ImageMagick v6 Examples -- Creating Thumbnails

Index

One of the biggest uses ImageMagick is put to is the creation of thumbnails for family photo albums, sports and hobby pages, catalogs, and so on. Typically for use on the world wide web or in photo CDs.

This page provides examples and techniques used to generate thumbnails.

Thumbnail Storage

I would like to first start with one very important point.

The original image from video cameras and photo scanning should be kept in a safe place in the original format, preferably a non-lossy format (not the JPEG image format), without any modification resizing or other change, except possibly a file name change. Of course a scanned image can be re-scanned, but it is simpler to re-use your original scanning than to re-do it again later.

This is VERY important as any form of modification will mean a loss of some information in the image, and provides a source from which you can re-work your image for other uses. The original image does not have to be your working image, which may be resized or color adjusted for display, just be sure to have your image saved and backed up somewhere safe for future use.

The next thing to do, even before you create any thumbnails is to decide how you want to save your thumbnail, relative to your normal sized image format, then stick to that scheme. This is especially important for web pages.

Schemes include...

- Save the main photo image in the lossy JPEG format at the size you want or need, then use the same name for the generated thumbnail but using the GIF image format. EG Same filename but a different format and suffix.

  Main Image: `photo_name.jpg` Thumbnail: `photo_name.gif`

- Use the same format as the original image, but with an extra string added to the file name. Typical string additions include "`_tn`", "`_small`", "`_thumb`", etc...

  Main Image: `photo_name.jpg` Thumbnail: `photo_name_tn.jpg`

- Store the thumbnails with the same name in a sub-directory called for example "`thumbs`" or whatever is convenient for you.

  Main Image: `photo_name.jpg` Thumbnail: `thumbs/photo_name.jpg`

- Some combination of the above.

  There is no reason why you can not save thumbnails in different image format, with a extra image suffix appended to the filename, and saved in a subdirectory!

Main Image: `images/photo_name.jpg` Thumbnail: `thumbs/photo_name.jpg.gif`

This is actually quite common on the WWW, and I have even seen the the two directories stored on completely separate machines!

The first scheme can use "`mogrify`" to generate all your thumbnails, without destroying the original image. The other will need you to either, first make a copy of the original image, before running "`mogrify`", create a special script to process the images, or some other method.

A number of the simpler non-IM methods are detailed at the end of the example section for Batch Processing - Without using "`mogrify`".

Whatever method you choose the important thing is to choose a scheme for thumbnail storage, and then stick with it.

By using the same scheme for all your thumbnails you can then write shell or Perl scripts to make thumbnail generation and even generation of the HTML links easy. More on this later.

Selection of the Thumbnail format

The format in which you save a thumbnail can make a big difference to its final size on disk and download speed for web pages. In this regard I recommend you study the summary of the various Common File Formats.

Specifically you should note...

JPEG compresses well and is lossy, but is designed for large real world images, not small thumbnails. It also does NOT allow any form of transparency. In summary, the format is good for large images, bad for thumbnails. Watch out for profiles (see next section).

While JPG is not recommended for thumbnails, for full window web viewing and downloading it is recommended you use smaller 800x600 pixel image, at a much lower "`-quality`" percentage (say 50 or even 30%).

If you wish to allow the user to download the full high quality image (say for digital printing), add a separate link to that file as it will probably take a very long time to download.

GIF works for simple small images, compresses okay though not the best. It has a color limit of 256, but for small images this is rarely noticeable. It can also do cartoon like animations of images, not that that is needed for thumbnails, unless you really what to get fancy.

What is a problem is that the format only has boolean (on/off) transparency, which make for horrible looking borders on shaped images. The solutions to that is to design the thumbnail to only use boolean transparency, or arrange it so it can only be used on a specific background color. For details see the examples on GIF's on a background color or pattern.

PNG is the modern ideal format for thumbnails. It has a good compression, and internal format styles. It is non-lossy, and can display all colors, and these days it is understood by almost all browsers (though for Microsoft Internet Explorer, before v7, needs some java scripting added to web pages).

More importunately this format understands semi-transparent color, making shadows and edges sharp and clear, or faded and blurry as you wish. This format however does not do animations, though the related MNG format does. Very few browsers seem to support that format however.

For thumbnails you can reduce the size of the final image by reducing the depth and number of colors, as well as setting a higher "`bzip`" compression quality (first digit in -quality) for your final thumbnail image.

For example, the following is suggested for PNG thumbnails that does not involve transparency...

```
-strip  -depth 8  -colors 256  -quality 95
```

You can also re-process the final image though a secondary application called "`pngcrush`" which finds the best PNG compression for that specific image. This application will not reduce the images color so may not produce as small a result as the above with color reduction.

One final word about formats... No matter what format you use for your thumbnails, if you must save an intermediate unfinished image, use a PNG (without any color reduction) or MIFF image format. Doing this will preserve as much color information about the image in the intermediate stage. Only do color reduction, or save to GIF or JPEG formats as a absolute final step.

This is important, so I repeat...

Do NOT use JPEG or GIF for intermediate working images!

A quick word about profiles..

Many images from digital cameras, scanning software, and some paint programs (photoshop is notorious for this), save extra information about the image in the form of profiles. This includes image formats such a JPEG, PNG and TIFF and even the IM internal format, MIFF.

These profiles can be up to 60 Kb in size, so can make a big difference to your file size, and by default IM will preserve this profile information. Thumbnails have no need for this data, and often not even the main image needs it.

You can also remove the profiles from say all your JPEG images with the IM commands...

```
convert input.jpg  +profile '*' output.jpg

mogrify +profile '*'  *.jpg
```

With IM version 6 you can also use the option "-strip" to remove the profiles.

```
convert input.jpg -strip output.jpg

mogrify -strip  *.jpg
```

Unfortunately as JPEG is a 'lossy' format the above methods are actually a *bad* idea to do to your original image, as the image quality will be slightly degraded. Better ways of handling JPEG images without loss of image quality can be found in Non-ImageMagick JPEG Processing.

On the other hand if you are resizing the image, than you must reading and writing the image anyway, and in this case using IM to "-strip" the profiles at the same time is a *good* idea.

Stripping profiles while resizing, particularly for generating smaller thumbnail images, is so common that both "-resize" and "-strip" were combined into a new operation, just for this very purpose. Naturally enough this resize operation is called "-thumbnail".

For example...

```
convert -size 120x90 input.jpg -thumbnail 120x90 output_thumbnail.gif

mogrify -format gif -size 120x90 -thumbnail 120x90  *.jpg
```

Also see a number of the simpler non-IM methods are detailed at the end of the example section for Batch Processing - Without using "mogrify".

For very large images the "-thumbnail" resize operator goes further and first scales the image down to 5 times the final thumbnail size first, before doing the actual resize operation. This speeds up the thumbnail generation further.

However for thumbnailing JPEG images, a even better method of limiting the initial image size can be used, by just not reading in all of the image into memory in the first place.

The "-size" setting (as shown in the above example) is a special hint to the JPEG image library to reduce the amount data that is read in from

VERY BIG JPEG images. See JPEG File options. Caution is however needed if you are also using this setting for other purposes, such as canvas creation.

From IM version 6.2.6-2, a new image input option was added, which lets you resize the input image immediately after it is read in. This option will work with ANY image format, not just JPEG image. It is however no substitute for using a "`-size`" with JPEG images.

As such the recommended way of resizing ANY input image format is now...

```
convert -size 240x180 input.img'[120x90]' -strip  output_thumbnail.gif
```

Well on with the practical IM thumbnail examples...

General Thumbnail Creation

Generate Thumbnails to a specific height

Lets convert a large sample JPEG image to a GIF thumbnail 90 pixels high, with the width automatically adjusted (within the 250 pixel width limit) preserve the aspect ratio of the image.

```
convert -size 500x180  hatching.jpg  -thumbnail 250x90 thumbnail.gif
```

Note that I used the "`-thumbnail`" option above. This not only resizes the image, but strips any and all profile and comment information that may be present in the original JPEG image.

I also set a minimum "`-size`" for the image being read in. This is passed to the JPEG library, which will return an image somewhere between this size and double this size (if possible), rather that the whole very large original image. Basically don't overflow the computers memory with an huge image when it isn't needed.

The "`-size`" value I use is double that of the final thumbnail so that resize will still generate an reasonable result.

The result is a thumbnail of a specific height, but variable width. I use this thumbnail for my own web pages so that a series of image in a

row, will all match up height wise, forming a neat look.

The 250 pixel width limit in the above is important. If left unset, I would give IM complete width freedom (EG: using the option "`-thumbnail x90`"). This could result in problems when generating thumbnails of long thin images such as Web Line Images. The result in that case would be very very long, *enlargement* of the image, instead of the desired thumbnail.

The "`mogrify`" version is the same as the "`convert`" command (with no initial input images), but will but will generate automatic thumbnails of *every* JPEG image in the current directory. The image argument is quoted so that IM itself will scan the directory, and not the command line shell. This prevents 'line limit overflow errors' on directories containing a huge number of images.

```
mogrify -format gif -size 500x180  -thumbnail 250x90  '*.jpg'
```

Note that "`mogrify`" will blindly create thumbnails, replacing any existing GIF images of the same name. Extreme caution is advised. Especially if you have previously hand created thumbnails (see Manual Thumbnail Creation below).

Also if the "`-format gif`" option is not given, the original image will be overwritten, which unless a backup has been made could mean complete disaster.

If you are creating thumbnails with a different name or in a sub-directory, you will have to make a copy the original image to that name then use "mogrify" without any "`-format`" option. Or write a shell script to go though the images one at a time using "`convert`".

Resize to fit a rectangular box

Another form of automatic thumbnail generation is shrink image to fit a fixed sized box, say "`100x100`" but keeping the images aspect ratio. Well that is the default meaning for a resize geometry setting.

However I prefer not to enlarge images which already fit such a box. For that you need to add a "`>`" to the geometry string.

```
convert -size 200x200 hatching.jpg  -thumbnail '100x100>' rectangle.gif
```

As before the aspect ratio of the image is preserved, as such the thumbnail is unlikely to be exact 100 pixels square. However at least one of the images dimensions will be 100 pixels.

Pad Out the Image

The next most common request is to generate thumbnails that fill out the image with borders of a specific color (usually '`black`', or '`transparent`' but for these examples I will use '`skyblue`') so the thumbnail is exact the size you wanted.

For example: An which is 400x300 pixels shrunk to fit a 100x100 pixel box will normal (with the above) have a size of 100x75 pixels. We want to add some padding borders to the top and bottom of the image (and to the sides to be sure) to make the final thumbnail image always 100x100 pixels in size.

There are a number of ways to do this.. one is to add a border and crop...

```
convert -size 200x200 hatching.jpg -thumbnail '100x100>' \
        -bordercolor skyblue  -border 50 \
        -gravity center  -crop 100x100+0+0 +repage pad_crop.gif
```

While the "`-crop`" image operator will cut out part of an image, it will not adjust the images page size or offset position within that 'page'. Not only that the GIF file format will preserve this information, and many web browsers will make use of it. The result is that displaying a GIF image that had "`-crop`" applied, may display in a larger space, with an offset.

The "`+repage`" operator removed this, and as such is very important after any "`-crop`" operation.

For ImageMagick version 5 and earlier you would has used "`-page +0+0`" instead, but that is only a read/create setting in version 6 and later.

As of IM version 6.2.4-5, a new crop method make this image padding, a lot easier. Viewport Cropping adjusts the page/canvas settings of the resulting image to the actual 'viewport' of the area cropped. As such all that is needed is to fill out the extra space on the images virtual canvas.

```
convert -size 200x200 hatching.jpg -thumbnail '100x100>' \
        -gravity center  -crop 120x120+0+0\! \
        -background skyblue  -flatten    pad_view.gif
```

Another method to pad out an image is to overlay the thumbnail onto a background image (actual image, solid color or tile) that is the right size, in this case the 128x128 "`granite:`" built-in image.

```
convert -size 200x200 hatching.jpg -thumbnail '100x100>' \
        granite: +swap -gravity center -composite pad_compose.gif
```

This method is probably the best method to use with older versions of IM (such as IM v5), though the "`-composite`" operation will need to be done by the separate "`composite`" command in that case.

Cut the Image to Fit

An alternative, is rather than pad out the image to fit the specific thumbnail size we want, is to instead cut off the parts of the image that does not fit the final size.

Of course this means you actually lose some parts of the original image, particularly the edges of the image, but the result is a enlarged thumbnail of the center part of the image. This is usually (but not always) the main subject of the image, so it is a practical method of thumbnail creation.

However the resize commands of IM is not specifically designed with this method in mind, so to do this we need to employ some very special tricks. This trickiness is fully detailed in Resizing to Fill a Given Space.

For example, the following, will resize our image to fit our 100 by 100 pixel area, but then junk the edges that did not fit into this area.

```
convert -size 300x300 hatching.jpg  \
        -thumbnail x200   -resize '200x<'   -resize 50% \
        -gravity center -crop 100x100+0+0  +repage   cut_to_fit.gif
```

As you can see the thumbnail of the image is much larger and more detailed, but at a cost of cutting of the sides off the original image.

The biggest problem with this method is that you are now resizing your image 2 to 3 times, and it only works well for images that are 3 to 4 times the final size. That means of course it is much slower, and can add more resize distortions to the final thumbnail (though they are very hard to see).

The only solution to this is figure out the correct resize parameters yourself so you only resize the image once (preferably in a API such as PerlMagick). For thumbnails this is not so important, as the shrinking the image removes any multi-resize distortions.

Manual Cropping

The normal way I generate thumbnail images for use on my web pages, is a mix of automatic and manual scripts. the final setup of my images are..

- I use a PNG or TIFF for the original ,VERY large, scanning of the photo.
- The JPEG image format for a full-size image linked to by the thumbnail. This image is resized to fit a 800x800 pixel box suitable for viewing by most web users.
- And lastly a thumbnail resized to a fixed 90 pixel high, and variable width. This allows centered rows of thumbnails on web pages to look reasonable neat and tidy, but which automatically fills the browser windows width, no matter what size browser they are using.

I first generate the full-size web JPEG image using "`morgify`" from the original scanned image. This reduces the download and viewing size of the image to something that is practical for the typical web user (who is often logged in via modem).

From these images I generate thumbnails, again using "`morgify`". However I often find in typical photos that the subject of the thumbnails becomes too small to make an effective thumbnail, when viewed.

To fix this I examine the automatically generated thumbnails, and in about half the cases manually create my own 'zoomed in on subject' thumbnail.

I read in the JPEG image, and crop it down the main subject of the image effectively 'zooming in' on the subject of the photo, and removing bulk of the background context of the image. This is then smoothed and thumbnailed, either using a "`convert -thumbnail`", or more often in the same graphic program (usally "`XV`", see below), I used to manually crop the image.

So instead of a thumbnail where the people in the photo are hardly visible (left), I have manually zoomed in on the subject, highlighting the main point of the photo (right). That allows users to see the image content more clearly and thus better decide if they actually want to download and look at the larger JPEG version of the image.

Queensland KiteFlyers, Ron and Val Field

Automatically Generated Thumbnail

Manually Cropped and Resized Thumbnail

(Click on image for full sized photo)

This is of course more manually intensive, but only needs to be done once per image, and only on images that have a lot of space such as in the above example.

Of course as mogrify will overwrite any existing, possibly hand generated thumbnails, you can NOT use it again after you perform any manual thumbnail generation. Mogrify is useful, but also very dangerous as it overwrites lots of images. Always think before you run "`mogrify`" globally across all your images.

ASIDE: HTML page generation

Once I have all the thumbnail images sorted out in the directory I use a special perl script called "`thumblinks`" I wrote that look for the images (JPEG photos and GIF thumbnails), and generate HTML links, and even full HTML photo pages.

The script will read and include size of the GIF thumbnail size in the HTML, and attach preprepared header and footer files around the thumbnail links. The script will also remove any thumbnail links from the list it generates, if it finds an existing link in the header or footer file itself.

This may sound complex, but it makes my HTML page generation very fast and flexible, and ensures ALL image thumbnailed images in a directory have been added to that directories index page, while still letting me comment on specific images in the index header. It also makes the page independant of the users window size, automatically adjusting to suit.

For a simple example of my "`thumblinks`" script output see Tomb of Castle Artworks.

Other Non-IM Techniques

The "`XV`" program I use for manual image processing also generates thumbnail images, in a sub-directory called "`.xvpics`". The format of the images in this directory is its own thumbnail format (ignore the filename suffix). These thumbnails are limited to 80x60 pixels so are a little on the "small" size (unless you hack "`xv`" to use larger thumbnails -- see link below).

IM understands the "`xv`" thumbnail format (which is based on the "`NetPBM`" image format), so you can generate thumbnails using XV, then convert the XV thumbnails of the JPEG images, into GIF thumbnail images...

```
xv -vsmap &               # generate thumbs with the "Update" button
rm .xvpics/*.gif          # delete XV thumbs of existing "gif" thumbnails
mogrify -format gif .xvpics/*.jpg
mv .xvpics/*.gif .        # move the new "gif" thumbnails to original dir
```

If you are sick of the small size of XV thumbnails, particularly with larger modern displays, you can hack the XV code.. See my XV modification notes, which allows you to get XV to use a larger thumbnail size, 120x90 pixels in my own case.

Further Processing -- Adding Fluff

The above is only the beginning of what you can to to make your thumbnails more interesting. Beyond the basic thumbnail image you can add borders, rotations even with some random selection of style to make your thumbnail gallery that much more interesting.

Additions to thumbnails like this, is what I term 'fluff' (as in the extra lint you find covering your clothes after you wash your clothes). That is unnecessary extras added to the thumbnail, but which can make web pages and index images that much more interesting.

Be warned that many of the following methods and processing is very complex and my require a deeper knowledge of the various image processing options options of ImageMagick.

A Raised Button

The "`-raise`" operator was basically created with the one purpose of highlighting the edges of rectangular images to form araised button. It is a simple, fast, and effective thumbnail transformation.

```
convert thumbnail.gif  -raise 8x8   raised_button.gif
```

The same operator has a 'plus' form that can be used to make a sunken highlighting effect.

```
convert thumbnail.gif  +raise 8x8   sunken_button.gif
```

Adding a 3D frame

In a sumular way the "`-frame`" operator make it easy to add a frame around the image

```
convert thumbnail.gif   -mattecolor peru  -frame 9x9+3+3  framed.gif
```

This operator also has a lot more options to create a dozen or so different styles of frames. You can see examples of the posibilities in Frame, adding a 3D-like border.

Using Montage

The montage command provides a much easier way of doing all the above, and much more. It can not only generate thumbnails (or whole pages of thumbnails), but it can label the thumbnails to include information like filenames, disk size, and dimensions, or a user specified string.

Here is a simple use of "`montage`" to generate a framed thumbnail.

```
montage -size 240x200  -label '%c'  hatching.jpg \
        -frame 6  -geometry '120x100>'  montage_simple.gif
```

The label comes from JPEG image file comment, which was added long ago to the image using the Non-IM command "`wrjpgcom`". See Non-IM JPEG Processing for more details.

Even with just "`montage`" you can get really fancy with your thumbnail generation.

```
montage -size 400x180  -label '%c'  hatching.jpg  -thumbnail '200x90>' \
        -geometry '130x100>'  -mattecolor peru  -frame 6 \
        -bordercolor skyblue  -font LokiCola  -pointsize 18 \
        montage_fancy.gif
```

See the "Montage, Arrays of Images" for more details.

You may be especially interesting in the Montage HTML Thumbnail Image Maps example. This creates a HTML index page of thumbnails in which clicking on the thumbnail will bring up the original image, in the same directory.

Adding image labels

During your thumbnail creation you can also add labels either above, below or even on top of your thumbnail.

This sort of image processing is however covered more throughly in Annotating Images with Labels. Just remember to use the "`-thumbnail`"

or "-strip" rather than a "-resize" in those examples.

Here is probably the best type of overlay of a compound font annotation (Specifically the Denser Soft Outline font) showing the complexity that can be achieved by this method.

```
convert -size 400x180  hatching.jpg  -resize '120x200>' \
    \( +clone -matte -fill transparent -draw 'color 0,0 reset' \
       -resize 1000x200\!  -font SheerBeauty -pointsize 72 -gravity Center \
       -strokewidth 8 -stroke black  -fill black  -annotate 0,0 '%c' \
       -channel RGBA -blur 0x8 \
       -strokewidth 1 -stroke white  -fill white  -annotate 0,0 '%c' \
       -fuzz 1% -trim +repage -resize 115x \
    \) -gravity North -composite    -strip  annotated.gif
```

Note how I do not use "-thumbnail" to strip the profiles and comments from the image, as I want to preserve the images 'comment' string for use by the "-annotate" operator. Also note how I use "+clone" to generate a working canvas, again to preserve the images 'comment' string. Only at the end do I "-strip" that information.

ASIDE: This example could make good use of a future development proposal of IM where various settings, attributes and image sizes can be extracted from other images.

Soft Edges

The "-vignette" operator provides a simple means to add a blurry edge around an image.

```
convert thumbnail.gif -matte   -background none -vignette 0x4  vignette.png
```

Of course as this thumbnail uses semi-transparent color so it needs to be saved in the PNG format.

If you don't really like a circular type edging you can use some trickiness involving Edge Effects and Virtual Pixels to blur a transparent color at the edge of the image.

```
convert  thumbnail.gif -matte -virtual-pixel transparent -channel A \
```

```
            -blur 0x8  -fx '(u-.51)*2.1'  soft_edge.png
```

If you also threshold the above (by saving to GIF format), you just round corners of the thumbnail image.

```
   convert  thumbnail.gif -matte -virtual-pixel transparent -channel A \
            -blur 0x8   rounded_corner_blur.gif
```

This however is not a good, or proper, way to round off the corners of the image. For one thing it is not antialiased, so you get aliasing (jaggies) effects on the rounded corners. Lets look at the better way of doing this.

Rounded Corners

`FUTURE: the proper anti-aliased way to ass rounded corners`

Border with Rounded Corner

The IM "`-draw`" operator comes with a '`roundrectange`' method that can be used to provide an interesting frame around the image. However you need to size the dimensions of this draw method to match the image. Luckly the latest version of Im provides methods to extract and calculate these parameters.

The draw commands are created using fancy FX escapes to calcualte the draw parameters needed directly from the image. This is saved as a Magick Vector Graphics File that can be directly used by draw in later commands.

```
   convert thumbnail.gif -border 2 \
           -format 'roundrectangle 1,1  %[fx:w-2],%[fx:h-2] \
                      %[fx:int((w+h)/15)],%[fx:int((w+h)/15)]' \
           info: > rounded_corner.mvg
```

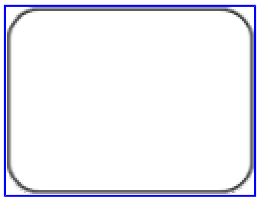
```
roundrectangle 1,1  122,92  15,15
```

Note that the image was enlarged slightly, before the calculations were done, to provide some extra space in the final thumbnail for the added border. That same amount of space is also needed in the next steps.

If you can figure out the image size in a different way you can substitute the appropriate draw command directly into the next examples, rather then use an FX mathematical expression. Basically the above makes this whole process independent of the actual size of the thumbnail. Any other way, including direct hard coding is also acceptable.

Now we can use this to generate overlay and a mask image. As part of this we also need to create a transparent canvas using the original image, to get the size right.

```
convert thumbnail.gif -border 2 -matte -channel RGBA -threshold -1 \
        -background none -fill none -stroke black -strokewidth 3 \
        -draw "@rounded_corner.mvg"    rounded_corner_overlay.png
convert thumbnail.gif -border 2 -matte -channel RGBA -threshold -1 \
        -background none -fill white -stroke black -strokewidth 1 \
        -draw "@rounded_corner.mvg"    rounded_corner_mask.png
```

These two images are very similar, but play very different roles. The first image is the rounded rectangle we want to overlay on top of the thumbnail. But before we overlay the new parts you need to erase or 'mask' the image to remove the parts you don't want. That is the job of the second image.

Note that I used a slightly smaller outline stoke width on the mask. This ensures we completely erase more of the image than is really needed so that the edges remain properly anti-aliased. This is important.

Now we can mask our thumbnail image to round off the corners, the overlay the new border.

```
convert thumbnail.gif -matte -bordercolor none -border 2  \
        rounded_corner_mask.png -compose DstIn -composite \
        rounded_corner_overlay.png -compose Over -composite \
        -depth 8 -quality 95 rounded_border.png
```

There is also an alternative method where you first overlay the new parts onto the image, then mask out the final shape. For that method the overlay image needs to extend beyond the mask.

Basically do not use the same anti-aliased transparent edging for both the overlay and the mask or you will have problems with either the double anti-aliasing of pixels, or some background image colors being added to the edge pixels. Neither result is particularly good.

Adding a Shadow

The "-shadow" operator makes the Generation of Shadows of any shaped image easy.

For example here a I add a semi-transparent colored shadow, to the rounded corner thumbnail.

```
convert rounded_border.png \
        \( +clone -background navy -shadow 60x0+4+4 \) +swap \
        -background none -mosaic  \
        -depth 8 -quality 95  shadow_hard.png
```

But you can just as easily create soft fuzzy shadows, too.

```
convert -page +4+4 rounded_border.png \
        \( +clone -background navy -shadow 60x4+4+4 \) +swap \
        -background none -mosaic \
        -depth 8  -quality 95  shadow_soft.png
```

Note that I used a PNG format image for the thumbnails output. that is because the shadowed image will contain a lot of semi-transparent pixels of the shadows background color.

If you do plan to use GIF or JPG format you will need to use a more appropriate "-background" color for the web page or larger canvas on which you plan to display your thumbnail, as these formats do not handle semi-transparent colors.

Adding Some Thickness

Adding a thickness to a image or a shape looks a little like a hard shadow (see above), but isn't quite the same, and needs some extra work to get right.

This is actually very tricky as we create a colored, mask of the image which is then replicated multiple times, and layered under the original

image (using '`DstOver`' composition), with increasing offsets to give the image thickness.

```
convert thumbnail.gif -matte \
        \( +clone -fx DarkSlateGrey -repage +0+1 \) \
        \( +clone -repage +1+2 \) \
        \( +clone -repage +1+3 \) \
        \( +clone -repage +2+4 \) \
        \( +clone -repage +2+5 \) \
        \( +clone -repage +3+6 \) \
        -background none -compose DstOver -mosaic  thickness.gif
```

You get the idea. Each '`\( +clone ... \)`' line adds one extra pixel to the image in a south by south-easterly direction.

Also as no semi-transparent pixels are involved (at least for a rectangular image) you can use the GIF image format for the result.

The major problem with this technique is that it is hard to specify a thickness as a variable argument, or at different angles, unless you write a specific script to add thickness. Also the edge of the angled parts of the thickness is not anti-aliased, so their is lots of room for improvement.

ASIDE: This may be a good operator to add to a future version of IM.

Polaroid-like Thumbnails

You can make your thumbnail image look like a polaroid photo, give it a shadow, and even rotate it a little so as to appear to be just sitting on a table.

```
convert thumbnail.gif \
        -bordercolor white  -border 6 \
        -bordercolor grey60 -border 1 \
        -background  none   -rotate 6 \
        -background  black  \( +clone -shadow 60x4+4+4 \) +swap \
        -background  none   -flatten \
        -depth 8  -quality 95   poloroid.png
```

With some extra cloning with random rotates, then stacking them up (centered) you can create a stack of photos...

```
convert thumbnail.gif \
        -bordercolor white  -border 6 \
        -bordercolor grey60 -border 1 \
        -bordercolor none  -background  none \
        \( -clone 0  -rotate `perl -e 'print rand() * 30 - 15'` \) \
        \( -clone 0  -rotate `perl -e 'print rand() * 30 - 15'` \) \
        \( -clone 0  -rotate `perl -e 'print rand() * 30 - 15'` \) \
        \( -clone 0  -rotate `perl -e 'print rand() * 30 - 15'` \) \
        -delete 0  -border 100x80  -gravity center \
        -crop 200x160+0+0  +repage  -flatten  -trim +repage \
        -background black \( +clone -shadow 60x4+4+4 \) +swap \
        -background none  -flatten \
        -depth 8  -quality 95   poloroid_stack.png
```

The "`perl ...`" argument generates a random floating point number from -15 to +15. Substitute specific numbers for the back-quoted expression if the above does not work for you.

Of course you could substitute a set of different images rather than repeating the same image when creating the stack. Or select a set of rotates angles so they are all reasonably different, or are more pleasing to look at. If you are really good you can even offset the rotated images (jitter their position a little) so they are not all stacked up perfectly centered. But you get the basic idea.

If you really want to avoid the use of the PNG format, due to its current problems with *some* browsers, you can use the GIF image format. To do this you must be willing to accept some color limitations, and know the exact background color on which the image will be displayed. The '`LightSteelBlue`' color in the case of these pages.

```
convert thumbnail.gif \
        -bordercolor white  -border 6 \
        -bordercolor grey60 -border 1 \
        -background  none   -rotate -9 \
        -background  black  \( +clone -shadow 60x4+4+4 \) +swap \
        -background  LightSteelBlue  -flatten    poloroid.gif
```

For details about this technique (and more) see GIF images on a solid color background.

The above techniques graciously provided by Ally of Ally's Trip and Stefan Nagtegaal for Muziekvereniging Sempre Crescendo, both of which use Polaroid-like thumbnails extensively on their web sites.

The user '`grazzman`' from the IM User Forums went a little further by overlaying images onto a rotating canvas to create a photo spread.

```
convert -size 150x150 xc:none -background none -fill white -stroke grey60 \
        -draw "rectangle 0,0 130,100" thumbnail.gif \
              -geometry +5+5 -composite -rotate -10 \
        -draw "rectangle 0,0 130,100" thumbnail.gif \
              -geometry +5+5 -composite -rotate -10 \
        -draw "rectangle 0,0 130,100" thumbnail.gif \
              -geometry +5+5 -composite -rotate +10 \
        -trim +repage -background LightSteelBlue -flatten \
        poloroid_spread.gif
```

Of course for a photo spread like this you really need to use a set of different photos rather using the same image over and over as I did here.

There are a few cavats you may like to consider with this technique.

- The framing has been hardcorded into the above, and depends on the size of the thumbnail image. In a real application the framing may be moved to the thumbnail generation stage rather than in the above photo spread.
- As "`-rotate`" also expands the size of the canvas the position in which images are added is changing, unless you place them using a offset from "`-gravity center`" position.
- And finally, a constantaly rotating the background frame is not a good idea in terms of quailty. Rotating an already rotated image, adds more pixel level distortions to the result.

If you come up with a better version of the above, please contribute.

I would also like to add one final warning. Adding a semi-transparent shadow to the above is much more difficult as shadows from two seperate photos, do not interact correctly. That is two overlapping shadows become very dark, where in reality they do not add together in the same way that overlaying images do.

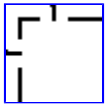
Basically the various parts of the final image will be either shadowed or not shadowed. You will not get two shadows, unless you have two separte light sources. If you are really serious about getting shadowing correct, you make like to consider layering the photos using a raytracer, so that all shadowing effects are correctly determined.

If you come up with a method of laying images, while still getting shadows correct, both on the background and lower image layers, then please let me and the rest of the IM community know. You will be worshipped as a great graphics designer by all (especially me) :-)

Fancy Borders

As a final example lets use some very fancy corner pieces. The original source for this corner (shown right) is available in the DIY section of Anthony's Icon Library. There are others in this section, so you may like to have a look. If you find something on the net, please let me know as I like to collect interesting corners, and edging techniques.

This corner image was manually re-colored and saved to produce the the two given images, an overlay and transparency mask. Notice that these corner images did not use any semi-transparent pixels, due to its rectangular nature. As such it is suitable for producing 'GIF' thumbnails, without any 'aliasing' edge effects.

To use the two images we first need to prepare our thumbnail by adding successive layers of borders. After that we overlay the corner overlay image, before masking the parts that needs to be made transparent with the transparency mask image. A total of 8 image composites...

```
convert thumbnail.gif   -matte  -compose Copy \
        -bordercolor Black -border 2  -bordercolor Sienna -border 3 \
        -bordercolor Black -border 1  -bordercolor none   -border 2 \
        -bordercolor Black -border 2  -bordercolor Peru   -border 3 \
        -bordercolor Black -border 1  -compose Over \
        \( fancy_add.gif             \) -gravity NorthWest -composite \
        \( fancy_add.gif -flip       \) -gravity SouthWest -composite \
        \( fancy_add.gif       -flop \) -gravity NorthEast -composite \
        \( fancy_add.gif -flip -flop \) -gravity SouthEast -composite \
        -compose DstOut \
        \( fancy_sub.gif             \) -gravity NorthWest -composite \
        \( fancy_sub.gif -flip       \) -gravity SouthWest -composite \
        \( fancy_sub.gif       -flop \) -gravity NorthEast -composite \
        \( fancy_sub.gif -flip -flop \) -gravity SouthEast -composite \
        fancy_border.gif
```

WARNING: to preserve the transparent border when other borders are being added latter, you must set "-compose" setting to 'Copy' rather than the default of 'Over'. If you don't than you will be left with black borders, instead of transparent borders in some parts of the final image.

The use of two images to overlay and then to mask out the transparent areas of the overlay, is very versatile. The mask can be used to add tears to the image, or even curl up the edge of a single corner.

```
FUTURE:

A example of a single corner rolled over, or curled up.  EG:
single corner modification, and moved introduction to 'masked composite
overlays'

Adding a tear into a photo on one edge.
```

Edge Tiled Borders

The next advancement to generating fancy borders is to use a tiling border image, and using different images for each corner. If you come up with something interesting, please mail it to me.

Would it be at all possible for you to give me a line of sudo code so I can start from something? Hmmm....

```
read in image

( clone,  viewport crop/flatten to get a image the right height,
     tile over that image with your horizontal edging image  )
(clone last image, flip it for the bottom edge )
swap first two images,  append vertically

repeat the last group but for side edges.

add corners as per 'fancy' example

save result.
```

The tricky part is getting the images that work together, and matching them up, especially to fit corners.

One solution may actually like to ensure the added edges are tiles to a whole tile, then resize it slightly (larger or smaller which ever is closer to the whole tile) so the tiles match the corner piece properly. All this however make require a much more complex script that calculates the number of tiles needed to fit the image best (rounding up or down for best match).

A generic script would be a useful addition. That is given 8 edging tiles (top, bottom, left right edges, plus four corners); it will edge the image as described. Of course some of those edges, could be rotated versions each other. You could even reduce this to a single 3x3 grid image, or corner-edge source image, for such a generic script.

An even fancier version may use edging images with transparency, masks, and the amount of extra border to add, for partial overlaping of the the original image.

The 'fancy' example given above in fact actually does overlap the image slightly in the corners, but it make no attempt to get the black 'thickness' right for all the corners. That however could be simple to rectify.

Further Ideas

The above is only a sampling of what you can do. You are only really limited by your imagination. If you do come up with soming interesting, please mail it to me so other can also benefit from your findings.

For other examples of just how far you can go with this process can be seen in Spymac Image Galleries which is impressive. (Reference courtesy of Santanu Misra).

How about a bubble-like glass overlay effects, as per Blair Art Studios.

Or adding a slight curve (with appropriate shadow effect), to the poloride photo.

Advanced Shadow Effects

Overlapping multiple shadowed images can be a tricky matter as shadows do not just simply overlap. You need to build up the shadow effects layer by layer, bit by bit. It gets even worse if you also want to take into account the height of different images, as the shadow blurs onto different layers also become different.

In such a case a top-down layering approach is recommended. Basically the top layer casts a shadow onto the next layer's opacity mask, which in turn produces changes the opacity mask for the next layer. That is intermediate layers will have shadow effects added to image part of that layer, and a seperate 'shadow producing' image for that layer. As long as the light source(s) do not move, the result should be correct.

Anyone like to give it a go, say with a stack of photos? Mail Me.

Created: 8 February 2004
Updated: 22 June 2006
Author: Anthony Thyssen, <A.Thyssen@griffith.edu.au>
Examples Generated with: ImageMagick 6.3.1 12/22/06
URL: `http://www.cit.gu.edu.au/~anthony/graphics/imagick6/thumbnails/`