NESolume — User Guide

NESolume puts a retro games console between your content and your output: a real machine's raster, its palette, its attribute cells — and the ways all three fail. It runs as an FFGL effect in Resolume Arena and Avenue, and as an OpenFX plugin in DaVinci Resolve, Vegas Pro, Nuke and Natron.

It is not a pixelate filter with a tint. The picture is averaged down onto the console's raster, forced through its colour system with the attribute cells enforced, and scaled back up as fat pixels. Everything you recognise — chunky dithered gradients, colours bleeding between objects, tiles landing in the wrong places — is a consequence of those constraints, which is why heavy settings read as a machine failing rather than a video effect succeeding.

Install

Resolume (FFGL):

- **macOS** — copy `NESolume.bundle` into `~/Documents/Resolume Arena/Extra Effects/` (or `Resolume Avenue`), then restart Resolume. The build is universal (Apple Silicon + Intel).
- **Windows** — copy `NESolume.dll` into `%USERPROFILE%\Documents\Resolume Arena\Extra Effects\`, or run the installer.

It appears in the effects list as **NESolume**.

Resolve and other OFX hosts: copy `NESolume.ofx.bundle` into `/Library/OFX/Plugins/` (macOS) or `C:\Program Files\Common Files\OFX\Plugins\` (Windows) and restart the host.

The builds are unsigned; if macOS complains, right-click the bundle in Finder once, or clear quarantine with `xattr -dr com.apple.quarantine <bundle>`.

Choosing a machine

The **Console** dropdown is the heart of the plugin. Each entry sets three things at once: how many scanlines the machine drew, what colours it could physically produce, and how big an attribute cell was — the area forced to share colours.

Console	Lines	Colours	Cell
Custom	240	set by Colour Depth, 1–8 bits per channel	8×8
Game Boy	144	the four DMG greens	8×8
NES	240	the 2C02's 64-entry master palette	16×16
ZX Spectrum	192	15 colours	8×8
Commodore 64	200	the 16 VIC-II colours	8×8
Master System	192	64 (2 bits per channel)	8×8
Mega Drive	224	512 (3 bits per channel)	8×8
SNES	224	32,768 (5 bits per channel)	8×8
PlayStation	240	32,768 (5 bits per channel)	8×8

The line count is the console's; the width follows your composition's aspect, so pixels stay square on any output. **Pixel Size** scales the whole raster — 0.5 is native, up is chunkier, down is finer than the machine ever was.

Two things follow from the palette being real. Corrupted colours are still that machine's colours — a glitched Game Boy can only choose among four greens. And a generous palette clashes gently — the SNES has colours to spare, the Spectrum does not, and the controls behave accordingly.

The Picture controls

- **Colour Depth** — the Custom console's DAC, 1 bit per channel (8 colours) to 8 (16 million). The named consoles ignore it; their depth is history's.
- **Dither** — ordered 4×4 Bayer dither, scaled to the quantisation step. This is the trade every 16-bit artist made by hand: resolution spent on colour. Turn it off for hard posterised bands, up for the classic chequered gradients.
- **Attribute Clash** — how strictly an attribute cell shares its colours. Within a cell you keep your own brightness but not your own colour: at 0 every pixel chooses freely, at 1 the cell speaks with one hue and two objects crossing it bleed into each other, Spectrum-style. The NES's 16×16 cells make its clash the chunkiest.
- **Pixel Grid** — the dark boundary between fat pixels, an LCD's cell gaps. It fades itself out when a raster pixel is too small on screen to have a boundary worth drawing, so it is safe to leave up.

Distortion and Glitch

All of the damage is *scroll-register damage*: it moves which raster pixel is shown where, in whole pixels, wrapping around the edge the way a scroll register wraps. None of it invents colours.

- **Wave** — a sine on the horizontal scroll, per line. Snapped to whole pixels, because fine scroll had no fractional bits.
- **Shake** — the whole frame knocked off its sync.
- **Block Glitch** — tile pointers reading the wrong address: whole attribute cells displaced by whole cells.
- **Line Glitch** — bands whose scroll register read back garbage: horizontal tears.
- **Palette Glitch** — CRAM corruption. Affected cells have their colour channels rotated or inverted *before* the palette lookup, so the result is wrong but always legal.
- **Garbage** — tile pointers into memory that never held graphics: noise tiles, quantised through the same palette as everything else.
- **Glitch Rate** — how often the machine's luck changes. Each tick re-rolls which cells and bands are corrupted. **At 0 the corruption is a still** — a crashed machine, not a screensaver — which is exactly what you want for a freeze-frame look.

Mix is a wet/dry against the untouched input. Note the effect snaps transparency to one bit (console video had transparency or it did not); Mix is how you bring soft edges back if you need them.

Presets

The **Preset** dropdown holds eight factory machines-in-states, from *Handheld* (a healthy Game Boy, grid up) through *Front Room NES* and *Attribute Clash* (the Spectrum at full strictness) to *Dirty Cartridge*, *Corrupted VRAM* and *Kill Screen*. Picking one copies its values into the sliders; touching any covered slider hands control back to Custom. Mix is never covered — how much of the effect is in the programme is yours.

Performance

The expensive work runs at the console raster regardless of composition size: about 0.17 ms per frame at 1080p and 0.56 ms at 4K on an Apple M4 Max. Chunkier pixels are cheaper still.

If the effect does nothing

The one real failure mode is a shader that would not compile on your GPU driver, which Resolume shows as an effect that silently does nothing. The plugin writes a log that names the failing stage:

- **macOS** — `~/Library/Logs/nesolume/`
- **Windows** — `%LOCALAPPDATA%\nesolume\logs\`

Include that file in any bug report: https://github.com/stoatworks-labs/nesolume/issues.