**Rendering With Two Threads**

The Replica Island renderer is based heavily on the GLSurfaceView class that ships with the Android SDK. I've made a couple of modifications but the code is pretty similar to the regular version: a derivation of GLSurfaceView.Renderer that draws the frame gets called every frame, followed by a call to eglSwapBuffers() to actually display the rendered frame.   
  
GLSurfaceView provides a way to run user code in the same thread as the renderer. This makes writing games pretty easy; you can just implement a Runnable, implement a Renderer, stick them both into a GLSurfaceView and get stuff moving around on the screen. Indeed, it's more than sufficient for many applications; my [SpriteMethodTest](http://code.google.com/p/apps-for-android/source/browse/trunk/SpriteMethodTest) demo works this way just fine.  
  
But for Replica Island I took a different approach. The problem with the single GLSurfaceView thread is that eglSwapBuffers() must block on the hardware until the previous frame finishes drawing. That means that even if you have nothing to draw, a call to eglSwapBuffers() takes 16.67ms to complete. (And of course, if you have a lot to draw, it could take a lot longer).  
  
Now, just in case you are not used to thinking in terms of milliseconds, here's a quick primer. To achieve the magical "60 frames per second" that many games strive for, you need to have a new frame displayed to the user every 16.67 ms. If you go for 30 fps, you have ~32 ms to complete a frame. All your game code, plus all your OpenGL code, plus the actual time it takes to draw the frame must fit within 16.67 ms to achieve 60fps.   
  
In Replica Island, the game code is fairly heavy-weight. I have all that collision to run, plus updates of all the active entities on the screen, plus sound playback and all that jazz. Turns out that it's usually more work to calculate a single simulation step than it is to actually draw the frame. Since this code takes time to execute, the 16 ms block that eglSwapBuffers() incurs makes it really hard to hit 60 fps. What I really want to be able to do is run game code while eglSwapBuffers() is blocking; that way I can pipeline the game updates while the hardware is busy drawing the frame.  
  
So I split the game code off into a separate thread. This makes three threads, by the way: the main UI thread that all Activities have by default, the GLSurfaceView render thread, and this new game thread (actually, there are a few more that are generated by the system for things like orientation sensor updates, but they don't affect the equation much). Now my game code and my renderer can run asynchronously, and I win back some of that time spent in eglSwapBuffers().  
  
Now comes the tricky part. I have two threads running in parallel that need to sync up once a frame so that the game thread can tell the render thread what to do. There's a lot of ways to go about synchronizing these two threads, but I went with a double buffer solution. The game thread fills up a buffer of commands to draw the next frame, and when it is ready it waits for the render thread to begin the next frame. At that point, the buffer is passed to to the render, which can then go off and draw the next frame asynchronously. The buffer that was used to draw the last frame is passed back to the game thread, which fills it up again the next frame. So drawing is the process of swapping these two buffers back and forth during a (hopefully short) choke point at which both threads stop and communicate.  
  
This solution was attractive to me because it was simple, and so far it seems to be plenty fast. However, another solution might be to have a queue that is shared by both threads, with the game thread pushing commands in one end and the renderer executing commands out of the other. In theory such a solution wouldn't need both threads to ever perfectly align--blocking would only occur when one thread or the other was starved. But I haven't done this yet because it is going to be significantly more complex than the double buffer.   
  
My render commands are objects that are allocated out of pools that the game thread owns, and must be returned to those pools when they have been drawn. In the double buffer system, the queue that is returned from the render thread contains commands that can be safely returned to their pools, but in the shared queue system there's no obvious way for the game thread to know how much has been drawn. I suppose there could be two shared queues, one in each direction, but that would still be a lot more complicated than what I have now. Right now almost no code outside of the buffer swap system knows about other threads; the pool objects and the objects they contain are not thread safe and, as it stands, don't need to be.   
  
Is my solution the best for Android apps? I don't know. It seems to work pretty well and it is uncomplicated, which are two points in its favor. Still, I'd like to give this shared queue idea a shot at some point; my gut tells me that it will be slightly faster than the double buffer (less blocking in the average case) but a lot more complex, which might make it not worth the effort. Programmer guts are, however, extremely unreliable, so I will probably give this method a shot after Replica Island ships.

Posted by[Chris](http://www.blogger.com/profile/16079744366139636224)at[6:59 AM](http://replicaisland.blogspot.de/2009/10/rendering-with-two-threads.html)[http://img2.blogblog.com/img/icon18_edit_allbkg.gif](http://www.blogger.com/post-edit.g?blogID=2127023973845776737&postID=1291966444964013038&from=pencil)

Labels:[code design](http://replicaisland.blogspot.de/search/label/code%20design),[targeting Android](http://replicaisland.blogspot.de/search/label/targeting%20Android)

**19 comments:**

http://img1.blogblog.com/img/anon36.png

**Anonymous**[October 2, 2009 8:42 AM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1254498138957#c2579781852080971369)

Nice. Just for curiosity, you could also experiment with "triple" buffer, depends on your command objects, but, I guess it shouldn't take much memory.  
  
Regards,  
Zura

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=2579781852080971369)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Matt Fowles**](http://www.blogger.com/profile/08369073330803992975)[October 2, 2009 10:13 AM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1254503619619#c5471833307067192378)

I know that you intend to release the code eventually. Are any intermediate snapshots available?

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=5471833307067192378)

1. http://img1.blogblog.com/img/anon36.png

[**Luis Estrada**](http://www.blogger.com/profile/12578483009723941404)[October 2, 2009 1:45 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1254516309752#c5188464440886266945)

Have you tried using the Double Buffer Technique on your SpriteMethodTest demo? It would be interesting to have a demo to look at with the Double Buffer Technique.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=5188464440886266945)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Chris**](http://www.blogger.com/profile/16079744366139636224)[October 2, 2009 5:08 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1254528506413#c4523862819698551276)

>Zura  
  
I actually implemented it as a N-way buffer, and I just have N set to 2. The problem with increasing it is that it means the game thread and the render thread will begin to diverge; the game will be N - 1 frames ahead of the renderer in its simulation, which doesn't feel good. Memory's not a problem at the moment but I didn't see any advantage to using more buffers.  
  
> Matt  
  
Not yet, sorry. I am working hard on finishing it up. There are some bad crash bugs lurking in there still so I wouldn't want people basing their projects off it until I make sure it is solid.  
  
> Luis  
  
I haven't tried the double buffer stuff with the SpriteMethodTest demo, for a couple of reasons. First, the point of that test is to benchmark the characteristics of the hardware it is running on, so I don't want other sources of slowdown (like thread contention) affecting the result. Second, the actual animation update is part of the test--it runs in a single thread so that you can see how adjusting the simulation time affects your overall frame rate. That said, the renderer in SpriteMethodTest is almost exactly the same as the one in Replica Island, so it wouldn't be hard to split off another thread in the main activity and see what the performance implications of different synchronization schemes are.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=4523862819698551276)

1. http://img2.blogblog.com/img/b36-rounded.png

[**shasheppard**](http://www.blogger.com/profile/17944324790725048822)[October 7, 2009 3:15 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1254953754495#c3586080808875721432)

I assume that eglSwapBuffers() is an operation run entirely on the video hardware/GPU (when available, like with the HTC G1). If that is the case, I understand the reason to want to continue processing game logic while the hardware does his bit (blocking the rendering thread) and the game logic thread will presumably have the spotlight during the video hardware processing.   
  
If my depiction/understanding is accurate, how are you able to prove this is actually taking place and the benefit you expect is happening for the reasons you lay out?  
  
I want to know how are you able to see that the video hardware/GPU is being used. I have steered clear of breaking rendering and game logic into 2 threads because I could feel confident I would be able to prove it worked as I suspected.  
  
BTW, your work is much appreciated amongst the community and many people anxiously follow you in awe.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=3586080808875721432)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Chris**](http://www.blogger.com/profile/16079744366139636224)[October 7, 2009 5:48 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1254962896080#c1246291804098321980)

> shasheppard  
  
This is actually pretty easy to test if you are using GLSurfaceView. GLSurfaceView gives you an interface to package your game up as a Runnable, which I used for SpriteMethodTest. If you write your game that way and stick it in GLSurfaceView, you can easily time how long it takes to run your simulation and draw and produce a single frame. Then you can take that same runnable, wrap a loop around the contents of run(), and put it in its own thread and repeat the timing experiment.  
  
When I did this I found that my simulation step ran very fast--much faster than the renderer (although now that the game is a lot more complex the situation has reversed). So I put some sleep time in my game loop; specifically, if the simulation step finishes in less than 16ms, it sleeps until 16 ms have elapsed since the start of the step. This is a way to yield to the rendering thread when the game doesn't have all that much to do.  
  
The results of the timing test were that my frame rate improved fairly dramatically; it's been a while now since I switched over to two threads but I remember a 30 ~ 40% jump. The actual execution time of the draw and simulation code actually gets slightly slower because I've thrown thread-related context switches into the mix, but since the threads don't block on each other (except for the one sync point) they run in parallel pretty well.  
  
I thought about doing some more dramatic pipelining, like forcing the game code to run ONLY when eglSwapBuffers() is about to get called, but it hasn't proven necessary. As it stands, the renderer may be waiting on the game at this point--my game logic is now the slower of the two threads.   
  
But yeah, fundamentally decoupling the block in eglSwapBuffers() from the game code was pretty beneficial and fairly easy to measure with a simple profile.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=1246291804098321980)

1. http://img2.blogblog.com/img/b36-rounded.png

[**shasheppard**](http://www.blogger.com/profile/17944324790725048822)[November 1, 2009 6:15 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1257128158175#c7828635515384418003)

Chris,  
  
What are the priorities of the Threads involved (Activity, Game Logic, Rendering)?

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=7828635515384418003)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Chris**](http://www.blogger.com/profile/16079744366139636224)[November 1, 2009 6:31 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1257129098641#c8772039746283911546)

> shasheppard   
  
I don't explicitly set thread priorities. I don't think it's a very good idea to manually adjust the Activity thread priority, as that thread is owned and managed by the Android framework. I experimented with adjusting the relative priorities of Game and Render and found no particular benefit (threads under Linux do not seem to be dramatically influenced by priority), so I left them at their defaults. Also, these threads go through periods of high priority followed by periods of very low priority, so I found it easier to manage them directly by explicitly sleeping when there's no more work to perform.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=8772039746283911546)

1. http://img2.blogblog.com/img/b36-rounded.png

[**shasheppard**](http://www.blogger.com/profile/17944324790725048822)[November 5, 2009 5:16 AM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1257426994462#c4480152079233387707)

I have noticed a 2-3 FPS increase when manipulating Thread priorities (both with Thread.setPriority() and Process.setThreadPriority()), but I do not think it is worth the uncertain effects it might have. I know the input (Activity) thread was not getting nearly as much CPU time when other thread(s) had heightened priorities.  
  
The main reason I attempted the Thread priority increase(s) was to address an inconsistent (as far as I could tell) stutter introduced in the game when I added a thread for the rendering. Frame rate is still higher than without the introduction of a new thread, but the stutter makes it crap. It may be due to the way I synchronize the threads with Object.wait() instead of Thread.sleep()....I just do not know how much time each thread needs to sleep and Object.wait() makes more sense to me. It might just be thread context switching and not much I can do.  
  
I am still investigating and testing to see what is best for my setup.  
  
Have you achieved a smooth frame rate with the threading model described in your post?

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=4480152079233387707)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Chris**](http://www.blogger.com/profile/16079744366139636224)[November 5, 2009 5:58 AM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1257429534013#c3464326457127011858)

> shasheppard  
  
Yeah, I don't have any stuttering. I considered a semaphore lock too (and actually I do have one explicit wait in my game loop, though I don't think it ends up waiting very often--it's just to keep the game thread from outpacing the render thread).   
  
I also only have exactly one sync point: a function that takes the current render queue and swaps it with the one the renderer is holding. This is a synchronized function and it waits on a lock that wraps all of my actual GL dispatch commands (so the queue can't be swapped in mid-traversal).

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=3464326457127011858)

1. http://img1.blogblog.com/img/anon36.png

**Marek**[December 3, 2009 3:58 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1259884685621#c1556196901574277876)

Hi,  
  
Just a quick question. In your game, do you use GLSurfaceView.Renderer? In other words, is eglSwapBuffers() called automatically at the end of onDrawFrame() or perhaps you create your own rendering thread and call eglSwapBuffers by yourself?  
  
I'm asking because in my game I'm using GLSurfaceView.Renderer class and I'm not able to take advantage of multithreading. Looks like at the end of onDrawFrame CPU is not freed to other threads and stalls for at least 16ms.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=1556196901574277876)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Chris**](http://www.blogger.com/profile/16079744366139636224)[December 3, 2009 7:37 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1259897821569#c3309250158067624608)

>Marek  
  
I use GLSurfaceView.Renderer. It's the "render thread" that I reference in this article (GLSurfaceView does the actual thread management). I have a totally separate thread that I start up to run the game simulation. The game sim thread maintains a reference to the Renderer, and when it can it swaps the render queue with the one that the Renderer is holding. That's all.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=3309250158067624608)

1. http://img2.blogblog.com/img/b36-rounded.png

[**James**](http://www.blogger.com/profile/07131489018085848559)[January 6, 2010 1:29 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1262813364003#c1491415530598611166)

When you say "render commands", what do you mean? What does one look like? You're about the only blog out there that goes into this type of thing in any detail.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=1491415530598611166)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Chris**](http://www.blogger.com/profile/16079744366139636224)[January 6, 2010 5:12 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1262826778615#c7139258185956621949)

> James  
  
My "render commands" are lightweight objects that know how to render a specific type of primitive. For example, I have a class called DrawableBitmap, which you fill out with a texture, a size, and an x, y, z location, and insert into the render queue. In the render thread this object's draw() function gets called, which causes the texture to be drawn to the screen (using the draw\_texture extension or as an axis-aligned quad). When the render loop is over, all the command objects are reset and returned to their respective pools. I actually only have a few types: bitmap, tiled world, etc.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=7139258185956621949)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Paul**](http://www.blogger.com/profile/01089461455120009816)[March 4, 2010 9:27 AM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1267723647288#c543849883875033192)

Hello Chris,  
  
I'm just starting to implement a similar design, but am wondering how real time events are handled? (ie, having an enemy run across a screen in exactly 5 seconds)...  
  
It's probably from my lack of understanding, but is it achieved simply by having the gamethread spawn timerTasks? Or am I missing something more simple and efficient?  
  
Thanks a lot!

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=543849883875033192)

1. http://img1.blogblog.com/img/anon36.png

**Anonymous**[March 21, 2010 4:35 AM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1269171313592#c577638597982300049)

Hey,  
I wonder if threads are really needed. In PC OpenGL the drawing commands go to the gpu as soon as they are called and SwapBuffers does this: block until gpu finishes drawing to backbuffer and then swap.  
So if it works the same way on Android, you should do this in draw callback: post all geometry, update logic (while geometry gets drawn), swap buffers (should block much shorter, because gpu was rendering while in logic update).  
Ok, there may still be blocks while uploading vertex arrays, but these would be shorter.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=577638597982300049)

1. http://img1.blogblog.com/img/anon36.png

**Johan**[December 10, 2010 12:26 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1292012775275#c673653789472064053)

Hello! I'm here again after watching through your other blog posts and I implemented some of those tips.  
  
Now I have some problem with another thing, the drawing causing some "hiccups" sometimes. I believe it is my setObjectsToDrawFunction() there it put the object info to draw next frame.   
  
I wonder if ArrayList's add function could do this? Is it necesseary to create pools according to the ArrayList for this?  
  
As it it right now, it causing some random lags on my moving player. The GC runs sometimes, but sometimes it stuttering without that GC was running.  
  
Hope you can give a hint according to this.   
However, thanks!  
  
/Johan  
  
P.S: Will you talk at I/O in year 2011 too? :)

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=673653789472064053)

1. http://img1.blogblog.com/img/anon36.png

**Anonymous**[January 12, 2011 12:59 PM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1294865980235#c1116610327525377172)

OpenGL on the PC is actually buffering everything up for you just as DirectX does. Almost all 3D drivers allocate a large buffer, usually called a push buffer, and write low level operations into it. So your SwapBuffers returns as soon as the push buffer has been filled. If there isn't space in the push buffer or there are too many outstanding frames, then it will block. Then usually a DMA engine will handle kicking off the next frame. It's be a long time since I wrote drivers so it could be a little different.  
  
I suspect embedded devices don't want to pay for this large use of memory. BlackBerry does the same thing.

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=1116610327525377172)

1. http://img1.blogblog.com/img/anon36.png

**David**[July 14, 2011 5:01 AM](http://replicaisland.blogspot.com/2009/10/rendering-with-two-threads.html?showComment=1310644864647#c8842467959927174170)

My solution is that the same object have the physic and the drawing component.  
  
From Game Thread I process the physics and update the drawing component. From Render thread I execute a draw() operation and paint the object.  
  
The problem can be that while I am drawing, the position of an object can change And the frames can merge.  
  
And I don't align the Render Thread and the Game Thread. I don't now if the RenderThread it's drawing 10 times for each frame in GameThread.  
  
I have to sleep the RenderThread when don't have nothing to do, really?

[Reply](javascript:;)[Delete](http://www.blogger.com/delete-comment.g?blogID=2127023973845776737&postID=8842467959927174170)

[**Add comment**](javascript:;)