

PERFORMANCE TIPS

View online: <https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/performance-tips>

This section covers some general performance tips. Many people unnecessarily waste time when thinking about performance. This guide is aimed at a practical approach so you only investigate performance where it matters. There's also some general advice on common problems and pitfalls to avoid.

Modern computers are fast

Many people underestimate just how powerful modern computers are. Modern hardware, operating systems, browsers, and Construct itself, are all exceptionally fast and well-optimized. Many projects will have excellent performance at the end even if you never put any effort in to improving performance. That's a good thing - it means you can spend your precious time making your project better rather than trying to figure out how to improve performance. Sometimes people try to improve performance before there's even a problem, which usually means you are just wasting your time. That includes questions like "which is faster, A or B?" - usually the answer is "it doesn't matter at all and you're wasting your time".

Regularly test on target devices

If you are previewing your project on a high-end gaming PC, it is likely to be able to handle high-intensity content that would slow to a crawl on weaker devices. That might be OK for a PC game intended for other gamers, but be too slow for content designed for typical consumer devices. For this reason it is still important to regularly test your project on the kinds of devices you expect a typical player will use, to make sure you can identify any problems before publishing. The sooner you can identify a performance problem, the more likely you are to know what change caused it. The worst case scenario is to finish an entire project without testing, then find it's too slow, in which case you may have no idea why (although measurements can help, as described later). As noted previously you may find everything is running smoothly all the way through development, which is a great outcome, but it's best to be sure!

In the past, mobile devices used to have significantly worse performance to desktops and would often need special consideration for performance. However many modern high-end mobile devices are now about as powerful as mid-range laptops, and are likely to handle most kinds of Construct content fine as well. If you want to target particularly low-power budget devices, you may need to take special care though. Either way, as noted, have a target device to hand and regularly test on it so you can be sure!

Measure, measure, measure

There is only one way to get a good answer about a performance question: **measure it**. If you don't measure it, or you guess, or you ask on the forum (so someone else guesses), you may get

an inaccurate answer, and you may end up wasting your time trying to optimise something that makes no difference at all to performance. You must use measurements and be scientific about proving what makes a difference.

The ultimate performance measurement is the **frames per second** (FPS). If the FPS rate is good, there is no need to optimise anything! Don't waste your time. If it's not good enough and your project is running slowly, the key measurements to look at are the CPU and GPU utilisation. You can find all these measurements in Construct's [debugger](#) Paid plans only.

CPU vs. GPU

There are two major tasks to running a Construct project: running the logic, including all your event sheets, JavaScript code, and behaviors - done by the CPU (Central Processing Unit) - and drawing the graphics, including visual effects, done by the GPU (Graphics Processing Unit). Usually these are two different pieces of hardware (or different components even if on the same chip).

If the CPU measurement is very high, you probably need to optimise the project logic. If the GPU measurement is very high, you probably need to optimise rendering. So these measurements will guide your overall approach.

In both cases, once again, the key is to **make measurements**. Be scientific: try making a change, and see how it affects the key performance measurements. If it doesn't make a difference, the change doesn't matter to performance. If you make the right change, you will see a measurable difference. This is also how you answer "which is faster, A or B?" type questions: try both and measure them. (Often you will find no measurable difference!)

The following sections provide some tips about how to track down the cause of a meaningful performance problem that you've identified, and some of the common culprits.

CPU performance

The CPU is generally responsible for everything except drawing graphics. Event sheets, JavaScript code, plugins, behaviors, and everything else apart from graphics tend to happen on the CPU.

The best place to start looking to identify a CPU performance problem is the [CPU profiler](#) Paid plans only in Construct's debugger. This can break down CPU time spent in different areas, including individual event groups in event sheets. If you see an item measuring a particularly high usage, that is generally the item to think about optimising. For example you may see a single event group using a substantial percentage of CPU time by itself. If you look at that event group you may then find it does something like repeat an event thousands of times; adjusting that to do less work could then solve the problem.

Other times it may be less clear, such as time spent processing behaviors. Here are some general tips about the types of things that can cause high CPU usage. As before the way to check for these is to rely on measurements: try removing them, or significantly reducing their usage, and see what difference it makes to the measurements.

- **Physics behavior:** physics simulations are highly CPU intensive. Using a relatively small number of objects with the Physics behavior should be manageable. However if you use thousands of objects with the Physics behavior it could cause a considerable slowdown.
- **Creating too many objects:** while modern computers and software are very fast, everything has a cost, even if small. Creating thousands of objects can significantly increase the CPU usage, depending on what kind of logic your event sheets, code, and behaviors run.
- **Using too many particles:** while a single particle is even cheaper than an object, it is still not free. Like creating too many objects, having too many particles could also cause high CPU usage.
- **Running too many events:** if you have a very large project with thousands of events, those events will all need to be run, which may end up with a significant processing overhead. Usually most of the events don't need to be checked all the time. You can significantly reduce the number of events that need to be run by organising them in to event groups, and disabling the groups that are not currently needed. (Events in disabled groups are skipped entirely.) Sub-events can perform a similar role, as sub-events are only checked if the parent event is true.
- **Using too many loops:** using too many loops like *For*, *For Each* and *Repeat* can cause the project to slow down if used intensively. Nested loops are especially likely to cause this, as it quickly multiplies the number of iterations run. To test if this is the problem, try temporarily disabling the looping events.

GPU performance

The GPU is generally responsible solely for drawing graphics - that is, rendering your artwork to the screen, as well as any effects.

The best place to start looking to identify a GPU performance problem is the [GPU profiler](#). Paid plans only in Construct's debugger. This can break down GPU time spent on each layer in the project. If one layer is showing a high measurement, it is likely the graphics content on that layer that is responsible for the high GPU usage. For example you may see a layer using hundreds of objects all with effects showing a high measurement; removing the effects or reducing the object count could then solve the problem.

Fill rate

A key point to understand about GPU performance is the **fill rate**. Drawing pixels on the screen (also known as "filling in" pixels) requires writing them to memory. Drawing more images, and larger images, requires writing more pixels to memory. The data rate of writing all this pixel data to memory is called the *fill rate*. Once the fill rate exceeds the GPU memory bandwidth (the rate at which data can be written), the project will start to slow down as the GPU cannot keep up. Note this is a hardware limitation, not a limitation in Construct or any other software on your device.

To fully understand fill rate, it's important to also know that Construct renders projects **back-to-front**. This means it starts by drawing the background (the objects lowest down in Z order) and

progressively drawing everything else on top until it reaches the front. Since the objects at the top of Z order are drawn last, they appear on top. However when objects overlap, this involves writing to the same pixels repeatedly. This is called *overdraw* and it still uses up fill rate, i.e. the object underneath still consumes memory bandwidth, even though it is later covered up by something on top. Therefore the worst-case scenario for fill rate is a stack of large overlapping images.

There are a couple more points to consider about fill rate:

- 1** Transparent areas of images still use up fill rate. In other words, transparent pixels are still rendered, they simply have no visual effect.
- 2** Layers which use their own texture (indicated by *Uses own texture* in the Properties Bar) must be copied to the screen after the layer finishes rendering. This means every layer that uses its own texture is equivalent to rendering a screen-sized sprite that covers up everything. Having too many layers which use their own texture can quickly use up your available fill rate.

In short, the more pixels that are drawn on the screen, the more work there is for the GPU to do. This includes transparent pixels, overlapping images, and "own texture" layers.

GPU performance tips

Here are some general tips about the types of things that can cause slow rendering performance. As before the way to check for these is to rely on measurements: try removing them, or significantly reducing their usage, and see what difference it makes to the measurements.

- **Avoid objects with large areas of transparency.** Crop all images you use to remove wasteful transparent space. (This also saves memory!) Split up large objects with large transparent areas in to a series of smaller objects. For example, adding a window border using a screen-sized transparent sprite with borders drawn at the edges will perform poorly as it still has to fill a large transparent area in the middle. Splitting it in to four separate objects for each edge is much more efficient since a smaller area is rendered.
- **Avoid large areas of overlap between objects.** The overlapped area will have the pixels rendered to repeatedly, which wastes fill rate.
- **Avoid too many layers which use their own texture.** Enabling *Force own texture*, changing the opacity or blend mode, or adding an effect, all cause the layer to render to its own texture, which uses a lot of fill rate. While this is necessary in some cases to get the visual effect you want, avoid doing it too many times with layers in the same layout.
- **Avoid using too many effects.** While effects can be visually impressive, adding lots of them to layers or objects can significantly increase the amount of rendering work for the GPU. Certain types of effects are also more performance intensive than others. If you have lots of objects all with the same effect, experiment with having the objects on a layer with a layer effect, or having the effect on the individual objects; sometimes one approach can be more efficient than the other.

- **Unnecessary use of effects.** Never use effects to process a static effect on an object. For example, do not use the Grayscale effect to make an object always appear grayscale. This will degrade performance when you could simply import a grayscale image to the object and not use any effects at all.
- **No hardware acceleration.** In some cases, the GPU may not be used at all, and then the CPU is forced to perform all rendering work, which is usually much slower. This is usually caused by a system-level problem, such as out-of-date hardware or software, or broken graphics drivers. It can also be affected by browser settings (e.g. turning off "Use hardware acceleration"). You can check the hardware-acceleration status in Chrome by visiting `chrome://gpu` in the address bar. You should see WebGL listed as *Hardware accelerated* in green. Construct supports both WebGL and WebGL 2; it is sufficient if either is hardware accelerated, since Construct will pick that one.

Interpreting performance measurements

An important caveat to note about Construct's performance measurements is that the CPU and GPU measurements are **based on timers**. This makes them subject to variance due to hardware power management, which both CPUs and GPUs use.

To understand how power management affects timer measurements, consider a modern processor that can run in a low-power mode that is half as fast. In full power mode a task might take 5ms to complete. However if not under significant load, it will switch in to the low-power mode not only to save power (especially important for battery-powered devices), but also to avoid overheating the chip. In this mode the task will then take 10ms to complete. As Construct's measurements are based on timers, this means it will look like it is taking up twice as much processor time. However it is not a true reflection of the usage of the processor's full capacity.

Modern processor power management schemes are much more complicated than this, often involving many power modes that have different trade-offs between power usage and performance, and they constantly switch between them depending on the system load and temperature measurements. However the point remains that timer-based measurements can vary in unexpected ways due to power management. This can result in some unusual measurements, especially in low-power modes (when the system is mostly idle) - you might see things like the usage measurements suddenly dropping as the amount of work increases, which is actually due to the processor stepping up in to a higher power mode. The CPU and GPU usage measurements can be helpful for solving performance problems, but remember they are not perfectly accurate. In particular they are not reliable for "micro-benchmarking", such as testing which of two small tests are faster, as the measurements will mostly reflect the hardware power mode. It's another reason to not try to optimise performance until you have identified a real problem!

It's also worth noting Construct's measurements are only for your project, and the CPU measurement is only for the main thread (i.e. a single core). They are not system-wide measurements, and can be affected by other activity on the system. The CPU measurement does not measure other work done on background threads, which can also be significant in some cases, but usually does not directly contribute to the framerate.

Remember the framerate (FPS) is the ultimate performance measurement: as long as the framerate is good, then the overall performance is OK. The other measurements are there to help you diagnose what the problem might be when the framerate is dropping.

Displaying performance at runtime

While Construct's debugger displays key performance measurements, sometimes it's useful to show these in the project itself, such as for testing with [Remote Preview](#) Paid plans only or in exported projects.

The following three system expressions provide basic performance measurements.

- **fps** - returns the current frames per second rate. The maximum framerate depends on the display refresh rate, which is commonly 60.
- **CPUUtilisation** - returns the current estimated CPU usage, ranging from 0 to 1. The expression `round(cpuutilisation * 100)` will return a percentage. Note this is subject to the caveats under *Interpreting performance measurements*.
- **GPUUtilisation** - returns the current estimated GPU usage, ranging from 0 to 1. The expression `round(gpuutilisation * 100)` will return a percentage. Note this measurement may not be available on some devices (in which case it will say `Nan`, which stands for Not A Number). This is also subject to the caveats under *Interpreting performance measurements*.

You can create an in-project display of these values with a Text object to keep an eye on performance while testing your project, using an action to update it *Every tick*:

```
Set text to fps & " FPS, " & round(cpuutilisation * 100) & "% CPU, " & round(gpuutilisation * 100) & "% GPU"
```

This will display a string like `60 FPS, 30% CPU, 40% GPU` indicating the framerate and approximate CPU and GPU usage.

Common misconceptions

The following things are often accused of affecting performance but usually have little or no effect.

- *Off-screen objects* are **not** still rendered. Construct does not issue draw calls for objects that do not appear in the window, and the GPU is also smart enough to know not to render any content that appears outside the window - even when a single image is only partially on-screen.
- *Image formats* (e.g. JPEG, PNG or WebP) affect the download size but have no effect on runtime performance or memory use. All images are decompressed to 32-bit bitmap on startup.
- *Audio formats* also only affect the download size but have no effect on runtime performance.

- *Number of layers* usually has no effect. Layers which use their own texture have a performance overhead, as described above. However a layer with the default settings does not use its own texture, and has no performance overhead by itself. You can use as many of these layers as you like.
- *Number of layouts* also is unlikely to have any effect on performance. The *layout size* also does not have any direct effect; larger layouts do not use more memory or require more processing, unless you use more objects.
- *Angle or opacity* of objects and *floating-point positions* (e.g. positioning a sprite at X = 10.5) generally has no effect, since modern graphics chips are very good at handling this, even on weak devices. Using lots of very large sprites can still sometimes cause a slowdown - see the section on fill rate.

Summary

In short, here are the key points when considering performance:

- Regularly test on target devices.
- Don't attempt to optimise the project if it's fast enough. You'll just be wasting your time.
- If it's not fast enough, rely on measurements to guide your optimisation.
- Frames per second (FPS) is the ultimate measurement. CPU and GPU usage are timer-based approximations and are mainly to help you make performance measurements to guide optimisation.