
MicroPress
Visual T_EX

**FastDraw
Macro Package**

Mike Krutikov

Copyright © 1989–2001 by MicroPress Inc.

Contents

1	Introduction	2
2	Simple Objects	2
2.1	Line <code>\Vline</code>	2
2.2	Vector <code>\Vvector</code>	3
2.3	Rectangle <code>\Vrectangle</code>	3
2.4	Filled rectangle <code>\Vfrectangle</code>	3
2.5	Arc <code>\Varc</code>	3
2.6	Circle <code>\Vcircle</code>	4
2.7	Filled circle <code>\Vfcircle</code>	4
2.8	Ellipse <code>\Vellipse</code>	4
2.9	Filled ellipse <code>\Vfellipse</code>	5
2.10	Rotated ellipse <code>\Vrellipse</code>	5
2.11	Filled rotated ellipse <code>\Vfrellipse</code>	5
2.12	Sector <code>\Vsector</code>	6
2.13	Filled sector <code>\Vfsector</code>	6
2.14	Bezier curves	6
2.14.1	<code>\Vbezier</code>	6
2.14.2	<code>\Vcurve</code>	7
3	Composed Objects	7
3.1	Polyline <code>\Vpolyline</code>	7
3.2	Polygon <code>\Vpolygon</code>	8
3.3	Filled polygon <code>\Vfpolygon</code>	8
4	Graphics Context	9
4.1	Line width <code>\Vsetlinewidth</code>	9
4.2	Line style <code>\Vsetlinestyle</code>	10
4.3	Fill style <code>\Vsetfillstyle</code>	10
4.4	Gray shade <code>\Vsetgray</code>	11

1 Introduction

FastDraw is a general-purpose drawing package for $\text{V}\text{T}\text{E}\text{X}$. It is based on the PostScript capabilities of $\text{V}\text{T}\text{E}\text{X}$ output drivers. Notice that starting from $\text{V}\text{T}\text{E}\text{X}$ 6.5 FastDraw in PDF mode is implemented via GeX calls; so the `-ox` command line switch is required. Both $\text{L}\text{A}\text{T}\text{E}\text{X}$ and $\text{P}\text{L}\text{A}\text{I}\text{N}\text{T}\text{E}\text{X}$ formats are supported.

`PICFAST.STY` and related macro packages enhance the $\text{L}\text{A}\text{T}\text{E}\text{X}$ picture environment by adding additional commands and redefining others to operate through

PostScript graphics rather than attempt to produce graphics with TEX commands.

The macro definition files are:

- `PICFAST.STY` — replacement picture environment for $\text{L}\text{A}\text{T}\text{E}\text{X}$, fully compatible with `PICEDIT`.
- `PICFAST.TEX` — replacement picture environment for $\text{P}\text{L}\text{A}\text{I}\text{N}\text{T}\text{E}\text{X}$, fully compatible with `PICEDIT`.
- `FAST.STY` — PostScript graphics definitions for `PICFAST.STY`.
- `FAST.TEX` — PostScript graphics definitions for `PICFAST.TEX`.

The material is organized as follows. In the first section we describe graphics primitives to draw simple objects. Second section deals with composed objects like polylines and polygons. Third section describes the pre-defined features of this macro package (like line and fill styles, etc.) known as graphics context (GC). The macros to change this GC are described as well.

The syntax of defining picture for FastDraw is just the same as for conventional $\text{L}\text{A}\text{T}\text{E}\text{X}$:

Syntax:

```
\setlength{\unitlength}{0.5cm}
\begin{picture}(10.0,10.0)
    drawing macros...
\end{picture}
```

where *drawing macros* will be described in the following sections.

2 Simple Objects

2.1 Line `\Vline`

The simplest drawing macro is the one for drawing a line.

Syntax:

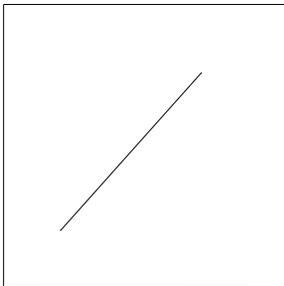
```
\Vline(x1,y1)-(x2,y2);
```

Note that semicolon after the operator is required by the syntax. The macros produce the line starting at (x_1, y_1) and ending at (x_2, y_2) . All the coordinates are simple numbers, that are mapped to the positions within the picture by the `\unitlength` value.

Example:

```
\setlength{\unitlength}{1mm}
\begin{picture}(50.0,50.0)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(50,0.1)-(0,0.1);
  \Vline(10,10)-(35,38);
\end{picture}
```

produces



2.2 Vector `\Vvector`

Same as `\Vline`, but draws vector instead of a line.

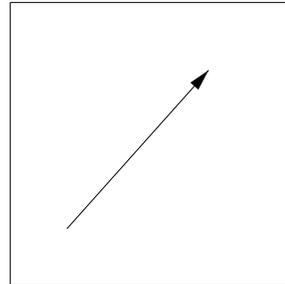
Syntax:

```
\Vvector(x1,y1)-(x2,y2);
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(50,0.1)-(0,0.1);
  \Vvector(10,10)-(35,38);
\end{picture}
```

produces



2.3 Rectangle `\Vrectangle`

Draws the rectangle defined by the coordinates of its opposite corners.

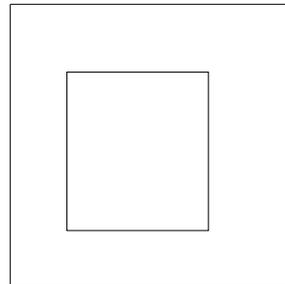
Syntax:

```
\Vrectangle(x1,y1)-(x2,y2);
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vrectangle(10,10)-(35,38);
\end{picture}
```

produces



2.4 Filled rectangle `\Vfrectangle`

Same as `\Vrectangle` but draws a filled object.

Syntax:

```
\Vfrectangle(x1,y1)-(x2,y2);
```

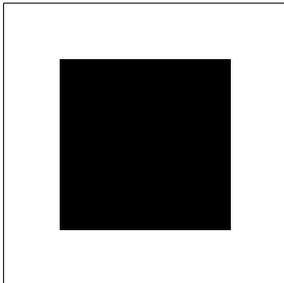
Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vfrectangle(10,10)-(40,40);
\end{picture}

```

produces



2.5 Arc \Varc

Draws the arc with the center at (x,y) , radius r from angle $a1$ to angle $a2$.

Syntax:

```
\Varc(x,y),r,a1,a2;
```

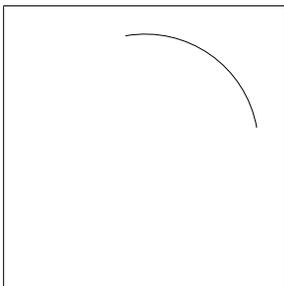
Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Varc(25,25),20,10-100;
\end{picture}

```

produces



2.6 Circle \Vcircle

Draws the circle with the center at (x,y) and radius r .

Syntax:

```
\Vcircle(x,y),r;
```

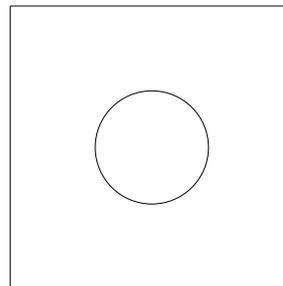
Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vcircle(25,25),20;
\end{picture}

```

produces



2.7 Filled circle \Vfcircle

Draws the filled circle with the center at (x,y) and radius r .

Syntax:

```
\Vfcircle(x,y),r;
```

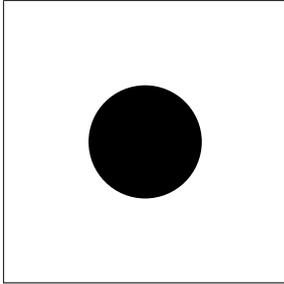
Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vfcircle(25,25),20;
\end{picture}

```

produces



2.8 Ellipse `\Vellipse`

Draws the ellipse with the center at (x,y) , the radius along the x -axis rx , and the radius along y -axis ry .

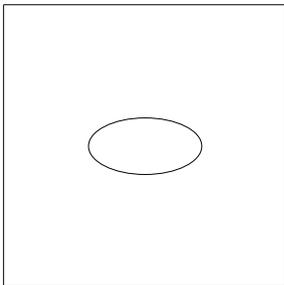
Syntax:

```
\Vellipse(x,y),rx,ry;
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vellipse(25,25),20,10;
\end{picture}
```

produces



2.9 Filled ellipse `\Vfellipse`

Draws the filled ellipse with the center at (x,y) , the radius along the x -axis rx , and the radius along the y -axis ry .

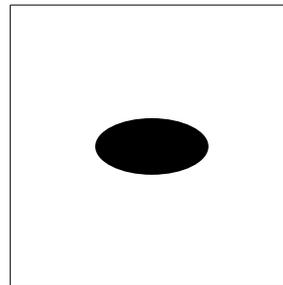
Syntax:

```
\Vfellipse(x,y),rx,ry;
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vfellipse(25,25),20,10;
\end{picture}
```

produces



2.10 Rotated ellipse `\Vrellipse`

Draws the ellipse with the center at (x,y) , radius along the x -axis rx , and the radius along the y -axis ry , rotated counterclockwise a degrees.

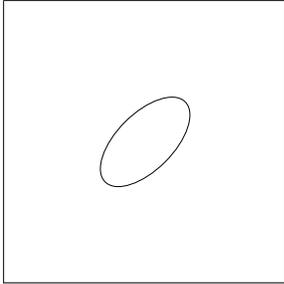
Syntax:

```
\Vrellipse(x,y),rx,ry,a;
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vrellipse(25,25),20,10,45;
\end{picture}
```

produces



2.11 Filled rotated ellipse `\Vfrellipse`

Draws the filled ellipse with the center at (x,y) , the radius along the x -axis rx , and the radius along the y -axis ry , rotated counterclockwise a degrees.

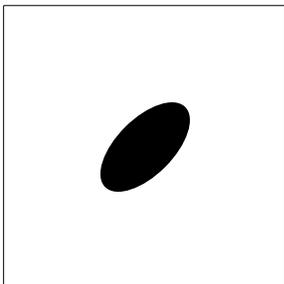
Syntax:

```
\Vfrellipse(x,y),rx,ry,a;
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vfrellipse(25,25),20,10,45;
\end{picture}
```

produces



2.12 Sector `\Vsector`

Draws the sector with the center at (x,y) , radius r , from angle $a1$ degrees to angle $a2$ degrees.

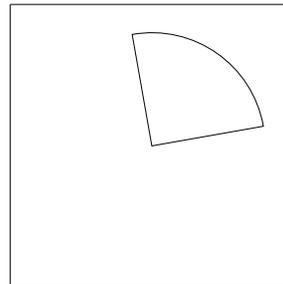
Syntax:

```
\Vsector(x,y),r,a1,a2;
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vsector(25,25),20,10-100;
\end{picture}
```

produces



2.13 Filled sector `\Vfsector`

Draws the filled sector with the center at (x,y) , radius r , from the angle $a1$ degrees to the angle $a2$ degrees.

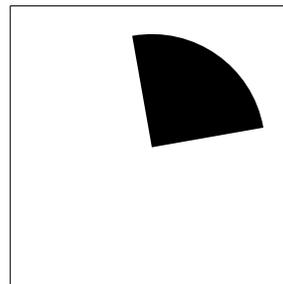
Syntax:

```
\Vfsector(x,y),r,a1,a2;
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vfsector(25,25),20,10-100;
\end{picture}
```

produces



2.14 Bezier curves

2.14.1 `\Vbezier`

Draws the Bezier *quadratic* curve from (x_1, y_1) to (x_3, y_3) with the slope at the starting and the end points defined by the third point (x_2, y_2) . Same as the \LaTeX `\bezier` operator.

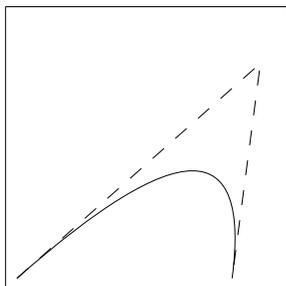
Syntax:

```
\Vbezier(x1,y1),(x2,y2),(x3,y3);
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vbezier(2,2)-(45,40)-(40,2);
  \Vsetlinestyle(dashed);
  \Vpolyline{
    \Vpoint(2,2);
    \Vpoint(45,40);
    \Vpoint(40,2);
  }
\end{picture}
```

produces



2.14.2 `\Vcurve`

Draws the Bezier *cubic* curve from (x_1, y_1) to (x_4, y_4) . The slope at the starting point is defined by the direction of the line from (x_1, y_1) to (x_2, y_2) . The slope at the end point is defined by the direction of the line $(x_3, y_3)-(x_4, y_4)$.

This is the same as the PostScript curve operator.

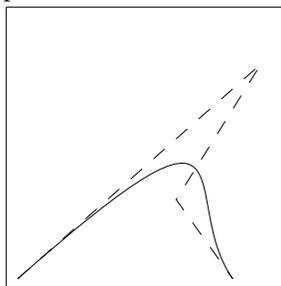
Syntax:

```
\Vcurve(x1,y1),(x2,y2),(x3,y3),(x4,y4);
```

Example. Typing:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vcurve(2,2)-(45,40)-(30,16)-(40,2);
  \Vsetlinestyle(dashed);
  \Vpolyline{
    \Vpoint(2,2);
    \Vpoint(45,40);
    \Vpoint(30,16);
    \Vpoint(40,2);
  }
\end{picture}
```

produces



3 Composed Objects

Presently there are only three composed objects. They are: `\Vpolyline`, `\Vpolygon` (closed polyline), and `\Vfpolygon`. Their difference from the simple objects is that a composed object is defined by an arbitrary number of points.

3.1 Polyline `\Vpolyline`

Let us start with the polyline. If you want to draw the polyline defined by the following four points: (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , and (x_4, y_4) , you are to use the following construction:

```
\Vpolyline{
```

```

\Vpoint(x1,y1);
\Vpoint(x2,y2);
\Vpoint(x3,y3);
\Vpoint(x4,y4);
}

```

Note, that `\Vpolyline` acts as a grouping macro for `\Vpoint`'s. That is why the enclosing brace `}` of `\Vpolyline` needs no semicolon. More formally, the syntax for polyline is:

Syntax:

```

\Vpolyline{
  points...
}

```

For example, typing

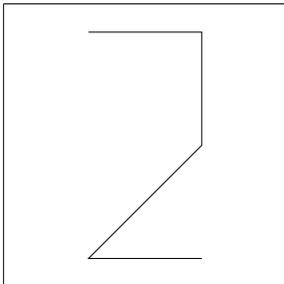
Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vpolyline{
    \Vpoint(15,45);
    \Vpoint(35,45);
    \Vpoint(35,25);
    \Vpoint(15,5);
    \Vpoint(35,5);
  }
\end{picture}

```

produces



3.2 Polygon `\Vpolygon`

Analogously, if you want to draw the polygon defined by the following four points: (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , and (x_4, y_4) , you are to use the following construction:

Syntax:

```

\Vpolygon{
  \Vpoint(x1,y1);
  \Vpoint(x2,y2);
  \Vpoint(x3,y3);
  \Vpoint(x4,y4);
}

```

Or, formally:

```

\Vpolygon{
  points...
}

```

Polygon is a closed polyline. One may draw a polygon using `\Vpolyline` macros with the first and the last point being the same. But it is a bad style. The advantage of `\Vpolygon` becomes apparent when drawing thick lines. `\Vpolygon` will produce the correct joint style in all the corners of the polygon, while `\Vpolyline` will not.

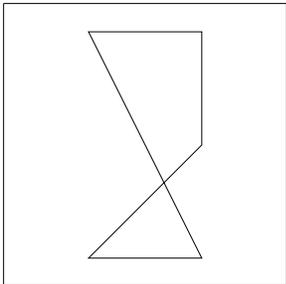
Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vpolygon{
    \Vpoint(15,45);
    \Vpoint(35,45);
    \Vpoint(35,25);
    \Vpoint(15,5);
    \Vpoint(35,5);
  }
\end{picture}

```

produces



3.3 Filled polygon `\Vfpolygon`

Syntax:

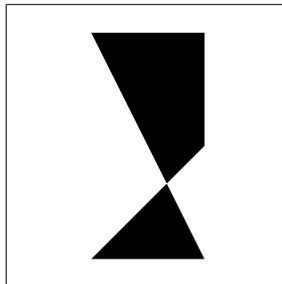
```
\Vfpolygon{  
  
  points...  
  
}
```

Note, that the filling of polygon will be produced according to the “non-zero” winding rule. This may be important when the defining polyline has self-intersections.

Example:

```
\begin{picture}(50,50)  
  \Vline(0,0)-(0,50);  
  \Vline(0,50)-(50,50);  
  \Vline(50,50)-(50,0);  
  \Vline(0,0.1)-(50,0.1);  
  \Vfpolygon{  
    \Vpoint(15,45);  
    \Vpoint(35,45);  
    \Vpoint(35,25);  
    \Vpoint(15,5);  
    \Vpoint(35,5);  
  }  
\end{picture}
```

produces



4 Graphics Context

The Graphics Context (GC) defines the default graphics settings like linewidth, linestyle (solid, dotted, dashed, etc.), fillstyle, and the dimensions of standard elements (arrow nose for `\vector`, etc.). In most cases you will be satisfied with the default settings. But for the advanced drawings you may want to change the GC. Following section describes how to manipulate the GC with \TeX macros.

The default settings are the following:

- Default line width: 0pt
- Default line style: solid
- Default dimensions of the arrow nose: base=4pt, height=20pt
- Default fill style: solid fill

The default settings are defined in absolute units. This ensures that for any user’s unit length corresponding elements will appear to be the same.

If one wants to scale the picture by changing the `\unitlength` parameter the result will be as follows: all drawings will be scaled except by the default elements in GS (i.e., arrow noses will not be scaled).

All macros that allow to redefine this elements of GC use abstract units, defined by the `\setunitlength` macro. Therefore the elements that were re-defined explicitly will demonstrate “normal” scaling behavior (i.e. arrow noses, that were re-defined explicitly, will be scaled).

4.1 Line width `\Vsetlinewidth`

This macro allows one to control the current drawing line width (default is zero, that instructs drivers to produce as thin a line as possible).

Syntax:

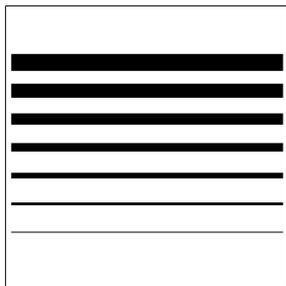
```
\Vsetlinewidth(w);
```

with `w` being the new line width, measured in the units of `\unitlength`.

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vline(1,10)-(49,10);
  \Vsetlinewidth(0.5);
  \Vline(1,15)-(49,15);
  \Vsetlinewidth(1);
  \Vline(1,20)-(49,20);
  \Vsetlinewidth(1.5);
  \Vline(1,25)-(49,25);
  \Vsetlinewidth(2);
  \Vline(1,30)-(49,30);
  \Vsetlinewidth(2.5);
  \Vline(1,35)-(49,35);
  \Vsetlinewidth(3);
  \Vline(1,40)-(49,40);
\end{picture}
```

produces



4.2 Line style `\Vsetlinestyle`

`\Vsetlinestyle` allows one to control the current line style, switching from the default solid style to other possible ones: dotted, dashed, or dashed-dotted.

Syntax:

```
\Vsetlinestyle(style);
```

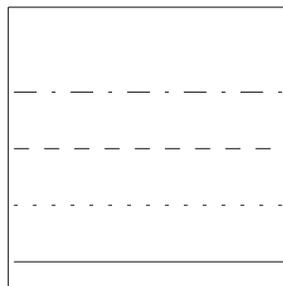
where `style` is one of the keywords:

```
solid    Solid line
dashed   Dashed line
dotted   Dotted line
ddotted  Dash-dotted line
```

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vline(1,5)-(49,5);
  \Vsetlinestyle(dotted);
  \Vline(1,15)-(49,15);
  \Vsetlinestyle(dashed);
  \Vline(1,25)-(49,25);
  \Vsetlinestyle(ddotted);
  \Vline(1,35)-(49,35);
\end{picture}
```

produces



4.3 Fill style `\Vsetfillstyle`

This macro controls current fill style.

Syntax:

```
\Vsetfillstyle(style);
```

where `style` is one of these keywords:

```
solid — Solid fill
```

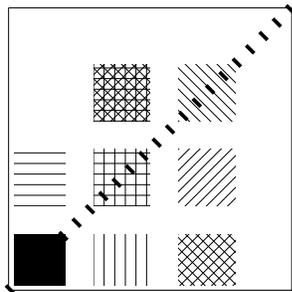
`vlines` — Vertical lines
`hlines` — Horizontal lines
`vhlines` — Vertical and Horizontal lines
`crossed` — Crossed lines
`slashed` — Slashed lines
`backslashed` — Backslashed lines
`cobweb` — Cobweb

Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vsetlinestyle(dotted);
  \Vsetlinewidth(2);
  \Vline(0,0)-(50,50);
  \Vsetfillstyle(solid);
  \Vfrectangle(1,1)-(10,10);
  \Vsetfillstyle(vlines);
  \Vfrectangle(15,1)-(25,10);
  \Vsetfillstyle(hlines);
  \Vfrectangle(1,15)-(10,25);
  \Vsetfillstyle(vhlines);
  \Vfrectangle(15,15)-(25,25);
  \Vsetfillstyle(crossed);
  \Vfrectangle(30,1)-(40,10);
  \Vsetfillstyle(slashed);
  \Vfrectangle(30,15)-(40,25);
  \Vsetfillstyle(backslashed);
  \Vfrectangle(30,30)-(40,40);
  \Vsetfillstyle(cobweb);
  \Vfrectangle(15,30)-(25,40);
\end{picture}
  
```

produces

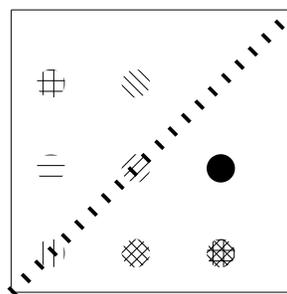


Example:

```

\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vsetlinestyle(dotted);
  \Vsetlinewidth(2);
  \Vline(0,0)-(50,50);
  \Vsetfillstyle(vlines);
  \Vfcircle(7,7),5;
  \Vsetfillstyle(hlines);
  \Vfcircle(7,22),5;
  \Vsetfillstyle(vhlines);
  \Vfcircle(7,37),5;
  \Vsetfillstyle(crossed);
  \Vfcircle(22,7),5;
  \Vsetfillstyle(slashed);
  \Vfcircle(22,22),5;
  \Vsetfillstyle(backslashed);
  \Vfcircle(22,37),5;
  \Vsetfillstyle(cobweb);
  \Vfcircle(37,7),5;
  \Vsetfillstyle(solid);
  \Vfcircle(37,22),5;
\end{picture}
  
```

produces



4.4 Gray shade `\Vsetgray`

This macro controls current fill gray shade.

Syntax:

```
\Vsetgray(num);
```

with *num* being a number in range 0–1.0, 0 means “white”, 1 means “black”. Note, that all the subsequent drawing will be affected by this operator (including stroked figures).

Since gray shades are mimicked by black-white patterning, you may get surprising results using this operator together with non-solid line and fill patterns.

Example:

```
\begin{picture}(50,50)
  \Vline(0,0)-(0,50);
  \Vline(0,50)-(50,50);
  \Vline(50,50)-(50,0);
  \Vline(0,0.1)-(50,0.1);
  \Vsetgray(0.5);
  \Vfrectangle(10,10)-(40,40);
\end{picture}
```

produces

