Writing a static blog with Typst

Tim Peko (TimerErTim)

Let's dive into the world of Typst and see how we can use it to power this very blog website. We will investigate potential solutions, the custom build process I settled on and the technologies necessary. For static hosting highly efficient compression and patching is necessary.
Or in other words: How to proberly misuse Typst for everthing.

Table of Contents

1 Introduction

Ever since I started programming at around 14 years old I have been fascinated by interactively exploring ideas and finding problems to my mediocre solutions. The projects coming out of that were slowly but almost always for sure forgotten over time, even by myself. In the last few weeks I figured that to be a real shame and that I should start writing down my ideas and solutions to them, especially because technical writing is a skill I enjoy and want to improve.

Because I don't do things by halves, I intent to reignite my online persona **TimerErTim** and use it to cover all these technical/semi-professional aspects of my life. It shall be used as a central point of reference for myself, including videos on **YouTube**, career and professional information, as well as (probably very unregular) blog posts on **my website**.

Now, there is one problem: The website does not yet exist. So let's go build it. Let's use **Typst** for writing the blog posts.

2 Why Typst?

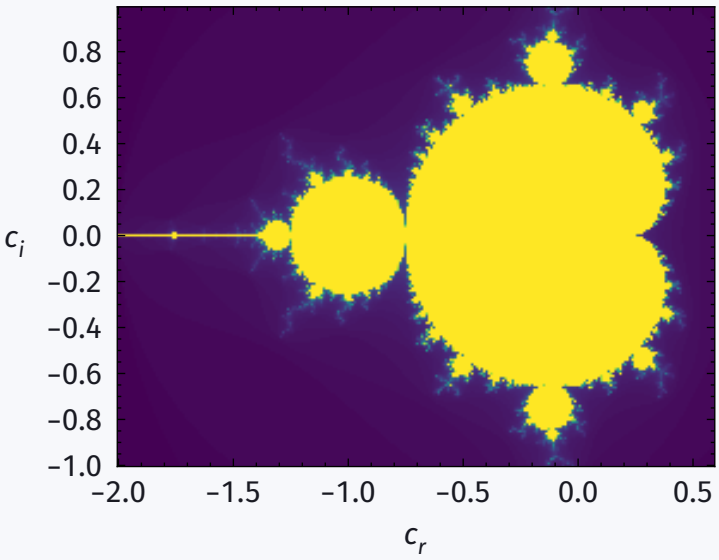
> **But why not use Markdown or some of the 1000 blog static site generators out there?**
>
> –You probably

Very good question. There are a few reasons for that, most importantly: **It's boring**.
Writing code is fun and with our custom implementation only our imagination[1] is the limit.

Furthermore, I really enjoy Typst for its capabilities and yet simplicity. It is a joy to write and the results are beautiful. Show me where Markdown can do this:

$$z_{n+1} = z_n^2 + c \tag{1}$$

Image of Equation 1



The above Mandelbrot set is generated entirely in Typst during document compilation with the following code:

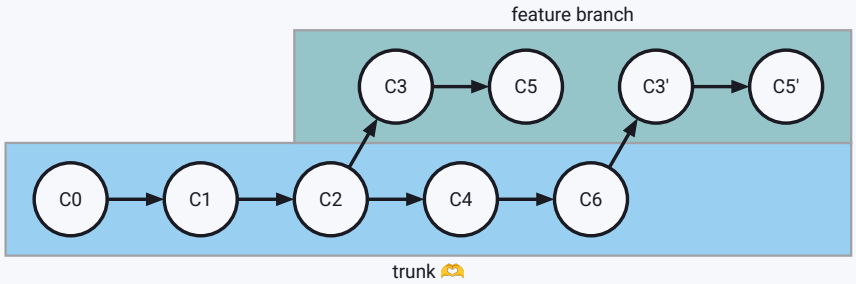
[1] Unfotunately artistic creativity is none of my strengths

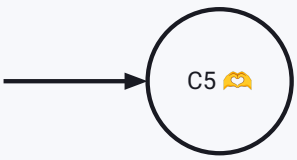
```typst
let iterations(c_r, c_i, max: 100) = {
  let z_r = 0
  let z_i = 0
  for i in range(max) {
    let z_r_next = z_r * z_r - z_i * z_i + c_r
    let z_i_next = 2 * z_r * z_i + c_i
    if z_r_next * z_r_next + z_i_next * z_i_next > 4 {
      return i
    }
    z_r = z_r_next
    z_i = z_i_next
  }
  return max
}
```

It is very easy to style and with correct abuse can be used to represent the central source of truth for the whole website's look and feel. More on that in Section 4.

3 The naive approach

With the tool choice being made, my first idea was to use the **typst-ts** project of the fantastic **Myriad-Dreamin** to create a React component that would render the blog post. It is a fantastic tool specifically designed for responsive embedding of Typst documents in web applications.





Later I dug into the **shiora**'s source code, because it being a static site generator using only Typst was exactly covering my usecase. Probably a skill issue but I could not figure out why the *typst-ts* integration worked beautifully there but not in my own project. It used a similar precompilation process but the browser rendering was way more low level. Anyway, for me it looked something like this:

4 Centralize look and feel

Topic collection:

- Introduction/General Goal
- Why Typst?
- typst-ts react tried, shiora reference, but wtf
- Custom process for creating embeddable svg documents
  1. typst-ts-cli build as svg_html
  2. replace `<image>` with svg base64 data with decoded svg content inline → allows embedding html content in typst (demo something iframe idk)
- How to make responsive in website
  1. Build multiple variants of the same document with different widths and themes
  2. Delta compress all of them with custom xdelta3 bindings to arbitary reference
  3. Brotli compress the delta patches
  4. Store these compressed files as base64 in the static website (handled by server components of NextJS)
  5. Decode with fitting browser side decompression library depending on browser div width and theme
- How does PDF download work?
- Centralized look and feel for blogs and website
- Potential room for improvements
  - Remove flickering of iframe content when new size loads
  - More powerful blog description/preview content support, currently onyl metadata string
  - Better compression by comparing all combinations of refs ↔ target
  - Replace custom update time detection with manual metadata field

Brotli customDictionary compression to python xdelta3 to create a vcdiff patch.