GlitchSort is a Processing application that uses pixel-sorting, degrading, quantization and Fast Fourier Transforms (FTT) to create glitchy images. This reference covers version 1.0b8, released for Processing 1.5.1. A bug in Processing 2.0b7 has hindered development for that version. Check for updates at http://paulhertz.net/factory/2012/08/glitchsort2/. GlitchSort requires ControlP5 version 1.5.2.

The Algorithms

GlitchSort provides four different sorting algorithms. It sorts bit-mapped images row by row or in zizgzag-scanned blocks. The sorting is randomly interrupted, with a probability determined by the breakpoint setting. You can also perfrom a complete sort of the rows of pixels (just uncheck the *break* checkbox). This can be an interesting way of analyzing the colors in an image. The complete sort is only available for the quick sort and shell sort algorithms. The other algorithms are simply too slow to use for a complete sort, but they have other uses.

Quicksort uses a divide and conquer approach to sorting. It partitions the array into smaller arrays, recursively. It operates over large distances within the array, and so tends to produce glitches over the entire image. **Warning:** This is a very fast sorting method for disordered arrays (most pictures, in other words) but will slow to a *total crawl* if fed an array that is already sorted or nearly sorted (or inverse sorted or nearly inverse sorted). Shellsort, though somewhat slower, does not have the same loss of efficiency on nearly sorted arrays. Unlike Shellsort, Quicksort will do color-swapping.

Shellsort also uses a divide and conquer approach. It partitions the array at intervals and sorts subarrays distributed over the intervals. The interval becomes apparent with successive interrupted sorts: Shellsort can be used to repeat images. It is relatively fast with better worst-case performance than Quicksort. Shellsort does not do color-swapping in the current implementation. Shellsort's *ratio* and *divisor* parameters can be "tuned" with the "<" and ">" keys, changing the sorting interval.

Bubblesort moves pixels one step at a time over to the left. As the sort moves from right to left, each pixel is exchanged with the one on its left until one with a smaller comparative value is encountered. Bubble sort is very slow, but the way it operates creates some interesting glitches, diffusing and smearing the image. It can be fun to watch for small images (if you're sufficiently geeky). Color-swapping also looks interesting with this sorting method.

Insert sort proceeds through the array from beginning to end, comparing every number against all remaining numbers. It is much slower than quick sort or shell sort, but it does leave all pixels in sorted order for as far as it proceeds. It can be used to create edge effects.

Implementations of these algorithms were based on examples in Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne (http://algs4.cs.princeton.edu/home/). See visual examples on the next page.

Figure 1: top left, Quick sort, single pass with breakpoint = 144, component sorting order = BSH; top right, Shell sort, 3 passes with breakpoint = 995, component sorting order = BSH; bottom left, Bubble sort, 5 passes with breakpoint = 999, component sorting order = BSH; bottom right, Insert sort, 100 passes with breakpoint = 999, component sorting order = BSH.

Commands and Shortcuts: Glitch Control Panel

There are extensive changes in GlitchSort compared with the previous release. There are not two control panels, accessed by tabs in the upper left corner, Glitch and FFT. The following commands are found in the Glitch Control panel for GlitchSort. When a command can be executed by typing a letter, the letter is indicated in parentheses.

Press and drag with the mouse to pan an image that is bigger than the window. Shift-drag works when the control panel is visible.

Open (O): shows a file dialog that lets you load an image file (JPEG, GIF or PNG).

Save (S): saves the file to a uniquely named file in PNG format in the local directory.

Revert (R): reloads the original file. The undo buffer is not affected.

Fit to Screen (F): when checked, fits the current image to the screen, scaling it up or down as necessary.

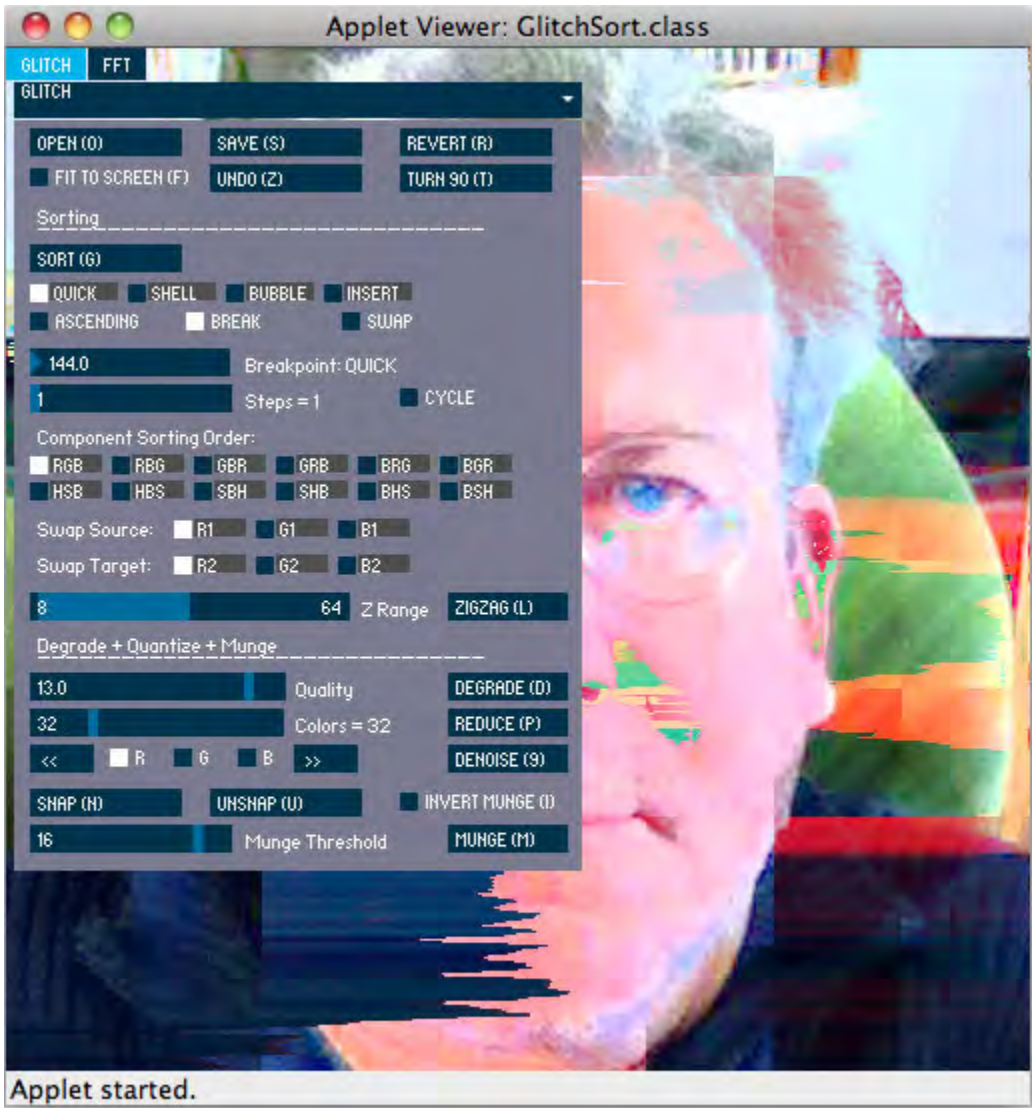


Figure 2: The Glitch Control Panel for GlitchSort, created with ControlP5.

Undo (Z): displays the image available before the most recent sort. More precisely, swaps the undo buffer and the display buffer, so typing 'z' a second time will restore the original image.

Turn 90 (T): rotates the image 90 degrees clockwise. This allows you to sort the image "vertically."

Sorting Section

The Quick (1), Shell (2), Bubble (3), Insert (4) check boxes select the sorting method.

Ascending (A): when this option is checked, the current sorting method will sort pixel values in ascending order. In an uninterrupted sort, this reverses the order of pixels that the unchecked option would provide. In interrupted sorts, behavior varies in interesting ways.

Break (B): when this option is checked, sorting is interrupted with a probability determined by the breakpoint setting. Only Quicksort and Shellsort permit uninterrupted sorting—there is no reason to use the extremely slow Bubble or Insert sort to do a complete sort.

Swap (X): when checked, this option causes color channels to be swapped when pixels are sorted, resulting in color glitches. This option is probably best used with Break also checked: results for complete sorts are slow and unpredictable.

Breakpoint: The probability of a sorting method being interrupted is determined by the *breakpoint* setting. In general, higher values decrease the probability of interruption, and so do more sorting. However, each sorting method behaves differently. Quick sort is sensitive from 1..999. Shell sort seems to do best from 900..999; lower values result in sorting only at the image edge. Bubble sort does well from 990..999: at 999 it will diffuse the pixels across the whole image. Insert sort seems to be effective from 990..999, but generally only affects the edge of an image. The breakpoint setting for each sorting method is stored separately and will be reloaded when you select the method.

Steps: the steps setting determines what portion of the rows of pixels in an image will be sorted. If steps is 1, all rows are sorted. When steps is 2, 1/2 the rows are sorted, when it's 3, 1/3 of the rows, and so forth.

Cycle (Y): when this option is unchecked and the Steps setting is greater than 1, the rows of pixels will be shuffled into a new order after each sort. If steps is 3, 1/3 of the randomly ordered rows will be sorted and on the next sort a different randomly ordered 1/3 of the rows will be sorted. Some rows are likely to be sorted repeatedly. When the option is checked, the rows will be exhausted before being shuffled into a new order. The checked option is potentially useful for animation.

Component Sorting Order

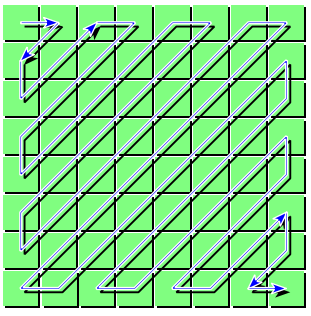
By default, the numerical value of a pixel represents four color channels, red, green, blue and alpha. When the pixels are sorted, the numerical value of the channels is used to reorder the position of the pixels in each row of pixels. The order of the channels can be changed to obtain different results from sorting. Additionally, the color can be represented in the HSB (hue, saturation, brightness) color space, with different channel orderings. See the Component Sorting Order example later this reference.

Swap Channels

When the Swap checkbox is not checked, sorting exchanges all channels in a pixel. When the Swap checkbox is checked, the values in two selected channels are exchanged before the full pixel values are swapped. In effect, the exchanged channels stay "in place" once the pixel values are swapped. The choice of source and target component determines which components are exchanged and thus which colors will appear in the sorted pixels. See the Swap Channels example for a visual explanation.

Zigzag Sorting

GlitchSort provides a generalized implementation of the zigzag scanning method used in one stage of JPEG compression. The Z Range range slider allows you to set maximum and minimum value of a randomly generated integer used as the edge dimension of a square of zigzag-scanned pixels. Set the two sliders to the same value to use a fixed value. The Zigzag (L) button will execute the current sorting algorithm over the zigzag scanned block. See the Zigzag Sorting example for details.

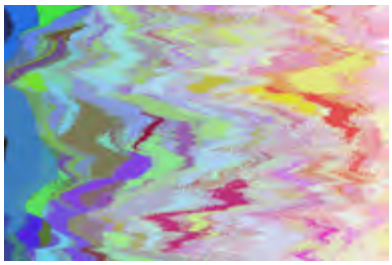


Zigzag scanning pattern

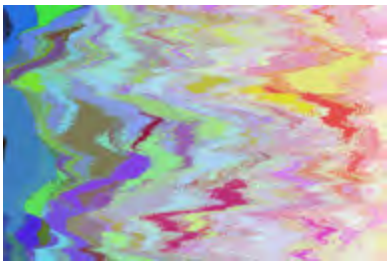
Example: Component Sorting Order

This series shows a complete sort in the horizontal direction, in descending order, using Quicksort on various different component sorting orders.

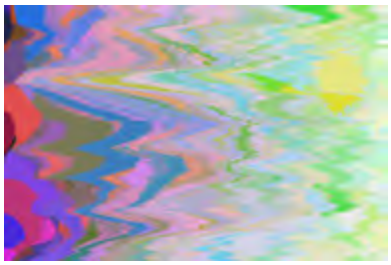
Original image



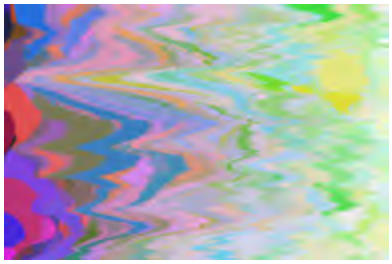
RGB



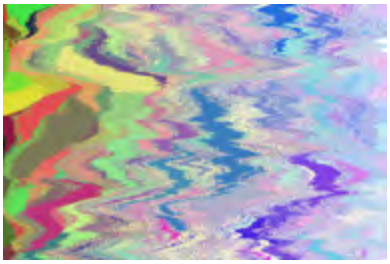
RBG



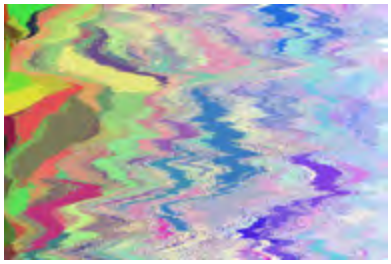
GBR



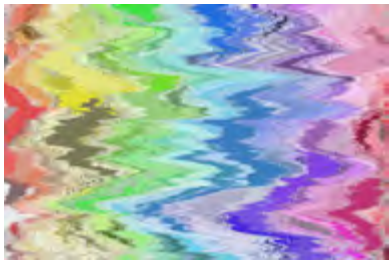
GRB



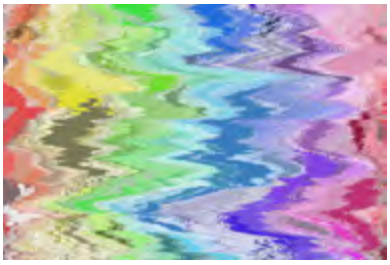
BRG



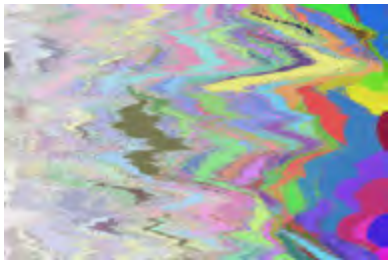
BGR



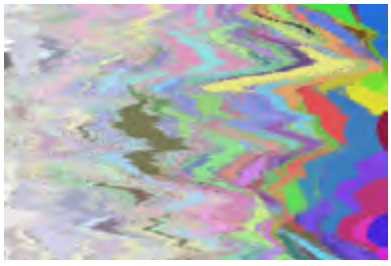
HSB



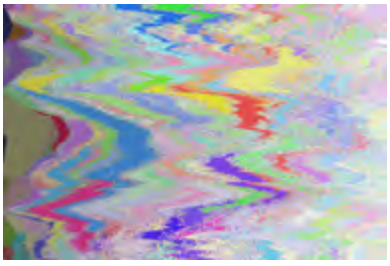
HBS



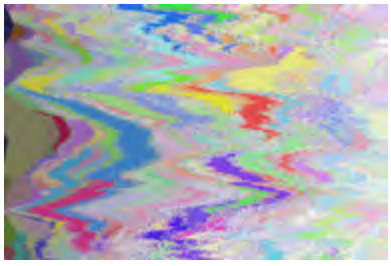
SBH



SHB



BHS



BSH

Example: Swap Channels

This series shows the effect of swapping selected channels when sorting. The image was sorted in HSB component order using Quicksort (breakpoint = 144) with an ascending sort. The letter codes below each image indicate the source and target channel swapped. Results can vary significantly according to the color characteristics of an image, ithe component sorting order, the channels swapped, the rotation of the image when it is sorted, the breakpoint setting, and whether the sort is ascending or descending. As always, experiment to find the best configuration for your purposes.

Original

RR

RG

RB

GR

GG

GB

BR

BG

BB

Example: Zigzag Sorting

This example shows how each sorting algorithm executes a zigzag sort, first without channel-swapping, then with channel-swapping. All sorts are in ascending order, unless otherwise noted. The breakpoints were set as follows: Quick, 100; Shell, 990; Bubble, 999; Insert, 999. The component sorting order is HSB. Where swapping is used, it is applied to B1 and B2. The Z Range was fixed at 32, resulting in blocks of 32 x 32 = 1024 pixels. The number of passes is noted with the image.

The Shell sort implementation currently does not support channel-swapping. This may change in a future release.

Zigzag sorting offers a number of interesting variations. Rotating and sorting can give interesting results, especially with channel-swapping enabled (Easter egg: type '_', underline, to execute four 90° rotations with zigzag sorting). Repeated sorts at random sizes set by the range control, possibly while rotating, will gradually produce an image that is similar to fully sorted images. As with linear sorting, changing the component sorting order can give very different results.

The same warnings about running Quicksort on fully sorted arrays hold here, but only if the zigzag block size is constant. Running a complete Quicksort on an image that is already completely zigzag-sorted will be extremely slow, unless the block size of the sorts is different. It's prbably a good idea to use Shellsort for complete sorting, to avoid problems.

Original

Quick, 1 pass

Shell, 8 passes

Bubble, 5 passes

Insert, 16 rotations, 64 passes

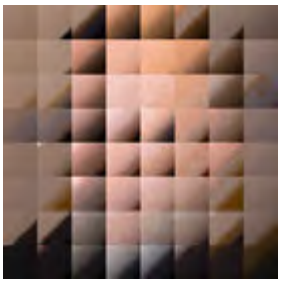
Quick + swap, 1 pass

Bubble + swap, 3 passes

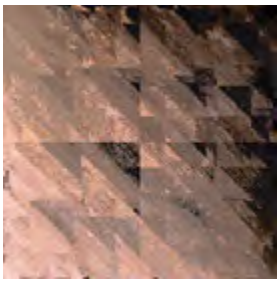
Insert + swap, 16 rotations, 64 passes

Shell, HSB, Z range 24–48, 90° rotation between sorts, 12 passes

Quick + swap, BHS, rotated 90° between sorts



A complete sort, BHS



Repeated quicksorts in the Z range 16–64, BHS

Degrade + Quantize + Munge

This section of the Glitch control panel provides two commands, Degrade (D) and Reduce (P), and a section for capturing a snapshot for later recall or for compositing with the currently displayed image using "dirty compositing," which I have chosen to call munging (see the Hacker's Dictionary for an explanation of the term: http://www.hackersdictionary.com/html/entry/mung.html). Expect glitchy and uncontrolled (but not completely uncontrollable) results.

Snapshots are easy. Munging is more complicated. Generally, the way to munge is as follows:

1. Open an image and glitch it until satisfied, then take a snapshot (N).
2. Reload the image (R).
3. Execute any undoable command, such as a sort, FFT operation, degrade or quantize.
4. Optionally, swap the undo and display buffers (Z).
5. Munge (M).

You can use two or even three different images. See the Munge Examples. Many effects are obtained by combining degrading or color reduction with repeated munging. Rotating before an undoable operation and then rotating back into place before munging can vary the direction of sorting and other operations.

Degrade (D): saves the image out as a JPEG and then reloads it. The JPEG quality setting is determined by the Quality slider. It's default value is low, 13.0, but can go all the way down to 0 if you really want to munge the file. The UP and DOWN arrows keys allow you to set Quality from the keyboard.

Reduce (R): reduces the number of colors in the image by running a color quantization algorithm. Set the number of colors desired with the Colors slider or the LEFT and RIGHT arrow keys.

The Channel Shift buttons, enigmatically marked with "<<" and ">>," shift the selected color channel left or right. The comma and period keys ("<" and ">" with Shift key) will shift from the keyboard.

Denoise (9): Removes noise from an image by scanning it with a 3 x 3 pixel window and replacing the center pixel with the median value pixel in the window.

Snap (N): grabs a snapshot of the current image and stashes it in a buffer.

Unsnap (U): replaces the current image with the snapshot. The snapshot is unchanged.

Munge (M): composites the currently displayed image with the snapshot using the undo buffer as a mask. When the largest absolute difference between a pixel in the image and the same pixel in the undo buffer is greater than the threshold set by the Munge Threhold slider, a pixel from the snapshot will be written to the image. The undo buffer and the snapshot will be resized to the image dimensions if necessary. The Munge command will scale the snapshot buffer to the same dimensions as the display. When the display is rotated, the undo buffer is also rotated, but the snapshot buffer is not. This can lead to some interesting munging, such as compositing an image with itself, rotated 180 degrees, etc.

Invert Munge (I): when checked, inverts munging, so that a pixel gets written from the snapshot when the difference between pixels in the display and undo buffers is less than the Munge Threshold.

Example: Munging

Munging is a glitchy compositing technique that uses the display image, the snapshot buffer and the undo buffer. Each buffer can hold a different image. As explained in the command reference, munge composites the currently displayed image with the snapshot using the undo buffer as a mask. In the images below, "/" connotes a munge operation, with the first element in the display buffer and the second one in the snapshot buffer.

Original (undo or display)

Glitched (snapshot)

Color quantized (display or undo)

Color quantized/Glitched

Original/Glitched

Inverted munge, quantized/Glitched

Munge with degrade, Glitched/Original (Original in snapshot, Glitched image in display)

Linear sorting, Original/Snapshot; the sorting steps were set to 5. The key sequence 'TGTTTM' (turn, sort, turn, turn, turn, munge) was repeated twice.

Zigzag sorting, HSB Shellsort, rotated Original/Snapshot

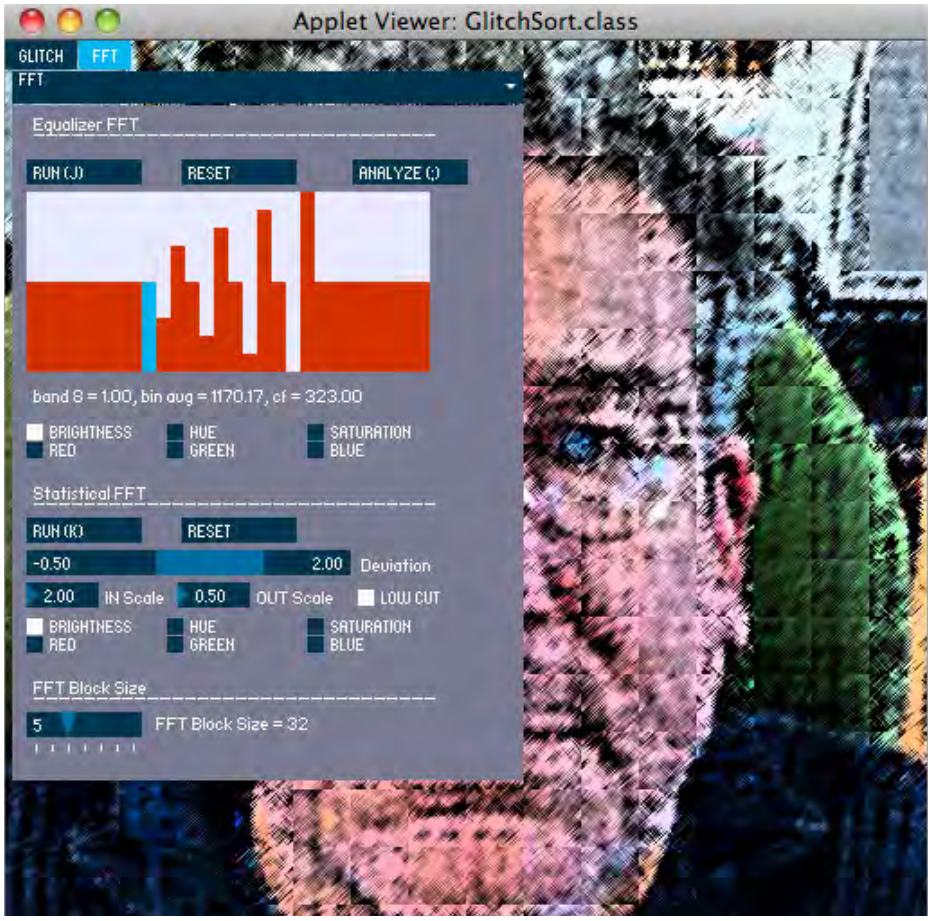


Figure 3: The FFT Control Panel for GlitchSort, created with ControlP5. FFT block size is 32, for a 1024-pixel array and 28 frequency bands. Information is displayed for band 8 in the equalizer interface (the cursor hovering over the slider is not shown).

The FFT Control Panel

The FFT control panel sets the parameters for a Fast Fourier Transform (FFT) executed on zigzag-scanned blocks of pixels. The blocks may be 8, 16, 32, 64, 128, 256, or 512 pixels square, and will be centered in the image. The panel provides one interface similar to an audio equalizer and another that performs a statistical evaluation of each block, plus a control for setting the current block size.

The Fast Fourier Transform is frequently used with time domain data such as audio signals. It transforms samples within a window in the time domain into amplitudes (and phases) in the frequency domain, with no loss of information. In the frequency domain, individual frequencies or bands of frequencies can be altered, and then moved back into the time domain, as a signal. In GlitchSort the signal consists of pixel values and the window is an array produced by zigzag-scanning a block of pixels. You can think of low frequencies as corresponding to large scale variations in pixel values, and high frequencies as corresponding to small scale variations—in other words, fine detail. The number of frequency bins produced by the FFT is half the number of samples plus 1. For an 8 x 8 block, that is 64/2 + 1 or 33 bins. For larger blocks, there can be thousands of bins, so it makes sense to aggregate them in some way. GlitchSort aggregates frequencies with an equalizer and with statistical analysis.

FFT Equalizer Interface

For audio signals, an equalizer typically groups frequency bins into bands consisting of octaves or portions of an octave (an octave is the span from a frequency to double the frequency). Depending on the size of the zigzag block, GlitchSort provides 17 to 33 bands. The default setting is for a 64 x 64 pixel block, with 32 bands. The JPEG algorithm uses a process similar to the FFT called the Discrete Cosine Transform (DCT), that can cause characteristic artifacts in highly compressed files or files where the JPEG header has been databent. The FFT in GlitchSort also creates a wide range of artifacts.

The equalizer arranges low frequency bands to the left and high frequency bands to the right. Each band can be dragged up or down to set its value. Initial values are set to 1.0. The maximum value is 2.0 and the minimum is 0. When you run the FFT, all the frequencies in the band will be multiplied by the value set for the band and constrained to the interval 0–255 used for pixel channel values. The first band will affect the entire zigzag block (this is handy for operations such as turning the whole image blue, or brightening or darkening it uniformly). Successive bands will affect smaller elements of the block, creating glitchy artifacts as the FFT is repeatedly applied. The bands on the far right will create finely detailed artifacts.

When you roll over individual sliders in the equalizer interface, GlitchSort posts some information about the band: the scaling factor set by the slider, the average value (amplitude) of the Brightness channel of all the frequency bins in the selected band over the whole image, and the center frequency of the band. These values will vary according to the number of samples in the FFT (determined by the size of the zigzag-scanned block). The scaling factor is the probably the most useful information. The amplitude values are refreshed when you click the Reset button or the Analyze button or when you load a new file. The Analyze button will also output current amplitudes to the console.

FFT Statistical Interface

Unlike the equalizer interface, which has a different set of frequency bands depending on the FFT block sizes, the statistical interface provides a way of scaling the amplitudes of color channels that is relatively independent of the blocksize. The interface provides a range slider, a check box, and two number boxes. Frequencies whose amplitude lies between the left and right bounds will be scaled by the IN Scale value. Frequencies whose amplitude lies outside the bounds will be scaled by the OUT Scale value. When the Low Cut checkbox is checked, the first bin, which has much greater amplitude than the others (think of it as representing global illumination) will not be used in calculating the mean amplitude values and the results will tend to be noisier but more fine-grained. The default settings in effect reinforce the values of many mid-range frequencies that are well-represented in the image, and cut most that are not, along with cutting some of the well-represented low range frequencies, providing a kind of noisy sharpening. Swapping the default IN and OUT scale values will cut well represented frequencies and boost poorly represented ones, resulting in a sort of blurring.

The Statistical Interface is still somewhat of an experiment. I have arrived at the default values empirically: they increase contrast, but in a glitchy way (what else were you expecting?). The values for the two bounds are not equal because the distribution for most images is evidently skewed. The bin averages output by the equalizer interface bear this supposition out, as does the statistical analysis of most images. Things may change in future development, particularly with respect to using standard deviations to measure the bounds—for now, it suffices to get interesting results.

Both interfaces provide a set of channel checkboxes, for the HSB and RGB color spaces. Any time the FFT is run from either interface, it processes each channel individually. Running multiple channels obviously will take longer, but can produce interesting results. The channels execute in order: H, S, B, R, G, B. The statistical interface will output useful information about the image to the console each time it executes: the average minimum, maximum, mean, medium, standard deviation, and skew for all the blocks in the image, and the average left edge and right edge of the range, expressed in units of amplitude.

FFT Block Size

The FFT Block Size slider changes the dimensions of the zigzag-scanned block of pixels that is passed in an array to the Fast Fourier Transform. For an FFT, array sizes must be powers of 2. GlitchSort currently supports block sizes of 8, 16, 32, 64, 128, 256, and 512 pixels on a side. If your image is smaller than the block size, the FFT will have no effect.

FFT Interface Commands

Run (J): saves current image to undo buffer and runs FFT over the image using the Equalizer Interface.

Run (K): saves current image to undo buffer and runs FFT over the image using the Statistical Interface..

Reset: resets either interface to its default settings.

Hue, Saturation, Brightness, Red, Green, Blue: select the color channels to process.

Analyze (;): calculates average values in each band of the equalizer, outputs the values to the console.

Sliders (equalizer): set the scaling factor for all the frequency bins aggregated by each band. *Tip:* press the mouse button down just outside the equalizer and then drag over it to set multiple sliders at once.

Deviation: a range slider where the 0 (zero) position represents the mean amplitude of the frequencies in each block. Negative values correspond to frequencies with less amplitude than the mean, positive values to frequencies with an amplitude greater than or equal to the mean. Frequencies whose amplitudes lie between the two sliders are scaled by the IN scale value. Frequencies whose amplitudes lie outside the two sliders are scaled by the OUT scale value. Frequencies above the mean tend to bewell-represented in the image. Those below the mean are less well-represented, and could in theory be discarded without overly affecting the image: most lossy compression codecs, including JPEG, use a similar strategy.

In Scale: value to scale frequencies whose amplitudes lie inside the selected range.

In Scale: value to scale frequencies whose amplitudes lie outside the selected range.

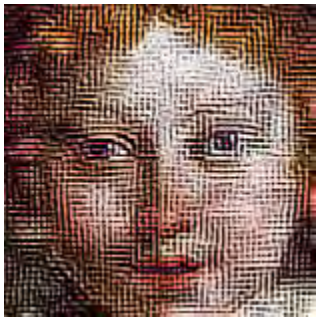
Low cut: when set, omits the first frequency bin from statistical calculations. In any case, the first bin is never scaled—you can do that with the equalizer interface.

FFT Block Size: a slider, marked off in powers of 2, that sets the dimensions of the zigzag-scanned square block of pixels that is fed to the FFT. Currently supports 8, 16, 32, 64, 128, 256, or 512 pixel square blocks.

FFT Cookbook: Equalizer Interface

Here are some examples of what the FFT tools can do. In most of these examples the FFT is applied (K or J key) then the image is rotated 90° (T) and the FFT is applied again. This performs the zigzag scans in two different directions, so that the FFT artifacts are less markedly oriented along diagonal lines.

Original image, 256 x 256 px.

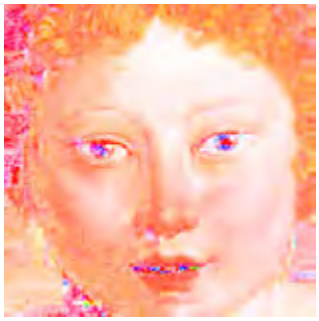


eQ bands 28–29 set to 2.0
FFT block size = 128

eQ bands 27–28 set to 2.0
FFT block size = 64

eQ bands 23–24 set to 2.
FFt block size = 32

These three images look very similar because all three were processed in bands representing approximately the same frequencies, centered on 8.2KHz and 10.1KHz. The FFT was run 6 times on each image, with a 90° turn in between (i.e., TJ was executed 6 times).



Band 0 = 2.0, FFT run 3x on Brightness, leaving only Hue and Saturation data intact.

Band 0 = 2.0, TJ 4x on Saturation, leaving only Brightness data.

Band 0 = 2.0, TJ 4x on Blue. RGB in various combinations can colorize an image.

Band 14 = 2.0, bands 22–24 = 0, TJ 8x on Saturation, Red, Green.

Band 0 = 1, bands 1–15 = 0, bands 16–31 = 2, TJ 2x. This emphasizes high frequencies.

Bands 24–31 = 0, TJ 4x. This removes high frequencies, in effect blurring the image.

Band 1 = 2, TJ 4x on Hue, then J once on Saturation, FFT block size = 16.

Bands 24, 27, 30 = 2
bands 25, 28, 31 = 0
TJ 4x on Brightness and Green
FFT Block Size = 256 (the whole image)

FFT Cookbook: Statisitcal Interface

Original image, 256 x 256 px.

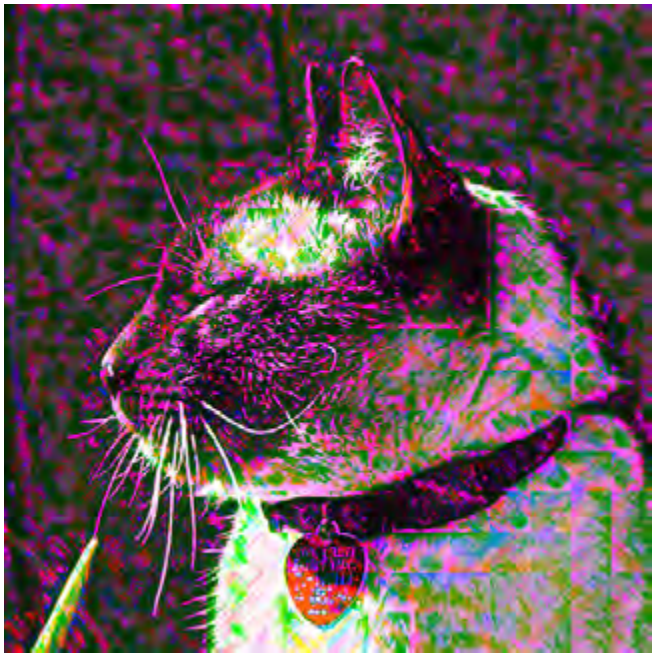
KTK, default settings

KTK, default, with low cut

KTK, IN = 0.5, OUT = 2.00

Original image, 512 x 512 px.

Original image, 512 x 512 px. Range -2.0–2.0, TK 4x on Blue, TK 2x on Blue and Brightness.



TK 4x on Red and Blue channels with range -2.0–2.0 and low cut turned off. TJ once with band 0 = 1.2 on Saturation. TK once more on Saturation, to brighten the image. Just one example of how complex the FFT processes can be.

Image processed multiple times with default statistical FFT settings, rotated between applications. 256 x 256 pixels.

The same image after multiple passes of the denoise filter, rotated between applications.

Audify Command

The Audify command was new in GlitchSort 1.0b7 pre-release c. It provided rudimentary conversion of an image into an audio signal, with some modest methods for creating audio signals an image. It allowed me to use GlitchSort as a real-time performance instrument at the GLi.TC/H festival in December 2012, in Chicago. It was buggy. It is still rudimentary and kludgey, but I think I've fixed the bugs.

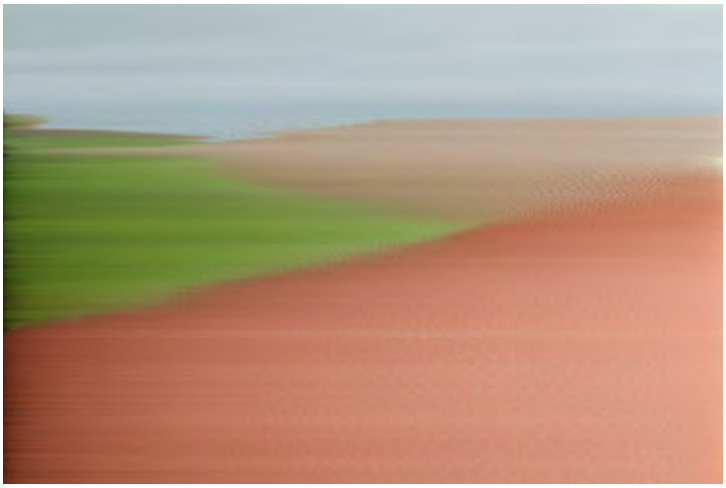
To turn on the Audify command, press the backslash key (/). To turn it off, press forward slash (\). Once Audify starts running, it will read 64 pixel square blocks from the image with a zigzag scan and treat their brightness component as an audio sample. This tends to be very noisy, but this is a glitching application, so you'd probably be disappointed if it wasn't. Pixel blocks of a uniform brightness will be silent. You can run all GlitchSort commands and they will change the audio output. Try repeated zigzag sorting (L).

While Audify is running, further presses of the backslash key will either sort the current pixel block, or run a statistical FFT or equalizer FFT operation on it. Other blocks are not affected. The choice of operation depends on the most recent command you executed: if it was one of the FFT operations, that will be selected, otherwise, you'll get a zigzag sort. You can use the equalizer interface in particular to embed audio signals in an image that you save. For example, in the image below (included with this release) I have filtered pixel blocks with many different equalizer settings, including approximations of the vowel sounds of English (over the G).

The Audify command uses the Minim audio library included with Processing. The bundled version of Minim has a number of limitations, which suggests that future development of Audify may rely on a different version of Minim, on a different library, or on the use of Open Sound Control to communicate with a dedicated audio processing applications such as PD. For now, I offer it as a simple and noisy way use GlitchSort to generate audio output for real time performance.

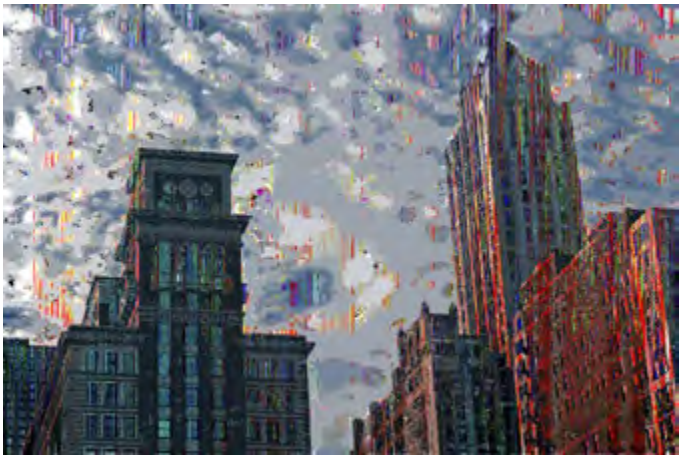
GlitchSort Gallery

Examples from my own work, from 2012, showing the possibilities of GlitchSort, all by itself.



Red Dirt Road, North Dakota

Reclining Nude with Atom Bomb



Downtown Sky, Chicago

Short Cut Keys

spacebar to show or hide the control panel
option-drag on control panel bar to move control panel
shift-drag on image to pan large image (no shift key needed if control panel is hidden)
f toggle fit image to screen
o open a file
s save a timestamped .png file
r revert to the current file
t turn the image 90 clockwise
g sort the pixels (glitch)
l sort in zigzag-scanned blocks
z undo the last action
1 select quick sort
2 select shell sort
3 select bubble sort
4 select insert sort
a change sort order to ascending or descending
b toggle random breaks in sorters
x toggle color channel swapping (glitchy!)
c step through the color channels swaps
+ or - step through color component orderings used for sorting
y turn glitch cycling on and off (for glitch steps > 1)
[or] to decrease or increase glitch steps
{ or } to cycle through Shell sort settings
d degrade the image with low quality JPEG compression
UP or DOWN arrow keys to change degrade quality
p reduce (quantize) the color palette of the image
LEFT or RIGHT arrow keys to change color quantization
< or , shift selected color channel one pixel left
> or . shift selected color channel one pixel right
9 denoise image with a median filter
n grab a snapshot of the current image
u load the most recent snapshot
m munge the current image with the most recent snapshot and the undo buffer
k apply statistical FFT
j apply equalizer FFT
/ turn audify on and execute commands on a single block of pixels
\ turn audify off
v turn verbose output on and off
h show help message

Hacking GlitchSort2

GlitchSort2 is open source software. Feel free to edit it. Among other hidden features, it has the beginnings of a "journaling" or command record and playback function. Enter '_' (underline) for an example of a command that executes the last command and turns the image 90°, repeated 4 times. See the commandSequence and exec methods for the code.

Image Credits

"404 Not Found" image from http://ppc-advice.com/wp-content/uploads/2008/06/404.jpg. Its copyright status is unknown, but hey, LOLcats are free aren't they?

The "Blobbyland" image in the Component Sorting Order examples is my own, created in Processing with my IgnoCodeLib, a Processing library that provides a vector graphics display list for Bézier curves and exports to Adobe Illustrator. Check it out at http://paulhertz.net/ignocodelib/.

Many thanks to Curt Cloninger, aka Play Damage, for giving me permission to use his image in some of the examples.

The image on the cover is "The Wanderer in the Glitch," my version of Caspar David Friedrich's famous painting, "The Wanderer above the Sea of Mist," http://commons.wikimedia.org/wiki/File:Caspar_David_Friedrich_032.jpg. The font on the cover is Dataface (http://openfontlibrary.org/font/dataface) by Antonio Roberts, who has written a tutorial how to glitch a font: http://www.hellocatfood.com/2010/07/16/create-your-own-glitch-typeface/.

Licensing and Updates

You can download GlitchSort2 and this manual from http://paulhertz.net/factory/2012/08/glitchsort2/. There is also information about GlitchSort2 on the site for the course I teach at the School of the Art Institute, "Code Sourcery," at http://paulhertz.net/saic/algoprac/glitch.html. I also post updates to the G L I T C H group on FB, https://www.facebook.com/groups/glitchglitch/, where some artists who have tried out post and where your questions _may_ get answers.

My own images created with GlitchSort2 can be found at http://www.flickr.com/photos/ignotus/sets/72157629445337238/.