Package ‘AV1R’

October 14, 2026

Type Package

Title 'AV1' Video Encoding for Biological Microscopy Data

Version 0.1.4.1

Description Converts microscopy video and image data (H.264/H.265, AVI/MJPEG, TIFF stacks) to the 'AV1' codec for compact storage, and archives 16-bit image stacks and videos losslessly in the 'FFV1' format. Encoding runs on the GPU via 'Vulkan' or 'VAAPI' when available and falls back to the CPU through 'FFmpeg' and 'SVT-AV1'. Whole experiment folders can be converted at once.

Depends R (>= 4.1.0)

License MIT + file LICENSE

URL https://github.com/Zabis13/AV1R

BugReports https://github.com/Zabis13/AV1R/issues

Encoding UTF-8

SystemRequirements FFmpeg (>= 4.4) with 'libsvtav1', libvulkan-dev and glslc (optional, for GPU encoding on Linux), Vulkan SDK (optional, for GPU encoding on Windows)

Suggests testthat (>= 3.0.0), tiff, withr

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation yes

Author Yuri Baramykov [aut, cre] (ORCID: <https://orcid.org/0009-0000-7627-4217>), The Khronos Group Inc. [cph], FFmpeg developers [cph]

Maintainer Yuri Baramykov <lbsbmsu@mail.ru>

Repository CRAN

Date/Publication 2026-10-14 16:02:15 UTC

Contents

AV1R-package 2
archive_tiff 3
archive_video 4
av1r_options 5
av1r_status 6
convert_folder 7
convert_to_av1 8
detect_backend 9
measure_ssim 9
read_tiff_stack 10
vulkan_available 11
vulkan_devices 11

Index **12**

`AV1R-package` *AV1R: 'AV1' Video Encoding for Biological Microscopy Data*

Description

Converts microscopy video and image data (H.264/H.265, AVI/MJPEG, TIFF stacks) to the 'AV1' codec for compact storage, and archives 16-bit image stacks and videos losslessly in the 'FFV1' format. Encoding runs on the GPU via 'Vulkan' or 'VAAPI' when available and falls back to the CPU through 'FFmpeg' and 'SVT-AV1'. Whole experiment folders can be converted at once.

Author(s)

Maintainer: Yuri Baramykov <lbsbmsu@mail.ru> (ORCID)

Other contributors:

- The Khronos Group Inc. [copyright holder]
- FFmpeg developers [copyright holder]

See Also

Useful links:

- https://github.com/Zabis13/AV1R
- Report bugs at https://github.com/Zabis13/AV1R/issues

archive_tiff *Archive a 16-bit TIFF stack losslessly (FFV1 in MKV)*

Description

Encodes a 16-bit grayscale TIFF stack to FFV1 version 3 (RFC 9043) in a Matroska container. Compression is lossless: decoding the archive gives back the original sample values bit for bit. Frames are processed in chunks, so stacks larger than RAM can be archived.

Usage

```
archive_tiff(
  input,
  output = NULL,
  fps = 25,
  slices = c(2L, 2L),
  context_model = 0L,
  verify = FALSE,
  sidecar = TRUE,
  verbose = TRUE,
  backend = c("auto", "vulkan", "cpu")
)
```

Arguments

`input`	Path to a multi-page 16-bit grayscale TIFF.
`output`	Path to the output `.mkv`. Defaults to `input` with the extension replaced by `.mkv`.
`fps`	Frame rate written to the container.
`slices`	Integer vector `c(horizontal, vertical)`: slice raster. Slices are coded independently (unit of parallelism).
`context_model`	`0` (small context set, default) or `1` (large context set; slower adaptation, can compress better on large frames).
`verify`	If `TRUE`, decode the archive with FFmpeg and compare it bit-exactly with the input; stops with an error on any mismatch. Requires FFmpeg.
`sidecar`	If `TRUE`, write `<output>.json` with codec parameters, versions, checksums and OME-XML.
`verbose`	Print progress messages.
`backend`	Encoder: `"auto"` (Vulkan GPU if available, else CPU), `"vulkan"` or `"cpu"`. Both produce byte-identical archives; the GPU path needs a Vulkan compute device and AV1R built with `glslc`.

Details

OME-XML found in the TIFF ImageDescription tag is stored as an MKV attachment (`metadata.ome.xml`) and in the sidecar file.

Value

Invisibly, the path to the archive.

Examples

```
## Not run:
# Requires the 'tiff' package
archive_tiff("stack.tif", verify = TRUE)

## End(Not run)
```

---

archive_video — *Archive a video losslessly as 16-bit grayscale (FFV1 in MKV)*

Description

Decodes a video with FFmpeg and encodes its luma (Y) channel, converted to 16-bit grayscale, to FFV1 version 3 (RFC 9043) in a Matroska container. Frames are decoded and encoded on the fly in chunks, so videos larger than RAM can be archived.

Usage

```
archive_video(
  input,
  output = NULL,
  slices = c(2L, 2L),
  context_model = 0L,
  audio = FALSE,
  verify = FALSE,
  sidecar = TRUE,
  verbose = TRUE,
  backend = c("auto", "vulkan", "cpu")
)
```

Arguments

`input`	Path to a video file FFmpeg can decode.
`output`	Path to the output `.mkv`. Defaults to `input` with the extension replaced by `.mkv` (`.ffv1.mkv` if the input is already `.mkv`).
`slices`	Integer vector `c(horizontal, vertical)`: slice raster. Slices are coded independently (unit of parallelism).
`context_model`	`0` (small context set, default) or `1` (large context set).
`audio`	If `TRUE`, copy the source's audio streams into the archive unchanged (FFmpeg remux after encoding; one extra pass over the file). Default `FALSE`: video only.
`verify`	If `TRUE`, decode the archive and the source with FFmpeg and compare the grayscale frames bit-exactly; stops with an error on any mismatch.

sidecar	If TRUE, write <output>.json with codec parameters, versions and checksums.
verbose	Print progress messages.
backend	Encoder: "auto" (Vulkan GPU if available, else CPU), "vulkan" or "cpu". Both produce byte-identical video streams.

Details

The archive is lossless with respect to the 16-bit grayscale frames that FFmpeg produces from the source (-pix_fmt gray16le): colour (chroma) is not stored, and 8-bit luma is scaled to 16 bits by FFmpeg. Every decoded frame is stored exactly once, at the source's nominal frame rate; variable frame rate timing is not preserved.

Use it for lossless or uncompressed sources (raw or uncompressed AVI, scientific camera recordings). FFV1 keeps every pixel of the decoded frames, including the noise and artefacts of a lossy codec, so for lossy sources (MPEG-4, H.264, H.265, ...) the archive is much larger than the original; keep the original file as the master copy instead.

Value

Invisibly, the path to the archive.

Examples

```
## Not run:
archive_video("recording.avi", verify = TRUE)
archive_video("recording.avi", audio = TRUE)   # keep the sound track

## End(Not run)
```

---

av1r_options *AV1R encoding options*

Description

AV1R encoding options

Usage

```
av1r_options(
  crf = 28L,
  preset = 8L,
  threads = 0L,
  bitrate = NULL,
  tiff_crf = 5L,
  tiff_scale = NULL,
  verbose = TRUE,
  backend = "auto"
)
```

Arguments

`crf`	Constant Rate Factor: 0 (best) to 63 (worst). Default 28. Lower = better quality, larger file. Used as fallback when `bitrate` is `NULL` and input bitrate cannot be detected.
`preset`	Encoding speed preset: 0 (slowest/best) to 13 (fastest). Default 8 (good balance for microscopy batch jobs).
`threads`	Number of CPU threads. 0 = auto-detect.
`bitrate`	Target video bitrate in kbps (e.g. `3000` for 3 Mbps). `NULL` (default) = auto-detect from input (55% of source bitrate for VAAPI, CRF for CPU).
`tiff_crf`	CRF for TIFF/image sequence input. Default 5 (near-lossless, suitable for microscopy data). Overrides `crf` when input is TIFF or image sequence.
`tiff_scale`	Scale factor or target size for TIFF/image input. `NULL` (default) = no scaling (only auto-scale to meet hardware minimum). A single number = multiplier (e.g. `2` doubles both dimensions). A vector `c(width, height)` = target bounding box (image is scaled proportionally to fit inside, preserving aspect ratio).
`verbose`	Logical. If `TRUE` (default), print informational messages about encoding decisions (e.g. TIFF CRF override).
`backend`	`"auto"` (best available: vaapi > vulkan > cpu), `"vaapi"` (VAAPI AV1, AMD/Intel), `"vulkan"` (Vulkan AV1), or `"cpu"` (SVT-AV1 via ffmpeg).

Value

A named list of encoding parameters.

Examples

```
# Default options (auto-detect backend and bitrate)
av1r_options()

# High-quality lossless-ish for publication figures
av1r_options(crf = 15, preset = 4)

# Explicit bitrate (GPU VAAPI)
av1r_options(bitrate = 2000, backend = "vaapi")

# Fast batch conversion of large TIFF stacks
av1r_options(crf = 32, preset = 12, threads = 16)
```

`av1r_status` *Show AV1R backend status*

Description

Prints the active encoding backend and available GPU/CPU capabilities.

Usage

```
av1r_status()
```

Value

Invisibly returns the active backend string.

Examples

```
av1r_status()
```

---

convert_folder *Convert all videos in a folder to AV1*

Description

Finds all supported video files in `input_dir` and converts them to AV1. Output files are written to `output_dir` with the same base name and `.mp4` extension.

Usage

```
convert_folder(
  input_dir,
  output_dir = input_dir,
  options = av1r_options(),
 ext = c("mp4", "avi", "mkv", "mov", "flv", "mpg", "mpeg", "webm", "tif", "tiff"),
  skip_existing = TRUE,
  recursive = FALSE,
  max_depth = 5L
)
```

Arguments

`input_dir`	Path to folder with input files.
`output_dir`	Path to folder for output files. Created if it does not exist. Defaults to `input_dir`.
`options`	An `av1r_options` list. Defaults to `av1r_options()`.
`ext`	Character vector of input extensions to process. Default: `c("mp4","avi","mkv","mov","flv","mpg","`
`skip_existing`	If `TRUE` (default), skip files where the output already exists.
`recursive`	If `TRUE`, scan subfolders up to `max_depth` levels deep. Default: `FALSE`.
`max_depth`	Maximum subfolder depth when `recursive = TRUE`. Default: 5.

Details

When a folder contains only single-page TIFF images (e.g. a microscopy image sequence), they are automatically combined into a single video named after the input folder.

When `recursive = TRUE`, subfolders (up to `max_depth` levels) are scanned. Each subfolder containing only TIFF images is combined into a single video named after the subfolder. Subfolders with video files are processed file-by-file. All output is written flat into `output_dir`, with subfolder names used as prefixes to avoid collisions.

Value

Invisibly returns a data.frame with columns `input`, `output`, `input_size` (bytes), `status` ("ok", "skipped", or "error"), and `message`.

Examples

```
## Not run:
# Requires FFmpeg installed
convert_folder("~/data/microscopy", file.path(tempdir(), "av1_output"))

# Recursive: each subfolder with TIFFs becomes a separate video
convert_folder("~/data/plates", "~/av1_output", recursive = TRUE)

## End(Not run)
```

`convert_to_av1` *Convert video to AV1*

Description

Converts biological microscopy video files (MP4/H.264, H.265, AVI/MJPEG) or TIFF stacks to AV1 format. Automatically selects the best available backend: Vulkan GPU (`VK_KHR_VIDEO_ENCODE_AV1`) when a compatible device is found, otherwise CPU via FFmpeg (`libsvtav1`).

Usage

```
convert_to_av1(input, output, options = av1r_options())
```

Arguments

`input`	Path to input file. Supported: .mp4, .avi, .mkv, .mov, .flv, .mpg, .mpeg, .webm, .tif/.tiff (multi-page), or printf pattern like `"frame%04d.tif"`.
`output`	Path to output file (.mp4 or .mkv).
`options`	An `av1r_options` list. Defaults to `av1r_options()`. Use `backend = "cpu"` or `backend = "vulkan"` to force a backend.

Value

Invisibly returns 0L on success. Stops with an error on failure.

Examples

```
# List available options
str(av1r_options())

## Not run:
# Requires FFmpeg installed
convert_to_av1("recording.mp4", file.path(tempdir(), "recording_av1.mp4"))

## End(Not run)
```

---

detect_backend *Detect best available encoding backend*

Description

Priority order: "vaapi" (VAAPI AV1 GPU via ffmpeg, best speed/quality ratio) > "vulkan" (Vulkan AV1 GPU, fastest but CQP only) > "cpu" (SVT-AV1 via ffmpeg).

Usage

```
detect_backend(prefer = "auto")
```

Arguments

prefer	"auto" (default), "vaapi", "vulkan", or "cpu".

Value

Character string: "vaapi", "vulkan", or "cpu".

Examples

```
detect_backend()
detect_backend(prefer = "cpu")
```

---

measure_ssim *Measure SSIM quality score between two video files*

Description

Compares encoded video against the original using SSIM. Values close to 1.0 indicate high similarity.

Usage

```
measure_ssim(original, encoded, duration = NULL)
```

Arguments

`original`	Path to original video file.
`encoded`	Path to encoded video file.
`duration`	Seconds to compare. `NULL` = full video.

Value

Numeric SSIM score (0-1), or `NA` on failure.

`read_tiff_stack` *Read a TIFF stack and return basic metadata*

Description

Does not load pixel data into R – only inspects the file. Useful for checking frame count and dimensions before encoding.

Usage

```
read_tiff_stack(path)
```

Arguments

`path`	Path to a multi-page TIFF file or printf-pattern (e.g. `"frame%04d.tif"`).

Value

A list with elements: `path`, `n_frames`, `width`, `height`, `size_mb`.

Examples

```
# Create a small temporary TIFF and inspect it
tmp <- tempfile(fileext = ".tif")
writeBin(raw(1024), tmp)
info <- read_tiff_stack(tmp)
str(info)
unlink(tmp)
```

vulkan_available *Check Vulkan AV1 availability*

Description

Check Vulkan AV1 availability

Usage

```
vulkan_available()
```

Value

`TRUE` if the package was compiled with Vulkan AV1 encode support and at least one device supports `VK_KHR_VIDEO_ENCODE_AV1`.

Examples

```
vulkan_available()
```

vulkan_devices *List Vulkan-capable GPU devices*

Description

List Vulkan-capable GPU devices

Usage

```
vulkan_devices()
```

Value

Character vector of device names. Devices supporting `VK_KHR_VIDEO_ENCODE_AV1` are marked with `[AV1]`.

Examples

```
vulkan_devices()
```

Index

archive_tiff, 3
archive_video, 4
AV1R *(*AV1R-package*)*, 2
AV1R-package, 2
av1r_options, 5
av1r_status, 6

convert_folder, 7
convert_to_av1, 8

detect_backend, 9

measure_ssim, 9

read_tiff_stack, 10

vulkan_available, 11
vulkan_devices, 11