

```
stream
01 00 >
endstream
endobj

35 0 obj                                % Second graphics state parameter dictionary
  << /Type /ExtGState
    /OP false
    /TR 36 0 R
  >>
endobj

36 0 obj                                % Second transfer function
  << /FunctionType 0
    /Domain [0.0 1.0]
    /Range [0.0 1.0]
    /Size 21
    /BitsPerSample 8
    /Length 63
    /Filter /ASCIIHexDecode
  >>
stream
FF CE A3 7C 5B 3F 28 16 0A 02 00 02 0A 16 28 3F 5B 7C A3 CE FF >
endstream
endobj
```

4.4 Path Construction and Painting

Paths define shapes, trajectories, and regions of all sorts. They are used to draw lines, define the shapes of filled areas, and specify boundaries for clipping other graphics. The graphics state includes a *current clipping path* that defines the clipping boundary for the current page. At the beginning of each page, the clipping path is initialized to include the entire page.

A path is composed of straight and curved line segments, which may connect to one another or may be disconnected. A pair of segments are said to *connect* only if they are defined consecutively, with the second segment starting where the first one ends. Thus the order in which the segments of a path are defined is significant. Nonconsecutive segments that meet or intersect fortuitously are not considered to connect.

A path is made up of one or more disconnected *subpaths*, each comprising a sequence of connected segments. The topology of the path is unrestricted: it may be concave or convex, may contain multiple subpaths representing disjoint areas, and may intersect itself in arbitrary ways. There is an operator, **h**, that explicitly connects the end of a subpath back to its starting point; such a subpath is said to be *closed*. A subpath that has not been explicitly closed is *open*.

As discussed in Section 4.1, “Graphics Objects,” a path object is defined by a sequence of operators to construct the path, followed by one or more operators to paint the path or to use it as a clipping boundary. PDF path operators fall into three categories:

- *Path construction operators* (Section 4.4.1) define the geometry of a path. A path is constructed by sequentially applying one or more of these operators.
- *Path-painting operators* (Section 4.4.2) end a path object, usually causing the object to be painted on the current page in any of a variety of ways.
- *Clipping path operators* (Section 4.4.3), invoked immediately prior to a path-painting operator, cause the path object also to be used for clipping of subsequent graphics objects.

4.4.1 Path Construction Operators

A page description begins with an empty path and builds up its definition by invoking one or more path construction operators to add segments to it. The path construction operators may be invoked in any sequence, but the first one invoked must be **m** or **re** to begin a new subpath. The path definition concludes with the application of a path-painting operator such as **S**, **f**, or **b** (see Section 4.4.2, “Path-Painting Operators”); this may optionally be preceded by one of the clipping path operators **W** or **W*** (Section 4.4.3, “Clipping Path Operators”). Note that the path construction operators in themselves do not place any marks on the page; only the painting operators do that. A path definition is not complete until a path-painting operator has been applied to it.

The path currently under construction is called the *current path*. In PDF (unlike PostScript), the current path is *not* part of the graphics state and is *not* saved and restored along with the other graphics state parameters. PDF paths are strictly internal objects with no explicit representation. Once a path has been painted, it is

no longer defined; there is then no current path until a new one is begun with the **m** or **re** operator.

The trailing endpoint of the segment most recently added to the current path is referred to as the *current point*. If the current path is empty, the current point is undefined. Most operators that add a segment to the current path start at the current point; if the current point is undefined, they generate an error.

Table 4.9 shows the path construction operators. All operands are numbers denoting coordinates in user space.

TABLE 4.9 Path construction operators

OPERANDS	OPERATOR	DESCRIPTION
$x\ y$	m	Begin a new subpath by moving the current point to coordinates (x, y) , omitting any connecting line segment. If the previous path construction operator in the current path was also m , the new m overrides it; no vestige of the previous m operation remains in the path.
$x\ y$	l (lowercase L)	Append a straight line segment from the current point to the point (x, y) . The new current point is (x, y) .
$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$	c	Append a cubic Bézier curve to the current path. The curve extends from the current point to the point (x_3, y_3) , using (x_1, y_1) and (x_2, y_2) as the Bézier control points (see “Cubic Bézier Curves,” below). The new current point is (x_3, y_3) .
$x_2\ y_2\ x_3\ y_3$	v	Append a cubic Bézier curve to the current path. The curve extends from the current point to the point (x_3, y_3) , using the current point and (x_2, y_2) as the Bézier control points (see “Cubic Bézier Curves,” below). The new current point is (x_3, y_3) .
$x_1\ y_1\ x_3\ y_3$	y	Append a cubic Bézier curve to the current path. The curve extends from the current point to the point (x_3, y_3) , using (x_1, y_1) and (x_3, y_3) as the Bézier control points (see “Cubic Bézier Curves,” below). The new current point is (x_3, y_3) .
—	h	Close the current subpath by appending a straight line segment from the current point to the starting point of the subpath. This operator terminates the current subpath; appending another segment to the current path will begin a new subpath, even if the new segment begins at the endpoint reached by the h operation. If the current subpath is already closed, h does nothing.

`x y width height re` Append a rectangle to the current path as a complete subpath, with lower-left corner (x, y) and dimensions *width* and *height* in user space. The operation

`x y width height re`

is equivalent to

```
x y m
(x + width) y l
(x + width) (y + height) l
x (y + height) l
h
```

Cubic Bézier Curves

Curved path segments are specified as *cubic Bézier curves*. Such curves are defined by four points: the two endpoints (the current point P_0 and the final point P_3) and two *control points* P_1 and P_2 . Given the coordinates of the four points, the curve is generated by varying the parameter t from 0.0 to 1.0 in the following equation:

$$R(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t) P_2 + t^3 P_3$$

When $t = 0.0$, the value of the function $R(t)$ coincides with the current point P_0 ; when $t = 1.0$, $R(t)$ coincides with the final point P_3 . Intermediate values of t generate intermediate points along the curve. The curve does not, in general, pass through the two control points P_1 and P_2 .

Cubic Bézier curves have two desirable properties:

- The curve can be very quickly split into smaller pieces for rapid rendering.
- The curve is contained within the convex hull of the four points defining the curve, most easily visualized as the polygon obtained by stretching a rubber band around the outside of the four points. This property allows rapid testing of whether the curve lies completely outside the visible region, and hence does not have to be rendered.

The Bibliography lists several books that describe cubic Bézier curves in more depth.

The most general PDF operator for constructing curved path segments is the **c** operator, which specifies the coordinates of points P_1 , P_2 , and P_3 explicitly, as shown in Figure 4.8. (The starting point, P_0 , is defined implicitly by the current point.)

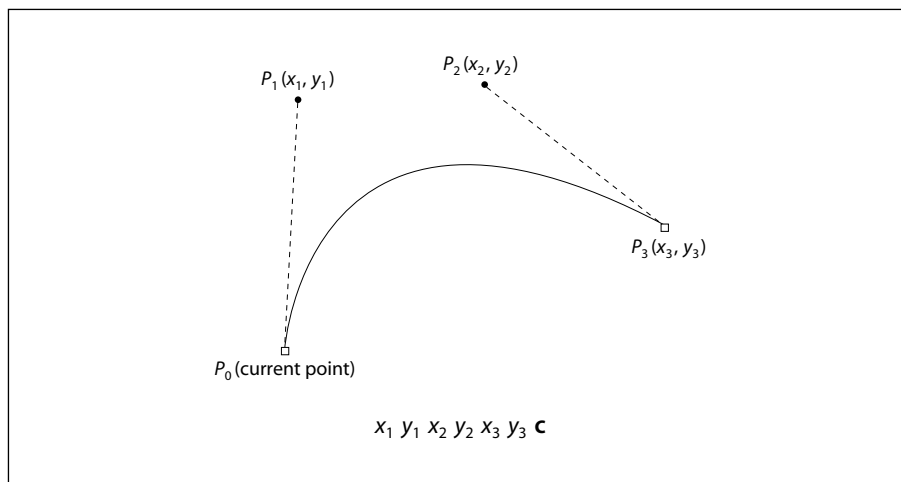


FIGURE 4.8 Cubic Bézier curve generated by the **c** operator

Two more operators, **v** and **y**, each specify one of the two control points implicitly (see Figure 4.9). In both of these cases, one control point and the final point of the curve are supplied as operands; the other control point is implied, as follows:

- For the **v** operator, the first control point coincides with initial point of the curve.
- For the **y** operator, the second control point coincides with final point of the curve.

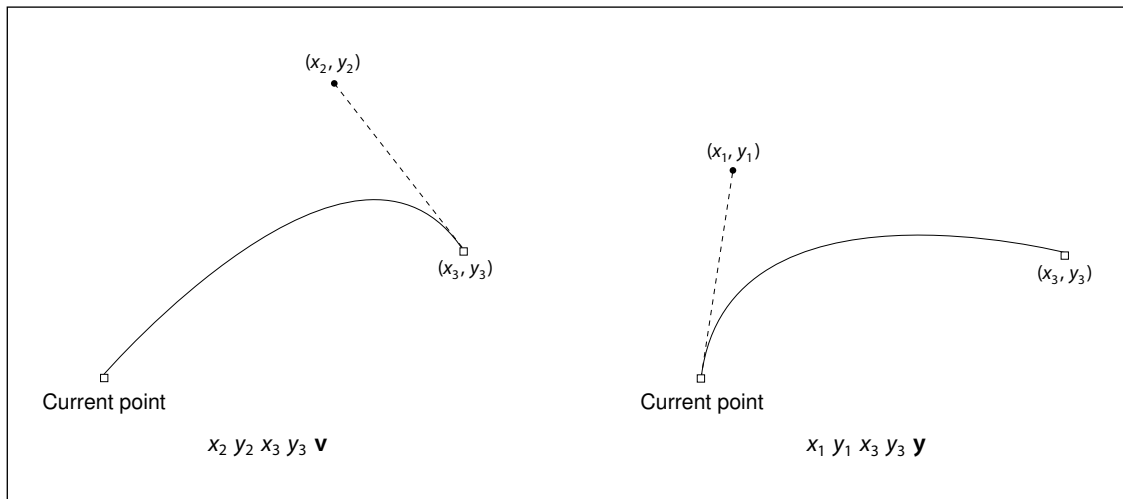


FIGURE 4.9 Cubic Bézier curves generated by the **v** and **y** operators

4.4.2 Path-Painting Operators

The path-painting operators end a path object, causing it to be painted on the current page in the manner that the operator specifies. The principal path-painting operators are **S** (for *stroking*) and **f** (for *filling*). Variants of these operators combine stroking and filling in a single operation or apply different rules for determining the area to be filled. Table 4.10 lists all the path-painting operators.

Stroking

The **S** operator paints a line along the current path. The stroked line follows each straight or curved segment in the path, centered on the segment with sides parallel to it. Each of the path's subpaths is treated separately.

The results of the **S** operator depend on the current settings of various parameters in the graphics state. See Section 4.3, "Graphics State," for further information on these parameters.

TABLE 4.10 Path-painting operators

OPERANDS	OPERATOR	DESCRIPTION
—	S	Stroke the path.
—	s	Close and stroke the path. This operator has the same effect as the sequence <code>h S</code> .
—	f	Fill the path, using the nonzero winding number rule to determine the region to fill (see “Nonzero Winding Number Rule” on page 169).
—	F	Equivalent to f ; included only for compatibility. Although applications that read PDF files must be able to accept this operator, those that generate PDF files should use f instead.
—	f*	Fill the path, using the even-odd rule to determine the region to fill (see “Even-Odd Rule” on page 170).
—	B	Fill and then stroke the path, using the nonzero winding number rule to determine the region to fill. This produces the same result as constructing two identical path objects, painting the first with f and the second with S . Note, however, that the filling and stroking portions of the operation consult different values of several graphics state parameters, such as the current color. See also “Special Path-Painting Considerations” on page 462.
—	B*	Fill and then stroke the path, using the even-odd rule to determine the region to fill. This operator produces the same result as B , except that the path is filled as if with f* instead of f . See also “Special Path-Painting Considerations” on page 462.
—	b	Close, fill, and then stroke the path, using the nonzero winding number rule to determine the region to fill. This operator has the same effect as the sequence <code>h B</code> . See also “Special Path-Painting Considerations” on page 462.
—	b*	Close, fill, and then stroke the path, using the even-odd rule to determine the region to fill. This operator has the same effect as the sequence <code>h B*</code> . See also “Special Path-Painting Considerations” on page 462.
—	n	End the path object without filling or stroking it. This operator is a “path-painting no-op,” used primarily for the side effect of changing the current clipping path (see Section 4.4.3, “Clipping Path Operators”).

- The width of the stroked line is determined by the current line width parameter (“Line Width” on page 152).
- The color or pattern of the line is determined by the current color and color space for stroking operations.
- The line can be painted either solid or with a dash pattern, as specified by the current line dash pattern (“Line Dash Pattern” on page 155).
- If a subpath is open, the unconnected ends are treated according to the current line cap style, which may be butt, rounded, or square (“Line Cap Style” on page 153).
- Wherever two consecutive segments are connected, the joint between them is treated according to the current line join style, which may be mitered, rounded, or beveled (“Line Join Style” on page 153). Mitered joins are also subject to the current miter limit (“Miter Limit” on page 153).

***Note:** Points at which unconnected segments happen to meet or intersect receive no special treatment. In particular, “closing” a subpath with an explicit **I** operator rather than with **h** may result in a messy corner, because line caps will be applied instead of a line join.*

- The stroke adjustment parameter (*PDF 1.2*) specifies that coordinates and line widths be adjusted automatically to produce strokes of uniform thickness despite rasterization effects (Section 6.5.4, “Automatic Stroke Adjustment”).

If a subpath is degenerate (consists of a single-point closed path or of two or more points at the same coordinates), the **S** operator paints it only if round line caps have been specified, producing a filled circle centered at the single point. If butt or projecting square line caps have been specified, **S** produces no output, because the orientation of the caps would be indeterminate. (Note that this rule applies only to zero-length subpaths of the path being stroked, and not to zero-length dashes in a dash pattern. In the latter case, the line caps are always painted, since their orientation is determined by the direction of the underlying path.) A single-point open subpath (specified by a trailing **m** operator) produces no output.

Filling

The **f** operator uses the current nonstroking color to paint the entire region enclosed by the current path. If the path consists of several disconnected subpaths, **f**

paints the insides of all subpaths, considered together. Any subpaths that are open are implicitly closed before being filled.

If a subpath is degenerate (consists of a single-point closed path or of two or more points at the same coordinates), **f** paints the single device pixel lying under that point; the result is device-dependent and not generally useful. A single-point open subpath (specified by a trailing **m** operator) produces no output.

For a simple path, it is intuitively clear what region lies inside. However, for a more complex path—for example, a path that intersects itself or has one subpath that encloses another—the interpretation of “inside” is not always obvious. The path machinery uses one of two rules for determining which points lie inside a path: the *nonzero winding number rule* and the *even-odd rule*, both discussed in detail below.

The nonzero winding number rule is more versatile than the even-odd rule and is the standard rule the **f** operator uses. Similarly, the **W** operator uses this rule to determine the inside of the current clipping path. The even-odd rule is occasionally useful for special effects or for compatibility with other graphics systems; the **f*** and **W*** operators invoke this rule.

Nonzero Winding Number Rule

The *nonzero winding number rule* determines whether a given point is inside a path by conceptually drawing a ray from that point to infinity in any direction and then examining the places where a segment of the path crosses the ray. Starting with a count of 0, the rule adds 1 each time a path segment crosses the ray from left to right and subtracts 1 each time a segment crosses from right to left. After counting all the crossings, if the result is 0 then the point is outside the path; otherwise it is inside.

Note: *The method just described does not specify what to do if a path segment coincides with or is tangent to the chosen ray. Since the direction of the ray is arbitrary, the rule simply chooses a ray that does not encounter such problem intersections.*

For simple convex paths, the nonzero winding number rule defines the inside and outside as one would intuitively expect. The more interesting cases are those involving complex or self-intersecting paths like the ones shown in Figure 4.10. For a path consisting of a five-pointed star, drawn with five connected straight

line segments intersecting each other, the rule considers the inside to be the entire area enclosed by the star, including the pentagon in the center. For a path composed of two concentric circles, the areas enclosed by both circles are considered to be inside, *provided that both are drawn in the same direction*. If the circles are drawn in opposite directions, only the “doughnut” shape between them is inside, according to the rule; the “doughnut hole” is outside.

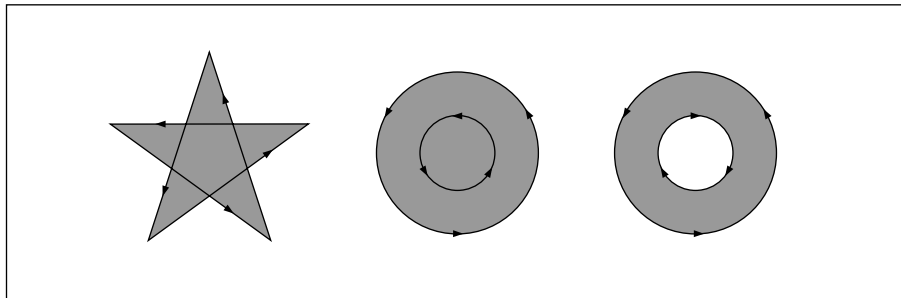
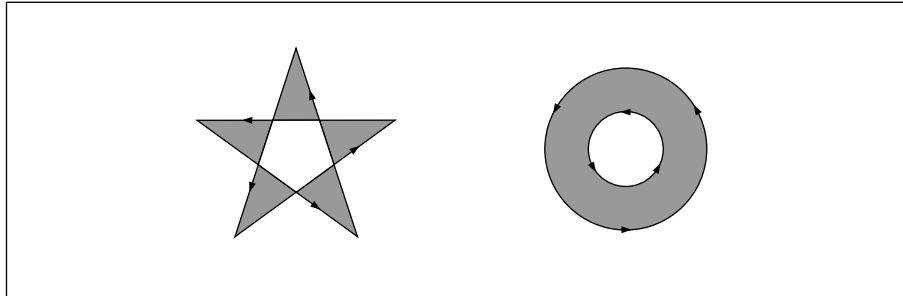


FIGURE 4.10 Nonzero winding number rule

Even-Odd Rule

An alternative to the nonzero winding number rule is the *even-odd rule*. This rule determines the “insideness” of a point by drawing a ray from that point in any direction and simply counting the number of path segments that cross the ray, regardless of direction. If this number is odd, the point is inside; if even, the point is outside. This yields the same results as the nonzero winding number rule for paths with simple shapes, but produces different results for more complex shapes.

Figure 4.11 shows the effects of applying the even-odd rule to complex paths. For the five-pointed star, the rule considers the triangular points to be inside the path, but not the pentagon in the center. For the two concentric circles, only the “doughnut” shape between the two circles is considered inside, regardless of the directions in which the circles are drawn.

**FIGURE 4.11** *Even-odd rule*

4.4.3 Clipping Path Operators

The graphics state contains a *current clipping path* that limits the regions of the page affected by painting operators. The closed subpaths of this path define the area that can be painted. Marks falling inside this area will be applied to the page; those falling outside it will not. (Precisely what is considered to be “inside” a path is discussed under “Filling,” above.)

Note: *In the context of the transparent imaging model (PDF 1.4), the current clipping path constrains an object’s shape (see Section 7.1, “Overview of Transparency”). The effective shape is the intersection of the object’s intrinsic shape with the clipping path; the source shape value is 0.0 outside this intersection. Similarly, the shape of a transparency group (defined as the union of the shapes of its constituent objects) is influenced both by the clipping path in effect when each of the objects is painted and by the one in effect at the time the group’s results are painted onto its backdrop.*

The initial clipping path includes the entire page. A clipping path operator (**W** or **W***, shown in Table 4.11) may appear after the last path construction operator and before the path-painting operator that terminates a path object. Although the clipping path operator appears before the painting operator, it does not alter the clipping path at the point where it appears. Rather, it modifies the effect of the succeeding painting operator. *After* the path has been painted, the clipping path in the graphics state is set to the intersection of the current clipping path and the newly constructed path.

TABLE 4.11 Clipping path operators

OPERANDS	OPERATOR	DESCRIPTION
—	W	Modify the current clipping path by intersecting it with the current path, using the nonzero winding number rule to determine which regions lie inside the clipping path.
—	W*	Modify the current clipping path by intersecting it with the current path, using the even-odd rule to determine which regions lie inside the clipping path.

Note: In addition to path objects, text objects can also be used for clipping; see Section 5.2.5, “Text Rendering Mode.”

The **n** operator (see Table 4.10 on page 167) is a “no-op” path-painting operator; it causes no marks to be placed on the page, but can be used with a clipping path operator to establish a new clipping path. That is, after a path has been constructed, the sequence **W n** will intersect that path with the current clipping path to establish a new clipping path.

There is no way to enlarge the current clipping path or to set a new clipping path without reference to the current one. However, since the clipping path is part of the graphics state, its effect can be localized to specific graphics objects by enclosing the modification of the clipping path and the painting of those objects between a pair of **q** and **Q** operators (see Section 4.3.1, “Graphics State Stack”). Execution of the **Q** operator causes the clipping path to revert to the value that was saved by the **q** operator, before the clipping path was modified.

4.5 Color Spaces

PDF includes powerful facilities for specifying the colors of graphics objects to be painted on the current page. The color facilities are divided into two parts:

- *Color specification.* A PDF file can specify abstract colors in a device-independent way. Colors can be described in any of a variety of color systems, or *color spaces*. Some color spaces are related to device color representation (grayscale, *RGB*, *CMYK*), others to human visual perception (CIE-based). Certain special features are also modeled as color spaces: patterns, color mapping, separations, and high-fidelity and multitone color.