MaskTools2

see also MaskTools

Contents [hide]

Abstract	
Author	pinterf, tp7, Manao, mg262, Kurosu
Version	2.2.13
Download	▪ AviSynth 2.6.0 x86 or AviSynth+ x86/x64 Masktools2 2.2.13
Category	Support filters
License	MIT but binaries are GPLv2
Discussion	Doom9 Thread /// Updated thread

MaskTools2 v2.2.x

This is a fork of tp7's MaskTools2 plugin. It works for high bit depth under AviSynth+ (10-16 bits, float), extends existing and add new features, contains a bugfix, and AVX and AVX2 optimizations for some filters. This branch will be called 2.2.x opposed to 2.0.* like the previous MaskTools2. At the moment "mt_polish" functionality is not available on XP builds.

Difference to Masktools2 b1

- project moved to Visual Studio 2017
  Requires Visual Studio redistributables (14.xx family)
- add back "none" and "ignore" for values to "chroma" parameter (2.2.9-)
- Fix: mt_merge at 8 bit clips: keep exact pixel values when mask is 0 or 255 (v2.2.7-)
- Fix: mt_merge (and probably other multi-clip filters) may result in corrupted results under specific circumstances, due to using video frame pointers which were already released from memory
- no special function names for high bit depth filters
- filters are auto registering their mt mode as MT_NICE_FILTER for Avisynth+
- Avisynth+ high bit depth support (incl. planar RGB, color spaces with alpha plane are supported from v2.2.7)

All filters are now supporting 10, 12, 14, 16 bits and float Threshold and sc_value parameters are scaled automatically to the current bit depth (v2.2.5-) from a default 8-bit value. Y,U,V,A (and parameters chroma/alpha) negative (memset) values are scaled automatically to the current bit depth (v2.2.7-, chroma/alpha v.2.2.8) from a default 8-bit value. Default range of such parameters can be overridden to 8-16 bits or float. Disable parameter scaling with paramscale="none"

- Supporting clips with alpha plane
  - new parameter "A" (default 1: no process) similar to "Y", "U", and "V"
  - new parameter aExpr for filters that can take yExpr, vExpr and uExpr
  - new parameter "alpha" to override alpha plane mode, similar to "chroma"
- New plane mode: 6 (copy from fourth clip) for "Y", "U", "V" and "A" (2.2.7-)
- New "chroma" and "alpha" plane mode override: "copy fourth" (2.2.7-)

Use for mt_lutxyza which has four clips

- YV411 (8 bit 4:1:1) support
- mt_merge accepts 4:2:2 clips when luma=true (8-16 bit)
- mt_merge accepts 4:1:1 clips when luma=true
- mt_merge to discard U and V automatically when input is greyscale
- some filters got AVX (float) and AVX2 (integer) support:
  - mt_merge: 8-16 bit: AVX2, float:AVX
  - mt_logic: 8-16 bit: AVX2, float:AVX
  - mt_edge: 8-16 bit: AVX2, float: AVX
- mt_polish to recognize new constants and scaling operator, and some other operators introduced in earlier versions.

Due to the stopped XP support of boost library mt_polish and mt_infix are not available on XP version.

- new: mt_lutxyza. Accepts four clips. 4th variable name is 'a' (besides x, y and z)
- new: mt_luts: weight expressions as an addon for then main expression(s) (martin53's idea)
  - wexpr
  - ywExpr, uwExpr, vwExpr, awExpr

If the relevant parameter strings exist, the weighting expression is evaluated for each source/neighborhood pixel values (lut or realtime, depending on the bit depth and the "realtime" parameter). Then the usual lut result is premultiplied by this weight factor before it gets accumulated. Weights are float values. Weight luts are x,y (2D) luts, similarly to the base working mode, where x is the base pixel, y is the current pixel from the neighbourhood, defined in "pixels".
When the weighting expression is "1", the result is the same as the basic weightless mode. For modes "average" and "std" the weights are summed up. Result is: sum(value_i*weight_i)/sum(weight_i). When all weights are equal to 1.0 then the expression will result in the average: sum(value_i)/n. Same logic works for min/max/median/etc., the "old" lut values are pre-multiplied with the weights before accumulation.

- expression syntax supporting bit depth independent expressions
  - bit-depth aware scale operators
    Note: operator @B and @F was #B and #F until v.2.2.4, but in the future use scaleb and scalef instead of them.
    - operator @B or scaleb scales from 8 bit to current bit depth using bit-shifts
      Use this for YUV. "235 scaleb" always results in max luma
    - operator @F or scalef scales from 8 bit to current bit depth using full range stretch
      "255 scalef" results in maximum pixel value of current bit depth.
      Calculation: x/255*65535 for a 8->16 bit sample (rgb)
  - hints for non-8 bit based constants:
    Added configuration keywords i8, i10, i12, i14, i16 and f32 in order to tell the expression evaluator the bit depth of the values that are to scale by scaleb and scalef operators (see "sbitdepth"). By default scaleb and scalef scales from 8 bit to the bit depth of the clip.

i8 .. i16 and f32 sets the default conversion base to 8..16 bits or float, respectively.
These keywords can appear anywhere in the expression, but only the last occurence will be effective for the whole expression.
Examples

```
8 bit video, no modifier: "x y - 256 scaleb *" evaluates as "x y - 256 *"
10 bit video, no modifier: "x y - 256 scaleb *" evaluates as "x y - 1024 *"
10 bit video: "i16 x y - 65536 scaleb *" evaluates as "x y - 1024 *"
8 bit video: "i10 x y - 512 scaleb *" evaluates as "x y - 128 *"
```

- - new pre-defined, bit depth aware constants
    - bitdepth: automatic silent parameter of the lut expression
    - range_half --> autoscaled 128 or 0.5 for float
    - range_max --> 255/1023/4095/16383/65535 or 1.0 for float
    - range_size --> 256/1024...65536
    - ymin, ymax, cmin, cmax --> 16/235 and 16/240 autoscaled.

Example #1 (bit depth dependent, all constants are treated as-is):

```
expr8_luma = "x 16 - 219 / 255 *"
expr10_luma = "x 64 - 876 / 1023 *"
expr16_luma = "x 4096 - 56064 / 65535 *"
```

Example #2 (new, with auto-scale operators)

```
expr_luma =  "x 16 scaleb - 219 scaleb / 255 scalef *"
expr_chroma =  "x 16 scaleb - 224 scaleb / 255 scalef *"
```

Example #3 (new, with constants)

```
expr_luma = "x ymin - ymax ymin - / range_max *"
expr_chroma = "x cmin - cmax cmin - / range_max *"
```

- new expression syntax: auto scale modifiers for float clips (test, may change):

Keyword at the beginning of the expression:

- - clamp_f_i8, clamp_f_i10, clamp_f_i12, clamp_f_i14 or clamp_f_i16 for scaling and clamping
  - clamp_f_f32 or clamp_f: for clamping the result to 0..1

Input values 'x', 'y', 'z' and 'a' are autoscaled by 255.0, 1023.0, ... 65535.0 before the expression evaluation, so the working range is similar to native 8, 10, ... 16 bits. The predefined constants 'range_max', etc. will behave for 8, 10,..16 bits accordingly.
The result is automatically scaled back to 0..1 _and_ is clamped to that range. When using clamp_f_f32 (or clamp_f) the scale factor is 1.0 (so there is no scaling), but the final clamping will be done anyway. No integer rounding occurs.

```
expr = "x y - range_half +"  # good for 8..32 bits but float is not clamped
expr = "clamp_f y - range_half +"  # good for 8..32 bits and float clamped to 0..1
expr = "x y - 128 + "  # good for 8 bits
expr = "clamp_f_i8 x y - 128 +" # good for 8 bits and float, float will be clamped to 0..1
expr = "clamp_f_i8 x y - range_half +" # good for 8..32 bits, float will be clamped to 0..1
```

- parameter "stacked" (default false) for filters with stacked format support

Stacked support is not intentional, but since tp7 did it, I did not remove the feature. Filters currently without stacked support will never have it.

- parameter "realtime" for lut-type filters, slower but at least works on those bit depths where LUT tables would occupy too much memory.

For bit depth limits where realtime = true is set as the default working mode, see table below.
realtime=true can be overridden, one can experiment and force realtime=false even for a 16 bit lutxy (8GBytes lut table!, x64 only) or for 8 bit lutxzya (4 GBytes lut table)

- parameter "paramscale" for filters working with threshold-like parameters (v2.2.5-)

Filters: mt_binarize, mt_edge, mt_inpand, mt_expand, mt_inflate, mt_deflate, mt_motion, mt_logic, mt_clamp
paramscale can be "i8" (default), "i10", "i10", "i12", "i14", "i16", "f32" or "none" or ""
Using "paramscale" tells the filter that parameters are given at what bit depth range.
By default paramscale is "i8", so existing scripts with parameters in the 0..255 range are working at any bit depths

```
 mt_binarize(threshold=80*256, paramscale="i16") # threshold is assumed in 16 bit range
 mt_binarize(threshold=80) # no param: threshold is assumed in 8 bit range

 thY1 = 0.1
 thC1 = 0.1
 thY2 = 0.1
 thC2 = 0.1
 paramscale="f32"
 mt_edge(mode="sobel",u=3,v=3,thY1=thY1,thY2=thY2,thC1=thC1,thC2=thC2,paramscale=paramscale) #
f32: parameters assumed as float (0..1.0)
```

- new: "swap" keyword in expressions (v2.2.5-)

swaps the last two results during RPN evaluation. Not compatible with mt_infix()

```
expr="x 2 /"
expr="2 x swap /"
```

- new: "dup" keyword in expressions (v2.2.5-)

duplicates the last result and put on the top of RPN evaluation stack. Not compatible with mt_infix()

```
expr="x 3 / x 3 / +"
expr="x 3 / dup +"
```

- Feature matrix

```
                 8 bit | 10-16 bit | float | stacked | realtime
    mt_invert      X         X         X        -
    mt_binarize    X         X         X        X
    mt_inflate     X         X         X        X
    mt_deflate     X         X         X        X
    mt_inpand      X         X         X        X
    mt_expand      X         X         X        X
    mt_lut         X         X         X        X      when float
    mt_lutxy       X         X         X        -      when bits>=14
    mt_lutxyz      X         X         X        -      when bits>=10
    mt_lutxyza     X         X         X        -      always
    mt_luts        X         X         X        -      when bits>=14
    mt_lutf        X         X         X        -      when bits>=14
    mt_lutsx       X         X         X        -      when bits>=10
    mt_lutspa      X         X         X        -
    mt_merge       X         X         X        X
    mt_logic       X         X         X        X
    mt_convolution X         X         X        -
    mt_mappedblur  X         X         X        -
    mt_gradient    X         X         X        -
    mt_makediff    X         X         X        X
    mt_average     X         X         X        X
    mt_adddiff     X         X         X        X
    mt_clamp       X         X         X        X
    mt_motion      X         X         X        -
    mt_edge        X         X         X        -
    mt_hysteresis  X         X         X        -
    mt_infix/mt_polish: available only on non-XP builds
```

MaskTools2 b1

This is a fork of Manao's MaskTools2 plugin. It mostly contains performance improvements, bugfixes and some little things that make the plugin more "mature". This branch will be called b* as opposed to a* like the original MaskTools2.

Difference to MaskTools2 a48

- Works correctly with AviSynth 2.6 Alpha 4/5 and RC 1 (including MT). Doesn't work with previous alphas.
- Much cleaner and easy to understand codebase, also the source code is now licensed under MIT.
- No MMX-optimized versions. MMX is too old to support and is always slower than SSE2.
- **all luts**: faster LUT calculation, faster startup, reduced memory footprint if the same LUT is used for multiple planes or some planes aren't processed. For example mt_lutxyz(c1, c2, c3, "x y + z -") will use only 16MBs of memory instead of 48MBs.
- **mt_lutspa**: does not depend on source clip performance as it doesn't get requested at all (unless mode 2 is used). Always much faster (5-o9k times).
- **all filters**: faster modes 2, 4 and 5 (copy), negative (memset) modes.
- **mt_hysteresis**: 3-4 times better performance.
- **all luts**: performance in mode 3 with an empty LUT is now identical to mode 2. Thus mt_lut(chroma="128") is a bit faster than Grayscale() instead of being much slower.
- **mt_merge**: luma=true now supports YV24.
- **mt_luts**: correct value is used as x. More info here.
- **sobel/roberts/laplace modes of mt_edge**: better performance when SSSE3 is available.
- **mt_edge("cartoon")**: 10 times faster when SSE2 is available.
- **all asm-optimized filters**: same performance on any resolution up to mod-1. Original MaskTools2 used unoptimized version for any non-mod8 clips.

All filters were tested on a Core i7 860. Performance might be a bit different on other CPUs.

Download

Latest revision of MaskTools2 v2.2.x (x86/x64)

- MaskTools2 v2.2.x release page - compatible with AviSynth 2.6.0 (x86) and AviSynth+ (x86/x64)

Runtime dependencies:

- Microsoft Visual C++ 2015 Redistributable Update 3 (x86/x64)

Filters

MaskTools2 contain a set of filters designed to create, manipulate and use masks. Masks, in video processing, are a way to give a relative importance to each pixel. You can, for example, create a mask that selects only the green parts of the video, and then replace those parts with another video. To give the most control over the handling of masks, the filters will use the fact that each luma and chroma planes can be uncorrelated. That means that a single video will always be considered by the filters as 3 independent planes. That applies for masks as well, which means that a mask clip will in fact contain 3 masks, one for each plane.

The filters have a set of common parameters, that mainly concern what processing to do on each plane. All filters only work with planar colorspaces (Y8, YV12, YV16, and YV24 (AviSynth 2.5.8 only supports YV12!).

Beginning with v2.2.4 YUV and planar RGB 10-16 bit and 32 bit float color spaces are supported when using AviSynth+ (r2294-).

Here is an exhaustive list of the filters contained in MaskTools2 (see developer's page here for more information)

Filter	Description	Color format
	Masks creation	
Mt_edge	Creates edge masks.	Y8, YV12, YV16, YV24
Mt_motion	Creates motion masks.	Y8, YV12, YV16, YV24
	Masks operation	
Mt_invert	Inverses masks.	Y8, YV12, YV16, YV24
mt_binarize	Transforms soft masks into hard masks.	Y8, YV12, YV16, YV24
mt_logic	Combines masks using logic operators.	Y8, YV12, YV16, YV24
mt_hysteresis	Combines masks making the first one to grow into the second.	Y8, YV12, YV16, YV24
	Masks merging	
Mt_merge	Merges two clips according to a mask.	Y8, YV12, YV16, YV24
	Morphologic operators	
mt_expand	Expands the mask / the video.	Y8, YV12, YV16, YV24
mt_inpand	Inpands the mask / the video.	Y8, YV12, YV16, YV24
mt_inflate	Inflates the mask / the video.	Y8, YV12, YV16, YV24
mt_deflate	Deflates the mask / the video.	Y8, YV12, YV16, YV24
	LUT operators	
mt_lut	Applies an expression to all the pixels of a mask / video.	Y8, YV12, YV16, YV24
mt_lutxy	Applies an expression to all the pixels of two masks / videos.	Y8, YV12, YV16, YV24
mt_lutxyz	Applies an expression to all the pixels of three masks / videos.	Y8, YV12, YV16, YV24
mt_lutxyza	Applies an expression to all the pixels of four masks / videos. (v2.2.4-)	Y8, YV12, YV16, YV24 (+high bit depth)
	Creates a uniform picture from the collection of computation	Y8, YV12,

mt_lutf	on pixels of two clips.	YV16, YV24
mt_luts	Applies an expression taking neighbouring pixels into.	Y8, YV12, YV16, YV24
mt_lutsx	Applies an expression taking neighbouring pixels into, in a different way.	Y8, YV12, YV16, YV24
mt_lutspa	Computes the value of a pixel according to its spatial position.	Y8, YV12, YV16, YV24
Support operators		
mt_makediff	Substracts two clips.	Y8, YV12, YV16, YV24
mt_adddiff	Adds back a difference of two clips.	Y8, YV12, YV16, YV24
mt_clamp	Clamps a clip between two other clips.	Y8, YV12, YV16, YV24
mt_average	Averages two clips.	Y8, YV12, YV16, YV24
Convolutions		
mt_convolution	Applies a separable convolution on the picture.	Y8, YV12, YV16, YV24
mt_mappedblur	Applies a special 3x3 convolution on the picture.	Y8, YV12, YV16, YV24
Helpers		
mt_square	Creates a string describing a square.	Y8, YV12, YV16, YV24
mt_rectangle	Creates a string describing a rectangle.	Y8, YV12, YV16, YV24
mt_freerectangle	Creates a string describing a rectangle.	Y8, YV12, YV16, YV24
mt_diamond	Creates a string describing a diamond.	Y8, YV12, YV16, YV24
mt_losange	Creates a string describing a lozenge.	Y8, YV12, YV16, YV24
mt_freelosange	Creates a string describing a lozenge.	Y8, YV12, YV16, YV24
mt_circle	Creates a string describing a circle.	Y8, YV12, YV16, YV24
mt_ellipse	Creates a string describing an ellipse.	Y8, YV12, YV16, YV24
mt_freeellipse	Creates a string describing an ellipse.	Y8, YV12, YV16, YV24
mt_polish	Creates a reverse polish expression from an infix one.	-
mt_infix	Creates an infix expression from a reverse polish one.	-

Common parameters

As said previously, all the filters - except the helpers - share a common set of parameters. These parameters are used to tell what processing to do on each plane / channel, and what area of the video to process.

float **Y** = *3*
or *int* **Y** = *3*
float **U** = *1*
or *int* **U** = *1*
float **V** = *1*
or *int* **V** = *1*

Parameter types are float since v2.2.7 to allow setting 32 bit float type memset values, e.g. -0.5

These three values describe the actual processing mode that is to be used on each plane / channel. Here is how the modes are coded :

- `x` = -255..0, or -1.0..0.0 (any negative or zero value): all the pixels of the plane will be set to -x. From v.2.2.7- Default: 8 bit range autoscaled. Use "paramscale" to change the autoscale mode
- `x` = 1 : the plane will not be processed. That means the content of the plane after the filter is pure garbage.
- `x` = 2 : the plane of the first input clip will be copied.
- `x` = 3 : the plane will be processed with the processing the filter is designed to do.
- `x` = 4 (when applicable) : the plane of the second input clip will be copied.
- `x` = 5 (when applicable) : the plane of the third input clip will be copied.
- `x` = 6 (when applicable) : the plane of the fourth input clip will be copied. Since v2.2.7

As you can see, defaults parameters are chosen to only process the luma, and not to care about the chroma. It's because most video processing doesn't touch the chroma when handling 4:2:0.

string **chroma** = *""*

When defined, the value contained in this string will overwrite the `u` & `v` processing modes. This is a nice addition proposed by mg262 that makes the filter more user friendly. Allowed values for chroma are:

- `"none"` or `"ignore"` : set u = v = 1. Since 2.2.9-
- `"process"` : set u = v = 3.
- `"copy"` or `"copy first"` : set u = v = 2.
- `"copy second"` : set u = v = 4.
- `"copy third"` : set u = v = 5.
- `"copy fourth"` : set u = v = 6. Since v2.2.7
- `"xxx"`, where xxx is a number : set u = v = -xxx.

int **offX** = *0*
int **offY** = *0*

`offx` and `offy` are the top left coordinates of the box where the actual processing shall occur. Everything outside that box will be garbage.

int **w** = *-1*

int **h** = *-1*

`w` and `h` are the width and height of the processed box. -1 means that the box extends to the lower right corner of the video.

This also means that default settings are meant to process the whole picture.

float **A** = *1*
or *int* **A** = *1*

Since 2.2.7

Describe the actual processing mode that is to be used on "A" plane

see Y, U and V description

string **alpha** = *""*

Since v2.2.7

When defined, the value contained in this string will overwrite the `A` processing modes.

Allowed values for alpha are:

- `"none"` or `"ignore"` : set u = v = 1. Since 2.2.9-
- `"process"` : set a = 3.
- `"copy"` or `"copy first"` : set a = 2.
- `"copy second"` : set a = 4.
- `"copy third"` : set a = 5.
- `"copy fourth"` : set a = 6.
- `"xxx"`, where xxx is a number : set a = -xxx.

string **paramscale** = *""*

Since v2.2.7

When not defined, all threshold-like and Y, U, V, A parameter values are treated as 8 bit values and will be autoscaled accordingly to match the clip bit depths.

Allowed values for paramscale are:

- `"i8"` : Default, parameters are recognized as 8 bit integer constants. (existing 8 bit scripts will work in any bit depths)
- `"i10"` : Parameters are treated as in 10 bit range (0..1023). Autoscale base will be this bit depth.
- `"i12"` : Parameters are treated as in 12 bit range (0..4095).
- `"i14"` : Parameters are treated as in 14 bit range (0..16383).
- `"i16"` : Parameters are treated as in 16 bit range (0..65535).
- `"f32"` : Parameters are treated as in 32 bit float (0.0..1.0).
- `"none"` : Parameters are treated as is. No scaling happens at all.

Scaling method is "stretch" for RGB, and bit shifts for YUV colorspaces. Note that the maximum pixel value is always mapped to the max mixel value of the target bit depth. E.g. 255 will be 65535 and not 255 lshift 8

Reverse polish notation

A lot of filters accept custom functions defined by an expression written in reverse polish notation. You may not be accustomed to this notation, so here are a few pointers :

- The basic concept behind the notation is to write the operator / function after the arguments. Hence, "x + y" in infix notation becomes in reverse polish "x y +". "(3 + 5) * x" would become "3 5 + x *".
- As you noticed in the last example, the great asset of the notation is that it doesn't need parenthesis. The expression that would have been enclosed in parenthesis ("3 + 5") is correctly computed, because we read the expression from left to right, and because when the "+" is

encountered, its two operands are unmistakeably known.

- The supported operators are : "+", "-", "*", "/", "%" (modulo) and "^" (power)
- The supported functions are : "sin", "cos", "tan", "asin", "acos", "atan", "exp", "log", "abs", "round", "clip", "min", "max", "ceil", "floor", "trunc".
- Making the assumption that a positive float is "true", and a negative one is "false", we can also define boolean operators : "&", "|", "&!" (and not), "°" (xor), "@" (xor).
- We can create boolean values with the following comparison operators : "<", ">", "<=", ">=", "!=", "==", "=" (same as "==").
- Binary operators. Since internally all intermediate values are double, the parameters are first converted to 64 bit integer (unsigned or signed), and after the bit operation is done, the result will be converted back to double. So working with binary data is not fast, nor has real 64 bit integer precision.
  - Unsigned: "&u" (and), "|u" (or), "°u" (xor), "@u" (xor), "~u" (negate), "<<" or "<<s" (shift left), ">>" or ">>u" (shift right).
  - Signed: "&s" (and), "|s" (or), "°s" (xor), "@s" (xor), "~s" (negate), "<<s" (shift left), ">>s" (shift right).
- autoscale operators (v2.2.4-)
  - scale from sbitdepth to bitdepth using bit-shift method: "@B" or "scaleb" (#B until v2.2.4)
  - scale from sbitdepth to bitdepth using full scale stretch method: "@F" or "scalef" (#F until v2.2.4)
- The variable "x", "y", "z" and "a" (when applicable) contains the value of the pixel. It's an integer 0 to (2^bitdepth)-1 (e.g. 0..255 for 8 bits). For float the range is generally 0..1.0
- The constant "pi" can be used.
- The constant "bitdepth" (8-16, 32) for the input bitdepth (v2.2.4-)
- The constant "sbitdepth" (8-16, 32) as the bitdepth of constants to scale (v2.2.4-)
- Other predefined constants for bit-depth dependent values (v2.2.4-)
  - "range_half": 128 for 8 bits, 2^(bitdepth-1) in general, 0.5 for 32 bit float
  - "range_max": 255 for 8 bits, (2^bitdepth)-1 in general, 1.0 for 32 bit float
  - "range_size": 256 for 8 bits, (2^bitdepth) in general, 1.0 for 32 bit float
  - "ymin", "ymax": luma min and max value. 16 and 235 for 8 bits, shifted left by (bitdepth-8) in general
  - "cmin", "cmax": chroma min and max value. 16 and 240 for 8 bits, shifted left by (bitdepth-8) in general
- Finally, there's a ternary operator : "?", which acts like a "if .. then .. else .."
- All the computations are made in 64 bit doubles, and the final result is rounded to the nearest integer, in the range [0..255], [0..1023], .. [0..1.0] etc. depending on the clip's bitdepth.
- Throughout the whole documentation, you'll be able to find plenty of examples.

Changelog

See the Github repository for v2.2.x, or MaskTools2 b1 GitHub commit log for tp7's changes; see this page for older changes.

Exernal Links

- GitHub - Source code repository for MaskTools2 2.2.x.
- GitHub - Source code repository for MaskTools2 b1.
- Doom9 Forum - Original MaskTools2 discussion thread.
- Doom9 Forum - v2.2.x discussion thread

Guides:

- Excellent MaskTools guide by tp7 [1]
- Another guide by tp7 in Russian
- MaskTools guide in Chinese by 06_taro [2]

Back to External Filters ←

Categories: External filters | Plugins | Other filters | Support filters