



Mec

Feb 23 9:52 PM

UI and threading is a complicated topic. When Apple designed their UI subsystem, almost all CPUs had only a single core and very, very little systems had more than one CPU (leaving special tasks CPUs aside).

Either all UI code must run on a single thread or it must all be made explicitly thread-safe (e.g. with locking) and locking is always a rather expensive operation (expensive for CPUs, often also for systems as it may require calls into the kernel, which is super expensive).

So Apple went for "let's do it all on one thread" and as most systems were single core, most applications actually were single thread, so the only thread that was present for sure was the main thread. If they had created a dedicated UI thread, all data access between data manipulated by the main thread and that UI thread would have to be synchronized... again, too expensive. So they directly made it all on main thread.

It's even worse with OpenGL. OpenGL completely neglects the existence of threads.

An OpenGL context exists implicitly and is always implicitly bound to a thread; not necessarily the main thread but if you mix OpenGL with other UI, it's obvious that you will run into issues if both happens on two different threads.

On top of that, it doesn't help at all that Apple themselves uses OpenGL behind the scene to accelerate their graphics operations... at least they used to up to 10.11.

There is a reason why Khronos is virtually giving up on OpenGL and replaces it with Vulkan. Vulkan uses a completely different, more modern approach to 3D rendering that better fits the way how computers are built today (many cores, lots of threading, dedicated GPU with own RAM and highly parallelized, programmable shaders)

Apple didn't want to wait for that to happen, so they created Metal, which may be a good alternative if backward compatibility with older systems is not a requirement. It's a much simpler API but simpler doesn't mean less powerful.

Otherwise I can only recommend the following: Do your UI stuff on main thread, do your event handling on main thread and whatever else you absolutely have to do on main thread, like the final OpenGL drawing that brings your images to screen. For everything else, use different threads ... or at least use one different thread for everything else.

Ideally don't use threads at all, use queues. GCD is a highly underestimated API that makes it very easy to parallelize applications. Avoid keeping the main thread busy and avoid doing any processing on main thread that doesn't have to happen on main thread.

Today most applications do 80-90% on main thread and use 10-20% threads or queues for some background tasks... ideally it should be the other way round.

11 hours later...

GitS 

Feb 24 8:44 AM

This is amazing. You may have saved me days of work trying to figure this out. UI animation is still in the dark-ages. And i dont understand why. If you look at UI animation designs on dribbble.com Its the future of UI animation but implementing it has been close to impossible. Probably because of what you are describing here.

So im probably asking all the wrong questions right now. Forgive me im not up to speed with Core animation, 60fps animation etc. What about CADisplayLink on IOS. Its seems much more trivial to implement. And when they added CADisplayLink to IOS apple already had multiple cpus in the iPhones. Or is CADisplayLink plagued by the same issues that you are describing?

Another question: Going forward with your display and flush trick to get the UI to render on the main thread. If im understanding you right. I should attempt to do other expensive cpu tasks on other threads. So if i made a zip archiver app with a fancy animated interactive UI then if start a compression it should be on thread nr 2 and then the UI on thread 1 would still work at 60fps Dancing around with fancy interactive animation?

Third question: Your mentioning GDS as well. Ive read a trick where you use GDC as a timer

fieryrobot.com/blog/2010/07/10/a-watchdog-timer-in-gcd I guess its not as good as using CVDisplayLink?

Fourth question: Im tempted to go forward with your CVDisplayLink display and flush technique. Adding interactions and easing at first. Trying gradients, path morphing etc. A thing that came to mind last night. What about NSTextField or NSView.frame. Can these be animated with CVDisplayLink. I will test this tonight. Anyways. Thx. Surly Apple is working to fix these problems them selfs. Or maybe not. Fancy UI animation drain batteries :)

Mec 

Feb 24 9:59 AM

On iOS everything is a bit different. That starts already with the fact that all UIView objects and their siblings are always layer backed. While layer backing is optionally on OS X, it is mandatory on iOS. That also means that all UI on iOS is always composed by OpenGL ES or Metal, there is no software composing (done by the CPU) as you still have on OS X.

That's why you didn't have something like a OpenGL ESView class. If you want to use OpenGL ES on lowest level, you create an OpenGL ES context first, bind it to a thread of your choice (**any thread**), create a OpenGL framebuffer object and then get the OpenGL backing buffer of a CAEAGLLayer and attach that as the colorbuffer of the framebuffer.

Now you can draw to that framebuffer on whatever thread you bound the context to (doesn't have to be main thread) and once you are done, you present the layer content with

```
glBindRenderbuffer(GL_RENDERBUFFER,  
colorRenderbuffer); [context  
presentRenderbuffer:GL_RENDERBUFFER];
```

And this also doesn't have to happen on main thread, it has to happen on the thread that the context is bound to as using a context from any thread but the thread it has been bound to is not thread-safe as the context itself is not thread-safe! You can change context binding (unbinding it from one thread, bind it to another), but at any time a context can only be bound to 0 or 1 thread.

I've done background thread OpenGL ES rendering on iOS without any problems, much easier than on OS X and never ran into issues with that.

The more modern approach is to not render directly to layers, but use GLKit and a GLKView. This is much easier to use but I don't know what implications it has on threading. I didn't rewrite my old code so far as it still works really well.

I've never used CADisplayLink on iOS. As my OpenGL renders at a fixed framerate, lower than screen refresh as it depends on external data and there won't be 60 updates a second, this didn't seem beneficial. Again, using that link will help to get in sync with the refresh rate of the screen (not necessarily the hardware rate - which may even be variable on iOS devices)

Is is a pure waste of CPU time and battery if your code was rendering more frames than the user will ever get to see - so if your animation is limited by some external or natural factors, you should probably use such a link instead of rendering as fast as you can for no benefit.

I haven't tested what thread CADisplayLink is firing on iOS. Make a small test project and break into the link callback, then you'll see if that is main thread... most likely it is.

Regarding your zip archiver app, yes, I'd only do UI output and user input (clicking on a button, etc) on main thread and to the whole compression thing on a background thread.

There isn't much difference between a GCD timer and a NSTimer. The main difference is that a NSTimer dispatches the timer callback to a runloop while the GCD timer dispatches it to a queue. Queues are much better than threads as they are much more lightweight. They need much less resources (less system resources, less kernel resources).

A thread usually has some unswappable memory and it needs kernel resources - further there is a limit how many threads a process may have and how many threads the kernel can handle at all (across all applications). Most of the time a thread will be idle and just waste resources for nothing.

On the other hand, creating and destroying threads when needed is no good idea either, that operation is way too expensive.

A queue almost needs no resources at all, it only owns a thread when it needs one (when there is something to do). You can create 1000 queues in your app, if you don't use them, that's like nothing (little memory, no CPU time at all). All queues in the system share a thread pool (shared across

all apps!) and thus queues can get threads from the pool if required and return to the pool once done - that is super quick.

Just be careful: Code running on the same queue will not always run on the same thread! If you need a "same thread enforcement", use a real thread. But even if the thread can change any time, all code running on the same serial queue is thread-safe to each other.

For OpenGL I would create a dedicated render thread, that does all the OpenGL rendering - except for the final call to get the rendered content to screen, that needs to be done on main thread. And to not block that main thread, even if you don't plan to use any other threading: Create a serial queue and perform all actions that are not directly related to UI or user interaction with a `dispatch_async` to that queue.

Once that task is done and you just need access to the results, e.g. to display them in the UI, do a `dispatch_async` from the queue code to the main thread.

To assure thread-safety: Try to avoid sharing objects! Objects are either to be used from the serial background queue, from the OpenGL render thread or from the main thread, never from two at once. Rather dispatch data between queues instead of using things like `NSLock`

You will only require a hand full of shared classes that need to be thread-safe. Wherever possible use immutable data or copy data instead of sharing it. Copying is slow but if you need to copy a lot of data, there's probably a design problem to begin with.

3 hours later...

GitS 

Feb 24 1:14 PM

This will take some time to process, I understand it. But suspect will understand the nuances of this later when i start implementing. One thing i keep thinking is that `CVDisplayLink` wont work with `NSTextField`. And I really need this class. And also `NSTextView`. These classes are not easy to implement your self, super complex, full on text capabilities etc.

There is `CATextField` etc but they are very limited. This was the main reason for using `NSView` in the first place. Or else I would use `CALayers` for everything. And just implement interaction capabilities my self. Anyways. There are 3 links that are important right now: this first one is a working example of `CVDisplayLink`: github.com/objcio/issue-12-interactive-animations-osx

The second one shows how one could implement anim + interactions in IOS with out using `CADisplayLink` [initwithfunk.com/blog/2014/05/22/...](http://initwithfunk.com/blog/2014/05/22/) The last one shows you how you can use `CADisplayLink` and it also references the first link mentioned: objc.io/issues/12-animations/interactive-animations

A side note: Im using `NSView` with a `CALayer`. This layer then has two other `CALayers`, one that draws the fill and another

one that draws the line. If its Layer backed or layer-hosted. I don't remember, I went through weeks of work figuring out the best solution.

I needed a system that didn't clip their siblings. And I needed CALayer so that sibling views overlapped correctly. Which drawRect didn't support. I also have a notion that this will translate well when porting to IOS: Since as you mentioned its all layer-backed. Time will tell. Anyways super thanks for your explanations. Ill implement this here ->

github.com/eonist/Element

4 hours later...

GitS 

Feb 24 4:50 PM


The second link that describes how to implement anim + interaction without resorting to CADisplayLink isnt an alternative for me as it doesnt facilitate a straight forward way to implement anim + interaction. Just checked the facebook pop framework mentioned in link nr 3. It uses CVDisplayLink for osx and CADisplayLink for IOS im the same framework. So it seems like CVDisplayLink is the way to go. Hopefully NSTextField will work aswell. Ill do a test tomorrow morning.

GitS 

Feb 24 5:09 PM

Ot seems as facebook pop uses GDC and CVDisplayLink see line: 119 in this link: github.com/facebook/pop/blob/master/pop/POPAnimator.mm. Is it to fire events on the mainthread? Maybe i can use this technique to get Animate NSTextField

1 hour later...

Mec 

Feb 24 6:25 PM

Facebook indeed uses GCD to move the display link callback to main thread if `_disableBackgroundThread` is set. Of course that is a bit unfortunate as dispatching to main thread adds an arbitrary, unpredictable delay (you don't know what the main thread is doing right now) and this destroys many of the reasons why people use a display link in the first place

Regarding your comment on CALayers: If you want a `NSView` to have a `CALayer` and from there you only have sub-layers and no sub-views, that is layer hosting. You must set `CALayer` first, then you set `wantsLayer` (must be **that** order!). If your view is supposed to have child-views and you just want all views to have layers, that's layer backed (just set `wantsLayer` and never touch the layers, they are not yours, interaction with them is undesired by OS X).

You are not supposed to mix both - even though that works, but there's a lot of undefined behavior involved, so it may not work the same with any OS X version and it may as well break any time.

Textfields are indeed very hard to implement. Been there, done that. For just displaying text, `CoreText` is actually a way to do it all yourself but if you need user interaction... that's

very hard.

room mode changed to Public: anyone may enter and talk

2 hours later...

GitS 

Feb 24 8:16 PM

By your definition of layer-backed view. That is what i use., but i do set the the layer my self. Haven't had a problem with this approach. It was the only way to get unclipped sub-views. Ive probably written 20+ pages of notes about layer-hosting vs layer-backed. All on stylekit.org And i still have no idea how they differ. Hopefully it wont give me any problems. It also worked with your display and flush technique. I have a feeling that the reason why facebook pop implemented the CVDisplayLink -> GDC was to support NSTextField.

I know twitter and their TWUI framework implemented their own Text class, maybe because they could get NSTextView to cooperate on some level. I should probably fork that framework before it disappears. They haven't updated it in 4-5 years. Officially anyway. Their twitter.app that they used to use TWUI with suffers from a lot of problems though. Im not sure if they still use TWUI.

GitS 

Feb 24 9:03 PM

This guy doesn't use the "display and flush technique" but still manages to draw in the context in the main thread: github.com/00buggy00/OpenGL-Swift-and-CVDisplayLink/blob/master/... from line 122. Its even done in swift. What do you think of his approach? If it works that is.

Mec 

Feb 24 9:26 PM

I don't know regarding that new code... might be that Apple has changed the behavior and calls that only worked from main thread now work from other threads as well. As long as Apple doesn't break old code, they don't always announce such improvements and to get Metal ported to OS X, they had to add more thread-safety to their frameworks, after all Metal itself is fully thread-safe.

GitS 

What do you think about this approach: It uses CABasicAnimation and a progress-call back for every frame tick: stackoverflow.com/questions/18827973/... I mean CABasicAnimation probably uses CGDisplayLink or OpenGL internally. Just trying to weigh all options here. CVDisplayLink seems to be the way to go but it also seems like a bit of a hornets nest to be honest. Especially as you said "Apple says you shouldn't call display() directly"

Mec 

As I said, since my code still works, there was no need to change it but maybe it is much easier today. I know that VLC, the media player basically also used my approach to render accelerated video to screen but I haven't looked recently how they are doing it now.

GitS 

Nice. Then ill try that approach as well. Except a report back

tomorrow morning when i have some free time to try this out.
Can't wait.

Ok, Ill try both approaches then. I always try to find the best solution not just a solution ;) Not that yours wasn't the best at the time. Might still be. Thanks again.

Mec



CoreAniation itself is basically just an OpenGL wrapper. If you use the OpenGL profiler tool and stop a core animation app, you can see the content of animation layers are in fact OpenGL textures and there are simple shaders loaded, that compose these textures to screen (using a simple quad and applying a matrix which is the CATransform3D set on the layer). Also all the CoreImage stuff is just OpenGL (all the effects on images are done by OpenGL shaders).

I will make a small demo project tomorrow and see how 10.11 actually behaves in all these aspects, maybe its time to update my render code - and if it is just for the latest OS X version :)

GitS



Sounds good. Looking forward to see what you figure out tomorrow...Yeah i suspected apple used OpenGL or similar tech internally. As these techs are pretty battery optimised etc.

11 hours later...

GitS



I tried to get this code to work but couldnt get it to work: github.com/00buggy00/OpenGL-Swift-and-CVDisplayLink/blob/master/... Maybe ill contact him if i cant find another solution.

I experimented with the display and flush technique and it turned out that i didnt need to call display() just the regular apple recommended setNeedsDisplay() The test can be found here:

github.com/eonist/GitSyncOSX/blob/master/src/window/view/... i wont work on that test anymore today. So it will remain as is until tonight probably.

If i dont call the CATransaction.flush() after ive drawn to the CGContext instance i get this message: CoreAnimation: warning, deleted thread with uncommitted CATransaction; set CA_DEBUG_TRANSACTIONS=1 in environment to log backtraces.

That message is omitted if i do call the flush method. Makes sense.


Im also watching some wwdc videos to try and extract some nuanced information they leave behind: LIke this one: developer.apple.com/videos/play/wwdc2013/215 also have to rewatch some other ones on CALayer and core anim etc.


Also started an article to try and collect useful information on CVDisplayLink and CADisplayLink

github.com/stylekit/stylekit.github.io/blob/master/_posts/...


That is not a perma link yet. The article doesnt have a lot of information yet. To be continued.

Feb 25 8:21 AM


GitS  Moved the CVDisplayLink article i'm writing to Feb 25 8:41 AM
this perma link: stylekit.org/blog/2016/02/24/CVDisplayLink

GitS  Ill try to make this with your CVDisplayLink Feb 25 9:24 AM
flush technique tonight: Currently using some other method
to achieve it: vimeo.com/156673422


25 hours later...

GitS  Q: Will the human brain notice if a frame is Feb 26 10:24 AM
dropped at 60fps? So 59fps for one second and then back to
60fps the next? Or does the dropped frame have other
implications?

19 hours later...

GitS  A: "Unfortunately the animation still stuttered Feb 27 5:49 AM
away like an old tractor plowing through a barren potato field
during dry season." source: bigspaceship.com/ios-animation-intervals


15 hours later...

GitS  Implemented the CVDisplayLink with the flush Feb 27 8:38 PM
trick: stylekit.org/blog/2016/02/24/CVDisplayLink
Links to code and 60fps .mov file after the jump. Going to try
NSTextField for a spin tomorrow. Crossing my toes that it will
work. Or else I need to make my own Text class or continue
to search for another solution. The callback from CATransition
seems like a plausible alternative.


11 hours later...

GitS  CVDisplayLink works with NSTextField. Feb 28 7:56 AM
vimeo.com/157008305
Still Using the flush technique with setNeedsDisplay.
NSTextField I just added to a view and it worked. Going to
implement all this in a ScrollList component with the infamous
RubberBand recoil animation. Also want to try more
interactive animations with this. Stay tuned. And thx again for
the brainstorming session.

10 hours later...

GitS  CVDisplayLink + ScrollWheel implementation: Feb 28 6:29 PM
stylekit.org/blog/2016/02/28/Scroll-wheel


97 hours later...

GitS  Rubberband animation for list: 60fps Mar 3 7:00 PM
dl.dropboxusercontent.com/u/2559476/...
I had a problem clicking a list item while in motion, hopefully
its because i didnt flush the state change for that listitem. Its

using catransitions to chnage color. I readsome where that cvsisplaylink and catransitions dont play well together. Ill resolve it tomorrow probably. By skipping the catransition and animating the color tween with cvdisplaylink on its own view.

1 hour later...

trilol 

 Hello

Mar 3 8:18 PM

I am trying to understand the math behind the transformation from world coordinates to view coordinates.

This is the formula to calculate the matrix:
www.imgur.com/wiXenHp

and here is an example, that should normally be correct...: www.imgur.com/OuZBmGy where b = width of the viewport and h= the height of the viewport

But I just don't know how to calculate the R matrix. How do you get Ux, Uy, Uz, Vx, Vy, etc... ?

11 hours later...

GitS 

Trilolil: wrong chat room ;) this is about CVDisplayLink and its implications related to frame animation. As for transforms, dont try to understand them, just use them ;) write alot of examples untill something works. Then store it in a method for later retrival

Mar 4 7:18 AM

56 hours later...

GitS 

Having issues with animating the same variable more than once per frame tick. Will try to find a solution.

Mar 6 3:11 PM

1 hour later...

GitS 

Crises averted. The problem was related to a trackingarea being set outside an if clause. Anyways. Now using multiple CVDisplayLinks. One for each View. I don't know how smart this is yet but I like the decentralisation of it. I don't like using singletons, but I will probably have to use only one CVDisplayLink in the future.

Mar 6 4:39 PM

3 hours later...

GitS 

Fixed the trackingArea problem only to disable the trackingArea feature. so no mouseOver etc. Need to dig deeper with this one. Ill try again tomorrow.

Mar 6 7:46 PM

20 hours later...

GitS 

Mar 7 4:14 PM

Seems like you can only call `display()` once per tick. Or the app will break randomly, not straight away but after a while. Very random. So i guess i need to implement a centralised way of calling the final `display()` call in every frame tick. This limits the flexibility but it seems apple struggles with the same issue since they don't provide any `progress callback` method with their `CATransition` calls either.

16 hours later...

GitS 

Mar 8 8:24 AM

Ok so mixing interaction that manipulates a `CGContext` while a frame-animation manipulates the same `CGContext` didn't work great. So I was thinking back to what apple does by default, and they actually commit every call to `CGContext` as an animation tick. I always found this strange. So I attempted something similar but a bit more lightweight: What i did was that when you manipulate the `CGContext` directly via interaction you assert if the global `CVDisplayLink` is running. If it is then you delay the final `setNeedsDisplay()` call to the current frame tick in the `CVDisplayLink`. You do this by adding the final call to a global array that is looped through and called when the frame-tick happens. This scheme has the effect that all calls are done when the frame tick happens and not when the interaction event happens.

Since the interaction event probably doesn't sync to the frame refresh rate. After some stress testing (5min) this seems to work.

22 hours later...

GitS 

Mar 9 6:27 AM

My current implementation works, but the flushing still sort of happens from the background thread, im going to see if i can use `performSelectorOnMainThread` on each frame tick instead. Just feels like a better idea, any thoughts?

GitS 

Mar 9 7:01 AM

Yepp. that worked. I went through your old posts on stackoverflow and found that you actually mentioned this scheme as well. Also found it here: medium.com/@eyeplum/cvdisplaylink-a0f878f8f053#.c5tfmy3db

So now im not using the flush and display scheme at all. Just the `self.performSelectorOnMainThread(ObjectiveC.Selector("onFrameOnMainThread"), withObject: nil, waitUntilDone: false)`

And no bugs.

Best of all, All my drawing code can stay the same. The animation stuff can be ad hock. And you can include it in projects or not.

120 hours later...

GitS 

Mar 14 6:41 AM

Here is some animation I made with the result

of this research: stylekit.org/blog/2016/03/09/Demo-app

349 hours later...

Feed



This room has been **automatically frozen for inactivity**

Mar 28 7:30 PM

Conversation ended Mar 28 at 19:30.