Dodges & Burns for your Digital Camera!

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2003 as GuruGram #17
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Back in the "slopping in the slush" bad old days of conventional darkroom photography, each phototech had their own secret methods of improving their final results. Two of the most important of these were the **dodging paddle** and the **burning card**.

A dodging paddle was used to **hold back** light by waving it over areas of the paper being exposed in the enlarger. This would **lighten** certain selected areas. The burning card had a small hole in it, so only selected photo portions would receive **more** light. This would **darken** selected areas instead. In this manner, errors of exposure or lighting could sometimes be corrected.

We've already seen in **GuruGram #15** on **Swings & Tilts** how it is possible to take any **BMP bitmap image** and replace each pixel with interpolated nearby ones, thus correcting geometrical distortion, creating architectural perspective, or doing intentional scanner-to-offaxis-photo fitting. A unique high resolution cubic spline **Basis Function** technique gets used for maximum image fidelity.

In my new **Dodges & Burns** utility, we instead modify each and every selected bitmap pixel **in place**, following a rule or set of rules. Besides traditional dodging and burning, you can selectively alter intensity, saturation, gamma, contrast, hue, chroma, and vignetting, and even handle selective **transparent alpha overlays**. Plus doing masking, gray conversions, waterfall backgrounds, color separations, knockouts, magic low JPEG artifact backgrounds, and gamma plots.

We'll note in passing that, yes, you can do much of this with **Photoshop**. And even some of it with **ImgViewer/32**. But here, all of the algorithms are out front where you can directly work with them and customize them to your own needs. You also have absolute and total control.

In general, the process is as follows: You first load the **Dodges & Burns** utilities into a favorite word processor or editor. You then decide what you want to do and then create a **map** or set of visual instructions over what portions of the image will get emphasized, shifted, or altered in what manner.

You then assign suitable filenames, both to locally available resources and target outputs. You next pick an appropriate **mode** for what you wish to accomplish. Followed by saving the routines under a new filename **as an ordinary ASCII textfile**. Finally, you send your file to **Acrobat Distiller**, and your corrected BMP image should appear in the expected file area.

These routines use my **Gonzo Utilities** and a copy needs to be made locally available. While proudly not WYSIWYG, the process is easily learned and the processing times are amazingly fast. A typical dodge and burn may take you a few seconds. There are detailed examples at the end of **DODBUR01.PSL** which we will expand upon here.

More on **Gonzo** and additional support appears on our **PostScript**, **Acrobat**, **Bitmapped Fonts**, and **GuruGram** library pages. While custom assistance can be found by way of our **InfoPack** library.

How it works

The basic dodge and burn concept is to grab a bitmap image one pixel at a time, perhaps modify that pixel in a **mode** selectable way, and then rewrite the changed image to a new file. A companion **map** array often gets used to determine which pixels will get affected to what strength. To both simplify and speed things up, the bitmaps are read and written one horizontal line at a time. Additional info on the **bitmap** format appears in **GuruGram #14**.

At present, there are about a dozen different modes available. An appropriate mode gets selected for the task at hand by predefining it as **/dodgeburn** before activating the main **dodge&burn** code loop. Some of the modes now include…

dbmap	Make an **image of the map** array. Certain maps may require scaling.
dbluminance	Lighten or darken each pixel's **luminance** per the current map. Does a classic dodge and/or burn.
dbsaturation	Increase or decrease each pixel's **color saturation** per the current map.
dbgamma	Increase or decrease each pixel's **gamma value** per the current map. Raises or lowers contrast or enhances extreme whites or blacks.
dbhue	Shifts the hue and **changes colors** but preserving saturation.

more…

dbvignette	Creates **Vignette edge effects** by altering edge luminance.
dbgray	Creates a **Gray Image** by using NTSC and human eye weighted luminance.
dbredsep	Creates a **red color separation** by blackening blue and green pixels.
dbgreensep	Creates a **green color separation** by blackening red and blue pixels.
dbbluesep	Creates a **blue color separation** by blackening red and green pixels.
dbmask	Creates a **black mask** of all pixels below a certain luminance.
dbtransblend	**Transparently alpha combines** a background and overlay image.

The Map

The **map** is an array of numbers that sets the strength of the desired effect. Its size can be anything from 2x2 on up and it need not be square. A key rule…

The array position is the same as the image position.

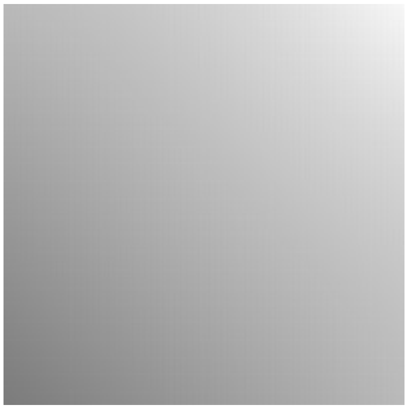
Maps are automatically expanded and smoothed so they end up pretty much the same size as the bitmap image. An final interpolation then gives an exact fit.

Let’s look at a map example. A typical "flash problem" is that the bottom left of your picture is "too hot" while the exposure darkens towards upper right. Here’s a luminance correcting mask that may fix this shading…

```
/burn0.5SW {     % 50% luminance reduction
   /dbdata [
       [ 0.750 0.825 1.000 ]
       [ 0.625 0.750 0.820 ]
       [ 0.500 0.625 0.750 ]
               ]    store } store
```

Note that we’ve told the bottom left to darken by half amplitude, to reduce the diagonal to three-quarters amplitude, but to leave the upper right at the full original luminance.

And here is what this map looks like after lowpass filtering and expansion…



Because the human eye is exceptionally sensitive to slope discontinuities, an expansion and filtering is normally done to get the map up to size. The present lowpass filter algorithm is somewhat "Gaussian like". Expansion and filtering is done by doubling the data points through averaging. **Each point is then replaced by half its value plus half the average of the previous and post point.**

Five repeat filterings are typically done, set by a **/filtertrips** variable. This 1024X data point multiplication eliminates anything disconcerting to the eye, but may need reduced when sharper mask edges or larger array sizes (>20x20) are in use. You can usually use the **dbmap** mode to view your actual maps.

While maps normally have data points in the 0 to 1 range, they may differ in certain modes. For instance, a hue map might be in degrees of change from -360 to +360, while a mask map might be an actual white threshold such as 254. Note that **it is usually better to darken than strengthen** if you are to avoid running into any "superwhite" or "supersaturation" problems.

Gamma Curves

By suitably tampering with the **transfer function** of any image processing system, you can selectively raise or lower midtone contrast, enhance or reduce highlights or shadows, and do any of a number of other special effects and improvement tricks. The **gamma curve** usually starts at black 0,0 and ends up as white 1,1. For image fidelity, you should have a straight line between.

But, if your gamma curve is suitably "bent", all sorts of corrections are possible. Ferinstance, a 0.5,0.5 slope **above** unity **raises** your midtone contrast. A **less** than unity slope **lowers** midtone contrast. A "fast start" from 0,0 expands shadow detail, while a "slow start" minimizes it. Doing so near 1,1 alters highlights.

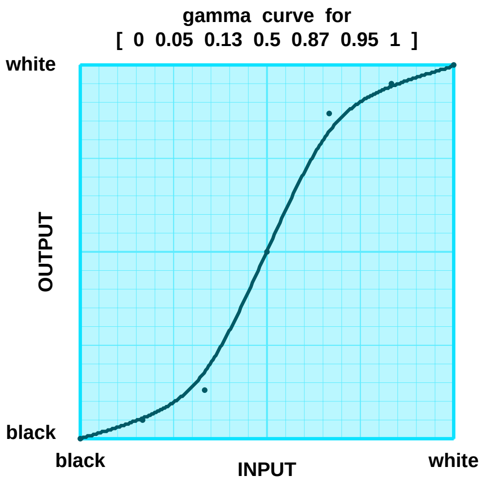
A **histogram** analyzer is one way to find out where all your present pixels are and how they can be improved. But most often, it will be obvious that there is a contrast, highlight, or shadow problem. The usual correction method is to create a 256 element **table lookup** array.

My approach here was to define a gamma array such as this contrast raiser…

```
/gamma [0 .05 .13 .5 .87 .95 1] store
```

Any number of **equally spaced** points can be used. This is expanded and 1D filtered the same way we did the 2D map expansion to yield a 256 entry table. The actual correction then consists of a fast and simple table lookup adjustment for red, blue, and green.

If you do a **/plotgammaflag true def** you can view the actual gamma curve as .PDF output…



Some Examples

Let’s briefly look at some of the sneaky things you can do with **Dodges & Burns**, along with their underlying algorithms. Largely following the end demos…

Show the Mode Map — Uses mode **dbmap** and simply replaces each image pixel with a gray pixel equal to the map value using 0 = black and 1 = white. Note that some special purpose maps will need scaled before using this feature. Handy to make sure the map is doing what it is supposed to and checking its alignment on the image to be corrected.

Adjust the Luminance — Uses mode **dbluminance** and multiplies each red, blue, and green pixel value by the map factor. Thus doing a classic dodge and burn. A map value of 1.0 leaves the pixel color the same. A map value of 0 replaces it with black. Map values above 1.0 will brighten the pixel, but this can lead to oversaturation or overwhite conditions. These are clipped automatically.

Setting **/holdwhite** to true will prevent any changes to true white (255) pixels. This is handy for paint "tow-over" transparency or later use of **MAGFILL1.PSL**.

Similarly, booleans **makeredchanges**, **makegreenchanges**, and **makebluechanges** let you selectively apply corrections to individual red, blue, or green pixel values.

Change Perceived Saturation — Uses mode **dbsaturation** to alter the "brilliance" or "depth" of each color. The perceived saturation of each color follows this NTSC and eye response formula...

gray equivalent = 0.59 green + 0.30 red + 0.11 blue

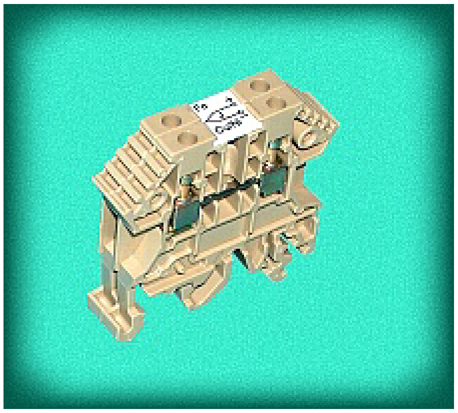
To change perceived saturation, the gray equivalent of the pixel triad is first calculated using the above formula. The current saturations are found as the **differences** of **red-gray**, **green-gray**, and **blue-gray**. These differences then get scaled by the map value to increase or decrease the color depths.

As before, booleans **makeredchanges makegreenchanges** and **makebluechanges** let you selectively apply corrections to individual red, blue, or green pixel values. Note that this is a different definition of saturation than in HSB.

Change to Gray Image — Uses mode **dbgray** to change a color image into an equivalently perceived black and white one. Calculates the gray equivalent of the pixel triad as above and then identically substitutes its value for all three color pixels. Also useful to help determine whether a subtle problem is one needing luminance or saturation correction.

Localized Blob Burn — Uses mode **dbluminance** to change only a small portion of the image. The only difference is in the map where a tight spot is created rather than a gradual gradient. As before, selective red, blue, or green corrections can be individually made. Handy to correct a smaller defect in an image.

Luminance Vignette — Uses mode **dbluminance** to do an edge fadeout or a self framing of an image such as this example...



Actually, vignetting started out as **defects** in plate coating and off-axis lens aberrations. While quiet useful as special effects, they easily can become way too garish or cutsey-poo. And thus should be used with caution. A vignette is just a **dbluminance** burn using a special map that either emphasizes or fades the borders. Here is a typical "self-framing" mask example...

```
/dbdata [  /vv 0.55 store /ww 0.7 store
    [ vv vv vv vv vv vv vv vv vv vv vv vv vv vv vv vv ]
    [ vv ww 1 1 1 1 1 1 1 1 1 1 1 1 ww vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv 1 1 1 1 1 1 1 1 1 1 1 1 1 1 vv ]
    [ vv ww 1 1 1 1 1 1 1 1 1 1 1 1 ww vv ]
    [ vv vv vv vv vv vv vv vv vv vv vv vv vv vv vv vv ]
        ] store } store
```

A "fade to white" vignette is a tad trickier. But can easily be done with use of our upcoming **variable transparency** dodge & burn mode.

Adjust the Hue — Uses mode **dbhue** to rotate all the colors uniformly around the color wheel. Small adjustments make an image "warmer" or "cooler". Major ones handle mind-boggling special effects.

HSB is simply a remapping of RGB that gives us direct control of hue, the actual saturation, and brightness. In the HSB color model, the **weakest** RGB value gets replaced by gray and becomes the **brightness**. The **difference** between your **strongest** and **weakest** of RGB becomes the **saturation**. And the **hue** is determined by the **ratio** of the **strongest** and **next strongest** RGB elements.

It turns out there are **six** different **hexants** in the 0 to 360 degree HSB color wheel. Covering red through yellow, yellow through green, green through aqua, aqua through blue, blue through violet, and finally violet back to red. Each one has to be separately dealt with...

0 to 60 degrees	Saturation is set by (R-B). Hue is 0 + 60*(G-B)/(R-B). Brightness is set by blue. Colors range red through yellow.
60 to 120 degrees	Saturation is set by (G-B). Hue is 60 + 60*(1-((R-B)/(G-B))) Brightness is set by blue. Colors range yellow through green.
120 to 180 degrees	Saturation is set by (G-R). Hue is 120 + 60*(B-R)/(G-R). Brightness is set by red. Colors range green through aqua.
180 to 240 degrees	Saturation is set by (B-R). Hue is 180 + 60*(1-((G-R)/(B-R))). Brightness is set by red. Colors range aqua through blue.
240 to 300 degrees	Saturation is set by (B-G). Hue is 240 + 60*(R-G)/(B-G). Brightness is set by green. Color range blue through violet.
300 to 360 degrees	Saturation is set by (R-G). Hue is 300 + 60*(1-((B-G)/(R-G))). Brightness is set by green. Color range violet through red.

Hue image pixel correction gets done by converting the present RGB pixel triad into HSB values, selecting the correct hexant, adding or subtracting your hue correction from the map, reselecting a correct hexant, converting back to RGB, and then storing to the new image. While avoiding any divide-by-zeros on hue calculations. Allowable map values range from -360 to +360 degrees.

Note that the "saturation" definition in the HSB model does **not** allow for human color perception differences. If you your shift hue too far, certain colors may appear much brighter or darker than normal. Moving any of our **dbsaturation** perception techniques over into HSB space gets tricky as it could cause both missed colors and supersaturation problems.

Make a Chroma Waterfall Background — Uses mode **dbchroma** with significant hue shifts to create rippling colors. Normally used on a background only that starts with its central color. Actual subjects are then pasted in later.

Adjust Gamma or Contrast — Uses mode **dbgamma** and the **gamma array** we looked at above to table lookup correct each pixel color value. The 256 entry table is created ahead of time for speedup. Once again, steep midrange slopes increase contrast, steep blacks expand shadows, steep white expand highlights.

Make a Black Mask — Uses mode **dbmask** to create an opaque knockout of the subject. A pixel-by-pixel comparison is made with a threshold value (often 254) and black (0) gets substituted for lower values and white (255) for higher. Also useful to create litho and other extreme contrast effects.

Do Color Separations — Uses **dbredsep**, **dbgreensep**, or **dbbluesep** modes to create red-only, green-only, or blue-only images. Each of the two unwanted colors is simply replaced by black 0. While retaining a 24-bit RGB color image. Besides traditional separations, these routines can sometimes extract extra detail for near-white or near-black portions of an image. For one particular color (the darkest of the lights and vice versa) may often contain more recoverable contrast and more luminance detail.

Transparent Alpha Overlay — Uses the **dbtransblend** mode to selectively combine two images into one. As an upgrade and improvement over our older **BLENDER.PSL** found in our **Bitmapped Fonts** library. Uses a pair of **background** and an **overlay** image files. If the map is 0, the background pixels are used. If the map is 1, the overlay pixels are used. Intermediate map values create a progressively transparent overlay. The algorithm used for each R, G, or B pixel component is...

new pixel value = background + map*(overlay-background)

At present, both input images **must** be both the same size and have the same horizontal and vertical resolution. If needed, the usual cropping or scaling can be used to make this happen ahead of time. The output combined image is in itself opaque and can be used like any other normal .BMP file.

As written, the **dbtransblend** code does a classic alpha overlay. It can easily be converted to any of the two dozen or so alpha mask/replace options offered by **Acrobat 5**. Plus dozens more. As can various custom true chroma or luminance keys. As can combining two different sized images. Such code extensions await your funding as a **Synergetics Partner** or **Banner Advertiser**.

For More Help

There are several support utilities on my website that can enhance your use of the **Doge & Burn** and the **Swings & Tilts** packages. These include **NOWHIT01.PSL** that eliminates any true white pixels to prevent "punchthru" on a white knockouts, our **MGFILL1.PSL** magic backgrounder to get rid of JPEG edge artifacts, **HISTOG01.PSL** histogram generator, and the **BMPRPT01.PSL** .BMP image analyzer.

Additional background along with related utilities and tutorials appears on our **GuruGram**, **PostScript**, **Acrobat**, and **Fonts & Bitmaps** library pages.

Consulting assistance on any and all of these and related topics can be found at **http://www.tinaja.com/info01.asp**. As can our image improvement services.

Additional **GuruGrams** await your ongoing support as a **Synergetics Partner**.