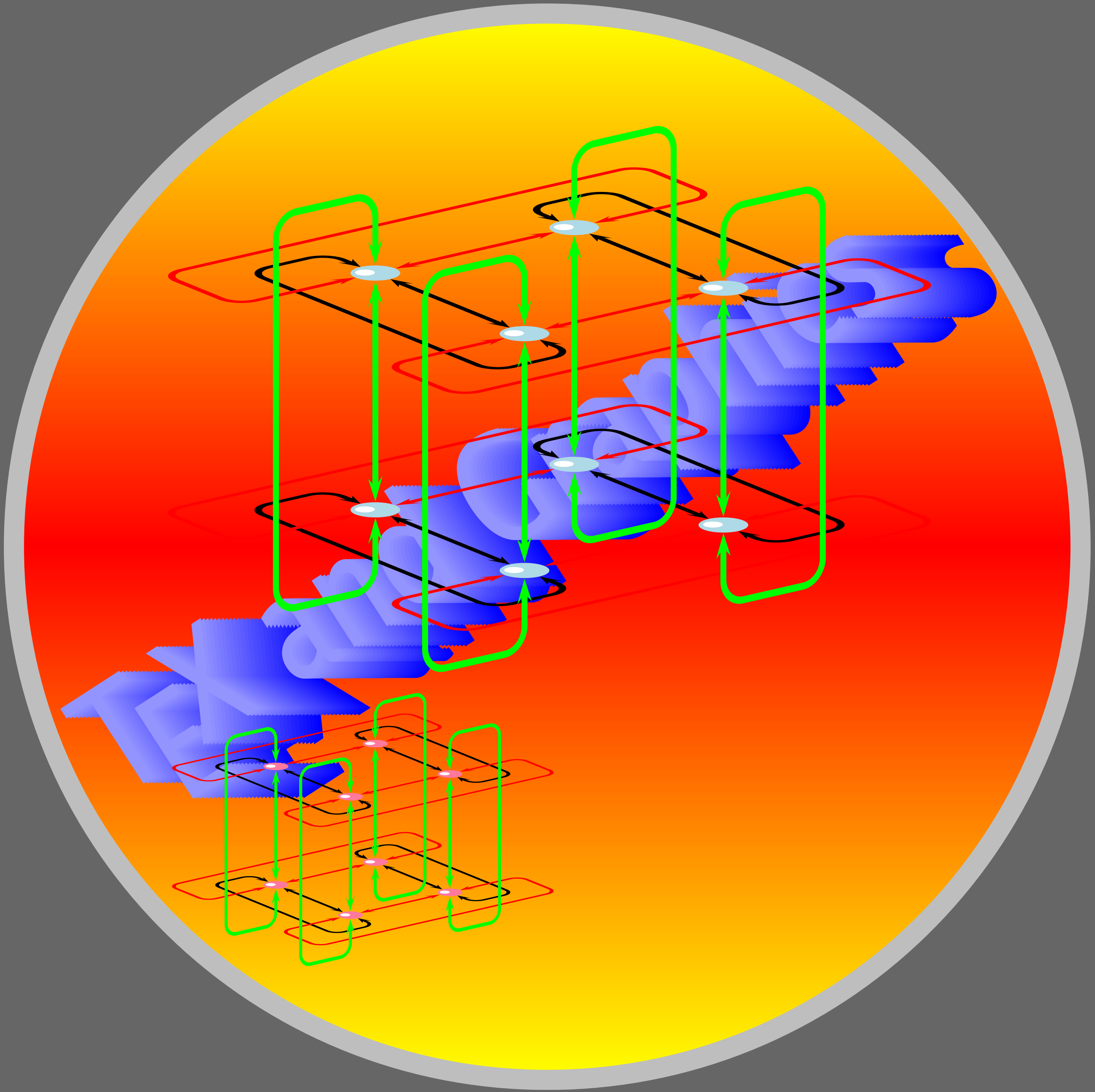

Graphics with PSTricks

Online LaTeX Tutorial

Part II – Graphics

The Indian TEX Users Group
Floor III, SJP Buildings, Cotton Hills
Trivandrum 695014, INDIA
http://www.tug.org.in

1. Graphics with PSTricks

LaTeX has only limited drawing capabilities, while PostScript is a page description language which has a rich set of drawing commands; and there are programs (such as dvips) which translate the `dvi` output to PostScript. So, the natural question is whether one can include PostScript code in a TeX source file itself for programs such as dvips to process after the TeX compilation? This is the idea behind the PSTricks package of Timothy van Zandt. The beauty of it is one need not know PostScript to use it—the necessary PostScript code can be generated by TeX macros defined in the package.

1.1. Getting the points

Any picture is drawn by stringing together appropriate points. How do we specify the points we need? We've a method of specifying each point in a plane using a pair of numbers, thanks to the 17th century French mathematicians Pierre de Fermat and René Descartes. The method is to fix a pair of perpendicular lines (called *axes*) and label each point with the numbers representing its distance from these two points (called *coordinates*) as shown in the figure below:

Note that the meeting point of the axes (called the *origin*) has coordinates (0,0). In order to associate each pair of numbers with a *unique* point, we make the convention that horizontal distances to the left of the origin and vertical distances below the origin are *negative* as illustrated below:

Another fact to note is that the coordinates of points depend on the position of the axes chosen, so that the same point has different pairs of coordinates with respect to different set of axes. This is illustrated in the figure below, where the point which originally had coordinates (3,2) with respect to the axes shown in gray has new coordinates (1,1) with respect to new axes shown in black.

The PSTricks package uses coordinates to specify points to plot and then various other commands to join them.

1.2. Drawing Dots

Now let's see how to draw pictures with PSTricks. The basic package to use is pstricks and so we assume in all the codes given below that this package has been loaded with the command `\usepackage{pstricks}` in the document preamble.

Let's start with the simplest of graphical objects—a single dot. Type in the code below in your document:

```
Look at this dot \psdots(1,0)
```

and TEX compile the document. To produce the PostScript, you'll have to use the dvips program or any other `dvi` to PostScript translator available in your system. With dvips, this done by the command

```
dvips filename -o
```

where *filename* is the name of your file *without any extension* (or with the extension `.dvi`). This creates a PostScript file of the same name but with the extension `.ps` which you can view using a PostScript previewer, such as ghostview. It looks like this:

Some explanations are in order. Evidently the command to draw a dot is `\psdots` followed by the coordinates of the point where the dot is to be placed. But we know that the assignment of coordinates to points (and *vice versa*) makes sense only after fixing the axes. So when we specify coordinates such as `(0,1)` as above, what are the axes used? By default, PSTricks uses the current point in TEX as the origin and horizontal and vertical lines through this point as the axes. Again, the default unit is 1 cm. Thus in the above example, a point is drawn 1 cm. away from the letter `t` in `dot`. This is illustrated in the figure below, where the (invisible) axes are shown in gray.

A single `\psdots` command can be used to plot any number of points. For example, the input

```
Look at these dots \psdots(0,0)(2,0)(1,1)
```

produce the (PostScript) output

Look at these dots   Now suppose we try

```
Look at these dots \psdots(0,0)(2,0)(1,1) forming the vertices
(corners)  of a triangle.
```

the output produced is

What happened? Why were the dots overwritten? What happened actually is that TEX did not reserve any space for the picture (recall that the picture is drawn *after* the TEX compilation) and so the dots were drawn over the text. (if you look closely, you can see that the dots are over the letters). This brings up an important point to be kept in mind: *most of the* PSTricks *commands produce 0-dimensional boxes in* TEX. So, we must ensure that TEX leaves enough space for the pictures to be drawn, by enclosing the picture in a TEX box of suitable size. PSTricks itself provides a convenient method of doing this, in the form of the `pspicture` environment. See how we can modify the previous example:

```
\begin{pspicture}(-0.5,0)(2.5,1)
  \psdots(0,0)(2,0)(1,1)
\end{pspicture}
```

This gives the output

Here the pairs (-0.5,0) and (2.5,1) are the coordinates of the bottom-left and top-right corners of a box which encloses the picture as shown in the figure below:

Look at these dots   forming the vertexes of a triangle.

In fact, the first pair of coordinates is optional and defaults to (0,0). Thus for example,

```
\begin{pspicture}(1,2)...\end{pspicture}
```

is equivalent to

```
\begin{pspicture}(0,0)(1,2) ... \end{pspicture}
```

We can also 'display' the picture by

```
\begin{pspicture}(-0.5,-0.5)(2.5,1.5)
  \psdots(0,0)(2,0)(1,1)
\end{pspicture}
```

This produces

Can you see why the *second* coordinate of the 'box' is changed to -0.5 and 1.5 from its values 0 and 1 in the previous example?

The dots we've been drawing so far are all circular and black. How about square and white dots? Change the input of the previous example as follows:

```
Look at these dots
 \begin{center}
   \begin{pspicture}(-0.5,-0.5)(2.5,1.5)
     \psdots[dotstyle=square](0,0)(2,0)(1,1)
   \end{pspicture}
 \end{center}
forming the vertices of a triangle.
```

We then get the output shown below:

Look at these dots

forming the vertexes of a triangle.

Thus the shape of the dots is controlled by the parameter `dotstyle` and it's to be specified within square brackets after the `\psdots` command. The various possible values of this parameter and the corresponding shape of the dots is shown in the table below:

Also, dots can be scaled using the parameter `dotscale` and rotated using the parameter `dotangle`. For example

```
 \begin{pspicture}(-0.5,-0.5)(2.5,2.5)
  \psdots[dotstyle=+,dotangle=45](0,0)
  \psdots[dotstyle=+,dotscale=1.5,
                dotangle=45](0.5,0.5)
  \psdots[dotstyle=+,dotscale=2,
                dotangle=45](1,1)
  \psdots[dotstyle=+,dotscale=2.5,
                dotangle=45](1.5,1.5)
  \psdots[dotstyle=+,dotscale=3,
               dotangle=45](2,2)
\end{pspicture}
```

gives

Instead of scaling, we can explicitly specify the size of dots. But this we'll discuss in the next section (with a reason, of course).

1.3. Simple Lines

Let's see how we draw lines next. The command is `\psline` with the coordinates of the points to be joined. For example

```
      Look at the line segment below
      \begin{center}
        \begin{pspicture}(0,0)(3.5,2.5)
          \psline(2,1)(3,2)
        \end{pspicture}
      \end{center}
equally slanted to the horizontal and the vertical.
```

gives
Look at the line segment below equally slanted to the horizontal and the vertical.

We can draw dashed or dotted lines using the `linestyle` parameter. Thus

```
\begin{pspicture}(0,0)(2,1)
    \psline(0,0)(2,0)
    \psline[linestyle=dashed](2,0)(1,1)
    \psline[linestyle=dotted](1,1)(0,0)
  \end{pspicture}
```

gives

In this and many of the pictures below, we include a "coordinate grid" for convenience of reference. It is not produced by the code given alongside.

In the `dashed` style, the length of the black and white segments is controlled by the parameter `dash` Thus `dash=3pt 2pt` produces dashed line with black segments of length 3 pt. and white segments of length 2 pt. Thus

```
  \begin{center}
   \begin{pspicture}(-0.5,-0.5)(2.5,1.5)
    \psline[linestyle=dashed,dash=2pt 2pt]
             (0,0)(2,0)
    \psline[linestyle=dashed,dash=2pt 5pt]
             (2,0)(1,1)
    \psline[linestyle=dashed,dash=5pt 5pt]
             (1,1)(0,0)
  \end{pspicture}
 \end{center}
```

gives

The default value of `dash` is 5 pt 3 pt. Again, in the `dotted` style, the distance between dots is controlled by the parameter `dotsep` whose default value is 3 pt.

We can also alter the thickness of the lines by changing the value of the parameter `linewidth` which has a default value of 0.8 pt. Look at the example below:

```
  \begin{center}
    \begin{pspicture}(0,-0.5)(2.5,4.5)
      \psline[linewidth=0.2pt](0,0)(0,2)
      \psline[linewidth=0.4pt](0.5,0)(0.5,2)
      \psline[linewidth=0.6pt](1,0)(1,2)
      \psline[linewidth=0.8pt](1.5,0)(1.5,2)
      \psline[linewidth=1pt](2,0)(2,2)
      \psline[linewidth=1.2pt](2.5,0)(2.5,2)
      \psline[linewidth=1.4pt](3,0)(3,2)
      \psline[linewidth=1.6pt](3.5,0)(3.5,2)
      \psline[linewidth=1.8pt](4,0)(4,2)
      \psline[linewidth=2pt](4.5,0)(4.5,2)
    \end{pspicture}
  \end{center}
```

produces

Table 1.1: Line terminators

1.4. Ends of Lines

Lines can be provided with arrowheads. This is done by the **arrows** parameter

```
  \begin{center}
    \begin{pspicture}(0,-0.5)(2,2.5)
      \psline[arrows=->](0,0)(1,2)
      \psline[arrows=<->](1,1)(2,1)
    \end{pspicture}
   \end{center}
```

produces

Instead of arrowheads, lines can be made to terminate with circles, T-bars and so on, using the parameter **arrows**. The available values of this parameter and the corresponding line terminators are given in the Table 1.1. We can mix and match these terminators as values for the **arrows** parameter such as `*->` or `|-<<`.

Certain terminators are clearly seen only for thick lines. For example

```
\begin{pspicture}(-0.5,-0.5)(2.5,2.5)
  \psline[linewidth=0.1cm,arrows=|-|]
          (1,0)(1,2)
  \psline[linewidth=0.1cm,arrows=|*-|*]
          (2,0)(2,2)
\end{pspicture}
```

gives

To see some other terminators clearly, thicker lines are needed. Thus

```
\begin{pspicture}(-0.5,-0.5)(3.5,2.5)
\psline[linewidth=0.5cm](0,0)(0,2)
\psline[linewidth=0.5cm,arrows=c-c]
        (1,0)(1,2)
\psline[linewidth=0.5cm,arrows=cc-cc]
        (2,0)(2,2)
\psline[linewidth=0.5cm,arrows=C-C]
        (3,0)(3,2)
\end{pspicture}
```

gives

The `arrows` parameter can also be specified as an optional argument within *braces* after the other options (in square brackets). Thus instead of

```
\psline[linestyle=dotted,arrows=<->](0,0)(2,0)
```

we can also write

```
\psline[linestyle=dotted]{<->}(0,0)(2,0)
```

Now is the time to talk of (no, not cabbages and kings) the size of dots. The diameter of a circular dot is 2.5 times the current linewidth plus .5 pt. This can be changed by the parameter `dotsize`. Thus for example

```
\begin{center}
\begin{pspicture}(0,0)(2,2)
  \psdot[linewidth=0.1cm,dotsize=1cm 10](1,1)
\end{pspicture}
\end{center}
```

gives

which is a circular disk of diameters $10 \times 0.1 + 1 = 2$ centimeters. (We'll soon see better method of drawing such disks). The polygonal dots are sized to have the same area as circles. The `dotsize` is made to depend on `linewidth` since dots are often used in conjunction with lines as in `arrows` (and `showpoints` which we will discuss later). Note that the dotsize can be set to any absolute value independent of the linewidth by setting the second number of the `dotsize` parameter to 0.

There are parameters determining the dimensions of the other types of line terminators also, which are given in Table 1.2. In this, *width* refers to a dimension perpendicular to the line and length refers to a dimension in the direction of the line.

PARAMETER	VALUE	DESCRIPTION	DEFAULT VALUE
`dotsize` = *dim num*	*num* × `linewidth` + *dim*	the diameter of a circle or disc	0.5 pt 5
`tbarsize` = *dim num*	*num* × `linewidth` + *dim*	the *width* of a T-bar, square bracket or round bracket	2 pt 5
`bracketlength` = *num*	*number* × *width*	the *length* of a square bracket	0.15
`rbracketlength` = *num*	*number* × *width*	the *length* of a round bracket	0.15

Table 1.2: Parameters for line terminators

The example below illustrates the use of some of these parameters

```
\begin{center}
\begin{pspicture}(-1,-1)(9,4)
\psline[tbarsize=1cm 0,bracketlength=0.5]{[-|}(0,0)(3,0)
\psline[tbarsize=1cm 0]{[-|}(0,3)(3,3)
\psline[tbarsize=1cm 0,rbracketlength=0.5]{(-|}(5,0)(8,0)
\psline[tbarsize=1cm 0]{(-|}(5,3)(8,3)
\end{pspicture}
\end{center}
```

which produces the output below.

Note that the coordinate grid in the picture above is not produced by the given code.

The shape of arrowheads is determined by its *length*, *width* and *inset* and the parameters controlling them are `arrowsize`, `arrowlength` and `arrowinset` as shown in the figure below:

$$\begin{aligned}\texttt{arrowsize} &= \textit{dim num}\\ \text{width} &= \textit{num} \times \texttt{linewidth} + \textit{dim}\\ \text{length} &= \texttt{arrowlength} \times \text{width}\\ \text{inset} &= \texttt{arrowinset} \times \text{length}\end{aligned}$$

The default values of the parameters are

$$\texttt{arrowsize} = 2\,\text{pt } 3 \quad \texttt{arrowlength} = 1.4 \quad \texttt{arrowinset} = 0.4$$

The example below illustrates the effect of changing these parameters.

```
\begin{center}
\begin{pspicture}(0.5,0.5)(5.5,4.5)
\psline[linewidth=2pt,
        arrowsize=2pt 2,
        arrowlength=5,
        arrowinset=0.1]
        {->}(1,1)(4,4)
\psline[linewidth=2pt]
       {->}(2,1)(5,4)
\end{pspicture}
\end{center}
```

We can also draw "double lines" by setting the parameter `doubleline` to `true` (by default, it's `false`). For example

```
\begin{center}
\begin{pspicture}(-0.5,-0.5)(2.5,2.5)
\psline[linewidth=0.06,
        doubleline=true,
        doublesep=0.05,
        (0,0)(2,2)
\end{pspicture}
\end{center}
```

gives

1.5. Bent Lines and Polygons

As in the case of \psdots we can draw multiple lines with a single \psline command. For example,

```
\begin{center}
\begin{pspicture}(0,0)(5,2)
\psline(1,1)(2,2)(3,1)(4,2)(5,1)
\end{pspicture}
\end{center}
```

gives

Note that the coordinate grid is not produced by the code given alongside.

The corners in the above picture can be rounded by giving a positive value to the linearc parameter which has default value 0 pt. It is actually the radius of the arc drawn at the corners. Thus

```
\begin{center}
\begin{pspicture}(0,0)(5,2)
\psline[linearc=0.25]%
       (1,1)(2,2)(3,1)(4,2)(5,1)
\end{pspicture}
\end{center}
```

gives

Now change the value of linearc to 0.5 in the above code and see what happens.

Polygons can be drawn with \psline by taking the first and the last points same. For example

```
\begin{center}
\begin{pspicture}(0,0)(5,3)
\psline(1,1)(2,2)(5,2)(4,1)(1,1)
\end{pspicture}
\end{center}
```

gives

We can also use the command `\pspolygon` to draw polygons. Here, we need not repeat the starting point as in `\psline`. Thus in the last example above, the parallelogram could also be drawn by the command

```
\pspolygon(1,1)(2,2)(5,2)(4,1)
```

instead of the command

```
\psline(1,1)(2,2)(5,2)(4,1)(1,1)
```

The `\pspolygon` command also has a "starred" version which draws a "filled up" polygon. For example

```
\begin{center}
\begin{pspicture}(0,0)(5,3)
\pspolygon*(1,1)(2,2)(5,2)(4,1)
\end{pspicture}
\end{center}
```

gives

For drawing rectangles, there's a simpler command `\psframe` in which we need only specify the bottom-left and top-right coordinates. There's also a `\psframe*` command for a filled-up version. For example,

```
\begin{center}
\begin{pspicture}(0,0)(6,4)
\psframe(1,1)(3,3)
\psframe*(1,1)(2,2)
\psframe*(2,2)(3,3)
\end{pspicture}
\end{center}
```

gives

The corners of a frame can also be rounded. The parameter to set is `framearc`. If we set `framearc=`*number*, then the radius of the rounded corners is half the *number* times the width or height of the frame, whichever is less. Thus

```
\begin{center}
\begin{pspicture}(-0.5,0.5)(5.5,3.5)
\psframe[framearc=0.5](0,0)(5,3)
\psframe[framearc=0.5](1,1)(4,2)
\end{pspicture}
\end{center}
```

gives

Note that the corners of the larger rectangle are more rounded, as should be obvious from the definition of the `framearc` parameter. The radius of the corners can be made he same by setting the parameter `cornersize` to `absolute` (its default setting is `relative`) and then setting the radius using the `linearc` parameter as in the example below:

```
\begin{center}
\begin{pspicture}(-0.5,-0.5)(5.5,3.5)
\psframe[cornersize=absolute,%
         linearc=0.5](0,0)(5,3)
\psframe[cornersize=absolute,%
         linearc=0.5](1,1)(4,2)
\end{pspicture}
\end{center}
```

There are also commands to draw isoceles triangles (that is, triangles in which two sides are equal) and rhombuses (diamonds). The command

\pstriangle($(x,y)(b,h)$)

draws an isoceles triangle with its base horizontal, the mid-point of its base at (x, y), length of base b and height h while the command

\psdiamond($(x,y)(d_1,d_2)$)

draws a rhombus with its diagonals along the horizontal and the vertical, which meet (x, y) and have lengths $2d_1$ and $2d_2$. Thus

```
\begin{center}
\begin{pspicture}(0,0)(5,5)
\pstriangle(1,0)(2,3)
\pstriangle*(4,1)(2,1.732)
\psdiamond(3,4)(2,1)
\end{pspicture}
\end{center}
```

gives

So far we've been drawing only straight lines (except for smoothing some corners). We'll discuss curves in the next few sections.

1.6. Simple Curves

Circles, ellipses, circular arcs and so on can be easily drawn in PSTricks, using preset commands. Let's start with circles. The command is \pscircle (what else?) and we've to specify the coordinates of the center and the length of the radius. Recall that the default unit is centimeter, so that to produce a circle of radius 0.5 cm centered at (2,1), we write

```
\pscircle(2,1){0.5}
```

Since a circle is only a curved "line", various line parameters discussed earlier can also be used. There is also a starred version \pscircle* which gives a "solid" circle. See the example below:

```
\begin{center}
 \begin{pspicture}(-1,-1)(3,8)
  \pscircle*(1,0.25){0.25}
  \pscircle[linewidth=0.33]%
          (1,1){0.5}
  \pscircle[linewidth=0.25]%
          (1,2.25){0.75}
  \pscircle(1,4){1}
  \pscircle[linestyle=dotted]%
          (1,6.25){1.25}
 \end{pspicture}
\end{center}
```

Pieces of circles can also be easily drawn. For example, the command \psarc draws a circular arc of specified center and radius from a given *angle* to another going *counterclockwise*. Note that the angles are measured from the horizontal. In the example below, we show the radii and the angles in gray along with the grid. (note that these are not produced by the given code).

```
\begin{center}
 \begin{pspicture}(-1,-1)(3,3)
  \psarc(0,0){3}{30}{60}
 \end{pspicture}
\end{center}
```

There's also a starred version `\psarc*` which draws a solid "segment" of a circle. For example,

```
\begin{center}
 \begin{pspicture}(-2,-2)(2,2)
  \psframe(-1,-1)(1,1)
  \psarc*(-1,-1){1}{0}{90}
  \psarc*(1,-1){1}{90}{180}
  \psarc*(1,1){1}{180}{270}
  \psarc*(-1,1){1}{270}{360}
 \end{pspicture}
\end{center}
```

gives

While making a picture containing circular arcs, it may sometimes be convenient to "see" the center and radii. If the parameter `showpoints` is set to `true` (its default value is `false`), then the command `\psarc` (or `\psarc`) draws dashed lines from the center to the extremities of the arc. (This setting can be used with other commands also, where it will draw appropriate control points or lines). See the example below:

```
\begin{center}
 \begin{pspicture}(0,0)(3,3)
  \psarc[showpoints=true]%
       (1,1){2}{30}{60}
  \end{pspicture}
\end{center}
```

If we want to draw an arc with its bounding radii, we can use the \pswedge command. The starred version \pswedge* draws a solid sector as shown in the example below:

```
\begin{center}
 \begin{pspicture}(-1.5,-1.5)(1.5,1.5)
  \pswedge(0,0){1}{90}{360}
  \pswedge*(0.1,0.1){1}{0}{90}
 \end{pspicture}
\end{center}
```

The line terminators discussed earlier can be used with arcs also. If we want to show the angle between two (thick) intersecting lines using an arc with an arrowhead, we'd like to have the tip of the arrow would *just* touch the line. For this, we can use the parameters arcsepA and arcsepB. If we set arcsepA=*dim*, then the first angle in the \psarc command would be adjusted so that the arc would just touch a line of width *dim* from the center of the arc in the direction of this angle. The parameter arcsepB makes a similar adjustment in the second angle. The parameter arcsep adjusts both the angles. The example below illustrates this.

```
\begin{center}
 \begin{pspicture}(-2,0)(2,2)
  \psline[linewidth=2pt]%
        (2,2)(0,0)(-2,2)
  \psarc[arcsepB=2pt]{->}%
        (0,0){1}{45}{135}
 \end{pspicture}
\end{center}
```

To see the difference, try the same code without the setting of arcsepB.

An ellipse is a sort of a stretched circle and can be drawn much the same way as s circle. The command is \psellipse and we have to specify the center

The Indian TEX Users Group
Floor III, SJP Buildings, Cotton Hills
Trivandrum 695014, INDIA
http://www.tug.org.in

and half the width and height (technically, the "semi-major" and "semi-minor" axes). Thus to draw an ellipse centered at (1,1) with width width 4 cm and height 2 cm, we type

```
\begin{center}
 \begin{pspicture}(-1,0)(3,2)
  \psellipse(1,1)(2,1)
 \end{pspicture}
\end{center}
```

which gives

There's also a `\psellipse*` which, as you've probably guessed, draws a solid black ellipse.

```
\begin{center}
 \begin{pspicture}(-2,-1)(2,1)
  \psellipse(0,0)(2,1)
  \psellipse*(0,0)(0.5,1)
 \end{pspicture}
\end{center}
```

Another curve for which a preset command is available is a parabola (the path of a stone thrown at an angle, for example). It is drawn by the command `\parabola` (surprise!). We must specify the starting point and the maximum or minimum. As usual, we have a `\parabola*` also. Thus

```
\begin{center}
 \begin{pspicture}(0,0)(4,2)
  \parabola(0,0)(2,2)
  \parabola*(1,1.5)(2,0)
 \end{pspicture}
\end{center}
```

gives