

Getting graphic with PHP

Create and manipulate images using the PHP GD library

Skill Level: Intermediate

[Ken Bluttman \(ken@kenbluttman.com\)](mailto:ken@kenbluttman.com)

Freelance Web Developer
Freelance Consultant

01 Jul 2008

Imagine creating Web-page graphics dynamically using just code — no need for a complex graphics program. Creating and manipulating images is yours for the doing with the power of PHP. Put your coding skills to work generating images for the Web.

Section 1. Before you start

You're advised to do a refresher of PHP basic programming methods before you begin this tutorial. Knowledge of image and graphics processing is helpful. You should understand pixels, shapes, graphic file types, resizing, and similar graphics-manipulation concepts.

About this tutorial

This tutorial steps through using the GD library, showing you how to create and alter images on Web pages. It starts with the basic GD construct, then builds on it to showcase various graphics techniques.

Objectives

The purpose of this tutorial is to show you how to get started with the GD library and

to provide a variety of techniques you can apply to your Web-page coding. The tutorial doesn't cover every GD function, but it teaches you the basics. The [Resources](#) section provides Web sites where you can learn more about using the GD library.

Prerequisites

A working knowledge of PHP will go a long way in making this tutorial easy to follow. You should also have general programming knowledge. The particular syntax requirements, although similar to some other programming languages, may take some study.

System requirements

Web server

Any operating system and any Web server can be used. Feel free to use Apache V2.X, or the IBM® HTTP Server. Download from [Apache](#) or [IBM](#).

PHP

PHP V5.1.4 or later is required.

GD

As of PHP V4.3, a version of the GD library comes bundled with new PHP installations. You won't need to worry about this requirement if you're using a later version. Use `phpinfo()` to see if your PHP installation is GD-enabled. The section "Introducing GD" discusses the history and versions of GD, and where to find the library.

Section 2. Introducing GD

PHP is a widely used Web server-based programming language. A preponderance of Internet sites and intranets use PHP for a variety of tasks. Working with a database is a well-known application of PHP.

This tutorial focuses on a different part of PHP's ability and power: It shows you how to use the GD library to create and manipulate graphics. This library provides numerous options that let you create shapes, such as rectangles, ellipses, lines, and arcs. In addition, several methods let you work with popular graphic file types, such as GIF, JPG, and PNG.

The tutorial takes you step by step through creating and changing images — both vector and raster. *Vector graphics* consist of rectangles, lines, circles, etc. *Raster graphics* typically are photographs or file types considered fixed to their image (bitmaps, GIFs, JPEGs, etc.). For the purpose of this tutorial, you don't need to know the details of vectors and rasters.

Check your GD library version

Vectors and rasters

Vectors are visual representations of mathematical formulas. You don't see the math, per se, because built-in functions are used as wrappers around the raw calculations. An advantage of vector graphics is that they maintain their sharpness regardless of resizing. Raster graphics, such as photographs saved as JPEGs, lose sharpness as they're resized larger. Making them smaller keeps the sharpness.

Multiple versions of the GD library are available. Installations of PHP V5 have V2.x of the library bundled with the PHP installation. Earlier versions of PHP may have a 1.x GD version; GD was an extra install in the older versions of PHP, so it's possible GD isn't installed. Your first task is to see which version is running on your Web server.

The GD function `gd_info` returns information about the GD library. Follow these steps:

1. Listing 1 shows a short routine that reports the library's details. Enter the code and save the file as `gd_info.php`. (You can name the file as you wish, as long as it has the `.php` extension.)

Listing 1. Using `gd_info`

```
<?php
$array=gd_info ();
foreach ($array as $key=>$val) {
    if ($val===true) {
        $val="Enabled";
    }
    if ($val===false) {
        $val="Disabled";
    }
    echo "$key: $val <br />\n";
}
?>
```

2. Upload the file to your Web server and navigate to the page through a browser. Following is a sample result as viewed in a browser.

Listing 2. A sample result

```
GD Version: bundled (2.0.34 compatible)
FreeType Support: Enabled
FreeType Linkage: with freetype
T1Lib Support: Disabled
GIF Read Support: Enabled
GIF Create Support: Enabled
JPG Support: Enabled
PNG Support: Enabled
WBMP Support: Enabled
XPM Support: Disabled
XBM Support: Enabled
JIS-mapped Japanese Font Support: Disabled
```

More information about PHP

Just as `gd_info()` returns information about the GD installation, `php_info()` returns information about your entire PHP setup. Put the single statement `phpinfo();` in a PHP file and run it.

The first line returns the GD version. In this case, the version is 2.0.34, but referring to it as 2.0 suffices. Some of the functions used in this tutorial are available only with GD V2.x. The tutorial is written with the assumption that you're running GD V2.x.

If you find that your PHP installation doesn't include the GD library, see [Resources](#) to download it.

Section 3. Create your first image

It's time to roll up your sleeves and get creative. This section demonstrates some basic graphics that incorporate elements of PHP — primarily looping.

Enter each example into a text-based editor and save it with a PHP extension (.php), then place it on your Web server and access it through a browser.

Start with the basics

Listing 3 shows a basic PHP GD script. A minimal number of GD statements are used to create a simple graphic.

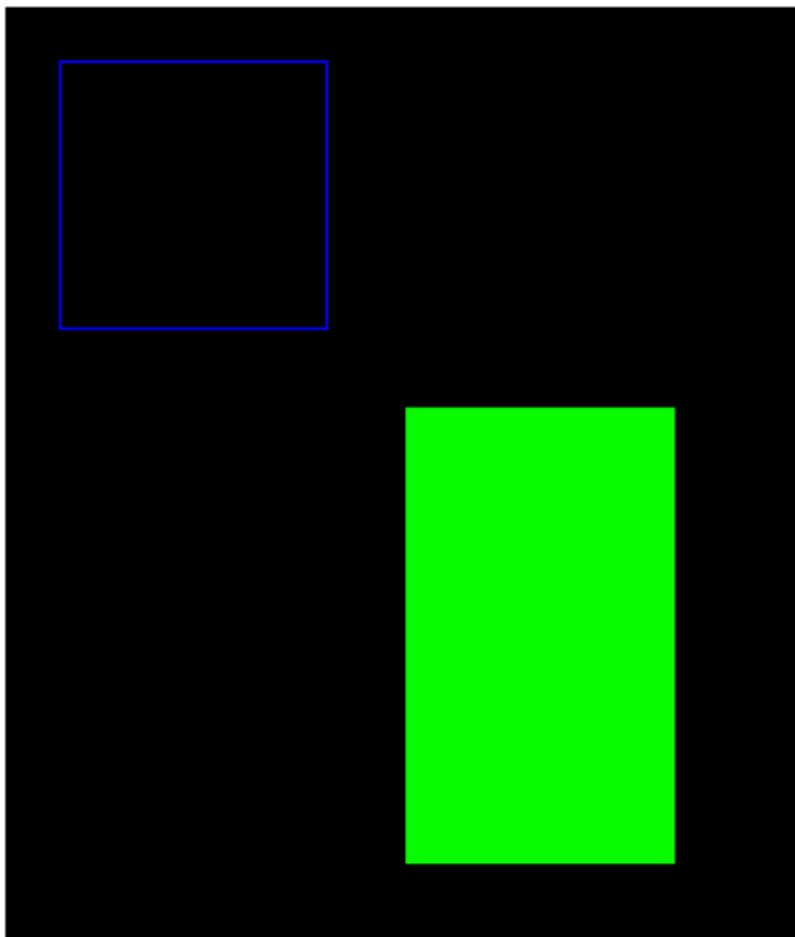
Listing 3. A basic GD script

```
<?php
```

```
// file type
header("Content-type: image/png");
// create image
$img = imagecreatetruecolor(300, 350);
// set colors
$blue = imagecolorallocate($img, 0, 0, 255);
$green = imagecolorallocate($img, 0, 255, 0);
// draw two rectangles, one is filled
imagerectangle($img, 20,20,120,120, $blue);
imagefilledrectangle($img, 150,150,250,320, $green);
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

Save the code in Listing 3 as a PHP file and run it from your Web server. The browser produces the graphic shown in Figure 1.

Figure 1. A simple graphic containing an image



Let's see how this works:

About RGB

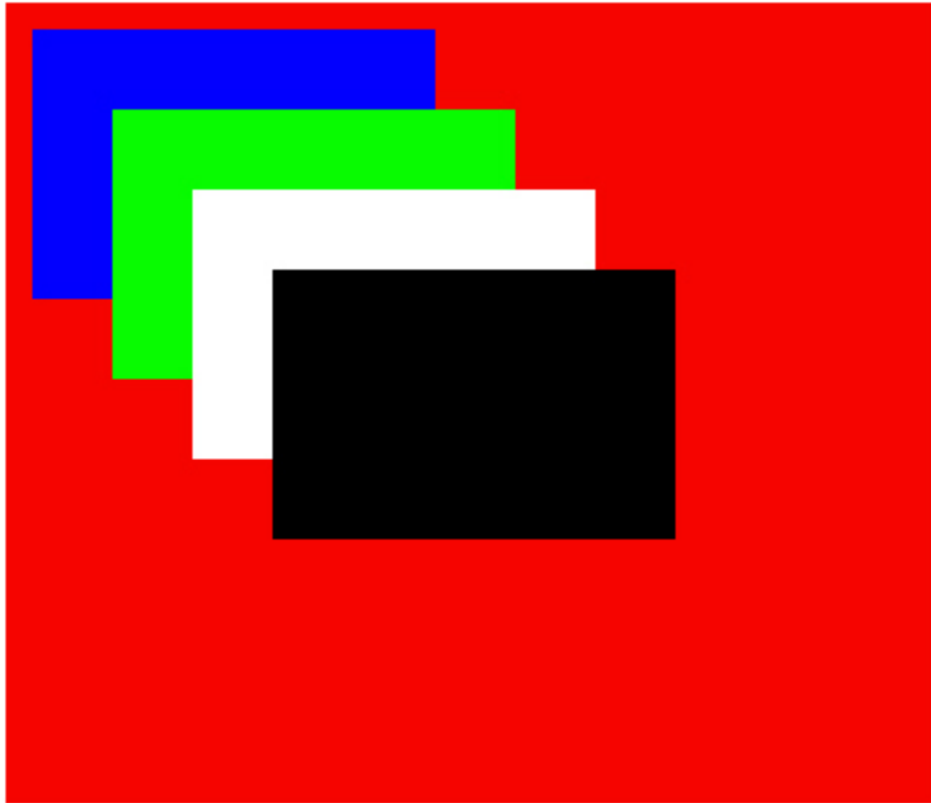
RGB is a standard color function used in many programming languages. The acronym stands for Red, Green, Blue. A number from 0 to 255 is applied for each of these three basic colors. For example, `RGB(255, 0, 0)` creates red. Mixing values creates all variations of colors. `RGB(0, 0, 0)` creates black, and `RGB(255, 255, 255)` creates white.

1. A file type is indicated. Referring to the example GD installation, PNG, GIF, and JPG are available. Here, a PNG file type is selected.
2. The `imagecreatetruecolor()` function is the workhorse that gets the graphic going. This essentially creates the canvas on which images are drawn. The two parameters of the function are the width and height of the canvas, in pixels. The `imagecreatetruecolor` function returns a handle. In this example, a variable named `$img`. `$img` is then used to link other functions to the canvas. By default, the canvas is black.
3. Colors are set with the `imagecolorallocate` function. The parameters here are the image handle (`$img`) and the color. In this case, the standard RGB method is used to indicate the color (see the sidebar "About RGB").
4. `imagerectangle` draws a nonfilled rectangle, and `imagefilledrectangle` draws a filled one. Both use the same parameters: the image handle, the coordinates (the first two values determine the upper-left corner, and the last two determine the lower-right corner), and the color. Note that the colors are selected using predetermined colors created with `imagecolorallocate`.
5. `imagepng(handle)` displays the image, and `imagedestroy` clears the memory used in processing the image. It doesn't remove the image from the browser.

Lay it on with layering

Let's make this example more interesting. Figure 2 shows a red canvas with four rectangles on it. Note how the rectangles overlap, providing a layered effect.

Figure 2. Overlapping rectangles



The code that created this graphic appears in Listing 4.

Listing 4. Creating rectangles

```
<?php
// file type
header ("Content-type: image/png");
// create image
$img = imagecreatetruecolor(350, 300);
// define colors
$red = imagecolorallocate($img, 255, 0, 0);
$blue = imagecolorallocate($img, 0, 0, 255);
$green = imagecolorallocate($img, 0, 255, 0);
$white = imagecolorallocate($img, 255, 255, 255);
$black = imagecolorallocate($img, 0, 0, 0);
// set background color
imagefill($img, 0, 0, $red );
// draw filled rectangles
imagefilledrectangle($img, 10,10,160,110, $blue);
imagefilledrectangle($img, 40,40,190,140, $green);
imagefilledrectangle($img, 70,70,220,170, $white);
imagefilledrectangle($img, 100,100,250,200, $black);
// display the image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

This example introduces a new function: `imagefill`. This function sets the canvas to a designated color — red, in this case. The four parameters are the image handle, the left and top coordinates, and the color.

Looking at the four `imagefilledrectangle` functions, a placement pattern is evident. The rectangles overlap based on the values used to determine the upper-left corner. The upper-left coordinate increases in each successive rectangle, but not enough to avoid the rectangle before it. This creates the overlap. The second set of numbers, which determine the bottom-right corners, increase by the same amount as the top coordinate values. This keeps the rectangles the same size.

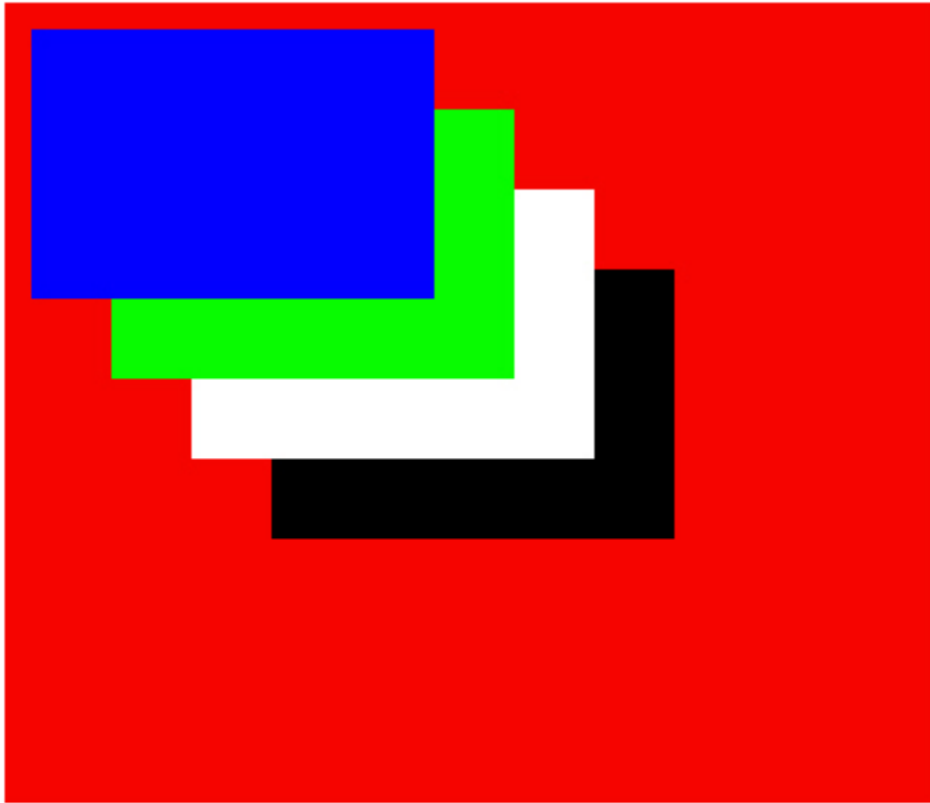
Listing 5 is nearly identical to Listing 3. The difference is that it lists four rectangles in the opposite order, which creates the appearance of the overlap working in the opposite direction.

Listing 5. Reordering the rectangles

```
<?php
// file type
header ("Content-type: image/png");
// create image
$img = imagecreatetruecolor(350, 300);
// define colors
$red = imagecolorallocate($img, 255, 0, 0);
$blue = imagecolorallocate($img, 0, 0, 255);
$green = imagecolorallocate($img, 0, 255, 0);
$white = imagecolorallocate($img, 255, 255, 255);
$black = imagecolorallocate($img, 0, 0, 0);
// set background color
imagefill($img, 0, 0, $red );
// draw filled rectangles
imagefilledrectangle($img, 100,100,250,200, $black);
imagefilledrectangle($img, 70,70,220,170, $white);
imagefilledrectangle($img, 40,40,190,140, $green);
imagefilledrectangle($img, 10,10,160,110, $blue);
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

Figure 3 shows how the rectangles now appear. At first glance, it appears the rectangles have changed direction, but it's just sleight of code. The rectangles are in the same spots as in Figure 2, but the overlap effect is reversed and gives a uniquely different perspective.

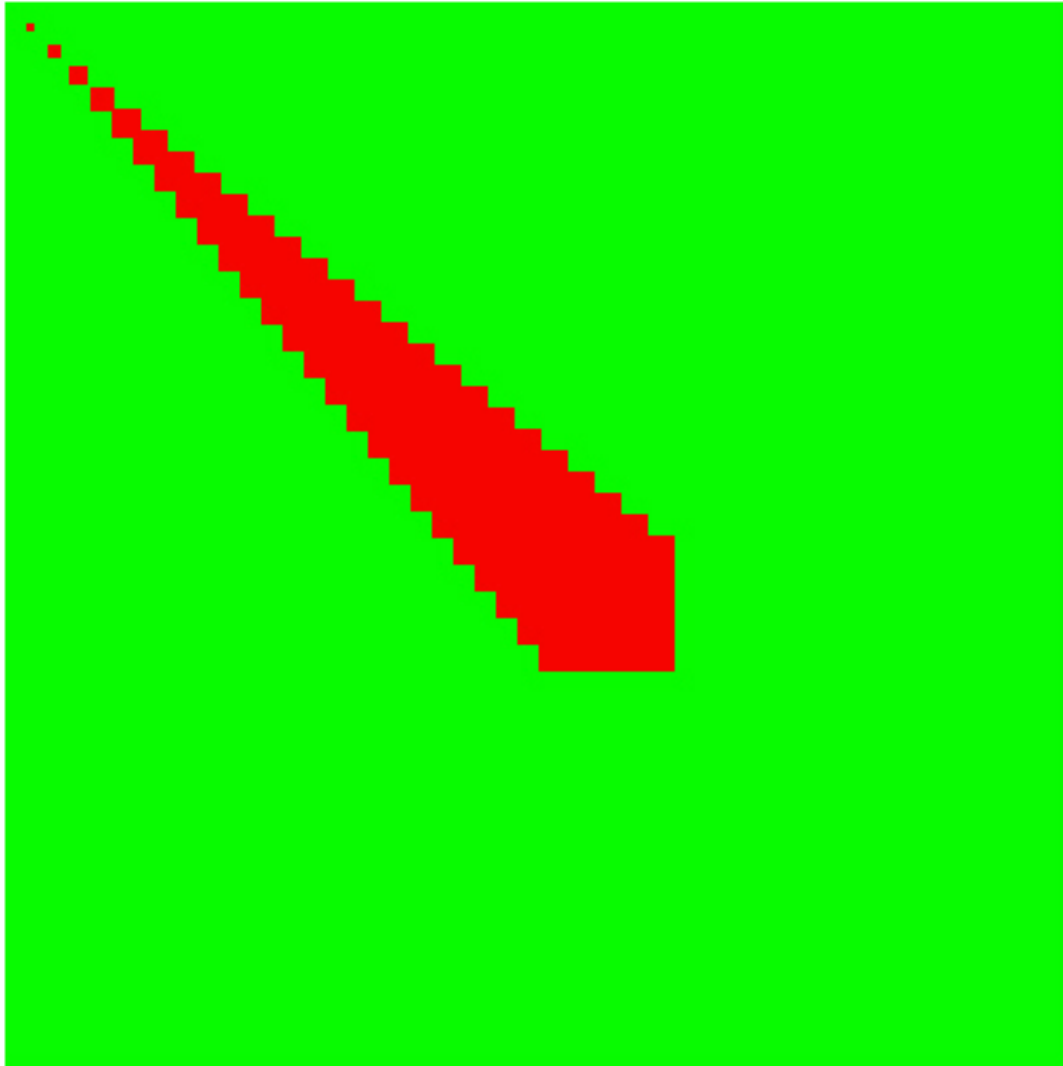
Figure 3. Rectangles reversed



Add a little math

You can create the image in Figure 4 with a set of red rectangles that increase in size and overlap, creating a unique image. It looks something like a necktie, although that wasn't the intention.

Figure 4. Succession of growing rectangles



Listing 6 shows the code that produces this image.

Listing 6. Using a loop to control size and placement

```
<?php
// file type
header("Content-type: image/png");
// create image
$img = imagecreatetruecolor(400, 400);
// set colors
$green = imagecolorallocate($img, 0, 255, 0);
$red = imagecolorallocate($img, 255, 0, 0);
// set background to green
imagefill($img, 0, 0, $green );
for ($i=1;$i<=25;$i++) {
    imagefilledrectangle($img, $i * 8, $i * 8, $i * 10, $i * 10, $red);
}
// display the image
imagepng($img);
```

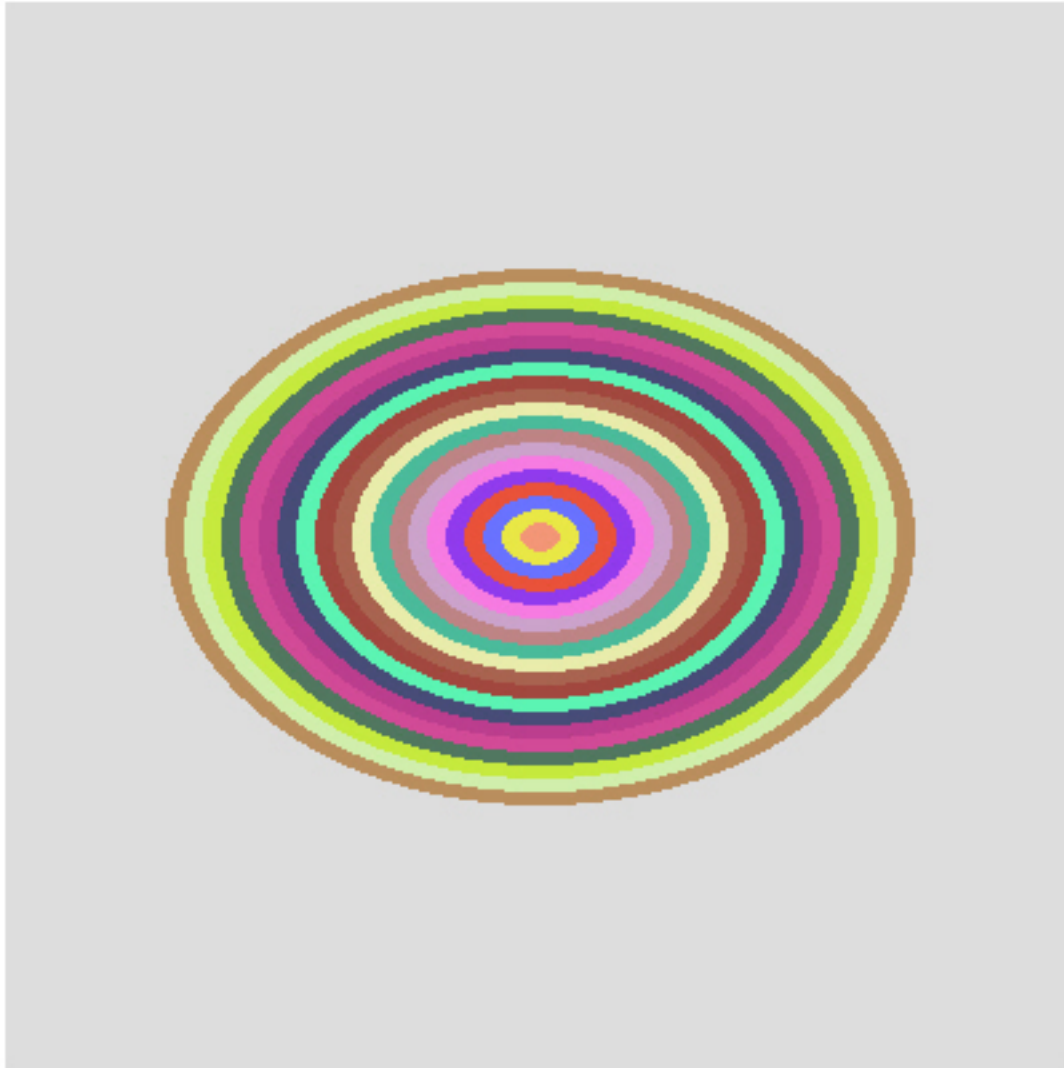
```
// release image from memory
imagedestroy($img);
?>
```

Listing 6 uses initial coordinate values and a `for` loop to create the succession of red triangles. The `imagefill` function uses green in this example to color the canvas.

Go circular

Figure 5 shows another image you can create with a looping mechanism. But, lo and behold, it moves away from rectangles and, instead, uses ellipses. An ellipse can be either a circle or an oval, depending on the settings.

Figure 5. Ellipses



You create the ovals using a succession of ellipses that shrink toward the center of the canvas. The colors are randomly generated. Each time you refresh the browser window, a different color scheme appears. The nitty-gritty is shown below.

Listing 7. Creating randomly colored ellipses

```
<?php
// file type
header("Content-type: image/png");
// create image
$img = imagecreatetruecolor(400, 400);
// set background to light gray
imagefill($img,0,0,imagecolorallocate($img, 221, 221, 221));
// draw ellipses with random colors
for ($i=20;$i>=1;$i--) {
    imagefilledellipse($img, 200,200,$i*14,$i*10,
        imagecolorallocate($img, rand(50,255), rand(50,255), rand(50,255)));
}
```

```
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

Note that the `for` loop is structured to run from the largest value down. If the loop value increased, the final ellipse would cover all the others. By running the loop from largest to smallest, you can see all the ellipses.

The longish code line (shown here as two lines to fit on this page)

```
imagefilledellipse($img, 200,200,$i*14,$i*10,
imagecolorallocate($img, rand(50,255), rand(50,255), rand(50,255)));
```

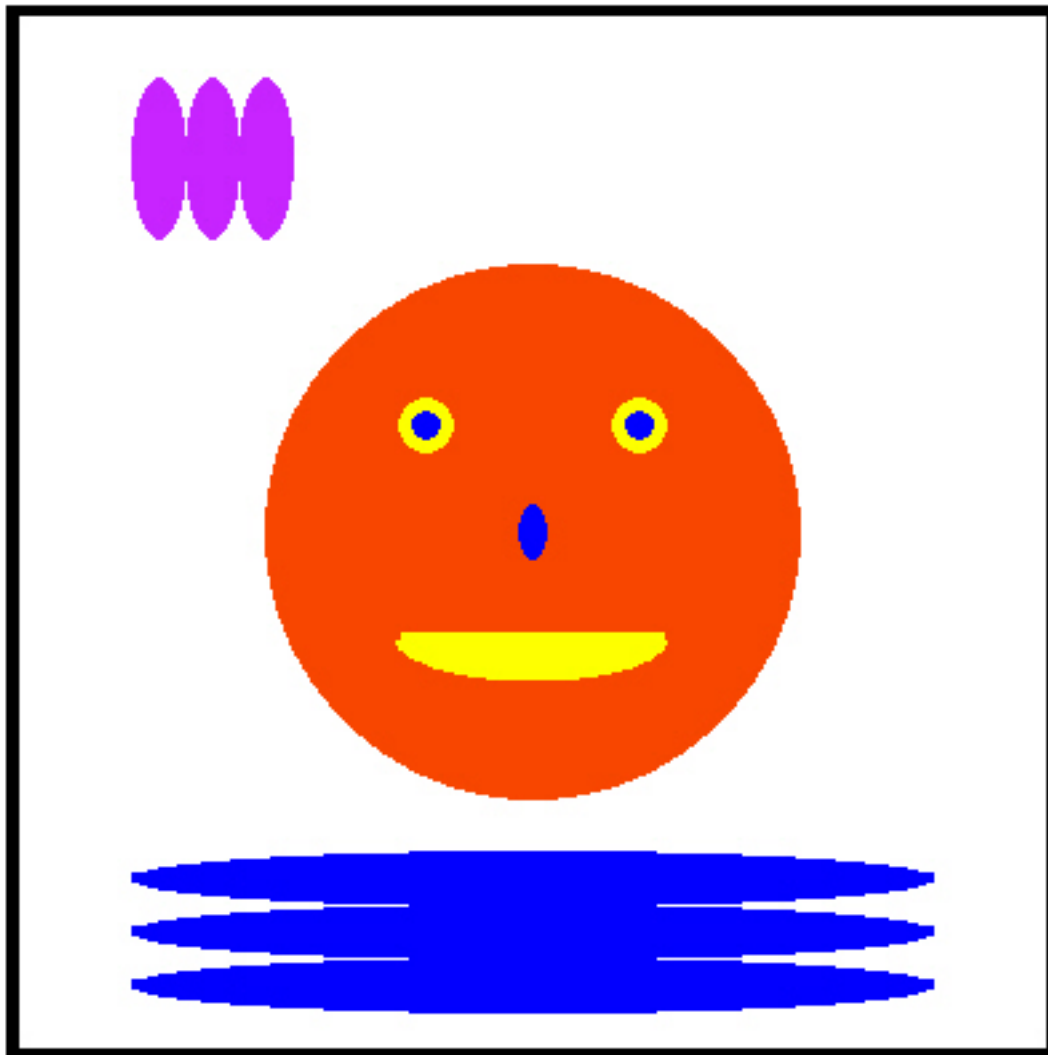
uses the `imagefilledellipse` function to create each ellipse. The parameters are typical: the image handle, the starting coordinates, and the width and height. Note that the coordinates of an ellipse indicate the middle of the ellipse, not a point along the edge. With a canvas sized at 400x400, the ellipses generate from the center of the canvas, at position 200,200.

You randomly generate the color of each ellipse by using the PHP `rand` function in conjunction with the RGB settings. For example, `rand(50,255)` returns a value between 50 and 255. Recall that RGB uses values between 0 and 255, so a random range of 50 through 255 works fine.

When the two parameters that define the width and height are equal, the resulting image is a circle. In Listing 6, the width is always larger than the height, creating horizontal oval shapes.

Controlling the placement and shape of the ellipses provides a variety of artistic options. Figure 6 shows a few of these possibilities. The face is created by laying down an orange circle. Each eye consists of two circles: an outer yellow circle and an inner blue circle. The nose is an ellipse in which the height parameter is larger than the width parameter.

Figure 6. Being creative with ellipses



The mouth is made with an arc. This tutorial hasn't discussed arcs, but you can check [Resources](#) to learn more about them. Briefly, the arc is created with `imagefilledarc($img, 200, 240, 100, 30, 0, 180, $yellow, IMG_ARC_PIE)`. Like ellipses, arcs give you the flexibility to make many shapes.

Figure 6 includes two sets of ellipses: three vertically aligned purple ones and three horizontally aligned blue ones. They illustrate the relation of the width and height parameters in the group of `imagefilledellipse` functions.

Also note the black border, which you can create by setting the thickness to 4 pixels and drawing an unfilled rectangle slightly within the dimensions of the canvas. A new function is introduced here: `imagesetthickness`. This function declares the thickness of lines, borders of rectangles, ellipses, etc. `imagesetthickness` takes two parameters: the image handle and the thickness, in pixels. Listing 7 contains the code that produces the image in Figure 6.

Listing 8. Drawing with ellipses

```
<?php
// file type
header("Content-type: image/png");
// create image
$img = imagecreatetruecolor(400, 400);
// set background to white
imagefill($img,0,0,imagecolorallocate($img, 255, 255, 255));
// set colors
$black = imagecolorallocate($img, 0, 0, 0);
$blue = imagecolorallocate($img, 0, 0, 255);
$purple = imagecolorallocate($img, 204, 33, 255);
$orange = imagecolorallocate($img, 255, 66, 0);
$yellow = imagecolorallocate($img, 255, 255, 0);
// draw a border
imagesetthickness($img,4);
imagerectangle($img, 5,5,395,395,$black);
// draw a face
imagefilledellipse($img, 200,200,200,200,$orange);
imagefilledellipse($img, 160,160,20,20,$yellow); // left eye
imagefilledellipse($img, 160,160,10,10,$blue); // left eye
imagefilledellipse($img, 240,160,20,20,$yellow); // right eye
imagefilledellipse($img, 240,160,10,10,$blue); // right eye
imagefilledellipse($img, 200,200,10,20,$blue); // nose
imagefilledarc($img, 200, 240, 100, 30, 0, 180, $yellow, IMG_ARC_PIE);
// draw three touching vertical ovals
imagefilledellipse($img, 60,60,20,60,$purple);
imagefilledellipse($img, 80,60,20,60,$purple);
imagefilledellipse($img, 100,60,20,60,$purple);
// draw three touching horizontal ovals
imagefilledellipse($img, 200,330,300,20,$blue);
imagefilledellipse($img, 200,350,300,20,$blue);
imagefilledellipse($img, 200,370,300,20,$blue);
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

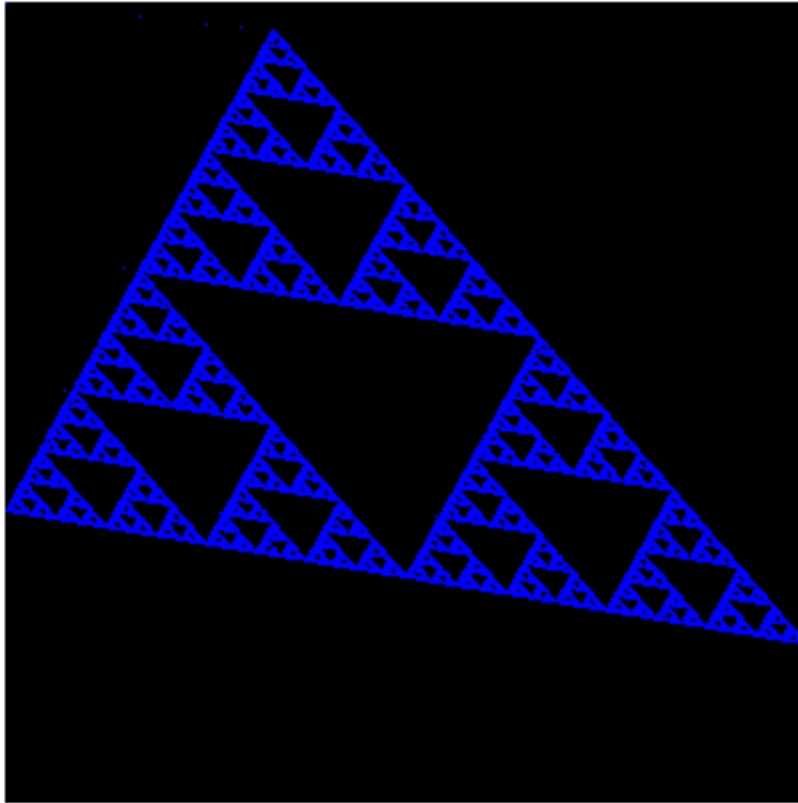
Section 4. Gettin' pixely with it

A *pixel* is the basic single point on the screen. A screen resolution of 1024x768 literally means there are 1,024 pixels to access in a single row, and 768 pixels in a single column. That means you have 1,024 x 768 = 786,432 pixels to work with. (The usable number is less because the browser window toolbars, borders, etc. take up some room.)

The `imagesetpixel` function sets a pixel to a given color. The parameters are the image handle, the x (horizontal) coordinate, the y (vertical) coordinate, and the color. The line of code `imagesetpixel($img, 700, 550, $blue)` sets the pixel at location 700, 500 to blue. Note that the coordinates are based on the upper-left corner of the browser window having the coordinates 0,0.

Figure 7 shows an interesting pattern made from 100,000 pixel settings. Using mathematical formulas causes the pixels to be placed in such a manner that the shape appears. Altering the math would product a different shape.

Figure 7. Pixels set into patterns



Listing 9 contains the code used to produce Figure 7. You use a loop that iterates 100,000 times in conjunction with various plot points stored in the `$pixels` array. The variable `$a` randomly selects points from the three elements of the array.

Listing 9. Working with pixels

```
<?php
// image type
header('Content-Type: image/png');
// create image
$img = imagecreatetruecolor(300, 300);
// create plot arrays
$pixels[0] = array('x' => 100, 'y' => 10);
$pixels[1] = array('x' => 0, 'y' => 190);
$pixels[2] = array('x' => 300, 'y' => 240);
// set color
$blue = imagecolorallocate($img, 0, 0, 255);
// draw 100,000 pixels, thereby creating a shape
for ($i = 0; $i < 100000; $i++) {
    imagesetpixel($img, round($x), round($y), $blue);
    $a = rand(0, 2);
    $x = ($x + $pixels[$a]['x']) / 2;
}
```

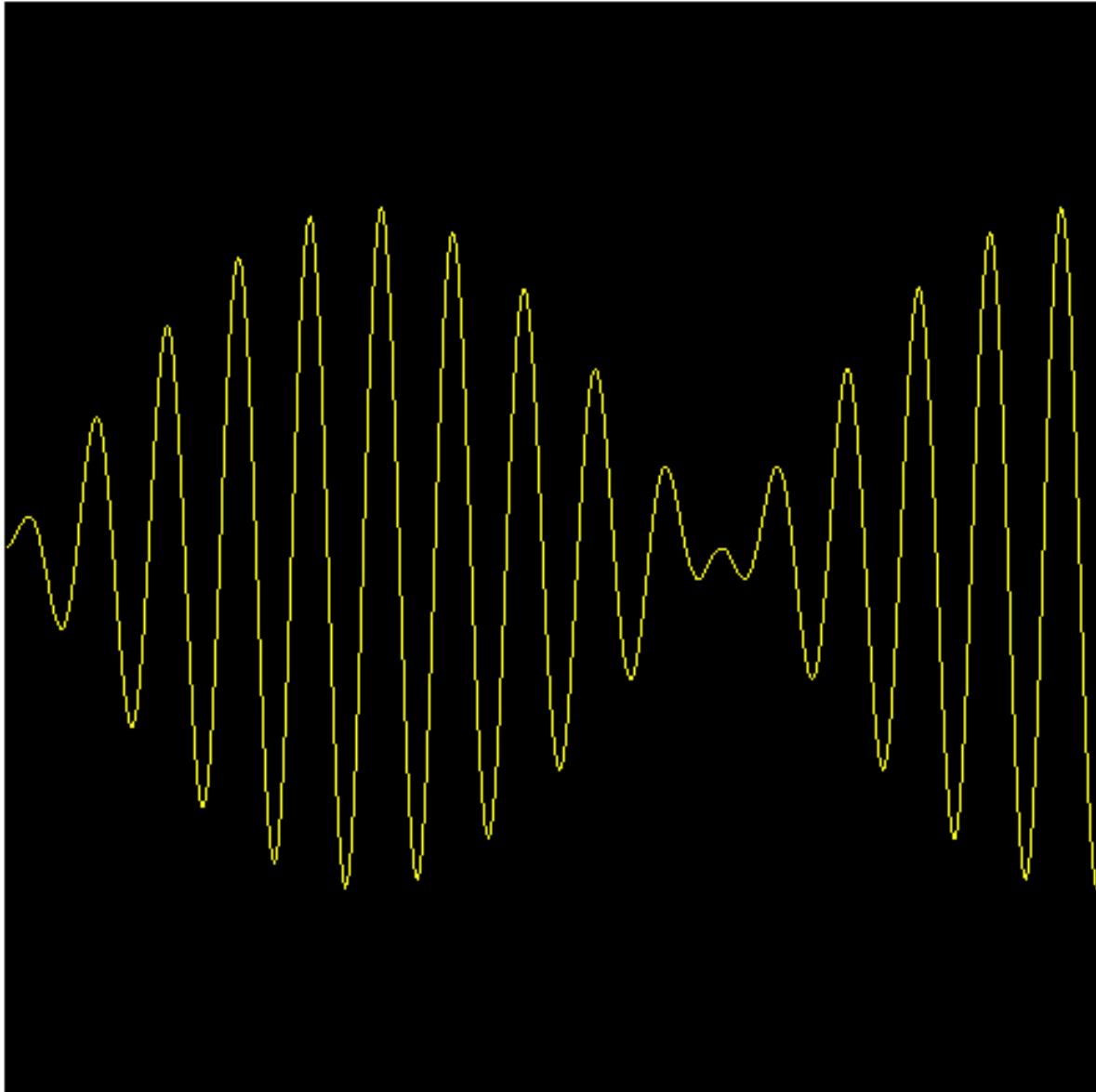


```
$y = ($y + $pixels[$a]['y']) / 2;
}
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

Section 5. A look at lines

Lines are straight by nature, but many small lines, each angled a little from the one before, combine to make a longer, nonstraight line. Still working with math and a loop, Figure 8 shows a continuous sine wave assembled from many smaller lines.

Figure 8. Arranging lines into a sine wave



Listing 9 shows the code to create the sine wave. The `imageLine` function does the work by plotting hundreds of small lines, whose angles are determined by the `sin` function.

Listing 10. Drawing lines

```
<?php
// file type
header ("Content-type: image/png");
// create image
$img = @imagecreatetruecolor(480, 480);
$yellow = imagecolorallocate($img, 255, 255, 33);
// fill array with coordinates
$points = array();
```

```

for ($i=1; $i<480; $i=$i+1) {
    $x = $i;
    $y = 150*sin($x/100)*sin($x/5);
    $points[] = $x;
    $points[] = 240-$y;
}
$totalPoints = count($points)/2;
for ($i=0; $i<$totalPoints-1; $i++) {
    imageLine($img, $points[2*$i], $points[1+2*$i],
    $points[2+2*$i], $points[3+2*$i], $yellow);
}
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>

```

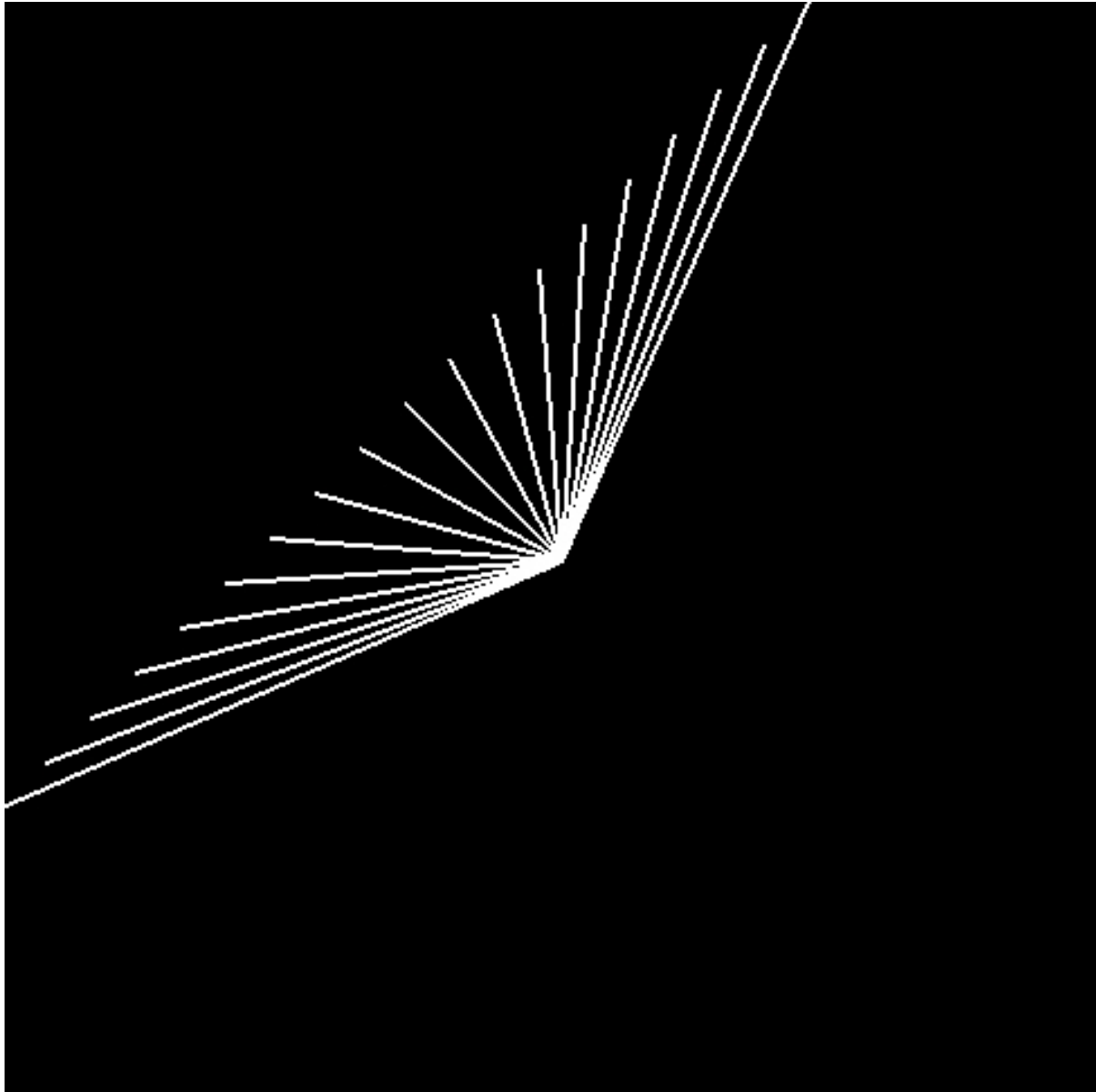
The `imageLine` function takes six parameters:

- The image handle
- The x and y starting coordinates of the line
- The x and y ending coordinates of the line
- The color

Note that the difference between the starting coordinates and the ending coordinates determines the length of a line. However, the relationship of all four coordinate parameters must be taken into account to determine the length. For example, a line that begins at 100,50 and ends at 200,50 has a length of 100 pixels. This is easy to determine because the y value is the same in the starting and ending coordinate sets. A line with parameters such as 100, 50, 200, 90 is longer than 100 pixels because both the x and y values change. Whenever this occurs, the line is at an angle, having some degree of diagonal placement.

Figure 9 shows another way you can use line drawing. In this case, the individual lines are easy to see.

Figure 9. Using lines to create a ray effect



Listing 11 shows the code you use to create the image in Figure 9.

Listing 11. Thicker lines

```
<?php
// image type
header ("Content-type: image/png");
// create image
$img=imagecreatetruecolor(500, 500);
// set how thick the lines will be: 2 pixels
imagesetthickness($img, 2);
// draw white lines
for ($i=0;$i<=360;$i=$i+20){
    imageline($img,250,250,360-$i,$i,
        imagecolorallocate($img,255,255,255));
}
```

```
}  
// display image  
imagepng($img);  
// release image from memory  
imagedestroy($img);  
?>
```

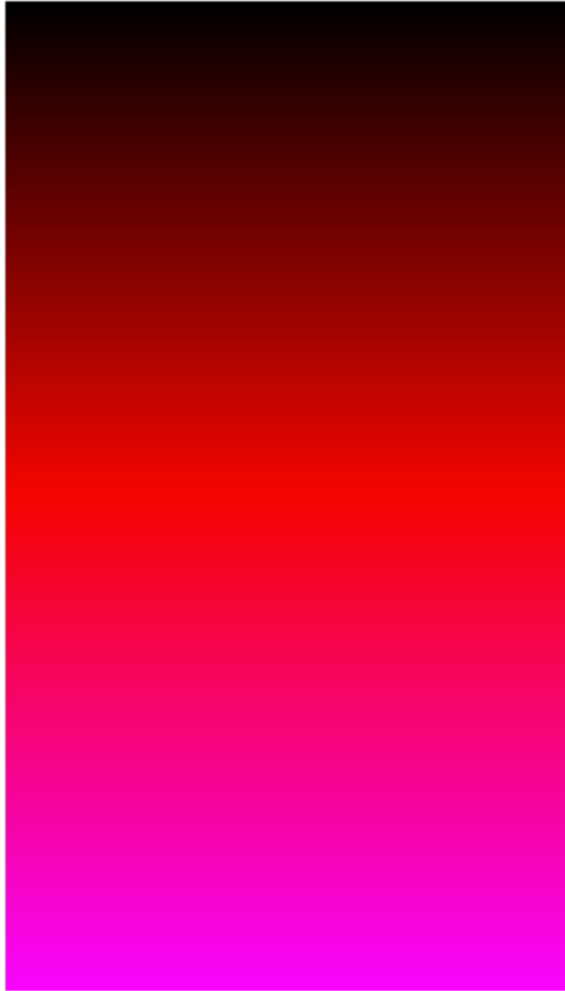
Section 6. Be radiant with a gradient

GD doesn't support gradients. But no matter. A workaround will do.

A *gradient* is a smooth transition between colors. It can include more complexity than that, such as a transition that skews toward one color over the other. But a smooth change between two colors is what you're after.

You can dynamically control colors and lines, so it's feasible to create a flow from color to color while lining up lines next to each other. With no space between the lines, the group looks like a single shape, as shown below.

Figure 10. Creating a smooth transition between colors



Listing 12 contains the code you use create the gradient. Actually, the code creates two gradients, one under the other. From top to bottom, the graphic flows from black to red, and from red to purple. In the middle of the code, one loop ends and another begins. Each loop creates a transition effect.

Listing 12. Create a gradient

```
<?php
// file type
header("Content-type: image/png");
// create image
$img = imagecreatetruecolor(400, 800);
// set background to white
imagefill($img,0,0,imagecolorallocate($img, 255, 255, 255));
//cycle through colors while drawing lines
for ($i=0;$i<=255;$i++) {
    imageline($img, 10, $i+10, 300, $i+10, imagecolorallocate($img,$i, 0, 0));
}
for ($i=0;$i<=255;$i++) {
    imageline($img, 10, $i+266, 300, $i+266, imagecolorallocate($img, 255, 0, $i));
}
```

```
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

You apply a loop that counts from 0 to 255 to an RGB construct. The loop increases the first parameter, `red`, from 0 to 255, thus creating a transition from black to red. While the value for red is being incremented, the same loop creates successive lines from top to bottom.

The second loop keeps the pattern going, but this time, the `red` value is left at 255 and the `blue` value increases from 0 to 255. Red and blue make purple. So purple is the color toward the end of the transition. The second loop places lines where the first loop ended.

Section 7. Just for the effect of it

GD has an interesting function that lets you apply special effects to an image. The `imageconvolution` function takes an array of parameter arrays, a divisor, and an offset value. The relationship of these values is complex, and experimenting with different values is a great way to learn. You can also search for information about this function on the Web and see [Resources](#) for more information about using it.

Figure 11 shows how to make a picture appear embossed. If it's hard to tell, the picture is a cat sleeping in a sink.

Figure 11. Embossing a cat with the `imageconvolution` function



Listing 13 shows the code you can use to create this effect.

Listing 13. Applying effects to images

```
<?php
$img = imagecreatefromjpeg('catnap.jpg');
$emboss = array(array(2, 0, 0), array(0, -1, 0), array(0, 0, -1));
imageconvolution($img, $emboss, 1, 127);
header('Content-Type: image/png');
imagepng($img);
imagedestroy($img);
?>
```

Section 8. Lively text

GD contains a few functions that let you work with text and fonts. Figure 12 shows one example, which uses the PHP `rand` function to randomly size, color, and position a message. "Good Morning" appears randomly thanks to a loop that iterates 50 times.

Figure 12. A very Good Morning



Listing 14 shows how to create this text effect. The `imagestring` function is the powerhouse here. It has these parameters:

- The image handle
- A value between 1 and 5 for the font size
- Two coordinate values that position the text
- The text string
- The color of the text, here created with random RGB settings

Listing 14. Working wonders with text

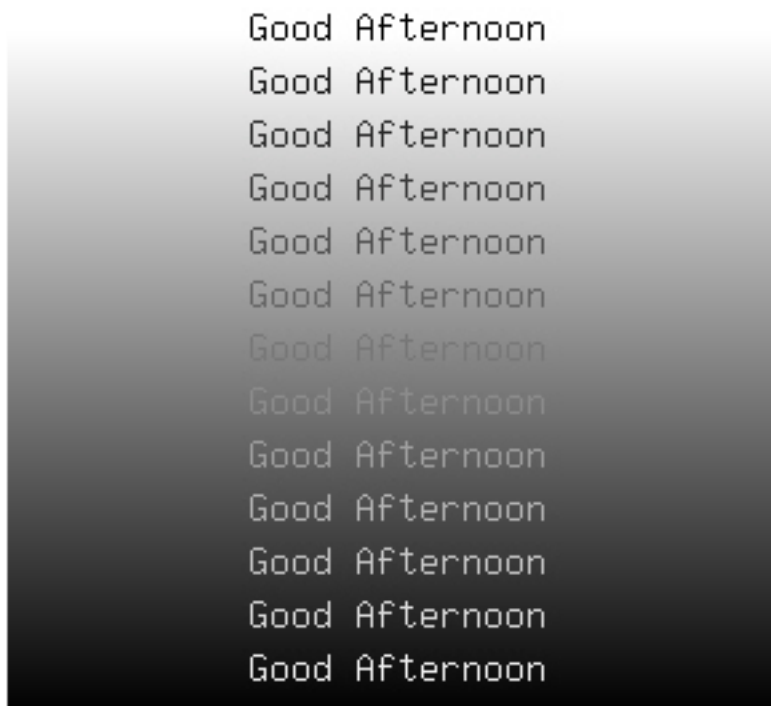
```
<?php
header("Content-type: image/png");
$img = imagecreate(500, 500);
```

```
$white=imagecolorallocate($img, 255, 255, 255);
$blue=imagecolorallocate($img, 0, 0, 255);
for ($i=1;$i<=50;$i++) {
    imagestring($img, rand(1, 5),rand(0, 425),rand(0, 425),"Good Morning",
    imagecolorallocate($img, rand(0, 255), rand(0,255), rand(0, 255)));
}
imagepng($img);
imagedestroy($img)
?>
```

Even using just a single `imagestring` function, a number of possibilities come to mind. One is to apply a random color to each letter. To do this, you can handle each letter in a separate `imagestring` function, placing the first character randomly and then placing each successive character a few points to the right.

The next text-treatment example creates two gradients that offset each other. Figure 13 shows a gradient background that shifts from white to black; layered on top is a text string that shifts from black to white.

Figure 13. Gradient text on top of a gradient background



As demonstrated back in Listing 12, you create a gradient effect by using a `for` loop to draw contiguous lines, with the color of each line progressing slightly from the one before. In Listing 15, you shift the gradient from white to black by cycling all the RGB components from 0 to 255. Once the gradient is complete, another `for` loop cycles the `imagestring` function, changing the color from black to white. (Actually, it changes to off-white. The loop ends at a value of 240, instead of 255 because the loop increments by 20. The next value would be 260, which is past the greatest

possible value of 255.)

Listing 15. Gradient effect applied to text

```
<?php
// file type
header("Content-type: image/png");
// create image
$img = imagecreatetruecolor(400, 400);
// set background to white
imagefill($img,0,0,imagecolorallocate($img, 255, 255, 255));
//cycle through colors while drawing lines
for ($i=0;$i<=255;$i++) {
    imageline($img, 10, $i+10, 300, $i+10, imagecolorallocate($img,255-$i, 255-$i, 255-$i));
}
// cycle colors while placing text
for ($i=0;$i<=240;$i=$i + 20) {
    imagestring($img, 4,100,$i,"Good Afternoon",imagecolorallocate($img, $i,$i,$i));
}
// display image
imagepng($img);
// release image from memory
imagedestroy($img);
?>
```

Section 9. Save your creations

You can run a PHP file and see the output anytime in your browser. But what if you want to create a graphics file of the output and save it separately?

Listing 16 shows is a variation of Listing 15. In Listing 16, the selected file type is JPEG, so the resulting saved file is of that type. The twist that allows you to save the output is to give the `imagejpeg` function a filename (or a path and filename). Here, the output is saved as `myartwork.jpg`. The output is the same as in Figure 13.

Listing 16. Saving a created image

```
<?php
// file type
header("Content-type: image/jpeg");
// create image
$img = imagecreatetruecolor(400, 400);
// set background to white
imagefill($img,0,0,imagecolorallocate($img, 255, 255, 255));
//cycle through colors while drawing lines
for ($i=0;$i<=255;$i++) {
    imageline($img, 10, $i+10, 300, $i+10, imagecolorallocate($img,255-$i, 255-$i, 255-$i));
}
// cycle colors while placing text
for ($i=0;$i<=240;$i=$i + 20) {
    imagestring($img, 4,100,$i,"Good Afternoon",imagecolorallocate($img, $i,$i,$i));
}
}
```

```
// remove file if it is already there
if (file_exists("myartwork.jpg")) {
    unlink ("myartwork.jpg");
}
// save the image as a jpeg
imagejpeg($img, "myartwork.jpg");
// release image from memory
imagedestroy($img);
?>
```

Section 10. Work with existing graphics

Until now, this tutorial has shown you how to create vector graphics. Now it's time to focus on manipulating an existing file. In this case, a JPEG. You must have the JPEG on your Web server to work with it. You'll do this the fun way — instead of placing a file on the server, you'll use an HTML form to upload the file. When the input type on a form is file, a browse feature is available that lets you find a file on your computer to upload. Figure 14 shows the Web page that contains this feature. You can use this page to get the file and later to manipulate it. For the purpose of uploading the file, the top half of the page is the focus.

Figure 14. Using a form to upload a file

Upload a photo:

Use the Browse button to find your picture file on your computer.

Then click the Upload button.

The file must be either a JPEG (.jpg) or a GIF (.gif).

The file must be less than 2,000,000 bytes.

Upload this file:

C:\picture1.JPG	<input type="button" value="Browse..."/>	<input type="button" value="Upload"/>
-----------------	--	---------------------------------------

Change a photo:



Rotation:

Resizing:

Figure 14 shows that a file has already been selected with the Browse feature. Click the **Upload** button to copy the file to the server. This can take a minute or two, depending on the file size and the speed of your connection. The code in this case limits the file size to 2 million bytes. In addition, you can upload only JPG or GIF files. Listing 17 shows the code for the Web page.

Listing 17. A Web page for uploading and changing pictures: Name this file form1.php

```
<html>
<head><title>GD and Photos</title></head>
<body>
<table Width="85%">
<tr>
<td width="5%"></td>
<td><h3>Upload a photo:</h3>
```

```

Use the Browse button to find your picture file on your computer.
<br />Then click the Upload button.
<br />The file must be either a JPEG (.jpg) or a GIF (.gif).
<br />The file must be less than 2,000,000 bytes.<br /><br /></td>
</tr>
<?php
if ($_REQUEST['Error']=="true") {
echo '<tr>';
echo '<td width="5%"></td>';
echo '<td><b>The file you just uploaded was either '
echo 'too large or the wrong type of file. The file must be a '
echo '.jpg or a .gif; and be smaller than 2,000,000 bytes.</b>';
echo '</td></tr>';
}
?>
<tr><td> </td>
<td>
<form enctype="multipart/form-data" action="pixupload.php" method="POST">
  <!-- MAX_FILE_SIZE must precede the file input field -->
  <input type="hidden" name="MAX_FILE_SIZE" value="2000000" />
  <!-- Name of input element determines name in $_FILES array -->
  Upload this file:<br /><input size="40" id="userfile" name="userfile"
  type="file" accept="image/gif, image/jpeg" />
  <input type="submit" value="Upload" />
</form>
<hr />
<h3>Change a photo:</h3>
</td></tr>
<tr><td> </td><td></td></tr>
<tr><td> </td><td>
<form name="process" id="process" action="processphoto.php">
Rotation:
<select id="rotation" name="rotation">
<option value="0">None</option>
<option value="90">90 degrees</option>
<option value="180">180 degrees</option>
<option value="270">270 degrees</option>
</select>
<br />
Resizing:
<select id="resize" name="resize">
<option value="100">100px</option>
<option value="200">200px</option>
<option value="300">300px</option>
</select>
<br /><br />
<input type="submit" value="Process Photo" /></form>
</td></tr>
</table>
</body>
</html>

```

In the code, the following line limits the file size: `<input type="hidden" name="MAX_FILE_SIZE" value="2000000" />`. The `accept` attribute of the input element limits the file type: `<input size="40" id="userfile" name="userfile" type="file" accept="image/gif, image/jpeg" />`. After the file is uploaded, it appears on the page.

Figure 15. The uploaded picture appears on the page

Upload a photo:

Use the Browse button to find your picture file on your computer.

Then click the Upload button.

The file must be either a JPEG (.jpg) or a GIF (.gif).

The file must be less than 2,000,000 bytes.

Upload this file:

Change a photo:



Rotation:

Resizing:

Listing 18 shows the server code that processes the uploaded file. This procedure checks the file type, resizes the image, and saves the file. At the end, `echo Header (Location: form1.php");` redirects back to the page with the form.

Listing 18. Server code that processes an uploaded file: Name this file pixupload.php

```

<?php
$bad_ext=="true";
// This is the temporary file created by PHP
$uploadedfile = $_FILES['userfile']['tmp_name'];
//get the extension to determine the file type
$ext=exif_imagetype($uploadedfile);
//IMAGETYPE_GIF
if ($ext==1) {
    // Create an Image from it so we can do the resize
    $src = @imagecreatefromgif($uploadedfile) or die ($errmsg);
    $bad_ext="false";
}
//IMAGETYPE_JPEG
if ($ext==2) {
    // Create an Image from it so we can do the resize
    $src = @imagecreatefromjpeg($uploadedfile) or die ($errmsg);
    $bad_ext="false";
}
if ($bad_ext=="false") {
    // Capture the original size of the uploaded image
    list($width,$height)=@getimagesize($uploadedfile);
    // The image is resized, while keeping the height and width ratio intact
    $newwidth=200;
    $newheight=($height/$width)*200;
    $tmp=@imagecreatetruecolor($newwidth,$newheight);
    // this line resizes the image, copying the original image into $tmp
    imagecopyresampled($tmp,$src,0,0,0,0,$newwidth,$newheight,$width,$height);
    // The resized image is written to disk.
    $filename = $_FILES['userfile']['name'];
    if ($ext==1) {
        imagegif($tmp,$filename,100);
        $newfilename="myphoto.gif";
    }
    if ($ext==2) {
        imagejpeg($tmp,$filename,100);
        $newfilename="myphoto.jpg";
    }
    imagedestroy($src);
    imagedestroy($tmp);
    // remove file if it is already there
    if (file_exists($newfilename)) {
        unlink ($newfilename);
    }
    rename($filename,$newfilename);
    echo header("Location: form1.php");
} else {
    echo header("Location: form1.php?Error=true");
}
?>

```

This code includes some new functions:

- `exif_imagetype` — Returns the file type
- `imagecopyresampled` — Resizes the image

The upload process works by setting a temporary file name to the uploaded file. Later, the `rename` function names the file `myphoto.jpg`. You can make the uploaded file retain its original name by setting the new filename to `$_FILES['userfile']['name']`.

The form includes options that let you change the image's size and rotation. Figure

16 shows selections have been made to rotate the image 90 degrees and to increase the size to 300 pixels.

Figure 16. Preparing to make changes to the picture

Change a photo:





Rotation: 
Resizing: 

Figure 17 shows the result. The image is now on its side, and it's grainy because it was enlarged. Although this result doesn't appear useful, it serves as an illustrative lesson.

Figure 17. A sideways, enlarged picture



Listing 19 shows the code that processes the image rotation and resizing.

Listing 19. Server code that resizes and rotates an image: Name this file processphoto.php

```
<?php
// File and rotation
$filename = 'myphoto.jpg';
// Content type
header('Content-type: image/jpeg');
// Get new sizes
list($width, $height) = getimagesize($filename);
$newwidth=$_REQUEST['resize'];
$newheight=($height/$width)*$_REQUEST['resize'];
// Load
$dest = imagecreatetruecolor($newwidth, $newheight);
$source = imagecreatefromjpeg($filename);
// Resize
imagecopyresized($dest,$source,0,0,0,0,$newwidth,$newheight,$width,$height);
imagejpeg($dest, $filename);
$source = imagecreatefromjpeg($filename);
// Rotate
$degrees = $_REQUEST['rotation'];
$rotate = imagerotate($source, $degrees, 0);
// Output
imagejpeg($rotate);
?>
```

You use the following new functions in Listing 19:

- `getimagesize` — Returns an image's width and height.
- `imagecopyresize` — Resizes an existing image. Its parameters are the destination (resized) file, the original file, and four coordinates to indicate how much of the original file should be included in the resized file. Using all 0's defaults to the entire image. The last four parameters are the new size attributes and the original size attributes.
- `imagerotate` — Rotates an existing image. Its parameters are the file, the degree of rotation, and a value indicating what color should be used in the uncovered area that can appear when an image is rotated. The value of 0 defaults to the background color.

More to explore

Several functions let you work further with raster images, including:

- `getimagesize` — Returns the dimensions and file type of a specified image
- `imagecolorat` — Returns the color at a specified coordinate position
- `imagecolortransparent` — Sets the transparent color in a given image; useful with GIF files
- `imagefilter` — Applies a number of effects to an image

You can find information about these and all the other functions at Web sites dedicated to PHP and the GD library (see [Resources](#)).

Summary

This tutorial took you through the steps of using the GD library to apply a number of graphics-based functions. Use GD to create graphics, alter existing artwork, and apply special effects to artwork.

At last count, the GD library included slightly more than 100 functions. This tutorial gave you the basics to get started. Some functions, such as `imagefilledellipse`, `imagefilledarc`, and `imageconvolution` open the door to some fantastic graphical possibilities. The beauty of the GD functions is that they can work with variables instead of just hard-coded values. This lets you apply looping and other programming constructs.

Use GD to create thumbnails from uploaded pictures, showing how an item would appear in different colors and much more. And this tutorial barely touched on text effects. Bone up, and get your GD smarts squared away. You may begin to believe that you're an artist, in addition to a developer.

Downloads

Description	Name	Size	Download method
Source code	os-php-graphic-PHPGraphics_code.zip	84KB	HTTP

[Information about download methods](#)

Resources

Learn

- To find the graphics functions, check out the [Documentation](#) area at PHP.net.
- [PHPGD.com](#) is a great Web site dedicated to helping you learn the GD library.
- Visit the [GD library](#) for more information.
- Visit the [World Wide Web Consortium \(W3C\)](#), where many Web standards are created.
- [W3 Schools](#) is a source for many technologies, including PHP and other Web concerns.
- [Tek-Tips.com](#) is a peer-to-peer forum board. Ask a question, get an answer.
- [Read comments](#) about the `imageconvolution` function to get going on using it.
- developerWorks has covered creating graphics with PHP before. To learn different or earlier techniques in PHP graphics, see the following articles and tutorials:
 - "[Create graphics the smart way with PHP](#)" —This article shows how to build an object-oriented graphics layer in PHP.
 - "[Paint 3-D images with PHP](#)" — Find how to use the Image_3D package, learn the limitations of using dynamic 3-D images, and investigate solutions and practical applications of 3-D graphics.
 - "[Generate dynamic bitmap graphics with PHP and gd](#)" — Learn how to generate dynamic bitmap images using PHP and the GD library. (Note that this tutorial was published in 2003, and the tools discussed therein may be superseded.)
- [PHP.net](#) is the central resource for PHP developers.
- Check out the "[Recommended PHP reading list](#)."
- Browse all the [PHP content](#) on developerWorks.
- Expand your PHP skills by checking out IBM developerWorks' [PHP project resources](#).
- To listen to interesting interviews and discussions for software developers, check out [developerWorks podcasts](#).
- Using a database with PHP? Check out the [Zend Core for IBM](#), a seamless, out-of-the-box, easy-to-install PHP development and production environment

that supports IBM DB2 V9.

- Stay current with developerWorks' [Technical events and webcasts](#).
- Check out upcoming conferences, trade shows, webcasts, and other [Events](#) around the world that are of interest to IBM open source developers.
- Visit the developerWorks [Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Watch and learn about IBM and open source technologies and product functions with the no-cost [developerWorks On demand demos](#).

Get products and technologies

- Download the [GD library](#).
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.
- Download [IBM product evaluation versions](#), and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.
- Participate in the developerWorks [PHP Forum: Developing PHP applications with IBM Information Management products \(DB2, IDS\)](#).

About the author

Ken Blattman



Ken Blattman has been tickling computer keyboards for nearly two decades. Along the way, he has conquered several programming languages and written a half-dozen computer books. And there is always tomorrow to do more.