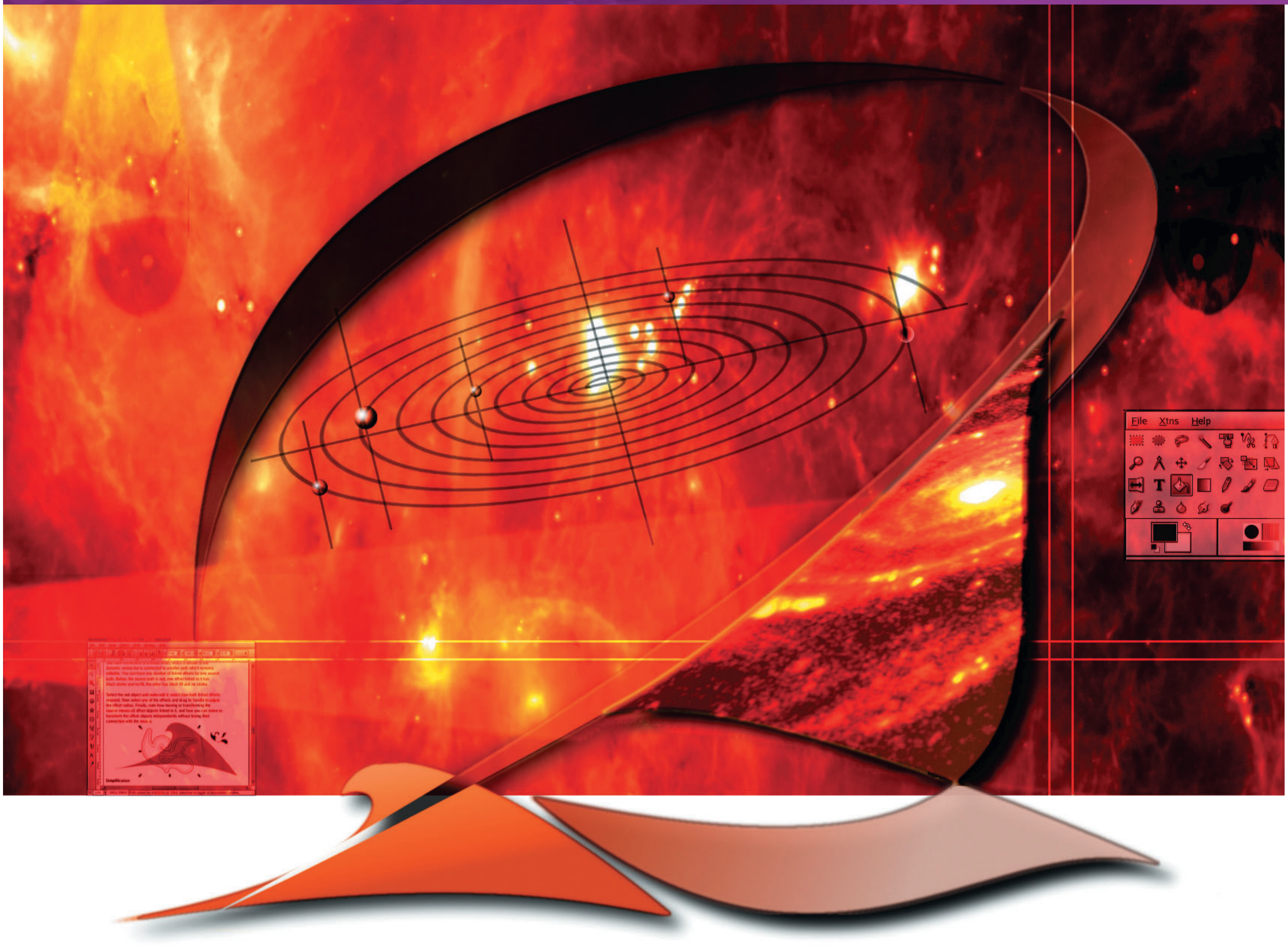


DESIGN AN IMAGE GALLERY

Gallery design using GIMP, PHP and CSS

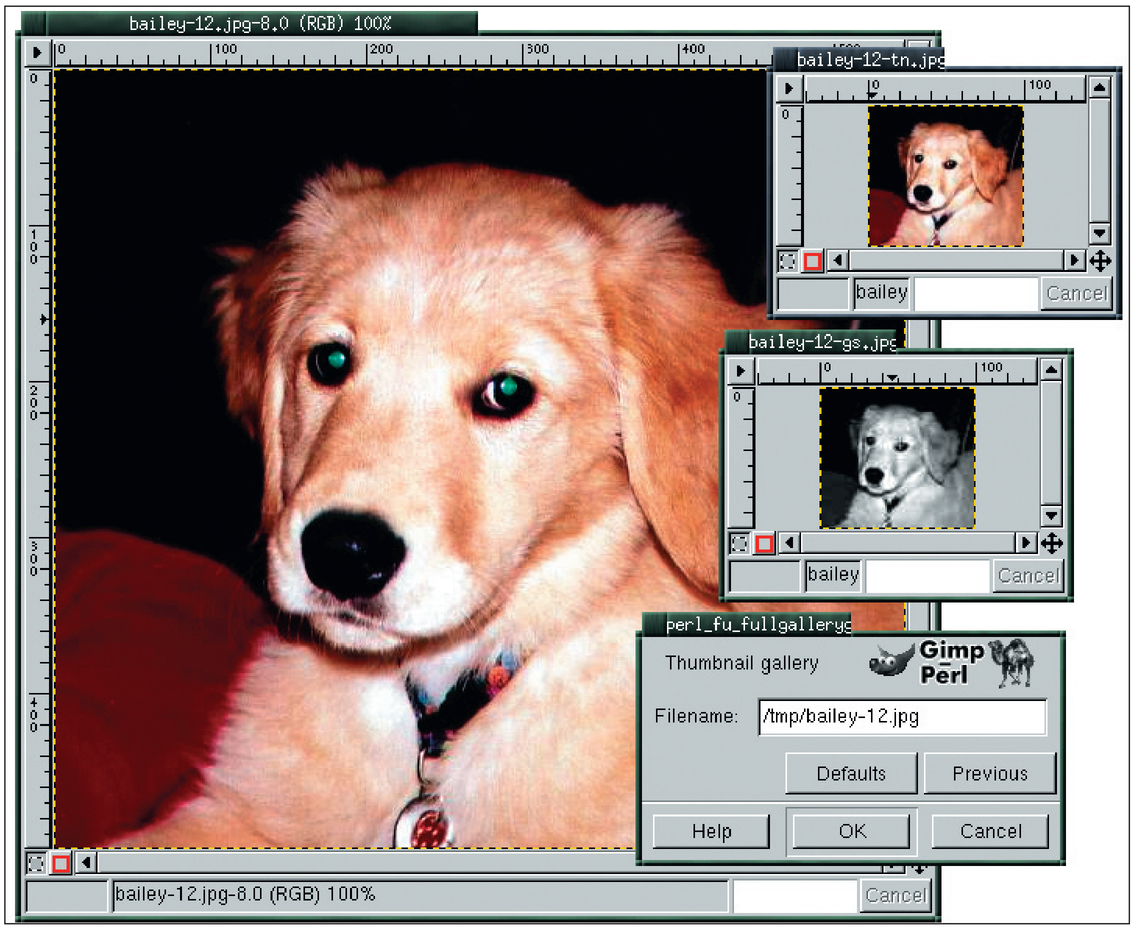
Michael J Hammel guides you through a complete task flow – from design to product – to see how to choose, mix and integrate tools to get the job done.

Reading over some of the reader mail for the past few issues I realised that this column is best received when it stays in tutorial format. Being as much a developer as I am an artist, I still like to talk about coding, a subject that is a bit more dry and, let's be honest, not really suited to the 'step 1: open a new file, step 2: select whizbang feature' style of commentary.

Fortunately for me (and for you, I hope), this conflict of interests has an obvious resolution: web design. Nowhere can an artist more easily expose their talent while at the same time be challenged by the intricacies of page layout mixed with interactivity. This mix is idea – all we need is a project.

Some time back, I wrote a book on creating web graphics with GIMP called *GIMP for Web Professionals*. In that text I showed how to write a GIMP Perl script to generate thumbnail images from larger originals. One reader took it upon himself to modify this script to generate an entire series of thumbnails from a directory of images. This idea can be expanded further to create a system for automatically generating gallery pages on a website. We have our project idea, but we also need to go in a little deeper.

One place software developers and graphic artists often run



Thumbnail gallery
Filename: /tmp/bailey-12.jpg

An original image and its much smaller cousins generated with our modified GalleryGFX script.

into walls is with the idea that software is written or layout begun immediately after the project is defined. But there are many issues that you need to be clear on before even one line of code or image is created. We have to think through what our end product should be and how it should work. We have to design.

The design process

Our project definition – a gallery of images – is too vague of a description. What kind of images? How should they be presented? The images could be artwork to display, products for sale or even members of a club. We need to identify what our visitors will want to do with those images. Define this right away or else your implementation efforts later may take you completely in the wrong direction.

We've decided (after too much coffee and a coin toss) that our visitors are coming to see images simply for review purposes. That means we want to show thumbnail collections where each thumbnail is a link to a larger, clearer version of the image. This differs from product images because we don't need to tie in pricing, sizing or purchasing features to our web pages. It might differ from member portfolios in that our images need not be tied to audio clips or personal data. In other words, we aren't going to need to link our pages to any databases. This simplifies our design greatly and leaves us free to focus on the creative aspects.

Despite being simple by nature, we still have requirements that need to be met:

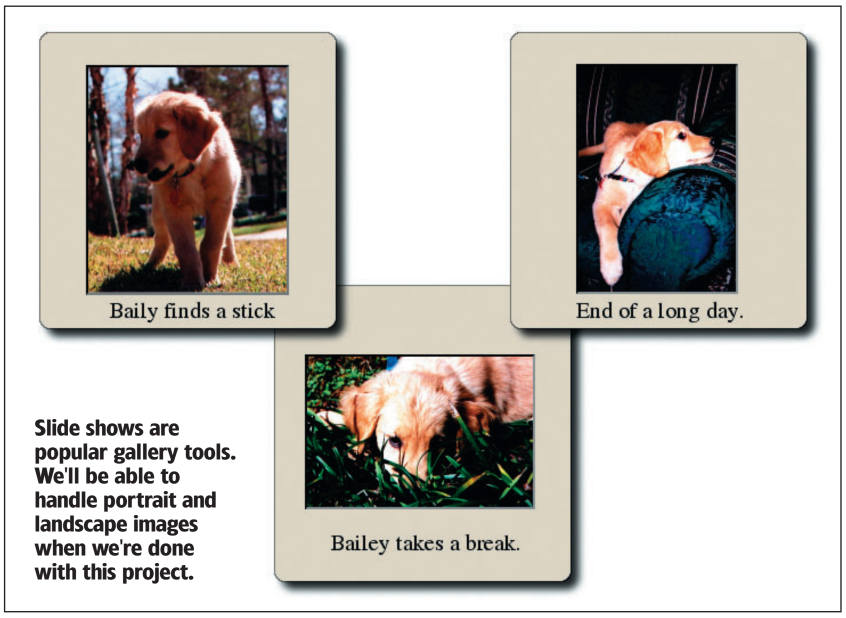
1. **We must be able to handle any number of images in a single directory.**
2. **The images must be sorted alphabetically by filename when displayed.**
3. **We must handle landscape, portrait or square images, and no thumbnail should ever be taller or wider than 100 pixels.**
4. **Each thumbnail must reference a page with a full size original of the image.**
5. **We must provide a 3x3 layout.**
6. **The interface must provide a 'film slide' presentation with image names listed on each slide frame.**
7. **For interactivity, we want hover images in colour, while default images are displayed in grayscale.**

The first four requirements are extensions to the original script that let us automate the display of the images. The extension actually comes in two parts: part one will be changes to the GIMP Perl script which we'll discuss later in this article, and part two will be new code written in PHP that handles page display. The PHP issues will be covered in next month's tutorial.

The last three are arbitrary requirements to add some style to our pages. We don't have anything in our project definition that says these things are required, but we added them with the right side of the brain: our artsy side.

This set of requirements has dependencies. For example, the ability to produce a slide frame means either clever CSS/DHTML or the use of a special background image (we'll opt for a little of both). Hover images definitely depend on CSS design. An implied dependency is a naming scheme for associating a thumbnail with its full size original and its greyscale cousin. The size of the slide frame is also dependent on the size requirement of the thumbnails.

As you can see there is a lot of design required for this simple project. A well-known saying for software developers is that coding is easy but design is hard. You will spend more time »



Slide shows are popular gallery tools. We'll be able to handle portrait and landscape images when we're done with this project.

« thinking through what you want to accomplish than actually implementing it. For our project, the detail we currently have in place should suffice.

Generating thumbnails

While our web server setup could allow us to resize images on the fly, image processing is CPU-intensive work and we don't want our site to appear to be slow to visitors. So we'll process all our images first, upload them to the server, then manage those images using PHP via the web server.

We start with a set of images. What format should they be in? Ah! Our design didn't cover that – see how thinking things through first can be useful? We'll limit our images to JPEG for now, though it wouldn't be considerably more difficult to add other file formats.

The original *GalleryGFX* script had lots of unneeded features. **Listing 1** shows the stripped down version that will process a single image into a colour thumbnail that is no more than 100 pixels wide. We first want to modify this to meet our design requirements for a single image, then to extend it to process the entire directory.

```
<Listing 1>
#!/usr/bin/perl
use Gimp qw(:auto);
use Gimp::Fu;

sub GalleryGFX_Run {
        my($img, $drawable, $filename) = @_;
        my $th_width = 100;

        my $img_width = $img->gimp_image_width();
        my $img_height = $img->gimp_image_height();
        my $height = int(($th_width/$img_width)*$img_
height);
        my $width = int($th_width);
        gimp_image_scale($img,$width,$height);

        $drawable = gimp_image_get_active_layer($img);
        gimp_image_flatten($img);
        gimp_displays_flush();
        $drawable = gimp_image_get_active_layer($img);
        file_jpeg_save($drawable,
                $filename, $filename, 0.30, 0, 1, 0, "", 0,
1, 0, 0);

        return();
}

register (
        "gallerygfx",
        "Thumbnail gallery", "Thumbnail gallery",
        "Michael J. Hammel", "GPL", "V1.0",
        "<Image>/Filters/GFXMuse/GalleryGFX",
        "RGB*",
        [ [ PF_STRING, "filename",
                "Full pathname of file", ""
        ] ],
        \&GalleryGFX_Run
);
exit main();
```

The code for the original *GalleryGFX* is included in the *Graphics Muse Tools* as is the version created here, which has been given the name *FullGalleryGFX*. The *Graphics Muse Tools* are available from http://ximba.org.

This first version assumes we want 100-pixel wide images, no matter what the original images' dimensions might be. Requirement #3 of our list says our thumbnails are either to be 100 pixels wide or 100 pixels tall, but no more than 100 pixels in either direction. So our first change is to test which dimension will be scaled to 100 pixels and which needs to be scaled to some other amount. **Listing 2** shows this change, with the lines before and after so you can see where it would fit in the original.

```
<Listing 2>
        my $img_height = $img->gimp_image_height();
        my $width, $height;
        if ($img_height == $img_width) {
                $width = 100;
                $height = 100;
        }
        elsif ( $img_height > $img_width ) {
                $height = 100;
                $width = int((100/$img_height)*$img_
width);
        }
        else {
                $width = 100;
                $height = int((100/$img_width)*$img_
height);
        }
        gimp_image_scale($img,$width,$height);
```

Scaling an image to smaller dimensions will blur it a bit so we'll add a little sharpening. This isn't one of our requirements but something we learn from experience working with scaling of images. There is no way to know exactly how much sharpen to apply but at 100 pixels it's a good bet that a setting between 10 and 20 per cent is plenty.

```
//code//
<Listing 3>
        gimp_image_flatten($img);
        plug_in_sharpen($img, $drawable, 10);
        sleep(2);
        gimp_displays_flush();
```

The **sleep()** call is required because calling external plug-ins (ie features not internal to GIMP itself) causes a separate process to be run and we need to wait for it to finish before moving on. Unfortunately, there is no easy way to know when it's done. In this case it's easy to guess that, for a 100x100 pixel image, two seconds is an eternity to the sharpen plug-in. If you want other processing of the image done by plug-ins (including those provided by *Script FU*) before saving you'll need to experiment to see if **sleep()** calls are necessary.

The image is scaled to its thumbnail size and it's still a colour image. We also need a greyscale image. We'll save this version then desaturate the image and save it to a new file. **Listing 4** shows how to do this. Desaturating removes the visible colour content but keeps the image as an RGB image. This is necessary so we can continue to save the images in the JPEG format. A true grayscale image is physically different from a desaturated RGB image, but visually they look the same.

```
<Listing 4>
        $drawable = gimp_image_get_active_layer($img);
        $filename =~ s/\.jpg/-tn\.jpg/;
        file_jpeg_save($drawable,
                $filename, $filename, 0.75, 0, 1, 0, "", 0,
1, 0, 0);
        gimp_desaturate($drawable);
        $filename =~ s/-tn\./-gs\./;
        file_jpeg_save($drawable,
                $filename, $filename, 0.75, 0, 1, 0, "", 0,
1, 0, 0);
```

The filename munging is a clever Perl trick. We take the input file name (which we assume is the name of the original image) and tack on **-tn** for the thumbnail and **-gs** for the grayscale version between the filename and its format extension. This naming scheme is part of requirement's number 2,4,6, and 7 of our original design. We could also add the filename in as a comment as well (in between the empty double quotes) but we'll leave that as an exercise.

So that's it for a single image. You can see the results in the screenshot on page 89. Now let's look at extending this to process a complete directory.

Thumbnail directories

The stripped down original version only prompted us for a file name. The script also assumes that it is being run against an open image. We're going to drop both of those features. We need the directory name instead. And we want to process that directory whether we started from an open image or not. In fact, we want to be able to run in GIMP batch mode so that we can run from the command line, making processing a directory of images very quick indeed.

The first change is to switch to using a directory instead of a filename. We'll prompt for a directory and save the value passed to us in a different variable name. We also change the menu location. This causes the script to be called with fewer arguments, as we'll see in the next section.

```
<Listing 5>
sub GalleryGFX_Run {
        my $dirname = shift;
        ...

register (
        "gallerygfx",
        "Thumbnail gallery", "Thumbnail gallery",
        "Michael J. Hammel", "GPL", "V1.0",
        "<Toolbox>/Xtns/GFXMuse/GalleryGFX",
        "RGB*",
        [ [ PF_STRING, "directory",
                "Full pathname of directory to process",
""
        ] ],
        \&GalleryGFX_Run
```

We then need to open the directory and process all files in it. It's important that we check each filename to be certain the file is of a type we can handle (based on its extension, which is far from optimal but good enough for this article). Notice in **listing 6** that nearly all the code from the original is now enclosed in a **while()** loop, plus we added some code to test the file type and open the file. We've also reduced the number of input parameters to just the single directory name. We can do this because we're going to run this from the command line in batch mode.

```
<Listing 6>
sub GalleryGFX_Run {
        my $dirname  = shift;
        my $th_width = 100;

        opendir(DIR, "$dirname") || die "Can't open
$dirname\n";
        while ($file = readdir(DIR))
        {
                ($file =~ /\.jpg/) || next;
                my $filename = $dirname . "/" . $file;
                my $img = file_jpeg_load($filename,
$filename);
                ...

        }
        closedir(DIR);
        return();
```

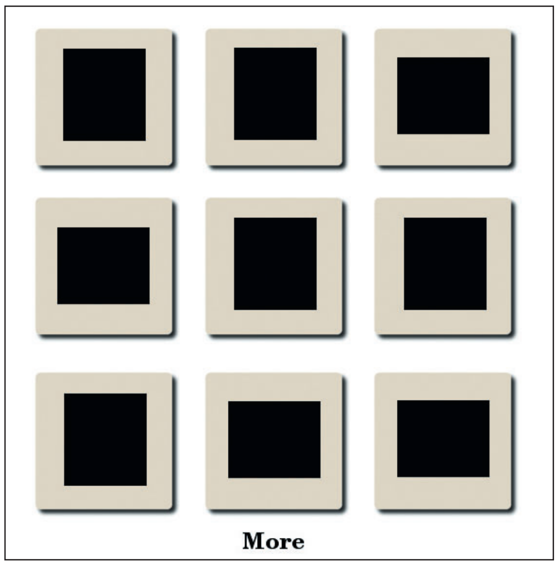
Once an image has been opened, the processing remains the same – scale and sharpen. We'll also keep the desaturation processing. We don't even need to change the way we save the file because we've already tacked on the full directory path. So that's it. Except for one thing: how do we run this from the command line?

GIMP batch mode is easy with GIMP Perl. You simply run this script manually, passing in two arguments, like so:

```
./GalleryGFX.pl -o /tmp/file.jpg -directory <filespec>
```

The **-o** option tells GIMP Perl to run without a user interface. GIMP Perl will find and run GIMP without opening the toolbox window. The argument to **-o** is meaningless for this operation, although when the script finishes it will save an image to that file. This argument is required even though we aren't using the file specified for our script.

The **-directory** option comes from our script via the **register()** function. Just pass in the directory name you want *GalleryGFX.pl* to process. It's interesting to note that if you use **-help** as the only argument to the script in batch mode it will tell you the arguments required by the script! GIMP Perl is quite smart about that, as it can get that information from the **register()** function. LXF



We want 3 rows and 3 columns, centered on the display with a link that allows visitors to see the next set of slides.

NEXT MONTH

This part of our project was easy. Our next part isn't much harder, but there is more coding to come. If you are trying to be artistic with your designs, you can't help but learn some coding. Especially, as we'll learn next month, with Cascading Style Sheets or CSS. We'll also put together a quick background image that will act as our slide border.