

vid2rgb121 — Video Converter for the PicoMite

vid2rgb121.py converts an ordinary video file (mp4, mkv, avi, gif, mov...) into

frames a PicoMite HDMI/USB (or buffered-TFT) can play, plus an optional

soundtrack. A small companion BASIC player reads each frame straight into the

framebuffer — no per-pixel BASIC, no decoding in the interpreter.

It can emit three formats; pick one with a flag:

Format	Flag	Bits/px	Display	Compression	Player	Firmware decode
RGB121	*(default)*	4	MODE 2 (320×240, 16 col)	RLE (4-bit run)	playrgb121.bas	BLIT MEMORY (built-in)
RGB332 raw	--rgb332	8	MODE 5 (320×240, 256 col)	none	playrgb332.bas	direct MEMORY INPUT
RGB332 RLE	--rgb332- rle	8	MODE 5 (320×240, 256 col)	count+value (~3×)	playrgb332rle.bas	BLIT MEMORY332

All three carry the same audio sidecar (name.aud) and the same per-frame

length-prefixed envelope, so the players share one fast read loop.

1. Requirements

- **Python 3.8+** (tested on 3.14).
- **FFmpeg** — used to decode/rescale video and extract audio.

[△](#) The PyPI package literally named `ffmpeg` (`pip install ffmpeg`, ~5 kB)

is not FFmpeg. It is a thin wrapper that calls an ffmpeg binary you must

already have. Installing it does nothing useful here.

2. Installing FFmpeg

Option A — bundled FFmpeg via pip (recommended, no system changes)

```
pip install imageio-ffmpeg
```

This downloads a self-contained static FFmpeg into your Python environment.

The converter finds it automatically. Remove later with `pip uninstall imageio-ffmpeg`.

Option B — system-wide FFmpeg

Put ffmpeg on your PATH:

- **Windows:** `winget install Gyan.FFmpeg` (then open a new terminal)
- **macOS:** `brew install ffmpeg`
- **Linux:** `sudo apt install ffmpeg`

The converter prefers a PATH FFmpeg and falls back to the `imageio-ffmpeg`

binary. Check with `ffmpeg -version`.

3. Basic usage

```
python vid2rgb121.py INPUT OUTPUT [options]
```

- INPUT — any video FFmpeg can read.
- OUTPUT — the frame file to create. Audio is written beside it with the same

base name and a `.aud` extension (`myclip.vc` → `myclip.aud`).

Examples:

```
python vid2rgb121.py myclip.mp4 myclip.vc # RGB121 (default)
python vid2rgb121.py myclip.mp4 myclip.r332 --rgb332 # raw RGB332
python vid2rgb121.py myclip.mp4 myclip.r332c --rgb332-rle # compressed RGB332
```

At the end the converter prints the exact player settings (`vidFile,`

`audFile, totalFrames, frame pacing`).

 *Pick a unique OUTPUT name. The audio step writes OUTPUT-with-.aud,*

overwriting any existing file of that name.

4. Options

Option	Default	Meaning
<code>--w N</code>	320	Output width. Must match the player's framebuffer.
<code>--h N</code>	240	Output height. Must match the player's framebuffer.
<code>--fps F</code>	22	Output frame rate. The player must be paced to the same value.
<code>--rgb332</code>	off	Raw RGB332, 1 byte/pixel, no header (MODE 5).
<code>--rgb332-rle</code>	off	Count+value RLE RGB332 (MODE 5); needs the BLIT MEMORY332 decoder.
<code>--arate Hz</code>	22050	Audio sample rate. Must be $\leq 44100 \times (\text{CPUspeed}/126000)$; 22050 is safe at any CPU speed.
<code>--stereo</code>	off	Keep 2 audio channels (default downmixes to mono).
<code>--no-audio</code>	off	Skip audio extraction.

`--rgb332` and `--rgb332-rle` are mutually exclusive; omit both for RGB121.

5. Which format to choose

- **RGB121 (default)** — 16 colours, but each frame is RLE-compressed in the format BLIT MEMORY already decodes, so it plays at full frame rate with no new firmware. Best when 16 colours are enough.
- **RGB332 raw (`--rgb332``)** — full 256-colour, simplest playback (a single MEMORY INPUT straight onto the screen), but **uncompressed**: 76 800 bytes per 320×240 frame. SD read throughput (~0.95 MB/s on a typical card) caps this at **~12 fps** — see §7.
- **RGB332 RLE (`--rgb332-rle``)** — 256-colour like raw, but count+value RLE shrinks frames ~3× (measured on both flat and photographic clips), lifting the sustainable rate to **~38 fps**. Requires the BLIT MEMORY332 firmware command. This is the best 256-colour option.

6. Playing on the PicoMite

1. Copy the output file (and .aud, if present) to the SD card.
2. Set the display base resolution once (persists). For 320×240 on HDMI/USB the

base must be 640×480, halved by the player's MODE:

```
```basic
OPTION RESOLUTION 640x480,315000
`
```

The ,315000 (315 MHz) only matters for tight audio sync; any speed works without audio.

3. Edit the matching player and set the three values the converter printed:

```
```basic
DIM STRING vidFile = "myclip.vc" ' or .r332 / .r332c
DIM STRING audFile = "myclip.aud" ' set to "" if no audio
DIM FLOAT fps = 22 ' must equal the converter's --fps
`
```

4. RUN the player. ESC stops playback.

Output	Player	Mode
*.vc	playrgb121.bas	MODE 2
*.r332	playrgb332.bas	MODE 5
*.r332c	playrgb332rle.bas	MODE 5

How the players stay fast. Each frame is pulled straight into a RAM buffer with MEMORY INPUT (no INPUT\$/LONGSTRING loop), then drawn directly to the **live screen** — no FRAMEBUFFER CREATE, no FRAMEBUFFER COPY. RGB121 and RGB332-RLE decode through BLIT MEMORY / BLIT MEMORY332; raw RGB332 is copied verbatim into MM.INFO(WRITEBUFF). Writing to the scanned-out buffer can tear slightly; that is the trade for skipping the copy. Each player paces frames to its fps so the audio track stays locked.

7. Performance & throughput

Playback of the RGB332 formats is limited by **SD read speed**, not the Pico.

A 320×240 raw RGB332 frame is 76 800 bytes; at a typical ~0.95 MB/s that is only about **12 fps**. So:

- **Raw `--rgb332`**: either accept ~12 fps, or **encode at a matching rate**

(`--fps 10 ... 12`) so playback runs in real time and the audio stays in sync.

Encoding a 20–22 fps clip and playing it raw will run long (it can't keep up).

- **`--rgb332-rle`**: ~3× smaller frames → ~38 fps sustainable, so the full 20–22 fps plays in real time. Prefer this over raw whenever the firmware has BLIT MEMORY332.

- **RGB121**: RLE-compressed and 4 bits/pixel, so it is comfortably real-time at the default 22 fps.

Audio is always extracted at real-time duration, so as long as the player is paced to the encode `--fps` (and that fps is actually sustainable for the format), A/V stays in sync.

8. Notes & tips

Aspect ratio. Frames are stretched to fill `--w × --h`; a non-4:3 source

looks squashed. To letterbox instead (keep proportions, pad with black), change

the `-vf` filter in `vid2rgb121.py` to:

```
"-vf",  
f"fps={args.fps},scale={W}:{H}:force_original_aspect_ratio=decrease:flags=area,pad={  
W}:{H}:(ow-iw)/2:(oh-ih)/2",
```

Audio. Extracted as mono 8-bit PCM WAV (`PLAY WAV`). Use `--stereo` to keep two channels, `--arate` to change the rate, `--no-audio` to skip. If the source has no audio track the converter says so and writes only the video; set `audFile = ""` (or comment out `PLAY WAV`) in the player.

Resolution must match the mode. --w/--h are tied to the player's MODE.

The players ship for 320×240. To target another size, change --w/--h and the player's MODE/buffer together.

9. Troubleshooting

Symptom	Cause / fix
ffmpeg not found ...	Install FFmpeg — pip install imageio-ffmpeg (§2A) or system-wide.
Audio: skipped (source has no audio track)	Source has no audio. Normal; only the video is written.
Video looks stretched/squashed	Source isn't 4:3. Use the letterbox filter in §8.
Max %KHz sample rate on the Pico	--arate too high for the CPU speed. Use 22050, or raise OPTION CPUSPEED.
Raw RGB332 plays slower than recorded	SD can't sustain 76 KB/frame at that fps. Use --rgb332-r1e, or lower --fps (§7).
Audio drifts out of sync	Player fps ≠ converter --fps, or the fps isn't sustainable for the format.
Only available with an RGB332 display	BLIT MEMORY332 was used outside MODE 5 / a non-RGB332 panel.
Integer array too small (RGB121/RLE players)	A frame exceeded the player buffer. Enlarge DIM INTEGER buf%(...).