lvsfunc

Release 0.1.0

LightArrowsEXE

Jun 24, 2022

CONTENTS:

CHAPTER

ONE

ABOUT

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

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

CHAPTER

TWO

DEPENDENCIES

lvsfunc depends on the following third-party scripts:

- debandshit
- edi_rpow2
- havsfunc
- kagefunc
- mvsfunc
- vsTAAmbk
- vsutil

The following Vapoursynth libraries are also required for full functionality:

- adaptivegrain
- combmask
- d2vsource
- dgdecnv
- fmtconv
- knlmeanscl
- rgsf
- vapoursynth-descale
- vapoursynth-nnedi3
- vapoursynth-eedi3
- vapoursynth-readmpls
- vapoursynth-retinex
- vapoursynth-tcanny
- vs-continuityfixer
- zimg
- znedi3

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

CHAPTER

THREE

MODULES

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

CHAPTER

FOUR

FUNCTIONS

lvsfunc.aa.nneedi3_clamp(clip[, strength, ...])	A function that clamps eedi3 to nnedi3 for the purpose of reducing eedi3 artifacts.
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 an upscaled 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.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.comparison.tvbd_diff(tv, bd[, thr, ...])	Creates a standard lvsfunc.comparison.Stack between frames from two clips that have differences.
lvsfunc.deinterlace.deblend(clip[, rep])	A simple function to fix deblending for interlaced video with an AABBA blending pattern, where A is a normal frame and B is a blended frame.
lvsfunc.deinterlace.decomb(clip, TFF[, ...])	Does some aggressive filtering to get rid of the combing on a interlaced/telecined source.
lvsfunc.deinterlace.dir_deshimmer(clip[, ...])	Directional deshimmering function.
lvsfunc.deinterlace.dir_unsharp(clip[, ...])	Diff'd directional unsharpening function.
lvsfunc.denoise.adaptive_mask	A wrapper to create a luma mask for denoising and/or debanding.
lvsfunc.denoise.detail_mask	A wrapper for creating a detail mask to be used during denoising and/or debanding.
lvsfunc.denoise.quick_denoise(clip[, ref, ...])	This wrapper is used to denoise both the luma and chroma using various denoisers of your choosing.

continues on next page

Table 1 – continued from previous page

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.fix_cr_tint(clip[, value])	Tries to forcibly fix Crunchyroll's green tint by adding pixel values.
lvsfunc.misc.frames_since_bookmark(clip, ...)	Displays frames since last bookmark to create easily reusable scenefiltering.
lvsfunc.misc.limit_dark(clip, filtered[, ...])	Replaces frames in a clip with a filtered clip when the frame's darkness exceeds the threshold.
lvsfunc.misc.load_bookmarks(bookmark_path)	VSEdit bookmark loader.
lvsfunc.misc.replace_ranges(clip_a, clip_b, ...)	A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string.
lvsfunc.misc.source(file[, ref, ...])	Generic clip import function.
lvsfunc.misc.wipe_row(clip[, secondary, ...])	Simple function to wipe a row with a blank clip.
lvsfunc.scale.descale(clip[, upscaler, ...])	A unified descaling function.
lvsfunc.scale.descale_detail_mask	Generate a detail mask given a clip and a clip rescaled with the same kernel.
lvsfunc.scale.reupscale	A quick 'n easy wrapper used to re-upscale a clip descaled with descale using znedi3.
lvsfunc.scale.test_descale(clip[, width, ...])	Generic function to test descales with; descales and re-upscales a given clip, allowing you to compare the two easily.
lvsfunc.util.pick_removegrain(clip)	Returns rgvs.RemoveGrain if the clip is 16 bit or lower, else rgsf.RemoveGrain.
lvsfunc.util.pick_repair(clip)	Returns rgvs.Repair if the clip is 16 bit or lower, else rgsf.Repair.
lvsfunc.util.quick_resample(clip, function)	A function to quickly resample to 16/8 bit and back to the original depth.

CHAPTER

FIVE

LVSFUNC.AA

lvsfunc.aa.nneedi3_clamp(clip[, strength, ...])	A function that clamps eedi3 to nnedi3 for the purpose of reducing eedi3 artifacts.
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 an upscaled single-rate AA to deal with heavy aliasing and broken-up lineart.

Functions for various anti-aliasing functions and wrappers.

lvsfunc.aa.**nneedi3_clamp**(*clip*, *strength=1*, *mask=None*, *ret_mask=False*, *show_mask=False*, *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.

Dependencies:

- kagefunc (optional: retinex edgemask)
- vapoursynth-retinex (optional: retinex edgemask)
- vapoursynth-tcanny (optional: retinex edgemask)
- vapoursynth-eedi3
- vapoursynth-nnedi3 or znedi3
- vapoursynth-nnedi3cl (optional: opencl)
- vsTAAmbk

Parameters

- **clip** (VideoNode) – Input clip
- **strength** (int) – Set threshold strength (Default: 1)
- **mask** (Optional[VideoNode]) – Clip to use for custom mask (Default: None)
- **ret_mask** (bool) – Replace default mask with a retinex edgemask (Default: False)
- **show_mask** (bool) – Return mask instead of clip (Default: False)
- **opencl** (bool) – OpenCL acceleration (Default: False)

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*, *rep=None*, *width=None*, *height=None*, *downscaler=vapoursynth.core.resize.Spline36*, ***eedi3_args*)

A function that performs an upscaled single-rate AA to deal with heavy aliasing and broken-up lineart. Useful for Web rips, where the source quality is not good enough to descale, but you still want to deal with some bad aliasing and lineart.

Original function written by Zastin, heavily modified by LightArrowsEXE.

Alias for this function is *lvsfunc.sraa*.

Dependencies: fmtconv, rgsf (optional: 32 bit clip), vapoursynth-eedi3, vapoursynth-nnedi3

Parameters

- **clip** (VideoNode) – Input clip
- **rfactor** (float) – Image enlargement factor. 1.3..2 makes it comparable in strength to vsTAAmbk It is not recommended to go below 1.3 (Default: 1.5)
- **rep** (Optional[int]) – Repair mode (Default: None)
- **width** (Optional[int]) – Target resolution width. If None, determine from *height*
- **height** (Optional[int]) – Target resolution height (Default: clip.height)
- **downscaler** (Callable[[VideoNode, int, int], VideoNode]) – Resizer used to downscale the AA'd clip
- **kwargs** – Arguments passed to znedi3 (Default: alpha=0.2, beta=0.6, gamma=40, nrad=2, mdis=20)

Return type VideoNode

Returns Antialiased and optionally rescaled clip

CHAPTER

SIX

LVSFUNC.COMPARISON

lvsfunc.comparison.compare(clip_a, clip_b[, ...])	Allows for the same frames from two different clips to be compared by interleaving them into a single clip.
lvsfunc.comparison.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.comparison.tvbd_diff(tv, bd[, thr, ...])	Creates a standard *lvsfunc.comparison.Stack* between frames from two clips that have differences.

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

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

Bases: abc.ABC

Base class for comparison functions.

Parameters

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

property clip

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

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

Return type VideoNode

class lvsfunc.comparison.**Direction**
Bases: enum.IntEnum

Enum to simplify direction argument.

HORIZONTAL = 0

VERTICAL = 1

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

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

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

Parameters

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

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

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

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

Parameters

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

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

Stacks clips horizontally or vertically.

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

Parameters

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

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

Bases: *lvsfunc.comparison.Comparer*

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

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

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

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

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

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

Parameters

- **clips** (Union[Dict[str, VideoNode], Sequence[VideoNode]]) – A dict mapping names to clips or simply a sequence of clips in a tuple or a list. If given a dict, the names will be overlayed on the clips using `VideoNode.text.Text`. If given a simple sequence of clips, the *label_alignment* parameter will have no effect and the clips will not be labeled. The order of the clips in either a dict or a sequence will be kept in the comparison.
- **arrangement** (Optional[List[List[int]]]) – 2-dimension array (list of lists) of 0s and 1s representing a list of rows of clips(*1*) or blank spaces(*0*) (Default: None)

- **label_alignment** (int) – An integer from 1-9, corresponding to the positions of the keys on a numpad. Only used if *clips* is a dict. Determines where to place clip name using VideoNode.text.Text (Default: 7)

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

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

Alias for this function is *lvsfunc.comp*.

Dependencies: mvsfunc

Parameters

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

Return type VideoNode

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

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

Small convenience function for interleaving clips.

Parameters

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

Return type VideoNode

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

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

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

Parameters

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

Return type VideoNode

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

lvsfunc.comparison.**stack_compare**(**clips*, *make_diff=True*, *height=288*, *warn=None*)
A simple wrapper that allows you to compare two clips by stacking them.

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

Parameters

- **clips** (VideoNode) – Clips to compare
- **make_diff** (bool) – Create and stack a diff (only works if two clips are given) (Default: True)
- **height** (int) – Height in px to rescale clips to if *make_diff* is True (MakeDiff clip will be twice this resolution) (Default: 288)
- **warn** (Optional[Any]) – Unused parameter kept for backward compatibility

Return type VideoNode

Returns Clip with *clips* stacked

lvsfunc.comparison.**stack_horizontal**(**clips*, ***namedclips*)
Small convenience function for stacking clips horizontally.

Parameters

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

Return type VideoNode

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

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

Parameters

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

Return type VideoNode

Returns Clip with stacked planes

lvsfunc.comparison.**stack_vertical**(**clips*, ***namedclips*)
Small convenience function for stacking clips vertically.

Parameters

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

Return type VideoNode

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

lvsfunc.comparison.**tile**(**clips*, ***namedclips*)
Small convenience function for tiling clips in a rectangular pattern. All clips must have the same dimensions

(width and height). If 3 clips are given, a 2x2 square with one blank slot will be returned. If 6 clips are given, a 3x2 rectangle will be returned.

Parameters

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

Return type VideoNode

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

lvsfunc.comparison.**tvbd_diff**(*tv*, *bd*, *thr=72*, *height=288*, *return_array=False*)

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

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

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

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

Alias for this function is *lvsfunc.diff*.

Parameters

- **tv** (VideoNode) – TV clip
- **bd** (VideoNode) – BD clip
- **thr** (float) – Threshold, <= 1 uses PlaneStatsDiff, >1 uses Max/Min. Value must be below 128 (Default: 72)
- **height** (int) – Height in px to downscale clips to if *return_array* is False (MakeDiff clip will be twice this resolution) (Default: 288)
- **return_array** (bool) – Return frames as an interleaved comparison (using *lvsfunc.comparison.Interleave*) (Default: False)

Return type VideoNode

Returns Either an interleaved clip of the differences between the two clips or a stack of both input clips on top of MakeDiff clip

CHAPTER

SEVEN

LVSFUNC.DEINTERLACE

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

Functions to help with deinterlacing or deinterlaced content.

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

A simple function to fix deblending for interlaced video with an AABBA blending pattern, where A is a normal 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 C frame by getting halves of CD and DA, and using that to fix up CD. DA can then be dropped due to it being an interlaced frame.

However, doing this will result in some of the artifacting being added to the deblended frame. We can mitigate this 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: 32bit clip)

Parameters

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

Return type VideoNode

Returns Deblended clip

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

Does some aggressive filtering to get rid of the combing on a interlaced/telecined source. You can also allow it to decimate the clip, or keep it disabled if you wish to handle the decimating yourself. Enabling vinverse will result in more aggressive decombing at the cost of potential detail loss.

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

Dependencies: combmask, havsfunc (QTGMC), rgsf (optional: 32bit clip)

Deciphering havsfunc's dependencies is left as an exercise for the user.

Parameters

- **clip** (VideoNode) – Input clip
- **TFF** (bool) – Top-Field-First
- **decimate** (bool) – Decimate the video after deinterlacing (Default: True)
- **vinv** (bool) – Use vinverse to get rid of additional combing (Default: False)
- **sharpen** (bool) – Unsharpen after deinterlacing (Default: False)
- **dir** (str) – Directional vector. 'v' = Vertical, 'h' = Horizontal (Default: v)
- **rep** (Optional[int]) – Repair mode for repairing the decombed clip using the original clip (Default: None)

Return type VideoNode

Returns Decombed clip

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

Directional deshimmering function.

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

Dependencies: vapoursynth-nnedi3

Parameters

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

Return type VideoNode

Returns Deshimmered clip

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

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

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

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

Dependencies: knlmeanscl

Parameters

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

Return type `VideoNode`

Returns Unsharpened clip

CHAPTER

EIGHT

LVSFUNC.DENOISE

lvsfunc.denoise.adaptive_mask	A wrapper to create a luma mask for denoising and/or debanding.
lvsfunc.denoise.detail_mask	A wrapper for creating a detail mask to be used during denoising and/or debanding.
lvsfunc.denoise.quick_denoise(clip[, ref, ...])	This wrapper is used to denoise both the luma and chroma using various denoisers of your choosing.

Wrappers and masks for denoising.

lvsfunc.denoise.**adaptive_mask**

A wrapper to create a luma mask for denoising and/or debanding.

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

Dependencies: adaptivegrain

Parameters

- **clip** – Input clip
- **luma_scaling** – Luma scaling factor (Default: 8.0)

Returns Luma mask

lvsfunc.denoise.**detail_mask**

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

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

Dependencies: knlmeans (optional: pre_denoise), debandshit

Parameters

- **clip** – Input clip
- **pre_denoise** – Denoise the clip before creating the mask (Default: False)
- **rad** – The luma equivalent of gradfun3's "mask" parameter
- **radc** – The chroma equivalent of gradfun3's "mask" parameter

Brz_a Binarizing for the detail mask (Default: 0.05)

Brz_b Binarizing for the edge mask (Default: 0.05)

Returns Detail mask

lvsfunc.denoise.**quick_denoise**(*clip*, *ref=None*, *cmode='knlm'*, *sigma=2*, ***kwargs*)

This wrapper is used to denoise both the luma and chroma using various denoisers of your choosing. If you wish to use just one denoiser, you're probably better off using that specific filter rather than this wrapper.

A rewrite of my old 'quick_denoise'. I still hate it, but whatever. This will probably be removed in a future commit.

BM3D is used for denoising the luma.

Special thanks to kageru for helping me out with some ideas and pointers.

Alias for this function is *lvsfunc.qden*.

Dependencies: havsfunc (optional: SMDegrain mode), mvsfunc

Deciphering havsfunc's dependencies is left as an excercise for the user.

Parameters

- **clip** (VideoNode) – Input clip
- **cmode** (str) – Chroma denoising modes: 'knlm' - Use knlmeans for denoising the chroma (Default), 'tnlm' - Use tnlmeans for denoising the chroma, 'dft' - Use dfttest for denoising the chroma (requires setting 'sbsize' in kwargs), 'smd' - Use SMDegrain for denoising the chroma,
- **sigma** (float) – Denoising strength for BM3D (Default: 2)
- **ref** (Optional[VideoNode]) – Optional reference clip to replace BM3D's basic estimate
- **kwargs** (Any) – Parameters passed to chroma denoiser

Return type VideoNode

Returns Denoised clip

CHAPTER

NINE

LVSFUNC.KERNELS

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

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

Built-in bicubic resizer.

Parameters

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

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

Return type VideoNode

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

Return type VideoNode

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

Built-in bilinear resizer.

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

Return type VideoNode

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

Return type VideoNode

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

Abstract scaling kernel interface.

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

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

Return type VideoNode

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

Return type VideoNode

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

Built-in lanczos resizer.

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

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

Return type VideoNode

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

Return type VideoNode

class lvsfunc.kernels.**Spline16**(***kwargs*)

Bases: *lvsfunc.kernels.Kernel*

Built-in spline16 resizer.

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

Return type VideoNode

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

Return type VideoNode

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

Bases: *lvsfunc.kernels.Kernel*

Built-in spline36 resizer.

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

Return type VideoNode

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

Return type VideoNode

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

Bases: *lvsfunc.kernels.Kernel*

Built-in spline64 resizer.

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

Return type VideoNode

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

Return type VideoNode

CHAPTER

TEN

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.fix_cr_tint(clip[, value])	Tries to forcibly fix Crunchyroll's green tint by adding pixel values.
lvsfunc.misc.frames_since_bookmark(clip, ...)	Displays frames since last bookmark to create easily reusable scenefiltering.
lvsfunc.misc.limit_dark(clip, filtered[, ...])	Replaces frames in a clip with a filtered clip when the frame's darkness exceeds the threshold.
lvsfunc.misc.load_bookmarks(bookmark_path)	VSEdit bookmark loader.
lvsfunc.misc.replace_ranges(clip_a, clip_b, ...)	A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string.
lvsfunc.misc.source(file[, ref, ...])	Generic clip import function.
lvsfunc.misc.wipe_row(clip[, secondary, ...])	Simple function to wipe a row with a blank clip.

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

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

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

Parameters

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

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

Returns Function decorator for the given output format.

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

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

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

Return type ~F

Returns Decorated function

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

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

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

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

Written by Dave <orangechannel@pm.me>.

Parameters

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

Return type List[VideoNode]

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

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

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

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

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

Alias for this function is *lvsfunc.ef*.

Dependencies: vs-continuityfixer

Parameters

- **clip** (VideoNode) – Input clip

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

Return type VideoNode

Returns Clip with fixed edges

lvsfunc.misc.**fix_cr_tint**(*clip*, *value=128*)

Tries to forcibly fix Crunchyroll's green tint by adding pixel values.

Parameters

- **clip** (VideoNode) – Input clip
- **value** (int) – Value added to every pixel (Default: 128)

Return type VideoNode

Returns Clip with CR tint fixed

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

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

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

Parameters

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

Return type VideoNode

Returns Conditionally filtered clip

lvsfunc.misc.**load_bookmarks**(*bookmark_path*)

VSEdit bookmark loader.

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

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

Return type List[int]

Returns A list of bookmarked frames

lvsfunc.misc.**replace_ranges**(*clip_a*, *clip_b*, *ranges*)

A replacement for ReplaceFramesSimple that uses ints and tuples rather than a string. Frame ranges are inclusive.

Written by louis.

Alias for this function is *lvsfunc.rfs*.

Parameters

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

Return type VideoNode

Returns Clip with ranges from clip_a replaced with clip_b

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

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

Alias for this function is *lvsfunc.src*.

Dependencies:

- d2vsource (optional: d2v sources)
- dgdecodenv (optional: dgi sources)
- mvsfunc (optional: reference clip mode)
- vapoursynth-readmpls (optional: mpls sources)

Parameters

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

- **mpls_angle** (int) – Angle number to select in the mpls playlist (Default: 0)

Return type VideoNode

Returns Vapoursynth clip representing input file

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

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

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

Dependencies: kagefunc

Parameters

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

Return type VideoNode

Returns Clip with rows wiped

CHAPTER

ELEVEN

LVSFUNC.SCALE

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

class lvsfunc.scale.**Resolution**

Bases: tuple

Tuple representing a resolution.

width
Width.

height
Height.

class lvsfunc.scale.**ScaleAttempt**

Bases: tuple

Tuple representing a descale attempt.

descaled
Descaled frame in native resolution.

rescaled
Descaled frame reupscaled with the same kernel.

resolution
The native resolution.

diff
The subtractive difference between the original and descaled frame.

Functions for (de)scaling.

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

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

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

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

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

Dependencies: vapoursynth-descale, znedi3

Parameters

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

Return type `VideoNode`

Returns Descaled and re-upscaled clip

lvsfunc.scale.**descale_detail_mask**

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

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

Parameters

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

Returns Mask of lost detail

lvsfunc.scale.**reupscale**

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

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

Stolen from Varde with some adjustments made.

Dependencies: znedi3

Parameters

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

Returns Reupscaled clip

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

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

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

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

Dependencies: vapoursynth-descale

Parameters

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

Return type Tuple[VideoNode, *ScaleAttempt*]

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

CHAPTER

TWELVE

LVSFUNC.UTIL

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

Helper functions for the main functions in the script.

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

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

Parameters

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

Return type ~T

Returns frame.prop[key]

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

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

Dependencies: rgsf

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

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

Returns Appropriate RemoveGrain function for input clip's depth

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

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

Dependencies: rgsf

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

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

Returns Appropriate repair function for input clip's depth

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

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

Parameters

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

Return type VideoNode

Returns Filtered clip in original depth

CHAPTER

THIRTEEN

SPECIAL CREDITS

A special thanks to every contributor that contributed to lvsfunc.

The list of contributors can be found here.

CHAPTER

FOURTEEN

FOOTER

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

INDEX

N

P

Q

R

S

T

U

V

W