lvsfunc

Release 0.3.5

LightArrowsEXE

May 02, 2022

CONTENTS:

CHAPTER

ONE

ABOUT

lvsfunc, a collection of VapourSynth functions and wrappers written and/or modified by LightArrowsEXE.

If you spot any issues, please do not hesitate to send in a Pull Request or reach out to me on Discord (LightArrowsEXE#0476)!

CHAPTER

TWO

DEPENDENCIES

lvsfunc depends on the following third-party scripts:

- havsfunc
- kagefunc
- vsutil

The following VapourSynth libraries are also required for full functionality:

- combmask
- d2vsource
- dgdecnv
- ffms2
- KNLMeansCL
- L-SMASH-Works
- RGSF
- VapourSynth-Bilateral
- VapourSynth-BM3D
- VapourSynth-descale
- VapourSynth-EEDI3
- VapourSynth-nnedi3
- VapourSynth-NNEDI3CL
- VapourSynth-ReadMpls
- VapourSynth-Retinex
- vs-ContinuityFixer
- zimg
- znedi3

This list is non-exhaustive, as dependencies may have their own dependencies. An attempt has been made to document major dependencies on a per-function basis. Unfortunately, *func family modules have complex dependency graphs and documenting them is beyond the scope of this module.

CHAPTER

THREE

MODULES

lvsfunc.aa	Functions for various anti-aliasing functions and wrappers.
lvsfunc.comparison	Functions intended to be used to make comparisons between different sources or to be used to analyze something from a single clip.
lvsfunc.dehalo	Wrappers for dehaloing functions.
lvsfunc.dehardsub	Dehardsubbing helpers.
lvsfunc.deinterlace	Functions to help with deinterlacing or deinterlaced content.
lvsfunc.denoise	Denoising functions.
lvsfunc.kernels	Kernels for vapoursynth internal resizers.
lvsfunc.mask	Wrappers and masks for denoising.
lvsfunc.misc	Miscellaneous functions and wrappers that didn't really have a place in any other submodules.
lvsfunc.recon	Wrappers and functions for chroma reconstruction.
lvsfunc.render	Clip rendering helpers.
lvsfunc.scale	Functions for (de)scaling.
lvsfunc.types	
lvsfunc.util	Helper functions for the main functions in the script.

CHAPTER

FOUR

FUNCTIONS

lvsfunc.aa.clamp_aa(src, weak, strong[, ...])	Clamp stronger AAs to weaker AAs.
lvsfunc.aa.eedi3([opencl])	Generate eedi3 antialiaser.
lvsfunc.aa.kirsch_aa_mask(clip[, mthr])	Kirsh-based AA mask.
lvsfunc.aa.nnedi3([opencl])	Generate nnedi3 antialiaser.
lvsfunc.aa.nneedi3_clamp(clip[, strength, ...])	A function that clamps eedi3 to nnedi3 for the purpose of reducing eedi3 artifacts.
lvsfunc.aa.taa(clip, aafun)	Perform transpose AA.
lvsfunc.aa.transpose_aa(clip[, eedi3, rep])	Function that performs anti-aliasing over a clip by using nnedi3/eedi3 and transposing multiple times.
lvsfunc.aa.upscaled_sraa(clip[, rfactor, ...])	A function that performs a supersampled single-rate AA to deal with heavy aliasing and broken-up lineart.
lvsfunc.comparison.compare(clip_a, clip_b[, ...])	Allows for the same frames from two different clips to be compared by interleaving them into a single clip.
lvsfunc.comparison.diff(*clips[, thr, ...])	Creates a standard lvsfunc.comparison.Stack between frames from two clips that have differences.
lvsfunc.comparison.diff_hardsub_mask(a, b, ...)	Diff func for lvsfunc.comparison.diff() to use a hardsub mask.
lvsfunc.comparison.interleave(*clips, ...)	Small convenience function for interleaving clips.
lvsfunc.comparison.split(*clips, **named-clips)	Small convenience funciton for splitting clips along the x-axis and then stacking.
lvsfunc.comparison.stack_compare(*clips[, ...])	A simple wrapper that allows you to compare two clips by stacking them.
lvsfunc.comparison.stack_horizontal(*clips, ...)	Small convenience function for stacking clips horizontally.
lvsfunc.comparison.stack_planes(clip, /[, ...])	Stacks the planes of a clip.
lvsfunc.comparison.stack_vertical(*clips, ...)	Small convenience function for stacking clips vertically.
lvsfunc.comparison.tile(*clips, **named-clips)	Small convenience function for tiling clips in a rectangular pattern.
lvsfunc.dehalo.bidehalo(clip[, ref, sigmaS, ...])	A simple dehaloing function using bilateral and BM3D to remove bright haloing around edges.
lvsfunc.dehardsub.bounded_dehardsub(hrdsb, ...)	Apply a list of lvsfunc.dehardsub.HardsubSign
lvsfunc.dehardsub.get_all_masks(hrdsb, ref, ...)	Get a clip of lvsfunc.dehardsub.HardsubSign masks.
lvsfunc.dehardsub.hardsub_mask(hrdsb, ref[, ...])	Zastin's spatially-aware hardsub mask.

continues on next page

Table 1 – continued from previous page

lvsfunc.deinterlace.deblend(clip[, rep])	A simple function to fix deblending for interlaced video with an AABBA blending pattern, where A is a regular frame and B is a blended frame.
lvsfunc.deinterlace.decomb(clip, TFF[, ...])	A filter that performs relatively aggressive filtering to get rid of the combing on a interlaced/telecined source.
lvsfunc.deinterlace.dir_deshimmer(clip[, ...])	Directional deshimmering function.
lvsfunc.deinterlace.dir_unsharp(clip[, ...])	Diff'd directional unsharpening function.
lvsfunc.deinterlace.SIVTC(clip[, pattern, ...])	A very simple fieldmatching function.
lvsfunc.denoise.bm3d(clip[, sigma, radius, ...])	A wrapper function for the BM3D denoiser.
lvsfunc.mask.detail_mask(clip[, sigma, rad, ...])	A wrapper for creating a detail mask to be used during denoising and/or debanding.
lvsfunc.mask.halo_mask(clip[, rad, brz, ...])	A halo mask to catch basic haloing, inspired by the mask from FineDehalo.
lvsfunc.mask.range_mask(clip[, rad, radc])	Min/max mask with separate luma/chroma radii.
lvsfunc.misc.allow_variable([width, height, ...])	Decorator allowing a variable-res and/or variable-format clip to be passed to a function that otherwise would not be able to accept it.
lvsfunc.misc.chroma_injector(func)	Decorator allowing injection of reference chroma into a function which would normally only receive luma, such as an upscaler passed to *lvsfunc.scale.descale()*.
lvsfunc.misc.colored_clips(amount[, ...])	Returns a list of BlankClips with unique colors in sequential or random order.
lvsfunc.misc.edgefixer(clip[, left, right, ...])	A wrapper for ContinuityFixer (https://github.com/MonoS/VS-ContinuityFixer).
lvsfunc.misc.frames_since_bookmark(clip, ...)	Displays frames since last bookmark to create easily reusable scenefiltering.
lvsfunc.misc.get_matrix(clip)	Helper function to get the matrix for a clip.
lvsfunc.misc.limit_dark(clip, filtered[, ...])	Replaces frames in a clip with a filtered clip when the frame's darkness exceeds the threshold.
lvsfunc.misc.load_bookmarks(bookmark_path)	VSEdit bookmark loader.
lvsfunc.misc.replace_ranges(clip_a, clip_b, ...)	A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string.
lvsfunc.misc.scale_thresh(thresh, clip[, assume])	Scale binarization thresholds from float to int.
lvsfunc.misc.shift_tint(clip[, values])	A function for forcibly adding pixel values to a clip.
lvsfunc.misc.source(file[, ref, ...])	Generic clip import function.
lvsfunc.misc.wipe_row(clip[, secondary, ...])	Simple function to wipe a row with a blank clip.
lvsfunc.recon.chroma_reconstruct(clip[, ...])	A function to demangle messed-up chroma, like for example chroma that was downscaled using Nearest Neighbour, or the chroma found on DVDs.
lvsfunc.render.clip_async_render(clip[, ...])	Render a clip by requesting frames asynchronously using clip.get_frame_async, providing for callback with frame number and frame object.
lvsfunc.scale.descale(clip[, upscaler, ...])	A unified descaling function.
lvsfunc.scale.descale_detail_mask	Generate a detail mask given a clip and a clip rescaled with the same kernel.

continues on next page

Table 1 – continued from previous page

lvsfunc.scale.reupscale	A quick 'n easy wrapper used to re-upscale a clip descaled with descale using znedi3.
lvsfunc.scale.test_descale(clip[, width, ...])	Generic function to test descales with; descales and re-upscales a given clip, allowing you to compare the two easily.
lvsfunc.util.pick_removegrain(clip)	Returns rgvs.RemoveGrain if the clip is 16 bit or lower, else rgsf.RemoveGrain.
lvsfunc.util.pick_repair(clip)	Returns rgvs.Repair if the clip is 16 bit or lower, else rgsf.Repair.
lvsfunc.util.quick_resample(clip, function)	A function to quickly resample to 16/8 bit and back to the original depth.

CHAPTER

FIVE

LVSFUNC.AA

lvsfunc.aa.clamp_aa(src, weak, strong[, ...])	Clamp stronger AAs to weaker AAs.
lvsfunc.aa.eedi3([opencl])	Generate eedi3 antialiaser.
lvsfunc.aa.kirsch_aa_mask(clip[, mthr])	Kirsh-based AA mask.
lvsfunc.aa.nnedi3([opencl])	Generate nnedi3 antialiaser.
lvsfunc.aa.nneedi3_clamp(clip[, strength, ...])	A function that clamps eedi3 to nnedi3 for the purpose of reducing eedi3 artifacts.
lvsfunc.aa.taa(clip, aafun)	Perform transpose AA.
lvsfunc.aa.transpose_aa(clip[, eedi3, rep])	Function that performs anti-aliasing over a clip by using nnedi3/eedi3 and transposing multiple times.
lvsfunc.aa.upscaled_sraa(clip[, rfactor, ...])	A function that performs a supersampled single-rate AA to deal with heavy aliasing and broken-up lineart.

Functions for various anti-aliasing functions and wrappers.

lvsfunc.aa.**clamp_aa**(*src*, *weak*, *strong*, *strength=1*)

Clamp stronger AAs to weaker AAs. Useful for clamping upscaled_sraa or eedi3 to nnedi3 for a strong but precise AA.

Stolen from Zastin.

Parameters

- **src** (VideoNode) – Non-AA'd source clip.
- **weak** (VideoNode) – Weakly-AA'd clip (eg: nnedi3)
- **strong** (VideoNode) – Strongly-AA'd clip (eg: eedi3)
- **strength** (float) – Clamping strength (Default: 1)

Return type VideoNode

Returns Clip with clamped anti-aliasing.

lvsfunc.aa.**eedi3**(*opencl=False*, ***override*)

Generate eedi3 antialiaser.

Dependencies: * vapoursynth-EEDI3

Parameters

- **opencl** (bool) – Use OpenCL (Default: False)
- **override** (Any) – eedi3 parameter overrides

Return type Callable[[VideoNode], VideoNode]

Returns Configured eedi3 function

lvsfunc.aa.**kirsch_aa_mask**(*clip*, *mthr=0.25*)
Kirsh-based AA mask.

Dependencies: * kagefunc

Parameters

- **clip** (VideoNode) – Input clip
- **mthr** (float) – Mask threshold, scaled to clip range if between 0 and 1 (inclusive). (Default: 0.25)

Return type VideoNode

lvsfunc.aa.**nnedi3**(*opencl=False*, ***override*)
Generate nnedi3 antialiaser.

Dependencies: * vapoursynth-nnedi3 * vapoursynth-NNEDI3CL (Optional: opencl)

Parameters

- **opencl** (bool) – Use OpenCL (Default: False)
- **override** (Any) – nnedi3 parameter overrides

Return type Callable[[VideoNode], VideoNode]

Returns Configured nnedi3 function

lvsfunc.aa.**nneedi3_clamp**(*clip*, *strength=1*, *mask=None*, *mthr=0.25*, *opencl=False*)
A function that clamps eedi3 to nnedi3 for the purpose of reducing eedi3 artifacts. This should fix every issue created by eedi3. For example: https://i.imgur.com/hYVhetS.jpg

Original function written by Zastin, modified by LightArrowsEXE.

Parameters

- **clip** (VideoNode) – Input clip
- **strength** (float) – Set threshold strength for over/underflow value for clamping eedi3's result to nnedi3 +/- strength * 256 scaled to 8 bit (Default: 1)
- **mask** (Optional[VideoNode]) – Clip to use for custom mask (Default: None)
- **mthr** (float) – Binarize threshold for the mask, scaled to float (Default: 0.25)
- **opencl** (bool) – OpenCL acceleration (Default: False)

Return type VideoNode

Returns Antialiased clip

lvsfunc.aa.**taa**(*clip*, *aafun*)
Perform transpose AA. Example for nnedi3cl: taa(clip, nnedi3(opencl=True))

Parameters

- **clip** (VideoNode) – Input clip.
- **aafun** (Callable[[VideoNode], VideoNode]) – Antialiasing function

Return type VideoNode

Returns Antialiased clip

lvsfunc.aa.**transpose_aa**(*clip*, *eedi3=False*, *rep=13*)
Function that performs anti-aliasing over a clip by using nnedi3/eedi3 and transposing multiple times. This results in overall stronger anti-aliasing. Useful for shows like Yuru Camp with bad lineart problems.

Original function written by Zastin, modified by LightArrowsEXE.

Dependencies: * RGSF (optional: 32 bit clip) * vapoursynth-EEDI3 * vapoursynth-nnedi3 * znedi3

Parameters

- **clip** (VideoNode) – Input clip
- **eedi3** (bool) – Use eedi3 for the interpolation (Default: False)
- **rep** (int) – Repair mode. Pass it 0 to not repair (Default: 13)

Return type VideoNode

Returns Antialiased clip

lvsfunc.aa.**upscaled_sraa**(*clip*, *rfactor=1.5*, *width=None*, *height=None*, *supersampler=<function _nnedi3_supersample>*, *downscaler=<bound method Bicubic.scale of <lvsfunc.kernels.Bicubic object>>*, *aafun=<function _eedi3_singlerate>*)

A function that performs a supersampled single-rate AA to deal with heavy aliasing and broken-up lineart. Useful for heavy antialiasing.

It works by supersampling the clip, performing AA, and then downscaling again. Downscaling can be disabled by setting *downscaler* to *None*, returning the supersampled luma clip. The dimensions of the downscaled clip can also be adjusted by setting *height* or *width*. Setting either *height* or *width* will also scale the chroma accordingly.

Original function written by Zastin, heavily modified by LightArrowsEXE.

Alias for this function is *lvsfunc.sraa*.

Dependencies: * vapoursynth-eedi3 (default aafun) * vapoursynth-nnedi3 (default supersampler and aafun)

Parameters

- **clip** (VideoNode) – Input clip
- **rfactor** (float) – Image enlargement factor. 1.3..2 makes it comparable in strength to vsTAAmbk It is not recommended to go below 1.3 (Default: 1.5)
- **width** (Optional[int]) – Target resolution width. If None, determined from *height*
- **height** (Optional[int]) – Target resolution height (Default: clip.height)
- **supersampler** (Callable[[VideoNode, int, int], VideoNode]) – Supersampler used for upscaling before AA (Default: nnedi3 supersampler)
- **downscaler** (Optional[Callable[[VideoNode, int, int], VideoNode]]) – Downscaler to use after supersampling (Default: Bicubic(b=0, c=1/2)
- **aafun** (Callable[[VideoNode], VideoNode]) – Function used to antialias after supersampling (Default: eedi3 with nnedi3 sclip)

Return type VideoNode

Returns Antialiased clip

CHAPTER

SIX

LVSFUNC.COMPARISON

lvsfunc.comparison.compare(clip_a, clip_b[, ...])	Allows for the same frames from two different clips to be compared by interleaving them into a single clip.
lvsfunc.comparison.diff(*clips[, thr, ...])	Creates a standard *lvsfunc.comparison.Stack* between frames from two clips that have differences.
lvsfunc.comparison.diff_hardsub_mask(a, b, ...)	Diff func for *lvsfunc.comparison.diff()* to use a hardsub mask.
lvsfunc.comparison.interleave(*clips, ...)	Small convenience function for interleaving clips.
lvsfunc.comparison.split(*clips, **named-clips)	Small convenience funciton for splitting clips along the x-axis and then stacking.
lvsfunc.comparison.stack_compare(*clips[, ...])	A simple wrapper that allows you to compare two clips by stacking them.
lvsfunc.comparison.stack_horizontal(*clips, ...)	Small convenience function for stacking clips horizontally.
lvsfunc.comparison.stack_planes(clip, /[, ...])	Stacks the planes of a clip.
lvsfunc.comparison.stack_vertical(*clips, ...)	Small convenience function for stacking clips vertically.
lvsfunc.comparison.tile(*clips, **named-clips)	Small convenience function for tiling clips in a rectangular pattern.

Functions intended to be used to make comparisons between different sources or to be used to analyze something from a single clip.

class lvsfunc.comparison.**Comparer**(*clips*, */*, *, *label_alignment=7*)

Bases: abc.ABC

Base class for comparison functions.

Parameters

- **clips** (Union[Dict[str, VideoNode], Sequence[VideoNode]]) – A dict mapping names to clips or simply a sequence of clips in a tuple or a list. If given a dict, the names will be overlayed on the clips using VideoNode.text.Text. If given a simple sequence of clips, the *label_alignment* parameter will have no effect and the clips will not be labeled. The order of the clips in either a dict or a sequence will be kept in the comparison.
- **label_alignment** (int) – An integer from 1-9, corresponding to the positions of the keys on a numpad. Only used if *clips* is a dict. Determines where to place clip name using VideoNode.text.Text (Default: 7)

property clip

Returns the comparison as a single VideoNode for further manipulation or attribute inspection.

`comp_clip = Comparer(...).clip` is the intended use in encoding scripts.

Return type `VideoNode`

class lvsfunc.comparison.**Direction**(*value*)
Bases: `enum.IntEnum`

Enum to simplify direction argument.

HORIZONTAL = 0

VERTICAL = 1

class lvsfunc.comparison.**Interleave**(*clips*, */*, ***, *label_alignment=7*)
Bases: *lvsfunc.comparison.Comparer*

From the VapourSynth documentation: *Returns a clip with the frames from all clips interleaved. For example, Interleave(A=clip1, B=clip2) will return A.Frame 0, B.Frame 0, A.Frame 1, B.Frame 1, ...*

Acts as a convenience combination function of `vapoursynth.core.text.Text` and `vapoursynth.core.std.Interleave`.

Parameters

- **clips** (`Union[Dict[str, VideoNode], Sequence[VideoNode]]`) – A dict mapping names to clips or simply a sequence of clips in a tuple or a list. If given a dict, the names will be overlayed on the clips using `VideoNode.text.Text`. If given a simple sequence of clips, the *label_alignment* parameter will have no effect and the clips will not be labeled. The order of the clips in either a dict or a sequence will be kept in the comparison.
- **label_alignment** (`int`) – An integer from 1-9, corresponding to the positions of the keys on a numpad. Only used if *clips* is a dict. Determines where to place clip name using `VideoNode.text.Text` (Default: 7)

class lvsfunc.comparison.**Split**(*clips*, */*, ***, *direction=<Direction.HORIZONTAL: 0>*, *label_alignment=7*)
Bases: *lvsfunc.comparison.Stack*

Split an unlimited amount of clips into one VideoNode with the same dimensions as the original clips. Handles odd-sized resolutions or resolutions that can't be evenly split by the amount of clips specified.

The remaining pixel width/height (`clip.dimension % number_of_clips`) will be always given to the last clip specified. For example, five *104 x 200* clips will result in a *((20 x 200) * 4) + (24 x 200)* horiztonal stack of clips.

Parameters

- **clips** (`Union[Dict[str, VideoNode], Sequence[VideoNode]]`) – A dict mapping names to clips or simply a sequence of clips in a tuple or a list. If given a dict, the names will be overlayed on the clips using `VideoNode.text.Text`. If given a simple sequence of clips, the *label_alignment* parameter will have no effect and the clips will not be labeled. The order of the clips in either a dict or a sequence will be kept in the comparison.
- **direction** (*Direction*) – Determines the axis to split the clips on (Default: *lvsfunc.comparison.Direction.HORIZONTAL*)
- **label_alignment** (`int`) – An integer from 1-9, corresponding to the positions of the keys on a numpad. Only used if *clips* is a dict. Determines where to place clip name using `VideoNode.text.Text` (Default: 7)

class lvsfunc.comparison.**Stack**(*clips*, */*, ***, *direction=<Direction.HORIZONTAL: 0>*, *label_alignment=7*)
Bases: *lvsfunc.comparison.Comparer*

Stacks clips horizontally or vertically.

Acts as a convenience combination function of `vapoursynth.core.text.Text` and either `vapoursynth.core.std.StackHorizontal` or `vapoursynth.core.std.StackVertical`.

Parameters

- **clips** (Union[Dict[str, VideoNode], Sequence[VideoNode]]) – A dict mapping names to clips or simply a sequence of clips in a tuple or a list. If given a dict, the names will be overlayed on the clips using `VideoNode.text.Text`. If given a simple sequence of clips, the *label_alignment* parameter will have no effect and the clips will not be labeled. The order of the clips in either a dict or a sequence will be kept in the comparison.
- **direction** (*Direction*) – Direction of the stack (Default: *lvsfunc.comparison.Direction.HORIZONTAL*)
- **label_alignment** (int) – An integer from 1-9, corresponding to the positions of the keys on a numpad. Only used if *clips* is a dict. Determines where to place clip name using `VideoNode.text.Text` (Default: 7)

class lvsfunc.comparison.**Tile**(*clips*, */*, ***, *arrangement=None*, *label_alignment=7*)

Bases: *lvsfunc.comparison.Comparer*

Tiles clips in a mosaic manner, filling rows first left-to-right, then stacking.

The arrangement of the clips can be specified with the *arrangement* parameter. Rows are specified as lists of ints inside of a larger list specifying the order of the rows. Think of this as a 2-dimensional array of 0s and 1s with *0* representing an empty slot and *1* representing the next clip in the sequence.

If *arrangement* is not specified, the function will attempt to fill a square with dimensions *n x n* where *n* is equivalent to `math.ceil(math.sqrt(len(clips))`. The bottom rows will be dropped if empty.

```
# For example, for 3 clips, the automatic arrangement becomes:
[
 [1, 1],
 [1, 0]
]

# For 10 clips, the automatic arrangement becomes:
[
 [1, 1, 1, 1],
 [1, 1, 1, 1],
 [1, 1, 0, 0]
]

# For custom arrangements, such as (for 4 clips):
[
 [0, 1, 0, 1],
 [1],
 [0, 1]
]
# the rows will be auto-padded with 0's to be the same length.
```

Parameters

- **clips** (Union[Dict[str, VideoNode], Sequence[VideoNode]]) – A dict mapping names to clips or simply a sequence of clips in a tuple or a list. If given a dict, the names will be overlayed on the clips using `VideoNode.text.Text`. If given a simple sequence of clips, the *label_alignment* parameter will have no effect and the clips will not be labeled. The order of the clips in either a dict or a sequence will be kept in the comparison.

- **arrangement** (Optional[List[List[int]]]) – 2-dimension array (list of lists) of 0s and 1s representing a list of rows of clips(*1*) or blank spaces(*0*) (Default: None)
- **label_alignment** (int) – An integer from 1-9, corresponding to the positions of the keys on a numpad. Only used if *clips* is a dict. Determines where to place clip name using VideoNode.text.Text (Default: 7)

lvsfunc.comparison.**compare**(*clip_a*, *clip_b*, *frames=None*, *rand_total=None*, *force_resample=True*, *print_frame=True*, *mismatch=False*)

Allows for the same frames from two different clips to be compared by interleaving them into a single clip. Clips are automatically resampled to 8 bit YUV -> RGB24 to emulate how a monitor shows the frame. This can be disabled by setting *force_resample* to False.

Alias for this function is *lvsfunc.comp*.

Parameters

- **clip_a** (VideoNode) – Clip to compare
- **clip_b** (VideoNode) – Second clip to compare
- **frames** (Optional[List[int]]) – List of frames to compare (Default: None)
- **rand_total** (Optional[int]) – Number of random frames to pick (Default: None)
- **force_resample** (bool) – Forcibly resamples the clip to RGB24 (Default: True)
- **print_frame** (bool) – Print frame numbers (Default: True)
- **mismatch** (bool) – Allow for clips with different formats and dimensions to be compared (Default: False)

Return type VideoNode

Returns Interleaved clip containing specified frames from *clip_a* and *clip_b*

lvsfunc.comparison.**diff**(**clips*, *thr=72*, *height=288*, *return_array=False*, *return_frames=False*, *diff_func=<function <lambda>>*, ***namedclips*)

Creates a standard *lvsfunc.comparison.Stack* between frames from two clips that have differences. Useful for making comparisons between TV and BD encodes, as well as clean and hardsubbed sources.

There are two methods used here to find differences. If *thr* is below 1, PlaneStatsDiff is used to figure out the differences. Else, if *thr* is equal than or higher than 1, PlaneStatsMin/Max are used.

Recommended is PlaneStatsMin/Max, as those seem to catch more outrageous differences more easily and not return too many starved frames.

Note that this might catch artifacting as differences! Make sure you verify every frame with your own eyes!

Alias for this function is *lvsfunc.diff*.

Parameters

- **clips** (VideoNode) – Clips for comparison (order is kept)
- **namedclips** (VideoNode) – Keyword arguments of *name=clip* for all clips in the comparison. Clips will be labeled at the top left with their *name*.
- **thr** (float) – Threshold, <= 1 uses PlaneStatsDiff, >1 uses Max/Min. Value must be below 128 (Default: 72)
- **height** (int) – Height in px to downscale clips to if *return_array* is False (MakeDiff clip will be twice this resolution) (Default: 288)
- **return_array** (bool) – Return frames as an interleaved comparison (using *lvsfunc.comparison.Interleave*) (Default: False)

- **return_frames** (bool) – Adds *frames list* to the return. (Default: False)
- **diff_func** (Callable[[VideoNode, VideoNode], VideoNode]) – Function for calculating diff in PlaneStatsMin/Max mode. (Default: core.std.MakeDiff)

Return type Union[VideoNode, Tuple[VideoNode, List[int]]]

Returns Either an interleaved clip of the differences between the two clips or a stack of both input clips on top of MakeDiff clip. Optionally, you can allow the function to also return a list of frames as well.

lvsfunc.comparison.**diff_hardsub_mask**(*a*, *b*, ***kwargs*)

Diff func for *lvsfunc.comparison.diff()* to use a hardsub mask. This is kinda slow.

Parameters

- **a** (VideoNode) – Clip A
- **b** (VideoNode) – Clip B

Return type VideoNode

Returns Diff masked with *lvsfunc.dehardsub.hardsub_mask()*

lvsfunc.comparison.**interleave**(**clips*, ***namedclips*)

Small convenience function for interleaving clips.

Parameters

- **clips** (VideoNode) – Clips for comparison (order is kept)
- **namedclips** (VideoNode) – Keyword arguments of *name=clip* for all clips in the comparison. Clips will be labeled at the top left with their *name*.

Return type VideoNode

Returns An interleaved clip of all the *clips*/*namedclips* specified

lvsfunc.comparison.**split**(**clips*, ***namedclips*)

Small convenience funciton for splitting clips along the x-axis and then stacking. Accounts for odd-resolution clips by giving overflow columns to the last clip specified. All clips must have the same dimensions (width and height).

Parameters

- **clips** (VideoNode) – Clips for comparison (order is kept left to right)
- **namedclips** (VideoNode) – Keyword arguments of *name=clip* for all clips in the comparison. Clips will be labeled at the bottom with their *name*.

Return type VideoNode

Returns A clip with the same dimensions as any one of the input clips with all *clips*/*namedclips* represented as individual vertical slices.

lvsfunc.comparison.**stack_compare**(**clips*, *make_diff=True*, *height=288*, *warn=None*)

A simple wrapper that allows you to compare two clips by stacking them.

Best to use when trying to match two sources frame-accurately. Alias for this function is *lvsfunc.scomp*.

Parameters

- **clips** (VideoNode) – Clips to compare
- **make_diff** (bool) – Create and stack a diff (only works if two clips are given) (Default: True)

- **height** (int) – Height in px to rescale clips to if *make_diff* is True (MakeDiff clip will be twice this resolution) (Default: 288)
- **warn** (Optional[Any]) – Unused parameter kept for backward compatibility

Return type VideoNode

Returns Clip with *clips* stacked

lvsfunc.comparison.**stack_horizontal**(**clips*, ***namedclips*)

Small convenience function for stacking clips horizontally.

Parameters

- **clips** (VideoNode) – Clips for comparison (order is kept left to right)
- **namedclips** (VideoNode) – Keyword arguments of *name=clip* for all clips in the comparison. Clips will be labeled at the top left with their *name*.

Return type VideoNode

Returns A horizontal stack of the *clips*/*namedclips*

lvsfunc.comparison.**stack_planes**(*clip*, */*, *stack_vertical=False*)

Stacks the planes of a clip. For 4:2:0 subsampled clips, the two half-sized planes will be stacked in the opposite direction specified (vertical by default), then stacked with the full-sized plane in the direction specified (horizontal by default).

Parameters

- **clip** (VideoNode) – Input clip (must be in YUV or RGB planar format)
- **stack_vertical** (bool) – Stack the planes vertically (Default: False)

Return type VideoNode

Returns Clip with stacked planes

lvsfunc.comparison.**stack_vertical**(**clips*, ***namedclips*)

Small convenience function for stacking clips vertically.

Parameters

- **clips** (VideoNode) – Clips for comparison (order is kept top to bottom)
- **namedclips** (VideoNode) – Keyword arguments of *name=clip* for all clips in the comparison. Clips will be labeled at the top left with their *name*.

Return type VideoNode

Returns A vertical stack of the *clips*/*namedclips*

lvsfunc.comparison.**tile**(**clips*, ***namedclips*)

Small convenience function for tiling clips in a rectangular pattern. All clips must have the same dimensions (width and height). If 3 clips are given, a 2x2 square with one blank slot will be returned. If 6 clips are given, a 3x2 rectangle will be returned.

Parameters

- **clips** (VideoNode) – Clips for comparison
- **namedclips** (VideoNode) – Keyword arguments of *name=clip* for all clips in the comparison. Clips will be labeled at the top left with their *name*.

Return type VideoNode

Returns A clip with all input *clips*/*namedclips* automatically tiled most optimally into a rectangular arrrangement

CHAPTER

SEVEN

LVSFUNC.DEHALO

lvsfunc.dehalo.bidehalo(clip[, ref, sigmaS, ...])	A simple dehaloing function using bilateral and BM3D to remove bright haloing around edges.

Wrappers for dehaloing functions.

lvsfunc.dehalo.**bidehalo**(*clip*, *ref=None*, *sigmaS=1.5*, *sigmaR=0.0196078431372549*, *bilateral_args={}*, *bm3d_args={}*)

A simple dehaloing function using bilateral and BM3D to remove bright haloing around edges. If a ref clip is passed, that will be masked onto the clip instead of a blurred clip.

Parameters

- **clip** (VideoNode) – Source clip
- **ref** (Optional[VideoNode]) – Ref clip
- **sigmaS** (float) – Bilateral's spatial weight sigma
- **sigmaS** – Bilateral's range weight sigma
- **bilateral_args** (Dict[str, Any]) – Additional parameters to pass to bilateral
- **bm3d_args** (Dict[str, Any]) – Additional parameters to pass to denoise.bm3d

Return type VideoNode

Returns Dehalo'd clip

CHAPTER

EIGHT

LVSFUNC.DEHARDSUB

lvsfunc.dehardsub.bounded_dehardsub(hrdsb, ...)	Apply a list of *lvsfunc.dehardsub.HardsubSign*
lvsfunc.dehardsub.get_all_masks(hrdsb, ref, ...)	Get a clip of *lvsfunc.dehardsub.HardsubSign* masks.
lvsfunc.dehardsub.hardsub_mask(hrdsb, ref[, ...])	Zastin's spatially-aware hardsub mask.

class lvsfunc.dehardsub.**HardsubMask**(*ranges*, *bound=None*, *, *blur=False*, *refframes=None*)

Bases: *lvsfunc.mask.DeferredMask*, abc.ABC

Dehardsub masking interface.

Provides extra functions potentially useful for dehardsubbing.

Parameters

- **range** – A single range or list of ranges to replace, compatible with *lvsfunc.misc.replace_ranges*
- **bound** (Union[*BoundingBox*, Tuple[Tuple[int, int], Tuple[int, int]], None]) – A *lvsfunc.mask.BoundingBox* or a tuple that will be converted. (Default: None, no bounding)
- **blur** (bool) – Blur the bounding mask (Default: True)
- **refframe** – A single frame number to use to generate the mask or a list of frame numbers with the same length as range

get_progressive_dehardsub(*hrdsb*, *ref*, *partials*)

Dehardsub using multiple superior hardsubbed sources and one inferior non-subbed source.

Parameters

- **hrdsb** (VideoNode) – Hardsub master source (eg Wakanim RU dub)
- **ref** (VideoNode) – Non-subbed reference source (eg CR, Funi, Amazon)
- **partials** (List[VideoNode]) – Sources to use for partial dehardsubbing (eg Waka DE, FR, SC)

Return type Tuple[List[VideoNode], List[VideoNode]]

Returns Dehardsub stages and masks used for progressive dehardsub

apply_dehardsub(*hrdsb*, *ref*, *partials*)

Apply dehardsubbing to a clip.

Parameters

- **hrdsb** (VideoNode) – Hardsubbed source
- **ref** (VideoNode) – Non-hardsubbed source
- **partials** (Optional[List[VideoNode]]) – Other hardsubbed sources

Return type VideoNode

Returns Dehardsubbed clip

ranges: List[Union[int, Tuple[int, int]]]

bound: Optional[*lvsfunc.mask.BoundingBox*]

refframes: List[int]

blur: bool

class lvsfunc.dehardsub.**HardsubSign**(**args*, *thresh=0.06*, *minimum=1*, *expand=8*, *inflate=7*, ***kwargs*)

Bases: *lvsfunc.dehardsub.HardsubMask*

Hardsub scenefiltering helper using Zastin's hardsub mask.

Parameters

- **thresh** (float) – Binarization threshold, [0, 1] (Default: 0.06)
- **expand** (int) – std.Maximum iterations (Default: 8)
- **inflate** (int) – std.Inflate iterations (Default: 7)

thresh: float

minimum: int

expand: int

inflate: int

class lvsfunc.dehardsub.**HardsubSignKgf**(**args*, *highpass=5000*, *expand=8*, ***kwargs*)

Bases: *lvsfunc.dehardsub.HardsubMask*

Hardsub scenefiltering helper using kgf.hardsubmask_fades.

Dependencies: * kagefunc

Parameters

- **highpass** (int) – Highpass filter for hardsub detection (16-bit, Default: 5000)
- **expand** (int) – kgf.hardsubmask_fades expand parameter (Default: 8)

highpass: int

expand: int

class lvsfunc.dehardsub.**HardsubLine**(**args*, *expand=None*, ***kwargs*)

Bases: *lvsfunc.dehardsub.HardsubMask*

Hardsub scenefiltering helper using kgf.hardsubmask.

Dependencies: * kagefunc

Parameters **expand** (Optional[int]) – kgf.hardsubmask expand parameter (Default: clip.width // 200)

expand: Optional[int]

class lvsfunc.dehardsub.**HardsubLineFade**(*ranges*, **args*, *refframe=0.5*, ***kwargs*)
Bases: *lvsfunc.dehardsub.HardsubLine*

Hardsub scenefiltering helper using kgf.hardsubmask. Similar to *lvsfunc.dehardsub.HardsubLine* but automatically sets the reference frame to the range's midpoint.

Parameters **refframe** (float) – Desired reference point as a percent of the frame range. 0 = first frame, 1 = last frame, 0.5 = midpoint (Default)

expand: Optional[int]

ranges: List[Range]

bound: Optional[BoundingBox]

refframes: List[int]

blur: bool

Dehardsubbing helpers.

class lvsfunc.dehardsub.**HardsubSignFade**(*ranges*, **args*, *refframe=0.5*, ***kwargs*)
Bases: *lvsfunc.dehardsub.HardsubSign*

Hardsub scenefiltering helper using Zastin's sign mask. Similar to *lvsfunc.dehardsub.HardsubSign* but automatically sets the reference frame to the range's midpoint.

Parameters **refframe** (float) – Desired reference point as a percent of the frame range. 0 = first frame, 1 = last frame, 0.5 = midpoint (Default)

blur: bool

bound: Optional[BoundingBox]

expand: int

inflate: int

minimum: int

ranges: List[Range]

refframes: List[int]

thresh: float

lvsfunc.dehardsub.**bounded_dehardsub**(*hrdsb*, *ref*, *signs*, *partials=None*)
Apply a list of *lvsfunc.dehardsub.HardsubSign*

Parameters

- **hrdsb** (VideoNode) – Hardsubbed source
- **ref** (VideoNode) – Reference clip
- **signs** (List[*HardsubMask*]) – List of *lvsfunc.dehardsub.HardsubSign* to apply

Return type VideoNode

Returns Dehardsubbed clip

lvsfunc.dehardsub.**get_all_masks**(*hrdsb*, *ref*, *signs*)
Get a clip of *lvsfunc.dehardsub.HardsubSign* masks.

Parameters

- **hrdsb** (VideoNode) – Hardsubbed source

- **ref** (VideoNode) – Reference clip
- **signs** (List[*HardsubMask*]) – List of *lvsfunc.dehardsub.HardsubSign* to generate masks for

Return type VideoNode

Returns Clip of all hardsub masks

lvsfunc.dehardsub.**hardsub_mask**(*hrdsb*, *ref*, *thresh=0.06*, *minimum=1*, *expand=8*, *inflate=7*)

Zastin's spatially-aware hardsub mask.

Parameters

- **hrdsb** (VideoNode) – Hardsubbed source
- **ref** (VideoNode) – Reference clip
- **thresh** (float) – Binarization threshold, [0, 1] (Default: 0.06)
- **minimum** (int) – Times to minimize the max (Default: 1)
- **expand** (int) – Times to maximize the mask (Default: 8)
- **inflate** (int) – Times to inflate the mask (Default: 7)

Return type VideoNode

Returns Hardsub mask

CHAPTER

NINE

LVSFUNC.DEINTERLACE

lvsfunc.deinterlace.deblend(clip[, rep])	A simple function to fix deblending for interlaced video with an AABBA blending pattern, where A is a regular frame and B is a blended frame.
lvsfunc.deinterlace.decomb(clip, TFF[, ...])	A filter that performs relatively aggressive filtering to get rid of the combing on a interlaced/telecined source.
lvsfunc.deinterlace.dir_deshimmer(clip[, ...])	Directional deshimmering function.
lvsfunc.deinterlace.dir_unsharp(clip[, ...])	Diff'd directional unsharpening function.
lvsfunc.deinterlace.SIVTC(clip[, pattern, ...])	A very simple fieldmatching function.

Functions to help with deinterlacing or deinterlaced content.

lvsfunc.deinterlace.**SIVTC**(*clip*, *pattern=0*, *TFF=True*, *decimate=True*)

A very simple fieldmatching function.

This is essentially a stripped-down JIVTC offering JUST the basic fieldmatching and decimation part. As such, you may need to combine multiple instances if patterns change throughout the clip.

Parameters

- **clip** (VideoNode) – Input clip
- **pattern** (int) – First frame of any clean-combed-combed-clean-clean sequence
- **TFF** (bool) – Top-Field-First
- **decimate** (bool) – Drop a frame every 5 frames to get down to 24000/1001

Return type VideoNode

Returns IVTC'd clip

lvsfunc.deinterlace.**deblend**(*clip*, *rep=None*)

A simple function to fix deblending for interlaced video with an AABBA blending pattern, where A is a regular frame and B is a blended frame.

Assuming there's a constant pattern of frames (labeled A, B, C, CD, and DA in this function), blending can be fixed by calculating the D frame by getting halves of CD and DA, and using that to fix up CD. DA is then dropped because it's a duplicated frame.

Doing this will result in some of the artifacting being added to the deblended frame, but we can mitigate that by repairing the frame with the non-blended frame before it.

For more information, please refer to this blogpost by torchlight: https://mechaweaponsvidya.wordpress.com/2012/09/13/adventures-in-deblending/

Dependencies: * RGSF (optional: 32 bit clip)

Parameters

- **clip** (VideoNode) – Input clip
- **rep** (Optional[int]) – Repair mode for the deblended frames, no repair if None (Default: None)

Return type VideoNode

Returns Deblended clip

lvsfunc.deinterlace.**decomb**(*clip*, *TFF*, *mode=1*, *ref=None*, *decimate=True*, *vinv=False*, *sharpen=False*, *dir='v'*, *rep=None*, *show_mask=False*, ***kwargs*)

A filter that performs relatively aggressive filtering to get rid of the combing on a interlaced/telecined source. Decimation can be disabled if the user wishes to decimate the clip themselves. Enabling vinverse will result in more aggressive decombing at the cost of potential detail loss. Sharpen will perform a directional unsharpening. Direction can be set using *dir*. A reference clip can be passed with *ref*, which will be used by VFM to create the output frames.

Base function written by Midlifecrisis from the WEEB AUTISM server, and modified by LightArrowsEXE.

Dependencies: * combmask * havsfunc * RGSF (optional: 32 bit clip)

Parameters

- **clip** (VideoNode) – Input clip
- **TFF** (bool) – Top-Field-First
- **mode** (int) – Sets the matching mode or strategy to use for VFM
- **ref** (Optional[VideoNode]) – Reference clip for VFM's *clip2* parameter
- **decimate** (bool) – Decimate the video after deinterlacing (Default: True)
- **vinv** (bool) – Use vinverse to get rid of additional combing (Default: False)
- **sharpen** (bool) – Unsharpen after deinterlacing (Default: False)
- **dir** (str) – Directional vector. 'v' = Vertical, 'h' = Horizontal (Default: v)
- **rep** (Optional[int]) – Repair mode for repairing the decombed clip using the original clip (Default: None)
- **show_mask** (bool) – Return combmask
- **kwargs** (Any) – Arguments to pass to QTGMC (Default: SourceMatch=3, Lossless=2, TR0=1, TR1=2, TR2=3, FPSDivisor=2)

Return type VideoNode

Returns Decombed and optionally decimated clip

lvsfunc.deinterlace.**dir_deshimmer**(*clip*, *TFF=True*, *dh=False*, *transpose=True*, *show_mask=False*)

Directional deshimmering function.

Only works (in the few instances it does, anyway) for obvious horizontal and vertical shimmering. Odds of success are low. But if you're desperate, it's worth a shot.

Dependencies: * vapoursynth-nnedi3

Parameters

- **clip** (VideoNode) – Input clip
- **TFF** (bool) – Top Field First. Set to False if TFF doesn't work (Default: True)
- **dh** (bool) – Interpolate to double the height of given clip beforehand (Default: False)
- **transpose** (bool) – Transpose the clip before attempting to deshimmer (Default: True)
- **show_mask** (bool) – Show nnedi3's mask (Default: False)

Return type VideoNode

Returns Deshimmered clip

lvsfunc.deinterlace.**dir_unsharp**(*clip*, *strength=1.0*, *dir='v'*, *h=3.4*)

Diff'd directional unsharpening function. Performs one-dimensional sharpening as such: "Original + (Original - blurred) * Strength"

This particular function is recommended for SD content, specifically after deinterlacing.

Special thanks to thebombzen and kageru for writing the bulk of this.

Dependencies: * knlmeanscl

Parameters

- **clip** (VideoNode) – Input clip
- **strength** (float) – Amount to multiply blurred clip with original clip by (Default: 1.0)
- **dir** (str) – Directional vector. 'v' = Vertical, 'h' = Horizontal (Default: v)
- **h** (float) – Sigma for knlmeans, to prevent noise from getting sharpened (Default: 3.4)

Return type VideoNode

Returns Unsharpened clip

CHAPTER

TEN

LVSFUNC.DENOISE

lvsfunc.denoise.bm3d(clip[, sigma, radius, ...])	A wrapper function for the BM3D denoiser.

Denoising functions.

lvsfunc.denoise.**bm3d**(*clip*, *sigma=0.75*, *radius=None*, *ref=None*, *pre=None*, *refine=1*, *matrix_s='709'*, *basic_args={}*, *final_args={}*)

A wrapper function for the BM3D denoiser.

Dependencies: * VapourSynth-BM3D

Parameters

- **clip** (VideoNode) – Input clip
- **sigma** (Union[float, List[float]]) – Denoising strength for both basic and final estimations
- **radius** (Union[int, List[int], None]) – Temporal radius for both basic and final estimations
- **ref** (Optional[VideoNode]) – Reference clip for the final estimation
- **pre** (Optional[VideoNode]) – Prefiltered clip for the basic estimation
- **refine** (int) – Iteration of the final clip. 0 = basic estimation only 1 = basic + final estimation n = basic + n final estimations
- **matrix_s** (str) – Color matrix of the input clip
- **basic_args** (Dict[str, Any]) – Args to pass to the basic estimation
- **final_args** (Dict[str, Any]) – Args to pass to the final estimation

Return type VideoNode

Returns Denoised clip

CHAPTER

ELEVEN

LVSFUNC.MASK

lvsfunc.mask.detail_mask(clip[, sigma, rad, ...])	A wrapper for creating a detail mask to be used during denoising and/or debanding.
lvsfunc.mask.halo_mask(clip[, rad, brz, ...])	A halo mask to catch basic haloing, inspired by the mask from FineDehalo.
lvsfunc.mask.range_mask(clip[, rad, radc])	Min/max mask with separate luma/chroma radii.

class lvsfunc.mask.**BoundingBox**(*pos*, *size*)

Bases: object

A positional bounding box. Basically kagefunc.squaremask but can be configured then deferred.

Uses Position + Size, like provided by GIMP's rectangle selection tool.

Parameters

- **pos** (Union[*Position*, Tuple[int, int]]) – Offset of top-left corner of the bounding box from the top-left corner of the frame. Supports either a *lvsfunc.types.Position* or a tuple that will be converted.
- **size** (Union[*Size*, Tuple[int, int]]) – Offset of the bottom-right corner of the bounding box from the top-left corner of the bounding box. Supports either a *lvsfunc.types.Size* or a tuple that will be converted.

pos: *lvsfunc.types.Position*

size: *lvsfunc.types.Size*

get_mask(*ref*)

Get a mask representing the bounding box

Parameters **ref** (VideoNode) – Reference clip for format, resolution, and length.

Return type VideoNode

Returns Square mask representing the bounding box.

class lvsfunc.mask.**DeferredMask**(*ranges*, *bound=None*, ***, *blur=False*, *refframes=None*)

Bases: abc.ABC

Deferred masking interface.

Provides an interface to use different preconfigured masking functions. Provides support for ranges, reference frames, and bounding.

Parameters

- **range** – A single range or list of ranges to replace, compatible with *lvsfunc.misc.replace_ranges*

- **bound** (Union[*BoundingBox*, Tuple[Tuple[int, int], Tuple[int, int]], None]) – A *lvsfunc.mask.BoundingBox* or a tuple that will be converted. (Default: None, no bounding)
- **blur** (bool) – Blur the bounding mask (Default: False)
- **refframe** – A single frame number to use to generate the mask or a list of frame numbers with the same length as range

ranges: List[Union[int, Tuple[int, int]]]

blur: bool

bound: Optional[*lvsfunc.mask.BoundingBox*]

refframes: List[int]

get_mask(*clip*, *ref*)

Get the bounded mask.

Parameters

- **clip** (VideoNode) – Source
- **ref** (VideoNode) – Reference clip

Return type VideoNode

Returns Bounded mask

Wrappers and masks for denoising.

lvsfunc.mask.**detail_mask**(*clip*, *sigma=None*, *rad=3*, *radc=2*, *brz_a=0.025*, *brz_b=0.045*)

A wrapper for creating a detail mask to be used during denoising and/or debanding. The detail mask is created using debandshit's range mask, and is then merged with Prewitt to catch lines it may have missed.

Function is curried to allow parameter tuning when passing to denoisers that allow you to pass your own mask.

Dependencies: * VapourSynth-Bilateral (optional: sigma) * RGSF (optional: 32 bit clip)

Parameters

- **clip** (VideoNode) – Input clip
- **sigma** (Optional[float]) – Sigma for Bilateral for pre-blurring (Default: False)
- **rad** (int) – The luma equivalent of gradfun3's "mask" parameter
- **radc** (int) – The chroma equivalent of gradfun3's "mask" parameter
- **brz_a** (float) – Binarizing thresh for the detail mask. Scaled to clip's depth if between 0 and 1 (inclusive), else assumed to be in native range. (Default: 0.025)
- **brz_b** (float) – Binarizing thresh for the edge mask. Scaled to clip's depth if between 0 and 1 (inclusive), else assumed to be in native range. (Default: 0.045)

Return type VideoNode

Returns Detail mask

lvsfunc.mask.**halo_mask**(*clip*, *rad=2*, *brz=0.35*, *thmi=0.315*, *thma=0.5*, *thlimi=0.195*, *thlima=0.392*, *edgemask=None*)

A halo mask to catch basic haloing, inspired by the mask from FineDehalo. Most was copied from there, but some key adjustments were made to center it specifically around masking.

rx and ry are now combined into rad and expects an integer. Float made sense for FineDehalo since it uses DeHalo_alpha for dehaloing, but the masks themselves use rounded rx/ry values, so there's no reason to bother with floats here.

All thresholds are float and will be scaled to `clip`'s format. If thresholds are greater than 1, they will be asummed to be in 8-bit and scaled accordingly.

Parameters

- **clip** (`VideoNode`) – Input clip
- **rad** (`int`) – Radius for the mask
- **brz** (`float`) – Binarizing for shrinking mask (Default: 0.35)
- **thmi** (`float`) – Minimum threshold for sharp edges; keep only the sharpest edges
- **thma** (`float`) – Maximum threshold for sharp edges; keep only the sharpest edges
- **thlimi** (`float`) – Minimum limiting threshold; includes more edges than previously, but ignores simple details
- **thlima** (`float`) – Maximum limiting threshold; includes more edges than previously, but ignores simple details
- **edgemask** (`Optional[VideoNode]`) – Edgemask to use. If None, uses `clip.std.Prewitt()` (Default: None).

Return type `VideoNode`

Returns Halo mask

`lvsfunc.mask.`**range_mask**(*clip*, *rad=2*, *radc=0*)

Min/max mask with separate luma/chroma radii.

rad/radc are the luma/chroma equivalent of gradfun3's "mask" parameter. The way gradfun3's mask works is on an 8 bit scale, with rounded dithering of high depth input. As such, when following this filter with a Binarize, use the following conversion steps based on input:

- 8 bit = Binarize(2) or Binarize(thr_det)
- 16 bit = Binarize(384) or Binarize((thr_det - 0.5) * 256)
- floats = Binarize(0.005859375) or Binarize((thr_det - 0.5) / 256)

When radii are equal to 1, this filter becomes identical to mt_edge("min/max", 0, 255, 0, 255).

Parameters

- **clip** (`VideoNode`) – Input clip
- **rad** (`int`) – Depth in pixels of the detail/edge masking
- **radc** (`int`) – Chroma equivalent to `rad`

Return type `VideoNode`

Returns Range mask

CHAPTER

TWELVE

LVSFUNC.KERNELS

Kernels for vapoursynth internal resizers. Intended for use by *lvsfunc.scale* functions.

class lvsfunc.kernels.**BSpline**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**Bicubic**(*b=0*, *c=0.5*, ***kwargs*)
Bases: *lvsfunc.kernels.Kernel*

Built-in bicubic resizer.

Dependencies: * VapourSynth-descale

Parameters

- **b** (float) – B-param for bicubic kernel
- **c** (float) – C-param for bicubic kernel

descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**BicubicSharp**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**Bilinear**(***kwargs*)
Bases: *lvsfunc.kernels.Kernel*

Built-in bilinear resizer.

descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Catrom**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**Hermite**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**Kernel**(***kwargs*)
Bases: abc.ABC

Abstract scaling kernel interface.

Additional kwargs supplied to constructor are passed only to the internal resizer, not the descale resizer.

abstract descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

abstract scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Lanczos**(*taps=4*, ***kwargs*)
Bases: *lvsfunc.kernels.Kernel*

Built-in lanczos resizer.

Dependencies: * VapourSynth-descale

Parameters **taps** (int) – taps param for lanczos kernel

descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Mitchell**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**Point**(***kwargs*)
Bases: *lvsfunc.kernels.Kernel*

Built-in point resizer.

descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Robidoux**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**RobidouxSharp**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**RobidouxSoft**(***kwargs*)
Bases: *lvsfunc.kernels.Bicubic*

class lvsfunc.kernels.**Spline16**(***kwargs*)
Bases: *lvsfunc.kernels.Kernel*

Built-in spline16 resizer.

Dependencies: * VapourSynth-descale

descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Spline36**(***kwargs*)

Bases: *lvsfunc.kernels.Kernel*

Built-in spline36 resizer.

Dependencies: * VapourSynth-descale

descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Spline64**(***kwargs*)

Bases: *lvsfunc.kernels.Kernel*

Built-in spline64 resizer.

Dependencies: * VapourSynth-descale

descale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

scale(*clip*, *width*, *height*, *shift=(0, 0)*)

Return type VideoNode

CHAPTER

THIRTEEN

LVSFUNC.MISC

lvsfunc.misc.allow_variable([width, height, ...])	Decorator allowing a variable-res and/or variable-format clip to be passed to a function that otherwise would not be able to accept it.
lvsfunc.misc.chroma_injector(func)	Decorator allowing injection of reference chroma into a function which would normally only receive luma, such as an upscaler passed to *lvsfunc.scale.descale()*.
lvsfunc.misc.colored_clips(amount[, ...])	Returns a list of BlankClips with unique colors in sequential or random order.
lvsfunc.misc.edgefixer(clip[, left, right, ...])	A wrapper for ContinuityFixer (https://github.com/MonoS/VS-ContinuityFixer).
lvsfunc.misc.frames_since_bookmark(clip, ...)	Displays frames since last bookmark to create easily reusable scenefiltering.
lvsfunc.misc.get_matrix(clip)	Helper function to get the matrix for a clip.
lvsfunc.misc.limit_dark(clip, filtered[, ...])	Replaces frames in a clip with a filtered clip when the frame's darkness exceeds the threshold.
lvsfunc.misc.load_bookmarks(bookmark_path)	VSEdit bookmark loader.
lvsfunc.misc.replace_ranges(clip_a, clip_b, ...)	A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string.
lvsfunc.misc.scale_thresh(thresh, clip[, assume])	Scale binarization thresholds from float to int.
lvsfunc.misc.shift_tint(clip[, values])	A function for forcibly adding pixel values to a clip.
lvsfunc.misc.source(file[, ref, ...])	Generic clip import function.
lvsfunc.misc.wipe_row(clip[, secondary, ...])	Simple function to wipe a row with a blank clip.

Miscellaneous functions and wrappers that didn't really have a place in any other submodules.

lvsfunc.misc.**allow_variable**(*width=None*, *height=None*, *format=None*)

Decorator allowing a variable-res and/or variable-format clip to be passed to a function that otherwise would not be able to accept it. Implemented by FrameEvaling and resizing the clip to each frame. Does not work when the function needs to return a different format unless an output format is specified. As such, this decorator must be called as a function when used (e.g. `@allow_variable()` or `@allow_variable(format=vs.GRAY16)`). If the provided clip is variable format, no output format is required to be specified.

Parameters

- **width** (Optional[int]) – Output clip width
- **height** (Optional[int]) – Output clip height
- **format** (Optional[int]) – Output clip format

Return type Callable[[Callable[..., VideoNode]], Callable[..., VideoNode]]

Returns Function decorator for the given output format.

lvsfunc.misc.**chroma_injector**(*func*)

Decorator allowing injection of reference chroma into a function which would normally only receive luma, such as an upscaler passed to *lvsfunc.scale.descale()*. The chroma is resampled to the input clip's width, height, and pixel format, shuffled to YUV444PX, then passed to the function. Luma is then extracted from the function result and returned. The first argument of the function is assumed to be the luma source. This works with variable resolution and may work with variable format, however the latter is wholly untested and likely a bad idea in every conceivable use case.

Parameters **func** (*~F*) – Function to call with injected chroma

Return type ~F

Returns Decorated function

lvsfunc.misc.**colored_clips**(*amount*, *max_hue=300*, *rand=True*, *seed=None*, ***kwargs*)

Returns a list of BlankClips with unique colors in sequential or random order. The colors will be evenly spaced by hue in the HSL colorspace.

Useful maybe for comparison functions or just for getting multiple uniquely colored BlankClips for testing purposes.

Will always return a pure red clip in the list as this is the RGB equivalent of the lowest HSL hue possible (0).

Written by Dave <orangechannel@pm.me>.

Parameters

- **amount** (int) – Number of vapoursynth.VideoNodes to return
- **max_hue** (int) – Maximum hue (0 < hue <= 360) in degrees to generate colors from (uses the HSL color model). Setting this higher than 315 will result in the clip colors looping back towards red and is not recommended for visually distinct colors. If the *amount* of clips is higher than the *max_hue* expect there to be identical or visually similar colored clips returned (Default: 300)
- **rand** (bool) – Randomizes order of the returned list (Default: True)
- **seed** (Union[bytearray, bytes, float, int, str, None]) – Bytes-like object passed to random.seed which allows for consistent randomized order of the resulting clips (Default: None)
- **kwargs** (Any) – Arguments passed to vapoursynth.core.std.BlankClip (Default: keep=1)

Return type List[VideoNode]

Returns List of uniquely colored clips in sequential or random order.

lvsfunc.misc.**edgefixer**(*clip*, *left=None*, *right=None*, *top=None*, *bottom=None*, *radius=None*, *full_range=False*)

A wrapper for ContinuityFixer (https://github.com/MonoS/VS-ContinuityFixer).

Fixes the issues with over- and undershoot that it may create when fixing the edges, and adds what are in my opinion "more sane" ways of handling the parameters and given values.

. . . If possible, you should be using bbmod instead, though.

Alias for this function is *lvsfunc.ef*.

Dependencies: * VS-ContinuityFixer

Parameters

- **clip** (VideoNode) – Input clip
- **left** (Union[int, List[int], None]) – Number of pixels to fix on the left (Default: None)
- **right** (Union[int, List[int], None]) – Number of pixels to fix on the right (Default: None)
- **top** (Union[int, List[int], None]) – Number of pixels to fix on the top (Default: None)
- **bottom** (Union[int, List[int], None]) – Number of pixels to fix on the bottom (Default: None)
- **radius** (Optional[List[int]]) – Radius for edgefixing (Default: None)
- **full_range** (bool) – Does not run the expression over the clip to fix over/undershoot (Default: False)

Return type VideoNode

Returns Clip with fixed edges

lvsfunc.misc.**frames_since_bookmark**(*clip*, *bookmarks*)

Displays frames since last bookmark to create easily reusable scenefiltering. Can be used in tandem with *lvsfunc.misc.load_bookmarks()* to import VSEdit bookmarks.

Parameters

- **clip** (VideoNode) – Input clip
- **bookmarks** (List[int]) – A list of bookmarks

Return type VideoNode

Returns Clip with bookmarked frames

lvsfunc.misc.**get_matrix**(*clip*)

Helper function to get the matrix for a clip.

Parameters **clip** (VideoNode) – src clip

Return type int

Returns Value representing a matrix

lvsfunc.misc.**limit_dark**(*clip*, *filtered*, *threshold=0.25*, *threshold_range=None*)

Replaces frames in a clip with a filtered clip when the frame's darkness exceeds the threshold. This way you can run lighter (or heavier) filtering on scenes that are almost entirely dark.

There is one caveat, however: You can get scenes where every other frame is filtered rather than the entire scene. Please do take care to avoid that if possible.

Parameters

- **clip** (VideoNode) – Input clip
- **filtered** (VideoNode) – Filtered clip
- **threshold** (float) – Threshold for frame averages to be filtered (Default: 0.25)
- **threshold_range** (Optional[int]) – Threshold for a range of frame averages to be filtered (Default: None)

Return type VideoNode

Returns Conditionally filtered clip

lvsfunc.misc.**load_bookmarks**(*bookmark_path*)
VSEdit bookmark loader.

load_bookmarks(os.path.basename(__file__)+".bookmarks") will load the VSEdit bookmarks for the current Vapoursynth script.

Parameters **bookmark_path** (str) – Path to bookmarks file

Return type List[int]

Returns A list of bookmarked frames

lvsfunc.misc.**replace_ranges**(*clip_a*, *clip_b*, *ranges*)
A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string. Frame ranges are inclusive.

Written by louis.

Alias for this function is *lvsfunc.rfs*.

Parameters

- **clip_a** (VideoNode) – Original clip
- **clip_b** (VideoNode) – Replacement clip
- **ranges** (Union[int, Tuple[int, int], List[Union[int, Tuple[int, int]]]]) – Ranges to replace clip_a (original clip) with clip_b (replacement clip). Integer values in the list indicate single frames, Tuple values indicate inclusive ranges.

Return type VideoNode

Returns Clip with ranges from clip_a replaced with clip_b

lvsfunc.misc.**scale_thresh**(*thresh*, *clip*, *assume=None*)
Scale binarization thresholds from float to int.

Parameters

- **thresh** (float) – Threshold [0, 1]. If greater than 1, assumed to be in native clip range
- **clip** (VideoNode) – Clip to scale to
- **assume** (Optional[int]) – Assume input is this depth when given input >1. If None, assume clip's format. (Default: None)

Return type float

Returns Threshold scaled to [0, 2^clip.depth - 1] (if vs.INTEGER)

lvsfunc.misc.**shift_tint**(*clip*, *values=16*)
A function for forcibly adding pixel values to a clip. Can be used to fix green tints in Crunchyroll sources, for example. Only use this if you know what you're doing!

This function accepts a single integer or a list of integers. Values passed should mimic those of an 8bit clip. If your clip is not 8bit, they will be scaled accordingly.

If you only pass 1 value, it will copied to every plane. If you pass 2, the 2nd one will be copied over to the 3rd. Don't pass more than three.

Parameters

- **clip** (VideoNode) – Input clip
- **values** (Union[int, Sequence[int]]) – Value added to every pixel, scales accordingly to your clip's depth (Default: 16)

Return type VideoNode

Returns Clip with pixel values added

lvsfunc.misc.**source**(*file*, *ref=None*, *force_lsmas=False*, *mpls=False*, *mpls_playlist=0*, *mpls_angle=0*, ***index_args*)

Generic clip import function. Automatically determines if ffms2 or L-SMASH should be used to import a clip, but L-SMASH can be forced. It also automatically determines if an image has been imported. You can set its fps using 'fpsnum' and 'fpsden', or using a reference clip with 'ref'.

Alias for this function is *lvsfunc.src*.

Dependencies: * ffms2 * L-SMASH-Works (optional: m2ts sources or when forcing lsmas) * d2vsource (optional: d2v sources) * dgdecodenv (optional: dgi sources) * VapourSynth-ReadMpls (optional: mpls sources)

Parameters

- **file** (str) – Input file
- **ref** (Optional[VideoNode]) – Use another clip as reference for the clip's format, resolution, and framerate (Default: None)
- **force_lsmas** (bool) – Force files to be imported with L-SMASH (Default: False)
- **mpls** (bool) – Load in a mpls file (Default: False)
- **mpls_playlist** (int) – Playlist number, which is the number in mpls file name (Default: 0)
- **mpls_angle** (int) – Angle number to select in the mpls playlist (Default: 0)
- **kwargs** – Arguments passed to the indexing filter

Return type VideoNode

Returns Vapoursynth clip representing input file

lvsfunc.misc.**wipe_row**(*clip*, *secondary=None*, *width=1*, *height=1*, *offset_x=0*, *offset_y=0*, *width2=None*, *height2=None*, *offset_x2=None*, *offset_y2=None*, *show_mask=False*)

Simple function to wipe a row with a blank clip. You can also give it a different clip to replace a row with.

if width2, height2, etc. are given, it will merge the two masks.

Dependencies: * kagefunc

Parameters

- **clip** (VideoNode) – Input clip
- **secondary** (Optional[VideoNode]) – Clip to replace wiped rows with (Default: None)
- **width** (int) – Width of row (Default: 1)
- **height** (int) – Height of row (Default: 1)
- **offset_x** (int) – X-offset of row (Default: 0)
- **offset_y** (int) – Y-offset of row (Default: 0)
- **width2** (Optional[int]) – Width of row 2 (Default: None)
- **height2** (Optional[int]) – Height of row 2 (Default: None)
- **offset_x2** (Optional[int]) – X-offset of row 2 (Default: None)
- **offset_y2** (Optional[int]) – Y-offset of row 2 (Default: None)

Return type `VideoNode`

Returns Clip with rows wiped

CHAPTER

FOURTEEN

LVSFUNC.RECON

lvsfunc.recon.chroma_reconstruct(clip[, ...])	A function to demangle messed-up chroma, like for example chroma that was downscaled using Nearest Neighbour, or the chroma found on DVDs.

Wrappers and functions for chroma reconstruction. Original functions written by shane on Discord, but since he doesn't seem to be releasing them, I will be the one to do it.

lvsfunc.recon.**ChromaReconstruct**(*clip*, *radius=2*, *i444=False*)

A function to demangle messed-up chroma, like for example chroma that was downscaled using Nearest Neighbour, or the chroma found on DVDs. This function should be used with care, and not blindly applied to anything.

This function can also return a 4:4:4 clip. This is not recommended except for very specific cases, like for example where you're dealing with a razor-sharp 1080p source with a lot of bright colours. Otherwise, have it return the 4:2:0 clip instead.

Original function by shane, modified by Ichunjo and LightArrowsEXE.

Aliases for this function are *lvsfunc.demangle* and *lvsfunc.crecon*.

Parameters

- **clip** (VideoNode) – Input clip
- **radius** (int) – Boxblur radius
- **i444** (bool) – Return a 4:4:4 clip

Return type VideoNode

Returns Clip with demangled chroma in either 4:2:0 or 4:4:4

class lvsfunc.recon.**RegressClips**(*slope*, *intercept*, *correlation*)

Bases: tuple

correlation: vapoursynth.VideoNode
Alias for field number 2

intercept: vapoursynth.VideoNode
Alias for field number 1

slope: vapoursynth.VideoNode
Alias for field number 0

lvsfunc.recon.**chroma_reconstruct**(*clip*, *radius=2*, *i444=False*)

A function to demangle messed-up chroma, like for example chroma that was downscaled using Nearest Neighbour, or the chroma found on DVDs. This function should be used with care, and not blindly applied to anything.

This function can also return a 4:4:4 clip. This is not recommended except for very specific cases, like for example where you're dealing with a razor-sharp 1080p source with a lot of bright colours. Otherwise, have it return the 4:2:0 clip instead.

Original function by shane, modified by Ichunjo and LightArrowsEXE.

Aliases for this function are *lvsfunc.demangle* and *lvsfunc.crecon*.

Parameters

- **clip** (`VideoNode`) – Input clip
- **radius** (`int`) – Boxblur radius
- **i444** (`bool`) – Return a 4:4:4 clip

Return type `VideoNode`

Returns Clip with demangled chroma in either 4:2:0 or 4:4:4

CHAPTER

FIFTEEN

LVSFUNC.RENDER

lvsfunc.render.clip_async_render(clip[, ...])	Render a clip by requesting frames asynchronously using clip.get_frame_async, providing for callback with frame number and frame object.
lvsfunc.render.find_scene_changes(clip[, mode])	Generate a list of scene changes (keyframes).

Clip rendering helpers.

class lvsfunc.render.**RenderContext**(*clip*, *queued*)

Bases: object

Contains info on the current render operation.

clip: vapoursynth.VideoNode

condition: threading.Condition

frames: Dict[int, vapoursynth.VideoFrame]

frames_rendered: int

queued: int

timecodes: List[float]

class lvsfunc.render.**SceneChangeMode**(*value*)

Bases: enum.Enum

An enumeration.

SCXVID = 1

WWXD = 0

WWXD_SCXVID_INTERSECTION = 3

WWXD_SCXVID_UNION = 2

lvsfunc.render.**clip_async_render**(*clip*, *outfile=None*, *timecodes=None*, *callback=None*)

Render a clip by requesting frames asynchronously using clip.get_frame_async, providing for callback with frame number and frame object.

This is mostly a re-implementation of VideoNode.output, but a little bit slower since it's pure python. You only really need this when you want to render a clip while operating on each frame in order or you want timecodes without using vspipe.

Parameters

- **clip** (VideoNode) – Clip to render.

- **outfile** (Optional[BinaryIO]) – Y4MPEG render output BinaryIO handle. If None, no Y4M output is performed. Use sys.stdout.buffer for stdout. (Default: None)
- **timecodes** (Optional[TextIO]) – Timecode v2 file TextIO handle. If None, timecodes will not be written.
- **callback** (Union[Callable[[int, VideoFrame], None], List[Callable[[int, VideoFrame], None]], None]) – Single or list of callbacks to be preformed. The callbacks are called when each sequential frame is output, not when each frame is done. Must have signature Callable[[int, vs.VideoNode], None] See *lvsfunc.comparison.diff()* for a use case (Default: None).

Return type List[float]

Returns List of timecodes from rendered clip.

lvsfunc.render.**find_scene_changes**(*clip*, *mode=<SceneChangeMode.WWXD: 0>*)

Generate a list of scene changes (keyframes).

Dependencies: * vapoursynth-wwxd * vapoursynth-scxvid (Optional: scxvid mode)

Parameters

- **clip** (VideoNode) – Clip to search for scene changes. Will be rendered in its entirety.
- **mode** (*SceneChangeMode*) – Scene change detection mode:
  - WWXD: Use wwxd
  - SCXVID: Use scxvid
  - WWXD_SCXVID_UNION: Union of wwxd and sxcvid (must be detected by at least one)
  - WWXD_SCXVID_INTERSECTION: Intersection of wwxd and scxvid (must be detected by both)

Return type List[int]

Returns List of scene changes.

lvsfunc.render.**finish_frame**(*outfile*, *timecodes*, *ctx*)

Output a frame.

Parameters

- **outfile** (Optional[BinaryIO]) – Output IO handle for Y4MPEG
- **timecodes** (Optional[TextIO]) – Output IO handle for timecodesv2
- **ctx** (*RenderContext*) – Rendering context

Return type None

CHAPTER

SIXTEEN

LVSFUNC.SCALE

lvsfunc.scale.descale(clip[, upscaler, ...])	A unified descaling function.
lvsfunc.scale.descale_detail_mask	Generate a detail mask given a clip and a clip rescaled with the same kernel.
lvsfunc.scale.reupscale	A quick 'n easy wrapper used to re-upscale a clip descaled with descale using znedi3.
lvsfunc.scale.test_descale(clip[, width, ...])	Generic function to test descales with; descales and re-upscales a given clip, allowing you to compare the two easily.

class lvsfunc.scale.**Resolution**(*width: int*, *height: int*)

Bases: tuple

Tuple representing a resolution.

width: int
Width.

height: int
Height.

class lvsfunc.scale.**ScaleAttempt**(*descaled: vapoursynth.VideoNode*, *rescaled: vapoursynth.VideoNode*, *resolution:* lvsfunc.scale.Resolution, *diff: vapoursynth.VideoNode*)

Bases: tuple

Tuple representing a descale attempt.

descaled: vapoursynth.VideoNode
Descaled frame in native resolution.

rescaled: vapoursynth.VideoNode
Descaled frame reupscaled with the same kernel.

resolution: *lvsfunc.scale.Resolution*
The native resolution.

diff: vapoursynth.VideoNode
The subtractive difference between the original and descaled frame.

Functions for (de)scaling.

lvsfunc.scale.**descale**(*clip*, *upscaler=<function reupscale>*, *width=None*, *height=720*, *kernel=<lvsfunc.kernels.Bicubic object>*, *threshold=0.0*, *mask=<function descale_detail_mask>*, *src_left=0.0*, *src_top=0.0*, *show_mask=False*)

A unified descaling function. Includes support for handling fractional resolutions (experimental), multiple resolutions, detail masking, and conditional scaling.

If you want to descale to a fractional resolution, set src_left and src_top and round up the target height.

If the source has multiple native resolutions, specify `height` as a list.

If you want to conditionally descale, specify a non-zero threshold.

Dependencies: * vapoursynth-descale * znedi3

Parameters

- **clip** (`VideoNode`) – Clip to descale
- **upscaler** (`Optional[Callable[[VideoNode, int, int], VideoNode]]`) – Callable function with signature upscaler(clip, width, height) -> vs.VideoNode to be used for reupscaling. Must be capable of handling variable res clips for multiple heights and conditional scaling. If a single height is given and upscaler is None, a constant resolution GRAY clip will be returned instead. Note that if upscaler is None, no upscaling will be performed and neither detail masking nor proper fractional descaling can be preformed. (Default: *lvsfunc.scale.reupscale()*)
- **width** (`Union[int, List[int], None]`) – Width to descale to (if None, auto-calculated)
- **height** (`Union[int, List[int]]`) – Height(s) to descale to. List indicates multiple resolutions, the function will determine the best. (Default: 720)
- **kernel** (*Kernel*) – Kernel used to descale (see *lvsfunc.kernels.Kernel*, (Default: kernels.Bicubic(b=0, c=1/2))
- **threshold** (`float`) – Error threshold for conditional descaling (Default: 0.0, always descale)
- **mask** (`Optional[Callable[[VideoNode, VideoNode], VideoNode]]`) – Function used to mask detail. If `None`, no masking. Function must accept a clip and a reupscaled clip and return a mask. (Default: *lvsfunc.scale.descale_detail_mask()*)
- **src_left** (`float`) – Horizontal shifting for fractional resolutions (Default: 0.0)
- **src_top** (`float`) – Vertical shifting for fractional resolutions (Default: 0.0)
- **show_mask** (`bool`) – Return detail mask

Return type `VideoNode`

Returns Descaled and re-upscaled clip with float bitdepth

`lvsfunc.scale.`**descale_detail_mask**

Generate a detail mask given a clip and a clip rescaled with the same kernel.

Function is curried to allow parameter tuning when passing to *lvsfunc.scale.descale()*

Parameters

- **clip** – Original clip
- **rescaled_clip** – Clip downscaled and reupscaled using the same kernel
- **threshold** – Binarization threshold for mask (Default: 0.05)

Returns Mask of lost detail

`lvsfunc.scale.`**reupscale**

A quick ‘n easy wrapper used to re-upscale a clip descaled with descale using znedi3.

Function is curried to allow parameter tuning when passing to *lvsfunc.scale.descale()*

Stolen from Varde with some adjustments made.

Dependencies: * znedi3

Parameters

- **clip** – Input clip
- **width** – Upscale width. If None, determine from *height* assuming 16:9 aspect ratio (Default: None)
- **height** – Upscale height (Default: 1080)
- **kernel** – Kernel used to downscale the doubled clip (see *lvsfunc.kernels.Kernel*, Default: kernels.Bicubic(b=0, c=1/2))
- **kwargs** – Arguments passed to znedi3 (Default: nsize=4, nns=4, qual=2, pscrn=2)

Returns Reupscaled clip

lvsfunc.scale.**test_descale**(*clip*, *width=None*, *height=720*, *kernel=<lvsfunc.kernels.Bicubic object>*, *show_error=True*)

Generic function to test descales with; descales and reupscales a given clip, allowing you to compare the two easily. Also returns a *lvsfunc.scale.ScaleAttempt* with additional information.

When comparing, it is recommended to do atleast a 4x zoom using Nearest Neighbor. I also suggest using 'compare' (*lvsfunc.comparison.compare()*), as that will make comparing the output with the source clip a lot easier.

Some of this code was leveraged from DescaleAA found in fvsfunc.

Dependencies: * vapoursynth-descale

Parameters

- **clip** (VideoNode) – Input clip
- **width** (Optional[int]) – Target descale width. If None, determine from *height*
- **height** (int) – Target descale height (Default: 720)
- **kernel** (*Kernel*) – Kernel used to descale (see *lvsfunc.kernels.Kernel*, Default: kernels.Bicubic(b=0, c=1/2))
- **show_error** (bool) – Render PlaneStatsDiff on the reupscaled frame (Default: True)

Return type Tuple[VideoNode, *ScaleAttempt*]

Returns A tuple containing a clip re-upscaled with the same kernel and a ScaleAttempt tuple.

CHAPTER

SEVENTEEN

LVSFUNC.TYPES

class lvsfunc.types.**Coordinate**(*x*, *y*)
Bases: object

A positive set of (x, y) coodinates.

x: int

y: int

class lvsfunc.types.**Position**(*x*, *y*)
Bases: *lvsfunc.types.Coordinate*

x: int

y: int

class lvsfunc.types.**Size**(*x*, *y*)
Bases: *lvsfunc.types.Coordinate*

x: int

y: int

CHAPTER

EIGHTEEN

LVSFUNC.UTIL

lvsfunc.util.pick_removegrain(clip)	Returns rgvs.RemoveGrain if the clip is 16 bit or lower, else rgsf.RemoveGrain.
lvsfunc.util.pick_repair(clip)	Returns rgvs.Repair if the clip is 16 bit or lower, else rgsf.Repair.
lvsfunc.util.quick_resample(clip, function)	A function to quickly resample to 16/8 bit and back to the original depth.

Helper functions for the main functions in the script.

lvsfunc.util.**get_prop**(*frame*, *key*, *t*)

Gets gets FrameProp `prop` from frame `frame` with expected type `t` to satisfy the type checker.

Parameters

- **frame** (VideoFrame) – Frame containing props
- **key** (str) – Prop to get
- **t** (Type[~T]) – Type of prop

Return type ~T

Returns frame.prop[key]

lvsfunc.util.**pick_removegrain**(*clip*)

Returns rgvs.RemoveGrain if the clip is 16 bit or lower, else rgsf.RemoveGrain. This is done because rgvs doesn't work with float, but rgsf does for whatever reason.

Dependencies: * RGSF

Parameters **clip** (VideoNode) – Input clip

Return type Callable[..., VideoNode]

Returns Appropriate RemoveGrain function for input clip's depth

lvsfunc.util.**pick_repair**(*clip*)

Returns rgvs.Repair if the clip is 16 bit or lower, else rgsf.Repair. This is done because rgvs doesn't work with float, but rgsf does for whatever reason.

Dependencies: rgsf

Parameters **clip** (VideoNode) – Input clip

Return type Callable[..., VideoNode]

Returns Appropriate repair function for input clip's depth

lvsfunc.util.**quick_resample**(*clip*, *function*)

A function to quickly resample to 16/8 bit and back to the original depth. Useful for filters that only work in 16 bit or lower when you're working in float.

Parameters

- **clip** (VideoNode) – Input clip
- **function** (Callable[[VideoNode], VideoNode]) – Filter to run after resampling (accepts and returns clip)

Return type VideoNode

Returns Filtered clip in original depth

CHAPTER

NINETEEN

SPECIAL CREDITS

A special thanks to every contributor that contributed to lvsfunc.

The list of contributors can be found here.

CHAPTER

TWENTY

FOOTER

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

INDEX

N

P

Q

R

S

T

U

V

W

X

Y