lvsfunc

Release 0.4.1

LightArrowsEXE

May 02, 2022

CONTENTS:

1 About 1

2 Dependencies 3

3 Disclaimer 5

4 Modules 7

5 Functions 9

6 lvsfunc.aa 13

7 lvsfunc.comparison 19

8 lvsfunc.deblock 27

9 lvsfunc.dehalo 29

10 lvsfunc.dehardsub 33

11 lvsfunc.deinterlace 37

12 lvsfunc.denoise 43

13 lvsfunc.mask 45

14 lvsfunc.kernels 49

15 lvsfunc.misc 57

16 lvsfunc.recon 65

17 lvsfunc.render 69

18 lvsfunc.scale 71

19 lvsfunc.types 77

20 lvsfunc.util 79

21 Special credits 85

22 Footer 87

Python Module Index **89**

Index **91**

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)!

For further support, drop by *#lvsfunc* in the IEW Discord server.

CHAPTER

TWO

DEPENDENCIES

lvsfunc depends on the following third-party scripts:

- havsfunc
- kagefunc
- vsutil

The following VapourSynth libraries are also required for full functionality:

- akarinVS
- combmask
- d2vsource
- dgdecnv
- ffms2
- fmtconv
- KNLMeansCL
- L-SMASH-Works
- RGSF
- TIVTC
- VapourSynth-Bilateral
- VapourSynth-BM3D
- VapourSynth-descale
- VapourSynth-EEDI3
- VapourSynth-fillborders
- VapourSynth-RemapFrames
- VapourSynth-nnedi3
- VapourSynth-NNEDI3CL
- VapourSynth-ReadMpls
- VapourSynth-Retinex
- vs-ContinuityFixer
- vs-imwri

- 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

DISCLAIMER

Anything **MAY** change at any time. The public API **SHOULD NOT** be considered stable. If you use lvsfunc in any of your projects, consider hardcoding a version requirement.

CHAPTER

FOUR

MODULES

lvsfunc.aa	Anti-aliasing functions and wrappers.
lvsfunc.comparison	Comparison and analysis functions and wrappers.
lvsfunc.deblock	Deblocking functions and wrappers.
lvsfunc.dehalo	Dehaloing functions and wrappers.
lvsfunc.dehardsub	Dehardsubbing functions and wrappers.
lvsfunc.deinterlace	Deinterlacing, IVTC, and post-deinterlacing functions and wrappers.
lvsfunc.denoise	Denoising functions and wrappers.
lvsfunc.kernels	Kernels for VapourSynth internal resizers.
lvsfunc.mask	Masking functions and wrappers.
lvsfunc.misc	Miscellaneous functions and wrappers that don't really have a place elsewhere.
lvsfunc.recon	Chroma reconstruction functions and wrappers.
lvsfunc.render	Clip rendering helpers.
lvsfunc.scale	(De)scaling and conversion functions and wrappers.
lvsfunc.types	Basic types to be used by certain functions.
lvsfunc.util	Helper functions for module functions and wrapper.

CHAPTER

FIVE

FUNCTIONS

lvsfunc.aa.based_aa(clip[, shader_file, ...])	As the name implies, this is a based anti-aliaser.
lvsfunc.aa.clamp_aa(src, weak, strong[, ...])	Clamp stronger AAs to weaker AAs.
lvsfunc.aa.eedi3([opencl])	Generate eedi3 antialiaser.
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()	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.deblock.autodb_dpir(clip[, ...])	A rewrite of fvsfunc.AutoDeblock that uses vspdir instead of dfttest to deblock.
lvsfunc.deblock.vsdpir(clip[, strength, ...])	A simple vs-mlrt DPIR wrapper for convenience.
lvsfunc.dehalo.bidehalo(clip[, ref, sigmaS, ...])	A simple dehaloing function using bilateral and BM3D to remove bright haloing around edges.
lvsfunc.dehalo.fine_dehalo(clip[, ref, rx, ...])	Slight rewrite of fine_dehalo.

continues on next page

Table 1 – continued from previous page

lvsfunc.dehalo.masked_dha(clip[, ref, rx, ...])	A combination of the best of DeHalo_alpha and Blind-DeHalo3, plus a few minor tweaks to the masking.
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.
lvsfunc.deinterlace.deblend(clip[, start, ...])	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.descale_fields(clip[, ...])	Simple descaling wrapper for interwoven upscaled fields.
lvsfunc.deinterlace.fix_telecined_fades(clip)	A filter that gives a mathematically perfect solution to fades made *after* telecining (which made perfect IVTC impossible).
lvsfunc.deinterlace.ivtc_credits(clip, frame_ref)	Deinterlacing function for interlaced credits (60i/30p) on top of telecined video (24p).
lvsfunc.deinterlace.seek_cycle(clip[, ...])	Purely visual tool to view telecining cycles.
lvsfunc.deinterlace.SIVTC(clip[, pattern, ...])	A very simple fieldmatching function.
lvsfunc.deinterlace.TIVTC_VFR(clip[, ...])	Wrapper for performing TFM and TDecimate on a clip that is supposed to be VFR, including generating a metrics/matches/timecodes txt file.
lvsfunc.deinterlace.vinverse(clip[, sstr, ...])	A simple function to clean up residual combing after a deinterlacing pass.
lvsfunc.denoise.bm3d(clip[, sigma, radius, ...])	A wrapper function for the BM3D denoiser.
lvsfunc.mask.BoundingBox(pos, size)	A positional bounding box.
lvsfunc.mask.DeferredMask([ranges, bound, ...])	Deferred masking interface.
lvsfunc.mask.detail_mask(clip[, sigma, rad, ...])	A wrapper for creating a detail mask to be used during denoising and/or debanding.
lvsfunc.mask.detail_mask_neo(clip[, sigma, ...])	A detail mask aimed at preserving as much detail as possible within darker areas, even if it winds up being mostly noise.
lvsfunc.mask.halo_mask(clip[, rad, brz, ...])	A halo mask to catch basic haloing, inspired by the mask from FineDehalo.
lvsfunc.mask.mt_xxpand_multi(clip[, sw, sh, ...])	Mask expanding/inpanding function written by Zastin.
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().

continues on next page

Table 1 – continued from previous page

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.overlay_sign(clip, overlay[, ...])	Wrapper to overlay a logo or sign onto another clip.
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.unsharpen(clip[, strength, ...])	Diff'd unsharpening function.
lvsfunc.misc.wipe_row(clip[, ref, pos, ...])	Simple function to wipe a row or column 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.recon.reconstruct_multi(c, r[, radius])	**rtype** VideoNode
lvsfunc.recon.regress(x, *ys[, radius, eps])	Fit a line for every neighborhood of values of a given size in a clip with corresponding neighborhoods in one or more other clips.
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).
lvsfunc.render.get_render_progress()	**rtype** Progress
lvsfunc.scale.comparative_descale(clip[, ...])	Easy wrapper to descale to SharpBicubic and an additional kernel, compare them, and then pick one or the other.
lvsfunc.scale.comparative_restore(clip[, ...])	Companion function to go with comparative_descale to reupscale the clip for descale detail masking.
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.gamma2linear(clip, curve[, ...])	**rtype** VideoNode
lvsfunc.scale.linear2gamma(clip, curve[, ...])	**rtype** VideoNode
lvsfunc.scale.mixed_rescale(clip[, width, ...])	InsaneAA rewrite to be a much saner and easier to read function.

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.ssim_downsample(clip[, width, ...])	muvsfunc.ssim_downsample rewrite taken from a Vardë gist.
lvsfunc.util.clamp_values(x, max_val, min_val)	Forcibly clamps the given value x to a max and/or min value.
lvsfunc.util.force_mod(x[, mod])	Force output to fit a specific MOD.
lvsfunc.util.get_coefs(curve)	**rtype** *Coefs*
lvsfunc.util.normalize_ranges(clip, ranges)	Normalize `Range`(s) to a list of inclusive positive integer ranges.
lvsfunc.util.padder(clip[, left, right, ...])	Pads out the pixels on the side by the given amount of pixels.
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 32/16/8 bit and back to the original depth in a one-liner.
lvsfunc.util.replace_ranges(clip_a, clip_b, ...)	A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string.
lvsfunc.util.scale_peak(value, peak)	Full-range scale function that scales a value from [0, 255] to [0, peak]
lvsfunc.util.scale_thresh(thresh, clip[, assume])	Scale binarization thresholds from float to int.

CHAPTER

SIX

LVSFUNC.AA

lvsfunc.aa.based_aa(clip[, shader_file, ...])	As the name implies, this is a based anti-aliaser.
lvsfunc.aa.clamp_aa(src, weak, strong[, ...])	Clamp stronger AAs to weaker AAs.
lvsfunc.aa.eedi3([opencl])	Generate eedi3 antialiaser.
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.

Anti-aliasing functions and wrappers.

lvsfunc.aa.**based_aa**(*clip*, *shader_file='FSRCNNX_x2_56-16-4-1.glsl'*, *rfactor=2.0*, *tff=True*, *mask_thr=60*, *show_mask=False*, *lmask=None*, ***eedi3_args*)

As the name implies, this is a based anti-aliaser. Thank you, based Zastin. This relies on FSRCNNX being very sharp, and as such it very much acts like the main "AA" here.

Original function by Zastin, modified by LightArrowsEXE.

Dependencies:

- vapoursynth-eedi3
- vs-placebo

Parameters

- **clip** – Input clip
- **shader_file** – Path to FSRCNNX shader file
- **rfactor** – Image enlargement factor
- **tff** – Top-Field-First if true, Bottom-Field-First if alse
- **mask_thr** – Threshold for the edge mask binarisation. Scaled internally to match bitdepth of clip.
- **show_mask** – Output mask
- **eedi3_args** – Additional args to pass to eedi3
- **lmask** – Line mask clip to use for eedi3

Returns AA'd clip or mask clip

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.**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** – Input clip

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

Returns Antialiased clip

lvsfunc.aa.**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** – Input clip
- **rfactor** – 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** – Target resolution width. If None, determined from *height*
- **height** – Target resolution height (Default: `clip.height`)
- **supersampler** – Supersampler used for upscaling before AA (Default: nnedi3 supersampler)
- **downscaler** – Downscaler to use after supersampling (Default: Bicubic(b=0, c=1/2)
- **aafun** – Function used to antialias after supersampling (Default: eedi3 with nnedi3 sclip)

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** – Input clip
- **rfactor** – 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** – Target resolution width. If None, determined from *height*
- **height** – Target resolution height (Default: clip.height)
- **supersampler** – Supersampler used for upscaling before AA (Default: nnedi3 supersampler)
- **downscaler** – Downscaler to use after supersampling (Default: Bicubic(b=0, c=1/2)

- **aafun** – Function used to antialias after supersampling (Default: eedi3 with nnedi3 sclip)

Returns Antialiased clip

CHAPTER

SEVEN

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()	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.

Comparison and analysis functions and wrappers.

These functions are intended to be used for comparing encodes, filterchains, and other kinds of personal analysis.

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

Bases: abc.ABC

Base class for comparison functions.

Parameters

- **clips** – 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** – 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** – 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** – 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** – 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** – Determines the axis to split the clips on (Default: *lvsfunc.comparison.Direction.HORIZONTAL*)
- **label_alignment** – 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** – 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 of the stack (Default: *lvsfunc.comparison.Direction.HORIZONTAL*)
- **label_alignment** – 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** – 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** – 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** – 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.`**comp**(*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`.

This is not recommended over setting multiple outputs and checking between those, but in the event that is unavailable to you, this function may be useful.

Alias for this function is *lvsfunc.comp*.

Parameters

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

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

`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`.

This is not recommended over setting multiple outputs and checking between those, but in the event that is unavailable to you, this function may be useful.

Alias for this function is *lvsfunc.comp*.

Parameters

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

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

lvsfunc.comparison.**diff**(**clips: vapoursynth.VideoNode, thr: float = 72, height: int = 288, interleave: bool = False, return_ranges: Literal[True] = 'True', exclusion_ranges: Sequence[int | Tuple[int, int]] | None = None, diff_func: Callable[[vapoursynth.VideoNode, vapoursynth.VideoNode], vapoursynth.VideoNode] = <function <lambda>>, **named-clips: vapoursynth.VideoNode*) → Tuple[vapoursynth.VideoNode, List[Tuple[int, int]]]

lvsfunc.comparison.**diff**(**clips: vapoursynth.VideoNode, thr: float = 72, height: int = 288, interleave: bool = False, return_ranges: Literal[False], exclusion_ranges: Sequence[int | Tuple[int, int]] | None = None, diff_func: Callable[[vapoursynth.VideoNode, vapoursynth.VideoNode], vapoursynth.VideoNode] = <function <lambda>>, **namedclips: vapoursynth.VideoNode*) → vapoursynth.VideoNode

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 without returning 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** – Clips for comparison (order is kept)
- **namedclips** – Keyword arguments of *name=clip* for all clips in the comparison. Clips will be labeled at the top left with their *name*.
- **thr** – Threshold, <= 1 uses PlaneStatsDiff, > 1 uses Max/Min. Higher values will catch more differences. Value must be lower than 128
- **height** – Height in px to downscale clips to if *interleave* is `False` (MakeDiff clip will be twice this resolution)
- **interleave** – Return clip as an interleaved comparison (using *lvsfunc.comparison.Interleave*). This will not return a diff clip
- **return_ranges** – Return a list of ranges in addition to the comparison clip
- **exclusion_ranges** – Excludes a list of frame ranges from difference checking output (but not processing)
- **diff_func** – Function for calculating diff in PlaneStatsMin/Max mode

Returns Either an interleaved clip of the differences between the two clips or a stack of both input clips on top of MakeDiff clip. Furthermore, the function will print the ranges of all the diffs found.

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.**scomp**(**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*.

When not using *make_diff*, *Stack* is heavily recommended instead.

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.**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*.

When not using *make_diff*, *Stack* is heavily recommended instead.

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

EIGHT

LVSFUNC.DEBLOCK

lvsfunc.deblock.autodb_dpir(clip[, ...])	A rewrite of fvsfunc.AutoDeblock that uses vspdir instead of dfttest to deblock.
lvsfunc.deblock.vsdpir(clip[, strength, ...])	A simple vs-mlrt DPIR wrapper for convenience.

Deblocking functions and wrappers.

lvsfunc.deblock.**autodb_dpir**(*clip*, *edgevalue=24*, *strs=[30, 50, 75]*, *thrs=[(1.5, 2.0, 2.0), (3.0, 4.5, 4.5), (5.5, 7.0, 7.0)]*, *matrix=None*, *cuda=True*, *write_props=False*, ***vsdpir_args*)

A rewrite of fvsfunc.AutoDeblock that uses vspdir instead of dfttest to deblock.

This function checks for differences between a frame and an edgemask with some processing done on it, and for differences between the current frame and the next frame. For frames where both thresholds are exceeded, it will perform deblocking at a specified strength. This will ideally be frames that show big temporal *and* spatial inconsistencies.

Thresholds and calculations are added to the frameprops to use as reference when setting the thresholds.

Keep in mind that vsdpir is not perfect; it may cause weird, black dots to appear sometimes. If that happens, you can perform a denoise on the original clip (maybe even using vsdpir's denoising mode) and grab the brightest pixels from your two clips. That should return a perfectly fine clip.

Thanks Vardë, louis, setsugen_no_ao!

Dependencies:

- vs-dpir

Parameters

- **clip** – Input clip
- **edgevalue** – Remove edges from the edgemask that exceed this threshold (higher means more edges removed)
- **strs** – A list of DPIR strength values (higher means stronger deblocking). You can pass any arbitrary number of values here. Sane deblocking strenghts lie between 1–20 for most regular deblocking. Going higher than 50 is not recommended outside of very extreme cases. The amount of values in strs and thrs need to be equal.
- **thrs** – A list of thresholds, written as [(EdgeValRef, NextFrameDiff, PrevFrameDiff)]. You can pass any arbitrary number of values here. The amount of values in strs and thrs need to be equal.

- **matrix** – Enum for the matrix of the input clip. See `types.Matrix` for more info. If *None*, gets matrix from the "_Matrix" prop of the clip unless it's an RGB clip, in which case it stays as *None*.
- **cuda** – Use CUDA backend if True, else CPU backend
- **write_props** – Will write verbose props
- **vsdpir_args** – Additional args to pass to `vsdpir`

Returns Deblocked clip

`lvsfunc.deblock.`**vsdpir**(*clip*, *strength=25*, *mode='deblock'*, *matrix=None*, *tiles=None*, *cuda=True*, *i444=False*, *kernel=<lvsfunc.kernels.Bicubic object>*, ***dpir_args*)

A simple vs-mlrt DPIR wrapper for convenience.

You must install vs-mlrt. For more information, see the following links:

- https://github.com/AmusementClub/vs-mlrt
- https://github.com/AmusementClub/vs-mlrt/wiki/DPIR
- https://github.com/AmusementClub/vs-mlrt/releases/latest

Converts to RGB -> runs DPIR -> converts back to original format, and with no subsampling if `i444=True`. For more information, see https://github.com/cszn/DPIR.

Dependencies:

- vs-mlrt

Parameters

- **clip** – Input clip
- **strength** – DPIR strength. Sane values lie between 1–20 for `mode='deblock'`, and 1–3 for `mode='denoise'`
- **mode** – DPIR mode. Valid modes are 'deblock' and 'denoise'.
- **matrix** – Enum for the matrix of the input clip. See `types.Matrix` for more info. If not specified, gets matrix from the "_Matrix" prop of the clip unless it's an RGB clip, in which case it stays as *None*.
- **cuda** – Use CUDA backend if True, else CPU backend
- **i444** – Forces the returned clip to be YUV444PS instead of the input clip's format
- **dpir_args** – Additional args to pass to vs-mlrt. Note: strength, tiles, and model cannot be overridden!

Returns Deblocked or denoised clip in either the given clip's format or YUV444PS

CHAPTER

NINE

LVSFUNC.DEHALO

lvsfunc.dehalo.bidehalo(clip[, ref, sigmaS, ...])	A simple dehaloing function using bilateral and BM3D to remove bright haloing around edges.
lvsfunc.dehalo.fine_dehalo(clip[, ref, rx, ...])	Slight rewrite of fine_dehalo.
lvsfunc.dehalo.masked_dha(clip[, ref, rx, ...])	A combination of the best of DeHalo_alpha and Blind-DeHalo3, plus a few minor tweaks to the masking.

Dehaloing functions and wrappers.

lvsfunc.dehalo.**bidehalo**(*clip*, *ref=None*, *sigmaS=1.5*, *sigmaR=0.0196078431372549*, *sigmaS_final=None*, *sigmaR_final=None*, *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** – Source clip
- **ref** – Ref clip
- **sigmaS** – Bilateral's spatial weight sigma
- **sigmaR** – Bilateral's range weight sigma
- **sigmaS_final** – Final bilateral call's spatial weight sigma. You'll want this to be much weaker than the initial *sigmaS*. If *None*, 1/3rd of *sigmaS*.
- **sigmaR_final** – Bilateral's range weight sigma. if *None*, same as *sigmaR*
- **bilateral_args** – Additional parameters to pass to bilateral
- **bm3d_args** – Additional parameters to pass to lvsfunc.denoise.bm3d

Returns Dehalo'd clip

lvsfunc.dehalo.**fine_dehalo**(*clip*, *ref=None*, *rx=1.8*, *ry=1.8*, *brightstr=1.0*, *darkstr=0.0*, *thmi=80*, *thma=128*, *thlimi=50*, *thlima=100*, *lowsens=50*, *highsens=50*, *rfactor=1.25*, *show_mask=False*)

Slight rewrite of fine_dehalo.

This is a slight rewrite of the standalone script that has been floating around with support for a `ref` clip. Original can be found in havsfunc if requested.

There have been changes made to the way the masks are expanded/inpanded, as well as strengths. This isn't strictly better or worse than the original version, just different.

This function is rather sensitive to the rx and ry settings. Set them as low as possible! If the radii are set too high, it will start missing small spots.

darkstr is set to 0 by default in this function. This is because more often than not, it simply does more damage than people will likely want.

The sensitivity settings are rather difficult to define. In essence, they define the window between how weak an effect is for it to be processed, and how strong it has to be before it's fully discarded.

Parameters

- **clip** – Input clip
- **ref** – Reference clip. Will replace regular dehaloing.
- **rx** – Horizontal radius for halo removal. Must be greater than 1.
- **ry** – Vertical radius for halo removal. Must be greater than 1.
- **brightstr** – Strength for bright halo removal
- **darkstr** – Strength for dark halo removal. Must be between 0 and 1.
- **thmi** – Minimum threshold for sharp edges. Keep only the sharpest edges (line edges). To see the effects of this setting take a look at the strong mask (show_mask=4).
- **thma** – Maximum threshold for sharp edges. Keep only the sharpest edges (line edges). To see the effects of this setting take a look at the strong mask (show_mask=4).
- **thlimi** – Minimum limiting threshold. Includes more edges than previously, but ignores simple details.
- **thlima** – Maximum limiting threshold. Includes more edges than previously, but ignores simple details.
- **lowsens** – Lower sensitivity range. The lower this is, the more it will process. Must be between 0 and 100.
- **highsens** – Upper sensitivity range. The higher this is, the more it will process. Must be between 0 and 100.
- **rfactor** – Image enlargment factor. Set to >1 to enable some form of aliasing-protection. Must be greater than 1.
- **show_mask** – Return mask clip.

Returns Dehalo'd clip or halo mask clip

lvsfunc.dehalo.**masked_dha**(*clip*, *ref=None*, *rx=2.0*, *ry=2.0*, *brightstr=1.0*, *darkstr=0.0*, *lowsens=50*, *highsens=50*, *rfactor=1.0*, *maskpull=48*, *maskpush=192*, *show_mask=False*)

A combination of the best of DeHalo_alpha and BlindDeHalo3, plus a few minor tweaks to the masking.

This function is rather sensitive to the rx and ry settings. Set them as low as possible! If the radii are set too high, it will start missing small spots.

darkstr is set to 0 by default in this function. This is because more often than not, it simply does more damage than people will likely want.

The sensitivity settings are rather difficult to define. In essence, they define the window between how weak an effect is for it to be processed, and how strong it has to be before it's fully discarded.

Adopted from G41Fun, original by Orum <https://forum.doom9.org/showthread.php?t=148498>. Heavily modified by LightArrowsEXE.

Parameters

- **clip** – Input clip
- **ref** – Reference clip. Will replace regular dehaloing with the given clip.

- **rx** – Horizontal radius for halo removal. Must be greater than 1.
- **ry** – Vertical radius for halo removal. Must be greater than 1.
- **brightstr** – Strength for bright halo removal
- **darkstr** – Strength for dark halo removal. Must be between 0 and 1.
- **lowsens** – Lower sensitivity range. The lower this is, the more it will process. Must be between 0 and 100.
- **highsens** – Upper sensitivity range. The higher this is, the more it will process. Must be between 0 and 100.
- **rfactor** – Image enlargment factor. Set to >1 to enable some form of aliasing-protection. Must be greater than 1.
- **maskpull** – Mask pulling factor
- **maskpush** – Mask pushing factor
- **show_mask** – Return mask clip

Returns Dehalo'd clip or halo mask clip

CHAPTER

TEN

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=None*, *bound=None*, ***, *blur=False*, *ref-frames=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** – A *lvsfunc.mask.BoundingBox* or a tuple that will be converted. (Default: None, no bounding)
- **blur** – 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** – Hardsubbed source
- **ref** – Non-hardsubbed source
- **partials** – Other hardsubbed sources

Returns Dehardsubbed clip

ranges: List[Range]

bound: BoundingBox | None

refframes: List[int | None]

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** – kgf.hardsubmask expand parameter (Default: clip.width // 200)

expand: **int | None**

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

ref_float: **float**

get_mask(*clip*, *ref*)
Get the bounded mask.

Parameters

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

Return type VideoNode

Returns Bounded mask

Dehardsubbing functions and wrappers.

class lvsfunc.dehardsub.**HardsubASS**(*filename*, **args*, *fontdir=None*, *shift=None*, ***kwargs*)
Bases: *lvsfunc.dehardsub.HardsubMask*

Generate a mask using an ass script, such as for dehardubbing AoD with CR DE.

Parameters

- **filename** – Path to ASS script.
- **fontdir** – Extra fonts path.
- **shift** – Offset to apply to the script, in frames. May misbehave due to timestamp rounding.

filename: **str**

fontdir: **str | None**

shift: **int | None**

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

get_mask(*clip*, *ref*)
Get the bounded mask.

Parameters

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

Return type VideoNode

Returns Bounded mask

ref_float: float

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

Parameters

- **hrdsb** – Hardsubbed source
- **ref** – Reference clip
- **signs** – List of *lvsfunc.dehardsub.HardsubSign* to apply

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

ELEVEN

LVSFUNC.DEINTERLACE

lvsfunc.deinterlace.deblend(clip[, start, ...])	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.descale_fields(clip[, ...])	Simple descaling wrapper for interwoven upscaled fields.
lvsfunc.deinterlace.fix_telecined_fades(clip)	A filter that gives a mathematically perfect solution to fades made *after* telecining (which made perfect IVTC impossible).
lvsfunc.deinterlace.ivtc_credits(clip, frame_ref)	Deinterlacing function for interlaced credits (60i/30p) on top of telecined video (24p).
lvsfunc.deinterlace.seek_cycle(clip[, ...])	Purely visual tool to view telecining cycles.
lvsfunc.deinterlace.SIVTC(clip[, pattern, ...])	A very simple fieldmatching function.
lvsfunc.deinterlace.TIVTC_VFR(clip[, ...])	Wrapper for performing TFM and TDecimate on a clip that is supposed to be VFR, including generating a metrics/matches/timecodes txt file.
lvsfunc.deinterlace.vinverse(clip[, sstr, ...])	A simple function to clean up residual combing after a deinterlacing pass.

Deinterlacing, IVTC, and post-deinterlacing functions and wrappers.

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.**TIVTC_VFR**(*clip*, *tfm_in='.ivtc/__main___matches.txt'*, *tdec_in='.ivtc/__main___metrics.txt'*, *timecodes_out='.ivtc/__main___timecodes.txt'*, *decimate=True*, *tfm_args={}*, *tdecimate_args={}*)

Wrapper for performing TFM and TDecimate on a clip that is supposed to be VFR, including generating a metrics/matches/timecodes txt file.

Largely based on, if not basically rewritten from, atomchtools.TIVTC_VFR.

Dependencies:

- TIVTC

Parameters

- **clip** – Input clip
- **tfmIn** – File location for TFM's matches analysis
- **tdecIn** – File location for TDecimate's metrics analysis
- **mkvOut** – File location for TDecimate's timecode file output
- **decimate** – Perform TDecimate on the clip if true, else returns TFM'd clip only. Set to -1 to use TDecimate without TFM
- **tfm_args** – Additional arguments to pass to TFM
- **tdecimate_args** – Additional arguments to pass to TDecimate

Returns IVTC'd VFR clip

lvsfunc.deinterlace.**deblend**(*clip*, *start=0*, *rep=None*, *decimate=True*)

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** – Input clip
- **start** – First frame of the pattern (Default: 0)
- **rep** – Repair mode for the deblended frames, no repair if None (Default: None)
- **decimate** – Decimate the video after deblending (Default: True)

Returns Deblended clip

lvsfunc.deinterlace.**decomb**(*clip*, *tff=True*, *mode=1*, *decimate=True*, *vinv=False*, *rep=None*, *show_mask=False*, *tfm_args={}*, *tdec_args={}*, *vinv_args={}*, *qtgmc_args={}*)

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. A reference clip can be passed with *ref*, which will be used by TFM to create the output frames.

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

Dependencies:

- combmask
- havsfunc
- RGSF (optional: 32 bit clip)

Parameters

- **clip** – Input clip
- **tff** – Top-Field-First
- **mode** – Sets the matching mode or strategy to use for TFM
- **decimate** – Decimate the video after deinterlacing (Default: True)
- **vinv** – Use vinverse to get rid of additional combing (Default: False)
- **rep** – Repair mode for repairing the decombed clip using the original clip (Default: None)
- **show_mask** – Return combmask
- **tfm_args** – Arguments to pass to TFM
- **vinv_args** – Arguments to pass to vinverse
- **qtgmc_args** – Arguments to pass to QTGMC

Returns Decombed and optionally decimated clip

lvsfunc.deinterlace.**descale_fields**(*clip*, *tff=True*, *width=None*, *height=720*, *kernel=<lvsfunc.kernels.Bicubic object>*, *src_top=0.0*)

Simple descaling wrapper for interwoven upscaled fields. This function also sets a frameprop with the kernel that was used.

The kernel is set using an lvsfunc.Kernel object. You can call these by doing for example `kernel=lvf.kernels.Bilinear()`. You can also set specific values manually. For example: `kernel=lvf.kernels.Bicubic(b=0, c=1)`. For more information, check the documentation on Kernels.

`src_top` allows you to to shift the clip prior to descaling. This may be useful, as sometimes clips are shifted before or after the original upscaling.

Parameters

- **clip** – Input clip
- **tff** – Top-field-first. *False* sets it to Bottom-Field-First
- **width** – Native width. Will be automatically determined if set to *None*
- **height** – Native height. Will be divided by two internally
- **kernel** – lvsfunc.Kernel object. This can also be a string (default: Catrom)

- **src_top** – Shifts the clip vertically during the descaling

Returns Descaled GRAY clip

lvsfunc.deinterlace.**fix_telecined_fades**(*clip*, *tff=None*, *thr=2.2*)

A filter that gives a mathematically perfect solution to fades made *after* telecining (which made perfect IVTC impossible). This is an improved version of the Fix-Telecined-Fades plugin that deals with overshoot/undershoot by adding a check.

Make sure to run this *after* IVTC/deinterlacing!

If the value surpases thr * original value, it will not affect any pixels in that frame to avoid it damaging frames it shouldn't need to. This helps a lot with orphan fields as well, which would otherwise create massive swings in values, sometimes messing up the fade fixing.

If you pass your own float clip, you'll want to make sure to properly dither it down after. If you don't do this, you'll run into some serious issues!

Taken from this gist and modified by LightArrowsEXE. <https://gist.github.com/blackpilling/bf22846bfaa870a57ad77925c3524eb1>

Parameters

- **clip** – Input clip
- **tff** – Top-field-first. *False* sets it to Bottom-Field-First. If None, get the field order from the _FieldBased prop.
- **thr** – Threshold for when a field should be adjusted. Default is 2.2, which appears to be a safe value that doesn't cause it to do weird stuff with orphan fields.

Returns Clip with only fades fixed

lvsfunc.deinterlace.**ivtc_credits**(*clip*, *frame_ref*, *tff=None*, *interlaced=True*, *dec=None*, *bob_clip=None*, *qtgmc_args={}*)

Deinterlacing function for interlaced credits (60i/30p) on top of telecined video (24p). This is a combination of havsfunc's dec_txt60mc, ivtc_txt30mc, and ivtc_txt60mc functions. The credits are interpolated and decimated to match the output clip.

The function assumes you're passing a telecined clip (that's native 24p). If your clip is already fieldmatched, decimation will automatically be enabled unless set it to False. Likewise, if your credits are 30p (as opposed to 60i), you should set *interlaced* to False.

The recommended way to use this filter is to trim out the area with interlaced credits, apply this function, and *vsutil.insert_clip* the clip back into a properly IVTC'd clip. Alternatively, use *muvsfunc.VFRSplice* to splice the clip back in if you're dealing with a VFR clip.

Parameters

- **clip** – Input clip. Framerate must be 30000/1001.
- **frame_ref** – First frame in the pattern. Expected pattern is ABBCD, except for when `dec` is enabled, in which case it's AABCD.
- **tff** – Top-field-first. *False* sets it to Bottom-Field-First.
- **interlaced** – 60i credits. Set to false for 30p credits.
- **dec** – Decimate input clip as opposed to IVTC. Automatically enabled if certain fieldmatching props are found. Can be forcibly disabled by setting it to *False*.
- **bob_clip** – Custom bobbed clip. If *None*, uses a QTGMC clip. Framerate must be 60000/1001.

- **qtgmc_args** – Arguments to pass on to QTGMC. Accepts any parameter except for FPS-Divisor and TFF.

Returns IVTC'd/decimated clip with deinterlaced credits

lvsfunc.deinterlace.**seek_cycle**(*clip*, *write_props=True*, *scale=- 1*)

Purely visual tool to view telecining cycles. This is purely a visual tool! This function has no matching parameters, just use wobbly instead if you need that.

Displays the current frame, two previous and two future frames, as well as whether they are combed or not.

P indicates a progressive frame, C a combed frame.

Dependencies:

- VapourSynth-TDeintMod

Parameters

- **clip** (VideoNode) – Input clip
- **write_props** (bool) – Write props on frames. Disabling this will also speed up the function.
- **scale** (int) – Integer scaling of all clips. Must be to the power of 2.

Return type VideoNode

Returns Viewing UI for standard telecining cycles

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.**tivtc_vfr**(*clip*, *tfm_in='.ivtc/__main___matches.txt'*, *tdec_in='.ivtc/__main___metrics.txt'*, *timecodes_out='.ivtc/__main___timecodes.txt'*, *decimate=True*, *tfm_args={}*, *tdecimate_args={}*)

Wrapper for performing TFM and TDecimate on a clip that is supposed to be VFR, including generating a metrics/matches/timecodes txt file.

Largely based on, if not basically rewritten from, atomchtools.TIVTC_VFR.

Dependencies:

- TIVTC

Parameters

- **clip** – Input clip

- **tfmIn** – File location for TFM's matches analysis
- **tdecIn** – File location for TDecimate's metrics analysis
- **mkvOut** – File location for TDecimate's timecode file output
- **decimate** – Perform TDecimate on the clip if true, else returns TFM'd clip only. Set to -1 to use TDecimate without TFM
- **tfm_args** – Additional arguments to pass to TFM
- **tdecimate_args** – Additional arguments to pass to TDecimate

Returns IVTC'd VFR clip

lvsfunc.deinterlace.**vinverse**(*clip*, *sstr=2.0*, *amount=128*, *scale=1.5*)

A simple function to clean up residual combing after a deinterlacing pass. This is Setsugen_no_ao's implementation, adopted into lvsfunc.

Parameters

- **clip** (VideoNode) – Input clip.
- **sstr** (float) – Contrasharpening strength. Increase this if you find the decombing blurs the image a bit too much.
- **amount** (int) – Maximum difference allowed between the original pixels and adjusted pixels. Scaled to input clip's depth. Set to 255 to effectively disable this.
- **scale** (float) – Scale amount for vertical sharp * vertical blur.

Return type VideoNode

Returns Clip with residual combing largely removed

CHAPTER

TWELVE

LVSFUNC.DENOISE

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

Denoising functions and wrappers.

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** – Input clip
- **sigma** – Denoising strength for both basic and final estimations
- **radius** – Temporal radius for both basic and final estimations
- **ref** – Reference clip for the final estimation
- **pre** – Prefiltered clip for the basic estimation
- **refine** – Iteration of the final clip. 0 = basic estimation only 1 = basic + final estimation n = basic + n final estimations
- **matrix_s** – Color matrix of the input clip
- **basic_args** – Args to pass to the basic estimation
- **final_args** – Args to pass to the final estimation

Returns Denoised clip

CHAPTER

THIRTEEN

LVSFUNC.MASK

lvsfunc.mask.BoundingBox(pos, size)	A positional bounding box.
lvsfunc.mask.DeferredMask([ranges, bound, ...])	Deferred masking interface.
lvsfunc.mask.detail_mask(clip[, sigma, rad, ...])	A wrapper for creating a detail mask to be used during denoising and/or debanding.
lvsfunc.mask.detail_mask_neo(clip[, sigma, ...])	A detail mask aimed at preserving as much detail as possible within darker areas, even if it winds up being mostly noise.
lvsfunc.mask.halo_mask(clip[, rad, brz, ...])	A halo mask to catch basic haloing, inspired by the mask from FineDehalo.
lvsfunc.mask.mt_xxpand_multi(clip[, sw, sh, ...])	Mask expanding/inpanding function written by Zastin.
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** – 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** – 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=None*, *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** – A *lvsfunc.mask.BoundingBox* or a tuple that will be converted. (Default: `None`, no bounding)
- **blur** – 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[Range]

blur: bool

bound: *BoundingBox* | None

refframes: List[int | None]

get_mask(*clip*, *ref*)

Get the bounded mask.

Parameters

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

Return type `VideoNode`

Returns Bounded mask

Masking functions and wrappers.

`lvsfunc.mask.`**detail_mask**(*clip*, *sigma=None*, *rad=3*, *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.

Dependencies:

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

Parameters

- **clip** – Input clip
- **sigma** – Sigma for Bilateral for pre-blurring (Default: False)
- **rad** – The luma equivalent of gradfun3's "mask" parameter
- **brz_a** – 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** – 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)

Returns Detail mask

lvsfunc.mask.**detail_mask_neo**(*clip*, *sigma=1.0*, *detail_brz=0.05*, *lines_brz=0.08*, *blur_func=None*, *edgemask_func=vapoursynth.core.std.Prewitt*, *rg_mode=17*)

A detail mask aimed at preserving as much detail as possible within darker areas, even if it winds up being mostly noise.

Parameters

- **clip** – Input clip
- **sigma** – Sigma for the detail mask. Higher means more detail and noise will be caught.
- **detail_brz** – Binarizing for the detail mask. Default values assume a 16bit clip, so you may need to adjust it yourself. Will not binarize if set to 0.
- **lines_brz** – Binarizing for the prewitt mask. Default values assume a 16bit clip, so you may need to adjust it yourself. Will not binarize if set to 0.
- **blur_func** – Blurring function used for the detail detection. Must accept the following parameters: `clip`, `ref_clip`, `sigma`. Uses *bilateral.Bilateral* by default.
- **edgemask_func** – Edgemasking function used for the edge detection
- **rg_mode** – Removegrain mode performed on the final output

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

Returns Halo mask

lvsfunc.mask.**maxm**(*clip, sw=1, sh=None, mode=<Shapes.ELLIPSE: 1>, start=0, *, m__imum=vapoursynth.core.std.Maximum, planes=[0, 1, 2], **m_params*)

Mask expanding/inpanding function written by Zastin.

Performs multiple Minimums/Maximums.

lvsfunc.mask.**minm**(*clip, sw=1, sh=None, mode=<Shapes.ELLIPSE: 1>, start=0, *, m__imum=vapoursynth.core.std.Minimum, planes=[0, 1, 2], **m_params*)

Mask expanding/inpanding function written by Zastin.

Performs multiple Minimums/Maximums.

lvsfunc.mask.**mt_xxpand_multi**(*clip, sw=1, sh=None, mode=<Shapes.ELLIPSE: 1>, start=0, m__imum=vapoursynth.core.std.Maximum, planes=[0, 1, 2], **m_params*)

Mask expanding/inpanding function written by Zastin.

Performs multiple Minimums/Maximums.

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

FOURTEEN

LVSFUNC.KERNELS

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

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

Bicubic b=1, c=0

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Bessel**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Bessel kernel.

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

Built-in bicubic resizer. b=0, c=0.5

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

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *shift=(0, 0)*)

Return type VideoNode

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

Kernel inspired by a Didée post. See: https://forum.doom9.org/showthread.php?p=1748922#post1748922.

Bicubic b=-0.5, c=0.25

This is useful for downscaling content, but might not help much with upscaling.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Kernel inspired by DogWay.

Bicubic b=-0.6, c=0.4

This is useful for downscaling content with ringing.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Bicubic b=0, c=1

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Built-in bilinear resizer.

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

Return type VideoNode

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**BlackHarris**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Black-Harris kernel.

class lvsfunc.kernels.**BlackMan**(*taps=4*, ***kwargs*)
Bases: *lvsfunc.kernels.FmtConv*

fmtconv's blackman resizer.

kernel: str = 'blackman'

class lvsfunc.kernels.**BlackManMinLobe**(*taps=4*, ***kwargs*)
Bases: *lvsfunc.kernels.FmtConv*

fmtconv's blackmanminlobe resizer.

kernel: str = 'blackmanminlobe'

class lvsfunc.kernels.**BlackNuttall**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

BlackNuttall kernel.

class lvsfunc.kernels.**Bohman**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Bohman kernel.

class lvsfunc.kernels.**Box**(*taps=4*, ***kwargs*)
Bases: *lvsfunc.kernels.FmtConv*

fmtconv's box resizer.

kernel: str = 'box'

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

Bicubic b=0, c=0.5

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Cosine**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Cosine kernel.

class lvsfunc.kernels.**FlatTop**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

FlatTop kernel.

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

Abstract fmtconv's resizer.

Dependencies:

- fmtconv

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

Return type VideoNode

kernel: str

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Gaussian**(*curve=30*, ***kwargs*)
Bases: *lvsfunc.kernels.FmtConv*

fmtconv's gaussian resizer.

kernel: str = 'gaussian'

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Ginseng**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Ginseng kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Hamming**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Hamming kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Hann**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Hann kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Bicubic b=0, c=0

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Impulse**(*impulse*, *oversample=8*, *taps=1*, ***kwargs*)
Bases: *lvsfunc.kernels.FmtConv*

fmtconv's impulse resizer.

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

Return type VideoNode

kernel: str = 'impulse'

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Return type VideoNode

class lvsfunc.kernels.**Kaiser**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Kaiser kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Lanczos**(*taps=3*, ***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

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**MinSide**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

MinSide kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Bicubic b=1/3, c=1/3

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**NearestNeighbour**(***kwargs*)
Bases: *lvsfunc.kernels.Gaussian*

Nearest Neighbour kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Parzen**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Parzen kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Built-in point resizer.

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

Return type VideoNode

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Quadratic**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Quadratic kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Bicubic b=0.37822, c=0.31089

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Bicubic b=0.26201, c=0.36899

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

Bicubic b=0.67962, c=0.16019

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Sinc**(*taps=4*, ***kwargs*)
Bases: *lvsfunc.kernels.FmtConv*

fmtconv's gaussian resizer.

kernel: str = 'sinc'

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

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

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *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

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *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

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

resample(*clip*, *format*, *matrix=None*, *matrix_in=None*)

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

Return type VideoNode

shift(*clip*, *shift=(0, 0)*)

Return type VideoNode

class lvsfunc.kernels.**Welch**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Welch kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

class lvsfunc.kernels.**Wiener**(*oversample*, ***kwargs*)
Bases: *lvsfunc.kernels.Impulse*

Wiener kernel.

kwargs: Dict[str, Any]
Arguments passed to the kernel filter

lvsfunc.kernels.**get_all_kernels**()
Get all kernels as a list.

Return type List[Type[Kernel]]

lvsfunc.kernels.**get_kernel**(*name*)
Get a kernel by name.

Parameters **name** (str) – Kernel name.

Return type Type[Kernel]

Returns Kernel class.

CHAPTER

FIFTEEN

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.overlay_sign(clip, overlay[, ...])	Wrapper to overlay a logo or sign onto another clip.
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.unsharpen(clip[, strength, ...])	Diff'd unsharpening function.
lvsfunc.misc.wipe_row(clip[, ref, pos, ...])	Simple function to wipe a row or column with a blank clip.

Miscellaneous functions and wrappers that don't really have a place elsewhere.

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** – Output clip width
- **height** – Output clip height
- **format** – Output clip format

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** – Number of `vapoursynth.VideoNode`s to return
- **max_hue** – 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** – Randomizes order of the returned list (Default: True)
- **seed** – Bytes-like object passed to `random.seed` which allows for consistent randomized order of the resulting clips (Default: None)
- **kwargs** – Arguments passed to `vapoursynth.core.std.BlankClip` (Default: keep=1)

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*.

WARNING: This function may be rewritten in the future, and functionality may change!

Dependencies:

- VS-ContinuityFixer

Parameters

- **clip** – Input clip
- **left** – Number of pixels to fix on the left (Default: None)
- **right** – Number of pixels to fix on the right (Default: None)
- **top** – Number of pixels to fix on the top (Default: None)
- **bottom** – Number of pixels to fix on the bottom (Default: None)
- **radius** – Radius for edgefixing (Default: None)
- **full_range** – Does not run the expression over the clip to fix over/undershoot (Default: False)

Returns Clip with fixed edges

lvsfunc.misc.**ef**(*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*.

WARNING: This function may be rewritten in the future, and functionality may change!

Dependencies:

- VS-ContinuityFixer

Parameters

- **clip** – Input clip
- **left** – Number of pixels to fix on the left (Default: None)
- **right** – Number of pixels to fix on the right (Default: None)
- **top** – Number of pixels to fix on the top (Default: None)
- **bottom** – Number of pixels to fix on the bottom (Default: None)
- **radius** – Radius for edgefixing (Default: None)
- **full_range** – Does not run the expression over the clip to fix over/undershoot (Default: False)

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

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.**overlay_sign**(*clip*, *overlay*, *frame_ranges=None*, *fade_length=0*, *matrix=None*)
Wrapper to overlay a logo or sign onto another clip. Rewrite of fvsfunc.InsertSign. This wrapper also allows you to set fades to fade a logo in and out.

Requires:

- vs-imwri

Parameters

- **clip** – Base clip
- **overlay** – Sign or logo to overlay. Must be the png loaded in through imwri.Read() or a path string to the image file.
- **frame_ranges** – Frame ranges or starting frame to apply the overlay to. See types.Range for more info. If None, overlays the entire clip. If a Range is passed, the overlaid clip will only show up inside that range. If only a single integer is given, it will start on that frame and stay until the end of the clip. Note that this function only accepts a single Range! You can't pass a list of them!
- **fade_length** – Length to fade the clips into each other. The fade will start and end on the frames given in frame_ranges. If set to 0, it won't fade and the sign will simply pop in.

- **matrix** – Enum for the matrix of the input clip. See `types.Matrix` for more info. If not specified, gets matrix from the "_Matrix" prop of the clip unless it's an RGB clip, in which case it stays as *None*.

Returns Clip with a logo or sign overlaid ontop for the given frame ranges, either with or without a fade

`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** – Input clip
- **values** – Value added to every pixel, scales accordingly to your clip's depth (Default: 16)

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*.

WARNING: This function may be rewritten in the future, and functionality may change! No warning is currently printed for this in your terminal to avoid spam.

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

Returns Vapoursynth clip representing input file

lvsfunc.misc.**src**(*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*.

WARNING: This function may be rewritten in the future, and functionality may change! No warning is currently printed for this in your terminal to avoid spam.

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

Returns Vapoursynth clip representing input file

lvsfunc.misc.**unsharpen**(*clip*, *strength=1.0*, *sigma=1.5*, *prefilter=True*, *prefilter_sigma=0.75*)

Diff'd unsharpening function. Performs one-dimensional sharpening as such: "Original + (Original - blurred) * Strength" It then merges back noise and detail that was prefiltered away,

Negative values will blur instead. This can be useful for trying to undo sharpening.

This function is not recommended for normal use, but may be useful as prefiltering for detail- or edgemasks.

Parameters

- **clip** (`VideoNode`) – Input clip.
- **strength** (`float`) – Amount to multiply blurred clip with original clip by. Negative values will blur the clip instead.
- **sigma** (`float`) – Sigma for the gaussian blur.
- **prefilter** (`bool`) – Pre-denoising to prevent the unsharpening from picking up random noise.
- **prefilter_sigma** (`float`) – Strength for the pre-denoising.
- **show_mask** – Show halo mask.

Return type `VideoNode`

Returns Unsharpened clip

lvsfunc.misc.**wipe_row**(*clip*, *ref=None*, *pos=(1, 1)*, *size=None*, *show_mask=False*)

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

Parameters

- **clip** – Input clip
- **secondary** – Clip to replace wiped rows with (Default: None)
- **width** – Width of row (Default: 1)
- **height** – Height of row (Default: 1)
- **offset_x** – X-offset of row (Default: 0)
- **offset_y** – Y-offset of row (Default: 0)

Returns Clip with given rows or columns wiped

CHAPTER

SIXTEEN

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.
lvsfunc.recon.reconstruct_multi(c, r[, radius])	**rtype** `VideoNode`
lvsfunc.recon.regress(x, *ys[, radius, eps])	Fit a line for every neighborhood of values of a given size in a clip with corresponding neighborhoods in one or more other clips.

Chroma reconstruction functions and wrappers.

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

lvsfunc.recon.**crecon**(*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

lvsfunc.recon.**demangle**(*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

lvsfunc.recon.**reconstruct_multi**(*c*, *r*, *radius=2*)

Return type VideoNode

lvsfunc.recon.**regress**(*x*, **ys*, *radius=2*, *eps=1e-07*)

Fit a line for every neighborhood of values of a given size in a clip with corresponding neighborhoods in one or more other clips.

More info: https://en.wikipedia.org/wiki/Simple_linear_regression

Return type List[*RegressClips*]

CHAPTER

SEVENTEEN

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).
lvsfunc.render.get_render_progress()	**rtype** Progress

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*, *progress='Rendering clip...'*, *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** – Clip to render.
- **outfile** – Y4MPEG render output BinaryIO handle. If None, no Y4M output is performed. Use `sys.stdout.buffer` for stdout. (Default: None)
- **timecodes** – Timecode v2 file TextIO handle. If None, timecodes will not be written.
- **progress** – String to use for render progress display. If empty or `None`, no progress display.
- **callback** – 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).

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** – Output IO handle for Y4MPEG
- **timecodes** – Output IO handle for timecodesv2
- **ctx** – Rendering context

lvsfunc.render.**get_render_progress**()

Return type `Progress`

CHAPTER

EIGHTEEN

LVSFUNC.SCALE

lvsfunc.scale.comparative_descale(clip[, ...])	Easy wrapper to descale to SharpBicubic and an additional kernel, compare them, and then pick one or the other.
lvsfunc.scale.comparative_restore(clip[, ...])	Companion function to go with comparative_descale to reupscale the clip for descale detail masking.
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.gamma2linear(clip, curve[, ...])	**rtype** VideoNode
lvsfunc.scale.linear2gamma(clip, curve[, ...])	**rtype** VideoNode
lvsfunc.scale.mixed_rescale(clip[, width, ...])	InsaneAA rewrite to be a much saner and easier to read function.
lvsfunc.scale.reupscale	A quick 'n easy wrapper used to re-upscale a clip descaled with descale using znedi3.
lvsfunc.scale.ssim_downsample(clip[, width, ...])	muvsfunc.ssim_downsample rewrite taken from a Vardë gist.

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: vs.VideoNode*, *rescaled: vs.VideoNode*, *resolution: Resolution*, *diff: vs.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.

(De)scaling and conversion functions and wrappers.

lvsfunc.scale.**comparative_descale**(*clip*, *width=None*, *height=720*, *kernel=None*, *thr=5e-08*)
Easy wrapper to descale to SharpBicubic and an additional kernel, compare them, and then pick one or the other.

The output clip has props that can be used to frameeval specific kernels by the user.

Parameters

- **clip** – Input clip
- **width** – Width to descale to (if None, auto-calculated)
- **height** – Descale height
- **kernel** – Kernel to compare BicubicSharp to. This can also be a string (Default: Spline36 if None).
- **thr** – Threshold for which kernel to pick

Returns Descaled clip

lvsfunc.scale.**comparative_restore**(*clip*, *width=None*, *height=720*, *kernel=None*)
Companion function to go with comparative_descale to reupscale the clip for descale detail masking.

Parameters

- **clip** – Input clip
- **width** – Width to upscale to (if None, auto-calculated)
- **height** – Upscale height
- **kernel** – Kernel to compare BicubicSharp to (Default: Spline36 if None)

Returns Reupscaled clip

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** – Clip to descale

- **upscaler** – 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** – Width(s) to descale to (if None, auto-calculated)
- **height** – Height(s) to descale to. List indicates multiple resolutions, the function will determine the best. (Default: 720)
- **kernel** – Kernel used to descale the clip. This can also be a string. (see `lvsfunc.kernels.Kernel`, (Default: kernels.Bicubic(b=0, c=1/2))
- **threshold** – Error threshold for conditional descaling (Default: 0.0, always descale)
- **mask** – Function or mask clip 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** – Horizontal shifting for fractional resolutions (Default: 0.0)
- **src_top** – Vertical shifting for fractional resolutions (Default: 0.0)
- **show_mask** – Return detail mask

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.**gamma2linear**(*clip*, *curve*, *gcor=1.0*, *sigmoid=False*, *thr=0.5*, *cont=6.5*, *epsilon=1e-06*)

Return type `VideoNode`

lvsfunc.scale.**linear2gamma**(*clip*, *curve*, *gcor=1.0*, *sigmoid=False*, *thr=0.5*, *cont=6.5*)

Return type `VideoNode`

lvsfunc.scale.**mixed_rescale**(*clip*, *width=None*, *height=720*, *kernel=<lvsfunc.kernels.Bicubic object>*, *downscaler=<function ssim_downsample>*, *credit_mask=<function descale_detail_mask>*, *mask_thr=0.05*, *mix_strength=0.25*, *show_mask=False*, *nnedi3_args={}*, *eedi3_args={}*)

InsaneAA rewrite to be a much saner and easier to read function. Descales and downscales the given clip and merges them together with a set strength.

This can be useful for dealing with a source that you can't accurately descale, but you still want to force it. Not recommended to use it on everything, however.

A string can be passed instead of a Kernel object if you want to use that. This gives you access to every kernel object in *lvsfunc.kernels()*. For more information on what every kernel does, please refer to their documentation.

This is still a work in progress at the time of writing. Please use with care.

Parameters

- **clip** – Input clip.
- **width** – Upscale width. If None, determine from *height* (default: None).
- **height** – Upscale height (Default: 720).
- **kernel** – Kernel used to descale the clip. This can also be a string. (see `lvsfunc.kernels.Kernel`, Default: kernels.Bicubic(b=0, c=1/2) (Catrom)).
- **downscaler** – Kernel or custom scaler used to downscale the clip. This can also be a string. (see `lvsfunc.kernels.Kernel`, Default: `ssim_downsample`).
- **credit_mask** – Function or mask clip 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()*).
- **mask_thr** – Binarization threshold for *lvsfunc.scale.descale_detail_mask()* (Default: 0.05).
- **mix_strength** – Merging strength between the descaled and downscaled clip. Stronger values will make the lineart look closer to the downscaled clip. This can get pretty dangerous very quickly if you use a sharp `downscaler`!
- **show_mask** – Return the `credit_mask`. If set to *2*, it will return the lineart mask instead.
- **nnedi3_args** – Additional args to pass to nnedi3.
- **eedi3_args** – Additional args to pass to eedi3.

Returns Rescaled clip with a downscaled clip merged with it and credits masked.

`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.**ssim_downsample**(*clip*, *width=None*, *height=720*, *smooth=0.816496580927726*, *kernel=<lvsfunc.kernels.Catrom object>*, *gamma=False*, *curve=vapoursynth.TransferCharacteristics.TRANSFER_BT709*, *sigmoid=False*, *epsilon=1e-06*)

muvsfunc.ssim_downsample rewrite taken from a Vardë gist. Unlike muvsfunc's implementation, this function also works in float and does not use nnedi3_resample. Most of the documentation is taken from muvsfunc.

SSIM downsampler is an image downscaling technique that aims to optimize for the perceptual quality of the downscaled results. Image downscaling is considered as an optimization problem where the difference between the input and output images is measured using famous Structural SIMilarity (SSIM) index. The solution is derived in closed-form, which leads to the simple, efficient implementation. The downscaled images retain perceptually important features and details, resulting in an accurate and spatio-temporally consistent representation of the high resolution input.

Original gist: https://gist.github.com/Ichunjo/16ab1f893588aafcb096c1f35a0cfb15

Parameters

- **clip** – Input clip
- **width** – Output width. If None, autocalculates using height
- **height** – Output height (default: 720)
- **smooth** – Image smoothening method. If you pass an int, it specifies the "radius" of the internel used boxfilter, i.e. the window has a size of (2*smooth+1)x(2*smooth+1). If you pass a float, it specifies the "sigma" of core.tcanny.TCanny, i.e. the standard deviation of gaussian blur. If you pass a function, it acts as a general smoother. Default uses a gaussian blur.
- **curve** – Gamma mapping
- **sigmoid** – When True, applies a sigmoidal curve after the power-like curve (or before when converting from linear to gamma-corrected). This helps reduce the dark halo artefacts found around sharp edges caused by resizing in linear luminance.
- **epsilon** – Machine epsilon

Returns Downsampled clip

CHAPTER

NINETEEN

LVSFUNC.TYPES

Basic types to be used by certain functions.

class lvsfunc.types.**Coefs**(*k0*, *phi*, *alpha*, *gamma*)
Bases: tuple

alpha: float
Alias for field number 2

gamma: float
Alias for field number 3

k0: float
Alias for field number 0

phi: float
Alias for field number 1

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

A positive set of (x, y) coordinates.

x: int

y: int

class lvsfunc.types.**Matrix**(*value*)
Bases: enum.IntEnum

Matrix coefficients (ITU-T H.265 Table E.5)

BT2020C = 10

BT2020NC = 9

BT470BG = 5

BT709 = 1

CHROMA_DERIVED_C = 13

CHROMA_DERIVED_NC = 12

FCC = 4

GBR = 0

ICTCP = 14

property RESERVED

Return type NoReturn

RGB = 0

SMPTE170M = 6

SMPTE2085 = 11

SMPTE240M = 7

UNKNOWN = 2

YCGCO = 8

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

class lvsfunc.types.**VSFunction**(**args*, ***kwargs*)

Bases: Protocol

CHAPTER

TWENTY

LVSFUNC.UTIL

lvsfunc.util.clamp_values(x, max_val, min_val)	Forcibly clamps the given value x to a max and/or min value.
lvsfunc.util.force_mod(x[, mod])	Force output to fit a specific MOD.
lvsfunc.util.get_coefs(curve)	**rtype** *Coefs*
lvsfunc.util.normalize_ranges(clip, ranges)	Normalize Range(s) to a list of inclusive positive integer ranges.
lvsfunc.util.padder(clip[, left, right, ...])	Pads out the pixels on the side by the given amount of pixels.
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 32/16/8 bit and back to the original depth in a one-liner.
lvsfunc.util.replace_ranges(clip_a, clip_b, ...)	A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string.
lvsfunc.util.scale_peak(value, peak)	Full-range scale function that scales a value from [0, 255] to [0, peak]
lvsfunc.util.scale_thresh(thresh, clip[, assume])	Scale binarization thresholds from float to int.

Helper functions for module functions and wrapper. Some of these may also be useful for regular scripting or other modules.

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

Return type None

lvsfunc.util.**clamp_values**(*x*, *max_val*, *min_val*)

Forcibly clamps the given value x to a max and/or min value.

Return type float

lvsfunc.util.**force_mod**(*x*, *mod=4*)

Force output to fit a specific MOD. Minimum returned value will always be mod^2.

Return type int

lvsfunc.util.**get_coefs**(*curve*)

Return type *Coefs*

lvsfunc.util.**get_neutral_value**(*clip*, *chroma=False*)

Taken from vsutil. This isn't in any new versions yet, so mypy complains. Will remove once vsutil does another version bump.

Returns the neutral value for the combination of the plane type and bit depth/type of the clip as float.

Parameters

- **clip** (VideoNode) – Input clip.
- **chroma** (bool) – Whether to get luma or chroma plane value

Return type float

Returns Neutral value.

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

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.**normalize_ranges**(*clip*, *ranges*)

Normalize Range(s) to a list of inclusive positive integer ranges.

Parameters

- **clip** – Reference clip used for length.
- **ranges** – Single Range or list of Ranges.

Returns List of inclusive positive ranges.

lvsfunc.util.**padder**(*clip*, *left=32*, *right=32*, *top=32*, *bottom=32*)

Pads out the pixels on the side by the given amount of pixels. For a 4:2:0 clip, the output must be an even resolution.

Parameters

- **clip** (VideoNode) – Input clip
- **left** (int) – Padding added to the left side of the clip
- **right** (int) – Padding added to the right side of the clip
- **top** (int) – Padding added to the top side of the clip
- **bottom** (int) – Padding added to the bottom side of the clip

Return type VideoNode

Returns Padded clip

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 32/16/8 bit and back to the original depth in a one-liner. 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

lvsfunc.util.**replace_ranges**(*clip_a*, *clip_b*, *ranges*, *use_plugin=True*)

A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string. Frame ranges are inclusive. Optionally strings can still be used.

This function will try to call the *VapourSynth-RemapFrames* plugin before doing any of its own processing. This should come with a speed boost, so it's recommended you install it.

Examples with clips `black` and `white` of equal length:

- `replace_ranges(black, white, [(0, 1)])`: replace frames 0 and 1 with `white`
- `replace_ranges(black, white, [(None, None)])`: replace the entire clip with `white`
- `replace_ranges(black, white, [(0, None)])`: same as previous
- `replace_ranges(black, white, [(200, None)])`: replace 200 until the end with `white`
- `replace_ranges(black, white, [(200, -1)])`: replace 200 until the end with `white`, leaving 1 frame of `black`

Dependencies: VapourSynth-RemapFrames

Parameters

- **clip_a** – Original clip
- **clip_b** – Replacement clip
- **ranges** – 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.

  Negative integer values will be wrapped around based on clip_b's length.

None values are context dependent:

- None provided as sole value to ranges: no-op
- Single None value in list: Last frame in clip_b
- None as first value of tuple: 0
- None as second value of tuple: Last frame in clip_b

- **use_plugin** – Use the ReplaceFramesSimple plugin for the rfs call.

Returns Clip with ranges from clip_a replaced with clip_b

lvsfunc.util.**rfs**(*clip_a*, *clip_b*, *ranges*, *use_plugin=True*)

A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string. Frame ranges are inclusive. Optionally strings can still be used.

This function will try to call the *VapourSynth-RemapFrames* plugin before doing any of its own processing. This should come with a speed boost, so it's recommended you install it.

Examples with clips `black` and `white` of equal length:

- `replace_ranges(black, white, [(0, 1)])`: replace frames 0 and 1 with `white`
- `replace_ranges(black, white, [(None, None)])`: replace the entire clip with `white`
- `replace_ranges(black, white, [(0, None)])`: same as previous
- `replace_ranges(black, white, [(200, None)])`: replace 200 until the end with `white`
- `replace_ranges(black, white, [(200, -1)])`: replace 200 until the end with `white`, leaving 1 frame of `black`

Dependencies: VapourSynth-RemapFrames

Parameters

- **clip_a** – Original clip
- **clip_b** – Replacement clip
- **ranges** – 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.

  Negative integer values will be wrapped around based on clip_b's length.

  None values are context dependent:

  - None provided as sole value to ranges: no-op
  - Single None value in list: Last frame in clip_b
  - None as first value of tuple: 0
  - None as second value of tuple: Last frame in clip_b

- **use_plugin** – Use the ReplaceFramesSimple plugin for the rfs call.

Returns Clip with ranges from clip_a replaced with clip_b

lvsfunc.util.**scale_peak**(*value*, *peak*)

Full-range scale function that scales a value from [0, 255] to [0, peak]

Return type `float`

lvsfunc.util.**scale_thresh**(*thresh*, *clip*, *assume=None*)

Scale binarization thresholds from float to int.

Parameters

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

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

CHAPTER

TWENTYONE

SPECIAL CREDITS

A special thanks to every contributor that contributed to lvsfunc.

The list of contributors can be found here.

CHAPTER

TWENTYTWO

FOOTER

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

lvsfunc, 1
lvsfunc.aa, 13
lvsfunc.comparison, 19
lvsfunc.deblock, 27
lvsfunc.dehalo, 29
lvsfunc.dehardsub, 35
lvsfunc.deinterlace, 37
lvsfunc.denoise, 43
lvsfunc.kernels, 49
lvsfunc.mask, 46
lvsfunc.misc, 57
lvsfunc.recon, 65
lvsfunc.render, 69
lvsfunc.scale, 72
lvsfunc.types, 77
lvsfunc.util, 79

INDEX

A

allow_variable() (*in module lvsfunc.misc*), 57
alpha (*lvsfunc.types.Coefs attribute*), 77
apply_dehardsub() (*lvsfunc.dehardsub.HardsubMask method*), 33
autodb_dpir() (*in module lvsfunc.deblock*), 27

B

based_aa() (*in module lvsfunc.aa*), 13
Bessel (*class in lvsfunc.kernels*), 49
Bicubic (*class in lvsfunc.kernels*), 49
BicubicDidee (*class in lvsfunc.kernels*), 49
BicubicDogWay (*class in lvsfunc.kernels*), 50
BicubicSharp (*class in lvsfunc.kernels*), 50
bidehalo() (*in module lvsfunc.dehalo*), 29
Bilinear (*class in lvsfunc.kernels*), 50
BlackHarris (*class in lvsfunc.kernels*), 50
BlackMan (*class in lvsfunc.kernels*), 50
BlackManMinLobe (*class in lvsfunc.kernels*), 50
BlackNuttall (*class in lvsfunc.kernels*), 50
blur (*lvsfunc.dehardsub.HardsubMask attribute*), 34
blur (*lvsfunc.mask.DeferredMask attribute*), 46
bm3d() (*in module lvsfunc.denoise*), 43
Bohman (*class in lvsfunc.kernels*), 51
bound (*lvsfunc.dehardsub.HardsubMask attribute*), 34
bound (*lvsfunc.mask.DeferredMask attribute*), 46
bounded_dehardsub() (*in module lvsfunc.dehardsub*), 36
BoundingBox (*class in lvsfunc.mask*), 45
Box (*class in lvsfunc.kernels*), 51
BSpline (*class in lvsfunc.kernels*), 49
BT2020C (*lvsfunc.types.Matrix attribute*), 77
BT2020NC (*lvsfunc.types.Matrix attribute*), 77
BT470BG (*lvsfunc.types.Matrix attribute*), 77
BT709 (*lvsfunc.types.Matrix attribute*), 77

C

Catrom (*class in lvsfunc.kernels*), 51
check_variable() (*in module lvsfunc.util*), 79
CHROMA_DERIVED_C (*lvsfunc.types.Matrix attribute*), 77
CHROMA_DERIVED_NC (*lvsfunc.types.Matrix attribute*), 77
chroma_injector() (*in module lvsfunc.misc*), 58
chroma_reconstruct() (*in module lvsfunc.recon*), 65
ChromaReconstruct() (*in module lvsfunc.recon*), 65
clamp_aa() (*in module lvsfunc.aa*), 14
clamp_values() (*in module lvsfunc.util*), 79
clip (*lvsfunc.render.RenderContext attribute*), 69
clip() (*lvsfunc.comparison.Comparer property*), 19
clip_async_render() (*in module lvsfunc.render*), 69
Coefs (*class in lvsfunc.types*), 77
colored_clips() (*in module lvsfunc.misc*), 58
comp() (*in module lvsfunc.comparison*), 22
comparative_descale() (*in module lvsfunc.scale*), 72
comparative_restore() (*in module lvsfunc.scale*), 72
compare() (*in module lvsfunc.comparison*), 22
Comparer (*class in lvsfunc.comparison*), 19
condition (*lvsfunc.render.RenderContext attribute*), 69
Coordinate (*class in lvsfunc.types*), 77
correlation (*lvsfunc.recon.RegressClips attribute*), 65
Cosine (*class in lvsfunc.kernels*), 51
crecon() (*in module lvsfunc.recon*), 66

D

deblend() (*in module lvsfunc.deinterlace*), 38
decomb() (*in module lvsfunc.deinterlace*), 38
DeferredMask (*class in lvsfunc.mask*), 45
demangle() (*in module lvsfunc.recon*), 66
descale() (*in module lvsfunc.scale*), 72
descale() (*lvsfunc.kernels.Bicubic method*), 49
descale() (*lvsfunc.kernels.Bilinear method*), 50
descale() (*lvsfunc.kernels.FmtConv method*), 51
descale() (*lvsfunc.kernels.Impulse method*), 52
descale() (*lvsfunc.kernels.Lanczos method*), 53
descale() (*lvsfunc.kernels.Point method*), 53

descale() (*lvsfunc.kernels.Spline16 method*), 54
descale() (*lvsfunc.kernels.Spline36 method*), 55
descale() (*lvsfunc.kernels.Spline64 method*), 55
descale_detail_mask (*in module lvsfunc.scale*), 73
descale_fields() (*in module lvsfunc.deinterlace*), 39
descaled (*lvsfunc.scale.ScaleAttempt attribute*), 71
detail_mask() (*in module lvsfunc.mask*), 46
detail_mask_neo() (*in module lvsfunc.mask*), 46
diff (*lvsfunc.scale.ScaleAttempt attribute*), 72
diff() (*in module lvsfunc.comparison*), 22
diff_hardsub_mask() (*in module lvsfunc.comparison*), 23
Direction (*class in lvsfunc.comparison*), 20

E

edgefixer() (*in module lvsfunc.misc*), 58
eedi3() (*in module lvsfunc.aa*), 14
ef() (*in module lvsfunc.misc*), 59
expand (*lvsfunc.dehardsub.HardsubLine attribute*), 34
expand (*lvsfunc.dehardsub.HardsubSign attribute*), 34
expand (*lvsfunc.dehardsub.HardsubSignKgf attribute*), 34

F

FCC (*lvsfunc.types.Matrix attribute*), 77
filename (*lvsfunc.dehardsub.HardsubASS attribute*), 35
find_scene_changes() (*in module lvsfunc.render*), 70
fine_dehalo() (*in module lvsfunc.dehalo*), 29
finish_frame() (*in module lvsfunc.render*), 70
fix_telecined_fades() (*in module lvsfunc.deinterlace*), 40
FlatTop (*class in lvsfunc.kernels*), 51
FmtConv (*class in lvsfunc.kernels*), 51
fontdir (*lvsfunc.dehardsub.HardsubASS attribute*), 35
force_mod() (*in module lvsfunc.util*), 79
frames (*lvsfunc.render.RenderContext attribute*), 69
frames_rendered (*lvsfunc.render.RenderContext attribute*), 69
frames_since_bookmark() (*in module lvsfunc.misc*), 59

G

gamma (*lvsfunc.types.Coefs attribute*), 77
gamma2linear() (*in module lvsfunc.scale*), 73
Gaussian (*class in lvsfunc.kernels*), 51
GBR (*lvsfunc.types.Matrix attribute*), 77
get_all_kernels() (*in module lvsfunc.kernels*), 56
get_all_masks() (*in module lvsfunc.dehardsub*), 36
get_coefs() (*in module lvsfunc.util*), 79
get_kernel() (*in module lvsfunc.kernels*), 56
get_mask() (*lvsfunc.dehardsub.HardsubLineFade method*), 35
get_mask() (*lvsfunc.dehardsub.HardsubSignFade method*), 35
get_mask() (*lvsfunc.mask.BoundingBox method*), 45
get_mask() (*lvsfunc.mask.DeferredMask method*), 46
get_matrix() (*in module lvsfunc.misc*), 59
get_neutral_value() (*in module lvsfunc.util*), 79
get_progressive_dehardsub() (*lvsfunc.dehardsub.HardsubMask method*), 33
get_prop() (*in module lvsfunc.util*), 80
get_render_progress() (*in module lvsfunc.render*), 70
Ginseng (*class in lvsfunc.kernels*), 52

H

halo_mask() (*in module lvsfunc.mask*), 47
Hamming (*class in lvsfunc.kernels*), 52
Hann (*class in lvsfunc.kernels*), 52
hardsub_mask() (*in module lvsfunc.dehardsub*), 36
HardsubASS (*class in lvsfunc.dehardsub*), 35
HardsubLine (*class in lvsfunc.dehardsub*), 34
HardsubLineFade (*class in lvsfunc.dehardsub*), 35
HardsubMask (*class in lvsfunc.dehardsub*), 33
HardsubSign (*class in lvsfunc.dehardsub*), 34
HardsubSignFade (*class in lvsfunc.dehardsub*), 35
HardsubSignKgf (*class in lvsfunc.dehardsub*), 34
height (*lvsfunc.scale.Resolution attribute*), 71
Hermite (*class in lvsfunc.kernels*), 52
highpass (*lvsfunc.dehardsub.HardsubSignKgf attribute*), 34
HORIZONTAL (*lvsfunc.comparison.Direction attribute*), 20

I

ICTCP (*lvsfunc.types.Matrix attribute*), 77
Impulse (*class in lvsfunc.kernels*), 52
inflate (*lvsfunc.dehardsub.HardsubSign attribute*), 34
intercept (*lvsfunc.recon.RegressClips attribute*), 65
Interleave (*class in lvsfunc.comparison*), 20
interleave() (*in module lvsfunc.comparison*), 24
ivtc_credits() (*in module lvsfunc.deinterlace*), 40

K

k0 (*lvsfunc.types.Coefs attribute*), 77
Kaiser (*class in lvsfunc.kernels*), 52
kernel (*lvsfunc.kernels.BlackMan attribute*), 50
kernel (*lvsfunc.kernels.BlackManMinLobe attribute*), 50
kernel (*lvsfunc.kernels.Box attribute*), 51
kernel (*lvsfunc.kernels.FmtConv attribute*), 51
kernel (*lvsfunc.kernels.Gaussian attribute*), 51
kernel (*lvsfunc.kernels.Impulse attribute*), 52
kernel (*lvsfunc.kernels.Sinc attribute*), 54

kwargs (*lvsfunc.kernels.Bicubic attribute*), 49
kwargs (*lvsfunc.kernels.BicubicDidee attribute*), 50
kwargs (*lvsfunc.kernels.BicubicDogWay attribute*), 50
kwargs (*lvsfunc.kernels.BicubicSharp attribute*), 50
kwargs (*lvsfunc.kernels.Bilinear attribute*), 50
kwargs (*lvsfunc.kernels.BSpline attribute*), 49
kwargs (*lvsfunc.kernels.Catrom attribute*), 51
kwargs (*lvsfunc.kernels.Gaussian attribute*), 51
kwargs (*lvsfunc.kernels.Ginseng attribute*), 52
kwargs (*lvsfunc.kernels.Hamming attribute*), 52
kwargs (*lvsfunc.kernels.Hann attribute*), 52
kwargs (*lvsfunc.kernels.Hermite attribute*), 52
kwargs (*lvsfunc.kernels.Impulse attribute*), 52
kwargs (*lvsfunc.kernels.Kaiser attribute*), 52
kwargs (*lvsfunc.kernels.Lanczos attribute*), 53
kwargs (*lvsfunc.kernels.MinSide attribute*), 53
kwargs (*lvsfunc.kernels.Mitchell attribute*), 53
kwargs (*lvsfunc.kernels.NearestNeighbour attribute*), 53
kwargs (*lvsfunc.kernels.Parzen attribute*), 53
kwargs (*lvsfunc.kernels.Point attribute*), 53
kwargs (*lvsfunc.kernels.Quadratic attribute*), 54
kwargs (*lvsfunc.kernels.Robidoux attribute*), 54
kwargs (*lvsfunc.kernels.RobidouxSharp attribute*), 54
kwargs (*lvsfunc.kernels.RobidouxSoft attribute*), 54
kwargs (*lvsfunc.kernels.Sinc attribute*), 54
kwargs (*lvsfunc.kernels.Spline16 attribute*), 54
kwargs (*lvsfunc.kernels.Spline36 attribute*), 55
kwargs (*lvsfunc.kernels.Spline64 attribute*), 55
kwargs (*lvsfunc.kernels.Welch attribute*), 55
kwargs (*lvsfunc.kernels.Wiener attribute*), 56

L

Lanczos (*class in lvsfunc.kernels*), 52
limit_dark() (*in module lvsfunc.misc*), 60
linear2gamma() (*in module lvsfunc.scale*), 73
load_bookmarks() (*in module lvsfunc.misc*), 60
lvsfunc
    module, 1
lvsfunc.aa
    module, 13
lvsfunc.comparison
    module, 19
lvsfunc.deblock
    module, 27
lvsfunc.dehalo
    module, 29
lvsfunc.dehardsub
    module, 35
lvsfunc.deinterlace
    module, 37
lvsfunc.denoise
    module, 43
lvsfunc.kernels
    module, 49
lvsfunc.mask
    module, 46
lvsfunc.misc
    module, 57
lvsfunc.recon
    module, 65
lvsfunc.render
    module, 69
lvsfunc.scale
    module, 72
lvsfunc.types
    module, 77
lvsfunc.util
    module, 79

M

masked_dha() (*in module lvsfunc.dehalo*), 30
Matrix (*class in lvsfunc.types*), 77
maxm() (*in module lvsfunc.mask*), 47
minimum (*lvsfunc.dehardsub.HardsubSign attribute*), 34
minm() (*in module lvsfunc.mask*), 47
MinSide (*class in lvsfunc.kernels*), 53
Mitchell (*class in lvsfunc.kernels*), 53
mixed_rescale() (*in module lvsfunc.scale*), 73
module
    lvsfunc, 1
    lvsfunc.aa, 13
    lvsfunc.comparison, 19
    lvsfunc.deblock, 27
    lvsfunc.dehalo, 29
    lvsfunc.dehardsub, 35
    lvsfunc.deinterlace, 37
    lvsfunc.denoise, 43
    lvsfunc.kernels, 49
    lvsfunc.mask, 46
    lvsfunc.misc, 57
    lvsfunc.recon, 65
    lvsfunc.render, 69
    lvsfunc.scale, 72
    lvsfunc.types, 77
    lvsfunc.util, 79
mt_xxpand_multi() (*in module lvsfunc.mask*), 48

N

NearestNeighbour (*class in lvsfunc.kernels*), 53
nnedi3() (*in module lvsfunc.aa*), 14
nneedi3_clamp() (*in module lvsfunc.aa*), 14
normalize_ranges() (*in module lvsfunc.util*), 80

O

overlay_sign() (*in module lvsfunc.misc*), 60

P

padder() (*in module lvsfunc.util*), 80
Parzen (*class in lvsfunc.kernels*), 53
phi (*lvsfunc.types.Coefs attribute*), 77
pick_removegrain() (*in module lvsfunc.util*), 80
pick_repair() (*in module lvsfunc.util*), 81
Point (*class in lvsfunc.kernels*), 53
pos (*lvsfunc.mask.BoundingBox attribute*), 45
Position (*class in lvsfunc.types*), 78

Q

Quadratic (*class in lvsfunc.kernels*), 54
queued (*lvsfunc.render.RenderContext attribute*), 69
quick_resample() (*in module lvsfunc.util*), 81

R

range_mask() (*in module lvsfunc.mask*), 48
ranges (*lvsfunc.dehardsub.HardsubMask attribute*), 34
ranges (*lvsfunc.mask.DeferredMask attribute*), 46
reconstruct_multi() (*in module lvsfunc.recon*), 67
ref_float (*lvsfunc.dehardsub.HardsubLineFade attribute*), 35
ref_float (*lvsfunc.dehardsub.HardsubSignFade attribute*), 35
refframes (*lvsfunc.dehardsub.HardsubMask attribute*), 34
refframes (*lvsfunc.mask.DeferredMask attribute*), 46
regress() (*in module lvsfunc.recon*), 67
RegressClips (*class in lvsfunc.recon*), 65
RenderContext (*class in lvsfunc.render*), 69
replace_ranges() (*in module lvsfunc.util*), 81
resample() (*lvsfunc.kernels.Bicubic method*), 49
resample() (*lvsfunc.kernels.Bilinear method*), 50
resample() (*lvsfunc.kernels.FmtConv method*), 51
resample() (*lvsfunc.kernels.Lanczos method*), 53
resample() (*lvsfunc.kernels.Point method*), 53
resample() (*lvsfunc.kernels.Spline16 method*), 54
resample() (*lvsfunc.kernels.Spline36 method*), 55
resample() (*lvsfunc.kernels.Spline64 method*), 55
rescaled (*lvsfunc.scale.ScaleAttempt attribute*), 71
RESERVED() (*lvsfunc.types.Matrix property*), 77
Resolution (*class in lvsfunc.scale*), 71
resolution (*lvsfunc.scale.ScaleAttempt attribute*), 71
reupscale (*in module lvsfunc.scale*), 74
rfs() (*in module lvsfunc.util*), 82
RGB (*lvsfunc.types.Matrix attribute*), 77
Robidoux (*class in lvsfunc.kernels*), 54
RobidouxSharp (*class in lvsfunc.kernels*), 54
RobidouxSoft (*class in lvsfunc.kernels*), 54

S

scale() (*lvsfunc.kernels.Bicubic method*), 49
scale() (*lvsfunc.kernels.Bilinear method*), 50
scale() (*lvsfunc.kernels.FmtConv method*), 51
scale() (*lvsfunc.kernels.Impulse method*), 52
scale() (*lvsfunc.kernels.Lanczos method*), 53
scale() (*lvsfunc.kernels.Point method*), 53
scale() (*lvsfunc.kernels.Spline16 method*), 54
scale() (*lvsfunc.kernels.Spline36 method*), 55
scale() (*lvsfunc.kernels.Spline64 method*), 55
scale_peak() (*in module lvsfunc.util*), 82
scale_thresh() (*in module lvsfunc.util*), 82
ScaleAttempt (*class in lvsfunc.scale*), 71
SceneChangeMode (*class in lvsfunc.render*), 69
scomp() (*in module lvsfunc.comparison*), 24
SCXVID (*lvsfunc.render.SceneChangeMode attribute*), 69
seek_cycle() (*in module lvsfunc.deinterlace*), 41
shift (*lvsfunc.dehardsub.HardsubASS attribute*), 35
shift() (*lvsfunc.kernels.Bicubic method*), 49
shift() (*lvsfunc.kernels.Bilinear method*), 50
shift() (*lvsfunc.kernels.FmtConv method*), 51
shift() (*lvsfunc.kernels.Lanczos method*), 53
shift() (*lvsfunc.kernels.Point method*), 54
shift() (*lvsfunc.kernels.Spline16 method*), 55
shift() (*lvsfunc.kernels.Spline36 method*), 55
shift() (*lvsfunc.kernels.Spline64 method*), 55
shift_tint() (*in module lvsfunc.misc*), 61
Sinc (*class in lvsfunc.kernels*), 54
SIVTC() (*in module lvsfunc.deinterlace*), 37
sivtc() (*in module lvsfunc.deinterlace*), 41
Size (*class in lvsfunc.types*), 78
size (*lvsfunc.mask.BoundingBox attribute*), 45
slope (*lvsfunc.recon.RegressClips attribute*), 65
SMPTE170M (*lvsfunc.types.Matrix attribute*), 78
SMPTE2085 (*lvsfunc.types.Matrix attribute*), 78
SMPTE240M (*lvsfunc.types.Matrix attribute*), 78
source() (*in module lvsfunc.misc*), 61
Spline16 (*class in lvsfunc.kernels*), 54
Spline36 (*class in lvsfunc.kernels*), 55
Spline64 (*class in lvsfunc.kernels*), 55
Split (*class in lvsfunc.comparison*), 20
split() (*in module lvsfunc.comparison*), 24
sraa() (*in module lvsfunc.aa*), 15
src() (*in module lvsfunc.misc*), 62
ssim_downsample() (*in module lvsfunc.scale*), 74
Stack (*class in lvsfunc.comparison*), 20
stack_compare() (*in module lvsfunc.comparison*), 24
stack_horizontal() (*in module lvsfunc.comparison*), 25
stack_planes() (*in module lvsfunc.comparison*), 25
stack_vertical() (*in module lvsfunc.comparison*), 25

T

taa() (*in module lvsfunc.aa*), 15
thresh (*lvsfunc.dehardsub.HardsubSign attribute*), 34
Tile (*class in lvsfunc.comparison*), 21
tile() (*in module lvsfunc.comparison*), 25
timecodes (*lvsfunc.render.RenderContext attribute*), 69
TIVTC_VFR() (*in module lvsfunc.deinterlace*), 37
tivtc_vfr() (*in module lvsfunc.deinterlace*), 41
transpose_aa() (*in module lvsfunc.aa*), 15

U

UNKNOWN (*lvsfunc.types.Matrix attribute*), 78
unsharpen() (*in module lvsfunc.misc*), 62
upscaled_sraa() (*in module lvsfunc.aa*), 16

V

VERTICAL (*lvsfunc.comparison.Direction attribute*), 20
vinverse() (*in module lvsfunc.deinterlace*), 42
vsdpir() (*in module lvsfunc.deblock*), 28
VSFunction (*class in lvsfunc.types*), 78

W

Welch (*class in lvsfunc.kernels*), 55
width (*lvsfunc.scale.Resolution attribute*), 71
Wiener (*class in lvsfunc.kernels*), 55
wipe_row() (*in module lvsfunc.misc*), 63
WWXD (*lvsfunc.render.SceneChangeMode attribute*), 69
WWXD_SCXVID_INTERSECTION (*lvsfunc.render.SceneChangeMode attribute*), 69
WWXD_SCXVID_UNION (*lvsfunc.render.SceneChangeMode attribute*), 69

X

x (*lvsfunc.types.Coordinate attribute*), 77
x (*lvsfunc.types.Position attribute*), 78
x (*lvsfunc.types.Size attribute*), 78

Y

y (*lvsfunc.types.Coordinate attribute*), 77
y (*lvsfunc.types.Position attribute*), 78
y (*lvsfunc.types.Size attribute*), 78
YCGCO (*lvsfunc.types.Matrix attribute*), 78