POSTSCRIPT®

THE DISPLAY POSTSCRIPT® SYSTEM: COORDINATE SYSTEM and the STROKE ADJUST APPLICATION

Technical Note #5053

March 6, 1990
PostScript® Developer Support Group

Adobe Systems Incorporated
1585 Charleston Road PO Box 7900
Mountain View, CA 94039-7900
(415) 961-4400

PN LPS5053

THE DISPLAY POSTSCRIPT® SYSTEM: COORDINATE SYSTEM and the STROKE ADJUST APPLICATION

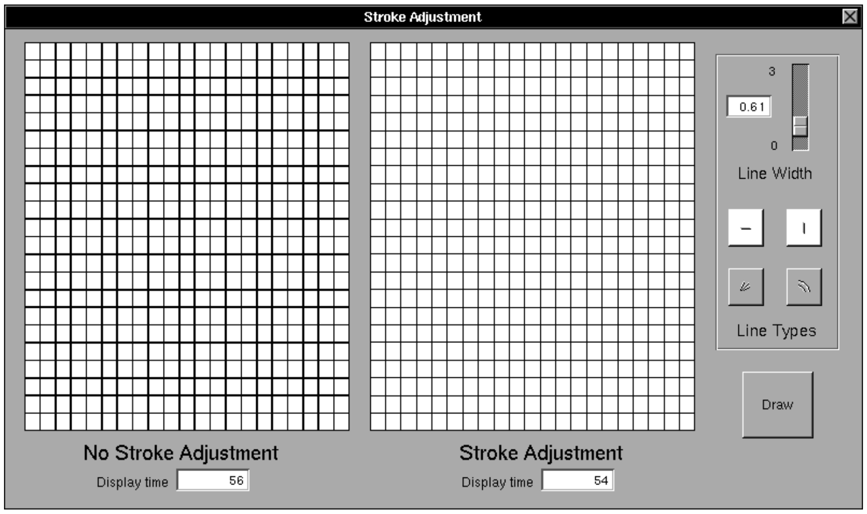
Technical Note #5053

March 6, 1990
PostScript® Developer Support Group
(415) 961-4111

1. INTRODUCTION

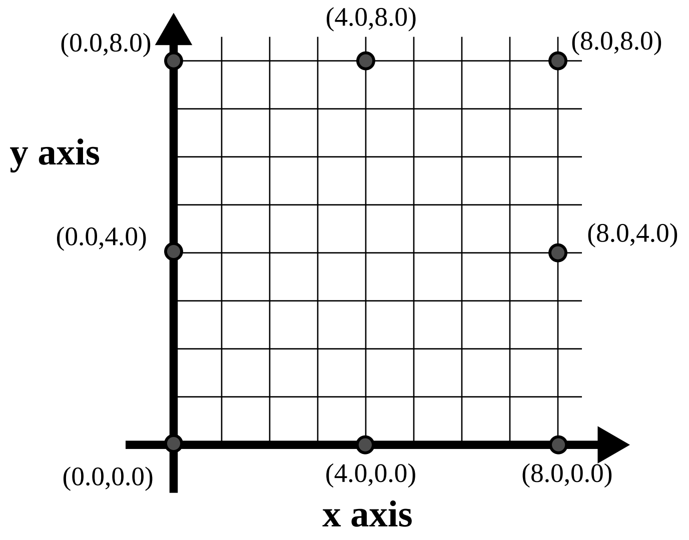
One of the key features of the PostScript language system is its device independence. This technical note will show how this independence is possible by explaining the PostScript language coordinate system. In addition to device independence, the PostScript language coordinate system provides the capability to perform linear distortions such as scaling and rotating text and graphics. This capability is what allows scalable fonts, text on a curve and intricate objects rotated at any angle.

The meaning of a path and its relation to pixels will also be highlighted. The **StrokeAdjust** application is used to show the effect that stroke adjustment has at the pixel level. Although not an issue on high resolution devices, stroke adjustment can make a noticeable difference on low resolution devices. The PostScript interpreter can adjust lines to a uniform width despite the difference in line placement with regard to pixel boundary. This capability, termed *automatic stroke adjustment*, is the default behavior for the Display PostScript system and can be turned on and off. Automatic stroke adjustment is not available in current PostScript interpreters in current printer products, so some applications may need to include stroke adjustment procedures in their printer driver when printing on low resolution devices. (A 300 dpi printer is considered a low resolution device.) These procedures are contained in the *Printing* section.



2. CARTESIAN COORDINATE SYSTEM

All drawing and creation of paths is performed using x and y coordinate points within a Cartesian coordinate system. This coordinate system has an origin and x and y axes. The common representation of a coordinate system is shown below. The origin is (0.0,0.0), the y axis increases from bottom to top and the x axis increases from left to right.



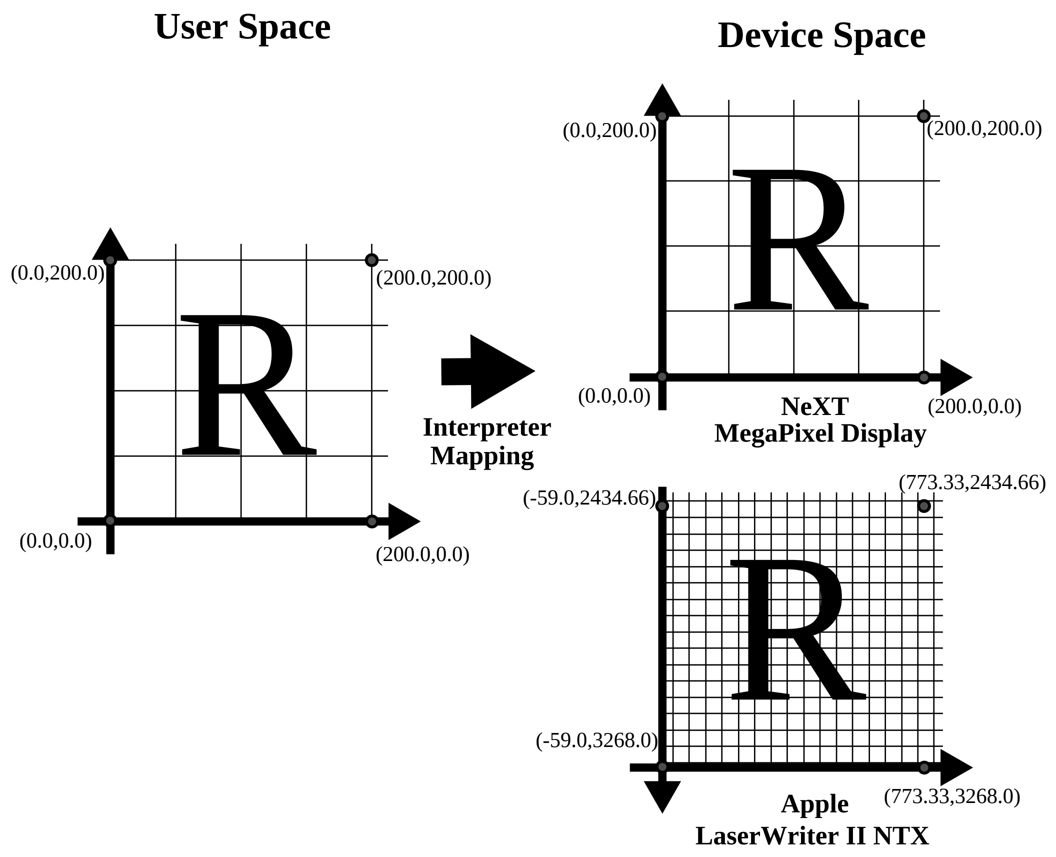
This figure is the standard representation for a coordinate system. Other representations, such as the y axis increasing from top to bottom instead of bottom to top, are just as legitimate. Whatever type is used does not really matter as long as the drawing commands match the particular origin location, axes orientation and unit length.

3. USER SPACE AND DEVICE SPACE

The PostScript language coordinate system is actually made up of two coordinate systems. One system is called the *user space* and is used for all the drawing instructions. This system has a look that is consistent across all devices. Any point drawn in this system will appear in the same location in user space in one device as in user space in another. The second coordinate system does not share this property. A point in this system on one device can, and most often will, appear in quite a different location on a different device. This second coordinate system is called the *device space*. The device space is used to address actual pixels within the imageable area. Since devices vary in resolution and imaging direction, different device space coordinate system representations are necessary, containing different origins, axes orientations and unit lengths.

The application developer does not have to be aware of the differences in device spaces because the PostScript interpreter performs the translation from the user space to the device space automatically. As a result, the application does not have to be concerned with printing or displaying to different types of devices. One set of drawing instructions drawn with respect to a single coordinate system is sufficient. The interpreter will handle the translation from user space to device space for a given device, deciding which pixels are turned on and which ones are not.

In the drawing below, the user space is on the left and two different device spaces are shown on the right. The first device space is from the NeXT™ MegaPixel Display. Its device space coordinate system happens to match the user space coordinate system. The second space is from the Apple LaserWriter® II NTX. Its origin and unit length are different from the user space. In both cases, the interpreter manages the mapping from user space to device space.



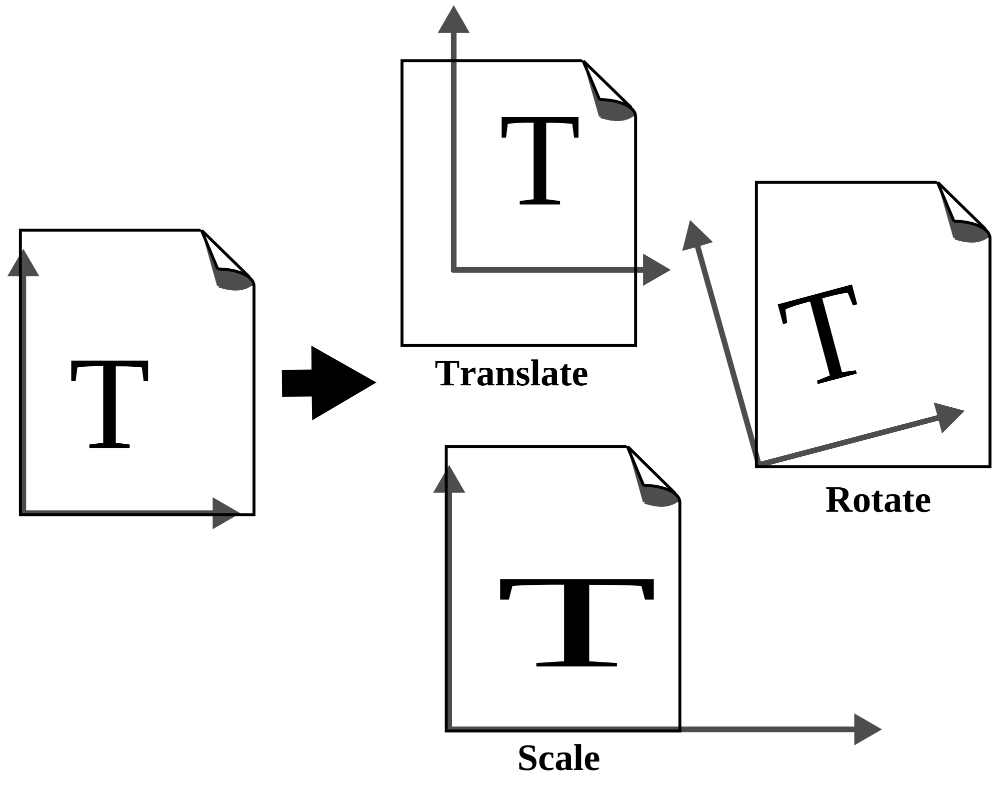
The standard or default user space has the origin in the lower left corner of the page with the positive y axis extending vertically upward and with the positive x axis extending horizontally to the right. The length of a unit along the x axis and along the y axis is 1/72 of an inch.[1] The NeXT MegaPixel Display, an Apple LaserWriter II NTX and even a View object in the NeXT Application Kit have the same default user space. In each case, the dimensions of the display or page (the imageable area) may be different but the origin, axes orientation and unit size are all the same. (Well almost the same: 72 units on the NeXT monitor does not equal 1 inch. The monitor reduces 72 units to 72/92 of an inch. This reduction has no real bearing on display because it is constant across all drawing. It's not even noticed until someone holds a ruler up to a ruler in an application and says, "Wait. One inch at 100% magnification is really only 72/92 of an inch".)

1. The unit size, 1/72 of an inch, is very close to the size of a printer's point (1/72.27 inch), which is a standard measuring unit in the printing industry.

4. TRANSFORMATIONS

The conversion from one coordinate system to the other is specified by a transformation matrix. This matrix specifies how the x and y values for a point in one coordinate space are mapped into the x and y values of the corresponding point in another coordinate space. The matrix used to transform coordinates from the user space to the device space is called the *current transformation matrix* (CTM). The CTM can be altered and changed to cause a different mapping from user space to device space.

A convenient way to understand the alteration of the CTM is to visualize the user space as changing either by moving the origin to a different position on the display or page (translating), altering the angle of the axes (rotating) or changing the length of a unit (scaling). The PostScript language system contains specific operators to perform these types of transformations - **translate**, **rotate** and **scale**.



The **translate** operator moves the origin to a different location on the page leaving the orientation of the axes and the unit size unchanged. The **rotate** operator rotates the coordinate system about the origin by the specified angle. The **scale** operator alters the units of the axes independent of each other. For example, the coordinate system can be scaled so that a unit in the x direction is actually twice the size of a unit in the y direction (similar to what is shown in the Scale example above).

There are a variety of uses for these types of transformations. A single arrow description can be used to display at any angle by rotating the user space before displaying the arrow. The origin of the user space can be moved to the upper left corner of the page and then the user space scaled so the y axes is positive towards the bottom of the page. This setup facilitates the placement of text and, in fact, the **Text** object in the Application Kit uses this orientation. A single circle description can be used to display any type of oval by rotating and scaling the circle before display. These types of transformations provide a number of ways to simplify drawing and produce many interesting results.

For an explanation of the matrix mathematics involved with transformations of the CTM, please refer to the Coordinate Systems and Transformations section in the PostScript Language Reference Manual (commonly referred to as the *Red Book*).

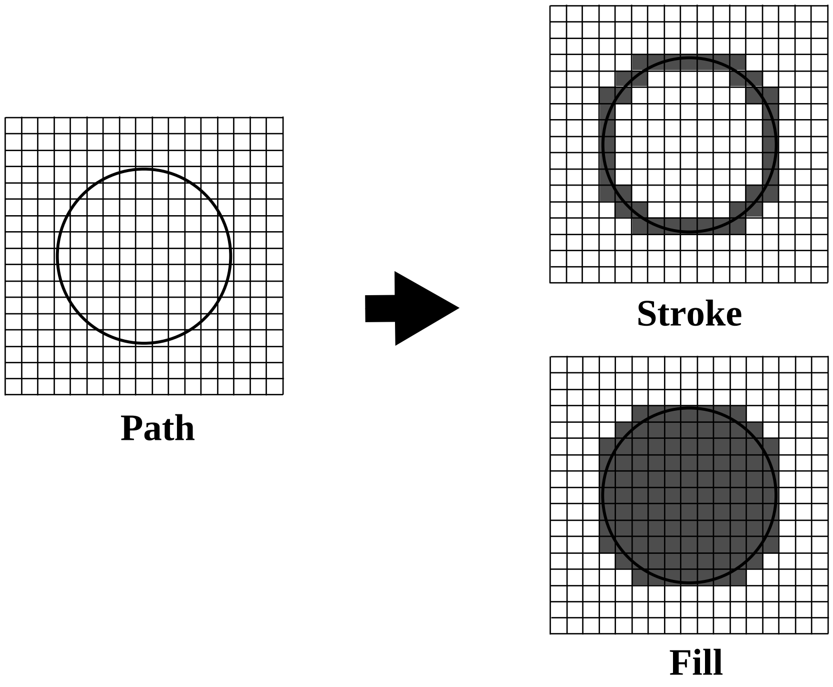
5. PATHS

A path in the PostScript language is a set of line, arc and Bezier curve segments. These segments can be continuous (a single **moveto**) or disjoint (multiple **moveto**'s interspersed within the set). The segments within a path have no width and the shape formed by the path has no fill until the path is rendered with one of the painting operators such as **stroke** or **fill**. At the time of rendering, the values for the current graphic state variables are used to determine what pixels to turn on. Before the execution of the **stroke** or **fill**, the line width or color settings have no bearing on the path. Only at the time of rendering are these values taken into account.

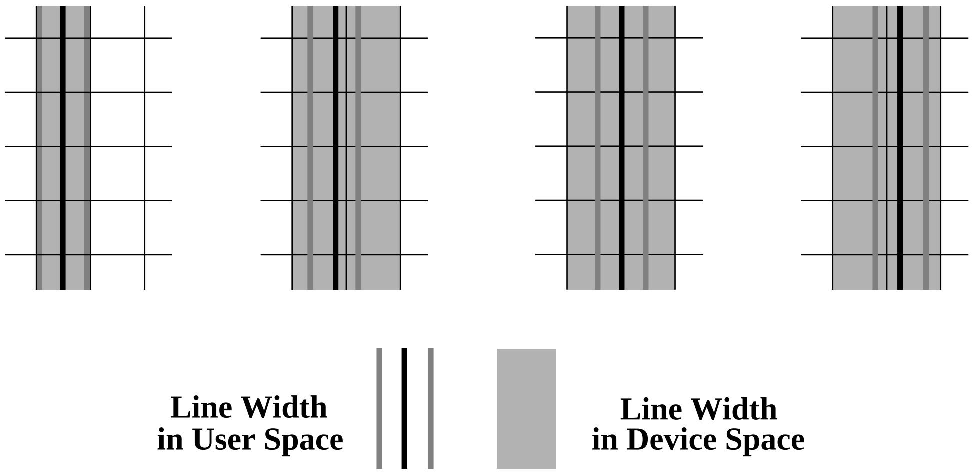
Filling a path fills the inside of the path with the current fill rule. The fill does not extend outside the path, although any pixel that the path passes through will be colored as well.

Stroking a path extends the width of the path by the current line width setting. One half of the line width extends on each side of the path. The line width corresponds to points in the current user space so that any scaling operation will have a corresponding effect on the line width.

The examples below show a path and its subsequent stroking and filling in some unspecified device space with an unspecified line width. The path is simply a mathematical representation of an arc. The stroked path and the filled path are the pixel representations in device space.



The placement of the path within a pixel is an important factor in determining what pixels to turn on when stroking. The example below shows four instances where a vertical path falls at different locations within a pixel column. In the first case, a line width setting equivalent to one pixel in device space will result in a one pixel line. In the second, third and fourth cases, a line width setting equal to one pixel will result in a two pixel line. Only in the first case, does the line width not extend across pixel boundaries. In the others, it does.



6. STROKE ADJUSTMENT

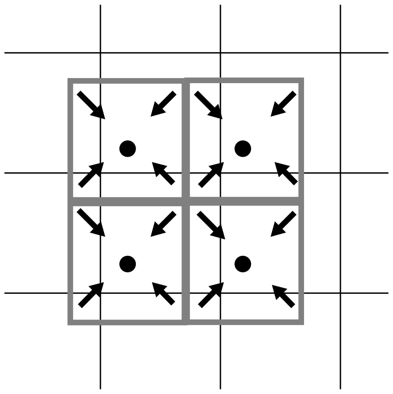
For high resolution devices, the placement does not really matter because the pixel size is so small that an additional pixel here and there is inconsequential. For lower resolution devices such as a display or a 300 dpi printer, this issue can be important. The **StrokeAdjust** application shows the effect that random line position has on the actual pixel width of lines. The Display PostScript system can perform an automatic adjustment to paths that will render them with uniform widths. This capability is called *automatic stroke adjustment* and it works by adjusting the placement of every path so that the paths fall at precise locations within pixels. The stroke adjustment is turned on by default but it can be turned off with the **setstrokeadjust** operator.

7. PRINTING

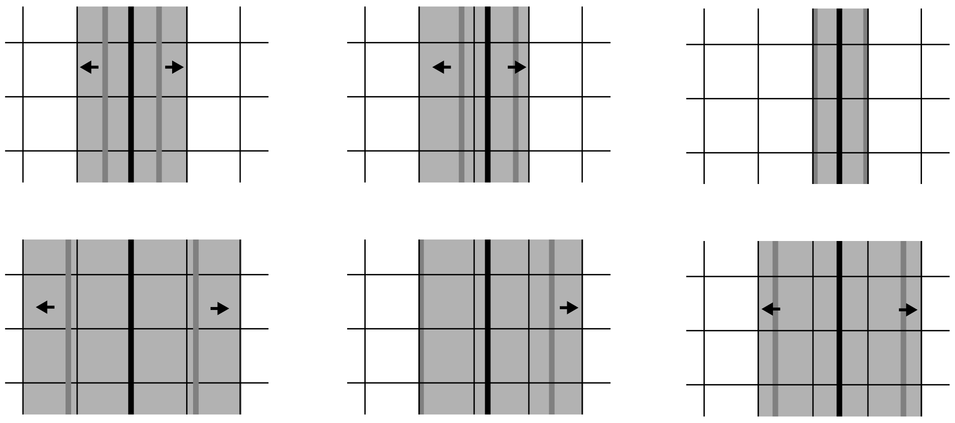
The Display PostScript system can perform the stroke adjustment automatically. The default setting is on and once set, no additional effort is necessary. When printing to a device that does not have the automatic stroke adjustment capability, however, some additional work may be required, depending on the nature of the application. The Display PostScript system uses a very sophisticated algorithm for performing the automatic stroke adjustment that would be difficult to emulate in the PostScript language. Though, a fairly close implementation can be described as follows:

- convert the user space coordinates of each point to device space coordinates
- subtract one quarter of a pixel from each device space value
- round to the nearest whole device pixel
- add one quarter of a pixel
- convert from device space coordinates back to user space coordinates

Stroke Adjustment rounds the endpoints of lines or curves to fall at precise places in device space. The optimal position is 1/4 pixel length in from the bottom left corner.



The repositioning of the endpoints at the one quarter mark within a pixel seems an unusual location. But if the pixel *boundary* were chosen then all the lines would be of *even* pixel width. Likewise, if the *midpoint* of each pixel were chosen, all the lines would be of *odd* pixel width. The choice of one quarter selects one of the optimal locations for equal distribution of even and odd pixel widths. The drawings below try to clarify this issue.



Rounded to Pixel Boundary
Even Pixel Widths

Rounded to Quarterpoint
Even/Odd Pixel Widths

Rounded to Midpoint
Odd Pixel Widths

The procedures below can be used to perform the algorithm described above. The idea is to replace the current path construction operators with modified ones to perform the rounding. The comments indicate the operands of the particular operator.

```
/setstrokeadjust where
{ % stroke adjustment is available so make sure it is on
  pop true setstrokeadjust
}
{ % stroke adjustment is not available so redefine the
  % path construction ops

  /m
  { % x y m - moveto
    transform
    .25 sub round .25 add exch
    .25 sub round .25 add exch
    itransform
    moveto
  } bind def

  /l
  { % x y l - lineto
    transform
    .25 sub round .25 add exch
    .25 sub round .25 add exch
    itransform
    lineto
  } bind def
  /c
  { % $x_1$ $y_1$ $x_2$ $y_2$ $x_3$ $y_3$ c - curveto
    transform
    .25 sub round .25 add exch
    .25 sub round .25 add exch
    itransform
    curveto
  } bind def

  /rm
  { % dx dy rm - rmoveto
    transform
    round exch
    round exch
    itransform
    rmoveto
  } bind def

  /rl
  { % dx dy rl - rlineto
    transform
    round exch
    round exch
    itransform
    rlineto
  } bind def

  /rc
  { % $dx_1$ $dy_1$ $dx_2$ $dy_2$ $dx_3$ $dy_3$ rc - rcurveto
    transform
    round exch
    round exch
    itransform
    rcurveto
  } bind def
} ifelse
```

The **curveto** and **rcurveto** redefinitions only adjust the endpoints. They do not reposition the control points. It is not necessary to adjust the control points except for the case where the curve is a line or, in other words, where the control points lie on the line segment between the start and end points of the curve. In this case, either the application may want to turn the curve into a line segment or create a special **curveto** or **rcurveto** definition that repositions the control points as well.

Note: *The font handling mechanism performs this stroke adjustment positioning automatically for Type 1 fonts on all PostScript interpreters. As a result, the* ***moveto*** *operator used to set the current point before a show operation does* ***not*** *need to be stroke adjusted.*

8. SUMMARY

The device independence of the PostScript language coordinate system is an important feature for application developers. A common coordinate system, the user space, is used for drawing on any device. The PostScript interpreter handles the conversion from the user space to the device coordinate system, the device space. This conversion process uses a matrix called the current transformation matrix or CTM to map a coordinate in one space to the corresponding coordinate in the other space. Applications can draw in a common coordinate system with the PostScript interpreter handling the particulars of the actual device. One page description will suffice for all devices.

In addition to providing device independence, the user space/device space model permits user space transformations such as translating of the origin, rotating of the axes and scaling of the unit lengths of the axes. These operations can have a variety of benefits such as using a single path representation to image in several locations at a different rotations and even in different x-y proportions.

A path is a set of lines, arcs and curves that can be filled or stroked. On low resolution devices, the location of a path within a pixel can cause lines with the same line width to turn on a different number of pixels. This behavior can cause unattractive results when a grid or other symmetrical group of lines is displayed. The Display PostScript system can adjust the location automatically to produce lines with uniform pixel width. This adjustment is, by default, turned on but it can be turned off with the **setstrokeadjust** operator. When outputting to PostScript interpreters without automatic stroke adjustment, application developers might want to redefine the path construction operators to manually adjust the path placement.