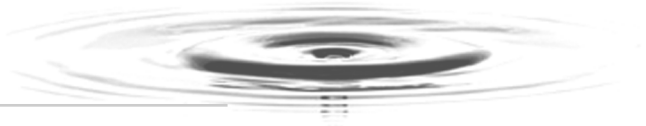

Interactive Graphics—On the Server

This chapter is dedicated to creating graphics interactively—on the server. Here, we're going to see how to use the GD graphics module with PHP to draw graphics on the server at run time and send them back to the browser. You can draw lines, rectangles, ellipses, and text, modify existing images, and more, sending the results back in a variety of formats: JPEG, GIF, PNG, and so forth. Think of what you can do with this technology—you can draw real-time stock reports, add time and date to images, create weather maps, draw graphs or hotel occupancy charts, and any number of other things.

To make this happen, you have to enable support for the GD module in PHP. To enable that support, configure PHP using `--with-gd[=`*`DIR`*`]`, where *DIR* is the GD base install directory. To use the bundled version of the GD library (recommended), use the configure option: `--with-gd`. That's all it takes. In Windows, you can find the needed DLL, php_gd2.dll, in the PHP ext directory. Copy it to the main PHP directory, uncomment the php_gd2.dll line in php.ini, and restart PHP:

```
;extension=php_fdf.dll
extension=php_gd2.dll
;extension=php_gettext.dll
        .
        .
        .
```

Using GD, we'll be able to use functions such as `imagecreate` to create images and `imagerectangle` to draw in them, like this:

```
$image_height = 100;
$image_width = 300;
$image = imagecreate($image_width, $image_height);
$rect_color = imagecolorallocate($image, 128, 128, 128);
imagerectangle($image, 0, 0, 50, 50, $rect_color);
```

OK, we're set; let's start creating graphics! For reference, you can find the GD manual at http://us4.php.net/manual/en/ref.image.php.

Getting Started

To get started, we're going to create a simple image—just an image showing a background color—and then send it as a JPEG image to the browser. To start, you create an image object with `imagecreate`, which you use like this:

```
imagecreate(int x_size, int y_size)
```

The *x_size* and *y_size* values are in pixels. Here's how we create our first image:

```
$image_height = 100;
$image_width = 300;

$image = imagecreate($image_width, $image_height);
         .
         .
         .
```

To set colors for the image, you use `imagecolorallocate`, passing it the image you're working with, as well as the red, green, and blue components as values from 0 to 255:

```
imagecolorallocate(resource image, int red, int green, int blue)
```

The first time you call `imagecolorallocate`, this function sets the background color. Subsequent calls set various drawing colors, as we'll see. Here's how we set the background color to light gray (red = 200, green = 200, blue = 200):

```
$image = imagecreate($image_width, $image_height);

$back color = imagecolorallocate($image, 200, 200, 200);
         .
         .
         .
```

To send a JPEG image back to the browser, you have to tell the browser that you're doing so with the `header` function to set the image's type, and then you send the image with the `imagejpeg` function like this (do this before any other output is sent to the browser):

```
$image_height = 100;
$image_width = 300;

$image = imagecreate($image_width, $image_height);

$back_color = imagecolorallocate($image, 200, 200, 200);

header('Content-Type: image/jpeg');
imagejpeg($image);
         .
         .
         .
```

Here are some of the image-creating functions for various image formats:

- `imagegif`. Output a GIF image to browser or file
- `imagejpeg`. Output a JPEG image to browser or file
- `imagewbmp`. Output a WBMP image to browser or file
- `imagepng`. Output a PNG image to browser or file

After sending the image, you can destroy the image object with the `imagedestroy` function; all this is shown in phpbox.php, Example 1.

EXAMPLE 1 Displaying an image, phpbox.php

```
<?php
    $image_height = 100;
    $image_width = 300;

    $image = imagecreate($image_width, $image_height);

    $back_color = imagecolorallocate($image, 200, 200, 200);

    header('Content-Type: image/jpeg');
    imagejpeg($image);

    imagedestroy($image);
?>
```

The results appear in Figure 1, where you can see our image, which is simply all background color. Cool, a JPEG image created on the server!

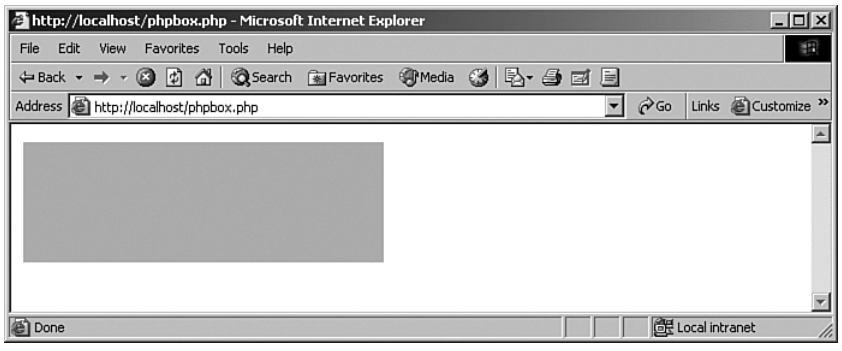

FIGURE 1 Displaying an image.

Embedding an Image in an HTML Page

Because phpbox.php creates a JPEG, you can assign it to the SRC attribute in an <IMG> element in an HTML document, as you see in phpbox.html, Example 2.

EXAMPLE 2 Displaying an image in an HTML page, phpbox.html

```
<HTML>
    <HEAD>
        <TITLE>
            Embedding interactive images in HTML pages
        </TITLE>
    </HEAD>

    <BODY>
        <CENTER>
            <H1>
                Embedding interactive images in HTML pages
            </H1>
            Here's a blank image created on the server:
            <BR>
            <IMG SRC="phpbox.php">
        </CENTER>
    </BODY>
</HTML>
```

This HTML page appears in Figure 2—complete with our image, as created on the server. Very cool.

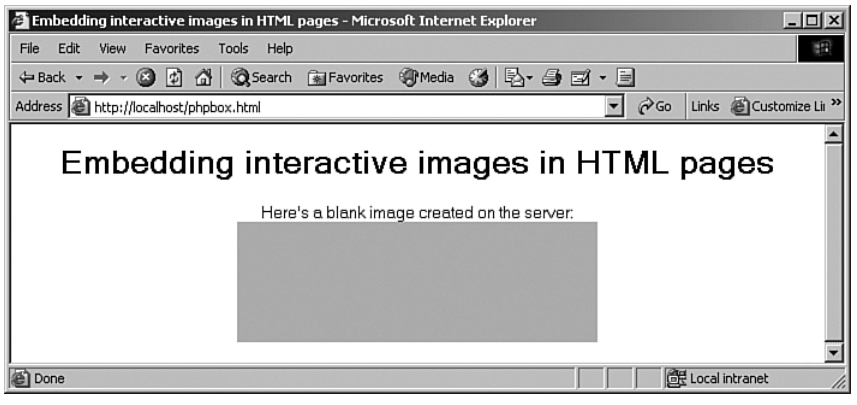

FIGURE 2 Displaying an interactive image in an HTML page.

Drawing a Line

So far, we've just drawn a blank image using light gray as a background color. How about drawing a visible figure? We'll start by drawing lines using the `imageline` function, which works like this:

```
imageline(resource image, int x1, int y1, int x2, int y2, int color)
```

This function draws a line from (*x1*, *y1*) to (*x2*, *y2*) in image *image* of color *color*. Note that the top left of the image is (0, 0), and all measurements are in pixels.

Here's an example where we'll draw some lines in a JPEG image using black as a drawing color:

```
$image_height = 100;
$image_width = 300;

$image = imagecreate($image_width, $image_height);

$back_color = imagecolorallocate($image, 200, 200, 200);

$draw_color = imagecolorallocate($image, 0, 0, 0);

imageline($image, 20, 20, 80, 80, $draw_color);

imageline($image, 20, 90, 200, 10, $draw_color);

imageline($image, 120, 20, 160, 80, $draw_color);
```

After drawing these lines, you can send the new image back to the browser with `imagejpeg`, as you see in Example 3, phpline.php.

EXAMPLE 3 Drawing lines, phpline.php

```
<?php

    $image_height = 100;
    $image_width = 300;

    $image = imagecreate($image_width, $image_height);

    $back_color = imagecolorallocate($image, 200, 200, 200);

    $draw_color = imagecolorallocate($image, 0, 0, 0);

    imageline($image, 20, 20, 80, 80, $draw_color);

    imageline($image, 20, 90, 200, 10, $draw_color);
```

continues

EXAMPLE 3 continued

```
    imageline($image, 120, 20, 160, 80, $draw_color);

    header('Content-Type: image/jpeg');

    imagejpeg($image);

    imagedestroy($image);
?>
```

Here's an HTML document, phpline.html, which will display our JPEG:

```
<HTML>
    <HEAD>
        <TITLE>
            Drawing lines on the server
        </TITLE>
    </HEAD>

    <BODY>
        <CENTER>
            <H1>
                Drawing lines on the server
            </H1>
            Here are some lines in an image created on the server:
            <BR>
            <IMG SRC="phpline.php">
        </CENTER>
    </BODY>
</HTML>
```

The new image appears in Figure 3. Not bad.

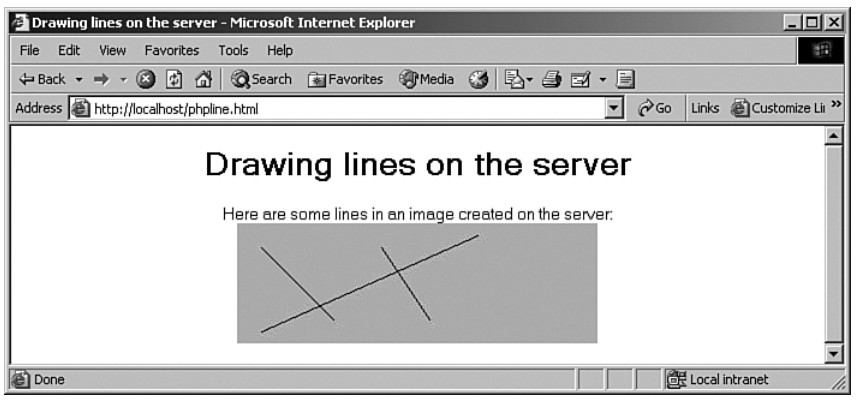

FIGURE 3 Displaying lines in an HTML page.

Drawing Thick Lines

Want to control the thickness of the lines you draw? Use the `imagesetthickness` function:

```
imagesetthickness(resource image, int thickness)
```

When you call this function and pass an image to it, it sets the drawing width for that image, in pixels. For example, here's how you might add a call to that function in phpthickline.php, where we want to draw lines that are four pixels thick:

```
<?php

    $image_height = 100;
    $image_width = 300;

    $image = imagecreate($image_width, $image_height);
    imagesetthickness($image, 4);

    imageline($image, 20, 20, 70, 80, $draw_color);
        .
        .
        .
```

You can see this at work in phpthickline.php and phpthickline.html, Figure 4, where we're drawing the lines from the previous chunk, now four pixels wide.

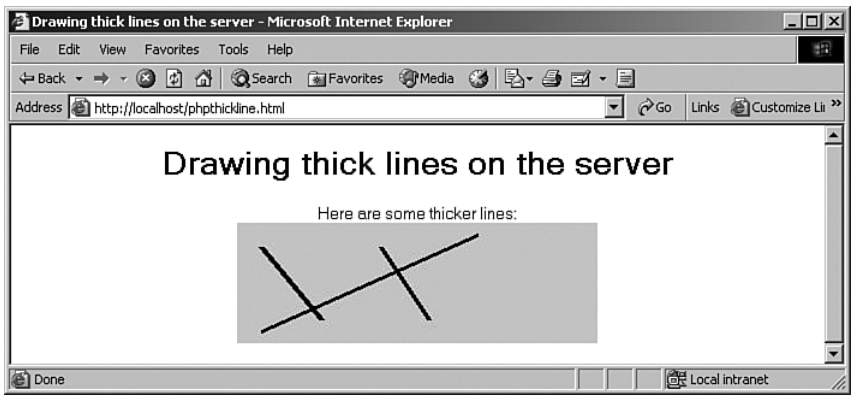

FIGURE 4 Drawing thick lines.

Drawing a Rectangle

You can draw plenty of figures using lines alone, but far more GD functions are available. One of them, `imagerectangle`, draws rectangles:

`imagerectangle(resource` *`image`*`, int` *`x1`*`, int` *`y1`*`, int` *`x2`*`, int` *`y2`*`, int` *`color`*`)`

This function creates a rectangle of color *color* in image *image* starting at upper-left coordinate *x1*, *y1* and ending at bottom-right coordinate *x2*, *y2*.

Here's an example. We'll start by using black as our drawing color:

```
$draw_color = imagecolorallocate($image, 0, 0, 0);
    .
    .
    .
```

Then we can draw a few rectangles using `imagerectangle`:

```
$draw_color = imagecolorallocate($image, 0, 0, 0);

imagerectangle($image, 20, 20, 40, 80, $draw_color);

imagerectangle($image, 60, 20, 140, 80, $draw_color);

imagerectangle($image, 160, 20, 270, 80, $draw_color);

    .
    .
    .
```

All that's left is to send the results to the browser, as in phprectangle.php, Example 4.

EXAMPLE 4 Drawing rectangles, phprectangle.php

```
<?php

    $image_height = 100;

    $image_width = 300;

    $image = imagecreate($image_width, $image_height);

    $back_color = imagecolorallocate($image, 200, 200, 200);

    $draw_color = imagecolorallocate($image, 0, 0, 0);

    imagerectangle($image, 20, 20, 40, 80, $draw_color);

    imagerectangle($image, 60, 20, 140, 80, $draw_color);
```

```
    imagerectangle($image, 160, 20, 270, 80, $draw_color);

    header('Content-Type: image/jpeg');

    imagejpeg($image);

    imagedestroy($image);
?>
```

We'll embed the image created by phprectangle.php in a web page, phprectangle.html:

```
<HTML>
    <HEAD>
        <TITLE>
            Drawing rectangles
        </TITLE>
    </HEAD>

    <BODY>
        <CENTER>
            <H1>
                Drawing rectangles
            </H1>
            <BR>
            <IMG SRC="phprectangle.php">
        </CENTER>
    </BODY>
</HTML>
```

The results appear in Figure 5.

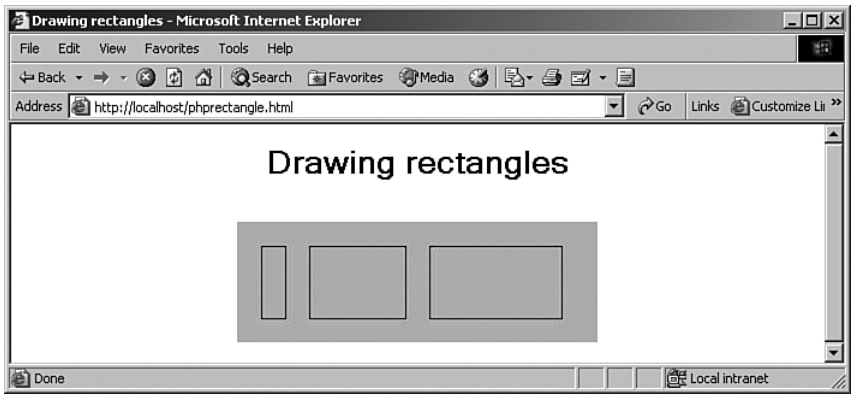

FIGURE 5 Drawing rectangles.

Drawing an Ellipse

The `imageellipse` function, as you can guess, draws ellipses:

```
imageellipse(resource image, int cx, int cy, int w, int h, int color)
```

This function draws an ellipse centered at *cx*, *cy* in the image represented by *image*. The *w* and *h* values specify the ellipse's width and height, respectively. The color of the ellipse is specified by *color*. This function is pretty close to `imagerectangle`, but instead of specifying the upper-left and lower-right corners, you specify the center of the ellipse and its width and height.

Here's an example of putting this drawing function to work. As usual, we'll start by setting the drawing color. Then we'll draw a few ellipses with this function; here's what it looks like:

```
$back_color = imagecolorallocate($image, 200, 200, 200);

$draw_color = imagecolorallocate($image, 0, 0, 0);

imageellipse($image, 100, 50, 150, 50, $draw_color);

imageellipse($image, 150, 40, 150, 50, $draw_color);

imageellipse($image, 200, 30, 150, 50, $draw_color);
```

You can see all the code in phpellipse.php, Example 5.

EXAMPLE 5 Drawing ellipses, phpellipse.php

```
<?php

    $image_height = 100;
    $image_width = 300;

    $image = imagecreate($image_width, $image_height);

    $back_color = imagecolorallocate($image, 200, 200, 200);

    $draw_color = imagecolorallocate($image, 0, 0, 0);

    imageellipse($image, 100, 50, 150, 50, $draw_color);

    imageellipse($image, 150, 40, 150, 50, $draw_color);

    imageellipse($image, 200, 30, 150, 50, $draw_color);

    header('Content-Type: image/jpeg');
```

```
    imagejpeg($image);

    imagedestroy($image);
?>
```

And we'll embed phpellipse.php in phpellipse.html:

```
<HTML>
    <HEAD>
        <TITLE>
            Drawing ellipses
        </TITLE>
    </HEAD>

    <BODY>
        <CENTER>
            <H1>
                Drawing ellipses
            </H1>
            <BR>
            <IMG SRC="phpellipse.php">
        </CENTER>
    </BODY>
</HTML>
```

The new ellipses appear in phpellipse.html, Figure 6. Perfect.

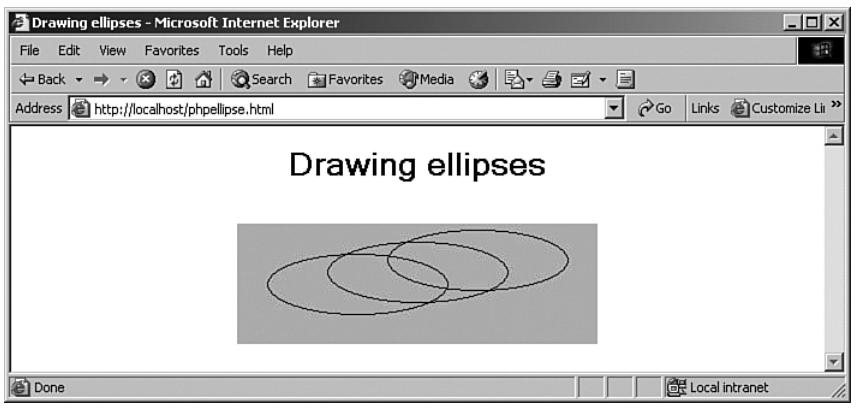

FIGURE 6 Drawing ellipses.

That gets lines, rectangles, and ellipses into our drawing arsenal. A good start.

Drawing a Polygon

If you want to draw your own figures, you can trace them together using multiple lines, but there's an easier way—you can use the `imagepolygon` function to draw a polygon simply by passing it an array of points. Here's how you use this function in general:

```
imagepolygon(resource image, array points, int num_points, int color)
```

This function creates a polygon in *image*. The *points* array is a PHP array containing the polygon's vertices (`points[0] = x0`, `points[1] = y0`, `points[2] = x1`, `points[3] = y1`, and so on). The *num_points* value holds the total number of points in the polygon, and *color* is the drawing color you want to use.

For example, here's an array of points, `$points`, in which every pair of values (`array[0]` and `array[1]`, `array[2]` and `array[3]`, and so on) specifies a point on the screen:

```
$points = array(
    0  => 120, 1  => 60,
    2  => 130, 3  => 60,
    4  => 160, 5  => 80,
    6  => 180, 7  => 20,
    8  => 150, 9  => 40,
    10 => 110, 11 => 10,
    12 => 110, 13 => 80
);
```

Now we can draw the resulting polygon with `imagepolygon`:

```
imagepolygon($image, $points, 7, $draw_color );
```

The whole example appears in phppolygon.php, Example 6.

EXAMPLE 6 Drawing polygons, phppolygon.php

```
<?php
    $points = array(
        0  => 120, 1  => 60,
        2  => 130, 3  => 60,
        4  => 160, 5  => 80,
        6  => 180, 7  => 20,
        8  => 150, 9  => 40,
        10 => 110, 11 => 10,
        12 => 110, 13 => 80
    );

    $image_height = 100;
    $image_width = 300;
```

```
    $image = imagecreate($image_width, $image_height);
    $back_color = imagecolorallocate($image, 200, 200, 200);

    $draw_color = imagecolorallocate($image, 0, 0, 0);

    imagepolygon($image, $points, 7, $draw_color );

    header('Content-type: image/jpeg');
    imagejpeg($image);
    imagedestroy($image);
?>
```

We'll use an HTML document, phppolygon.html, to host the new image:

```
<HTML>
    <HEAD>
        <TITLE>Drawing polygons</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>
                Drawing polygons
            </H1>
            <BR>
            <IMG SRC="phppolygon.php">
        </CENTER>
    </BODY>
</HTML>
```

The new polygon appears in Figure 7. Very nice.

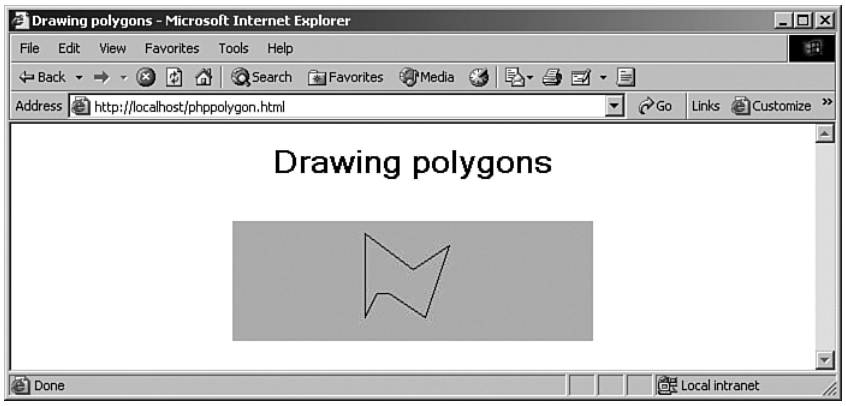

FIGURE 7 Drawing a polygon.

Drawing an Arc

The `imagearc` function lets you draw arcs, which include partial circles and partial ellipses as well as complete circles and ellipses:

```
imagearc(resource image, int cx, int cy, int w, int h, int s, int e, int color)
```

This function is designed to draw arcs centered at *cx*, *cy* in the image represented by *image*. The *w* and *h* values specify the ellipse's width and height, respectively, while the start and end points are specified in degrees indicated by the *s* and *e* arguments (here, 0° is located at the three-o'clock position). The arc itself is drawn clockwise.

Say that you wanted to draw a smiling face using `imagearc`. All you need to do is to create the image, set the drawing color, and use `imagearc`. Here's what it looks like, where we're drawing a complete circle:

```
$image_height = 100;
$image_width = 300;

$image = imagecreate($image_width, $image_height);

$backcolor = imagecolorallocate($image, 200, 200, 200);

$drawing_color = imagecolorallocate($image, 0, 0, 0);

imagearc($image, 150, 50, 50, 50, 10, 170, $drawing_color);
        .
        .
        .
```

Then you can use `imagearc` to draw the rest of the face, as shown in phparc.php, Example 7.

EXAMPLE 7 Drawing arcs, phparc.php

```
<?php

    $image_height = 100;
    $image_width = 300;

    $image = imagecreate($image_width, $image_height);

    $backcolor = imagecolorallocate($image, 200, 200, 200);

    $drawing_color = imagecolorallocate($image, 0, 0, 0);

    imagearc($image, 150, 50, 50, 50, 10, 170, $drawing_color);

    imagearc($image, 150, 50, 70, 70, 0, 360, $drawing_color);
```

```
    imagearc($image, 135, 45, 20, 20, 190, -10, $drawing_color);

    imagearc($image, 165, 45, 20, 20, 190, -10, $drawing_color);

    header("Content-type: image/jpeg");

    imagejpeg($image);

    imagedestroy($image);

?>
```

We'll embed phparc.php in phparc.html like this:

```
<HTML>
    <HEAD>
        <TITLE>Drawing arcs</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>
                Drawing arcs
            </H1>
            <BR>
            <IMG SRC="phparc.php">
        </CENTER>
    </BODY>
</HTML>
```

You can see the resulting smiling face in Figure 8.

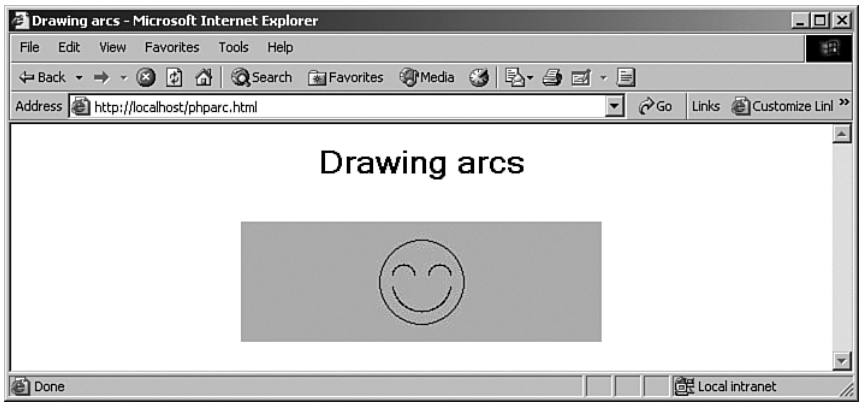

FIGURE 8 Drawing arcs.

Setting Individual Pixels

Want more graphics control? You can set individual pixels using the `imagesetpixel` method:

```
imagesetpixel(resource image, int x, int y, int color)
```

As you'd expect, this function draws a pixel at *x*, *y* in image *image* of color *color*. When you're drawing graphics, working pixel-by-pixel is usually too slow. But you can generally get away with drawing extensive pixel graphics on the server because the extra few seconds usually aren't noticed—the Internet is slow enough anyway.

For example, say you wanted to draw a dotted line, which you can do with `imagesetpixel`. You can use a `for` loop, as here, where we're drawing one pixel and then skipping three pixels before drawing the next pixel:

```
$image_height = 100;
$image_width = 300;

$image = imagecreate($image_width, $image_height);

$back_color = imagecolorallocate($image, 200, 200, 200);
$drawing_color = imagecolorallocate($image, 0, 0, 0);

for($loop_index = 20; $loop_index < 280; $loop_index += 3){
        .
        .
        .
}
```

All that's left is to use `imagesetpixels` to actually draw the individual pixels inside the `for` loop that we've just set up, as you can see in phppixels.php, Example 8.

EXAMPLE 8 Drawing pixels, phppixels.php

```
<?php

    $image_height = 100;
    $image_width = 300;

    $image = imagecreate($image_width, $image_height);

    $back_color = imagecolorallocate($image, 200, 200, 200);
    $drawing_color = imagecolorallocate($image, 0, 0, 0);

    for($loop_index = 20; $loop_index < 280; $loop_index += 3){

        imagesetpixel($image, $loop_index, 50, $drawing_color);
```

```
    }

    header("Content-type: image/jpeg");

    imagejpeg($image);

    imagedestroy($image);

?>
```

We'll embed phppixels.php in a web page, phppixels.html:

```
<HTML>
    <HEAD>
        <TITLE>Drawing individual pixels</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>
                Drawing individual pixels
            </H1>
            <BR>
            <IMG SRC="phppixels.php">
        </CENTER>
    </BODY>
</HTML>
```

You can see the results in Figure 9, where the line we're drawing is indeed dotted, thanks to the `for` loop we've used.

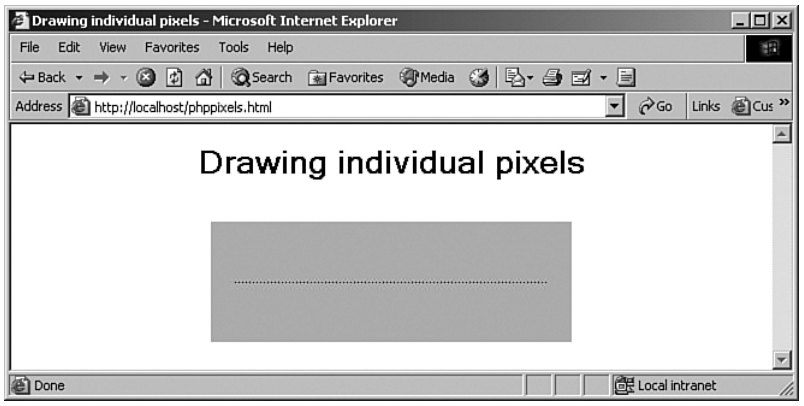

FIGURE 9 Drawing individual pixels.

Filling in Figures

Besides drawing figures in outline, you can also fill in figures with color using various functions such as these:

- `imagefilledarc`. Draw a partial ellipse and fill it
- `imagefilledellipse`. Draw a filled ellipse
- `imagefilledpolygon`. Draw a filled polygon
- `imagefilledrectangle`. Draw a filled rectangle

For example, take a look at `imagefilledrectangle`:

```
imagefilledrectangle(resource image, int x1, int y1, int x2, int y2, int color)
```

This function creates a filled rectangle of color *color* in image *image* starting at upper-left coordinates *x1*, *y1* and ending at bottom-right coordinates *x2*, *y2*. This function is easy enough to use; start out as usual by creating an image and setting a drawing color:

```
$image = imagecreate($image_width, $image_height);
$back_color = imagecolorallocate($image, 200, 200, 200);
$draw_color = imagecolorallocate($image, 0, 0, 0);
        .
        .
        .
```

Now you can draw a number of filled rectangles using `imagefilledrectangle`:

```
$image = imagecreate($image_width, $image_height);
$back_color = imagecolorallocate($image, 200, 200, 200);
$draw_color = imagecolorallocate($image, 0, 0, 0);

imagefilledrectangle($image, 20, 20, 40, 80, $draw_color);
imagefilledrectangle($image, 60, 20, 140, 80, $draw_color);
imagefilledrectangle($image, 160, 20, 270, 80, $draw_color);
```

All this is put to work in phpfilledrectangle.php, Example 9, where we're drawing three filled-in rectangles.

EXAMPLE 9 Filling in rectangles, phpfilledrectangle.php

```
<?php

    $image_height = 100;
    $image_width = 300;

    $image = imagecreate($image_width, $image_height);
    $back_color = imagecolorallocate($image, 200, 200, 200);
```

```
    $draw_color = imagecolorallocate($image, 0, 0, 0);

    imagefilledrectangle($image, 20, 20, 40, 80, $draw_color);
    imagefilledrectangle($image, 60, 20, 140, 80, $draw_color);
    imagefilledrectangle($image, 160, 20, 270, 80, $draw_color);

    header('Content-Type: image/jpeg');
    imagejpeg($image);

    imagedestroy($image);
?>
```

Here's the HTML page for this example, phpfilledrectangle.html:

```
<HTML>
    <HEAD>
        <TITLE>
            Drawing filled rectangles
        </TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>
                Drawing filled rectangles
            </H1>
            <BR>
            <IMG SRC="phpfilledrectangle.php">
        </CENTER>
    </BODY>
</HTML>
```

The results appear in Figure 10.

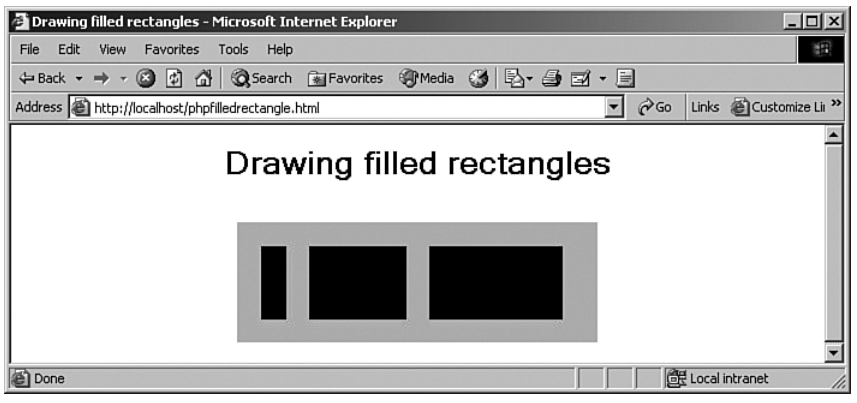

FIGURE 10 Drawing filled rectangles.

Drawing Text

How about drawing text? There are a number of functions that draw text, such as `imagestring`:

```
imagestring(resource image, int font, int x, int y, string s, int color)
```

This function draws the string *s* in the image specified by *image* with the upper-left corner at coordinates *x*, *y* in color *color*. The graphics package comes with built-in fonts—if *font* is 1, 2, 3, 4, or 5, a built-in font is used. Note that because we're working with graphics, this text is drawn as an image, not as the editable text that would appear in a text field.

Say for example that you want to display some text centered in an image. We'll use built-in font number 4 to show how that works. To center the text, you need the text's dimensions, which you can get with the `imagefontwidth` and `imagefontheight` functions. Here's how we determine the *x* and *y* positions at which to draw our text:

```
$font_number = 4;
$text = "No worries.";

$width = 2 * strlen($text) * imagefontwidth($font_number);
$height = 3 * imagefontheight($font_number);

$image = imagecreate($width, $height);

$back_color = imagecolorallocate($image, 200, 200, 200);
$drawing_color = imagecolorallocate($image, 0,   0,   0);
$x_position = ($width - (strlen($text) * imagefontwidth($font_number))) / 2;
$y_position = ($height - imagefontheight($font_number)) / 2;
```

Now that you have the *x* and *y* positions at which to draw the text, you can use `imagestring` to draw that text, as shown in phpdrawtext.php, Example 10.

EXAMPLE 10 Drawing text, phpdrawtext.php

```
<?php

    $font_number = 4;
    $text = "No worries.";

    $width = 2 * strlen($text) * imagefontwidth($font_number);
    $height = 3 * imagefontheight($font_number);

    $image = imagecreate($width, $height);

    $back_color = imagecolorallocate($image, 200, 200, 200);
```

```
    $drawing_color = imagecolorallocate($image, 0,   0,   0);

    $x_position = ($width - (strlen($text) * imagefontwidth($font_number)))/ 2;
    $y_position = ($height - imagefontheight($font_number)) / 2;

    imagestring($image, $font_number, $x_position, $y_position, $text,
        $drawing_color);

    header('Content-Type: image/jpeg');
    imagejpeg($image);
    imagedestroy($image);

?>
```

We'll embed phpdrawtext.php in phpdrawtext.html this way:

```
<HTML>
    <HEAD>
        <TITLE>Drawing text</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>
                Drawing text
            </H1>
            <BR>
            <IMG SRC="phpdrawtext.php">
        </CENTER>
    </BODY>
</HTML>
```

You can see the results in Figure 11. Very cool.

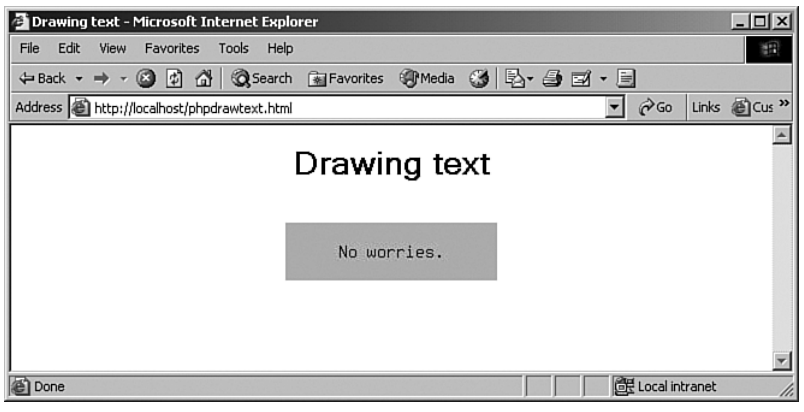

FIGURE 11 Drawing text.

Drawing Text Vertically

You can draw text with `imagestring`—horizontally, anyway. What about drawing text vertically, as when you want to label the *y*-axis of a graph? Use the `imagestringup` function:

```
imagestringup(resource image, int font, int x, int y, string s, int color)
```

This function draws the string *s* vertically in the image specified by *image* at coordinates *x*, *y* in color *color*. If *font* is 1, 2, 3, 4, or 5, a built-in font is used.

To use this function in code, we can reverse the height and width from the example in the previous chunk to create a vertical image:

```
$width = 3 * imagefontheight($font_number);
$height = 2 * strlen($text) * imagefontwidth($font_number);
```

Now that you're flipping *x* and *y* coordinates, you also have to change the - to a + in the calculation of the *y* position:

```
$x_position = ($width - imagefontheight($font_number)) / 2;
$y_position = ($height + (strlen($text) * imagefontwidth($font_number))) / 2;
```

Then you draw the text using `imagestringup`, as shown in phpdrawtextup.php, which appears in Example 11.

EXAMPLE 11 Drawing text vertically, phpdrawtextup.php

```
<?php

    $font_number = 4;
    $text = "No worries.";

    $width = 3 * imagefontheight($font_number);
    $height = 2 * strlen($text) * imagefontwidth($font_number);

    $image = imagecreate($width, $height);

    $back_color = imagecolorallocate($image, 200, 200, 200);
    $drawing_color = imagecolorallocate($image, 0,   0,   0);

    $x_position = ($width - imagefontheight($font_number)) / 2;
   $y_position = ($height + (strlen($text) * imagefontwidth($font_number)))/2;

    imagestringup($image, $font_number, $x_position, $y_position, $text,
        $drawing_color);

    header('Content-Type: image/jpeg');
    imagejpeg($image);
```

```
    imagedestroy($image);

?>
```

We'll embed phpdrawtextup.php in phpdrawtextup.html like this:

```
<HTML>
    <HEAD>
        <TITLE>Drawing text vertically</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>
                Drawing text vertically
            </H1>
            <BR>
            <IMG SRC="phpdrawtextup.php">
        </CENTER>
    </BODY>
</HTML>
```

The results, including the vertical text, appear in Figure 12.

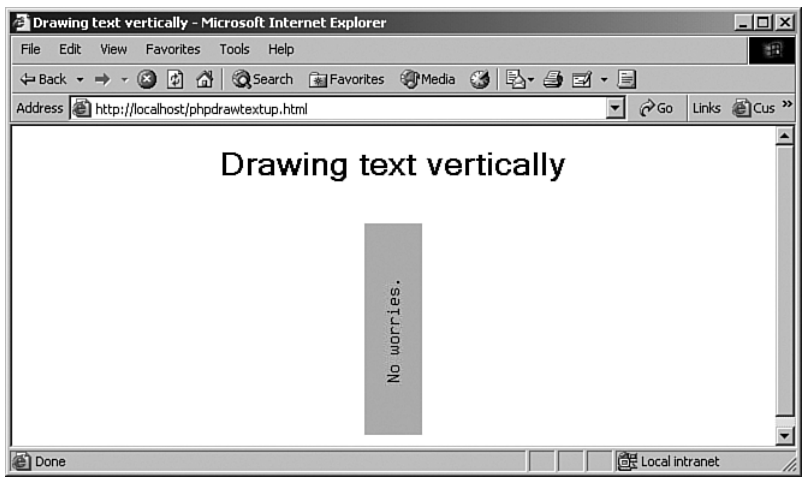

FIGURE 12 Drawing text vertically.

Embedding Existing Images

You can create graphics image objects from existing image files using these functions:

- `imagecreatefromgif`. Create a new image from GIF file or URL
- `imagecreatefromjpeg`. Create a new image from JPEG file or URL
- `imagecreatefrompng`. Create a new image from PNG file or URL
- `imagecreatefromwbmp`. Create a new image from WBMP file or URL
- `imagecreatefromxbm`. Create a new image from XBM file or URL
- `imagecreatefromxpm`. Create a new image from XPM file or URL

This is great when you want to embed images in web pages but also want to add something to them yourself. You also can use this function to embed an image of yourself or a logo inside another image. As an example, we'll use `imagecreatefromjpeg` here, loading an existing JPEG image, image.jpg, and adding a border to it. Here's how you use this function:

```
imagecreatefromjpeg (string filename)
```

This function returns an image identifier representing the image obtained from the given filename, and it returns an empty string on failure.

In this example, we plan to read in image.jpg. This image just displays the word "Cool!" against a green background, and we'll add to it by drawing a black border around the image when we create a new JPEG. We'll also create a black drawing color:

```
$image = imagecreatefromjpeg ("image.jpg");
$back_color = imagecolorallocate($image, 200, 200, 200);
$draw_color = imagecolorallocate($image, 0, 0, 0);
```

You can use `imagesx($image)` and `imagesy($image)` to determine the dimensions of an image, so here's how we draw a black border around the image, using `imagerectangle`:

```
$image = imagecreatefromjpeg ("image.jpg");
$back_color = imagecolorallocate($image, 200, 200, 200);
$draw_color = imagecolorallocate($image, 0, 0, 0);

imagerectangle($image, 10, 10, imagesx($image) - 10, imagesy($image) - 10,
    $draw_color);
```

This all appears in phpimage.php, Example 12.

EXAMPLE 12 Adding to an existing image, phpimage.php

```
<?php
    $image_height = 100;
    $image_width = 300;
    $image = imagecreatefromjpeg ("image.jpg");
    $back_color = imagecolorallocate($image, 200, 200, 200);
    $draw_color = imagecolorallocate($image, 0, 0, 0);
    imagerectangle($image, 10, 10, imagesx($image) - 10, imagesy($image) - 10,
        $draw_color);
    header('Content-Type: image/jpeg');
    imagejpeg($image);
    imagedestroy($image);
?>
```

We'll embed the new image in an HTML document, phpimage.html:

```
<HTML><HEAD><TITLE>Using existing images</TITLE></HEAD>
    <BODY><CENTER>
            <H1>Using existing images</H1>
            <BR>
            <IMG SRC="phpimage.php">
        </CENTER></BODY>
</HTML>
```

You can see the new images, complete with their new border, in Figure 13.

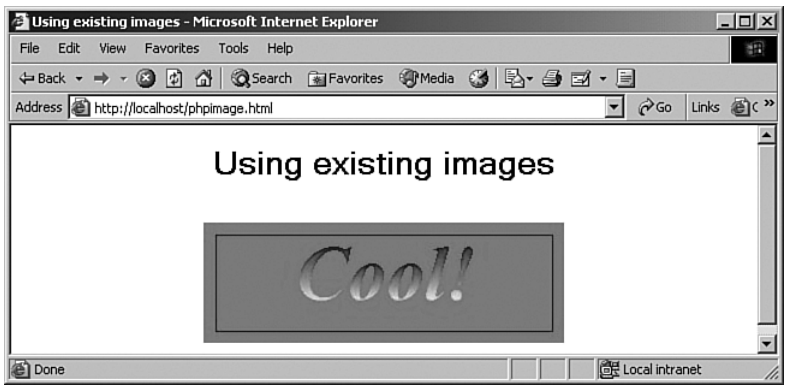

FIGURE 13 Embedding an image.

Flipping an Image

More graphics manipulation power is available; for example, the `imagecopy` function lets you copy all or part of an image:

```
imagecopy ( resource dest_image, resource src_image, int dest_x, int dest_y, int
src_x, int src_y, int src_w, int src_h)
```

This function copies a part of *src_image* onto *dest_image* starting at the coordinates *src_x*, *src_y* with a width of *src_w* and a height of *src_h*. The portion defined will be copied onto the coordinates *dest_x* and *dest_y*.

This function is very handy when you want to manipulate images on a pixel-by-pixel basis. Here's an example where we'll flip an image, image.jpg (from the previous chunk), horizontally. We read in that image with `imagecreatefromjpeg` and get its size with `imagesx` and `imagesy`:

```
$image_original = imagecreatefromjpeg("image.jpg");

$image_width = imagesx($image_original);
$image_height = imagesy($image_original);
```

Then we can create a new image of the same dimensions, `$image_new`, and set up a nested `for` loop to loop over every pixel in the image:

```
$image_original = imagecreatefromjpeg("image.jpg");

$image_width = imagesx($image_original);
$image_height = imagesy($image_original);

$image_new = imagecreate($image_width, $image_height);

for ($col = 0 ; $col < $image_width ; $col++)
{
    for ($row = 0 ; $row < $image_height ; $row++)
    {
        .
        .
        .
    }
}
```

Finally, we'll use `imagecopy` inside this nested `for` loop to copy individual pixels from one image to the new image, reversing the image as we do so, as you can see in phpimage-flip.php, Example 13.

EXAMPLE 13 Flipping an image, phpimageflip.php

```
<?

    $image_original = imagecreatefromjpeg("image.jpg");

    $image_width = imagesx($image_original);

    $image_height = imagesy($image_original);

    $image_new = imagecreate($image_width, $image_height);

    for ($col = 0 ; $col < $image_width ; $col++)
    {
        for ($row = 0 ; $row < $image_height ; $row++)
        {
            imagecopy($image_new, $image_original, $image_width - $col - 1,
                $row, $col, $row, 1, 1);
        }
    }

    imagejpeg($image_new);

    imagedestroy($image_original);
    imagedestroy($image_new);

?>
```

The flipped image appears in Figure 14, phpimageflip.html. Not bad. As you can see, this kind of pixel-by-pixel graphics manipulation is very powerful.

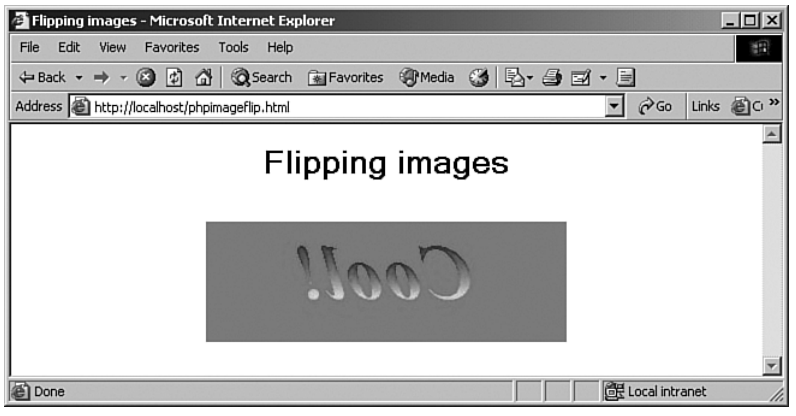

FIGURE 14 Flipping an image.

Tiling Images

You can also use one image to tile another image by appearing repeatedly in the background. Here's how you use this function:

```
imagesettile(resource image, resource tile)
```

This function sets the tile image to be used by all region-filling functions (such as `imagefilledrectangle` and `imagefilledpolygon`) when filling with the special color `IMG_COLOR_TILED`.

Here's an example, where we're going to tile an image with our image.jpg image and display some text on top of the results. First, we create a new image—we're going to use the `imagecreatetruecolor` function here, not just `imagecreate`, to preserve the image in the tiles as much as possible—and then use `imagecreatefromjpeg` to create the tile we'll use to cover the background:

```
$image_width = 300;
$image_height = 200;
$image = imagecreatetruecolor($image_width, $image_height);
$tile = imagecreatefromjpeg ('image.jpg');
```

Now we're able to use `imagesettile` to tile the background of our image and then draw a filled rectangle with multiple tiles using `imagefilledrectangle`:

```
$image_width = 300;
$image_height = 200;
$image = imagecreatetruecolor($image_width, $image_height);
$tile = imagecreatefromjpeg ('image.jpg');
imagesettile($image, $tile);
imagefilledrectangle ($image, 0, 0, $image_width, $image_height,
    IMG_COLOR_TILED);
```

Then we use `imagestring` to add some text to the result, as you see in phptile.php, Example 14.

EXAMPLE 14 Tiling images, phptile.php

```
<?php
    $image_width = 300;
    $image_height = 200;
    $image = imagecreatetruecolor($image_width, $image_height);
    $tile = imagecreatefromjpeg ('image.jpg');
    imagesettile($image, $tile);
    imagefilledrectangle ($image, 0, 0, $image_width, $image_height,
        IMG_COLOR_TILED);
    $drawing_color = imagecolorallocate($image, 0, 0, 0);
```

```
     imagestring($image, 4, 20, 90, 'No worries....', $drawing_color);

     Header("Content-type: image/jpeg");
     imagejpeg($image);
     imagedestroy ($image);
     imagedestroy ($tile);
?>
```

We'll embed the results in an HTML document, phptile.html:

```
<HTML>
    <HEAD>
        <TITLE>Tiling images</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>Tiling images</H1>
            <BR>
            <IMG SRC="phptile.php">
        </CENTER>
    </BODY>
</HTML>
```

You can see what everything looks like in Figure 15. Nice.

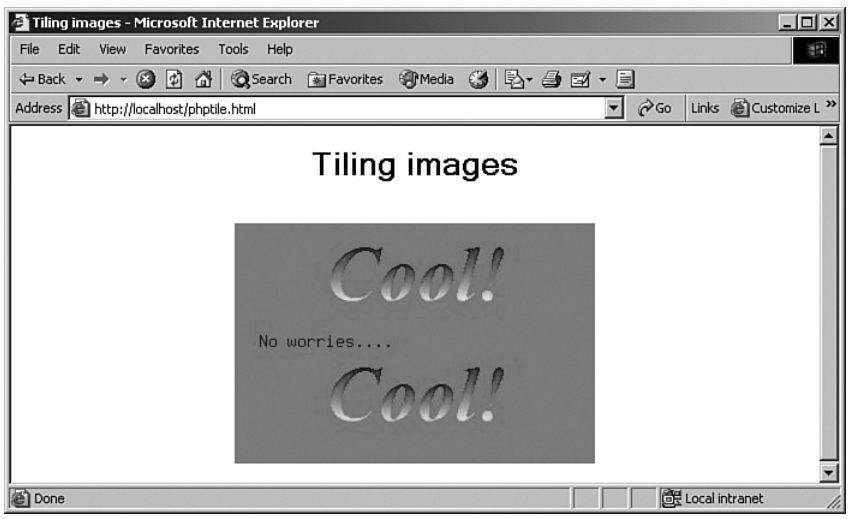

FIGURE 15 Tiling images.

Doing Gamma Corrections

The graphics functions let you adjust the *gamma* of images as well, which effectively adjusts their brightness, with `imagegammacorrect`:

```
imagegammacorrect(resource image, float inputgamma, float outputgamma)
```

This function applies gamma correction to an image given an input gamma, the parameter *inputgamma*, and an output gamma, the parameter *outputgamma*.

You can see an example that will lighten our image, image.jpg, and display the result in phpgamma.php, Example 15.

EXAMPLE 15 Adjust image gamma, phpgamma.php

```
<?php

    $image_height = 100;
    $image_width = 300;
    $image = imagecreatefromjpeg ("image.jpg");
    imagegammacorrect($image, .5, 2.5);
    header('Content-Type: image/jpeg');
    imagejpeg($image);

    imagedestroy($image);
?>
```

We'll embed the result in phpgamma.html, and you can see it in Figure 16.

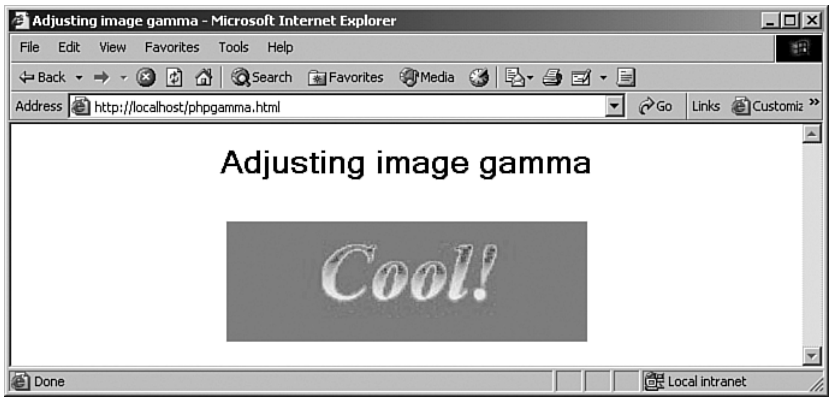

FIGURE 16 Adjusting image gamma.

Creating a Gradient

The GD graphics package doesn't come with functions to draw graphics gradients that will let you create shaded figures; however, you can do the job yourself. For example, say that you want a sphere that looks three-dimensional. You can create that yourself with a little bit of work—in this case, by drawing successive filled circles of different colors to make it appear shaded.

First, we'll create a `for` loop that loops over every radius needed to create the figure, pixel-by-pixel. Then we'll create a new color for each iteration through the loop, getting brighter toward the center:

```
$image_width = 200;

for ($loop_index = 0; $loop_index <= $image_width / 2; $loop_index++)
{
    $current_color = imagecolorallocate($image, 2 * 255 * $loop_index /
        $image_width, 2 * 255 * $loop_index / $image_width, 0);
        .
        .
        .
}
```

When you have your drawing color, draw a filled ellipse with `imagefilledellipse` like this:

```
$image_width = 200;

for ($loop_index = 0; $loop_index <= $image_width / 2; $loop_index++)
{
    $current_color = imagecolorallocate($image, 2 * 255 * $loop_index /
        $image_width, 2 * 255 * $loop_index / $image_width, 0);

    imagefilledellipse($image, $image_width / 2, $image_width / 2,
        $image_width - 2 * $loop_index, $image_width - 2 * $loop_index,
        $current_color);
}
```

All the code appears in phpgradient.php, Example 16, where we're creating the 3D sphere in a JPEG image.

EXAMPLE 16 Drawing a gradient, phpgradient.php

```
<?php

    $image_width = 200;

    $image = imagecreate($image_width, $image_width);
    $back_color = imagecolorallocate($image, 255, 255, 255);
```

continues

EXAMPLE 16 continued

```
    for ($loop_index = 0; $loop_index <= $image_width / 2; $loop_index++)
    {
        $current_color = imagecolorallocate($image, 2 * 255 * $loop_index /
            $image_width, 2 * 255 * $loop_index / $image_width, 0);

        imagefilledellipse($image, $image_width / 2, $image_width / 2,
            $image_width - 2 * $loop_index, $image_width - 2 * $loop_index,
            $current_color);
    }

    header('Content-Type: image/jpeg');
    imagejpeg($image);

    imagedestroy($image);

?>
```

You can see the results in Figure 17, where the sphere does indeed have a 3D appearance. Very cool.

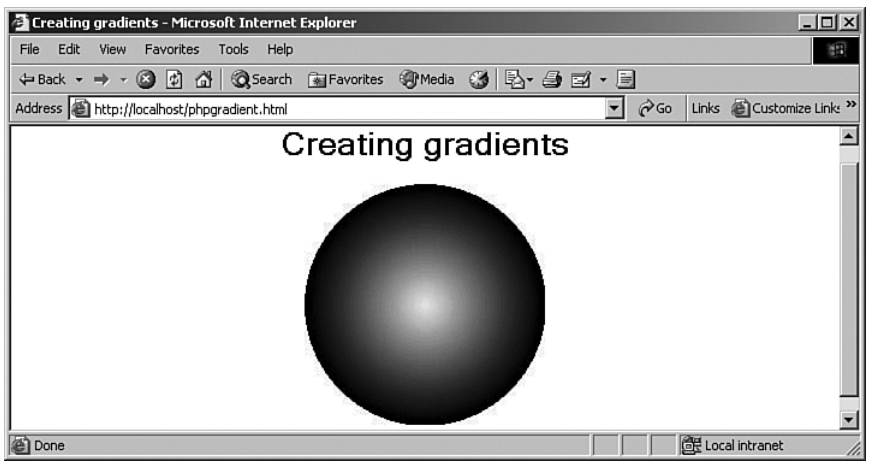

FIGURE 17 Drawing a gradient.

Summary

This chapter showed how to create graphics on the server. Being able to create graphics interactively is very powerful for interacting with the user. Here are some of this chapter's salient points:

- You can create an image in memory with the `imagecreate` function.
- To set the background and drawing colors, use the `imagecolorallocate` function.
- To indicate to the browser that you're creating an image file, such as a JPEG file, use the `header('Content-Type: image/jpeg')` function.
- To actually create the image, use a function such as `imagejpeg($image)`.
- To embed an image in an HTML document, use an element such as `<IMG SRC="phpscript.php">`.
- You can draw lines with the `imageline` function.
- You can draw rectangles with the `imagerectangle` function.
- You can draw ellipses with the `imageellipse` function.
- You can draw polygons with the `imagepolygon` function.
- You can draw arcs with the `imagearc` function.
- You can draw individual pixels with the `imagesetpixel` function.
- You can fill in figures with functions such as `imagefilledellipse`.
- You can draw text with the `imagestring` function.
- You can correct an image's gamma with the `imagegammacorrect` function.