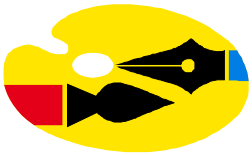
Воронежский Государственный Педагогический Университет

Компьютерная графика

Конспекты лекций

Автор Сазонов Д.О.
1999 год, ВГПУ
e-mail: nimnul@yandex.ru



Компьютерная графика

Раздел компьютерной графики является одним из самых сложных в информатике, хотя очень часто люди думают, что проще графики и быть ничего не может - самое сложное тема процедур или массивов или файлов, на самом деле глубоко заблуждаются. Это происходит потому, что студенты и учащиеся не достаточно понимают, что же на самом деле стоит за словом **графика**. Также ошибочными являются представления о простоте данной темы из за того, что некоторые студенты считают пределом умение рисовать в Paint Brush и строить различные фигуры в PASCAL, BASIC и C++.
В данной главе я постараюсь раскрыть что же на самом деле такое графика и насколько это сложно, а также научить вас некоторым из графических приемов, методике рисования и непосредственно программированию качественных изображений.
Начнем же с понятия самой графики: скажу сразу - все что вы видите на экране является графикой. **Итак графическое изображение - это любое изображение видимое на экране**.
Мы не будет рассматривать графику символов в текстовом режиме, а займемся сразу графическими режимами.
Вообще все графические изображения строятся на экране по точкам, которые (если их приблизить) имеют вид квадрата. Но когда они очень маленькие мы не замечаем их "острых краев". И потому изображения нам кажутся более реалистическими и четкими когда точки очень маленькие. В информатике такие точки называют "ПИКСЭЛ" (pixcel - от английского Picture Cell : ячейка картинки).
Количество точек на экране называется разрешением экрана. Современные мониторы позволяют работать в нескольких режимах экрана. 320x200, 640x480, 800x600,1024x768 и т.д. (первое число означает кол-во точек по горизонтали, второе по вертикали)

Каждая точка имеет свой цвет. Выстраивая рядом точки разного цвета мы можем получать вполне качественные изображения. Компьютер представляет цвет точки в виде аналогово сигнала расщепляя цвет на три его составляющих: красный, зеленый и синий компоненты. Но значения цвета передаются на экран монитора не в виде аналоговоых сигналов, а в виде вариаций тока этого сигнала. Далее монитор преобразует этот сигнал и выводит цвет на экран.
Чем больше цветов позволяет вывести на экран монитор тем больше необходимо памяти, так как необходимо где то хранить значения цвета для каждой точки. Не разумно было бы использовать ОЗУ

компьютера т.к. оперативной памяти всегда недостаточно. Для этих целей предусмотрена специальная видео память. Чем больше видео память тем больше кол-во точек можно поставить на экране и тем больше кол-во цветов можно увидеть. В общем виде кол-во требуемой памяти под разрешение экрана можно посчитать по формуле:

$\frac{H * V * B}{8}$, где H- кол-во точек по горизонтали, V - кол-во точек по вертикали, B - кол-во бит необходимых для представления цвета одной точки.

Следующая таблица показывает сколько бит необходимо для представления различных цветов:

Кол-во цветов	Специальное название данной глубины цвета	Количество бит на одну точку
2	Черно-белое изображение	1
256 черно-белое фотографическое	Grayscale	8
16	Стандартное	4
256	VGA	8
65535	Hi color	16
16777216	True color	24
4294967296	CMYK color	32

Человеческий глаз может воспринять не больше 24 бит оттенков и по этому не понятно зачем используется режим с 32 битами. Картинки использующие 24 бита на одну точку называют картинками фотографического качества.

Учитывая эту таблицу нетрудно посчитать сколько память будет занимать тот или иной режим. По мимо расходования памяти недо знать одну важную вещь: чем больше цветов мы используем и чем больше разрешение экрана ставим - тем медленнее будет обрабатываться изображение на экране.

Далее рассмотрим таблицу зависимости расхода памяти от разрешения экрана и глубины цвета.

Разрешение экрана	**Глубина цвета**	**Необходимо видео памяти в байтах**
320x200	**2**	**16000**
	4	**32000**
	8	**64000**
	16	**128000**
	24	**192000**

640x480	**2**	**76800**
	4	**153600**
	8	**307200**
	16	**614400**
	24	**921600**
800x600	**2**	**120000**
	4	**240000**
	8	**480000**
	16	**960000**
	24	**1440000**
1024x768	**2**	**196608**
	4	**393216**
	8	**786432**
	16	**1572864**
	24	**2359296**

В дальнейшем мы будем использовать лишь несколько из этих режимов: 320x200 при 256 цветов в Qbasic, 640x480 при 16 цветов и 800x600 при 256 цветах на PASCAL. Программирование на C++ практически не будет рассматриваться т.к. языки C++ и PASCAL производила одна фирма BORLAND и графические операторы в них практически идентичны.

Итак, рассмотрим как представляется изображение в памяти компьютера. Мы уже знаем, что все картинки составлены из точек, а каждая точка имеет цвет. Используя это программисты не стали выдумывать ничего сложного для IBM совместимых компьютеров (в отличии от SPECTRUM) и представили экран в виде двумерного массива. И если ширина экрана 640, а высота 480, то массив будет выглядеть следующим образом:

```
Dim screen(480,640)
```

Или

```
VAR screen:array[1..480,1..640] of byte;
```

И чтобы поставить точку на экран надо поставить соответствующее значение в элемент массива. Например: A[10,10]:=10;

Значения цветов также были заранее определены, ведь раньше в компьютерах было всего 16 цветов и потому для каждого из этих цветов был определен соответствующий номер и название:

Номер цвета	Русское название	Английское название
0	Черный	Black
1	Синий	Blue

2	Зеленый	Green
3	Голубой	Cyan
4	Красный	Red
5	Фиолетовый	Magenta
6	Коричневый	Brown
7	Светлосерый	LightGray
8	Темносерый	DarkGray
9	Светлосиний	LightBlue
10	Яркозеленый	LightGreen
11	Светлоголубой	LightCyan
12	Яркокрасный	LightRed
13	Яркофиолетовый	LightMagenta
14	Желтый	Yellow
15	Белый	White

Значит, чтобы, например, провести прямую линию на экране нам придется заполнить соответствующие элементы массивы нужным цветом, который мы можем взять из таблицы.
На самом деле обращение к видео памяти происходит несколько по другому, но принцип тот же, что и при работе с массивами.
Если же используется больше чем 16 цветов, то нам самим придется задавать каждый из цветов. Причем каждый цвет кодируется тремя числами: значение красного, зеленого и синего. Таким образом, распределяя для цветов значения мы создаем так называемую палитру. Палитра - это двумерный массив Nx3, где N - количество цветов, а 3 - значения красного, зеленого и синего для каждого цвета.
Например: VAR Palette:array[1..256,1..3] of byte;
Или DIM Palette(256,3)
Когда мы указываем компьютеру, что на экран надо поместить точку с цветом 100, но компилятор считывает значение из массива палитры со значением 100 и берет составляющие для трех цветов, далее преобразовывает три составляющих в один цвет и выводит его на экран.
В общем виде преобразование идет по следующей формуле (в шеснадцатиричной системе счисления)
00BBGGRRh, где BB -синий цвет, GG - зеленый, а RR - красный цвет. Крайние левые байты не используются, но заразервированны.
В результате преобразования получается 32 битное число. Обратное разложение этого числа на составляющие можно произвести следующим образом:
RR=(Число and $000000FF)
GG=(Число and $0000FF00) shr 8
BB=(Число and $00FF0000) shr 16
где BB -синий цвет, GG - зеленый, а RR - красный цвет
Надо сразу отметить, что в основном работа с графикой основана на работе с ассемблером или по крайней мере с процедурами

ассемблера именно по этому все числа принято записывать в шеснадцатиричной системе счисления. Знак "$" перед числом или буква "h" после числа означает, что данное число записано в шеснадцатиричной системе.
Таким образом мы можем задать одну точку, теперь рассмотрим как строятся более сложные фигуры, такие как окружность и линия.

Алгоритм построения линии

Разумеется и линия и другие так называемые графические примитивы строятся по точкам и проблем не возникает, когда необходимо провести горизонтальную или вертикальную прямую.
Тогда нам достаточно лишь одного цикла. Но что делать когда надо провести произвольную прямую, соединяющую две точки? На этот счет существует несколько алгоритмов, рассмотрим один из них - алгоритм Брезенхейма:

Суть алгоритма:
При построении растрового изображения отрезка всегда выбирается ближайший по вертикали пиксел. При этом из двух точек A и B выбирается та, которая ближе к исходной прямой (в данном случае выбирается точка A, так как a < b). Для этого вводится число d, равное (x2-x1)*(b-a). В случае если d > 0 значение y от предыдущей точки увеличивается на 1, а d на 2*(delta_y - delta_x). В противном случае значение y не изменяется, а значение d заменяется на 2*delta_y.
Проиллюстрирую алгоритм в действии на BASIC.
(я думаю вы сможете в нем разобраться сами)

```
REM Рисование линий
 INPUT "Начало отрезка X1,Y1"; x1, y1
 INPUT "Конец отрезка X2,Y2"; x2, y2
SCREEN 13: REM режим 320x200 при 256 цветах
dx = ABS(x2 - x1)
dy = ABS(y2 - y1)
IF x2 >= x1 THEN sx = 1 ELSE sx = -1
IF y2 >= y1 THEN sy = 1 ELSE sy = -1

REM ------------------------------ ОБЩИЙ IF-------------------
IF (dy <= dx) THEN
                    d = dy * 2 - dx
                    d1 = dy * 2
                   d2 = (dy - dx) * 2
                   PSET (x1, y1), 15
                   x = x1 + sx
                   y = y1
REM *********** цикл for ***********
FOR i = 1 TO dx
   x = x + sx
```

```
      REM ---------- if в for ----------
 IF d > 0 THEN
              d = d + d2
              y = y + sy
             REM -- иначе --
           ELSE d = d + d1
 END IF
    PSET (x, y), 15
    REM ---------- закрылся if ----------
NEXT i
   REM ******* закрылся FOR ********
END IF

IF (dy > dx) THEN
                 d = (dx * 2) - dy
                 d1 = dx * 2
                d2 = (dx - dy) * 2
                  PSET (x1, y1), 15
                x = x1: y = y1 + sy
     REM ****** открылся цикл FOR ********
FOR i = 1 TO dy
   y = y + sy
   REM ---- Открылся if ----
      IF (d > 0) THEN
                     d = d + d2
                     x = x + sx
                  ELSE d = d + d1
       END IF
     PSET (x, y), 15
         REM ---- закрылся if ----
      NEXT i
   REM ---------- закрылся if ----------
END IF
```

В BASIC довольно легко запутаться в конструкции if …then … else, кроме того этот алгоритм я привожу для того, чтобы можно было работать непосредственно с экраном, не опираясь на стандартные функции компилятора, связанные с графикой. Это нам пригодиться, когда мы будем работать с режимом 300x200 в PASCAL - ведь графические библиотеки PASCALя не поддерживают этот режим и мы не сможем пользоваться ни одной графической функцией при работе в нем. Соответственно и другие более мощные режимы адаптера SVGA паскалем также не предусмотрены - для этого напишем свои собственные библиотеки и собственный модуль GRAPH.
Итак пример рисования линии на PASCAL в режиме 320x200:
{рисование отрезка}

```
var screen:array[1..200,1..320] of byte absolute $0a000:$00000;

procedure line(x1,y1,x2,y2:word;c:byte);
var x,y,i,dx,dy,sx,sy,d1,d,d2:integer;
begin
dx:=abs(x2-x1);
```

```
dy:=abs(y2-y1);
if x2>=x1 then sx:=1 else sx:=-1;
if y2>=y1 then sy:=1 else sy:=-1;

{общий IF}
if (dy<=dx) then
begin
d:=(dy shl 1)-dx; {shl 1 - аналогично умножению на 2}
d1:=(dy shl 1); {только работает гораздо быстрее}
d2:=(dy-dx) shl 1;

y:=y1;
x:=x1+sx;
for i:=1 to dx do begin
x:=x+sx;
if (d>0) then begin
              d:=d+d2;
              y:=y+sy;
             end else d:=d+d1;
screen[x,y]:=c
               end; {of for}
end else
begin
d:=(dx shl 1)-dy;
d1:=(dx shl 1);
d2:=(dx-dy) shl 1;

x:=x1;
y:=y1+sy;
for i:=1 to dy do begin
y:=y+sy;
if d>0 then begin
            d:=d+d2;
            x:=x+sx;
           end else d:=d+d1;
screen[x,y]:=c;
               end; {of for}
end;
end;

begin
{используя ассемблер переходим в нужный режим}
asm
mov ax,0013h
int 10h
end;

{рисуем произвольную линию.Значения вы можете менять по усмотрению}
line(10,30,70,160,15);
end.
```

Заметьте, что в этой программе мы не используем никаких стандартный модулей паскаля и тем самым программа получается несколько меньше. Экран же рассматривается как массив - это позволяет сделать директива absolute, которая просто указывает место положения экрана в памяти.

Дело в том, что у экрана тоже есть память и ее место (или адрес) всегда одни и тот же - $0a000:00000. Нам достаточно указать, что наш массив будет располагаться по этому же адресу и тогда, мы сможем обращаться к видео памяти, как к элементу массива!
Далее все алгоритмы будут приводиться на BASIC т.к. этот язык изучается первым почти во всех школах и во многих институтах. Кроме того любой алгоритм с BASIC можно легко перевести на любой другой язык программирования!

Итак, сама программа:

```
REM рисование окружности по алгоритму Брезенхейма
SCREEN 13: REM переходим в режим 320x200

INPUT "Position X,Y:"; x, y
INPUT "Radius:"; r
INPUT "Color (0..255):"; c
CLS
x1 = 0
y1 = r
d = (1 - r) * 2

DO
PSET (x + x1, y + y1), c
PSET (x - x1, y + y1), c
PSET (x + x1, y - y1), c
PSET (x - x1, y - y1), c
f = 0
REM выход из цикла по окончании работы

IF (y1 < 0) THEN EXIT DO

REM ------------------------------ Основной алгоритм -----------------
IF d < 0 THEN s = 2 * (d + y1) - 1: IF s <= 0 THEN x1 = x1 + 1: d = d + 2 * x1 + 1: f = 1

IF d > 0 THEN IF d > 0 THEN s = 2 * (d - x1) - 1: IF s > 0 THEN y1 = y1 - 1: d = d + 1 - 2 * y1: f = 1

IF f = 0 THEN x1 = x1 + 1: y1 = y1 - 1: d = d + 2 * (x1 - y1 - 1)

REM do...loop - обеспечивает бесконечный цикл
LOOP
```

Данный алгоритм может показаться запутанным и потому иногда (в случае, когда скорость выполнения не важна) круг рисуют с помощью тригонометрических формул Sin и Cos. Делается это следующим образом:

```
REM рисование круга с помощью тригонометрии
SCREEN 13

INPUT "Position X,Y:"; x, y
INPUT "Radius:"; r
INPUT "Color (0..255):"; c
CLS
```

```
FOR i = -360 TO 360
PSET (x + r * SIN(i), y + r * COS(i)), c
NEXT i
```

Этот алгоритм гораздо меньше и легче запоминается, но работает он гораздо медленнее предыдущего - даже видно как он прорисовывается на экране.
Из последнего алгоритма видно как можно получить эллипс:
Достаточно умножать функции на разные радиусы. Например:

```
PSET (x +A * SIN(i), y +B * COS(i)), c
```

Рисование квадрата и треугольника можно получить, используя уже известный алгоритм с линиями.

Палитра

Так как рисуя мы используем 256 цветов в картинках необходимо знать каким образом можно задавать каждый из этих цветов.

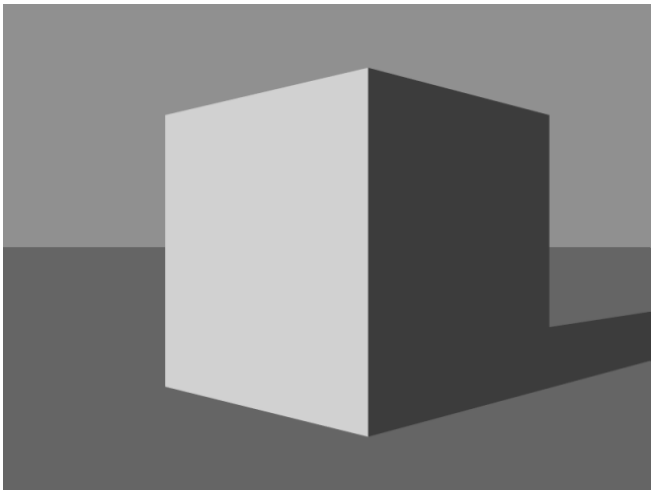
В Basic существует специальная команда, позволяющая менять значения каждого из цветов - palette.
Формат команды:
Palette цвет, значение цвета в кодировки RGB (длинное значение. См. выше о формате $00BBGGRR)
Тремя комбинациями цветов в компьютере так же как и в рисовании красками можно создать любой из цветов путем смешивания красного, синего и зеленого. Поэкспериментируйте с ней! В этом вам поможет небольшая программа на BASIC, которая позволяет генерировать любой цвет путем смешения красного, зеленого и синего цветов. Управление программой клавишами:z,x, c,v, b,n

```
REM моделирование цвета (C)
SCREEN 13

REM задаем значение цвета по формуле 00BBGGRRh
REM в шеснадцатиричной системе
REM переход в эту систему в BASIC можно сделать написав перед числом &H
REM ------------------------------------------------------------
REM правда в BASIC составляющая каждого цвета должна быть меньше 63 (3Fh)
c = &H342700

red = 0
blue = 0
green = 0

DO
c$ = INKEY$
```

```
IF (c$ = "z") AND (red < 63) THEN red = red + 1
IF (c$ = "x") AND (red > -1) THEN red = red - 1

IF (c$ = "c") AND (green < 63) THEN green = green + 1
IF (c$ = "v") AND (green > -1) THEN green = green - 1

IF (c$ = "b") AND (blue < 63) THEN blue = blue + 1
IF (c$ = "n") AND (blue > -1) THEN blue = blue - 1
c = 65536 * blue + 256 * green + red
PALETTE 0, c
LOCATE 1, 1
PRINT "Color is &H"; HEX$(c); "              "
PRINT "z - add red, x - sub red"
PRINT "c - add green, v - sub green"
PRINT "b - add blue, n - sub blue"

rem Выход если нажата клавиша ESC
LOOP WHILE c$ <> CHR$(27)
```

Общие цветовые решения

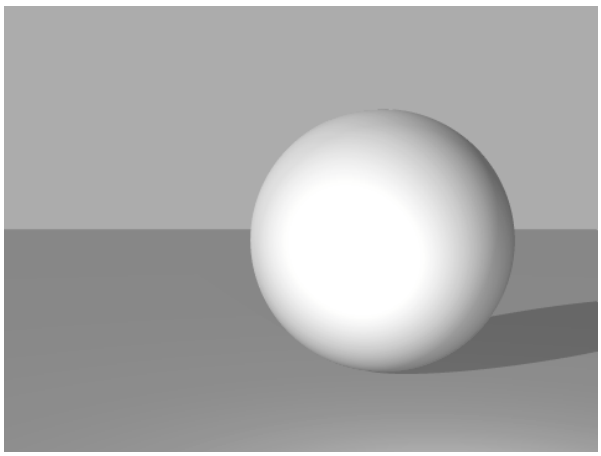
Как и живопись компьютерная графика подчиняются одним и тем же законам цвета и композиции в целом. Рассмотрим наиболее общие примеры подбора цвета. Например хорошим сочетанием может оказаться желтый и черный, оранжевый и черный, синий и белый, зеленый и белый. Плохое же сочетание - это желтый на белом, синий на красном, ярко зеленый на красном и т.д. Кроме того если всю картинку нарисовать одними яркими цветами изображение на ней будет расплываться и задний план картины будет отвлекать внимание от основного сюжета. По этому главный предмет обычно рисуется более ярким цветом, а фон более темным. Кроме того поверхности на которые падает свет должны быть более яркими, чем теневые стороны. Так как обычно стандартных шестнадцати цветов не хватает, вам наверняка придется работать с "палитрой", чтобы определить свои цвета. Рассмотрим пример: Допустим, нам надо нарисовать куб, на который слева спереди падает свет под углом 45^0. Для начала нарисуем его на бумаге, а затем с помощью линий нарисуем на компьютере.



Для того, чтобы нарисовать этот куб и задний фон нам потребуется 4 цвета. Первым делом нам придется задать общий фон картинки - обычно цвет неба (серый),затем нарисовать куб, далее

линию горизонта, а там остается создать требуемые цвета (создать цвет и получить его значение вы уже можете) и раскрасить рисунок. Вот какой получится программа:

```
REM трехмерный куб. Нарисован без использования 3d графики
SCREEN 13

REM заливаю всю область экрана серым цветом
PAINT (0, 0), 8

REM координаты вершин куба беду наугад
REM Сначала рисую вертикальные линии
LINE (180, 50)-(180, 160), 15
LINE (250, 70)-(250, 145), 15
LINE (100, 70)-(100, 145), 15
REM соединаю вершины.Верхние
LINE (180, 50)-(250, 70)
LINE (180, 50)-(100, 70)
REM Нижние
LINE (180, 160)-(250, 145)
LINE (180, 160)-(100, 145)
REM Провожу линию горизонта
LINE (0, 120)-(100, 120), 15
LINE (250, 120)-(320, 120), 15
REM Заливка цветом
PAINT (190, 100), 0, 15
PAINT (120, 100), 7, 15
REM определяю новый цвет для заливки пола
PALETTE 16, &HF0F0F
PAINT (1, 140), 16, 15
REM рисую тень от куба
LINE (250, 145)-(320, 133), 0
LINE (230, 130)-(320, 126), 0
PAINT (260, 132), 0, 0
REM убираю белый цвет
PALETTE 15, 8
END
```



По этому же принципу попробуйте нарисовать шар. Теперь нам уже понадобиться все 63 оттенка серого, поскольку у шара нет резких переходов с белого на черное и наоборот. Попробуйте нарисовать шар самостоятельно, но если что-либо не получиться или возникнут вопросы готовая программа поможет вам.

```
REM шар
SCREEN 13

FOR i = 63 TO 0 STEP -1
PALETTE i, 65536 * i + 256 * i + i
NEXT i

PAINT (0, 0), 25
LINE (0, 120)-(320, 120), 0

FOR i = 0 TO 63
CIRCLE (200, 100), 63 - i, i
CIRCLE (201, 100), 63 - i, i
CIRCLE (200, 101), 63 - i, i
NEXT i
CIRCLE (200, 100), 63, 0
CIRCLE (201, 100), 63, 0
CIRCLE (200, 101), 63, 0
PAINT (13, 190), 16, 0
```

Построение сложных графических объектов

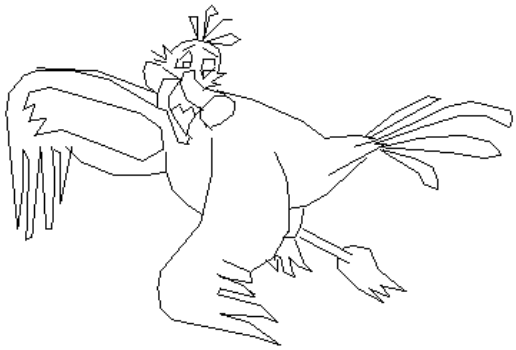
Иногда в практике требуется построить довольно сложное графическое изображение. Тогда на одно высчитывание координат отрезков потребуется уйма времени. В таком случае поступают проще - пишут небольшой графический редактор. Из управления нам понадобиться двигать указатель вверх, вниз, вправо и влево, а также ставить точку. Когда мы поставим две точки рядом - они соединяются и получается линия. Таким образом мы можем построить очень неплохие рисунки. Все координаты линий записываются в файл и в результате получается готовая программа и рисунок. Конечно, написание редактора вещь тоже не легкая, но зато уже при его наличии вы выграните во времени создавая необходимую и все последующие картинки. Проще всего такой редактор написать на языке Паскаль. Вот небольшой пример такого редактора:

```
{пример графического редактора}
uses graph,crt;

var i:integer;
   xold,yold,x,y,x1,y1:integer;
   press:boolean;
   c:char;
   maxX,maxY:integer;
   t:text;
   s,s1,s2,s3:string; {переменные для хранения промежуточных результатов}

begin
{подсоединяю графику}
i:=detect;
initgraph(i,i,'');
```

```
press:=false;

{открываю файл в который будет записан результат}
assign(t,'graf.bas');
rewrite(t);
{записываю заголовок файла}
writeln(t,'REM by GrafEdit.Методика программрования');
writeln(t,'Screen 12');

{координаты графического указателя}
x:=0;
y:=0;
maxX:=getmaxX;
maxY:=getmaxY;
{устанавливаю режим наложения}
SetWriteMode(XORPut);

repeat
{сл.две сточки нужны чтобы изображение не мигало}
{данный трюк применялся в ZX SPECTRUM}

line(xold,yold,xold+5,yold+4);
if press then line(x1,y1,xold,yold);
c:=readkey;

{проверка нажатия клавиш}
if (ord(c)=077) and (x<maxX) then inc(x,2);
if (ord(c)=072) and (y>0) then dec(y,2);
if (ord(c)=080) and (y<maxY) then inc(y,2);
if (ord(c)=075) and (y>0) then dec(x,2);

{если нажат пробел начать тянуть линию,
если пробел нажат еще раз нарисовать линию}
if c=' ' then begin
            {издается звуковой сигнал}
            sound(500);
            delay(300);
            nosound;
            press:=not press;
             if press then begin
                        x1:=x;
                        y1:=y;
                       end else begin
                                {рисуется линия на экране}
                                SetWriteMode(NormalPut);
                                 line(x1,y1,x,y);
                                SetWriteMode(XORPut);
                                str(x1,s);
                                str(y1,s1);
                                str(x,s2);
                                str(y,s3);
                    {Записывается команда в файл}
                    writeln(t,'Line('+s+','+s1+')-('+s2+','+s3+'),15');
                                end;

           end;
if press then line(x1,y1,xold,yold);

line(xold,yold,xold+5,yold+4);
xold:=x;
```

```
yold:=y;

until c=#27;
{закрывается файл}
close(t);
end.
```

С помощью такого редактора уже можно нарисовать довольно сложные картинки. Приведу пример картинки, нарисованной в этом редакторе:



Но данный метод опять же далек от совершенства - программу можно улучшить и добавить много новых возможностей, но на это будет потрачено слишком много времени (гораздо больше, чем на создание самого рисунка). Кроме того практически невозможно нарисовать с помощью линий фотографию или очень сложный рисунок. Но выход есть и из этого. Для работы с рисунками самого высокого качества, фотографиями есть уже готовые редакторы, написанные профессиональными программистами. Программы позволяющие рисовать или обрабатывать графическую информацию называются графическими редакторами. Существует великое множество графических редакторов, причем такие редакторы существуют на любом компьютере и на любой операционной системе. Например в ZX SPECTRUM - есть ART Studio и ARTIST 2, для IBM существует Paint Brush (который является самым простым из существующих редакторов на IBM), для LINUX - The Gimp и т.д.
Так как мы рассматриваем операционную систему MS DOS, будут рассмотрены графические редакторы только под эту операционную систему. Графический редактор Paint Brush установлен прочти на каждом компьютере использующем систему MS DOS и Windows по этому будем рассматривать картинки, нарисованные именно в Paint.
Нарисованную картинку в Paint можно записать на диск, при этом картинки записываются с расширениями BMP и PCX. Например: ariel.pcx или dom.bmp. Данные расширения называются форматом графического файла. Обычно программисты рисуют картинки в любом из графических редакторов, а затем эту картинку вставляют в свою программу. К сожалению объем данного пособия не позволяет рассмотреть подробно этот метод, но тем не менее в качестве

примера приведем небольшую упрощенную программу чтения картинки формата bmp и вывода ее на экран. Данная программа предназначена в основном для тех кого интересует этот прием.

```
program bmp256_test;
var i,j,k,ofs,mX,mY,rastr,len,bits:integer;
    buf:array[1..60000] of byte;
    f:file;

{устанавливает значение цвета}
procedure palette(old,r,g,b:byte);assembler;
asm
mov dx,03c8h;mov al,old;out dx,al
inc dx;mov al,r;shr al,2;out dx,al
mov al,g;shr al,2;out dx,al;mov al,b
shr al,2;out dx,al
end;

{ставит точку на экране в позиции x,y цветом C}
procedure pset(x,y:integer;c:byte);
begin
asm
mov bx,x;mov cx,y;mov dl,c
mov ax,0a000h;mov es,ax;mov al,160
mul cl;add ax,ax;add bx,ax;mov [es:bx],dl
end
end;

begin
 assign(f,'pic.bmp');
 reset(f);
 blockread(f,buf,50000,j);
{позиция самого рисунка}
rastr:=buf[11]+buf[12]*256;
{ширина изображения}
mx:=buf[19]+buf[20]*256;
{высота изображения}
my:=buf[23]+buf[24]*256;

{Установка графического режима}
asm
mov ax,0013h
int 10h
end;
{устанавливаем палитру рисунка}
I:=53;
for j:=0 to 255 do begin
if (buf[i+1]=0) and (buf[i+3]<>0) then begin buf[i+1]:=buf[i+3];buf[i+3]:=0;end;
Palette(j,buf[i],buf[i+1],buf[i+2]);
Inc(i,4);
                end;
{Выводим растр}
for i:=my downto 0 do
 for j:=0 to mx do begin
   Pset(j,i,buf[rastr]);
   Inc(rastr);
                    end;

end.
```