Николай Тюкачев,
Игорь Илларионов,
Виктор Хлебостроев

Программирование графики в Delphi

+CD

- Цветовые модели и форматы графических файлов
- Средства для разработки мультимедийных приложений
- Компоненты для работы с деловой графикой
- Методы построения кривых в задачах интерполяции, сглаживания, аппроксимации, методы Эрмита, Безье и B-сплайнов
- Создание растровых и векторных графических редакторов
- Работа с трехмерными изображениями
- Задачи триангуляции, проектирование и реализация ГИС

bhv

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.1
Т98

Тюкачѐв, Н. А.

Т98 Программирование графики в Delphi / Н. А. Тюкачѐв, И. В. Илларионов, В. Г. Хлебостроев. — СПб.: БХВ-Петербург, 2008. — 784 с.: ил. + CD-ROM

ISBN 978-5-9775-0253-5

Книга написана на базе курса лекций, читаемых авторами. Рассмотрены основные классы и функции среды Delphi, которые используются для создания графических и мультимедийных приложений. Описаны цветовые модели, основные форматы графических файлов, а также методы построения кривых в задачах интерполяции, сглаживания, аппроксимации, методы Эрмита, Безье и B-сплайнов. Приведены алгоритмы триангуляции поверхностей в трехмерном пространстве. На конкретных примерах показан весь процесс разработки основных типов приложений — пакетов деловой графики, работы с трехмерными объектами, растровых и векторных графических редакторов, геоинформационных систем. Каждый раздел сопровождается задачами различной сложности для самостоятельного решения. На прилагаемом компакт-диске представлено более 30 проектов, описанных в книге.

Для программистов

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Наталья Баскакова*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.06.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 63,21.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.003650.04.08
от 14.04.2008 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0253-5

Оглавление

Глава 1

Рисование в Delphi

1.1. Моделирование цветов

Мы видим мир цветным, и это одна из самых больших биологических загадок. Человеческий глаз может воспринимать цветовые волны с длинами от 380 нм до 780 нм. Это лишь незначительный диапазон спектра электромагнитных волн. Зрительные пигменты глаза состоят из колбочек трех типов, чувствительных к синему, зеленому и красному цвету. Эффективность поглощения световых волн существенно различается для различных типов колбочек. Особенно хорошо воспринимается зеленый свет, красный свет воспринимается несколько хуже, а чувствительность глаза к синему свету еще ниже. Многочисленные психологические тесты позволяют считать, что яркость можно вычислить по формуле [124]:

Яркость= 0,59×Зеленый + 0,3×Красный + 0,11×Синий

Цвет представляет собой индивидуальное ощущение, не позволяющее судить о его спектральном составе. Для обработки изображений на современной технике такая субъективность нежелательна. Именно для цели обработки изображений были разработаны математические методы точного описания цвета, называемые цветовыми моделями. Одна из них носит название `RGB` (`Red`, `Green`, `Blue`). В рамках этой модели предполагается получение цветов смешением красного, зеленого и синего цветов. Смесь синего и зеленого, например, дает голубой цвет, а смесь красного и синего — пурпурный.

Операционные системы, такие как Windows, позволяют устанавливать различные графические режимы:

- 16-цветный режим;
- 256-цветный режим, при котором для хранения цвета каждой точки (пиксела) выделяется один байт;
- режим `High Color`, при котором для хранения цвета каждой точки выделяется два байта, что позволяет использовать до 65000 цветов;
- режим `True Color`, при котором для хранения каждого пиксела выделяется четыре байта (32 бита). Чаще всего из этих 32 битов используется 24, что позволяет изменять доли красного, синего и зеленого цветов в диапазоне от 0 до 255.

Математически систему RGB можно представить в виде куба, каждая точка которого однозначно определяется координатами R, G и B. Так как в системе RGB цвета определяются смешением основных цветов, то она особенно удобна для устройств, излучающих цветовые волны (видеомонитор, цветной телевизор).

Цветовая система RGB очень проста, но при ее использовании возникают две проблемы. Первая — зависимость от аппаратуры, вторая — невозможность получить все цвета смешением красного, зеленого и синего цветов.

Другой, часто используемой цветовой системой, является модель CMYK. Название этой модели образовано из первых букв названий цветов Cyan, Magenta, Yellow (голубой, пурпурный, желтый) и последней буквы слова blacK (черный) (используется буква К, так как буква В уже задействована в названии цветовой модели RGB). Данная модель предназначена для описания цвета отражающих поверхностей. Именно эта модель служит теоретической основой цветной печати. В отличие от модели RGB, цвета в модели CMYK получаются не аддитивно (суммированием), а субтрактивно (вычитанием). Например, поверхность голубого цвета отражает синий и зеленый, но поглощает красный, пурпурный поглощает зеленый, а желтый поглощает синий цвет.

Применяемые на практике цветные краски далеко не идеальны, и смешением всех красок, как правило, не удается получить черный цвет. Поэтому для повышения контрастности применяется чисто черный краситель.

1.2. Полотно компонентов

Рисовать в Delphi проще, чем на бумаге. При разработке проекта в вашем распоряжении находятся полотно (свойство Canvas), карандаш (свойство Pen), кисть (свойство Brush) и некоторое количество примитивов (линий, прямоугольников, эллипсов и т. д.). Правда, опубликованным свойством Canvas обладают далеко не все компоненты. В частности, этим свойством обладают компоненты форма (класс TForm), таблица (класс TStringGrid), растровые изображения (класс TImage), принтер (класс TPrinter). У карандаша и кисти можно менять цвет (Color) и стиль (Style). Этот набор инструментов позволяет создавать достаточно сложные рисунки математического и инженерного содержания. Кроме того, Delphi позволяет использовать многие ресурсы Windows: графические файлы, фильмы и звуковые файлы.

Полотно — это прямоугольная сетка, состоящая из маленьких квадратов, называемых пикселами (свойство Pixels[X,Y]: TColor). Каждый пиксел имеет свой номер, точнее два номера. Первый номер указывает на горизонтальное расположение пиксела, а второй — на вертикальное. Левый верхний пиксел полотна имеет координаты 0,0 — Pixels[0,0]. Общее количество пикселов по горизонтали доступно через свойство Width, а по вертикали — через свойство Height. Каждый пиксел может быть закрашен любым доступным для Windows цветом. Рисование пиксела на по-

лотне происходит в тот момент, когда мы присваиваем элементу массива `Pixels` номер цвета. Например, присваивание:

```
Image1.Canvas.Pixels[100,100]:= clRed
```

приведет к рисованию красной точки с координатами `100,100`. Мы можем узнать номер цвета любого пиксела обратным действием:

```
Color:= Image1.Canvas.Pixels[100,100].
```

Класс цвета точки `TColor` определен как `longint`:

```
TColor = - $80000000 ..$7FFFFFFF.
```

Четыре байта переменных этого типа содержат информацию о долях синего (B), зеленого (G) и красного (R) цветов и устроены следующим образом: `$00BBGGRR`. Доля каждого цвета может меняться от 0 до 255. Поэтому для рисования красной точки, например, мы должны выбрать цвет с номером `$0000FF`. В Delphi определен набор констант для цветов. Список этих констант можно увидеть в инспекторе объектов или в модуле `Grahics`.

Впрочем, обращение к точкам полотна через двумерный массив `Pixels[X,Y]` — это самый медленный способ рисования. Для более быстрого рисования используют специальные методы канвы `Canvas`: например, `LineTo` — линии, `Arc` — дуги, `RectAngle` — прямоугольники, `TextOut` — вывод текста и т. д.

1.3. Пример использования графики

Для демонстрации возможностей использования графики, рассмотрим простой проект, в котором на полотне компонента `Image1` будем рисовать кривые Лиссажу (рис. 1.1). Проект приведен на компакт-диске (полное содержание которого приведено в *прил. 2*) в папке **Примеры | Глава 1 | Кривые Лиссажу**.

Все помнят из школьного курса физики, что если на вход X и вход Y осциллографа подать синусоидальные сигналы, то на экране появится отрезок прямой, но при некотором сдвиге фаз отрезок расслоится и превратится в эллипс, который превращается в окружность при сдвиге фаз $\pi/2$ и одинаковых амплитудах. Это обстоятельство используется для измерения сдвига фаз в любительских радиоизмерениях. В общем виде кривые Лиссажу задаются параметрическими уравнениями:

$$x = A \times \sin(w_x \times t + w_1),$$

$$y = A \times \sin(w_y \times t + w_2)$$

и зависят от четырех параметров: двух частот w_x, w_y и двух фазовых смещений w_1 и w_2. В проекте функции, определяющие кривые Лиссажу, обозначены как `Fx` и `Fy` (листинг 1.1).

Листинг 1.1. Функции, определяющие кривые Лиссажу

```
function TForm1.Fx(t: real): real;
begin
  Fx:=sin(wx*t+w1);
end;
function TForm1.Fy(t: real): real;
begin
  Fy:=sin(wy*t+w2);
end;
```

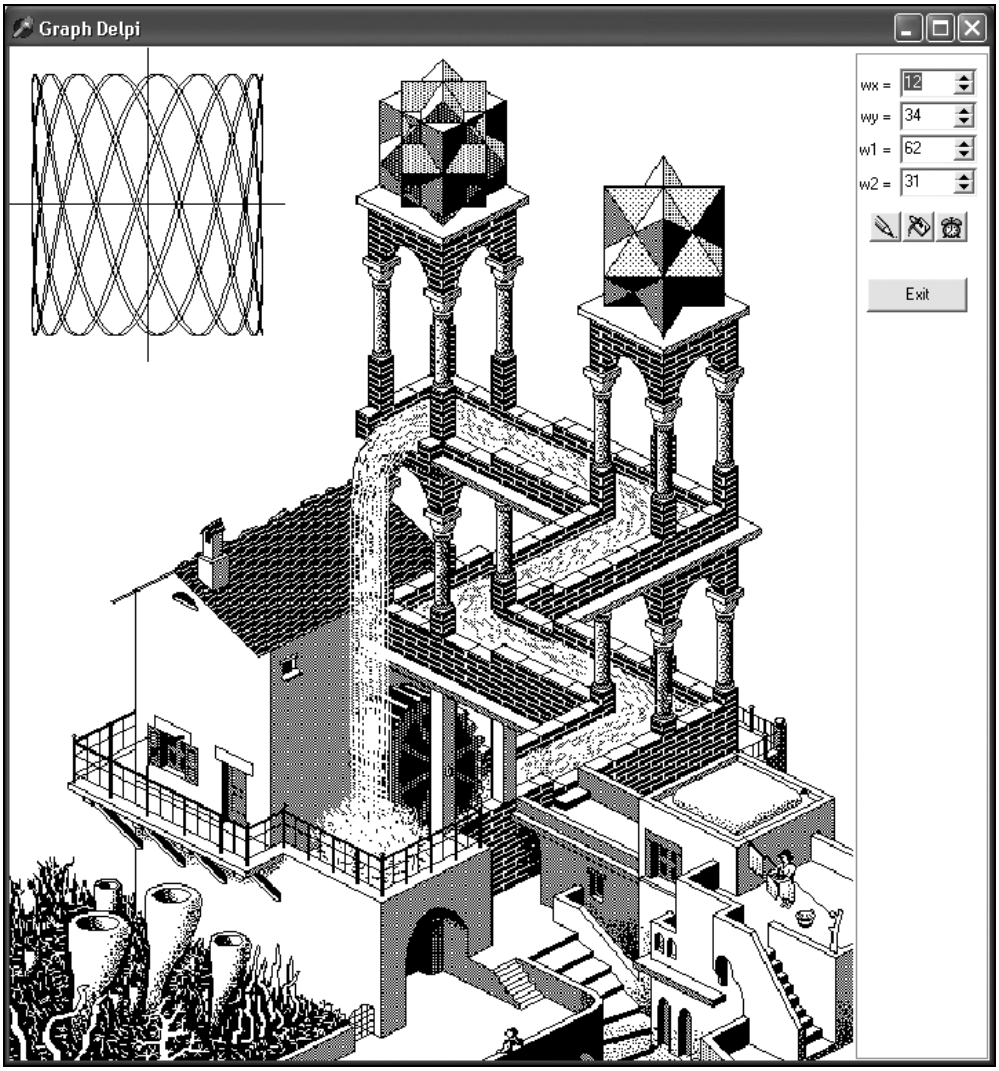

Рис. 1.1. Проект "Кривые Лиссажу" с копией картины Мориса Эшера

Для изменения параметров `wx`, `wy`, `w1` и `w2` во время выполнения проекта поместим на единственную форму (рис. 1.1) четыре компонента `TSpinEdit`. Для более плавного изменения значений параметров используем коэффициент масштабирования равный 1/10. Например:

```
wx:=SpinEditWx.Value/10.
```

Прежде чем рисовать график функции на бумаге, мы выбираем на ней окно рисования, задавая интервалы изменения переменных `x∈[x1,x2]` и `y∈[y1,y2]`. При переходе от "бумажного" окна `(x1,y1),(x2,y2)` к окну на экране `(I1,J1),(I2,J2)` придется масштабировать координаты точки с помощью функций `II(x)` и `JJ(y)` (листинг 1.2).

Листинг 1.2. Функции масштабирования

```
function TForm1.II(x: real): integer;
// функция масштабирования по оси OX
begin
  II:=I1+Trunc((x-X1)*(I2-I1)/(X2-X1));
end;

function TForm1.JJ(y: real): integer;
// функция масштабирования по оси OY
begin
  JJ:=J2+Trunc((y-Y1)*(J1-J2)/(Y2-Y1));
end;
```

Размеры окна на экране и на бумаге зададим при запуске проекта (листинг 1.3).

Листинг 1.3. Инициализация переменных

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Bitmap:=TBitmap.Create;
  Bitmap.Transparent := true;       //false;
  Bitmap.TransparentMode := tmAuto; //tmFixed;
  BitMap.LoadFromFile('Waterfal.bmp');
  BitMap.TransParentColor := clWhite;

  n:=200;
  X1:=-1.2; X2:=1.2; Y1:=-1.2; Y2:=1.2;
  // Задание окна на экране
  I1:=0; J1:=0; I2:=2*Width div 5; J2:=2*Height div 5;
  // вычисление шага по оси OX
  h:=2*Pi/n;
  DrawGraphic;
end;
```

Более подробное обсуждение функций масштабирования отложим до следующих глав, а пока займемся рассмотрением основного для данного проекта метода `DrawGraphic` (листинг 1.4). Первые четыре строчки метода `DrawGraphic` присваивают значения компонентов `SpinEdit` соответствующим параметрам функций. Все остальные события происходят на полотне компонента `Image1`. Сначала назначается белый цвет пера `Pen.Color:=clWhite` и кисти `Brush.Color:= clWhite` и вызовом метода `RectAngle(0,0,Width,Height)` рисуется белый прямоугольник размером во все полотно, т. е. полотно `Image1` очищается.

Листинг 1.4. Рисование графика

```
procedure TForm1.DrawGraphic;
var i: integer;
    t: real;
begin
  wx:=SpinEditWx.Value/10;
  wy:=SpinEditWy.Value/10;
  w1:=SpinEditW1.Value/10;
  w2:=SpinEditW2.Value/10;
  with Canvas do begin
    RectAngle(0,0,Width,Height);
    FillRect(Rect(0,0,Width,Height));
    StretchDraw(Rect(0,0,W,Height),BitMap);
    // Построение осей координат
    MoveTo(II(0),JJ(Y1)); LineTo(II(0),JJ(Y2));
    MoveTo(II(x1),JJ(0)); LineTo(II(x2),JJ(0));
    // Построение графика функции отрезками
    t:=0; x:=Fx(t); y:=Fy(t); MoveTo(II(x),JJ(y));
    for i:=1 to 5*n do begin
      t:=t+h; x:=Fx(t);
      y:=Fy(t); LineTo(II(x),JJ(y));
    end;
  end;
end;
```

Затем, изменив цвет карандаша на черный `Pen.Color:=clBlack`, рисуем оси координат — прямые линии от точки `(x1,0)` до `(x2,0)` и от точки `(0,y1)` до `(0,y2)`, масштабируя "бумажные" координаты в экранные с помощью функций `II(x)` и `JJ(y)`. Прорисовку отрезка прямой производим следующим образом: сначала методом полотна `MoveTo(x1,y1)` переводим экранный указатель в точку `(x1,y1)`, потом методом `LineTo(x2,y2)` перемещаем его в точку `(x2,y2)`.

Переходим теперь к рисованию графика. При активизации формы был вычислен шаг изменения `h:=2*Pi/n` параметра `t`, который на периоде `2*Pi` принимает `n` значений. Перед началом рисования параметру `t` присвоим значение `0` и переместим указатель в точку `(x,y)`.

```
t:=0; x:=Fx(t); y:=Fy(t); MoveTo(II(x),JJ(y));
```

Будем увеличивать `t` на `h` в цикле и рисовать отрезок до следующей точки.

```
t:=t+h; x:=Fx(t); y:=Fy(t); LineTo(II(x),JJ(y));
```

Для компонента `SpinEditWx` назначим обработчик события, который при изменении свойства `SpinEditWx.Value` перерисует график (листинг 1.5).

Листинг 1.5. Событие при изменении *SpinEditWx*

```
procedure TForm1.SpinEditWxChange(Sender: TObject);
begin
  DrawGraphic;
end;
```

Эту же процедуру используем в качестве обработчика события `onChange` для остальных компонентов.

Проект для рисования графика функции готов. Осталось нажать клавишу <F9> для запуска проекта на выполнение.

Усложним наш проект, введя возможность изменения цвета пера и кисти. Для этого поставим на форму две кнопки `SpeedButton1` и `SpeedButton2` и используем метод `ColorDialog1`, вызывающий диалоговое окно **Цвет** для выбора цвета. При нажатии кнопки `SpeedButton1` будем вызывать процедуру обработки события `SpeedButton1Click` (листинг 1.6).

Листинг 1.6. Изменение цвета кисти

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then begin
    Canvas.Brush.Color:=ColorDialog1.Color;
    DrawGraphic;
  end;
end;
```

После выбора в диалоговом окне **Цвет** нового цвета, цвет кисти изменится `Image1.Canvas.Brush.Color:=ColorDialog1.Color` и будет перерисован график вызовом метода `DrawGraphic`.

При нажатии кнопки `SpeedButton2` тот же метод `ColorDialog1` используется для изменения цвета пера (листинг 1.7).

Листинг 1.7. Изменение цвета пера

```
procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  if ColorDialog1.Execute then begin
    Canvas.Pen.Color:=ColorDialog1.Color;
    DrawGraphic;
  end;
end;
```

Поставим на форму компонент Timer1 и две кнопки SpeedButton3 и SpeedButton4. Единственное событие счетчика времени Timer1Timer(Sender) вызывается через интервал времени, определенный свойством Timer1.Interval, если Timer1.Enabled=true. До начала выполнения проекта зададим значение свойства Timer1.Enabled=false, а для кнопки SpeedButton3 определим событие при щелчке кнопки мыши (листинг 1.8).

Листинг 1.8. Подключение таймера

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  Timer1.Enabled:=not Timer1.Enabled;
end;
```

При нажатии кнопки SpeedButton3 будет запущен механизм таймера, который через 1000 мс будет вызывать обработчик события, который случайным образом меняет значения SpinEdit и цвета кисти и пера (листинг 1.9).

Листинг 1.9. Событие, контролируемое таймером

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  SpinEditWx.Value:=Random(100);
  SpinEditWy.Value:=Random(100);
  SpinEditW1.Value:=Random(100);
  SpinEditW2.Value:=Random(100);
end;
```

В этой процедуре параметры линии и цвета меняются случайным образом.

Для кнопки SpeedButton4 назначим событие выключения таймера (листинг 1.10).

Листинг 1.10. Выключение таймера

```
procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
  Timer1.Enabled:=false;
end;
```

Введем еще одно изменение в проект. На полотно формы выведем изображение из графического файла Waterfal.BMP. Нам потребуется переменная Bitmap: TBitmap, которую инициализируем при запуске проекта в процедуре FormCreate(Sender).

```
Bitmap:=TBitmap.Create;
Bitmap.Transparent := true; //false;
Bitmap.TransparentMode := tmAuto; //tmFixed;
BitMap.LoadFromFile('Waterfal.bmp');
BitMap.TransParentColor := clWhite;
```

Вносим изменения: в этой же процедуре назначаем значение свойства прозрачности изображения `Bitmap.Transparent := true`. Это означает, что точки одного из цветов (в нашем случае `BitMap.TransParentColor := clWhite`) не будут выводиться. Методом `BitMap.LoadFromFile('Waterfal.bmp')` загрузим изображение из файла `Waterfal.BMP` в `BitMap`.

Осталось ввести последнее изменение: в процедуру рисования `DrawGraphic` добавить строчку

```
Canvas.StretchDraw(Rect(0,0,W,Height),BitMap); ,
```

после чего изображение из файла `Waterfal.BMP`, ранее загруженное в `BitMap` методом `LoadFromFile`, переносится в прямоугольник `Rect(0,0,W,Height)` полотна формы.

1.4. Мультимедийные ресурсы Windows

В Delphi включен компонент `TMediaPlayer`, позволяющий легко получить доступ к мультимедийным ресурсам Windows. Этот компонент может воспроизводить аудио- и видеофайлы в форматах WAV, MIDI и AVI. Для нормальной работы компонента `TMediaPlayer` необходимы специальные драйверы и апплеты, но в среде Windows они, обычно, инсталлируются автоматически.

Для создания приложения, реализующего доступ к мультимедийным ресурсам Windows, достаточно разместить на форме (рис. 1.2) компонент `TMediaPlayer` и создать обработчик события для нажатия кнопки, запускающей этот компонент (листинг 1.11).

Листинг 1.11. Воспроизведение аудио- и видеофайлов

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  MediaPlayer1.Close;
  if OpenDialog1.Execute then begin
    MediaPlayer1.FileName:=OpenDialog1.FileName;
    MediaPlayer1.Display := Panel1;
    MediaPlayer1.Open;
    MediaPlayer1.Play;
  end;
end;
```

Если работа метода `OpenDialog1` заканчивается выбором AVI-файла, то начинается воспроизведение этого файла.

Воспроизведение звуковых файлов возможно, естественно, при наличии звуковых колонок и звуковой карты.

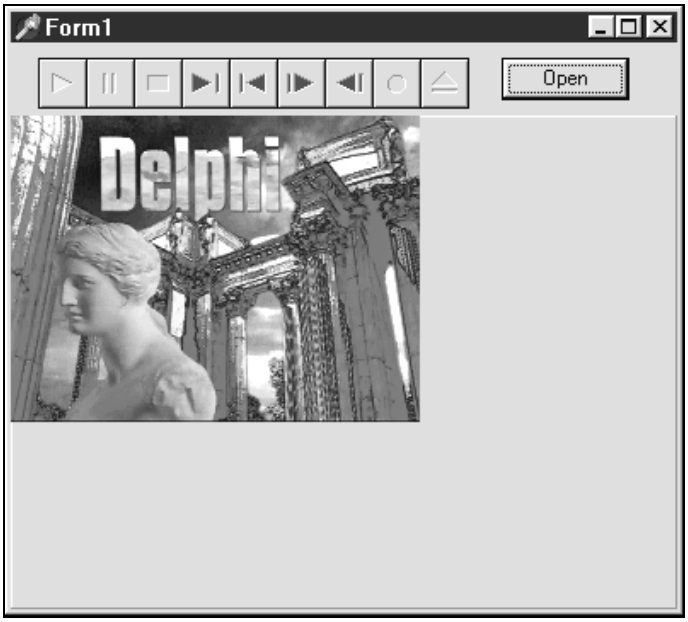

Рис. 1.2. Воспроизведение AVI-файла

На этом мы заканчиваем краткий обзор графических возможностей и переходим к более последовательному изучению графических структур Delphi.

Глава 2

Модуль *Graphics* и специальные приемы рисования

2.1. Структура классов

Основным источником информации о графических классах Delphi, их методах и свойствах является справочная система Delphi. Однако часть информации можно найти, например, в книгах А. Я. Архангельского [60], П. Дарахвелидзе [70].

В приложениях, разработанных в среде Delphi, рисование возможно только на компонентах, обладающих свойством `Canvas` (холстом, канвой, полотном). В частности, этим свойством обладают компоненты `TForm`, `TStringGrid`, `TImage`, `TPrinter`. Родительским для класса `TCanvas` и для всех остальных графических классов является класс `TPersistent`.

В листинге 2.1 представлена структура классов и их основных свойств, используемых при рисовании. Потомками класса `TPersistent` являются:

- класс `TGraphicsObject`, порождающий (инкапсулирующий) классы инструментов;
- класс `TCanvas`, содержащий инструменты и методы рисования;
- класс `TGraphic`, порождающий четыре класса изображений со своим форматом файлов (`TBitmap`, `TIcon`, `Tmetafile`, `TJPEGImage`);
- класс `TPicture`, надстройка над `TGraphic`, точнее, над его потомками. Он содержит поле `Graphic`, которое может содержать `TBitmap`, `Ticon`, `TMetafile` или `TJPEGImage`.

Листинг 2.1 достаточно велик, но мы решили включить его в настоящий раздел для того, чтобы более отчетливо отобразить иерархию классов и перечислить основные свойства объектов.

Листинг 2.1. Структура графических классов *TPersistent*

```
TGraphicsObject
    Tfont                    // класс шрифта
    PixelsPerInch: Integer   // число точек на дюйм
```

```
  Color: TColor           // цвет шрифта
  Height: Integer         // высота шрифта
  Name: TFontName         // имя шрифта
  Pitch: TfontPitch       // ширина символов шрифта
  Size: Integer           // размер
  Style: TFontStyles      // стиль
TPen                      // класс карандаша
  Color: TColor           // цвет пера
  Mode: TPenMode          // режим рисования линии
  Style: TPenStyle        // стиль линии
  Width: Integer          // толщина линии
TBrush                    // класс кисти
  Bitmap: TBitmap         // Bitmap для кисти
  Handle: HBrush          // указатель кисти
  Color: TColor           // цвет кисти
  Style: TBrushStyle      // стиль заливки
TCanvas                   // класс канвы
  ClipRect: TRect         // область отсечения канвы
  Handle: HDC             // указатель канвы
  PenPos: TPoint          // позиция указателя пера
  Pixels[X, Y: Integer]: TColor // массив пикселов канвы
  OnChange: TnotifyEvent  // событие после изменения
  OnChanging: TNotifyEvent// событие перед изменением
  Brush: TBrush           // свойство кисть
  CopyMode: TCopyMode     // режим копирования
  Font: TFont             // свойство шрифт
  Pen: TPen               // свойство перо
TControlCanvas            // родительский класс для канвы
  TBitmapCanvas           // канва класса TBitmap
  TPrinterCanvas          // канва принтера
  TMetafileCanvas         // канва метафайла
TGraphic
  TMetafile               // класс метафайлов
    CreatedBy: String     // имя автора
    Description: String   // описание файла
    Enhanced: Boolean     // способ сохранения на диске
    Handle: HENHMETAFILE  // указатель на экземпляр
    MMWidth: Integer      // ширина в сотых долях миллиметров
    MMHeight: Integer     // высота в сотых долях миллиметров
    Inch: Word            // число единиц на дюйм
  TBitmap                 // класс, содержащий изображение
    Canvas: TCanvas       // свойство полотно
    Handle: HBITMAP       // указатель на экземпляр
    HandleType: TBitmapHandleType // признак аппаратной зависимости
    Height                // высота в пикселах
```

```
    IgnorePalette: Boolean// игнорировать палитру
    MaskHandle: HBITMAP   // указатель на битовый массив GDI
    Monochrome: Boolean   // монохромность
    Palette               // палитра
    PixelFormat           // формат пикселов
    ScanLine[Row: Integer]: Pointer // указатель к линиям пикселов
    TransparentColor: TColor // прозрачный цвет
    TransparentMode: TTransparentMode // режим прозрачности
    Width                 // ширина канвы
    Modified              // модифицированность
    PaletteModified       // модифицированность палитры
    Transparent           // прозрачность
  TJPEGImage              // класс для файлов формата JPG
    CompressionQuality  // отношение между качеством и размером файла
    Empty                 // содержит ли JPG-графику
    GrayScale             // оттенки серого цвета
    Height                // высота в пикселах
    Palette               // палитра
    Performance           // отношение между качеством и скоростью
                          // декомпрессии
    PixelFormat: TPixelFormat // формат пиксела
    ProgressiveDisplay    // коэффициент уменьшения изображения
    ProgressiveEncoding   // показывает, уменьшенное ли изображение
    Scale                 // определяет размер изображения
    Smoothing             // сглаживание краев
    Width                 // ширина
    Modified              // модифицированность
    PaletteModified       // модифицированность палитры
    Transparent           // прозрачность
  TIcon                   // класс для файлов формата ICO
  Handle: HICON           // указатель на экземпляр класса
  TPicture                // класс изображений
    Bitmap: TBitmap       // свойство TBitmap
    Graphic: TGraphic     // свойство TGraphic
    PictureAdapter: IChangeNotifier // интерфейс OLE
    Height: Integer       // высота
    Icon: TIcon           // свойство типа TIco
    Metafile: TMetafile   // свойство типа метафайл
    Width: Integer        // ширина
    OnChange: TNotifyEvent// событие после изменения
    OnProgress: TprogressEvent // событие в процессе изменения
```

Для рисования используются методы класса `TCanvas` и три класса инструментов: `TFont` (шрифты), `TPen` (карандаш, перо), `TBrush` (кисть).

Абстрактный класс TGraphic является родительским для четырех видов изображений, соответствующих четырем форматам графических файлов: пиктограммы — формат ICO (класс TIcon), метафайлы — форматы EMF, WMF (класс TMetaflle), растровые изображения — формат BMP (класс TBitmap) и сжатые фотоизображения — формат JPG (класс TJPEGImage). Из этих четырех графических классов только TBitmap обладает опубликованным свойством TCanvas.

Эти классы, постоянные и некоторые функции описаны в модуле Graphics.

2.2. Цвет

Класс цвета точки TColor определен как длинное целое longint:

```
Tcolor = -$80000000..$7FFFFFFF.
```

Четыре байта переменных этого типа содержат информацию о долях синего B, зеленого G и красного R цветов и устроены следующим образом: $00BBGGRR, т. е. цвет можно вычислить через доли R, G, B по формуле:

$$Color = 256^2 \times B + 256 \times G + R,$$

или, используя битовую операцию "сдвиг налево" shl,

$$Color = B\ shl\ 16 + G\ shl\ 8 + R.$$

В модуле Graphics приведены некоторые константы для цветов (листинг 2.2).

Листинг 2.2. Предопределенные константы для цветов

```
Const
       clBlack = TColor($000000);
       clMaroon = TColor($000080);
       clGreen = TColor($008000);
       clOlive = TColor($008080);
       clNavy = TColor($800000);
       clPurple = TColor($800080);
       clTeal = TColor($808000);
       clGray = TColor($808080);
       clSilver = TColor($C0C0C0);
       clRed = TColor($0000FF);
       clLime = TColor($00FF00);
       clYellow = TColor($00FFFF);
       clBlue = TColor($FF0000);
       clFuchsia = TColor($FF00FF);
       clAqua = TColor($FFFF00);
       clLtGray = TColor($C0C0C0);
```