Graphics for JVM

Let's say I want to build high-quality desktop apps. I also want to do it on JVM. Don't get your hopes up—we are not there yet. But I have a plan.

Why JVM?

It's high level enough—performant, yet doesn't make you overthink every memory allocation. It is cross-platform. It has great languages — Kotlin, Scala and, of course, Clojure. C# would do, too, but it doesn't have Clojure.

Can't you already build desktop apps on JVM?

You can. But traditionally AWT, Swing, and JavaFX came with a lot of quality and performance drawbacks. They were so significant that only one company managed to build a decent-looking app in Swing. It is possible but requires a tremendous effort.

Aren't all Java UIs cursed?

No, not really. AWT, Swing and JavaFX have their problems, but they are *their* only. There's no fundamental reason why high-quality UI can't be built on JVM. It just wasn't done yet.

Why hasn't it been done yet?

Patience, I think. We are so used to things we can hack together in a week, nobody is thinking in terms of years. And good UI requires years of work. It's a big commitment.

Why not Electron?

The first reason is performance. JS is a great language for building UI, but it is much slower than JVM. Wasm can be fast but implies C++ or Rust.

The second is the DOM. It is a horrible collection of hacks that make simple things hard and hard things impossible. I have thought many times "if only was I drawing this control/layout directly, I would've finished *hours* ago."

That means there's a very low ceiling, performance-wise and quality-wise, of what a web app can do. I believe we can, and should, do better.

Electron taught us two good things, though:

- People *crave* for native apps. Nobody wants to work from the browser.
- People don't care if apps don't look native to the platform as long as they look good.

Is desktop still relevant?

I believe it is!

I watched an interview recently, between an Android developer and an iOS developer. One was asking:

"Does someone still writes desktop apps?"

To which the other answered:

"I have no idea... Maybe?"

Both of them were recording it on a desktop, in a desktop application, while having a call over another desktop application. Multiple other desktop apps were probably used for post-production. None of those were written by magic elves or left to us by a mighty ancient civilization. The desktop might be less trendy, but only because it's harder to sell useless crap here.

And I've been on both sides. I lived without a desktop for a few weeks once. You get used to it, but it's certainly not ideal. Any sort of information gathering and processing is very painful: it's hard to select text, hard to search on a page, hard to have multiple tabs, hard to move data between apps. For example, you are adding an event to the calendar. You need to lookup an address for the event in the mail, which has a link that opens a browser. By the time you found what you needed and returned to the calendar, it has been unloaded from memory and all context is lost. *Ability to have multiple windows open at the same time is the desktop's superpower.*

Phones are great for small, quick, single-purpose tasks. They have their place, but life is way more complex than a phone can handle. Many of us still need that bicycle for the mind.

Ok, what are you proposing?

The road to high-quality UI on JVM is a long one. We'll need:

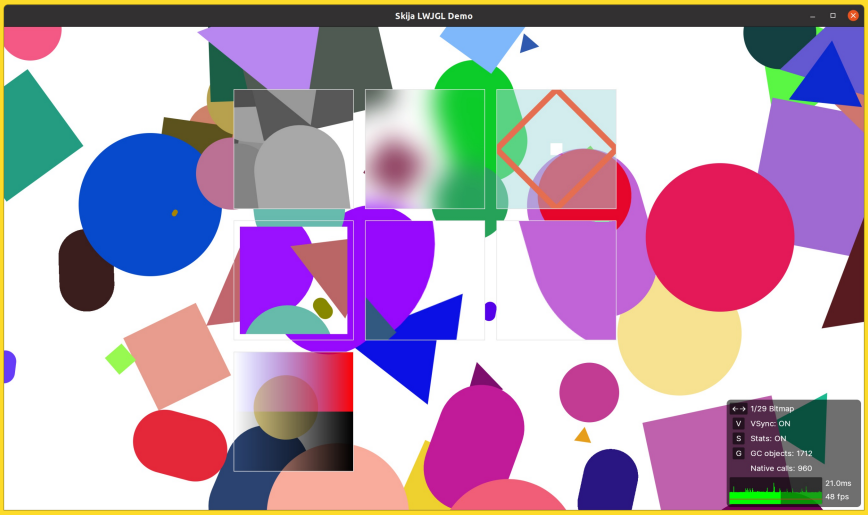
- a graphics library,
- a window/OS integration library,
- a UI toolkit.

Today I am happy to announce the first part of this epic quest: the graphics library. It's called Skija and it is just a collection of bindings to a very powerful, well-known Skia library. The one that powers Chrome, Android, Flutter and Xamarin.

GitHub.com/JetBrains/Skija

Like any other JVM library, it's cross-platform. It runs on Windows, Linux and macOS. It's as simple to use as adding a JAR file. Much simpler than massaging C++ compiler flags for days before you can even compile anything. Skija takes care of memory management for you. And the bindings are hand-crafted, so they always make sense and are a pleasure to use (as far as Skia API allows, at least).

What you can do with it? Draw things, mostly. Lines. Triangles. Rectangles. But also: curves, paths, shapes, letters, shadows, gradients.



Drawn with Skija

Think of it as a Canvas API. But, like, really advanced. It understands color spaces, modern typography, text layout, GPU acceleration, stuff like that.

Drawn with Skija by Robert Felker

Oh, and it's fast. Really fast. If it's good enough for Chrome, it will probably be enough for your app too.

What can I do with it?

Many things! Custom UI widget libraries and whole toolkits, graphs, diagrams, visualizations, games. For example, we've played with implementing `java.awt.Graphics2D` and running Swing on top of it—seems to work fine.

Why release a separate graphics library? How is it useful?

I am not a big fan of bundling everything together in one package. You can never guess all parts right—someone will always be unhappy about a particular decision.

Independent, interchangeable libraries, are more flexible. Unix-way.

What's with the rest of the puzzle?

Both things are in the works in JetBrains.

- For window management/OS integration, there's Skiko. It says that it is Skia for Kotlin, but it also implements window creation, events, packaging, the whole deal. It even integrates with AWT and Swing.
- And for UI toolkit there's Compose Desktop. It's a fork of Android Compose, a declarative UI framework, that runs in a desktop environment.

But the beauty is that it doesn't even have to be these two!

Don't like AWT? Bring your own window library.

Kotlin is not your cup of tea? Use any other JVM language.

Compose performs poorly on your load? Pray for someone to write an alternative, or write your own (sorry, no good solution yet. It's still early days).

And, by all means, if you *want* to roll your own window library or widget toolkit—please, do! That's what we hope to happen.

In conclusion

Skija is a part of the bigger picture. Java UI progress was blocked by the poor-performing Graphics2D. It's not anymore. What will grow out of it? Time will tell.

Please give Skija a try, and let us know what you think. Or, maybe, start using it—we'd be happy if you did! Here's the link:

https://github.com/JetBrains/skija/

November 14, 2020