# *JSlush_readme.pdf*

Base Image


JSlush output using Displace, Diffuser & HSV Compositor

*Contents*



JSlush created by Minë Poodle, www.satanic-ceramics.com, 2026.

# tl;dr

JSlush is a node-based .jpg manipulator.

Use splitters, math and transform tools to remix images.

Everything plugs into everything else.

It runs in your browser and nothing is uploaded.

You can apply the same effects to a different image using the “Select File” button at any time.

Right-click the displayed image to “Save Image As”.

# Controls

Left click and drag pans the nodegraph. You can zoom in and out.

Left click and drag from node outputs to node inputs to connect strings to pass the data.

Hold **S** for STRING mode and click a node output or input to delete a string.

Hold **S** and click the node to remove all strings from that node.

Hold **V** for VIEW and left click a node to view the result. You can then save the image.

If you view a float, you will see a greyscale image. For nodes with multiple outputs, click to cycle through each in sequence.

Hold **D** for DELETE and click a node to delete it.

Hold **T** and click to TIDY nodes.

*Warning:*

Using DCT and Diffuser nodes with high-res images and in complex nodegraphs may cause browser freezes while they process the image. This is because they have more complex math than other nodes. To find out how they work, see the Effects and Math sections.

# Basic tutorial

Add one Effects node and one Splitter node (from the Tool section). Connect the Source to the Image inputs on both.

Now, you have an Effects node that will change an image based on a float value.

Hold V and click the Splitter repeatedly to see what the float value versions of the image looks like. Every pixel is showing a 0 to 1 value.

Take an output of the Splitter node and plug it into the Effects node. Then, hold V and click the Effects node to see the results.

Now, add a Math node. Take an output of the Splitter node and add it to the Math node to change it, and use V to click the Math node to view the changes.

If the Math node needs more float inputs, you can use other outputs of the Splitter, or add another Splitter from the Source.

(You can't create a loop of a Splitter from the Effects node to change the Effects node - It won't work).

When you are happy with how you have changed the float value from the Splitter using the Math node, replace the connection between the Splitter and the Effects node with a connection from the Math node to the Effects node. Delete the string using S and Left click, then drag a new string.

Use V and click the Effects node again to see the changes you have created by changing the float input.

Congratulations, you're slushin'!

# About JSlush

Enjoy parellel process image manipulation using the JSlush nodegraph and color conversion engine. You can manipulate different image data individually and add them to each other with merge tools.

JSlush passes only two types of data between nodes: an image array, or a float (0-1) value. Red, Green, Blue, Hue, Saturation, Luma, etc. can be used to interchangeable to determine effects.

JSlush intentionally lacks adjustable parameters or generative algorithms. Instead, users are encouraged to have fun combining lossy compression, reformatting and composition.

JSlush is provided 'as is' and Satanic Ceramics LLC is not responsible for how, when or why you use JSlush or any attendant consequences, legal or otherwise.

# Tips, Tricks and Reminders

JSlush can be used in different styles and is best used creatively by problem solving.

Image input and output nodes can be chained end-to-end to create a cascading sequence of effects.

Different float inputs can shape the Effects at each stage.

Different destructive effects can be applied to the same source image simultaneously so you can experiment with Splitters to find which float variation works the best with each Effect.

If you like an image Effect output but aren't totally satisfied with it, you can split it into float channels and re-use only part of it, or mix in another variation.

Every image displayed in the view window is a different .jpg to the source image. (Image output nodes are simply the interface point).

If your computer can't handle the amount of nodes you have added, or the image is refreshing too slowly, you can simply save the image and then start fresh by selecting your saved image with "Select File."

Destructively rearranged .jpgs can have the the original source image rewritten into or superimposed onto the result by using the Overwrite and Overlay tools.

Quantized inputs can create blocky outputs, smooth gradients can create flowing shapes.

There is no limit to nodes and nothing needs to make sense.

# Effects

*Pixel Sort*

Organizes pixels from the input image the X or Y axis according to a float value.

*Channel Shift*

Displaces RGB values of the input image according to the float value.

*Chrominance Offset*

Converts the image to YCbCr, then displaces Chrominance (CbCr) according to float value, then converts back to an RGB image.

*Displacement*

Displaces the pixels of the input image along the X or Y axis according to float value.

*Scanline Displacement*

Takes the average value of the float input row or column, then shifts that row or column of the input image cyclically.

*Morph Erode & Dilate*

The pixel takes color of the darkest or lightest color in its neighborhood. The float determines the search radius. Produces destructive painterly effects with flat color ‘brushstrokes’ of local min. and max.

*Slitscan*

Each pixel ‘scans the slit’ according to the float: it samples the source image at the column (X) or row (Y) at a position determined by the float value. Continuous float fields reposition areas coherently; fragmented or noise floats produce discontinuous color cuts.

*DCT & Quantization*

Applies .jpg compression according to the float value.

Creates an 8x8 tile map and applies Discrete Cosine Transform, then uses the mean float value per tile to drive quantization. Compression traverses through the frequency coefficients in JPEG zigzag order; fine detail first, broad structure last.

Unlike standard .jpg compression which applies compression uniformly, this is a unique elaboration on the design of the file format that uses the zigzag ordering, designed for compression efficiency, as a progressive gate to determine the level of data loss.

This effect creates an unpredictable tile map where different chroma and detail frequencies are quantized on a per-tile basis. A curve parameter shapes the float response.

*DCT Chroma Offset*

Creates an 8x8 tile map and applies DCT, then uses the float value to offset the CbCr values along the X or Y axis. Similar to Chrominance Offset, but by using the DCT 8x8 tiling system, it produces a tiled result instead of per-pixel.

*Modulated Bit Crush*

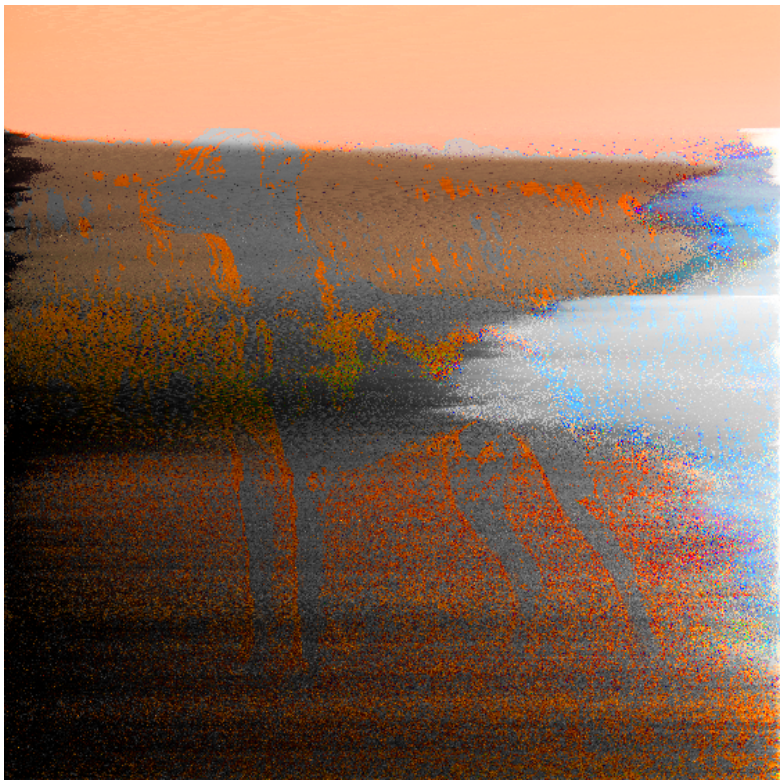
Posterizes the image according to the float value, but is not a true “bit crush” as it does not quantize to $2^n$ levels. The float value determines the number of levels according to a curve (max 255), then the color value is snapped to the nearest.

*Solarize (Luma)*

Emulates photographic solarization according to the float, or defaults to a 0.5 threshold. The luma of the image pixel is calculated according to Rec. 709 weights, then RGB inverted to mimic the Sabattier effect.

*Invert RGB*

Inverts the RGB.



JSlush output using PixelSortX and Luma Overwrite

# Data Tools

*RGB Splitter*

Outputs the RGB channels as discrete float values.

*YCbCr Splitter*

Converts the image to the YCbCr color space, then outputs the channels as discrete float values.

*Perceptual Luma*

Calculates the Luma at the ITU-R BT.709 coefficient set (standard for HD video and sRGB displays) and outputs it as a discrete float value.

# Compositors

*RGB Compositor*

Decodes the input float values as RGB value to build a composite image.

*HSV Compositor*

Decodes the input float values as HSV values to build a composite image.

*YCbrCr Compositor*

Decodes the input float values as YCbCr value to build a composite image.

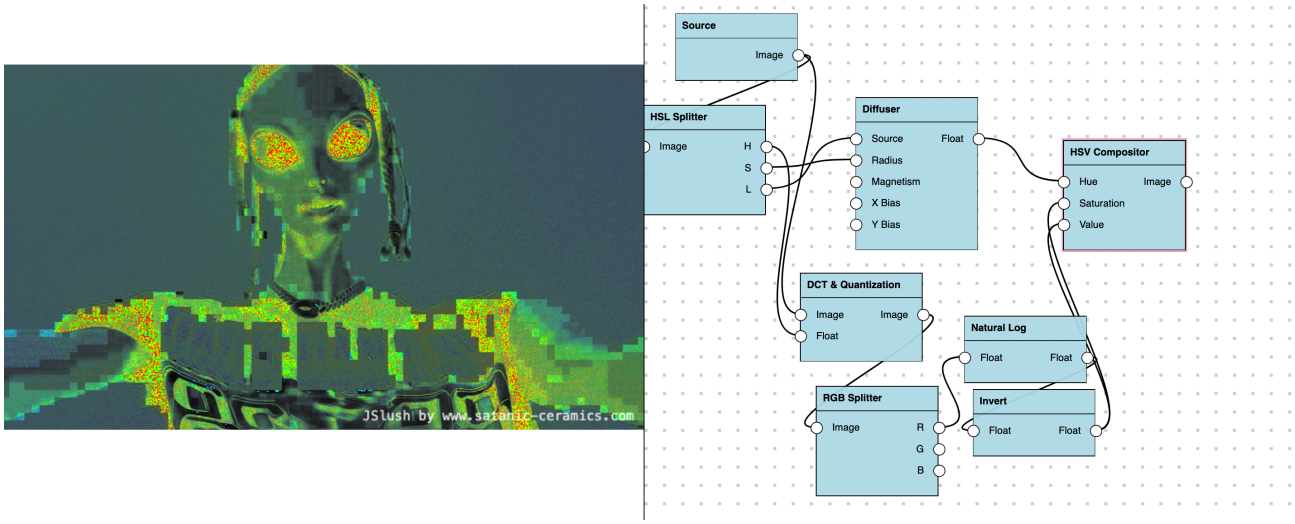
*Overwrite Channel*

Reads the origin of a float value (i.e. Luma, RGB, Hue), then overwrites that value in the input image with the float.

For example, split an image into RGB, modify only the green using Math, then using Overwrite, you can overwrite the Green values in the original image.

*Overlay*

The second image (input 2) is overlaid on the first, with the alpha channel (transparency) determined by the float.




Composite Image Example (base image: Dale Dan Tony)

# Math Functions

*Invert*

Inverts the float value.

*Difference*

Outputs the difference between two input float levels.

*Diffuser*

Creates a Gaussian blur of a float input. The Source's values are scattered on a per-pixel basis and overwrite where they land. For radius, dark inputs increase the per-pixel scatter radius and increases the scatter count to compensate for the larger area. Magnetism weights the gaussian curve so that light areas attract diffused pixels, and darker areas repel them; X and Y bias compress the gaussian curve on these axis.

*Fibonacci Levels*

Uses the Fibonacci retracement levels favoured by market traders to determine float thresholds. Values do not snap to the nearest threshold, once a value exceeds a threshold, it snaps to the one above.

*Feigenbaum*

Multiplies the float value by Feigenbaum's constant, then discards the integer and keeps the remainder, producing a 0-1 value. Produces chaotic bifurcation.

*Stepped Gradient*

Quantizes a gradient into a number of steps determined by the float, between 1-100. Defaults to 5 if no float is connected.

*Natural Log*

The float value matches a corresponding value on the e curve scaled between 0 and 1, raising midtones.

*Cantor Set*

Cantor construction to 12 iterations, the smallest Float32 precision allows.

*Modulo*

The float value is multiplied by the number of repetitions (0-100), then the integer discarded. Gradients produce graduated striping at the integer fold.

*Sobel*

Edge detection via gradient magnitude of 3x3 matricing. Normalises variation and produces relatively clean lines.

*Bayer*

Ordered dithering compares the float value against a 4x4 Bayer matrix to determine a binary (black or white) output. This is notable for being the one node that produces no greyscale float values. Newspaper or Manga comic look.


“Extreme Dog Grooming” Stegosaurus Poodle with Bayer effect

# Input & Output

*Sauce and Outputz*

Output is redundant as you can simply right click the image to “Save Image As”.

*Secret*

Advanced users can create their own nodes, add them to the /nodes folder and edit the index.html to ensure they are run.

Thanks for using JSlush!

www.satanic-ceramics.com