWindowProc автоматически рисует фон окна.

BeginPaint заполняет структуру PAINTSTRUCT такими сведениями, как размеры обновляемой части окна и флагом, указывающим, был ли нарисован фон окна. Приложение может использовать эти сведения для оптимизации рисования. Например, он может использовать измерения области обновления, заданные элементом rcPaint , чтобы ограничить рисование только теми частями окна, которые требуют обновления. Если приложение имеет очень простые выходные данные, оно может игнорировать область обновления и рисовать во всем окне, полагаясь на то, что система отменит (обрезать) все ненужные выходные данные. Так как система обрезает рисунок, который выходит за пределы области обрезки, отображается только рисунок, который находится в области обновления.

BeginPaint задает для области обновления окна значение NULL. Это очищает регион, предотвращая создание последующих WM_PAINT сообщений. Если приложение обрабатывает сообщение WM_PAINT , но не вызывает BeginPaint или иным образом не очищает регион обновления, приложение продолжает получать WM_PAINT сообщения, пока регион не пуст. Во всех случаях приложение должно очистить регион обновления, прежде чем возвращаться из сообщения WM_PAINT .

После завершения рисования приложение должно вызвать EndPaint. Для большинства окон EndPaint выпускает контекст устройства отображения, делая его доступным для других окон. EndPaint также отображает курсор, если он был ранее скрыт с помощью BeginPaint. BeginPaint скрывает курсор, чтобы предотвратить повреждение операции рисования.

Хотя приложения выполняют большинство операций рисования во время обработки сообщения WM_PAINT , иногда приложение эффективнее рисовать непосредственно в окне, не полагаясь на WM_PAINT сообщение. Это может быть полезно, когда пользователю требуется немедленная обратная связь, например при выборе текста и перетаскивании или изменении размера объекта. В таких случаях приложение обычно рисует при обработке сообщений с клавиатуры или мыши.

Для рисования в окне без использования сообщения WM_PAINT приложение использует функцию GetDC или GetDCEx для получения контекста отображаемого устройства для окна. С помощью контекста устройства отображения приложение может рисовать в окне и не вторгнуться в другие окна. После завершения рисования приложение вызывает функцию ReleaseDC , чтобы освободить контекст отображаемого устройства для использования другими приложениями.

При рисовании без использования сообщения WM_PAINT приложение обычно не делает окно недействительным. Вместо этого он рисует таким образом, что может легко восстановить окно и удалить рисунок. Например, когда пользователь выбирает текст или объект, приложение обычно рисует выделение, инвертируя все, что уже есть в окне. Приложение может удалить выделенный фрагмент и восстановить исходное содержимое окна, просто снова перевернуто.

Приложение отвечает за тщательное управление любыми изменениями, которые оно вносит в окно. В частности, если приложение рисует выделение и возникает промежуточное WM_PAINT сообщение, приложение должно убедиться, что любой рисунок, выполненный во время сообщения, не повреждает выделение. Чтобы избежать этого, многие приложения удаляют выделение, выполняют обычные операции рисования, а затем восстанавливают выделение после завершения рисования.

Система координат для окна основана на системе координат устройства отображения. Базовой единицей измерения является единица измерения устройства (обычно пиксель). Точки на экране описываются парами координат x и y. Координаты по оси X увеличиваются вправо; Координаты y увеличиваются сверху вниз. Источник (0,0) для системы зависит от типа используемых координат.

Система и приложения определяют положение окна на экране в координатах экрана. Для экранных координат точкой отсчета является верхний левый угол экрана. Полное положение окна часто описывается структурой RECT , содержащей экранные координаты двух точек, определяющих левый верхний и нижний правый углы окна.

Система и приложения определяют положение точек в окне с помощью клиентских координат. В этом случае источником является левый верхний угол окна или клиентской области. Координаты клиента гарантируют, что приложение может использовать согласованные значения координат при рисовании в окне независимо от положения окна на экране.

Размеры клиентской области также описываются структурой RECT , содержащей координаты клиента для этой области. Во всех случаях верхняя левая координата прямоугольника включается в область окна или клиента, а нижняя правая координата исключается. Графические операции в окне или клиентской области исключаются из правого и нижнего краев включающего прямоугольника.

Иногда приложениям может потребоваться сопоставить координаты в одном окне с координатами другого окна. Приложение может сопоставлять координаты с помощью функции MapWindowPoints . Если одно из окон является окном рабочего стола, функция эффективно преобразует координаты экрана в координаты клиента и наоборот; окно рабочего стола всегда указывается в координатах экрана.

Рисование с помощью мыши

Вы можете разрешить пользователю рисовать линии с помощью мыши, нарисовав процедуру окна во время обработки сообщения WM_MOUSEMOVE. Система отправляет WM_MOUSEMOVE сообщение в процедуру окна всякий раз, когда пользователь перемещает курсор в окне. Для рисования линий процедура окна может получить контекст отображаемого устройства и нарисовать линию в окне между текущей и предыдущей позицией курсора.

В следующем примере процедура окна подготавливается к рисованию, когда пользователь нажимает и удерживает левую кнопку мыши (отправляя сообщение WM_LBUTTONDOWN). Когда пользователь перемещает курсор в пределах окна, процедура окна получает ряд WM_MOUSEMOVE сообщений. Для каждого сообщения процедура окна рисует линию, соединяющую предыдущую и текущую позицию. Чтобы нарисовать линию, процедура использует GetDC для получения контекста отображаемого устройства; после завершения рисования и перед возвратом из сообщения процедура использует функцию ReleaseDC для освобождения контекста отображаемого устройства.

Как только пользователь отпустит кнопку мыши, процедура окна снимает флаг, и рисование останавливается (что отправляет сообщение WM_LBUTTONUP).

```
BOOL fDraw = FALSE;
POINT ptPrevious;

 case WM_LBUTTONDOWN:
   fDraw = TRUE;
   ptPrevious.x = LOWORD(lParam);
   ptPrevious.y = HIWORD(lParam);
   return 0L;
 case WM_LBUTTONUP:
   if (fDraw)
   {
      hdc = GetDC(hwnd);
      MoveToEx(hdc, ptPrevious.x, ptPrevious.y, NULL);
      LineTo(hdc, LOWORD(lParam), HIWORD(lParam));
      ReleaseDC(hwnd, hdc);
   }
   fDraw = FALSE;
   return 0L;
 case WM_MOUSEMOVE:
   if (fDraw)
   {
      hdc = GetDC(hwnd);
      MoveToEx(hdc, ptPrevious.x, ptPrevious.y, NULL);
      LineTo(hdc, ptPrevious.x = LOWORD(lParam),
        ptPrevious.y = HIWORD(lParam));
      ReleaseDC(hwnd, hdc);
   }
   return 0L;
```

Приложение, которое включает рисование, как в этом примере, обычно записывает точки или линии, чтобы линии можно было перерисовывать при каждом обновлении окна. Приложения для рисования часто используют контекст устройства памяти и связанное растровое изображение для хранения линий, нарисованных с помощью мыши.

Рисование с временными интервалами

Вы можете рисовать с временными интервалами, создав таймер с помощью функции SetTimer . Используя таймер для отправки WM_TIMER сообщений в процедуру окна через регулярные интервалы времени, приложение может выполнять простую анимацию в клиентской области, пока другие приложения продолжают работать.

В следующем примере приложение отскакивает star из стороны в сторону в клиентской области. Каждый раз, когда оконная процедура получает сообщение WM_TIMER, процедура удаляет star в текущей позиции, вычисляет новую позицию и рисует star в новой позиции. Процедура запускает таймер путем вызова SetTimer при обработке сообщения WM_CREATE .

```
RECT rcCurrent = {0,0,20,20};
POINT aptStar[6] = {10,1, 1,19, 19,6, 1,6, 19,19, 10,1}; int X = 2, Y = -1, idTimer = -1;
BOOL fVisible = FALSE;
HDC hdc;
```

```
LRESULT APIENTRY WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam) {
    PAINTSTRUCT ps;
    RECT rc;

    switch (message)
    {
      case WM_CREATE:

        // Calculate the starting point.

        GetClientRect(hwnd, &rc);
        OffsetRect(&rcCurrent, rc.right / 2, rc.bottom / 2);

        // Initialize the private DC.

        hdc = GetDC(hwnd);
        SetViewportOrgEx(hdc, rcCurrent.left,
          rcCurrent.top, NULL);
        SetROP2(hdc, R2_NOT);

        // Start the timer.

        SetTimer(hwnd, idTimer = 1, 10, NULL);
        return 0L;

      case WM_DESTROY:
        KillTimer(hwnd, 1);
        PostQuitMessage(0);
        return 0L;

      case WM_SIZE:
        switch (wParam)
        {
          case SIZE_MINIMIZED:

            // Stop the timer if the window is minimized.

            KillTimer(hwnd, 1);
            idTimer = -1;
            break;

          case SIZE_RESTORED:

            // Move the star back into the client area
            // if necessary.

            if (rcCurrent.right > (int) LOWORD(lParam))
            {
              rcCurrent.left =
                (rcCurrent.right =
                  (int) LOWORD(lParam)) - 20;
```

```
            }
            if (rcCurrent.bottom > (int) HIWORD(lParam))
            {
               rcCurrent.top =
                  (rcCurrent.bottom =
                     (int) HIWORD(lParam)) - 20;
            }

            // Fall through to the next case.

         case SIZE_MAXIMIZED:

            // Start the timer if it had been stopped.

            if (idTimer == -1)
               SetTimer(hwnd, idTimer = 1, 10, NULL);
            break;
      }
      return 0L;

   case WM_TIMER:

      // Hide the star if it is visible.

      if (fVisible)
         Polyline(hdc, aptStar, 6);

      // Bounce the star off a side if necessary.

      GetClientRect(hwnd, &rc);
      if (rcCurrent.left + X < rc.left ||
         rcCurrent.right + X > rc.right)
         X = -X;
      if (rcCurrent.top + Y < rc.top ||
         rcCurrent.bottom + Y > rc.bottom)
         Y = -Y;

      // Show the star in its new position.

      OffsetRect(&rcCurrent, X, Y);
      SetViewportOrgEx(hdc, rcCurrent.left,
         rcCurrent.top, NULL);
      fVisible = Polyline(hdc, aptStar, 6);

      return 0L;

   case WM_ERASEBKGND:

      // Erase the star.

      fVisible = FALSE;
      return DefWindowProc(hwnd, message, wParam, lParam);
```

```
        case WM_PAINT:

            // Show the star if it is not visible. Use BeginPaint
            // to clear the update region.

            BeginPaint(hwnd, &ps);
            if (!fVisible)
               fVisible = Polyline(hdc, aptStar, 6);
            EndPaint(hwnd, &ps);
            return 0L;
      }
      return DefWindowProc(hwnd, message, wParam, lParam);
   }
```

Это приложение использует контекст частного устройства, чтобы свести к минимуму время, необходимое для подготовки контекста устройства к рисованию. Процедура window извлекает и инициализирует контекст частного устройства при обработке сообщения WM_CREATE, задавая режим работы двоичного растра, чтобы позволить стереть и отрисовывать star с помощью того же вызова функции Polyline. Процедура окна также задает источник окна просмотра, чтобы star можно было нарисовать с использованием одного и того же набора точек независимо от положения star в клиентской области.

Приложение использует сообщение WM_PAINT для рисования star при каждом обновлении окна. Процедура окна рисует star, только если она не видна, то есть только в том случае, если она была стерта сообщением WM_ERASEBKGND. Процедура окна перехватывает сообщение WM_ERASEBKGND для задания переменной fVisible , но передает сообщение в DefWindowProc , чтобы система рисовать фон окна.

Приложение использует сообщение WM_SIZE для остановки таймера при свернутом окне и для перезапуска таймера при восстановлении свернутого окна. Процедура окна также использует сообщение для обновления текущей позиции star если размер окна был уменьшен, так что star больше не находится в клиентской области. Приложение отслеживает текущее положение star с помощью структуры, заданной параметром rcCurrent, который определяет ограничивающий прямоугольник для star. Сохранение всех углов прямоугольника в клиентской области сохраняет star в этой области. Процедура окна изначально размещает star в клиентской области при обработке сообщения WM_CREATE.

Для рисования графических примитивов в оконных приложениях используются 4 основных типа объектов:

точка (Pixel);
перо (Pen);
кисть (Brush);
фон (Background).

Точка

Цвет точки задается с помощью функции

```
COLORREF SetPixel(
  _In_  HDC hdc,   // дескриптор контекста устройства
  _In_  int X,    // x-координата точки
  _In_  int Y,   // y-координата точки
```

```
  _In_  COLORREF crColor ); // цвет точки
```

В случае удачного завершения возвращаемое значение функции дублирует цвет точки, в случае ошибки возвращает -1.

Цвет точки представляет собой 32-битное число, заданное в системе RGB:

Можно также воспользоваться функцией

```
RGB(
  _ Red As Integer,  // красный
  _ Green As Integer,  // зеленый
  _ Blue As Integer);  // синий
```

Значения красного, зеленого и синего используются в диапазоне 0…255.

Перо

Перо используется для рисования линий и контуров замкнутых фигур. Цвет пера задается функцией

```
HPEN CreatePen(
  _In_  int fnPenStyle,      // стиль пера
  _In_  int nWidth,      // ширина пера (в пикселях)
  _In_  COLORREF crColor );  // цвет пера
```

Стили пера fnPenStyle могут быть заданы согласно таблице

Значение		*Тип*	*Описание*
PS_SOLID	0		Сплошное перо
PS_DASH	1		Прерывистое (пунктирное) перо.
PS_DOT	2		Точечное (штриховое) перо.
PS_DASHDOT	3		Штрих-пунктир
PS_DASHDOTDOT	4		Две точки — пунктир
PS_NULL	5		Невидимое перо

При успешном завершении функция возвращает дескриптор пера, в случае неудачи — константу NULL.

Кисть

Кисть используется для закрашивания замкнутых объектов. Цвет кисти задается с помощью функции

```
HBRUSH CreateSolidBrush(
  _In_  COLORREF crColor );   // цвет кисти
```

При успешном завершении функция возвращает дескриптор кисти, в случае неудачи — константу NULL.

Эта же функция используется для задания цвета ***фона***.

Можно заранее создать несколько кистей и перьев, а затем выбирать нужные с помощью функции

```
HGDIOBJ SelectObject(
  _In_  HDC hdc,      // дескриптор контекста устройства
  _In_  HGDIOBJ hgdiobj );     // дескриптор объекта
```

Рисование графических примитивов

Перемещение в указанную точку осуществляется функцией:

```
BOOL MoveToEx(
  _In_   HDC hdc,    // дескриптор контекста устройства
  _In_   int X,  // координата x точки
  _In_   int Y,    // координата y точки
  _Out_  LPPOINT lpPoint );   // указатель на структуру POINT
```

Координаты точки x и y определяются в пикселях относительно левого верхнего угла. В случае успешного выполнения возвращает ненулевое значение.

Структура POINT имеет вид

```
typedef struct tagPOINT {
  LONG x;
  LONG y; } POINT, *PPOINT;
```

Рисование отрезков осуществляется функцией:

```
BOOL LineTo(
  _In_  HDC hdc,       // дескриптор контекста устройства
  _In_  int nXEnd,     // координата x конечной точки
  _In_  int nYEnd );   // координата y конечной точки
```

В случае успешного выполнения возвращает ненулевое значение.

Рисование прямоугольника осуществляется функцией:

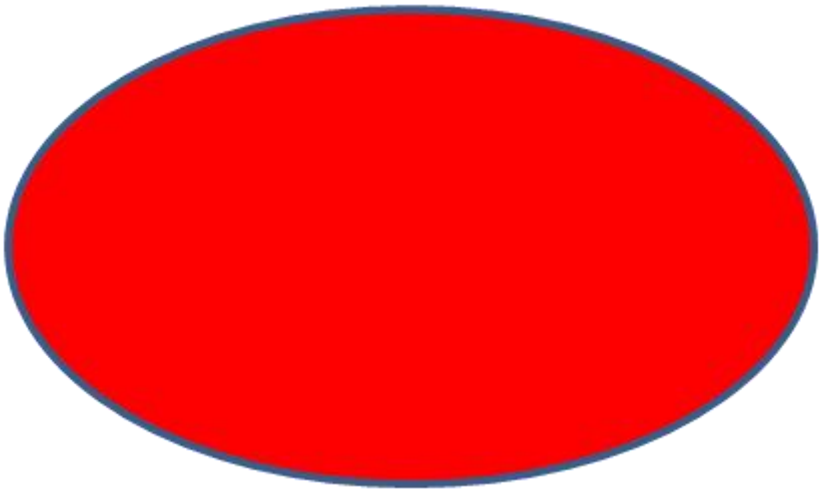
```
BOOL Rectangle(
_In_  HDC hdc,          // дескриптор контекста устройства
_In_  int nLeftRect,    // x-координата верхнего левого угла
_In_  int nTopRect,     // y-координата верхнего левого угла
_In_  int nRightRect,   // x-координата нижнего правого угла
_In_  int nBottomRect);   // координата нижнего правого угла
```

Рисование прямоугольника начинается из точки, в которую осуществлено перемещение с помощью функции MoveTo().

В случае успешного выполнения возвращает ненулевое значение.

Рисование эллипса осуществляется функцией:

```
BOOL Ellipse(
  _In_  HDC hdc,          // дескриптор контекста устройства
  _In_  int nLeftRect,    // x-координата верхнего левого угла
  _In_  int nTopRect,     // y-координата верхнего левого угла
  _In_  int nRightRect,   // x-координата нижнего правого угла
  _In_  int nBottomRect);   // координата нижнего правого угла
```



В случае успешного выполнения возвращает ненулевое значение.

Рисование дуги осуществляется функцией:

```
BOOL ArcTo(
```

```
_In_  HDC hdc,   // дескриптор контекста устройства
_In_  int nLeftRect,   // x-координата верхнего левого угла
_In_  int nTopRect,   // y-координата верхнего левого угла
_In_  int nRightRect,  // x-координата нижнего правого угла
_In_  int nBottomRect,   // y-координата нижнего правого угла
_In_  int nXRadial1,   // x- координата конца первого радиуса
_In_  int nYRadial1,   // y- координата конца первого радиуса
_In_  int nXRadial2,   // x- координата конца второго радиуса
_In_  int nYRadial2 );   // y- координата конца второго радиуса
```

В случае успешного выполнения возвращает ненулевое значение.

Вывод текста в окно

Для вывода текста в поле окна используется функция

```
BOOL TextOut(
_In_  HDC hdc,   // дескриптор контекста устройства
_In_  int nXStart,    // x-координата начала вывода текста
_In_  int nYStart,   // y-координата начала вывода текста
_In_  LPCTSTR lpString,  // указатель на строку текста
_In_  int cchString );  // количество символов для вывода
```

В случае успешного выполнения возвращает ненулевое значение.

Задать цвет фона под буквами можно с помощью функции

```
COLORREF SetBkColor(
_In_  HDC hdc,   // дескриптор контекста устройства
_In_  COLORREF crColor );    // цвет
```

Задать цвет букв можно с помощью функции

```
COLORREF SetETextColor(
_In_  HDC hdc,   // дескриптор контекста устройства
_In_  COLORREF crColor );    // цвет
```

В случае неудачного завершения эти функции возвращают константу CLR_INVALID=0xFFFF.

Использование графических функций

Для перерисовки окна, в котором будут отображаться графические объекты, будем использовать обработку сообщения WM_PAINT.

Обработка сообщения WM_PAINT почти всегда начинается с вызова функции:

```
HDC BeginPaint(
  _In_   HWND hwnd,
  _Out_  LPPAINTSTRUCT lpPaint );
```

При обработке вызова BeginPaint(), Windows обновляет фон рабочей области с помощью кисти, заданной в поле hbrBackground структуры WNDCLASS, описанной здесь. Вызов BeginPaint() делает всю рабочую область действительной (не требующей перерисовки) и возвращает описатель контекста устройства. Контекст устройства описывает физическое устройство вывода информации (например, экран) и его драйвер. Описатель контекста устройства необходим для вывода в рабочую область окна текста и графики.

Аргументы функции:
hwnd – дескриптор окна;
lpPaint – указатель на структуру PAINTSTRUCT.
Структура PAINTSTRUCT имеет вид

```
 typedef struct tagPAINTSTRUCT {
  HDC  hdc;
  BOOL fErase;
  RECT rcPaint;
  BOOL fRestore;
  BOOL fIncUpdate;
  BYTE rgbReserved[32]; } PAINTSTRUCT, *PPAINTSTRUCT;
```

Члены структуры:
hdc – дескриптор контекста устройства.
fErase – ненулевое значение стирает фон.
rcPaint – структура RECT, определяющая верхний левый и нижний правый углы рабочей области.

```
typedef struct _RECT {
  LONG left; LONG top;
  LONG right; LONG bottom; } RECT, *PRECT;
```

fRestore, **fIncUpdate**, **rgbReserved** – зарезервировано, используется системой.
Обработка сообщения WM_PAINT почти всегда заканчивается вызовом функции:

```
BOOL EndPaint(
  _In_  HWND hWnd,
  _In_  const PAINTSTRUCT *lpPaint );
```

Функция EndPaint() освобождает описатель контекста устройства, после чего его значение нельзя использовать. Возвращает всегда ненулевое значение.

Получение дескриптора контекста устройства осуществляется вызовом функции:

```
HDC GetDC(_In_  HWND hWnd);
```

hWnd – дескриптор окна, для которого используется контекст устройства.
Возвращаемое значение – дескриптор контекста устройства.

Функция

```
int ReleaseDC(
  _In_  HWND hWnd,
  _In_  HDC hDC );
```

освобождает контекст устройства hDC для данного окна hWnd, после чего значение контекста устройства нельзя использовать. Возвращает всегда ненулевое значение.

Пример График функции y=sin(x).
График функции **y=sin(x)** симметричен относительно горизонтальной оси, поэтому должен быть построен в окне со смещением относительно левого верхнего угла окна, принятого за начало отсчета.

Координата X меняется в пределах [0;7], координата Y - [-1;1]. Величины MAX_X и MAX_Y представляют собой область допустимых значений по осям координат:

- **MAX_X** = maxX - minX = 7;
- **MAX_Y** = maxY - minY = 2.

Смещение графика функции представляет собой положение первой точки относительно начала отсчета левого верхнего угла:

- смещение по оси X: **OffsetX = minX*width/MAX_X = 0**;
- смещение по оси Y: **OffsetY = maxY *height/MAX_Y**

Значение minX=0 представляет собой минимальное значение координаты x из области допустимых значений.

Значение maxY=1 представляет собой максимальное значение координаты y из области допустимых значений. Рассматривается именно максимальное значение, поскольку за начало отсчета принят левый верхний угол окна, и координата y увеличивается по направлению вниз.

Для того чтобы график функции разместился в окне необходимо рассчитать масштабные коэффициенты по осям. Масштабный коэффициент представляет собой отношение размера окна к области допустимых значений функции.

- масштабный коэффициент X: **ScaleX = width / MAX_X**;
- масштабный коэффициент Y: **ScaleY = height / MAX_Y**.

Вычисление координат следующей точки (x;y) графика в окне будет осуществляться по формулам:

- координата **X=OffsetX + x*ScaleX**;
- координата **Y=OffsetY + y*ScaleY**,

где (x;y) - координаты точки, полученные из функции y=sin(x).

```
#include <windows.h>
```

```
#include <math.h>
const int NUM = 70; // количество точек
LONG WINAPI WndProc(HWND, UINT, WPARAM, LPARAM);
double **x; // массив данных
     // Задание исходных данных для графика
     // (двумерный массив, может содержать несколько рядов данных)
double ** getData(int n)
{
 double **f;
 f = new double*[2];
 f[0] = new double[n];
 f[1] = new double[n];
 for (int i = 0; i<n; i++)
 {
  double x = (double)i * 0.1;
  f[0] = x;
  f[1] = sin(x);
 }
 return f;
}
// Функция рисования графика
void DrawGraph(HDC hdc, RECT rectClient,
 double **x, // массив данных
 int n, // количество точек
 int numrow = 1) // количество рядов данных (по умолчанию 1)
{
 double OffsetY, OffsetX;
 double MAX_X, MAX_Y;
 double ScaleX, ScaleY;
 double min, max;
 int height, width;
 int X, Y; // координаты в окне (в px)
 HPEN hpen;
 height = rectClient.bottom - rectClient.top;
 width = rectClient.right - rectClient.left;
 // Область допустимых значений X
 min = x[0][0];
 max = x[0][0];
 for (int i = 0; i<n; i++)
 {
  if (x[0] < min) min = x[0];
  if (x[0] > max) max = x[0];
 }
 double temp = max - min;
 MAX_X = max - min;
 OffsetX = min*width / MAX_X; // смещение X
 ScaleX = (double)width / MAX_X; // масштабный коэффициент X
            // Область допустимых значений Y
 min = x[1][0];
 max = x[1][0];
 for (int i = 0; i<n; i++)
 {
```

```
    for (int j = 1; j <= numrow; j++)
    {
     if (x[j] < min) min = x[j];
     if (x[j] > max) max = x[j];
    }
  }
  MAX_Y = max - min;
  OffsetY = max*height / (MAX_Y); // смещение Y
  ScaleY = (double)height / MAX_Y; // масштабный коэффициент Y
                // Отрисовка осей координат
  hpen = CreatePen(PS_SOLID, 0, 0); // черное перо 1px
  SelectObject(hdc, hpen);
  MoveToEx(hdc, 0, OffsetY, 0); // перемещение в точку (0;OffsetY)
  LineTo(hdc, width, OffsetY); // рисование горизонтальной оси
  MoveToEx(hdc, OffsetX, 0, 0); // перемещение в точку (OffsetX;0)
  LineTo(hdc, OffsetX, height); // рисование вертикальной оси (не видна)
  DeleteObject(hpen); // удаление черного пера
           // Отрисовка графика функции
  int color = 0xFF; // красное перо для первого ряда данных
  for (int j = 1; j <= numrow; j++)
  {
   hpen = CreatePen(PS_SOLID, 2, color); // формирование пера 2px
   SelectObject(hdc, hpen);
   X = (int)(OffsetX + x[0][0] * ScaleX); // начальная точка графика
   Y = (int)(OffsetY - x[j][0] * ScaleY);
   MoveToEx(hdc, X, Y, 0); // перемещение в начальную точку
   for (int i = 0; i<n; i++)
   {
    X = OffsetX + x[0] * ScaleX;
    Y = OffsetY - x[j] * ScaleY;
    LineTo(hdc, X, Y);
   }
   color = color << 8; // изменение цвета пера для следующего ряда
   DeleteObject(hpen); // удаление текущего пера
  }
}
// Главная функция
int WINAPI WinMain(HINSTANCE hInstance,
  HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
  HWND hwnd;
  MSG msg;
  WNDCLASS w;
  x = getData(NUM); // задание исходны данных
  memset(&w, 0, sizeof(WNDCLASS));
  w.style = CS_HREDRAW | CS_VREDRAW;
  w.lpfnWndProc = WndProc;
  w.hInstance = hInstance;
  w.hbrBackground = CreateSolidBrush(0x00FFFFFF);
  w.lpszClassName = "My Class";
  RegisterClass(&w);
  hwnd = CreateWindow("My Class", "График функции",
```

```
    WS_OVERLAPPEDWINDOW,
    500, 300, 500, 380, NULL, NULL,
    hInstance, NULL);
  ShowWindow(hwnd, nCmdShow);
  UpdateWindow(hwnd);
  while (GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return msg.wParam;
}
// Оконная функция
LONG WINAPI WndProc(HWND hwnd, UINT Message,
  WPARAM wparam, LPARAM lparam)
{
  HDC hdc;
  PAINTSTRUCT ps;
  switch (Message)
  {
  case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    DrawGraph(hdc, ps.rcPaint, x, NUM); // построение графика
                  // Вывод текста y=sin(x)
    SetTextColor(hdc, 0x00FF0000); // синий цвет букв
    TextOut(hdc, 10, 20, "y=sin(x)", 8);
    EndPaint(hwnd, &ps);
    break;
  case WM_DESTROY:
    PostQuitMessage(0);
    break;
  default:
    return DefWindowProc(hwnd, Message, wparam, lparam);
  }
  return 0;}
```

Работа с графикой (C++/CLI)

Функция OnPaint предназначена для приложения Windows Forms, скорее всего, созданного с помощью мастера приложений Visual Studio.

Изображение представлено классом Image . Данные изображения загружаются из JPG-файла с помощью System.Drawing.Image.FromFile метода. Прежде чем изображение нарисовано в форму, форма изменяется для размещения изображения. Рисунок изображения выполняется с System.Drawing.Graphics.DrawImage помощью метода.

Image Классы Graphics находятся в System.Drawing пространстве имен.

```
#using <system.drawing.dll>
using namespace System;
using namespace System::Drawing;

protected:virtual Void Form1::OnPaint(PaintEventArgs^ pe) override
{
    Graphics^ g = pe->Graphics;
    Image^ image = Image::FromFile("SampleImage.jpg");
    Form::ClientSize = image->Size;
    g->DrawImage( image, 0, 0, image->Size.Width, image->Size.Height );
}
```

Рисование фигур с помощью платформа .NET Framework

```
#using <system.drawing.dll>
using namespace System;
using namespace System::Drawing;
// ...
protected:virtual Void Form1::OnPaint(PaintEventArgs^ pe ) override
{
   Graphics^ g = pe->Graphics;
   g->Clear(Color::AntiqueWhite);

   Rectangle rect = Form::ClientRectangle;
   Rectangle smallRect;
   smallRect.X = rect.X + rect.Width / 4;
   smallRect.Y = rect.Y + rect.Height / 4;
   smallRect.Width = rect.Width / 2;
   smallRect.Height = rect.Height / 2;

   Pen^ redPen = gcnew Pen(Color::Red);
   redPen->Width = 4;
   g->DrawLine(redPen, 0, 0, rect.Width, rect.Height);

   Pen^ bluePen = gcnew Pen(Color::Blue);
   bluePen->Width = 10;
   g->DrawArc( bluePen, smallRect, 90, 270 );
}
```

System.Drawing Пространство имен

Предоставляет доступ к основным графическим функциям GDI+. Пространства System.Drawing.Drawing2Димен, System.Drawing.Imagingи System.Drawing. Text предоставляют более расширенные функциональные возможности.

Классы

ColorConverter	Преобразует цвета одного типа данных в другой. Доступ к данному классу осуществляется с помощью объекта TypeDescriptor.
ColorTranslator	Преобразует цвета в структуры GDI+ Color и из них. Этот класс не наследуется.
PointConverter	Преобразует объект Point из одного типа данных в другой.
RectangleConverter	Преобразует прямоугольники из одного типа данных в другой. Доступ к данному классу осуществляется с помощью объекта TypeDescriptor.
SizeConverter	Класс SizeConverter используется для преобразования одного типа данных в другой. Доступ к данному классу осуществляется с помощью объекта TypeDescriptor.
SizeFConverter	Преобразует объекты SizeF из одного типа в другой.
SystemColors	Каждое свойство класса SystemColors является структурой Color, которая представляет собой цвет элемента изображения Windows.

Структуры

Color	Представляет цвета в терминах каналов альфа, красного, зеленого и синего (ARGB).
Point	Представляет упорядоченную пару целых чисел — координат X и Y, определяющую точку на двумерной плоскости.
PointF	Представляет упорядоченную пару координат X и Y с плавающей запятой, определяющую точку на двумерной плоскости.
Rectangle	Содержит набор из четырех целых чисел, определяющих расположение и размер прямоугольника.
RectangleF	Содержит набор из четырех чисел с плавающей запятой, определяющих расположение и размер прямоугольника. Для расширения функций области используйте объект Region.
Size	Сохраняет упорядоченную пару целых чисел, указывающих Height и Width.
SizeF	Содержит упорядоченную пару чисел с плавающей запятой, обычно ширину и высоту прямоугольника.

Перечисления

KnownColor	Задает известные системные цвета.