2025-05-31

Myriad Dreamin Blog 2025-05

Archive of Blog posts in May 2025.

Contents

音律 2
1.1 五度相生律[1] 2
1.2 十二平均律 2
Maintaining GCC 3
2.1 List G++ Packages 3
2.2 build-essential 3
2.3 update-alternatives 3
2.4 Using specific version of gcc 3
2.5 Checking versions of configured gcc 3
2.6 GNU Toolchain Directory Layout 3
重温旧曲 4
和 Space Sniffer 说再见 5
4.1 Backend API: Event Iterator 5
4.2 CLI Frontend 5
4.3 Slint GUI Frontend 5
4.4 性能 6
4.5 可能的改进 6
4.6 后记 6
My Blog Setup without Backend 7
5.1 Using Cloudflare DNS 7
5.2 Verifying the ownership of the domain on GitHub 7
5.3 Configuring the GitHub Pages 7
Tinymist 2024 - 语言服务器部分 8
6.1 LSP 引擎库 8
Typst Syntax 9
7.1 Raw Blocks 9
7.2 Equations 9
7.3 Images 10

[1] 维基百科：五度相生律

2025-05-12

音律

一些关于音律的数字游戏。

1.1 五度相生律[2]

从一个起点音[3]出发, 毕达哥拉斯发现比例为$1:2$的音程最为和谐, 这个比例被称为纯八度。其次, 比例为$2:3$的音程也很和谐, 这个比例被称为纯五度。其中的规律是: 比例越简单的音程越和谐。若我们令$r=3/2$则有:

C	$r^2C/2$	$r^2D/2$	$2C/r$	rC	rD	rE	$2C$
C	D	E	F	G	A	B	C²

Figure 1: 五度相生律

1.2 十二平均律

五度相生律很好，但是它并不适合传统作曲和演奏。特别是C♯与D♭的音高不同，给升调、降调以及和弦的推导带来了困难。为了解决这个问题，音乐家们发明了十二平均律。它的原理是将一个八度音程分成12个相等的音程，每个音程的比例为$2^{1/12}$。在这种情况下，令r的值为$2^{1/12}$，十二平均律则为:

C	r^2C	r^4C	r^5C	r^7C	r^9C	$r^{11}C$	$r^{12}C$
C	D	E	F	G	A	B	C²

Figure 2: 十二平均律

十二平均律的代价是其音程并不完美，例如纯五度变为了$1.4983C$而非$1.5C$。

[2] 维基百科：五度相生律

[3] 在 FL Studio 等宿主软件中，这个音一般是 A，且$A=440$ Hz

2025-05-12

Maintaining GCC

Some useful commands to maintain gcc globally.

2.1 List G++ Packages

```
sudo apt list "gcc-*" | grep -P "gcc-\d+\/"
sudo apt list "g++-*" | grep -P "g++-\d+\/"
```

2.2 build-essential

```
sudo apt-get install build-essential
```

2.3 update-alternatives

```
sudo apt-get install update-alternatives
```

2.4 Using specific version of gcc

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 7
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-7 7
```

2.5 Checking versions of configured gcc

```
sudo update-alternatives --config gcc
sudo update-alternatives --config g++
```

2.6 GNU Toolchain Directory Layout

- `bin` - Executable files
- `include` - Header files
- `lib` - Libraries
- `libexec` - Executable files for internal use

The directory layout is important, because `gcc` will find files in these directories by path relative to `which gcc`.

2025-05-17

重温旧曲

重温我以前喜欢的歌曲。

最近学习了一些乐理知识，逼着自己扒了一些歌的和弦。为了保证正确性，将一首歌的细节反反复复听，似乎也是练了一些走向、色彩和动机的感悟能力。

今日偶然听到一首从前喜欢的歌，脑中不由自主的扒起了和弦，好像听得更懂了。然后便有些呆愣住了，这首歌的和弦和一首古典曲子的和弦是一样的。我开始想象，作者是如何是发挥和弦，以及如何利用和弦创造属于这首歌自己的色彩。CM7 与 Fm7 来回交错，构成了静谧的氛围。可能是巧合，也有可能是常规的作曲手法。

感觉到一丝不确切的共鸣感。我觉得还能再倒回去听很多以前喜欢的歌。可能会发现一些歌的瑕疵，也可能更喜欢某些歌。无论如何，似乎都值得把以前喜欢的那些歌重新拿出来听一听。

2025-05-20

和 Space Sniffer 说再见

我写了一个高性能工具来替代 Space Sniffer。

Space Sniffer 是一个很不错的工具，但是最近我不满足于他较慢的速度，自己写了一个扫描工具。发布页面在这里。

4.1 Backend API: Event Iterator

我将 shr 分为了前端和后端。后端接受参数，扫描文件系统，返回一个迭代器，每项是一个事件更新，这样前端就能不断从迭代器中获取事件，更新 UI。

4.2 CLI Frontend

`shr-cli` 是一个类 `du` 的命令行工具，使用了 `shr` 的后端 API。它支持所有主流平台。

4.3 Slint GUI Frontend

`shr-browser` 是一个 Slint GUI，使用了 `shr` 的后端 API。出乎意料的是，它也支持所有主流平台。

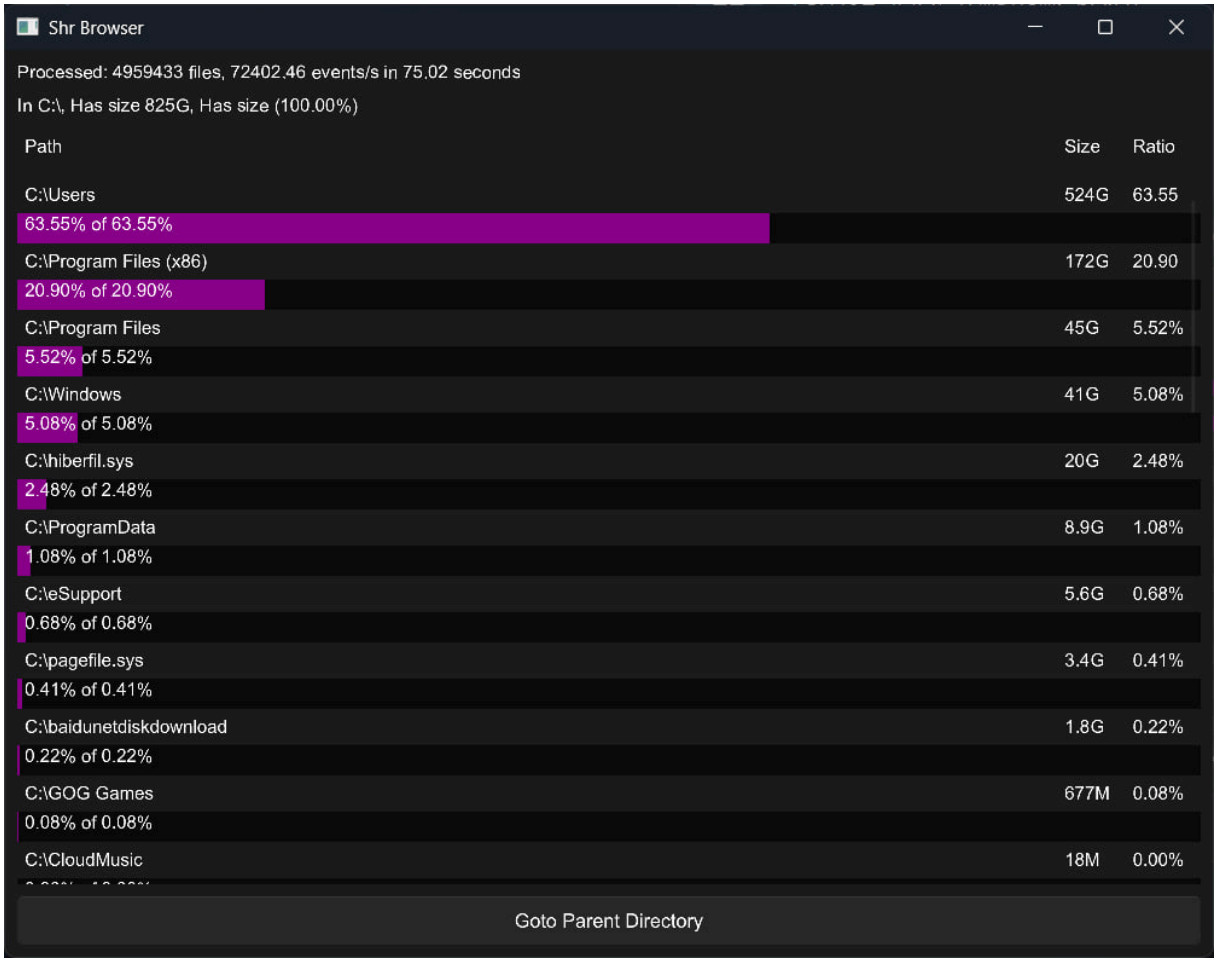

Figure 3: Slint GUI

4.4 性能

在我的机器上，`shr-browser` 比已有的工具 `dust` 快了大约 6.1% (77 秒对比 82 秒)。似乎并没有明显优势。

4.5 可能的改进

`shr` 的 IO 瓶颈在于 `std::fs::read_dir`，可能可以使用比 tokio 更好的后端，例如 compio。

`shr` 的内存瓶颈在于保存了太多完整路径。实测 5000k 个文件（800G）时，内存占用在 1.5GB 左右。这意味着，45G 的内存可以支持 32TB 的完整预览，已经基本满足扫全盘的需求了。未来如果有可能，会有限改进这一点。

4.6 后记

已经使用过几次了，速度很快，UI 也很流畅。但是每次扫盘的时候，都主要是扫描到大量占用的 cargo cache。感觉我只是需要一个好的 cargo 缓存清理工具。

2025-05-21

My Blog Setup without Backend

Configuring a blog via GitHub Pages and Cloudflare.

5.1 Using Cloudflare DNS

First, I change the nameservers of my domain to Cloudflare. I then test that the nameservers are set correctly by running the following command:

```
λ dig example.com +nostats +nocomments +nocmd
;example.com.            IN      A
example.com.     600     IN      SOA     ignat.ns.cloudflare.com.
dns.cloudflare.com. 2373316940 10000 2400 604800 1800`
```

5.2 Verifying the ownership of the domain on GitHub

Under "Setting > Pages", I add the custom domain `example.com` and verify the ownership of the domain by adding a TXT record in Cloudflare DNS settings.

5.3 Configuring the GitHub Pages

In the GitHub repository settings, I change the custom domain to `www.example.com`. Then, goto the Cloudflare DNS settings and add a CNAME record for `www` pointing to `myriad-dreamin.github.io`. Noted that the record shouldn't be **"Proxied" but "DNS Only"**. Test that the nameservers are set correctly by running the following command:

```
λ dig www.example.com +nostats +nocomments +nocmd
;www.example.com.                IN      A
www.example.com. 300     IN      CNAME   myriad-dreamin.github.io.
myriad-dreamin.github.io. 3600  IN      A       x.x.x.x
```

The blog finally should be accessible at `www.example.com`.

2025-05-23

Tinymist 2024 - 语言服务器部分

关于 tinymist，一个 typst 的语言服务器的开发思考。

6.1 LSP 引擎库

在 lsp 引擎上，nvarner 的选择是 tower-lsp，但是这个库事实上并没有尊重 lsp 协议（2024 年的时候，tower-lsp 的情况如此）。lsp 在时序上希望你能保证按顺序处理请求，而 tower-lsp 收到请求上会乱序触发上层 service 的函数。这会导致 language server 状态在启动后一段时间与编辑器状态 desync。tower-lsp 的这一做法也使得允许上层 service 有一些“fancy”的写法，直接导致 typst-lsp 需要完全重写。

rust-analyzer 是怎么做的呢。rust-analyzer 使用了 lsp-server，这是一个底层完全同步的 lsp 引擎。每当有一个请求到来，都会触发一个获得 `state: &mut State` 的 handler。

一位群友为 nix 写的 nil 用了这位群友自研的 async-lsp。其接口要比 lsp-server 整洁和 neat 的多。每当有一个请求到来，async-lsp 也会触发获得全局可变状态的 handler，区别是这个 handler 是 async 的。

我是希望使用 async-lsp 的。在接入的过程中，再一次挖掘到了 rust 的混沌之处。我们做一个表格，lsp-server，async-lsp，tower-lsp 的区别如下：

Name	Order to Accept Requests	Type of Handler
tower-lsp	Out of order	`Fn() -> Fut<Req>`
lsp-server	Sequential	`FnMut() -> Req`
async-lsp	Sequential	`FnMut() -> Fut<Req>`

虽然 async-lsp 看似 async 了，但是别扭之处在于，它的 handler 无法使用 `.await` 语法，取而代之，必须返回一个不引用 state 的 async 闭包。这显然是 rust 的局限性。其次 async-lsp 将 stdio 的读写异步化了，而在 windows 上，这必须要借助 tokio 的 compat IO（correct me if I’m wrong，因为我不是 async 专家）。对于 nix，这并非问题，因为 nix 只会在 unix 上运行（correct me if I’m wrong，因为我不是 nix 用户）。我觉得 nil 的作者不会为 windows 用户买单属于合情合理。

另外，为了方便测试，我有一些希望 async-lsp 改变的东西（目前已经忘了）。出于以上原因，尽管 tinymist 已经完全做好了迁移到 async-lsp 的准备，我还是选择了继续保留对 lsp-server 的包装。总的来说，我希望将来要么 rust 改进对 async 借用的支持，要么有一个更好的引擎框架。

2025-05-27

Typst Syntax

List of Typst Syntax, for rendering tests.

7.1 Raw Blocks

This is an inline raw block `class T`.

This is an inline raw block `class T`.

This is a long inline raw block `class T {}; class T {}; class T {}; class T {}; class T {}; class T {}; class T {}; class T {}; class T {};`.

Js syntax highlight are handled by syntect:

```
class T {};
```

Typst syntax hightlight are specially handled internally:

```
#let f(x) = x;
```

7.2 Equations

Example from academic-homepage-typst: GRPO.

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D}}\left[\min\left(\frac{\pi_{\theta(o_t|q,o_{<t})}}{\pi_{\theta_{\text{old}}}(o_t|q,o_{<t})}\hat{A}_t, \text{clip}\left(\frac{\pi_{\theta(o_t|q,o_{<t})}}{\pi_{\theta_{\text{old}}}(o_t|q,o_{<t})}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)\right]$$

- $r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q,o_{i,<t})}$ is the importance sampling ratio for the i-th response at time step t.
- $\hat{A}_{i,t}$ is the advantage for the i-th response at time step t.

7.3 Images

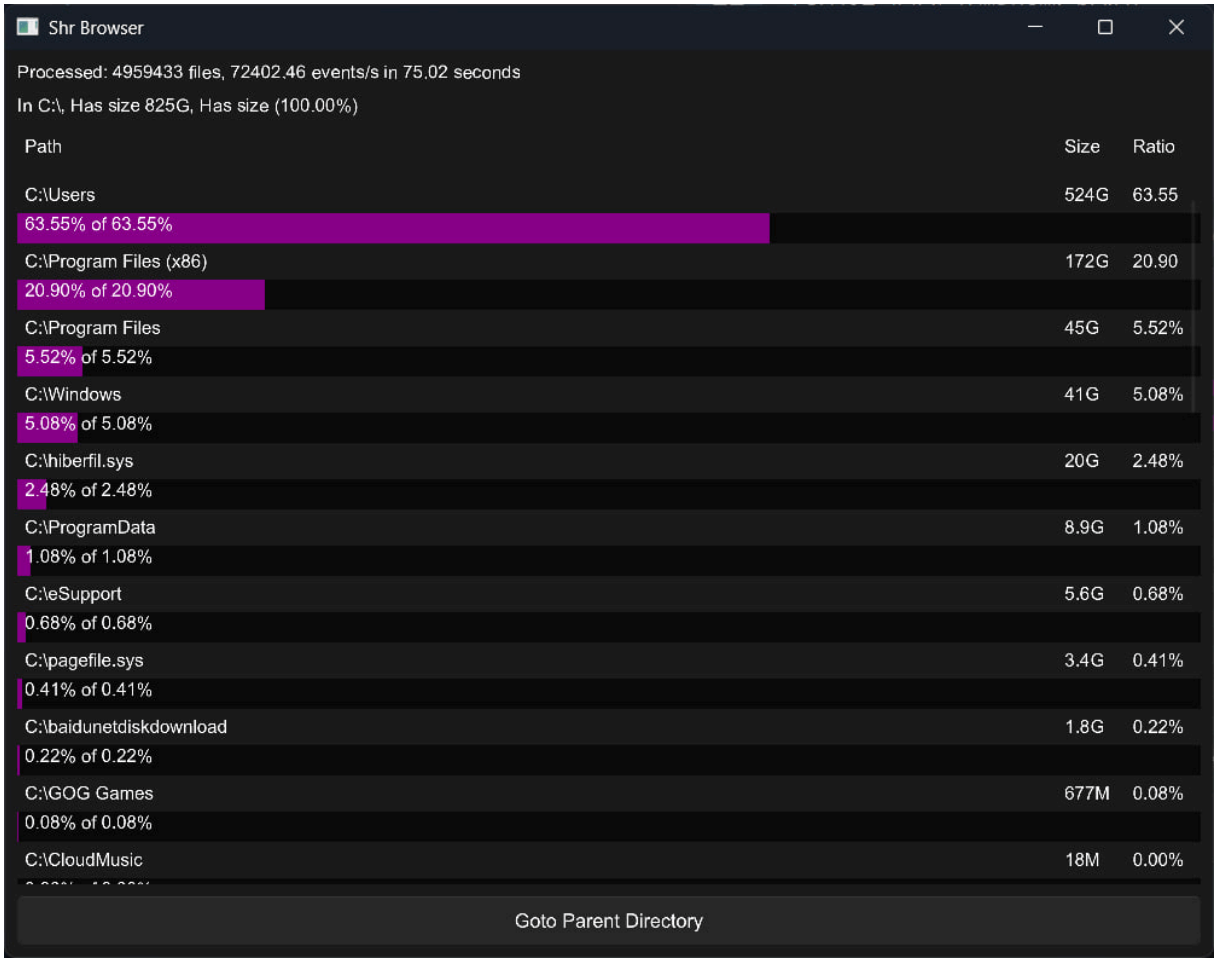

Figure 4: Slint GUI (Absolute Path)

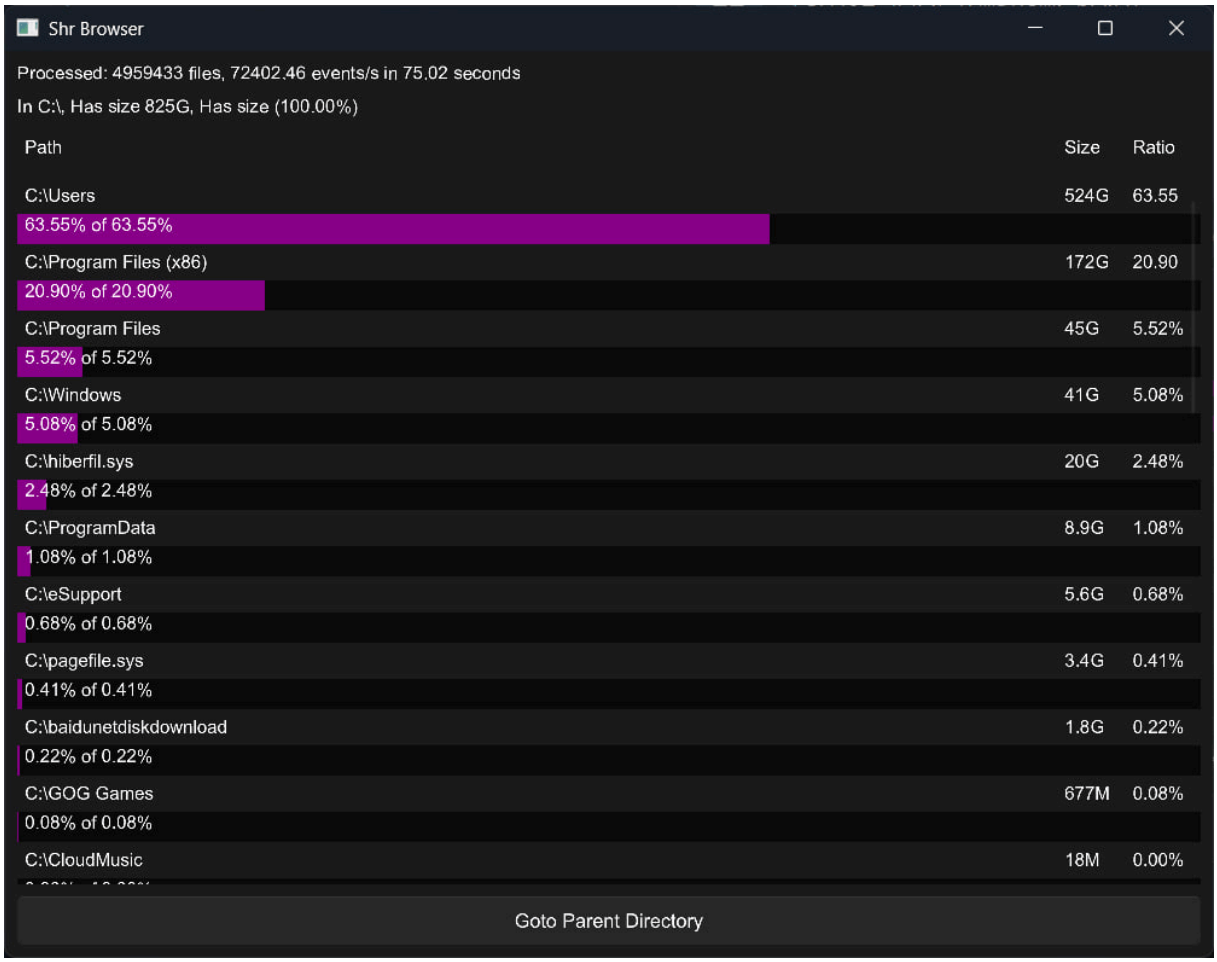

Figure 5: Slint GUI (Relative Path)