Fast multithreaded PNG blending

A case for Rust

Platform

ENEEC 2023, Aveiro

PlatformE

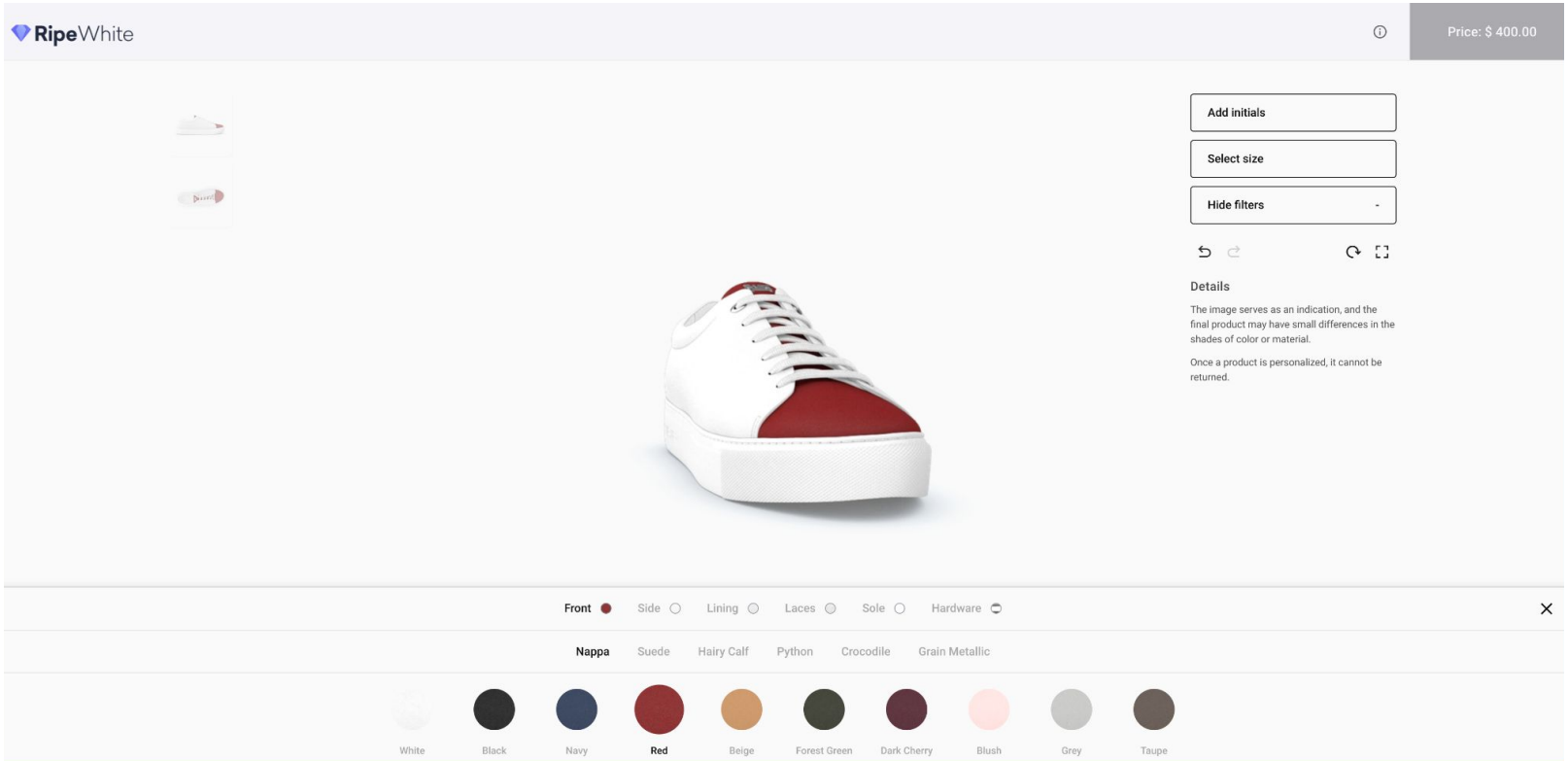
- On a mission to drastically reduce overproduction.
- Enable a new way for people to perceive, customize, personalize and buy products across several channels.
- Define standards and processes across the supply chain in order to fully enable Made-To-Order (MTO) production.


Distribution
Production
Managing
Selling
Digital creation

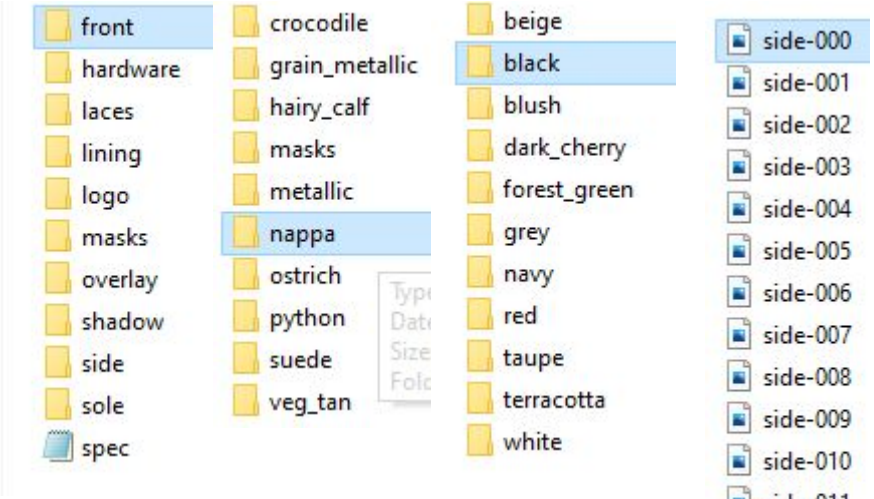

Distribution
Production
Managing
Selling
Digital creation

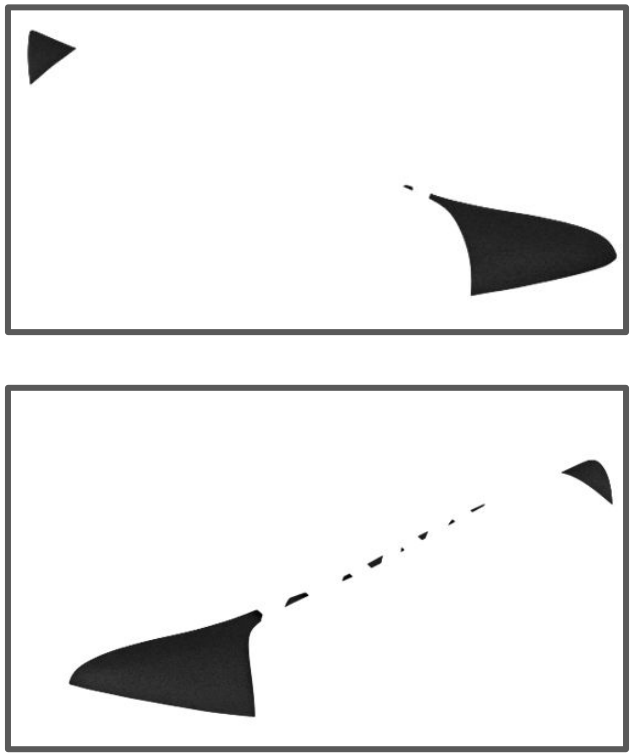
Customization

- PlatformE’s configurator allows for easy customization of a product’s parts, changing materials and colors.
- https://white.platforme.com/?brand=swear&model=vyner



- This behavior is sourced by our 3D builds which are directories of base images and metadata.
- A 3D build consists of metadata and a PNG image of each part, material and color for each frame.





- To render a specific product configuration for a given frame, we use a strategy called Pre-Render & Compose (PRC).
- PRC composition amounts to “gluing” these PNG files into one, for a given frame.
- This PRC Composition is done by multiple services who handle the metadata of the 3DB, but ultimately the blending of pixel data is done by PConvert (https://github.com/hivesolutions/pconvert)

PConvert

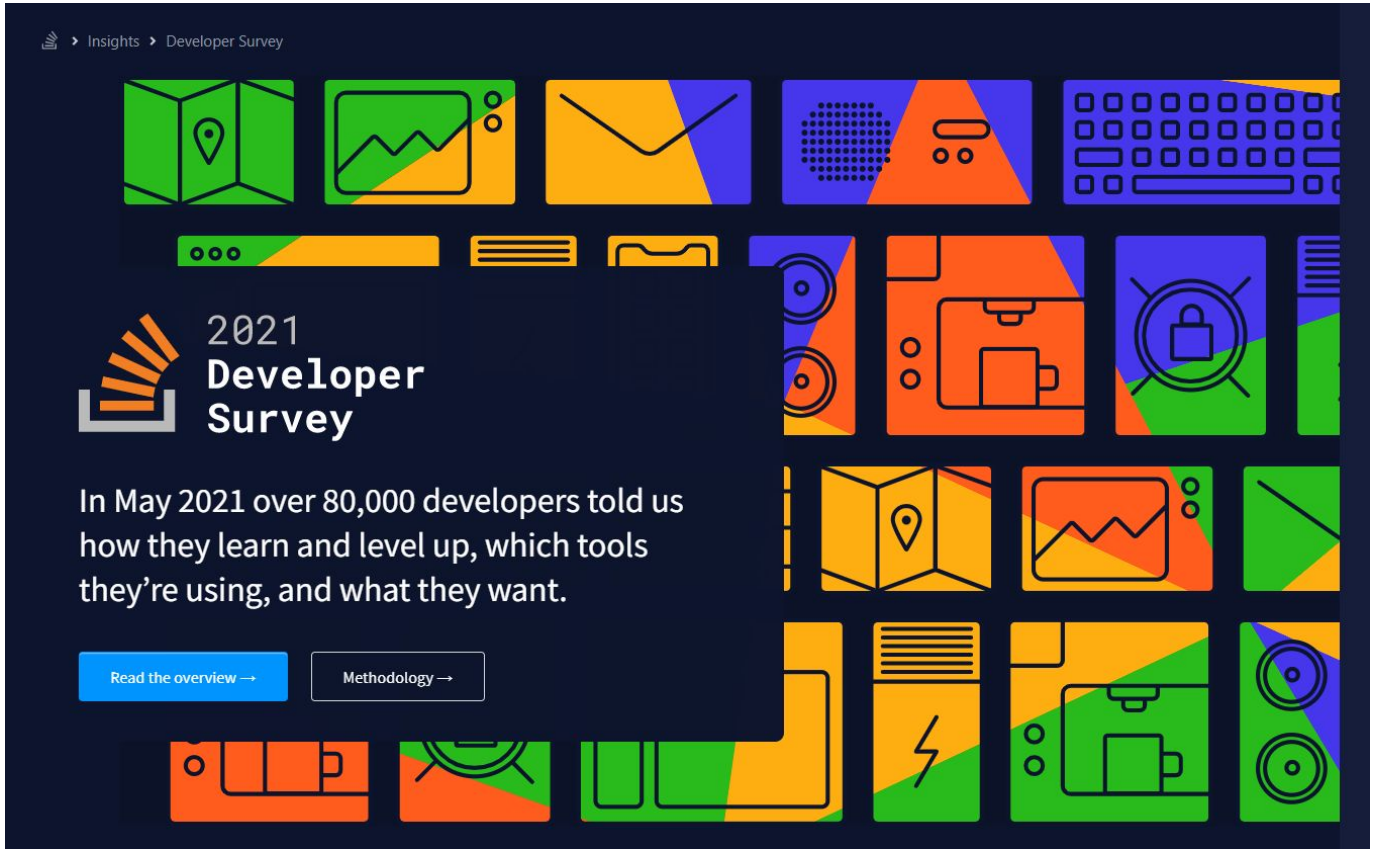
- https://github.com/hivesolutions/pconvert
- PConvert is a C program that takes PNG files and blends them into one according to the algorithm chosen.
- PConvert can be used a terminal application or as a python package (installable via PIP) which is how PlatformE's microservices actually use it.

Project 008

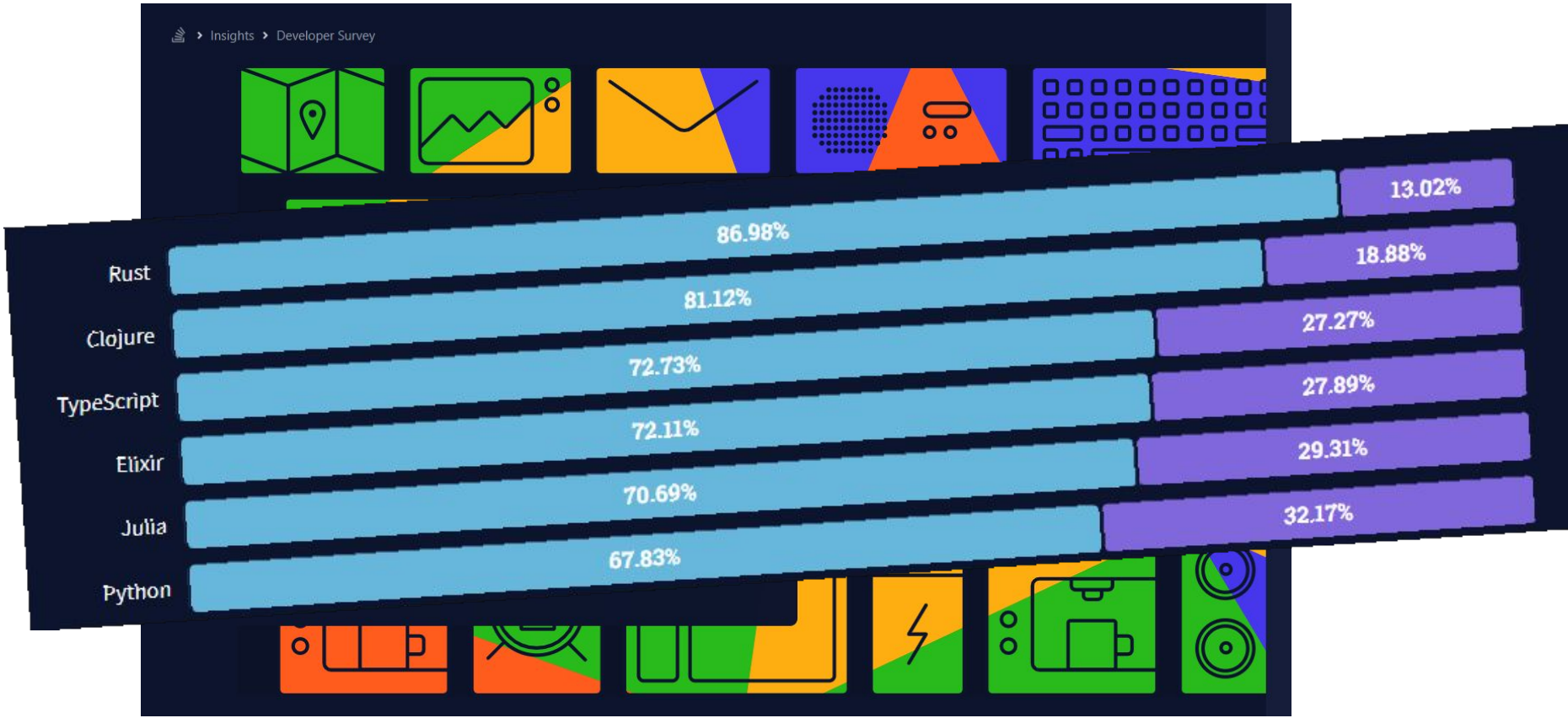
- https://tech.platforme.com/projects/008
- Project 008 consisted of a rewrite of PConvert using Rust.
- Main goals:
  - Retrocompatible
  - Expose an equivalent Python API
  - Expose an equivalent WebAssembly (WASM) API
  - Sub image parallelization performance improvements
  - Measure the impact of SIMD in the blending performance
  - Benchmark the solution

Rust

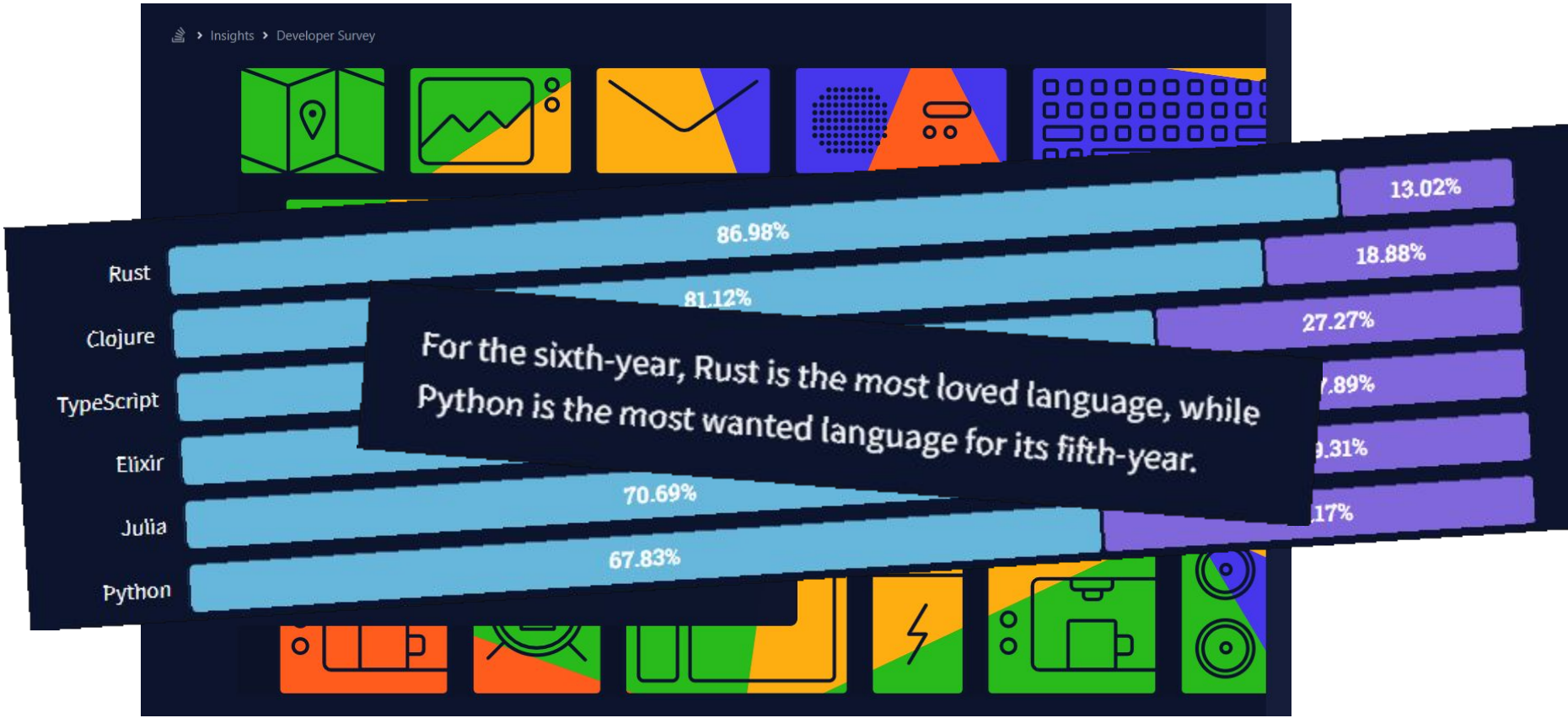
What is Rust


Insights › Developer Survey
2021
Developer
Survey
In May 2021 over 80,000 developers told us how they learn and level up, which tools they're using, and what they want.
Read the overview →
Methodology →

What is Rust


Insights › Developer Survey
Rust
86.98%
13.02%
Clojure
81.12%
18.88%
TypeScript
72.73%
27.27%
Elixir
72.11%
27.89%
Julia
70.69%
29.31%
Python
67.83%
32.17%

What is Rust


Insights › Developer Survey
Rust
Clojure
TypeScript
Elixir
Julia
Python
86.98%
81.12%
70.69%
67.83%
13.02%
18.88%
27.27%
For the sixth-year, Rust is the most loved language, while Python is the most wanted language for its fifth-year.

What is Rust

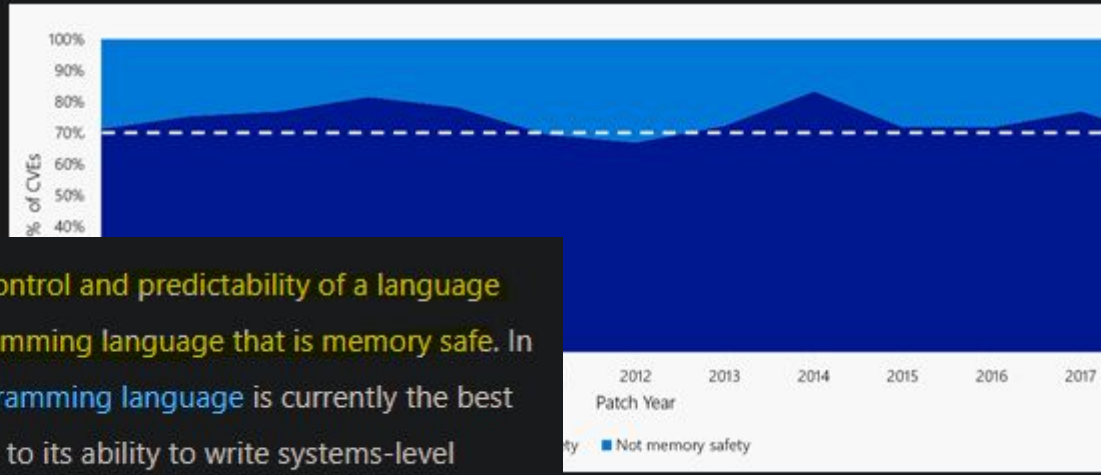
- A systems programming language
- From the book (https://doc.rust-lang.org/book/):
  - “helps you write faster, more reliable software”
  - “high-level ergonomics and low-level control”
  - “option to control low-level details (such as memory usage) without all the hassle traditionally associated with such control”
- The Rust programming language is fundamentally about empowering all programmers to write performant, memory and thread safe code while having fun doing it.

Why Rust

safer systems programming

efense / By MSRC Team / July 18, 2019 / Memory Safety,
Languages, Secure Development

As was pointed out in our previous post, the root cause of approximately 70% of secu
vulnerabilities that Microsoft fixes and assigns a CVE (Common Vulnerabilities and Exp
are due to memory safety issues. This is despite mitigations including intense code re
training, static analysis, and more.



If there are legitimate reasons for needing the speed, control and predictability of a language like C++, see if you can move to a systems-level programming language that is memory safe. In our next post, we'll explore why we think the Rust programming language is currently the best choice for the industry to adopt whenever possible due to its ability to write systems-level programs in a memory-safe way.

ssigns a CVE each year continue to be me
safety issues

Chrome: 70% of all security bugs are memory safety issues

Google software engineers are looking into ways of eliminating memory management-related bugs from Chrome.

Roughly 70% of all serious security bugs in the Chrome codebase are memory management and safety bugs, Google engineers said this week.

Half of the 70% are use-after-free vulnerabilities, a type of security issue that arises from incorrect management of memory pointers (addresses), leaving doors open for attackers to attack Chrome's inner components.

Image: Google

Why Rust for PConvert

- Rust guarantees thread-safety by design which is useful because one of our goals was writing a multithreaded PConvert
- Rust code is as performant as C/C++ code so performance is guaranteed.
- Rust provides a better package manager and build system (Cargo) than the C ecosystem
- Rust provides a more modern and ergonomic syntax

How does Rust ensure all of that?

- This is not a Rust talk but…
  - Rust's safety and performance guarantees boil down to the newly created concept of ownership.
  - A piece of memory has an owner but can be borrowed or given (transfer of ownership).
  - There are a set of rules for borrowing memory which ensure the discussed safety properties.

PConvert Rust

- https://github.com/ripe-tech/pconvert-rust
- PConvert Rust: PNG blending tool featuring:
  - A retrocompatible API with PConvert (C)
  - Multi-threaded PNG file reading and writing
  - Available as a Rust crate
  - Available as a Python package
  - Available as a WASM module
  - Available as a terminal application
  - Better error handling

A retrocompatible API with PConvert ©

- The terminal application takes in the same commands and arguments
- The Python package exposes a superset of the previous API methods and arguments
- TODO por alguns snippets de codigo

Available as a Rust crate

- https://crates.io/crates/pconvert-rust

Available as a Python package

- https://pypi.org/project/pconvert-rust/0.1.2/

Available as a WASM module

- https://github.com/ripe-tech/pconvert-rust/tree/master/examples/wasm

Available as a terminal application

```
~/Documents/pconvert-rust master !1 ❯ ./target/release/pconvert-rust benchmark assets/demo --parallel
Algorithm          Compression        Filter             Times
-------------------------------------------------------------------------------------------------------
multiplicative     Default            NoFilter           44ms (blend 8ms, read 10ms, write 26ms)
multiplicative     Fast               NoFilter           28ms (blend 13ms, read 10ms, write 5ms)
multiplicative     Best               NoFilter           44ms (blend 8ms, read 11ms, write 25ms)

source_over        Default            NoFilter           31ms (blend 9ms, read 11ms, write 11ms)
source_over        Fast               NoFilter           21ms (blend 8ms, read 8ms, write 5ms)
source_over        Best               NoFilter           45ms (blend 8ms, read 10ms, write 27ms)

alpha              Default            NoFilter           49ms (blend 16ms, read 11ms, write 22ms)
alpha              Fast               NoFilter           23ms (blend 12ms, read 6ms, write 5ms)
alpha              Best               NoFilter           44ms (blend 8ms, read 11ms, write 25ms)

disjoint_over      Default            NoFilter           65ms (blend 18ms, read 24ms, write 23ms)
disjoint_over      Fast               NoFilter           35ms (blend 14ms, read 16ms, write 5ms)
disjoint_over      Best               NoFilter           57ms (blend 12ms, read 16ms, write 29ms)

disjoint_under     Default            NoFilter           54ms (blend 20ms, read 16ms, write 18ms)
disjoint_under     Fast               NoFilter           39ms (blend 12ms, read 22ms, write 5ms)
disjoint_under     Best               NoFilter           76ms (blend 12ms, read 11ms, write 53ms)


Total time: 655ms (blend 178ms, read 193ms, write 284ms)
```

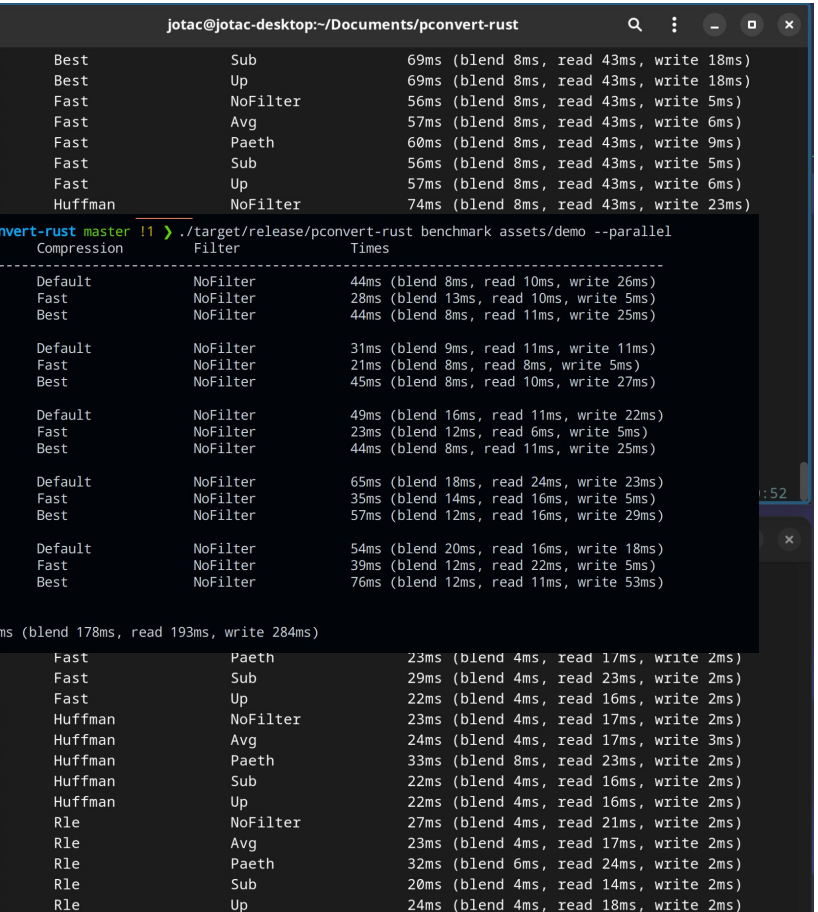
Error handling

- C version just seg faults
- Rust version tells you a descriptive error
- https://github.com/ripe-tech/ripe-core/issues/4747

Multi-threading: reading and writing PNG files

- Falar da thread pool, talvez por snippets de rust code
- Due to the parallelization of PNG file reading, the of reading increased linearly with the number of slices read. On average our models have 5 slices, which translates to a 5x speed up.
- The parallelization of writing the resulting PNG was made using the crate <INSERT NAME> and it roughly halved the time spent.

- Benchmarks
  - TABLE HERE INSTEAD OF PRINTS




```
~/Documents/pconvert master ❯ pconvert benchmark assets/demo/

multiplicative Z_NO_COMPRESSION CPU        61.11ms (blend 14.94ms, read 43.48ms, write
multiplicative Z_BEST_SPEED CPU            41.53ms (blend 9.77ms, read 24.34ms, write
multiplicative Z_BEST_COMPRESSION CPU      78.23ms (blend 9.78ms, read 24.52ms, write

source_over Z_NO_COMPRESSION CPU           35.86ms (blend 8.40ms, read 24.56ms, write
source_over Z_BEST_SPEED CPU               40.24ms (blend 8.38ms, read 24.51ms, write
source_over Z_BEST_COMPRESSION CPU         76.52ms (blend 8.39ms, read 24.51ms, write

alpha Z_NO_COMPRESSION CPU                 36.10ms (blend 8.74ms, read 24.51ms, write
alpha Z_BEST_SPEED CPU                     40.67ms (blend 8.77ms, read 24.53ms, write
alpha Z_BEST_COMPRESSION CPU               76.22ms (blend 8.76ms, read 24.51ms, write

disjoint_over Z_NO_COMPRESSION CPU         59.66ms (blend 16.24ms, read 40.55ms, write
disjoint_over Z_BEST_SPEED CPU             64.17ms (blend 16.25ms, read 40.57ms, write
disjoint_over Z_BEST_COMPRESSION CPU       109.04ms (blend 16.26ms, read 40.54ms, writ

disjoint_under Z_NO_COMPRESSION CPU        56.60ms (blend 13.16ms, read 40.58ms, write
disjoint_under Z_BEST_SPEED CPU            61.17ms (blend 13.17ms, read 40.62ms, write
disjoint_under Z_BEST_COMPRESSION CPU      97.04ms (blend 13.14ms, read 40.55ms, write

total_execution                            934.17ms (blend 174.15ms, read 482.88ms, wr
```

- SIMD Support
  - SIMD stands for Single Instruction, Multiple Data.
  - Think of it as vectorized operations.
  - Pixel operations map very well into SIMD.
  - SIMD support was tested but unfortunately the Rust ecosystem in 2020 was not very mature regarding SIMD and different computer architectures.

- Future
  - Explore SIMD in 2023
  - Parallelize PNG blending for 4+ slices

Q&A

(slides available at)

Thank You!

Project:

- github.com/hivesolutions/pconvert
- github.com/ripe-tech/pconvert-rust

Reach us at:

- github.com/ripe-tech
- github.com/joao-conde
- joaodiasconde@gmail.com

Platforme

For more information contact us

info@platforme.com