Государственный комитет Российской Федерации
по высшему образованию

Самарский государственный аэрокосмический университет
имени академика С. П. Королева

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ГРАФИКЕ В ТУРБО ПАСКАЛЕ

Методические указания к лабораторным работам

Самара 1996

Составители: Е. А. Логвинова, В. В. Семенов

УДК 681.3.06

Лабораторный практикум по графике в Турбо Паскале: Метод. указания к лабораторным работам / Самар. гос. аэрокосм. ун-т; Сост. Е. А. Логвинова, В. В. Семенов. Самара, 1996. - 27 с.

Содержатся рекомендации по выполнению лабораторных работ на популярном алгоритмическом языке программирования Турбо Паскаль с использованием графики. Рассматриваются различные вопросы построения статических и динамических изображений - простейшая компьютерная графика, анимация, построение графиков функций и гистограмм. Даны примеры программ и многочисленные варианты индивидуальных заданий к лабораторным работам по каждой теме. Предназначены для выполнения лабораторных работ по дисциплинам "ЭВМ и программирование", "Вычислительная техника и программирование", "Информатика" для студентов всех специальностей. Составлены на кафедре "Программное обеспечение вычислительных систем".

Печатаются по решению редакционно-издательского совета Самарского государственного аэрокосмического университета им. академика С. П. Королева

Рецензент А. О. Новиков

1. ПОСТРОЕНИЕ ПРОСТЕЙШИХ ИЗОБРАЖЕНИЙ

1.1. Инициализация и закрытие графического режима

Для инициализации графического режима используется функция InitGraph, которая определяет тип адаптера и подключает соответствующий драйвер, который и устанавливает графический режим:

```
InitGraph ( var GraphDriver: Integer;     { тип адаптера }
            var GraphMode: Integer;       { режим графики }
            DriverPath: String );         { путь к драйверу }
```

Если **GraphDriver = Detect** (const = 0) , то система включает режим автоопределения.

Процедура **CloseGraph** - завершает работу в графическом режиме и осуществляет переход в текстовый режим.

1.2. Процедуры и функции управления цветом

Процедура **SetBkColor (Color: Word)** задает цвет фона экрана.
Процедура **SetColor (Color: Word)** устанавливает текущий цвет.
В табл. 1 приведены коды цветов для процедур управления цветом.

Таблица 1

Коды цветов для процедур управления цветом

Имя	Значение	Назначение
Black	0	Черный
Blue	1	Синий
Green	2	Зеленый
Cyan	3	Голубой
Red	4	Красный
Magenta	5	Фиолетовый
Brown	6	Коричневый
LightGray	7	Светло-серый

Имя	Значение	Назначение
DarkGray	8	Темно-серый
LightBlue	9	Светло-синий
LightGreen	10	Светло-зеленый
LightCyan	11	Светло-голубой
LightRed	12	Розовый
LightMagenta	13	Светло-фиолетовый
Yellow	14	Желтый
White	15	Белый

1.3. Процедуры работы с линиями

В Турбо Паскале можно управлять стилем линий: задавать толщину и тип. Для этого определены следующие типы и константы стилей изображаемых линий :

```
LineSettingsType = record
                     LineStyle: Word; { стиль(тип) }
                     Pattern:   Word; { шаблон }
                     Thickness: Word; { толщина }
                   end;
```

Процедура SetLineStyle (LineStile: Word; Pattern: Word; Thickness: Word)
задает текущие параметры линии (стиль, шаблон и толщину):

LineStyle - стиль линии (см. табл. 2);

Pattern - шаблон линии - задается только в случае, если **LineStyle = UserBitLn**; представляется в виде двухбайтового числа, каждый бит которого равен единице, если очередной пиксель следует высветить, или равен нулю - в противном случае;

Thickness - толщина линии (см. табл. 3).

Процедура SetFillStyle (Pattern: Word; Color: Word)
задает стандартный орнамент и цвет заполнения фигур:

Pattern - определяет вид шаблона заливки (см. табл. 4);

Color - его цвет.

Процедура SetWriteMode (Mode: Integer)
устанавливает режим вычерчивания линий:

Mode = 0 - линия пересекает существующую картинку;

Mode = 1 - линия не проходит по местам, "занятым" существующими пикселями.

Таблица 2

Коды типов линий для процедур **SetLineStyle, GetLineSettings**

Имя	Значение	Назначение
SolidLn	0	Сплошная
DottedLn	1	Пунктирная
CenterLn	2	Штрихпунктирная
DashedLn	3	Штриховая
UserBitLn	4	Заданная пользователем

Таблица 3

Толщины линий для процедур **SetLineStyle, GetLineSettings**

Имя	Значение	Назначение
NormWidth	1	Нормальная
ThickWidth	3	Толстая

Таблица 4

Константы орнамента заполнения для процедур **SetFillStyle, GetFillSettings**

Имя	Значение	Назначение
EmptyFill	0	Заполнение цветом фона
SolidFill	1	Однородное заполнение цветом
LineFill	2	Заполнение ---
LtSlashFill	3	Заполнение ///
SlashFill	4	Заполнение /// толстыми линиями
BkSlashFill	5	Заполнение \\\ толстыми линиями
LtBkSlashFill	6	Заполнение \\\
HatchFill	7	Заполнение клеткой
XHatchFill	8	Заполнение косой клеткой
InterleaveFill	9	Заполнение частой сеткой
WideDotFill	10	Заполнение редкими точками
CloseDotFill	11	Заполнение частыми точками
UserFill	12	Определяется пользователем

1.4. Построение изображений

Процедура **Line (X1, Y1, X2, Y2: Integer)**
- процедура вывода линии (отрезка) на экран:
  (X1, Y1) - координаты начала линии (отрезка);
  (X2, Y2) - координаты конца линии (отрезка).

Процедура **Circle (X, Y: Integer; Radius: Word)**
- процедура изображения окружности:
  (X, Y) - координаты центра окружности;
  Radius - ее радиус.

Если коэффициент сжатия изображения будет соответствовать принятому BGI драйвером, то результатом работы будет окружность, иначе эллипс.

Процедура **Arc (X, Y: Integer; StartAngle, EndAngle, Radius: Word)**
- рисование дуги радиуса **Radius** из центра с координатами (**X, Y**) от угла **StartAngle** до **EndAngle**.

Процедура **Ellipse (X, Y: Integer; StartAngle, EndAngle, XRadius, YRadius: Word)**
- рисование эллиптической дуги с аналогичными параметрами:
XRadius и **YRadius** - размер горизонтальной и вертикальной полуосей соответственно.

Оси эллипса могут быть только параллельны осям X и Y. Для изображения полного эллипса надо задавать углы 0 и 360 градусов. Значение коэффициента сжатия изображения не влияет на его вид.

Процедура **Rectangle (X1, Y1, X2, Y2: Integer)**
- построение прямоугольника текущим цветом:
(X1, Y1), (X2, Y2) - координаты диагонали ($0 \le X1 < X2 \le GetMaxX$ и $0 \le Y1 < Y2 \le GetMaxY$).

Процедура **Bar (X1, Y1, X2, Y2: Integer)**
строит прямоугольник, закрашенный текущим орнаментом и цветом заполнения (орнамент и цвет заполнения задаются процедурой **SetFillStyle**):
(X1, Y1), (X2, Y2) - координаты левого верхнего и правого нижнего углов прямоугольника.

Процедура **Bar3D (X1, Y1, X2, Y2: Integer; Depth: Word; Top: Boolean)**
строит параллелепипед, закрашенный текущим орнаментом и цветом заполнения (орнамент и цвет заполнения задаются процедурой **SetFillStyle**):
(X1, Y1), (X2, Y2) - координаты левого верхнего и правого нижнего углов передней грани;
Depth - ширина боковой грани (отсчитывается по горизонтали);
Top - признак включения верхней грани. Если **Top = True**, то верхняя грань вычерчивается, в противном случае верхняя грань не отображается (см. табл. 5). Это позволяет рисовать несколько параллелепипедов, расположенных один на другом.

Таблица 5

Константы изображения верхней грани для Bar3D

Имя	Значение	Назначение
TopOn	**True**	Закрашивается верхняя грань
TopOff	**False**	Не закрашивается верхняя грань

Процедура **DrawPoly (NumPoints: Word; var PolyPoints)**

строит контур многоугольника линией с заданными параметрами и цветом (текущий цвет задается процедурой **SetColor**, текущие параметры линии - процедурой **SetLineStyle**):

NumPoints - число вершин многоугольника;

PolyPoints - переменная без типа, содержащая **NumPoints** + 1 пар координат вершин прямоугольника (координаты должны быть целого типа, перечисляться в той же последовательности, как они идут по контуру, причем первая вершина должна быть повторена и в конце перечисления).

Процедура **FillPoly (NumPoints: Word; var PolyPoints)**

строит многоугольник, закрашенный текущим орнаментом и цветом заполнения. Параметры определяются так же, как и в **DrawPoly**.

Процедура **FillEllipse (X, Y: Integer; XRadius, YRadius: Word)**

строит эллипс, закрашенный текущим орнаментом и цветом заполнения:

(X, Y) - координаты центра эллипса;

XRadius, YRadius - горизонтальная и вертикальная полуоси эллипса.

Процедура **Sector (X, Y: Integer; StartAngle, EndAngle, XRadius, YRadius: Word)**

строит сектор эллипса, закрашенный текущим орнаментом и цветом заполнения:

(X, Y) - координаты центра сектора эллипса;

StartAngle, EndAngle - начальный и конечный углы эллипса, отсчитываемые против часовой стрелки от горизонтальной оси, направленной вправо;

XRadius, YRadius - горизонтальная и вертикальная полуоси эллипса.

Процедура **PutPixel (X, Y: Integer; Pixel: Word)**

закрашивает пиксель с заданными координатами:

(X, Y) - координаты пикселя;

Pixel - цвет, в который закрашивается пиксель.

Процедура **LineTo (X, Y: Integer)**

проводит линию текущего цвета из текущей позиции в точку с заданными координатами:

(X, Y) - координаты точки, до которой проводится линия.

Процедура **LineRel (Dx, Dy: Integer)**

проводит линию текущего цвета из текущей позиции в точку, сдвинутую относительно исходной позиции на величины **Dx** и **Dy**:

Dx и Dy - смещение координат точки конца линии по отношению к исходным координатам.

Процедура **MoveRel (Dx, Dy: Integer)**
перемещает указатель координат на заданное расстояние **Dx, Dy** по отношению к его предыдущему положению (точка на экране не высвечивается):

Dx, Dy - смещение текущего указателя относительно его старого положения.

Процедура **PieSlice (X, Y: Integer; StartAngle, EndAngle, Radius: Word)**
вычерчивает сектор круга с заполнением:

(X, Y) - координаты центра сектора круга;

StartAngle, EndAngle - начальный и конечный углы сектора, отсчитываемые против часовой стрелки от горизонтальной оси, направленной вправо;

Radius - радиус сектора.

1.5. Варианты заданий

1. Построить треугольник, положение вершин которого на экране определяется парами чисел (50, 50), (100, 50) и (80, 190).
2. Построить четырехугольник, положение вершин которого на экране определяется парами чисел (70, 80), (170, 70), (170, 150) и (80, 140).
3. Построить шестиугольник, положение вершин которого на экране определяется парами чисел (120, 100), (140, 120), (140, 140), (120, 160), (100, 140) и (100, 120).
4. Построить квадрат со стороной 30 (число точек), центр которого совмещен с центром экрана. Стороны квадрата должны быть параллельны сторонам экрана.
5. Построить прямоугольник со сторонами 30 и 50, центр которого совмещен с центром экрана. Стороны прямоугольника должны быть параллельны сторонам экрана.
6. Дано шесть целых чисел X_1, Y_1, X_2, Y_2, X_3, Y_3. Каждая пара X_i, Y_i (i=1, 2, 3) определяет положение одной из вершин треугольника на экране. Если данные числа определяют прямоугольный треугольник, то построить его на экране, в противном случае вывести сообщение: "Треугольник не прямоугольный".
7. Дано шесть целых чисел, определяющих положение вершин треугольника, расположенного в левой половине экрана. Построить на экране этот треугольник, а также треугольник, симметричный данному относительно прямой, проходящей через середину экрана.

8. Дано три целых числа, определяющих положение центра окружности на экране и ее радиус. Если окружность не пересекает горизонтальную прямую, проходящую через середину экрана, то построить данную окружность и окружность, симметричную данной относительно этой прямой.

9. Четыре целых числа задают положение концов отрезка на экране. Получить изображение этого отрезка и изображение отрезка, центрально-симметричного данному относительно точки, расположенной в центре экрана.

10. Получить рисунки, составленные из отрезков, окружностей и точек на экране (рис. 1):

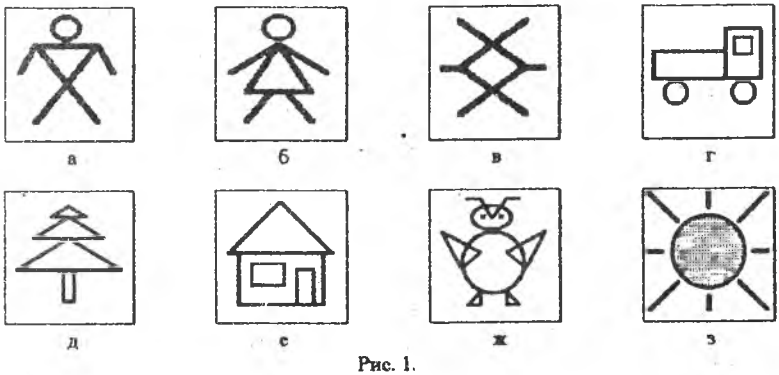


Рис. 1.

11. Получить в центре экрана изображение, состоящее из 10 вложенных квадратов со сторонами 10, 20, 30, ... , 100.

12. Вывести на экран два прямоугольника. Один заштриховать вертикальными прямыми, другой - горизонтальными.

13. Построить оси координат. Начало координат поместить вблизи левого нижнего угла экрана. Полуоси OX и OY разметить так, как показано на рис. 2:

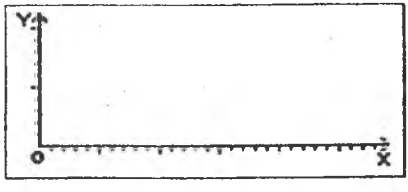


Рис. 2.

14. Построить 9 концентрических окружностей, окрашенных поочередно в зеленый, красный и коричневый цвета.
15. Построить 10 вложенных квадратов, окрашенных поочередно в зеленый и красный цвета.
16. Изобразить семицветную радугу, так чтобы ее самая верхняя дуга выходила из нижнего левого угла экрана. При рисовании радуги использовать процедуру построения закрашенного сектора **PieSlice**.
17. Изобразить на экране свое имя и фамилию.
18. Изобразить план футбольного поля, вид сверху - белые линии на зеленом фоне (рис. 3).
19. Построить фигуру, состоящую из трех параллелепипедов разного цвета, стоящих друг на друге (рис. 4):

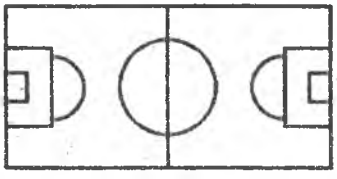


Рис. 3

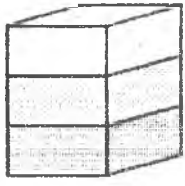


Рис. 4

20. Получить в центре экрана изображение, состоящее из девяти вложенных квадратов и раскрасить его тремя цветами.
21. Дано восемь натуральных чисел X_1, Y_1, l_1, h_1 и X_2, Y_2, l_2, h_2. Каждая четверка чисел задает положение на экране прямоугольника со сторонами, параллельными сторонам экрана. Значения X_i, Y_i определяют положение левого нижнего угла прямоугольника с номером i, l_i - длину основания, h_i - высоту ($i = 1, 2$). Построить прямоугольники и закрасить первый элемент зеленым цветом, второй - красным. Если прямоугольники пересекаются, то их общую часть закрасить коричневым цветом.
22. Получить на экране изображение красного полумесяца:
    а) построить две пересекающиеся окружности;
    б) закрасить полумесяц;
    в) удалить с экрана окружности.
23. Написать программу, выполняющую построение фигур, показанных на рис. 5 (фигуры закрасить любым цветом).
24. Даны натуральные числа X_1, Y_1, ... , X_{40}, Y_{40}. Построить на экране точки, задаваемые парами значений X_i, Y_i ($i = 1, 2, \ldots, 40$), и

прямоугольник со сторонами, параллельными сторонам экрана, содержащий все эти точки.

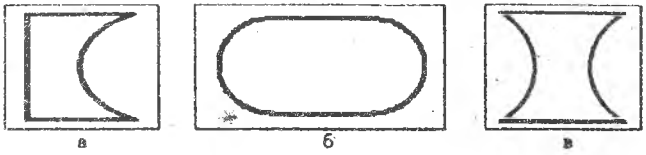


Рис. 5

25. Даны натуральные числа X_1, Y_1, ... , X_{20}, Y_{20}. Построить на экране точки, задаваемые парами значений X_i, Y_i (i = 1, 2, ... , 20) , и соединить отрезками прямых:
    а) каждую из точек со всеми остальными;
    б) точки с номерами одной четности;
    в) точки с номерами разной четности.
26. Даны натуральные числа X_1, Y_1, ... , X_{20}, Y_{20}. Построить на экране точки, задаваемые парами значений X_i, Y_i (i = 1, 2, ... , 20) , и соединить отрезками прямых пары точек, наиболее удаленные друг от друга.

2. ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ

2.1. Приемы построения графиков функций. Техника программирования

При построении графиков функций следует учитывать следующие основные моменты:

- выбор расположения системы координат;
- определение масштабного множителя;
- исследование области определения функции;
- переход от мировой системы координат к экранной.

Пусть, например, требуется построить график функции $Y = 2x^2 - 0{,}5$ на отрезке [-2; 2]. Кроме кривой, изображающей график функции, на экране должны быть высвечены координатные оси OX и OY.

Расположим оси OX и OY в середине экрана, т.е. в точке, определяемой парой чисел (320, 240), примем масштабный множитель равным 60. Перейдем от мировой системы координат к экранной. Учитывая, что левый верхний пиксель экрана имеет координаты (0, 0) и координата X увеличивается от нуля по горизонтали, координата Y увеличивается по вертикали сверху вниз, положение точки графика с координатами (X, Y) будет определяться парой

значений (320+60·X, 240-60·Y). Приведем пример фрагмента программы построения графика функции $Y = 2x^2 - 0,5$ с использованием процедуры **PutPixel**:

```
program graf;
...
      x := -2;
      while x <= 2 do
            begin
                  x := x + 0.01;          { Приращение ΔX = 0.01 }
                  y := 2*sqr(x) - 0.5;    { Вычисление функции }
                  A := Round (320+60*x);
                  B := Round ( 240-60*y );
                  PutPixel ( A, B, 4 ); { Вывод красной точки на экран }
            end; { Конец цикла }
End. { Конец программы }
```

Если увеличить приращение аргумента, например: x:=x+0.05 , то график будет построен примерно вдвое быстрее, но между соседними точками графика будут заметны просветы. Время построения графика можно уменьшить, не ухудшая качества изображения. Для этого нужно увеличить приращение аргумента и построить ломаную линию, состоящую из отрезков, соединяющих соседние точки графика. Фрагмент программы будет выглядеть следующим образом:

```
program fgraf;
...
      x := -2;
      y := 2*sqr(x) - 0.5;    { Вычисление функции }
      An := Round (320+60*x);
      Bn := Round ( 240-60*y );
      while x <= 2 do
            begin
                  x := x + 0.05;          { Приращение ΔX=0.05 }
                  y := 2*sqr(x) - 0.5;    { Вычисление функции }
                  Ak := Round (320+60*x);
                  Bk := Round ( 240-60*y );
                  Line ( An, Bn, Ak, Bk );
                  An := Ak;  Bn := Bk;
            end; { Конец цикла }
End. { Конец программы }
```

2.2. *Варианты заданий*

1-6. На заданном отрезке, выбрав расположение координатных осей на экране и масштаб, построить графики функций:

№ варианта	Функция	Отрезок
1	$y = \sin(x)$	$[-\pi;\ \pi]$
2	$y = \cos(x)$	$[-3\frac{\pi}{2};\ \frac{\pi}{2}]$
3	$y = tg(x)$	$[-\frac{\pi}{2};\ \frac{\pi}{2}]$
4	$y = ctg(x)$	$[0;\ \pi]$
5	$y = 3 \cdot \cos(2x+4)$	$[-\pi;\ \pi]$
6	$y = 1{,}5 \cdot \cos(\frac{2}{3} \cdot x - \frac{\pi}{9})$	$[-\pi;\ \pi]$

7-15. Исследовав область определения и выбрав расположение координатных осей на экране и масштаб, построить графики функций:

№ варианта	Функция
1	$y = \frac{1}{x} + 1$
2	$y = \frac{x+3}{x-2}$
3	$y = 1 + \frac{2}{x} + \frac{3}{x^2}$
4	$y = 3 - \frac{2}{x} - \frac{1}{x^2}$
5	$y = \frac{1}{3x^2 + 2x + 1}$
6	$y = \frac{\alpha^3}{\alpha^2 + x^2}$, $\alpha = 25$ Локон Аньези
7	$y = \frac{1}{x^2 + 2x + 1}$
8	$y = \alpha^x$ (при $\alpha = \frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \frac{1}{10}$)
9	$y = \alpha_0 x^4 + \alpha_1 x^3 + \alpha_2 x^2 + \alpha_3 x + \alpha_4$ $-1 < \alpha < 1;\ \alpha \neq 0$

16. Дано 15 чисел $\alpha_1, b_1, c_1, a_2, b_2, c_2, \ldots, a_5, b_5, c_5$. Каждая тройка чисел a_i, b_i, c_i ($i = 1, \ldots, 5$) определяет параболу $y = \alpha_i x^2 + b_i x + c_i$.

Построить графики всех этих парабол на отрезке [-5; 5]. График параболы с номером i должен иметь цвет с номером i div 3+1.

17-22. Построить кривые по их уравнениям, заданным в полярных координатах. Соотношение между полярными и декартовыми координатами определяется следующим образом: $x = \rho \cdot \cos\varphi$, $y = \rho \cdot \sin\varphi$:

№ варианта	Функция
1	Циссоида Диоклеса $\rho = \frac{\alpha \cdot \sin^2\varphi}{\cos\varphi}$
2	Строфоида $\rho = -\frac{\alpha \cdot \cos 2\varphi}{\cos\varphi}$
3	Лемниската Бернулли $\rho = \alpha \cdot \sqrt{2 \cdot \cos 2\varphi}$
4	Кардиоида $\rho = \alpha \cdot (1 + \cos\varphi)$, α - диаметр круга
5	Улитка Паскаля $\rho = \alpha \cdot \cos\varphi + \varepsilon$; $\alpha < \varepsilon < 2\alpha$, α - диаметр круга
6	Спираль Архимеда $\rho = \alpha \cdot \varphi$, $0 < \varphi < 4\pi$

23. Построить график функции, заданной таблично. Таблица значений функции задана числами X_1, Y_1, X_2, Y_2, ... , X_{40}, Y_{40}, при этом $X_1 < X_2 < ... < X_{40}$.

24. Написать программу, позволяющую получать графики функций $Y = f(x)$ в любой части экрана, при следующих заданных величинах:
- положение левого верхнего угла;
- ширина области экрана;
- высота области экрана;
- отрезок, на котором строится график;
- количество разбиений отрезка при построении графика.

25. Таблица значений некоторой функции задана числами X_1, Y_1, X_2, Y_2, ... , X_{40}, Y_{40}, где $X_1 < X_2 < ... < X_{40}$. Получить коэффициенты линейной зависимости $Y = k \cdot X + b$ с помощью метода наименьших квадратов и высветить на экране точки X_i, Y_i (i= 1, ... , 40) и прямую $Y = k \cdot X + b$.

26. Таблица значений некоторой функции задана числами X_1, X_2, ... , X_{40}, Y_1, Y_2, ..., Y_{40}, при этом значения X_i (i= 1, ... ,40) не упорядочены. Упорядочить таблицу значений функции и построить её график.

27,28. Проиллюстрировать графически решение системы уравнений. Точки, которые являются решением системы уравнений, выделить цветом:

а) $\begin{cases} y = \frac{1}{2} \cdot x^2 \\ y = \frac{1}{2} \cdot x + 2 \end{cases}$; б) $\begin{cases} y = 4 \cdot x \\ y = 2^x \end{cases}$;

29,30. Написать программу построения графика функции f(x) так, чтобы участки графика, на которых f(x) > 0 , и участки, на которых f(x) < 0, окрашивались в разные цвета:

а) $y = \sin\frac{3x}{4}$; б) $y = \cos 2 \cdot x$.

3. СОЗДАНИЕ ЭЛЕМЕНТАРНОЙ АНИМАЦИИ

3.1. Методы и техника программирования элементарной анимации

Основной метод создания движущихся изображений прост - создать изображение предмета, уничтожить его и создать вновь, но с некоторым небольшим смещением. Для получения приемлемого качества анимации скорость этого процесса должна быть достаточно высокой. Рассмотрим один из самых простых примеров: изобразим движение шарика белого цвета по экрану. Пусть шарик появляется у левой границы экрана и перемещается в горизонтальном направлении.

Пример 1:

```
Program GR;
  Uses GRAPH, CRT;
  Var X, GrDriver, GrMode: Integer;
Begin
  GrDriver := Detect;
  InitGraph( GrDriver, GrMode, 'D:\TP\BGI' );
  SetColor( 1 );
  SetBkColor( 1 ); { Текущий цвет фона экрана - синий }
  x := 60; { Радиус окружности }
  While x <= 640 Do
    begin
      { Рисуем новое изображение }
      SetFillStyle( 1, 15 );          { Устанавливаем цвет заполнения - белый }
      FillEllipse( x,240,60,60);      {Строим окружность, закрашиваем }
      { Стираем построенное изображение }
      SetFillStyle( 0, 15 );      {Устанавливаем цвет заполнения - фоновый }
      FillEllipse( x,240,60,60);      {Строим окружность, закрашиваем }
      x := x + 5;          { Меняем положение шарика относительно оси X }
    end;   { Конец цикла }
End. { Конец программы }
```

В данной программе для уничтожения изображения шарика он перерисовывается с цветом фона. Затем рассчитывается новое положение центра шарика и он изображается с использованием белого цвета вновь.

В этом примере требовалось перемещение очень простого объекта, но если было бы необходимо реализовать движение более сложного объекта (например, изображение падающей снежинки) ? В этом случае затраты времени на перерисовку будут значительно большими и качество анимации значительно ухудшиться. Если же изображать движение еще более сложного объекта (например, идущей человеческой фигуры), то скорость анимации значительно снизится и может стать неприемлемой для получения качественного изображения. Поэтому при изображении движения объектов более сложной формы используется совершенно иной, более быстрый и более универсальный способ, который заключается в следующем: изображение объекта рисуется только один раз, после чего с помощью процедуры **GetImage** заносится в оперативную память компьютера. После этого для стирания предыдущего изображения объекта и перерисовки его в новом месте используется процедура **PutImage** с соответствующими параметрами.

Процедура **GetImage (X1, Y1, X2, Y2: Integer; var BitMap)**
сохраняет изображение заданного участка экрана в буфере:

(X1, Y1), (X2, Y2) - координаты левого верхнего и правого нижнего углов сохраняемого участка экрана;

BitMap - переменная без типа, она является буфером для сохранения информации. В этой переменной первое слово отводится для записи ширины сохраняемого участка, второе - высоты участка, третье - зарезервировано, остальная область отводится под изображение. Размер буфера не должен превышать 64 Кбайт.

Процедура **PutImage (X, Y: Integer; var BitMap; BitBlt: Word)**
выдает на экран образ изображения, записанный в буфере.

(X, Y) - координаты левого верхнего угла прямоугольного участка экрана;

BitBlt - параметр, указывающий способ объединения передаваемой на экран информации с уже имеющейся там (см. табл. 6).

Таблица 6

Константы битовых операций

Имя	Значение	Назначение
NormalPut	**0**	**MOV** - перемещение
XORPut	**1**	**XOR** - исключающее ИЛИ
OrPut	**2**	**OR** - логическое ИЛИ
AndPut	**3**	**AND** - логическое И
NotPut	**4**	**NOT** - инверсия

Теперь с помощью процедуры **PutImage** изображение может быть моментально помещено в нужное место экрана (параметр **BitBlt** должен иметь значение **OR**). Для удаления изображения с экрана можно использовать повторный вывод изображения с помощью процедуры **PutImage** в ту же область экрана, но параметр **BitBlt** должен иметь значение **XOR**.

Приведем пример программы, которая выполняет ту же задачу, что и программа примера 1, но с использованием процедур **GetImage** и **PutImage**.

Пример 2:

```
Program Imag;
  Uses GRAPH, CRT;
  Var P: Pointer;
    - X, GrDriver, GrMode, Size: Integer;
Begin
  GrDriver := Detect;
  InitGraph( GrDriver, GrMode, 'D:\TP\BGI' );
  SetBkColor( 1 ); { Текущий цвет фона экрана - синий }
  x := 60;                              { Радиус окружности }
  SetFillStyle( 1, 15 );
  FillEllipse( x, 240, 60, 60 );        { Рисуем окружность }
  Size := ImageSize( 0, 180, 120, 300 ); GetMem( p, Size );
  GetImage( 0, 180, 120, 300, p^ );
  x:=0;
  While x <= 640 Do
    begin
      PutImage( x, 180, p^, XORPut );
      x := x + 5;
      PutImage( x, 180, p^, ORPut );
    end;   { Конец цикла }
  Dispose( p );
End. { Конец программы }
```

До сих пор мы ограничивались рассмотрением тех случаев, когда изображение объекта в процессе его перемещения по полю экрана не изменяется. Однако, как быть в том случае, если изменяется не только местоположение, но и внешний вид движущихся объектов? При решении задач такого рода можно использовать несколько различных подходов. Если число различных изображений объекта невелико, то целесообразно заранее нарисовать все возможные варианты изображений объекта и считать каждое из них в оперативную память компьютера. После этого программе остается только определить, какой из вариантов изображения объекта в данный момент времени должен быть отображен и вывести этот вариант изображения на экран с помощью процедуры **PutImage**. Возможно, необходимо будет также

стереть предыдущее изображение объекта, используя метод, изложенный выше (пример 2).

В качестве примера, иллюстрирующего описанный алгоритм, попробуйте написать программу, изображающую (с некоторой долей условности, конечно) вращение барабана из игры "Поле чудес". Заметьте, что основная работа в программе производится на этапе подготовки различных вариантов изображений. Та часть программы, которая реализует собственно мультипликацию, очень компактна. Попробуйте реализовать подобный алгоритм "в лоб", перерисовывая на каждом шаге изображение барабана, и вы увидите, насколько медленнее будет изменяться изображение на экране.

Если число возможных вариантов изображения объекта велико (более 20), то алгоритмы формирования анимации могут быть весьма разнообразными и зависят от условий конкретной задачи и опыта разработчика программ. Можно дать следующие рекомендации, которыми, на наш взгляд, стоит руководствоваться при программировании анимации такого рода:

- убедитесь, что для формирования изображения нельзя использовать ни один из вышеизложенных способов. Иногда кажущиеся сложными и запутанными задачи на самом деле решаются просто и красиво;
- постарайтесь перерисовывать только изменяющиеся части;
- существует множество возможностей для формирования анимации, связанных с особенностями работы конкретных видеоадаптеров в графических режимах. Поскольку освещение данного вопроса выходит за рамки настоящей работы, отсылаем интересующихся к соответствующей литературе.

В заключение отметим, что в задачах, в которых направление движения в каждый момент времени выбирается случайным образом, следует использовать функцию случайных величин **Random**.

Функция **Random [(Range: Word)]** формирует случайное число целого или вещественного типа (перед обращением к функции ее целесообразно инициализировать, использовав процедуру **Randomize**). **Range** - параметр, указывающий диапазон значений случайного числа. Результат в этом случае имеет тип **Word** и изменяется в пределах $0 \le X < Range$. Если параметр **Range** не задан, то результат будет типа **Real** в пределах $0.0 \le X < 1.0$.

3.2. Варианты заданий

1. Написать программу, которая моделирует движение бильярдного шара (рис. 6) по столу от центра при отсутствии трения (задан угол α).
2. Смоделировать движение плаката (рис. 7).
3. Смоделировать (рис. 8) движение бильярдного шара по столу от края при отсутствии трения (задан угол α).

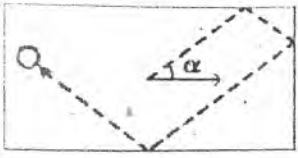


Рис. 6.

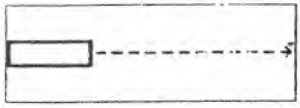


Рис. 7.

4. Смоделировать движение теннисного шарика (рис. 9) :

Рис. 8.

Рис. 9.

5. Смоделировать движение стрелок часов (рис. 10) .

6. Смоделировать полет волана для бадминтона (рис. 11) :

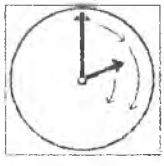


Рис. 10.

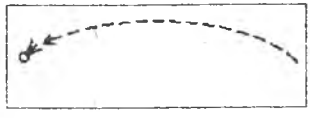


Рис. 11.

7. Смоделировать движение при перестановке элементов массива (рис.12):

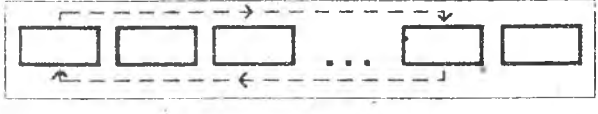


Рис. 12.

8. Смоделировать движение шарика по экспоненте (рис. 13) .

9. Смоделировать движение шарика по параболе (рис. 14) .

10. Движение шарика по синусоиде (рис. 15) .

11. Смоделировать перемещение "змеи" по плоской поверхности, ограниченной размерами экрана. "Змея" движется случайным образом.

12. Написать программу, моделирующую всплывание пузырька воздуха со дна емкости с водой на поверхность. Размеры и скорость перемещения

пузырька изменяются по линейному закону в зависимости от пройденного расстояния.

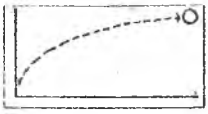


Рис. 13.

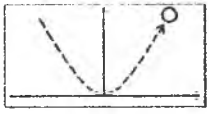


Рис. 14.

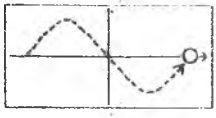


Рис. 15.

13. Нарисовать траекторию движения точки по спирали. Расстояние точки от центра спирали определяется по формуле R = k * j , где k - произвольный коэффициент от 1 до 10, j - угол, рад. Рисовать спираль до тех пор, пока она не выйдет за границы экрана.

14. Написать программу, в ходе выполнения которой круг зеленого цвета, появившись в центре экрана и постоянно расширяясь, увеличивается в размерах в три раза, а затем сжимается до начальных размеров.

15. Пропеллер состоит из двух закрашенных треугольников. Получить на экране вращающийся пропеллер.

16. Дано два натуральных числа. Написать программу, в ходе выполнения которой отрезок, появившись в левом верхнем углу экрана, передвинется по экрану так, что его левый конец совместится с точкой, положение которой определяется данными числами. Весь путь отрезка должен состоять из двух участков - горизонтального и вертикального.

17. Написать программу построения изображения колобка так, чтобы после появления изображения на экране точки, высвечиваемые внутри малых окружностей, перемещались одновременно несколько раз влево - вправо (т. е. , чтобы колобок двигал глазами, рис. 16) .

18. Написать программу, в ходе выполнения которой зеленый квадрат, появившись в левом верхнем углу экрана, перемещается вправо и вниз по диагонали.

19. Написать программу, которая моделирует движение по звездному небу.

20. Написать программу, в ходе выполнения которой на экране переливалась семицветная радуга.

21. Написать программу, отображающую на экране текущее время в цифровом виде в формате - ***часы : минуты : секунды***. Показания часов должны изменяться в соответствии с реальным временем. Размер символов (цифр) должен быть таким, чтобы изображение занимало не менее 50% площади экрана.

22. Написать программу, которая моделирует светофор - поочередная смена красного, желтого и зеленого цветов на фигуре (рис. 17) .

Рис. 16.

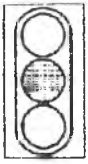


Рис. 17.

23. Написать программу, изображающую зевающего студента (схематично). Использовать процедуру Ellipse. Можно добавить уши и встающие дыбом волосы (рис. 18):

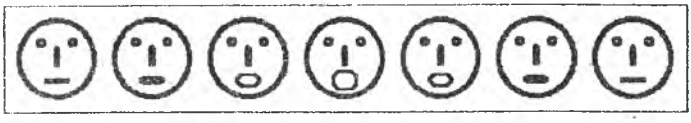


Рис. 18.

24. Написать программу, моделирующую движение кривошипно-шатунного механизма. Рычаг AB вращается вокруг оси A, вследствие чего поршень C перемещается вперед-назад вдоль оси X (рис. 19):

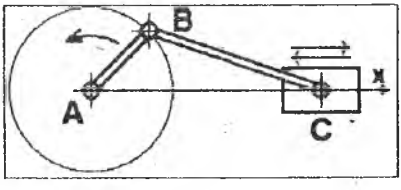


Рис. 19.

25. Написать программу, изображающую ползущую гусеницу (рис. 20). "Гусеница" должна перемещаться вправо до тех пор, пока не доползет до границы экрана:

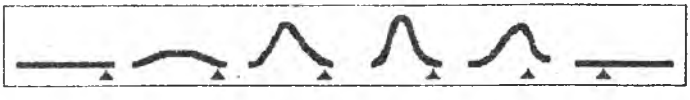


Рис. 20.

26. Написать программу, имитирующую работу песочных часов. Время устанавливается пользователем с экрана (рис. 21).

27. Написать программу, имитирующую качающийся маятник.

28. Написать программу, имитирующую падающие с неба снежинки.

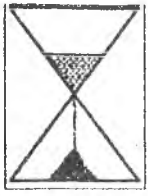


Рис. 21.

29. Написать программу, имитирующую индикатор динамического изменения частотных составляющих звукового сигнала панели магнитофона (частотные составляющие изобразить в виде столбчатой диаграммы).

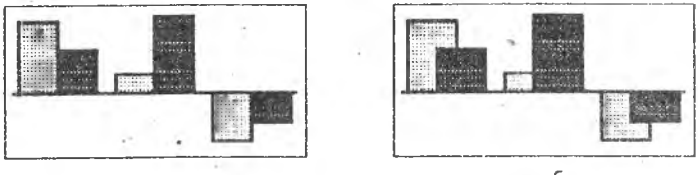
4. ПОСТРОЕНИЕ ДИАГРАММ

4.1. Типы диаграмм

Существует большое количество различных типов диаграмм, т. е. изображений, иллюстрирующих в наглядном графическом виде зависимости между двумя и более непрерывными и дискретными величинами.

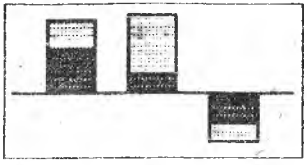
Диаграмма может быть плоская: график, линейчатая, круговая, кольцевая, гистограмма, диаграмма с областями, радар, XY-точечная и объемная: график, линейчатая, круговая, гистограмма, с областями, поверхность.

Построение плоских графиков непрерывных функций было рассмотрено в разд. 2. В настоящем разделе рассмотрим построение гистограмм и круговых диаграмм. На рис. 22 изображены следующие виды плоских диаграмм: а - простая гистограмма; б - гистограмма с перекрытием; в - совмещенная гистограмма; г - круговая диаграмма:

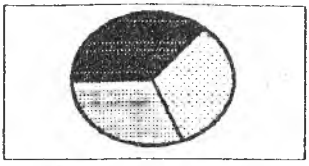


а б

Рис. 22.



в



г

Рис. 22.

4.2. Техника программирования

Перед разработкой программы отображения данных на экране следует учесть следующие моменты:

- перевод численных значений исходных данных в соответствующие координаты экрана;
- расположение диаграммы на экране.

Диаграмма выглядит более привлекательной и наглядной, если она занимает практически всю площадь экрана.

Допустим, что диаграмма будет изображаться в виде вертикальных прямоугольников, т. е. в виде плоской гистограммы. Исходные данные должны быть нормализованы таким образом, чтобы они принимали значения в диапазоне от 0 до Ymax . Для этого исходные данные умножают на коэффициент $k = \frac{Y\,max}{max - min}$, где min и max - соответственно минимальное и максимальное значения отображаемого параметра.

При вычислении нормирующего коэффициента k можно использовать меньшее число Ymax, что позволит использовать свободные строки экрана для вывода пояснительной информации. Для наглядности к построенной гистограмме можно добавить горизонтальную ось.

Для эффективного использования площади экрана по горизонтали следует соответствующим образом подобрать ширину выводимых элементов (ширину экрана Xmax разделить на количество элементов).

В качестве примера рассмотрим фрагмент программы, рисующей объемную простую гистограмму. Высота элементов определяется случайным образом.

Пример.

```
...
Const L = 9;                          { Количество параллелепипедов }
...
    m := Round ( (GetMaxX)/L );       { Ширина + глубина параллелепипеда }
    d := m div 4;                     { Глубина параллелепипеда }
```

```
For i := 1 to L do
  begin
    y := GetMaxY - Random ( GetMaxY - d ); { d <= y <= GetMaxY }
    { Закрасить переднюю грань светло-голубой клеткой }
    SetFillStyle ( LightCyan );
    Bar3D ( (i-1)*m, y, i*m-d, HatchFill, GetMaxY, d, True );
    { Закрасить верхнюю грань сплошным голубым цветом }
    SetFillStyle ( SolidFill, Cyan );
    FloodFill ( i*m-d , y-1, White );
    { Закрасить боковую грань частой голубой сеткой }
    SetFillStyle ( InterLeaveFill, Cyan );
    FloodFill ( i*m-d+1, y, White );
  end;
...
```

4.3. Варианты заданий

1-6. Дано 30 целых чисел X_1, X_2, ... , X_{15}, Z_1, Z_2, ... , Z_{15} . Рассматривая их как количества изделий, выпущенных двумя предприятиями за 15 дней, построить следующие типы гистограмм, отражающих сравнительный выпуск изделий по дням:

а) плоская простая гистограмма;
б) плоская гистограмма с перекрытием;
в) плоская совмещенная гистограмма;
г) объемная простая гистограмма;
д) объемная гистограмма с перекрытием;
е) объемная совмещенная гистограмма.

7-8. Информацию, содержащуюся в сообщении об успеваемости учащихся в группе: "... всего N учащихся, из них M хорошистов и отличников ...", графически можно представить в виде прямоугольника, разделенного по вертикали на две части, причем нижняя часть относится к верхней части, как N к M. Дано 20 натуральных чисел N_1, M_1, N_2, M_2, ... , N_{10}, M_{10} , взятых из сообщения об успеваемости в 10 группах. Построить гистограммы, отражающие эти данные в виде:

а) плоской совмещенной гистограммы;
б) объемной совмещенной гистограммы.

9. Написать программу построения гистограммы, содержащей 12 элементов, соответствующих как положительным, так и отрицательным значениям, формируемым функцией случайных величин **Random**. Отрицательные значения изображаются прямоугольниками, расположенными ниже горизонтальной оси.

10-13. Написать программу построения диаграммы с областями, имеющую следующий вид (рис. 23) :

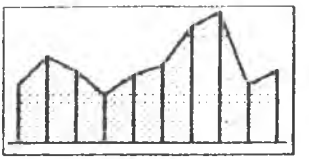


Рис. 23.

Для вычисления исходных данных использовать функцию случайных величин **Random**. Количество элементов диаграммы должно задаваться константой, определяемой в программе:

а) const = 4;
б) const = 6;
в) const = 8;
г) const = 20.

14-17. Нарисовать диаграмму, иллюстрирующую количество осадков, выпавших в Самарской области в течение каждого месяца 1995 года. Построить диаграмму в виде:

а) плоской простой гистограммы;
б) объемной простой гистограммы;
в) плоской круговой диаграммы;
г) объемной круговой диаграммы.

18-21. Нарисовать диаграмму, иллюстрирующую качественный состав компьютерного парка СГАУ, который характеризуется процентным отношением количества компьютеров с тем или иным типом микропроцессора (80486, 80386, 80286, 8086) к общему количеству всех компьютеров:

с микропроцессором 80486 — 15 % ;
с микропроцессором 80386 — 30 % ;
с микропроцессором 80286 — 48 % ;
с микропроцессором 8086 — 7 % .

Построить диаграмму в виде:

а) плоской простой гистограммы;
б) объемной простой гистограммы;
в) плоской круговой диаграммы;
г) объемной круговой диаграммы.

22. Даны натуральные числа V_1, V_2, ... , V_8 , задающие число дней за год, в которых преобладало, соответственно, северное, северо-восточное, восточное, юго-восточное, южное, юго-западное, западное и северо-западное направление ветра. Построить розу ветров. Длины отрезков,

соединяющих центр розы ветров с границами замкнутой ломаной линии, пропорциональны значениям V_1, V_2, ... , V_8 .

23-26. Построить диаграмму, показывающую изменение прибыли абстрактной корпорации за пять лет в виде:

а) плоской простой гистограммы;

б) объемной простой гистограммы;

в) плоской круговой диаграммы;

г) объемной круговой диаграммы.

Каждый элемент диаграммы пометить соответствующим годом.

РЕКОМЕНДУЕМЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Епанешников А., Епанешников В. Программирование в среде Turbo Pascal 7.0. М.: ДИАЛОГ - МИФИ, 1993. 288 с.

Абрамов С. А., Зима Е. В. Начала информатики. М.: Наука: Гл. ред. физ. - мат. лит., 1989. 256 с.

Лабораторный практикум по графике в Турбо Паскале

Составители: Логвинова Елена Александровна
Семенов Валерий Владимирович

Редактор Т. И. Кузнецова

Техн. редактор Н. М. Каленюк

Подписано в печать 29.04.96. Формат 60 x 84 1/16
Бумага офсетная. Печать офсетная.
Усл.печ.л. 1,63. Уч.-изд.л. 1,5. Усл.кр.-отт. 1,75.
Тираж 200 экз. Заказ 154.

Самарский государственный аэрокосмический университет имени академика С. П. Королева.
443086 г. Самара, Московское шоссе, 34.

Издательство Самарского государственного аэрокосмического университета.
443001 г. Самара, ул. Ульяновская, 18.