# A Tutorial and Guide to my Image Post Processing Tools

**Don Lancaster**
**Synergetics, Box 809, Thatcher, AZ 85552**

**http://www.tinaja.com**
**don@tinaja.com**
**(928) 428-4073**

Back in the days of "slopping in the slush" photography, it was essential that you got your image exactly right before you snapped the shutter. But these days, we have incredible collections of **image post processing tools** that let you repair and improve images to your heart's content. For superb results with today's tools, **at least ninety percent of your time and effort should be spent in postproc**.

We pride ourselves in having the finest product photos on **eBay**, bar none. Much of this was simple attention to detail, using plain old **Paint** and the shareware of **ImageView32**. To these, I have added an ongoing collection of my utilities and tutorials that can let you do absolutely stunning postproc image improvements.

Some of these simply (and often obsessively) do my things my way; others go far beyond what is available in PhotoShop and similar commercial packages. All of them are pretty much platform independent and can give you absolute personal total control of exactly what you do how. **All with open, unlocked, documented, and easily modifiable source code**. What I thought I'd do in this **GuruGram** is summarize what these tools are and how you can use them.

Key goals are **sharpness and resolution to one pixel accuracy**; shadowless or nearly shadow free images; true vertical lines by way of architect's perspective; effective airbrush blending; true blended and antialiased lettering as perfect as possible for a given bitmap resolution; dramatic vignetting when and where appropriate; true HSB airbrushing for smooth gradients; background knockouts; JPEG artifact reduction; symmetry improvements; gamma & color correction; "infinite" depth of field by combining digital photography with scanning; high quality pixel interpolation; white punchthru elimination, multiple exposure background bumping, and great heaping bunches more.

Many of the actual utilities make heavy use of the incredibly superb **PostScript** general purpose computing language. Often helped along by my **Gonzo Utilities**. Generally, simple modifications are made to an ordinary text file which is then sent to **Acrobat Distiller** or **GhostScript**. Making use of **Distiller as a Host Based PostScript Interpreter**.

The distilling process usually reads and writes **.BMP bitmap files** in your intended manner. Most utilities are purposely and proudly not WYSIWYG. For advantages of convenience, power, control, device independence, and modifiability. Very often, a viewable result image is a second or two away from any action.

One caution: **Acrobat Distiller versions above 8.1 default to preventing disk file reads and writes**. The workaround is to always use Distiller from your Windows command line and activate it with an **acrodist -F**. Solutions for other platforms appear **here**.

Here is a summary some of our postproc routines. Some are new and some are older, so we'll present them alphabetically…

## Airbrushing — AIRBRUSH.PDF (tutorial) AIRBRUT1.PSL (utility)

Airbrushing takes a quadrilateral on a **bitmap image** defined by its four corner points and then **creates a smooth two dimensional blended gradient** of all intermediate points. This can be used to eliminate burns or speckle. Or to make an area more uniform. Or get rid of a telephone pole or wires. Or wherever traditional airbrushing was appropriate. A randomizing feature is included to add texture. Normal operation is in the HSB space, with a RGB blending option.

The utility uses the following major variables…

**/bmpinfilename** — input short .BMP filename
**/bmpinfileprefix** — long prefix for input .BMP filename
**/bmpoutfilename** — output short .BMP filename
**/bmpoutfileprefix** — long prefix for output .BMP filename

**/textureflag** — true/false for texture enrichment
**/textdepth** — amount of texture enrichment when used

**/usehsb** — default true = hsb blend false = rgb blend
**/hueshift** — corrects red blend if +0.5 default = 0.

**/airbrushboundaries** — array of **[llx lly ulx uly urx ury lrx lry]**

That are interpreted by these routines…

**fixpaint** — inverts paint vertical values given **height** input
**airbrushquad** — does the blend, reading **airbrushboundaries**

Here is a **full panel example**. Cylindrical effects can be gotten as **in this example** by doing a left side and then mirroring. Code extensions can take care of the rare cases where a blend may go "the wrong way" around the hue circle. And deal with any unwanted color fringing near gray values. Processing time normally takes a few seconds for smaller areas.

We'll stuff our usual reminder in here that **PostScript strings demand a double reverse slash anytime a single one is needed**. Especially in filenames.

## Architect's Perspective — KEYCOR01.PDF (tutorial) FIXTLT01.PSL (utility)

Another name for Architect's perspective is **keystone correction**, based on the **tilt** adjustment on a view camera. The goal is to make all vertical lines in an image appear truly vertical. To one pixel accuracy. Especially the sides of buildings or telephone poles. Or, in this electronic product photography **image example**.

Several approaches are possible for keystone correction. With these particular files, the left edge to be "verticalized" is projected from its **center** position to its top edge position and **oldtopleft** and **newtopleft** are noted. The right edge to be "verticalized" is projected from its center position to its top edge position and positions of **oldtopright** and **newtopright** are also noted.

This version of the utility uses the following major variables…

**/rootfilename** — input short .BMP filename
**/diskfileheader** — long prefix for input .BMP filename

**/oldtopleft** — projected position of original left "vertical".
**/oldtopleft** — projected position of desired left "vertical".
**/oldtopright** — projected position of original right "vertical".
**/oldtopright** — projected position of desired right "vertical".

**/nowhite** — true = remove whites to prevent punchthru

That is activated by this routine…

**mainloop** — do keystone correction to new .BMP file

An optional white punchthru corrector is included and activated by way of the **nowhite** Boolean.

Some older approaches to tilt correction were found in **SWINGTLT.PDF** (tutorial) and **SWINGT01.PSL** (utility). In these, both the **tilt angle** and the **center neutral position** were entered instead of the projected verticals. While somewhat more intuitive, considerably more trial and error was sometimes involved.

All of these keystone routines to date do have an algorithmic flaw: **curvature may be introduced for large tilt correction values**. This is caused by attempting to remap rows only instead of moving both row and vertical position.

Curvature typically shows up at a gain of **1.15** and becomes significant at **1.20**. These values are often well above normal **eBay** product photography adjustments. A fix is in the works using our latest full .BMP remapping based on the **starwars transformation** and our new **BMP2PSA.PDF** core utilities.

This newer and more accurate method might end up significantly slower.

## Background Knockout — KNOCKBACK.PSL (utility)

Conventional automated knockout programs have a crucial problem: **They are worst where they are needed most**. Namely in deep shadow or otherwise ambiguous areas. **KNOCKBACK.PSL** is a "semi-automated" utility that removes most background to white. Once removed, the remaining image can be transparently pasted onto a new background. Or a new background (with or without vignetting) can be slid "underneath" any white areas.

Besides adding interest and reducing harshness, a new color background having slightly random pixel mottling can dramatically reduce ultimate JPEG edge artifacts on any later conversion.

To begin, a white **punchthru elimination** (see below) must first be used to eliminate any true whites inside the image itself. And **a continuous white outline is traced around the active image portion to be retained**. It is super important that image magnification or any color correction **not** be done after internal true whites are eliminated. And equally important that there are **absolutely no holes** in the white traced image outline.

**KNOCKBACK.PSL** then proceeds with the following algorithm: For each row, **white pixels are added from the left margin until a white pixel is encountered**.

Then white pixels are added from the right margin until a white pixel is found. The process is then repeated for each **column**, starting at the top and working down and starting at the bottom and working up.

Most of the background will usually end up knocked out to white. Although some occasional undercuts or unusual areas may need individual manual retouching.

This version of the utility uses the following major variables…

**/sourcefilename** — input short .BMP filename
**/bmpinfilenameprefix** — long prefix for input .BMP filename
**/targetfilename** — output short .BMP filename
**/targetfilenameprefix** — long prefix for output .BMP filename

That are interpreted by these routines…

**/grabbitmap** — capture bitmap for analysis.
**/knockback** — remove outside whites to border
**/savebitmap** — save knocked out image.

## Background Slideunder — NUBKG01.PSL (utility)

There are many advantages to providing a background other than white to an image. The final appearance can be less harsh; edges can be sharpened or emphasized; and any JPG artifacts can be significantly reduced.

**NUBKG01.PSL** is my automated workhorse routine for many of my **eBay Images**. There is no tutorial yet as such. This utility basically adds to **DODGEBUR.PDF** and **DODBUR01.PSL** in a purpose targeted manner.

The image to be processed must first be knocked out to a white background and have any white punchthru eliminated. An expandable choice of several slightly mottled color background patterns are available. The mottling adds texture and interest. More importantly, the slight random variations dramatically reduce JPG edge artifacts at the price of a slightly larger file size.

The utility works by seeking out white pixels and substituting the next available pixel in a chosen and predefined mottled background pattern.

## Bitmap Typewriter — BMFAUTO1.PDF (tutorial) AUTOBMF1.PSL (utility)

The Bitmap Typewriter provides the highest possible resolution full pixel typography and does so with incredibly legibility down to astonishingly small point sizes. Such tasks as relettering an entire test equipment panel are now quite feasible. As per **this example** that even includes rotated text.

In normal use, the needed lettering or relettering is created to a scratch bitmap and then cut-and-paste copied to the image being reworked. **The characters are absolutely pixel locked and fully antialiased** on a pixel-by-pixel basis **without** any damaging smoothing or low pass filtering in use.

Although any combination of **PostScript** fonts can be used, best low pixel results are often obtained using **Myriad Pro** or **Helvetica** families. Any number of letter colors can be fully blended to as many as four backgrounds. Letters and background can be any reasonable color combination.

Font sizes are defined by the pixel count width and pixel count height of an upper case letter "A". Wider and narrower letters are proportionally forced into the appropriate nearest available width. The portion of the chosen character that maps into any particular pixel is sampled 36 times. From those samples, an appropriate antialiased blend of letter to background color is created. Repeating for each of the RGB planes.

The utility uses the following major variables…

**/targetfilename** — output short .BMP filename
**/targetfilenameprefix** — long prefix for output filename
**/globalkern** — sets the global kerning, often 1.
**/kern+char** — sets the positive kerning character, often "~".
**/kern-char** — sets the negative kerning character, often "‘".
**/yinc** — sets the line to line vertical pixel spacing

That are interpreted by these routines…

**setfontfamily** — picks the current PS font family in use

**setbmsize** — sets upper case "A" pixel height and width
**curcolor** — defines the current RGB color
**setbackA** — maps the background color (also B,C,D).
**setgraystring** — images the current character string.

At present, imaging **x** and **y** positions input to **setgraystring** will start at that location. Use of **0,0** instead will continue on the present line. Characters are generated on the fly as they are needed and then saved for possible reuse.

Near the end of any line, a character break will force a new combined linefeed and carriage return. With distance set by **yinc**. The background can be split into as many as four color zones. Kerning is normally done by inserting appropriate "~" or "‘" characters whenever a positive or negative one pixel kern is wanted.

The results of the bitmap typewriter are usually stunningly impressive. Especially when lettering drops down below "pseudolegible" sizes. Results can be further improved by going to **subpixel** techniques, but these can be rather complex and are strictly limited to specific LCD displays.

## Core AOS Utilities — BMP2PSA.PDF (tutorial) AOSUTIL1.PSL (utility)

Some bitmap manipulations can be done one pixel at a time. Others demand at least a row of data being available. But for the really general and really powerful stuff, **each and every pixel in the original .BMP file should be simultaneously available for access.**

Total access becomes crucial when adjacent pixels in both directions are needed for fancy filtering or interpolation. Or when nonlinear transformations are to map data from one area of the original bitmap to somewhere else.

I recently created a set of core utilities that "open up" a bitmap so that everything is accessible all at once. This is based on using a **PostScript array of strings**. The code accepts a bitmap, converts it to three arrays of strings for processing, and then resaves to a new bitmap or to a PostScript image.

Our **Airbrush Utilities** were the first of an expected continuing series of apps that make extensive use of these AOS techniques. In general, these core AOS utilities get "built in" to fancier routines with more tightly targeted uses.

The core AOS utilities use the following major variables…

**/arrayfilepathprefix** —long prefix for input & output files
**/inputbitmapfilename** —input short .BMP filename
**/outputbitmapfilename** —output short .BMP filename

**/redAOSfile** — red plane of original bitmap image
**/greenAOSfile** — green plane of original bitmap image
**/blueAOSfile** — blue plane of original bitmap image

That are interpreted by these routines…

**/mergestr** — merges two strings (from Gonzo)
**/string2array** — converts string to array (from Gonzo)
**/array2string** — converts array to string (from Gonzo)
**/file1** — string read as a file (example)
**/file2** — string written as a file (example)

**/getredAOSrow** — extract one red row from redAOSfile
**/getgreenAOSrow** — extract one green row from greenAOSfile
**/getblueAOSrow** — extract one blue row from blueAOSfile

**/convertAOStoPSimage** — convert string array to PS image
**/convertAOS2BMPimage** — convert string array to output bitmap
**/inputbitmap2AOS** — convert input bitmap to PS array of strings

## Dodging and Burning— DODGEBUR.PDF (tutorial) DODBUR01.PSL (utility)

In a traditional darkroom, you used a **dodging paddle** to hold back on the negative's light in certain areas, making your print **lighter**. Or a **burning card** with a small hole in it to add to the negative's light. This time making your print **darker**. And thus enhancing selected image areas.

Digital dodging and burning uses a mask and a set of rules to selectively change certain image bits. **The rule is individually applied to one pixel at a time**. Besides traditional dodging and burning, you can selectively alter intensity, saturation, gamma, contrast, hue, chroma, vignetting, image substitution, and even transparent overlays. Plus doing masking, gray conversions, silhouettes, waterfall backgrounds, color seps, knockouts, backgrounds, and gamma plots.

Ferinstance, you might have a mask that is darker at lower left and lighter at upper right. And use this mask to improve any subject lighting that is excessively hot near the camera. The masks can be quite simple and of low resolution. They are automatically expanded to fit the exact image size and are triply (or more) filtered for the smoothest possible transitions between their areas.

The **DODBUR01.PSL** utilities use the following major variables…

**/diskfileheader** —long prefix for input & output files
**/diskfilesourcename** —input short .BMP filename
**/diskfiletargetname** —output short .BMP filename

**/arrayfilepathprefix** —long prefix for input & output files
**/inputbitmapfilename** —intput short .BMP filename
**/outputbitmapfilename** —output short .BMP filename

**/dbdata** — the redefined mask pattern to be used
**/dbmap** — the redefined rule set to apply to the mask
**/gamma** — an array of gamma values
**/redweight** — balance value for gray equivalence, usually 0.30
**/greenweight** — balance value for gray equivalence, usually 0.59

**/blueweight** — balance value for gray equivalence, usually 0.11
**/makeredchanges** — apply to red pixel plane? true/false
**/makegreenchanges** — apply to green pixel plane? true/false
**/makebluechanges** — apply to blue pixel plane? true/false

That are interpreted by these routines…

**dodge&burn** — apply dbmask and dbrule to input image file
**dbmap** — show shading mask only
**dbluminance** — alter image brightness only per mask
**dbsaturation** — alter image saturation only per mask
**dbgray** — change image to NTSC gray
**dbmap** — do traditional localized dodge or burn
**dbhue** — waterfall, rainbow, or alter image hue per mask
**dbgamma** — alter image gamma only per mask
**dbmask** — extract black mask
**dbredsep** — extract red color plane
**dbgreensep** — extract red color plane
**dbbluesep** — extract red color plane
**dbtransblend** — transparently blend two images

## Exploring the .BMP Bitmap Data Format — EXPBMP.PDF (tutorial)

A tutorial on the fundamentals of the .BMP data format. This is often the best choice for postproc work in that the file is easily opened up to make each and every pixel of all three colors readily available. Plus, there are no compression or generation losses.

Only **after** all postproc is done should your results be converted into more compact **.JPG** files for distribution.

A .BMP file consists of a header followed by a body of image data. It is extremely important that the header data **exactly match** that needed by the image data. Most any match error will cause severe distortion or outright file failure.

**.BMP Bitmaps build from the bottom up and left to right**. The **blue-green-red** data sequence is "backwards" from what you may expect.

The needed .BMP **padding bytes** can cause confusion. Because **each new row must start on a 32 bit boundary**, zero, one, two, or three padding bytes have to be added to the end of each data row.

While the actual math is obtuse, the required number of padding bytes simplifies on down to…

```
Padding bytes needed = xpixels 4 idiv
```

### Extreme Display Legibility — LEGIBLE1.PDF (tutorial)

A tutorial on how to gain screen legibility that is **better** than the printed page!

Includes subpixel direct digital display addressing, true post anti-aliasing, authoring techniques for improved legibility, and several additional techniques.

### False Color & Rainbow Improvements — FALSECLR.PDF (tutorial)

Tutorial and sourcecode shows how to improve false color and rainbow effects by equalizing saturation and modifying hues. Includes table lookups exportable to most any language.

### Gonzo Utilities — GONZOTUT.PDF (tutorial) GONZO.PS (utility)

These form my ongoing in-house custom **PostScript** combination illustration and pagemaking package. I use them for all of my presentation, engineering, and consulting work. They are enormously useful when developing new image postproc code.

The full utilities can be prepended to most any routine by modifying…

```
(C:Program Files\\gonzo\\gonzo.ps) run
```

As noted before, Distiller versions newer than 8.1 default to preventing diskfile reads or writes and must be activated by a command line **acrodist -F**.

Some of the more common Gonzo procs can be predefined in our postproc utilities so the routines can stand alone. Some more used routines include…

**/mergestr** — merges two strings
**/random** — generate random integer
**/stopwatchon** — start timing stopwatch
**/stopwatchoff** — stop timing stopwatch and report

The latter two routines also have to tow along the **/resettimer**, **/stoptimer**, and **/reporttimer** internal Gonzo resources.

### Imaginative Images — IMAGIMAG.PDF (tutorial)

One of our earliest tutorials on image postprocessing techniques. Covers many of the basic rules on propping, scanning, enhancement, and conversion of digital images.

Expanded upon and more detail added by our later tutorial of **STEPPREP.PDF**.

### Inverse Graphics Transforms — INVEGRAF.PDF (tutorial)

When modifying graphics images, an inverse or "comes from" transform may be needed instead of the usual "goes to" Tutorial and examples show some of the math techniques involved.

### JPG Artifact Reduction KNOCKOUT.BMP (sampler)

The **.BMP images** formats normally used in image postproc are often converted into more compact .JPG images for final distribution. Since .JPG images are lossy, artifacts can appear. In particular, a "ghosting" along white or solid color edges is both common and annoying.

A useful method to eliminate many .JPG ghosting artifacts is to have a "mottled" or a "randomized" background. The .JPG coding process will get confused and not create continuous ghosting artifacts. With only a slight penalty of a modest increase in file size.

One effective route to mottled backgrounds is to use our **KNOCKOUT.BMP** sampler. A second is to very early on isolate and expand an interesting image area into a suitable background pattern. A third is to use the ready-to-go automatic backgrounds of **NUBKG01.PSL**.

### Nonlinear Graphic Transforms — NONLINGR.PDF (tutorial)

This was an older tutorial I published in **Circuit Cellar**. It reviewed the fundamentals of both linear and nonlinear graphics transforms. Included were isometric, starwars, perspective, rootbeer, spherical, tunacan, glyphpath, and scribble transforms.

Bunches of earlier typography transforms appeared way back in **PSSECRETS.PDF**.

### Pixel Interpolation Algorithms — PIXINTPL.PDF (tutorial)

A tutorial review of popular pixel interpolation schemes including Bilinear, Nearest Neighbor, Bilinear with lookup, Modified 3x3, and Bicubic. Includes detailed bicubic math derivation.

Additional details on Bezier curves, cubic splines and bicubic techniques appear in our **Cubic Spline** library pages.

### Punchthru Elimination —

If a white background is going to get substituted and if there are still white pixels **inside** the image subject, then a **punchthru** can result. Where the background breaks through the middle of the image. With subtle to terrible results. Such as the map appearing in the middle of the weatherman on tv.

Punchthru elimination is simple enough that you usually will build it into a routine that is busy doing something else. Ferinstance, it is included as an internal part of **FIXTLT01.PSL**. And it was a second pass part of older **SWINGT01.PSL** .

The key is to **never write a 255**. Always write 254 instead...

```
punchflag {dup 255 ge {pop 254}if} if
```

Actually, just writing 254 to **one** of the three RGB bit planes should do the trick. Red, perhaps. Again, best done as a minor part of some other routine.

Punchthru elimination timing can be subtle and needs some thought. Obviously, it must be done **before** any background knockout to white. And **any scaling or gamma or contrast or brightness corrections are a big no-no** between the time elimination is done and whenever any white pixels are used as a replacement mask. And **every** pixel in the active portion of the saved image must be tested.

Stand alone punchthru elimination code can be extracted from **SWINGT01.PSL** .

### Some eBay Photo Secrets — EBAYFOTO.PDF (tutorial)

A tutorial on **eBay** specific photographic postprocessing. Details on the "hex" and "square" layout formats. Perspective correction. One pixel accuracy. Background fills. Relettering. Combined camera/scanner work. Vignetting. Image theft considerations.

Additional **eBay** related files appear in our **Auction Help** library pages.

### Step by Step Image Prep — STEPPREP.PDF (tutorial)

An intro tutorial on some insider postprocessing secrets. Why both cameras and scanners are useful. Fundamentals of background knockout. Pixel locking. Adding detail. Improving lettering. Final formatting. Heavily illustrated.

Uses a scanned image of a cable end and gives sequential examples of each postproc development step. A somewhat similar but earlier tutorial appeared as **IMAGIMAG.PDF**.

### Using Distiller to run PostScript — DISTLANG.PDF (tutorial)

Acrobat Distiller makes a superb host based PostScript computer. And is the crucial key to using the general purpose **PostScript** computing language for an astonishing array of useful tasks.

Many of our postproc utilities are based on taking a standard ASCII textfile, altering a few data values and then routing it to Distiller. Distiller in turn reads input bitmaps, suitably modifies them, and then rewrites them as new files.

This greatly expanded tutorial and update of an older file reveals the key concepts and insider tricks involved. More can be found in our **PostScript** library pages.

## Vignetting **DODGEBUR.PDF** (tutorial) **DODBUR01.PSL** (utility)

The fading of an image to white or black around its edges is sometimes known as **vignetting**. This was originally a **defect** in early photographs to cover erratic chemical solution coverage on the glass plate or off axis lens defects.

Today, vignetting can be used for special effects, but **should be reserved only for those times when it is appropriate**. A vignetting example **appears here**.

Vignetting is a variation on **DODBUR01.PSL** and also is a part of **NUBKG01.PSL**. A luminance mask is created with darkened and rounded edges, expanded and smoothed to fit, and then applied to the selected image. Punchthru elimination is usually required as well.

Save this "gee whiz" effect for special uses. It can get old really fast.

## For More Help

These routines evolved over many years. And may sometimes be spotty or uneven in places. The present intent is to use our fully two dimensional **BMP2PSA.PDF** core utilities to create new and improved versions of postproc routines. Our new **AIRBRUSH.PDF** was the first example of this ongoing code series. We intend to expand this **GuruGram** as new or improved routines become available.

News about the latest updates and addons should first appear in **WHTNU08.ASP** or later blog entries.

Similar tutorials and additional support materials are found on our **PostScript** and our **GurGram** library pages. As always, **Custom Consulting** is available on a cash and carry or contract basis. As are seminars and workshops. For details, you can email **don@tinaja.com**. Or call **(928) 428-4073**.