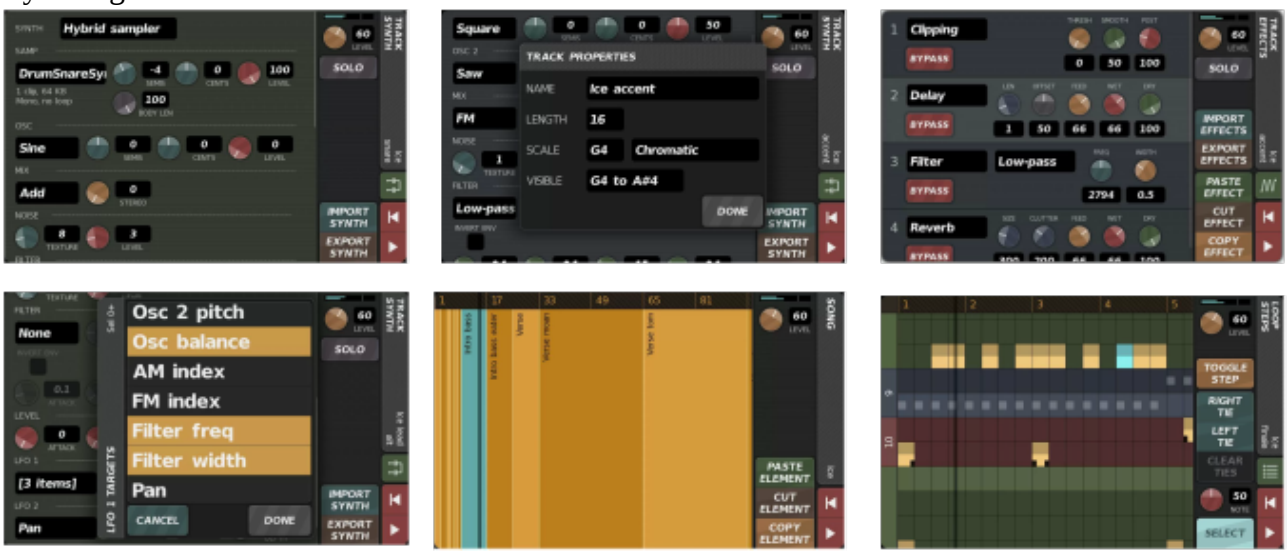
Syntheogen Article for Musical Android



Jeremy Neal Kelly

DESIGN

Syntheogen ultimately was inspired by a second-hand drum machine I bought some twenty years ago. The machine was a Yamaha RY-10; it had a row of sixteen tiny buttons in the middle, each with a red LED above, and it was the first step sequencer I had used.

Though I love hearing music, I've rarely enjoyed making it; practice is a bore, synthesizer interfaces are maddening, and I always seem to lack the one cable I need to record my amazing riff. Step sequencers are the exception to that rule. They are fun, easy, and immediate; the step sequencer is the only interface I've seen where pressing buttons at random is actually a great idea.

They do have limitations, however. The hardware sequencers I've used offer only one row of buttons, so you cannot view or edit multiple tracks at once. Work on the 'vertical' axis — whether pitch, volume, or whatever — is neither convenient nor enjoyable, especially when bending notes. Worst of all, traditional step sequencers perform poorly outside of quadruple time, since you cannot use odd-numbered meters without losing some of your buttons. I started to make a dub track on my EMX-1, but I had to give up because triplets were such a pain.

Syntheogen is an attempt to escape those limitations. Despite the popularity of skeuomorphism, I

say that software can and should transcend the limitations of hardware. In software, we can have as many buttons as we want, and those buttons can even move or change shape. At the beginning of this project, I knew I wanted a two-dimensional array of steps that would allow all tracks to be viewed and edited together. Pitched tracks would occupy multiple rows, allowing melodies and chords to be entered straightforwardly. I wanted an easy way to bend notes or entire chords. I wanted a way to divide the grid into different lengths, so that triplets and unusual time signatures could be used. I also wanted to have patterns with different lengths in the same grid, so that polyrhythms could be programmed easily. These are all things I had tried to do with hardware sequencers, but found to be difficult or impossible.

The result of all this was the Syntheogen LOOP STEPS dialog:

Where sequencing is concerned, Syntheogen offers some advanced features that (as far as I know) other Android apps lack; on the other hand, certain 'standard' features like controller automation aren't implemented at all. In some cases, I haven't had time to develop what I want, but so far I've excluded automation intentionally for reasons that relate to my design philosophy for this project.

Generally, I don't like the way automation is implemented in the applications and hardware I've used. The typical approach — where the user places the device in an automation record mode, then manipulates the control in real time — does not satisfy me. Usually there is no way to view the data without playing back and watching the controller, and no way to edit it without recording again. This design hides automation data the same way one-line step sequencers hide pitch and volume data, and that's not what Syntheogen is about.

Unfortunately, I haven't found a better way to implement this. In my work developing 'line of business' software, elegance is not required or typically even noticed. If a feature is requested, I must implement it, one way or another, even if the result is a bit awkward.

But this is a different type of project. I'm not a professional musician, and I don't expect Syntheogen to be used by many who are. My users and I make music for fun, so I think Syntheogen itself should be fun. Therefore, in this app, I would rather omit a feature if I cannot implement it in a fun and direct way. Not everything in Syntheogen meets this standard right now, but that is my goal. This approach will limit the app, in a sense, but I would rather do a few things well than do many things poorly. So, if I find a good way to implement automation, or any other feature, I will add it; otherwise, I plan to stick to the things I can do well.

DEVELOPMENT

Syntheogen was written in C++, which I know and like better than any other language. C++ supports object-oriented development, which is important for larger projects, yet it lets the developer work very close to the machine when performance is important. It also supports a resource-management strategy called RAII (Resource Acquisition Is Initialization) that is better and more flexible than the garbage collection offered by Java.

Unfortunately, Android is very much a Java platform, and does not give first-class status to C++ apps. Parts of the platform are represented in the Android NDK (Native Development Kit) with 'native' libraries that provide direct access to Android features from C or C++ code. Most platform features cannot be accessed this way, however; they can only be reached by passing through a layer called JNI (Java Native Interface) that 'translates' function calls to and from the format used by Java. JNI is difficult to use, and somewhat dangerous, as even small mistakes can crash your application. For this reason, many native developers use JNI — and by extension, much of Android — as little as possible.

This issue required that I implement my own window-management and UI control library, since using Android's controls would have required hundreds of JNI calls between the UI and the synthesis engine. Developing a full-featured UI framework is a big task, but it's something I've done before, and by relying on Android as little as possible, I was able to make Syntheogen largely platform-independent. In fact, Syntheogen was mostly developed in Windows, with Visual Studio. Even aside from the UI framework, this created a lot of extra work, but I've developed mobile and embedded applications this way for many years, and it's always turned out to be worth it. In this case, it allowed me to do most of my debugging in Visual Studio, which is fortunate, as the Android NDK debugger is almost unusable.

One regrettable early decision was to use version 1.1 of OpenGL ES rather than version 2.0.

Version 1.1 is simpler, and I had used it before, but to do any serious work with OpenGL you really have to use shaders, an advanced technique provided with version 2.0 for filling shapes with images or patterns. Having chosen version 1.1, I was forced to use stencils when clipping patterns to round corners, and that is a poorly-documented, convoluted, and slow solution to an otherwise simple problem.

Another questionable decision was to make the UI layout completely independent of the display aspect ratio, to the extent that black bars are not displayed, yet the images used to render controls are never distorted horizontally or vertically by stretching. I wanted to use as much of the display area as possible, and to render all straight lines with pixel-perfect sharpness, but this complicated the way controls are laid-out and rendered while simultaneously limiting the sort of patterns and gradients I could use to decorate them. A few years ago, when it was possible to see the pixels on an average display, this might have made a noticeable difference, but today it is worthless unless you're using a jeweler’s loupe. I will have to replace a lot of the rendering code before I can improve the application's appearance much further.

SYNTHESIS

The sequencing and synthesis engine presented numerous challenges.

Sequencing is much harder than it looks; in Syntheogen, patterns repeat within loops, loops repeat within songs, and a particular step may be tied on one or both sides to other steps, even steps in other pattern iterations. Simply determining what steps will play in a given span is very difficult, and I'm surprised sometimes that it works at all.

Syntheogen is my first audio application, and I developed the synthesis engine from scratch, so there was a lot of theory to be learned. I studied Dodge and Jerse's 'Computer Music', and I read much of Curtis Roads' 'The Computer Music Tutorial'. The Dodge and Jerse book was useful, but it contains serious gaps that a working synthesis developer must fill elsewhere. It also contains frustrating math errors that were especially aggravating given the book's ridiculous price. The Roads book is very popular, but I found it only occasionally worthwhile. Though it's very large, the book's coverage is surprisingly superficial, and I can't forgive an author who wastes my shelf space with a full-page photograph showing me what a compact disc looks like. I had to go to the KVR Audio forums to find a good general-purpose filter, and to learn more about reverberation algorithms.

There's a rule that warns software developers not to optimize any length of code unless they know with certainty that a bottleneck exists there. Normally I'm careful to honor this rule, but I rarely felt I had that luxury when developing Syntheogen. In most applications, the processor spends much of its time idling while waiting for user input. In a synthesis app, during playback, the processor

receives a constant stream of lengthy tasks, and it can idle only if it finishes a given block before receiving the next. My constant worry was that, if I did not optimize everything in the synthesis path, I would end up with an app that perhaps contained no bottlenecks, but was instead everywhere too slow. In a sense, the entire synthesis engine constitutes a bottleneck, simply because of the way it is used. Optimizing so liberally created a lot of code that was difficult to write and remains difficult to read. I've been pleased with Syntheogen's performance, though, so I think that was the right approach.

PLANS

I'm generally happy with how Syntheogen has turned out. It's interesting to compose a song with a synthesizer you wrote yourself, and to know in detail that everything you hear is the output of some relatively simple math operations.

There is still a great deal of work to be done, though. Some obvious omissions, like chorus and phase effects, must be remedied. Advanced users will want sample editing and MIDI export capabilities. None of these things are especially difficult, just time-consuming.

Now that I've used Syntheogen awhile, I find some tasks to be a bit awkward. When setting up a new loop, the user must create many loop elements and tracks, and I would like a way to automate that. Also, when setting synthesis parameters, there is no way to hear your changes without playing a pattern, which you may not have yet. I would like some way to audition the patch from the TRACK SYNTH and TRACK EFFECTS dialogs, but I'm not sure I want to sacrifice the display area needed for even a small keyboard. These and other issues will be addressed, and naturally, I'll be raising the price as I do so.

I could attempt to produce a full-featured DAW, but I don't think that's a realist goal for mobile devices, and it's also not what I want to use. What I really want is something like a harmonica. The harmonica can't do everything, but what it does do, it does better than anything else, and it does it in a way that it is compact, durable, inexpensive, easy, and fun. Hopefully, by narrowing my focus, I can approach that standard with Syntheogen.