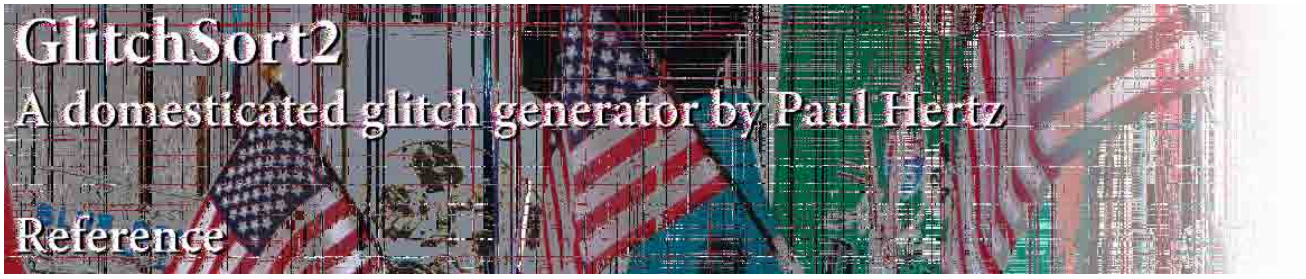
Glitch Art Reference



GlitchSort2 is a Processing application that uses broken pixel-sorting to create glitchy images. In version 1.0b4, released on August 1, 2012, there are four different sorting algorithms, each of which has a different behavior that can be used to damage images in different ways.

GlitchSort2 sorts bit-mapped images row by row. The sorting is randomly interrupted, with a probability determined by the breakpoint setting, resulting in a partially sorted rows of pixels. It is also possible to perform a complete sort of the rows of pixels (just uncheck the *break* checkbox). This can be an interesting way of analyzing the color of an image. The complete sort is only available for the quick sort and shell sort algorithms. The other algorithms are simply too slow for there to be any reason to use them for a complete sort, but they have other uses.

The Algorithms

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. It is also 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.

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

The following commands are found in the ControlP5 Control panel for GlitchSort2. 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.

Sort (G): sorts the pixels horizontally row by row. If the break checkbox is checked, interrupts the sort to create glitches. If the break checkbox is unchecked (only possible with Quicksort and Shellsort), does a complete sort of each row. The breakpoint and steps settings (and the cycle checkbox) affect sorting.

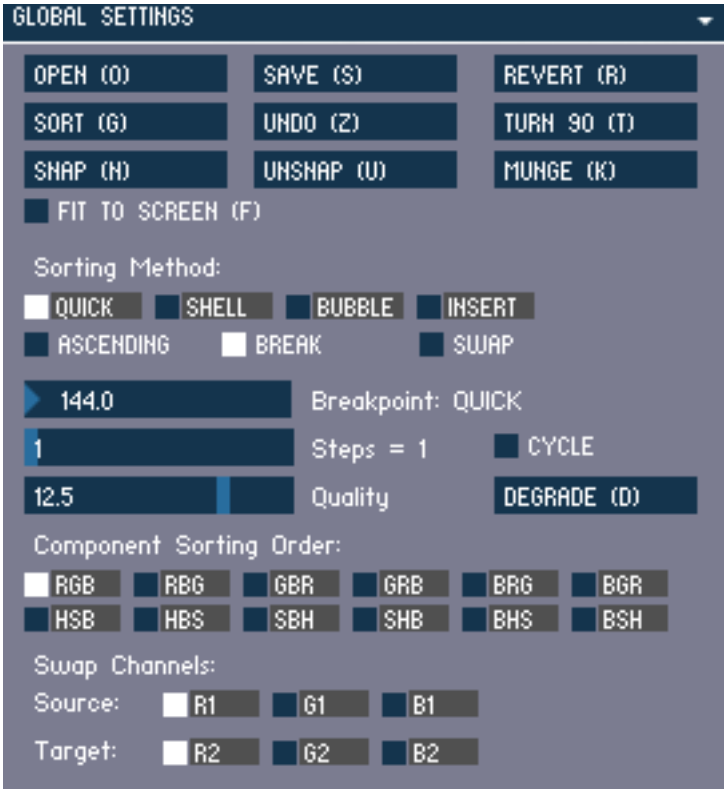


Figure 2: The Control Panel for GlitchSort2, created with ControlP5.

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."

Snap (N): captures the currently displayed image into the "snapshot" buffer.

Unsnap (U): replaces the display image with the image previously captured into the snapshot buffer.

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 mungeThreshold, 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. Use for general weirdness.

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

Quick (1), Shell (2), Bubble (3), Insert (4): selection of sorting method.

Ascending (A): when this option is checked, the current sorting method will sort pixel values in ascending order. For 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 apparent 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 (UP and DOWN arrow keys): 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.

Quality: this slider sets JPEG quality for the degrade operation. Lower values cause more degradation of the image. Really low values pretty much obliterate the image with blocky artifacts.

Degrade (D): degrades an image by saving it as a JPEG with low quality compression and then loading the saved file (a Revert command will still restore the original file).

Component Sorting Order

By default, numerical value of a pixel represents four color channels, red, green, blue and alpha, in that order. When the pixels are sorted, the numerical value of the ordered 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. This may be easier to see than to explain. Take a look at the examples section of this reference.

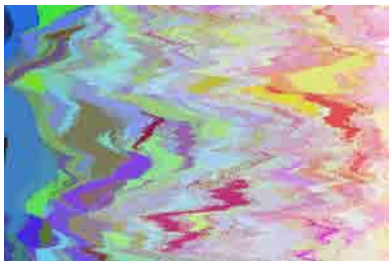
Swap Channels

The source and target in swap channels refer to different locations within the pixel array. When the values in two different locations are exchanged, normally, all channels are exchanged. 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. This in turn determines which colors will appear in the sorted pixels. Again, pictures may explain this better than words can: see the example.

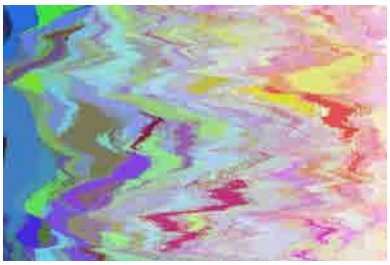
Example: Component Sorting Order

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

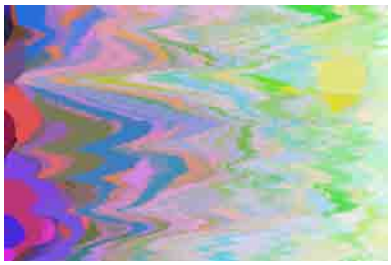
Original image



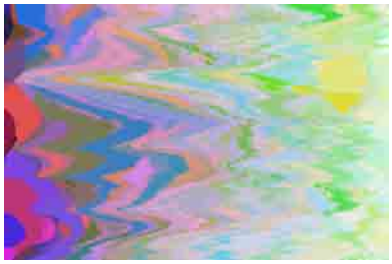
RGB



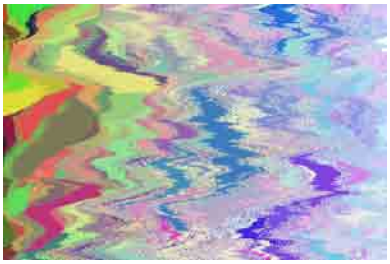
RBG



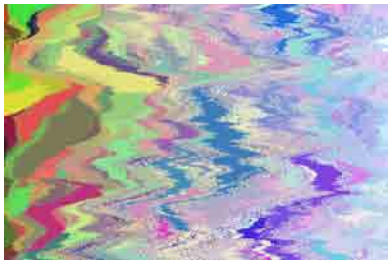
GBR



GRB



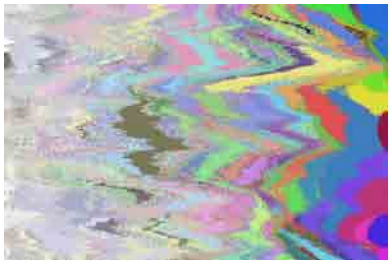
BRG



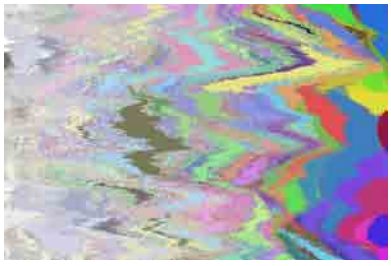
BGR

HSB

HBS

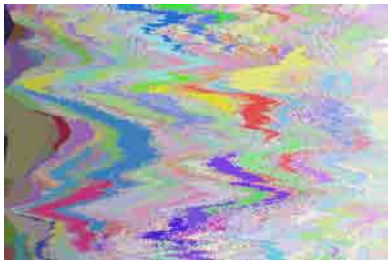


SBH



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

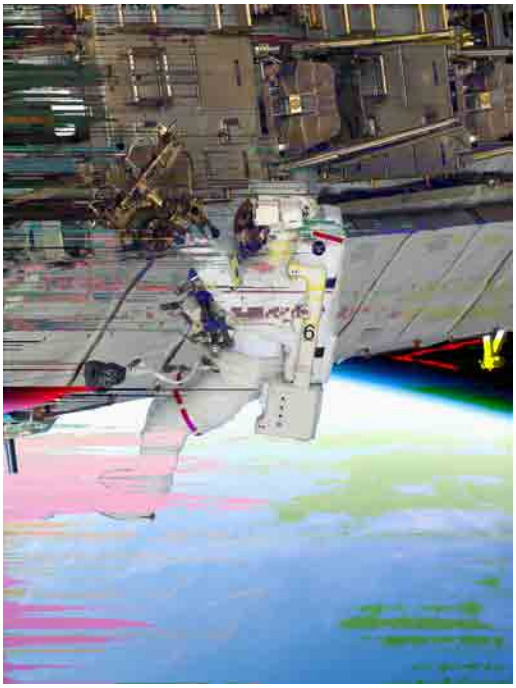
BR

BG

BB

Example: Degrading an Image

Both images have been sorted with single pass of an interrupted Quicksort (breakpoint = 411), component sorting order SBH, channel swaps RB, in ascending sort order. The second image was degraded (quality = 12.5) before the sort. Note how the degrading resulted in blocky pixels. Interrupted sorting, particularly with Quicksort or Shellsort, will reveal the compression artifacts in JPEG images. In that sense, it is a kind of analytic tool. JPEG compression works, in part, by reducing the saturation or hue variation in an image. The HSB/HBS and SBH/SHB component sorting orders are particularly useful for revealing JPEG artifacts, even in images where they are scarcely visible (as the JPEG consortium intended). At low quality settings (below 10.0) degrading will produce very blocky images.



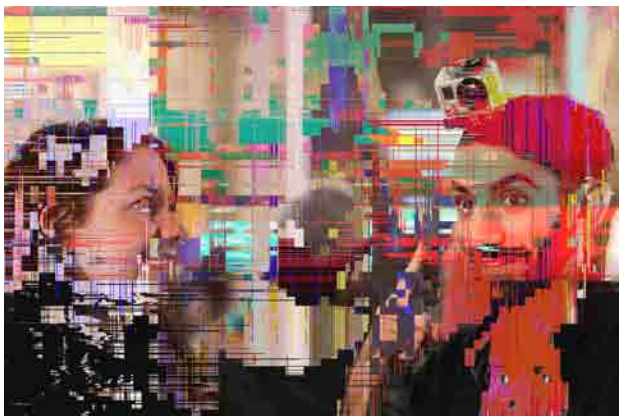
Without degrading

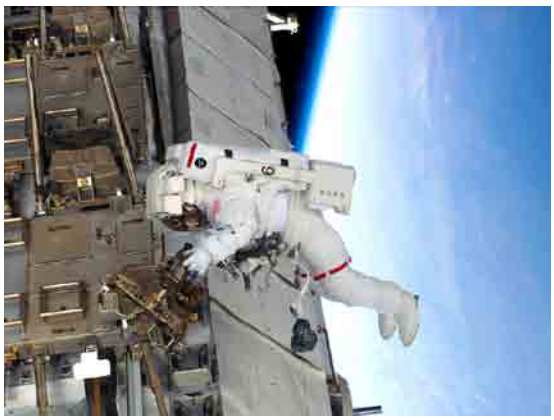
Degrading at 12.5 quality

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. When the largest absolute difference between a pixel in the image and the same pixel in the undo buffer is greater than mungeThreshold, 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 technique is meant to be unpredictable, though with some experimentation you can probably figure out uses for it.

Here is a formula I have found useful: Load in image into GlitchSort2. Alter it however you like and take a snapshot (N) (top left image). Then load a different image (top right). Run a single operation on it that creates a large difference, such as a complete Quicksort on half the rows. You can accomplish this by unchecking the Break checkbox and setting Steps to 2, then sorting (G)—use the RGB component order for fast sorting (bottom left). The original image is now in the undo buffer—type 'Z' to show it by swapping the undo buffer with the display buffer. At this point you could also load a third, new image, preferably one with some areas identical to image in the undo buffer. Now Munge (M): the image in the snapshot will appear in the glitched image wherever it is different (exceeds the munge threshold) from the original (bottom right).





Hacking GlitchSort2

GlitchSort2 is open source software. Feel free to edit it. Here some simple things to try:

Change the mungeThreshold value (default is 16) to alter the way compositing works with the Munge command.

In the ShellSorter, change the values of ratio and divisor to get different intervals of repetition in the interrupted Shell sort.

In the initialization methods for QuickSorter, ShellSorter, BubbleSorter and InsertSorter, change the default value of the breakpoint (handy to avoid having to use the control panel input).

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/.

The black and white image of a throng of women is in the public domain and can be found at http://commons.wikimedia.org/wiki/File:Throng_of_women_charge_on_New_York_city_hall_to_demand_bread.png.

The glitched image of two glitch artists/theorists, one with a webcam on his head, is my own, shot at the art gallery opening for GLI.TC/H 20111, in Chicago. See more at http://www.flickr.com/photos/ignotus/6335541994/in/pool-1502527@N24/.

The image of a spacewalking astronaut fixing an instrument on the International Space Station is from NASA, apparently in the public domain, as are most images financed by taxpayers. You can find a small version of it at http://spacestationlive.jsc.nasa.gov/resources/actcat.html.

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/.