Latency Reduction and Optimizing Performance

Many things can be set in Batocera to reduce input delay and increase performance of its emulators. First, let's go through the latency reduction settings typical to RetroArch cores (a core must explicitly support these options, you can tell by whether or not they appear in the system's **ADVANCED SYSTEM OPTIONS**):

Run-ahead

This solution approaches the situation from the game-side. When you press a button, RetroArch rewinds # frames in the past (where # is the amount of frames set in **RUN AHEAD FRAMES**) and calculates what would have happened if you had pressed the button for # frames up to the current point in time. Then it brings that frame to the moment you pressed the button. It can be thought of as doing a mini-rewind-fast-forward every time a button is pressed; most machines are powerful enough to perform this nearly instantaneously.

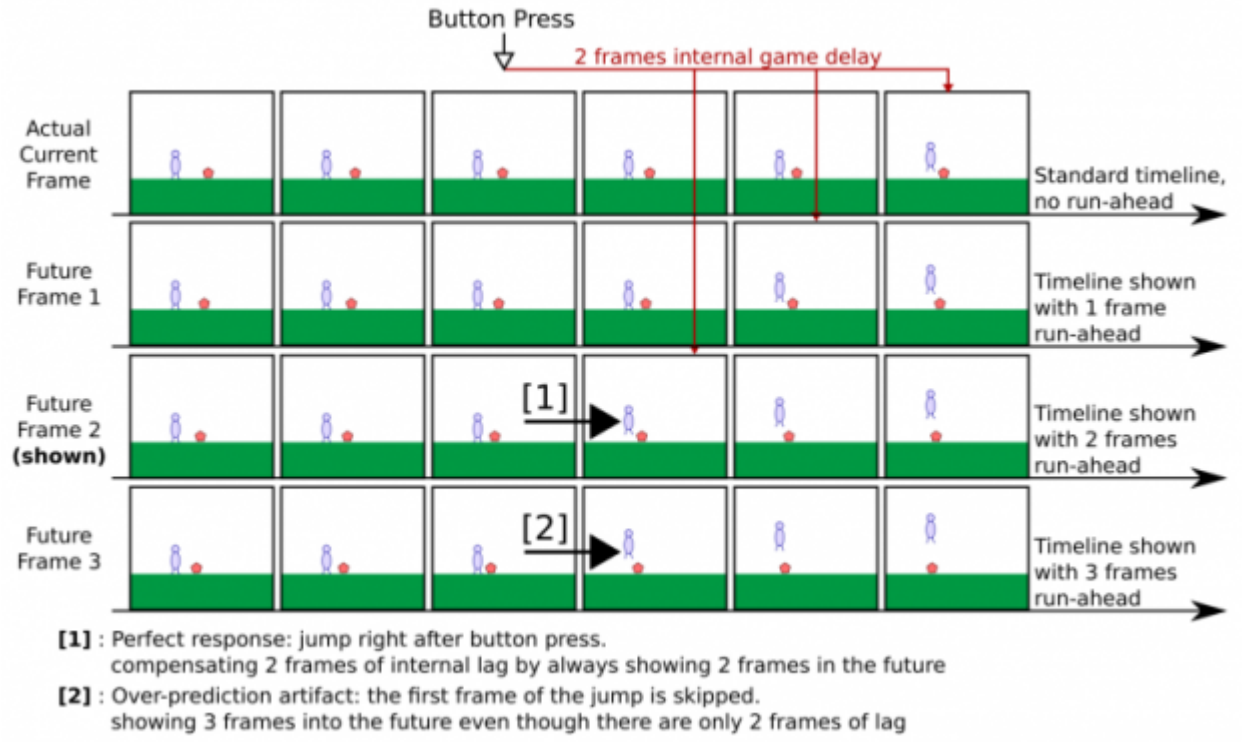

Image taken from libretro, created by Durante.

This feature is intended to reduce input lag in games that natively already have input lag in them, for instance the game itself taking one or two extra frames before it starts responding to your input (infamously, Super Mario World has two frames of input lag at all times). When used on games that instantly respond to input, it will cause the motion of the game to judder and animation frames to be skipped.

This feature cannot reduce input lag caused by your display. This setting should not be used as “compensation” for laggy displays, read below.

Be careful with the amount of frames chosen to skip. If the amount chosen is higher than the game's built-in input lag, it can result in dropped inputs and the game's motion becoming unpredictable and “choppy”, like you're playing an online game with bad network ping. This feature is very demanding and the higher amount of frames it is set to, the higher the demands are. If experiencing slow-down, this should be the first option to test turning off.

A more detailed explanation can be found on libretro's blog post introducing the feature and libretro's documentation.

Use second instance

By default, RetroArch will use the current game state being emulated to do all its calculations. This can result in choppy audio as the emulated game is rapidly fast-forwarded. By turning on the **USE SECOND INSTANCE** option, RetroArch will instead run a second version of the game in the background, and simply switch to it when the calculations are complete resulting in less audio crackle. Obviously, this doubles the CPU requirements to use, but most machines have plenty enough overhead to handle it.

Auto-frame delay

While your display renders frames, the machine in the background is processing the controller's inputs and rendering those frames independently. This delay can be felt in the extreme cases, if the machine is processing the input and rendering the frame right as the display had just finished showing the previous frame then you'll have to wait for the remaining refresh rate time period while the old frame is still being held on-screen before your current input could have an effect. Vice versa, if the machine is processing the input and rendering the frame *just before* the display refreshes, the input will have an effect almost instantaneously.

Taken from libretro's 1.9.13 blogpost.

The thing is, we're only talking about 16 ms for a standard 60 Hz display, a unit of time that is almost imperceptible in most gameplay situations. But none the less, you can manually adjust this timing to reduce/increase the delay of the rendering of each frame in relation to the display's refresh. Just keep in mind the closer the frame is rendered to when it's actually displayed, the less time the machine has to process other things such as audio and game logic, which can result in audio cutting out and/or lowered performance. Every game behaves to this setting differently and has a different “optimal” audio delay setting before negative effects set in. This can only be set in RetroArch's Quick Menu.

If that sounds bothersome to do for all your games, turning on the **AUTO-FRAME DELAY** setting analyzes the current overhead and adjusts the frame delay during gameplay to be as low as it thinks your hardware can handle. If the frame or audio stutters at all, it will increase it automatically.

The frame delay setting value setting still comes into play, that will be the “baseline” setting from which RetroArch will start making its guesses from. For more details, refer to the libretro blog post about it.

You might be wondering, how does this all apply if you are using Variable Refresh Rate? The technicalities are very complex, but essentially instead of syncing to whatever the display's refresh rate is for the delay, you're merely syncing to how much delay the machine itself is giving each frame.

G-sync/Freesync/VRR

One disadvantage that digital displays have when compared to traditional CRTs is that they run their own little computer micro-managing the display. What this amounts to is that the display's refresh rate (how often it “draws” a frame) is independent of the machine sending the frames; the computer running the emulator. The emulator has the frame ready, but must wait for the display to be ready to “draw” a new frame before the image can actually appear, causing a bit of a delay (at least when V-sync is enabled, which is the default for Batocera). In addition, digital displays typically only display in certain multiples (50, 60, 100, 120 Hz etc.) whereas game systems (especially handhelds and arcade) could have been running at any refresh rate, causing even more delay/frame judder.[1)]

But through the wonders of modern technology, on certain displays this disadvantage exists no longer!

G-sync/Freesync/Variable Refresh Rate all refer to the same concept; instead of sending frames to your display sixty times a second and having it wait until the next refresh, instead have the refresh rate dictated by the software. This eliminates the problems mentioned above with digital displays, but the downside is that the display itself must explicitly have support for the technology. Certain gaming PC monitors will have this, and a few TV displays as well.

There is usually no downside to leaving VRR on at all times. If your display's maximum refresh rate matches the exact (or in the unlikely scenario *higher*) frame rate of the content that's being displayed, then VRR would have no effect as it gets disabled by the

hardware. However, if this happens when running content with a variable frame rate (such as a PC game or an emulator that's just hard to run), this can cause (minimal) visible tearing when switching between the modes.

When using RetroArch, this will usually never happen outside of the “can't run the emulator fast enough” scenario.

For a video that goes more in-depth on the input delay aspect of VVR and its various technologies, check out this video by Battle(non)sense: FreeSync vs. G-Sync Delay Analysis. It does focus on the PC gaming aspect of it though, where variable frame rates are abundant, not necessarily the case with emulation.

Threaded video

Exchange stability for speed. Advantages/disadvantages. Quirks of splitting the threads between two cores. CPU/GPU acceleration. Practical applications.

ES options

These options are global to (most) emulators, and can be adjusted mainly to increase performance on weaker hardware.

- Turn off threaded loading for ES to allow weaker, single-core systems to run gameplay better. Only applicable to dual-core systems, really, as ES runs in the background. Dramatically slows down scraping, though.
- Graphics API options and what they do.
- Link to amdgpu/radeon driver swap trick.
- Emulator advanced system settings including options specifically for sacrificing accuracy for performance (eg. Dolphin, PSX emus)
- Use of MangoHUD/FPS counter slows performance down.
- Use of Decoration set also slows it down.
- Use of shader sets also slows it down.
- Use lower resolutions to help improve performance further, however non-native resolutions received by the TV may be prone to further post-processing to create more input-delay. Balance trade-off in unavoidable performance chokes.

Further tweaks

Now this is getting to the advanced stuff, this will mostly be hardware dependent and support for it is

“you”, you're on your own. Here be dragons.

In case that wasn't clear, a lot of these upcoming modifications will be modifying critical components of your system. These modifications can cause various issues throughout your system, and in worse cases **cause irreversible damage to your hardware**. Make regular backups and do your research properly before attempting any of these tweaks.

CPU governor

For a brief summarization on the topic: Arch wiki's entry on the topic. For a more opinionated editorial on the introduction and development of the feature: LWN's news article on its background. For a more recent update on that story: LWN's "Saving frequency scaling in the data center" article. For a more objective and in-depth documentation on the options: Kernel's official administrator documentation.

Batocera **v33** and higher feature the ability to set the CPU governor in `batocera.conf` by defining the value for the key `system.cpu.governor`. By default, it will grab the current governor profile for the first core and apply that to all cores.

This is a complicated topic and the above articles are required reading to properly understand it all and the context of when it should be used/altered, but at the risk of over-simplifying it here is a table:

Profile	Description
`performance`	Runs all CPU cores at maximum clocks at all times. It's not usually required to use this profile in the context of Batocera, as Batocera has already worked around the shortcomings of using the other profiles. Really only okay to use on desktop computers, but it's wasteful.
`schedutil`	The more modern, unified (but still lackluster compared to `pstate` on the hardware side) approach. This reads certain values from the CPU to essentially predict what it will be doing, and adjusts the clock accordingly. Functions similarly to `ondemand` in principle, but does so in a more elegant, responsive and future-proof way.
`ondemand`	Watches the percentage of CPU usage and adjusts accordingly. In application, `ondemand` is not afraid of immediately switching to the maximum clock speed when it needs to, and it will slowly lower the clock if the CPU is no longer under heavy load.
`conservative`	Watches the percentage of CPU usage and adjusts accordingly. Similar to `ondemand` in principle, but is much more conservative in how it applies the clock. A heavy load will only cause the CPU clock speed to slowly increase over time. Great for battery-powered devices that don't respond well to rapid changes in CPU clock speeds.
`powersave`	Run at the lowest CPU clock reported as usable by the system. This will obviously reduce power consumption, but at the cost of performance. Lowest amount of thermal output, however this can result in a poor experience and be wasteful due to requiring longer CPU execution time.
`userspace`	Uses the user-defined settings at `/sys/devices/system/cpu/cpuX/cpufreq/scaling_setspeed` for each core. This is not really recommended in Batocera as there is no mechanism to keep these settings permanent, outside of using an early start-up script to do so.

There's more.

But not all profiles are available on all hardware configurations. Run the following to get the current list of profiles available to your machine:

```
/etc/init.d/S18governor list
```

For example, to use the “performance” governor:

```
system.cpu.governor=performance
```

USB polling rate

It is possible to globally alter the USB polling rate, in addition to the driver's polling rate internally. USB polling rate is controlled by the `usbhid.jspoll` option, while the xpad driver's polling rate is controlled by `xpad.cpoll`. These are added to the boot line. For example:

```
    APPEND label=BATOCERA console=tty3 quiet loglevel=0
vt.global_cursor_default=0 mitigations=off usbhid.jspoll=1 xpad.cpoll=1
```

`usbhid.jspoll=0 xpad.cpoll=0` are the default settings, whatever the kernel itself requests.

`usbhid.jspoll=1 xpad.cpoll=1` changes the polling rate for USB devices and the xpad driver to be 1000mHz instead.

USB auto-suspend

Required reading: https://www.kernel.org/doc/html/v4.16/driver-api/usb/power-management.html (particularly the warnings part)

Certain USB devices will “auto-suspend” when they become inactive (ie. stop using power) and re-activate when interacted with. When this is functioning as intended, it is no different from operation without auto-suspend. However, if I had a coin for every time an electronic device functioned the way it was intended to…

It is possible to globally disable the auto-suspend option, such that no USB devices go into power saving mode. You would add `usbcore.autosuspend=-1` as an option to your boot line. So for example:

```
    APPEND label=BATOCERA console=tty3 quiet loglevel=0
vt.global_cursor_default=0 mitigations=off usbcore.autosuspend=-1
```

1)

RetroArch's timing skew feature forces games that run “close enough” to the display rate of your monitor to run the few percent faster/slower to make the game match the refresh rate, creating a smoother experience but at the obvious downside of the game being run at a different speed than native hardware. It's such a small amount that it's usually not even noticeable, but if it weren't

present then users who can't activate Variable Refresh Rate would notice issues like audio crackling. A notable example being Robocop (ran at 57 Hz, a very unusual refresh rate not supported by pretty much any digital display).