11.3 Using graphics primitives and directives

Concepts

Graphics structure: elements, primitives, directives, options
Graphics ADT and function Show
Point primitive and point size directives
Optical effects vs. relative sizing
Color and GrayLevel directives
Text primitive
Line primitive; thickness and dashing directives
Circle and Disk primitives
Aspect ratio
Polygon and Rectangle primitives
Approximating curves
Exporting graphics to other software
Exercises

Sections 11.3–11.6 digress from the presentation of the twoDG package in order to introduce general-purpose Mathematica graphics features on which it’s based. These also underlie the built-in functions used in earlier chapters to construct several kinds of graphs. Besides laying foundation for twoDG, these sections cover techniques for altering and enhancing those graphs. Discussion of the twoDG package is resumed in section 11.7.

Graphics structure

When you build a complex figure, you’re usually conscious of its hierarchical organization into parts, often called graphics elements. These have geometric form and nongeometric properties. For example, consider a grid consisting of squares, each of which, in turn, is composed of edges. One square is black; another has three black edges and one red. Geometric aspects of a graphics element include size, shape, orientation, and location. Nongeometric properties include color and thickness. How could you not regard thickness as geometric? Euclid didn’t: The Elements’ second sentence reads, “A line is length
without breadth.”¹ In graphics you often use thickness only to emphasize a particular line segment. Instead of using color, you simply make that edge thicker.

Some nongeometric aspects of a figure depend on its context in a document or presentation. For example, the size and shape of a frame may be influenced by the geometric content of a figure, but an editor might require a frame style consistent with the type of document. Some graphics systems don’t even consider a frame as part of a figure, but leave it to be provided by publishing or presentation software. However, *Mathematica* is designed for publishing and presentation as well as computation, so its graphics provisions include frame specification.

Some nongeometric features have both content-related and contextual aspects. For example, you might want to place a label $A$ just inside one of the vertices of a square. Its location refers to figure content, but its font probably depends on context—it may have to be consistent with that used in some descriptive text.

Thus you can classify the essential aspects of a graphics element as in the diagram at right. The classification isn’t perfect. The distinctions are sometimes technically fuzzy—whether that emphasized line segment is really red, for example, depends on hardware. And they’re always related to the psychology of the viewer and of the software designer.²

The simplest geometric forms that a graphics system can draw are called its *primitives*. The software doesn’t require you to break those into components. *Mathematica* uses *graphics directives* to assign noncontextual nongeometric properties to elements. Contextual properties are established by *graphics options*. Specifying directives and options for all possible nongeometric properties would be very tedious, so *Mathematica* provides default values for them. A complete figure, ready to draw, is represented by an object of an abstract data type (ADT) called *Graphics*. It has the form `Graphics[λ, ω]`, where $λ$ — the *display list* — is a graphics primitive or nested list of primitives, and $ω$ — the *option list* — is a graphics option or list of options. You may omit

---

¹ Euclid 1933, p. 1.
² To the author, for example, red lines among black often seem *de-emphasized*. 
\[ \omega \] if you accept Mathematica’s defaults on all options. This section will emphasize the display list; options are considered in detail in section 11.6.

For example, figure 1 consists of a point \( P \) on the left and points \( Q \) over \( R \) on the right. It’s represented by a nested display list of the form

\[ \lambda = \{ d_1, P, \mu \} \quad \mu = \{ Q, d_2, R \} \]

where \( P, Q, R \) are point primitives and \( d_1, d_2 \) are point size directives that make \( R \) twice as big as \( P \) and \( Q \). (Other representations are just as appropriate. See exercise 2.)

Figure 1 confirms the rule that directives apply to all succeeding entries of the list in which they occur, and (recursively) to sublist entries. When those provisions collide, the first takes precedence: \( d_1 \) applies to \( P \) and \( Q \) but not \( R \).

Mathematica’s graphics primitives and directives are objects of ADTs with suggestive names. References to primitives, directives, and Graphics objects look like function calls, but are not. They cause no action directly.

You draw a figure by invoking function Show. For example, figure 1 was created by first selecting appropriate point primitives and point size directives, building the nested display list \( \lambda \) described earlier, then the object \( \text{figure1} = \text{Graphics}[\lambda] \), and finally executing \( \text{Show}[\text{figure1}] \). All this book’s figures are rendered by Show when the target display device is a CRT. Function Export, described at the end of this section, rendered them when the target was a file for import into a word processor.

---

3 This is a slight simplification. See footnote 5.
Point primitive and point size directives

Mathematica’s simplest graphics primitive and graphics directive types, Point and AbsolutePointSize, were used to represent figure 1. Point objects were considered in detail in the previous section. That material served mainly to introduce some aspects of object-oriented programming and the twoDG package. The current section doesn’t use any of those features, beyond this: Point[{x,y}] represents the point with coordinates x, y. Directive AbsolutePointSize[p] causes Show to render points as disks with diameter p printer’s points. Here’s the code that produced figure 1:

```mathematica
P = {0,0.2}; Q = {0.5,0.5}; R = {0.5,0};
μ = {Point[Q], AbsolutePointSize[28], Point[R]};
λ = {AbsolutePointSize[14], Point[P], μ};
figure1 = Graphics[λ];
Show[Graphics[figure1], PlotRange -> {{-0.5,1.5}, {-0.5,1.5}}];
```

Point size directive PointSize[f] is also useful. It causes Show to display points as disks with diameter f times the width of the entire figure. Sometimes you can achieve the best effect with this relative point size. But when you use point size solely for emphasis in a figure, you’re exploiting an optical effect. Optical effects don’t scale according to figure size. If the figure is too small, they disappear; if too large, they become too dominant.

Color directives

Several directives cause Show to render graphics in color:

- CMYKColor[c,m,y,k]
- Hue[h,s,b]
- GrayLevel[g]
- RGBColor[r,g,b]

Each corresponds to a common system for specifying colors or grays. These systems, and how to find parameters appropriate for a desired effect, are discussed in section 11.5.

---

4 The printer’s point, a unit of length standard in typesetting, is approximately $\frac{1}{72}$ inch.
5 The `PlotRange` option makes Show display the entire figure. Without it, *Mathematica* would determine the figure boundaries from the point coordinates, and portions of points would consequently fall outside. Graphics options are discussed in section 11.6.
Text primitive

Sometimes you need to include text in a figure, to label individual elements or the whole figure. Graphics primitives of type Text provide this capability. Simple forms are easy to construct. For example, the primitive Text[“ABC”,{1,2}] causes Show to display text ABC in the default color centered at point P<1,2>. Additional forms of Text objects enable you to change the position of the text relative to P or rotate it about P by a multiple of 90°. Graphics options and the text processing function StyleForm let you choose the font and text size. The default is Mathematica’s standard output font. Options are discussed briefly in section 11.6. For more detail, consult Mathematica’s help browser.

Overlaying text on graphics is a typesetting task. Although Mathematica claims to be a complete mathematical typesetting system suitable for publishing books, the author uses those capabilities only for creating live presentations that must be done entirely by Mathematica to achieve the desired impact. For this book, Mathematica produced the figures and exported them as vector graphics files. A word processor imported them, and placed them in the document, resized if necessary, behind the body text. It then placed the desired text in the appropriate font atop the figure at the correct spot in the usual way. The author, perhaps old-fashioned, finds this overlay technique more flexible, easier to control, and more reliable than the Mathematica features.

Line primitive; thickness and dashing directives

Graphics primitives of type Line are used for drawing line segments or polygons. In particular, Line[{{x1,y1},{x2,y2}}] causes Show to draw the segment between points with coordinates {x1,y1} and {x2,y2}. With a longer list L of coordinate pairs, Line[L] causes Show to draw the polygon with the corresponding vertices.

Graphics directive AbsoluteThickness[f] causes Show to display line segments with thickness p printer’s points. For example, to draw figure 2 insert

---

6 The author used the WordPerfect word processor and encapsulated Postscript graphics files.
7 The terminology is confusing. Another primitive, Polygon, for closed filled polygons, is described later in this section.
8 Mathematica regards lists L = {{x,y}} and L = {} as corresponding to invisible polygons, hence in those cases Show[Graphics[L]] produces no error, and no figure.
AppendTo[\[Lambda],
{AbsoluteThickness[5],
Line[{P,Q,R}]}
];

after the line defining \[Lambda] in the code presented earlier for figure 1. This appends the AbsoluteThickness directive and Line primitive to the display list \[Lambda] before it’s made into a Graphics object. To specify thickness relative to the width of the figure, use the analogous Thickness directive.

Sometimes you want to distinguish various lines in a figure by making them dotted or dashed, with different patterns. This is accomplished by the directive AbsoluteDashing. You specify the length of its dots and dashes and the spaces between them in printer’s points. For example, the following code produced the dashed line in figure 3.

\[P = \{0,0\}; \quad Q = \{1,0\};\]
\[\lambda = \{\text{AbsoluteDashing}[\{4,8,16,8\}],\]
\[\text{AbsoluteThickness}[8], \text{Line}[\{P,Q\}] \};\]
\[\text{Show}[\text{Graphics}[\lambda]];\]

The AbsoluteDashing directive’s list of lengths corresponds to one cycle of dashes as indicated; the cycle is repeated until the segment is covered. Caution! Something’s wrong. Mathematica seems to be using different units for the lengths of dashes and spaces between. Moreover, experimentation shows that the effect of the dashing directive is somehow dependent on the thickness directive. This seems to be a bug in the author’s Mathematica installation—either the program or its documentation.

The Dashing directive works like AbsoluteDashing, except that dash and space lengths are specified relative to the width of the entire figure.

To specify solid lines after including some dashed lines in a graphics element, use directive Dashing[{}]. For example, see the code presented later for figure 4.
Circle and Disk primitives

Circle and Disk primitives correspond to arcs of ellipses and sectors of filled ellipses. You specify the center coordinates \( x_0, y_0 \), horizontal and vertical semiaxes \( a, b \), and initial and final parameters \( \theta_1, \theta_2 \). The primitives are

\[
\text{Circle}[\{x_0, y_0\}, \{a, b\}, \{\theta_1, \theta_2\}] \quad \text{Disk}[\{x_0, y_0\}, \{a, b\}, \{\theta_1, \theta_2\}]
\]

Parameters \( \theta_1, \theta_2 \) are the initial and final values for \( \theta \) in the curve’s parametric representation

\[
\begin{align*}
x &= x_0 + a \cos \theta \\
y &= y_0 + b \sin \theta
\end{align*}
\]

For a circular arc or sector you may specify a single radius \( r \) instead of the list of semiaxes. For the entire circle or ellipse you may omit the initial and final parameters and associated punctuation. Thickness and dashing directives apply to Circle objects. Use the color directives discussed in the next section to tailor the appearance of the interior of Disk objects.

As an example, consider figure 4, which was produced by the following code:

\[
Z = \{0,0\}; \\
r = 1; \\
\text{semiaces} = \{0.5, r\}; \\
\theta = 0.5; \\
C1 = \text{Circle}[Z, r, \{-\theta, \theta\}]; \\
C2 = \text{Circle}[Z, r, \{\theta, 2\pi - \theta\}]; \\
d = \text{Disk}[Z, \text{semiaces}, \{\theta, 2\pi - \theta\}]; \\
\lambda = \{\text{AbsoluteThickness}[1], \\
\text{AbsoluteDashing}[\{2, 4\}], C1, \\
\text{Dashing}[\{\}], C2, d\}; \\
\text{Show}[\text{Graphics}[\lambda], \\
\text{AspectRatio}\to \text{Automatic}]
\]

The last line is a graphics option that you’ve seen before in chapter 2. The appearance of a graph of parametric equations depends on the aspect ratio: the ratio of the actual lengths of units on the horizontal and vertical axes. The option causes Show to make the ratio 1. Without it, the outer curve would appear elliptical, too. Graphics options are discussed in more detail in section 11.6.
**Polygon and Rectangle primitives**

The **Polygon** primitive type is misnamed, because it only represents closed filled polygonal regions. It works just like **Line** except that it triggers construction of a closed polygon by joining the last point in its vertex list to the first, and with the resulting region filled. The following code draws figure 5, the region defined by points \( P, Q, R \) in figure 1.

\[
\begin{align*}
P &= \{0,0.2\}; \\
Q &= \{0.5,0.5\}; \\
R &= \{0.5,0\}; \\
\text{Show}[\text{Graphics}[\text{Polygon}[\{P,Q,R\}]]; \\
\end{align*}
\]

If a closed polygon \( \mathcal{P} \) is very complicated, it may be difficult to predict which region is filled. You may use this rule:

1. Imagine a circle \( \mathcal{C} \) that surrounds the entire polygon \( \mathcal{P} \).
2. To determine whether **Mathematica** will shade a point \( P \) in the filling process,
   (a) imagine a line segment \( \mathcal{S} \) from \( P \) to some point \( X \) exterior to \( \mathcal{C} \) that doesn’t pass through any vertex of \( \mathcal{P} \), and
   (b) count the number \( n \) of intersections of \( \mathcal{S} \) with \( \mathcal{P} \). Then
   (c) \( P \) is shaded just in case \( n \) is odd.

A deep theorem of topology says that you can always carry out this process and the result is independent of the choices of \( \mathcal{C} \) and \( \mathcal{S} \).\footnote{I need to find a reference for this.}

As an example, experiment with this algorithm on various points in figure 6, which was produced by the following code:

\[
\begin{align*}
\kappa &= \{\{0,0\},\{5,0\},\{5,4\},\{1,4\},\{1,2\},\{3,2\},\{3,3\},\{2,3\},
\{2,1\},\{4,1\},\{4,5\},\{0,5\}\}; \\
\lambda &= \{\text{GrayLevel}[0.8], \text{Polygon}[\kappa]\}; \\
\mu &= \text{Append}[\kappa,\{0,0\}]; \\
\nu &= \{\text{AbsoluteThickness}[1], \text{Line}[\mu]\}; \\
\text{Show}[\text{Graphics}[\{\lambda,\nu\}], \text{AspectRatio}\rightarrow\text{Automatic}]
\end{align*}
\]
The origin is in the lower left corner. The polygon corresponding to \( \kappa \) proceeds rightward from there. Display list \( \lambda \) includes color directive GrayLevel, discussed in the next section, in order to fill with some color besides black, which would obliterate the polygon itself. Color directives are discussed in the section 11.5. Finally, notice that the initial vertex, the origin, was appended to \( \kappa \) so that the Line object would represent the closed polygon.

![Figure 11.3.6 Complicated region](image)

Perhaps the most common filled polygons in graphics are rectangles with horizontal and vertical edges. With the Polygon primitive you must specify three vertices. With the Rectangle primitive you need only specify the lower-left and upper-right vertices. You can find further information via the Mathematica help browser.

**Approximating curves**

The only curves and curved regions that Mathematica will draw directly are those produced by the Line, Circle, Polygon, and Disk. Any other curve \( \mathcal{C} \) must be approximated, usually by a polygon \( \mathcal{P} \) whose vertices are points on \( \mathcal{C} \) so close to each other that you can’t distinguish it from \( \mathcal{C} \) visually. This normally requires \( \mathcal{P} \) to have so many vertices that you must use the Table function to prepare the vertex list for the corresponding Line or Polygon primitive.

For example, consider the swimming pool in figure 7, whose edge \( \mathcal{C} \) has polar equation \( r = 2 + 3 \cos^2(\theta + 1) \). To draw it, construct a vertex list \( \lambda \) of points \( <x, y> = <r \cos \theta, r \sin \theta> \) on \( \mathcal{C} \) corresponding to 200 evenly spaced values \( \theta \) such that \( 0 \leq \theta < 2\pi \). Put that in a display list for a closed polygon filled with light gray. Append to \( \lambda \) the initial point, and put the resulting vertex list \( \mu \) in a display list with a Line primitive and thickness directive. Here’s the code:
Additional primitives

Mathematica provides additional graphics primitives for bitmaps, PostScript code, and sound effects. They fall outside the scope of this book.

Graphics options, Show and Export

Discussion of the code for figures 1, 4, and 6 mentioned without fanfare the use of graphics options as arguments to the Show function. Before rendering a Graphics object \( \lambda \), Show copies it and updates the options list with the new specifications. After rendering \( \lambda \) with the new options, Show returns the updated copy as function value. This feature is handy for trying options one at a time until a figure is right. The last returned value corresponds to the most recent version of the figure.

Once you’ve successfully rendered a figure with Show, you can pass this final version of the corresponding Graphics object to function Export for conversion and output in one of several common graphics file formats. As mentioned earlier, the author used that technique to produce the figures for this book. The files were imported into a powerful word processor and placed in the body text so that required figure labels could be overlaid easily and precisely. Here’s a typical command sequence, actually used for figure 9:

```
Show[Graphics[\(\lambda\), AspectRatio -> Automatic];
Export["d:\M&M\Chapter11\11_03\11_03_09.eps",%]
```
This created an \textit{encapsulated PostScript} \texttt{*.eps} file in the appropriate folder on the author’s development computer. In the filename, backslash symbols `\' must be doubled because a single backslash character has a special function in \textit{Mathematica} keyboard input. Unfortunately, \texttt{Export} returns the file name, not a \texttt{Graphics} object, so you can’t use it to keep track of amendments.

\textit{Mathematica}’s help browser lists the file types that \texttt{Export} can output, divided into two classes: bitmap and vector graphics. Bitmap file types are inappropriate. They produce figures of the original size only, which is hardly ever what you’ll eventually want. Use a vector graphics file type that your document preparation or presentation software can import, because you can easily change the figure size.

\textit{Caution!} Except for incorporating the most recent options passed on by \texttt{Show}, \texttt{Export} operates entirely independently. The output file may differ in some details from what \texttt{Show} rendered. Inspect it carefully! The author used \texttt{*.eps} files for most figures in this book, but they failed occasionally. \textit{Windows Metafile} \texttt{*.wmf} files were used in most of those cases.

\textbf{Exercises}

1. Draw a figure consisting of a square with horizontal base, with an interior point joined to the vertices. Make the top triangle equilateral.

2. \textit{Part 1.} Draw figure 1 with a different display list structure. \textit{Part 2.} What would the ends of the line segments in figure 2, and the angle vertex, look like, if they weren’t hidden by the big points?

3. Draw a \textit{quincunx}: a square $S$ and five circles with equal radii. Four of the circles should each be tangent to two adjacent edges of $S$. The remaining one should be concentric with $S$ and tangent to the others.

4. Draw the church window in figure 8. The lower left and right points are the centers of the outer arcs.
5. Draw and fill the lune in figure 9, and the dashed lines.

![Church window](image1)

![Lune](image2)

Figure 8 Church window

Figure 9 Lune

6. Draw the old Army Air Force insignia as follows. First, draw and fill the unit circle with black. Next, locate five equally spaced points on the unit circle, with one at \( Y < 0, 1 > \); these will be the vertices. Draw and fill a pentagram with white; it's the polygon that connects every second vertex, starting and ending with \( Y \). Explain what happens in the central pentagon \( \mathcal{P} \). Finally, draw and fill the circumcircle \( \mathcal{C} \) of \( \mathcal{P} \) with medium gray. (You’ll need to calculate the radius of \( \mathcal{C} \).)

7. Draw and fill flower-like polygons built from the polar graphs of equations of the form \( r = a + b \sin n \theta \) for reasonably small integer values \( a, b, n \). Use the technique of figure 7. Investigate the effects of varying these integers.

8. Make a figure analogous to figure 6, but with a polygon having nine vertices instead of twelve. The polygon should have “alternate” filled and nonfilled regions, and it should be based on a triangular pattern, rather than rectangular.