Interactive 3D Web Applications with Rust + WebAssembly

Francesco Giordana
fra.giordana@gmail.com

The Goal

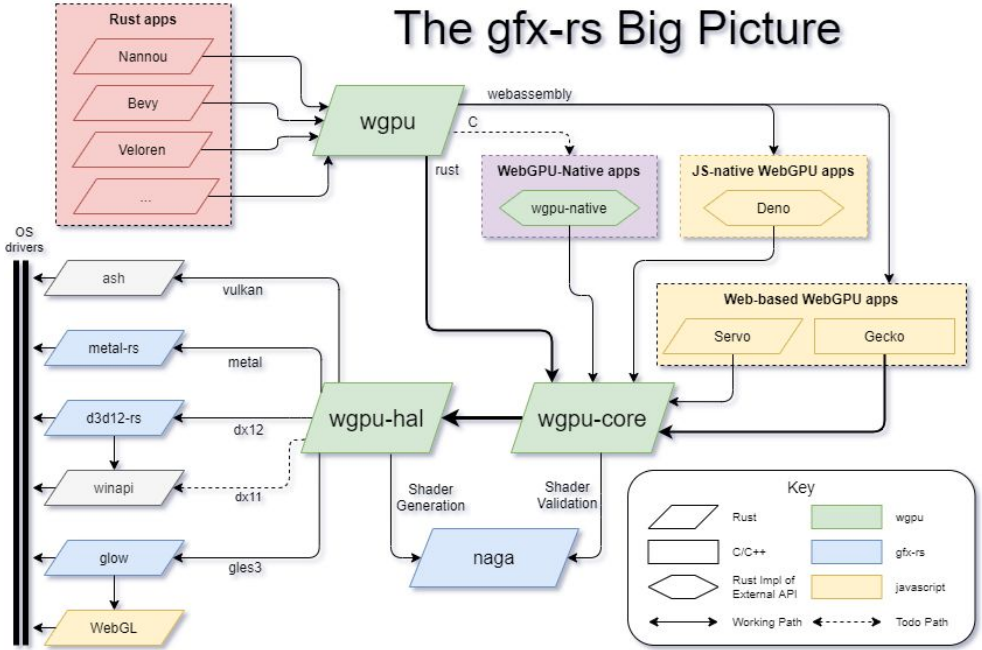
Separate frontend development from rendering backend and complex application logic.

- Frontend should be built with familiar JS frameworks (React, Vue.js, ...)
- Rendering and interactivity in browser needs performant code and complex logic

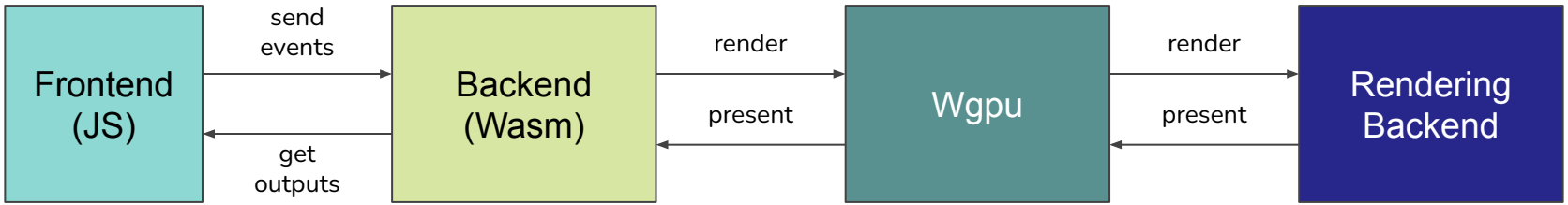
Why Rust

- Extremely performant (close to C)
- Very safe even for complex multi-threaded and concurrent code
- Can easily target WebAssembly [https://rustwasm.github.io/book/]
- Very good ecosystem of libraries (crates)
- Well structured code
- More familiar to backend/rendering engineers than JS

Rendering with Wgpu



Skeleton of a simple 3D app



1. Project Setup

Project template

git clone -b 1-project-setup https://github.com/fgiordana/glc-2022.git

Components:

- glc-ui: skeleton of simple React app, using Webpack5 as bundler
- glc-rs: skeleton of simple rust library exposing a js interface for a wasm module
- Webpack and npm configuration for loading the glc-rs wasm module
- Wasm-pack is a utility for compiling RUst to wasm and packaging up the module

2. Render to Canvas

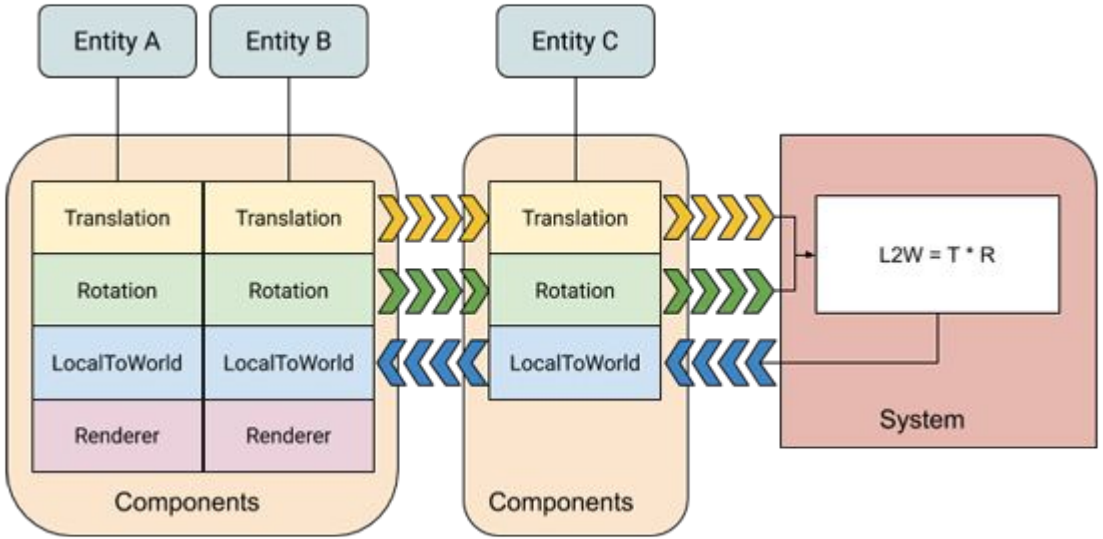
Bevy engine

[https://bevyengine.org/]

Entity

Component

System



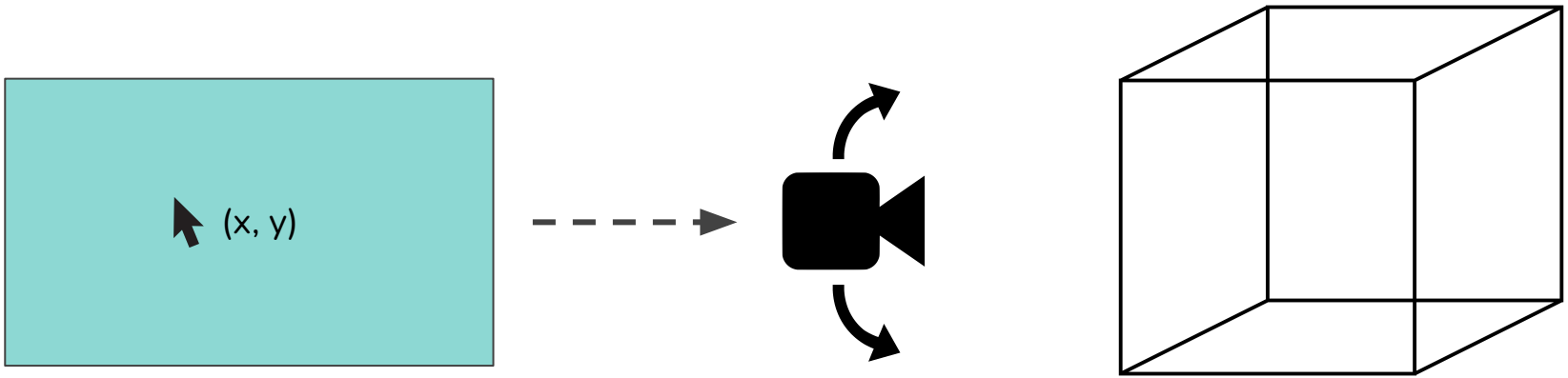
Rendering to a Html Canvas

- Wasm-bindgen generates JS bindings for Wasm [https://rustwasm.github.io/wasm-bindgen/introduction.html]

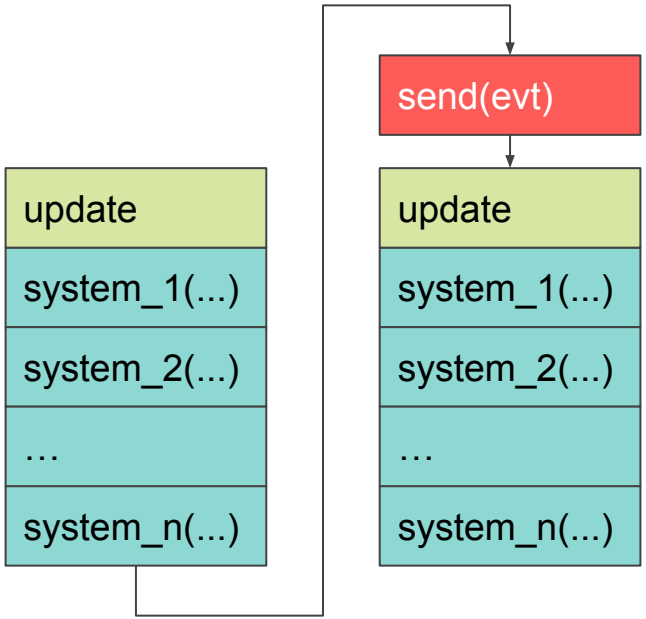
- Web-sys crate lets us interact with the DOM from Rust [https://rustwasm.github.io/wasm-bindgen/web-sys/index.html]

3. Event Loop

Sending Events from the UI

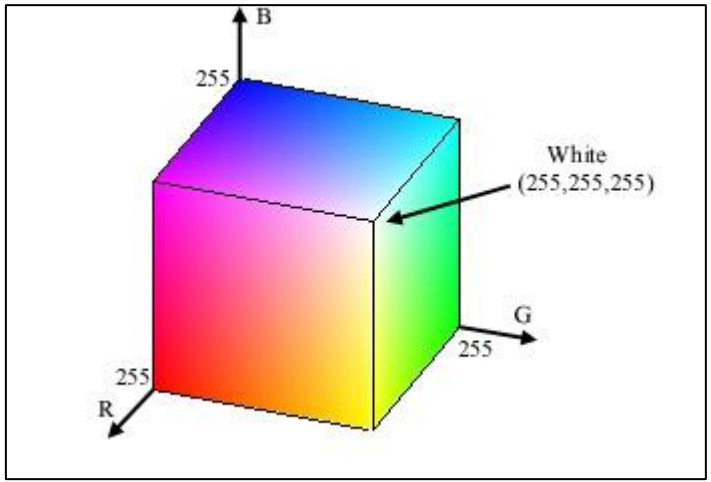


Processing Events



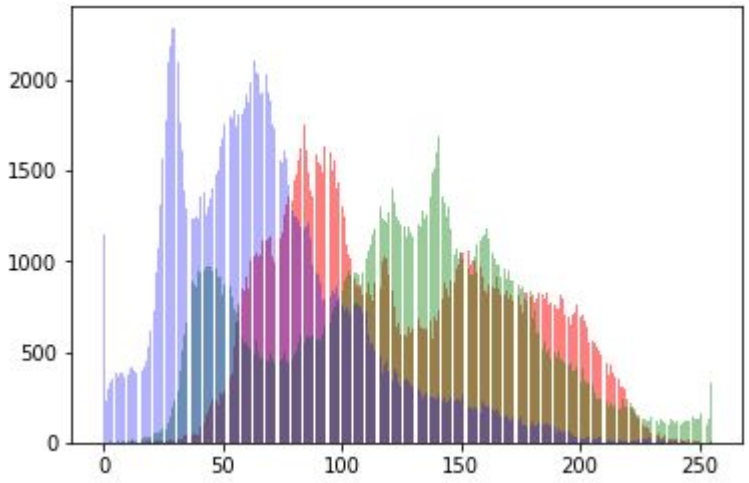
4. Color Cube

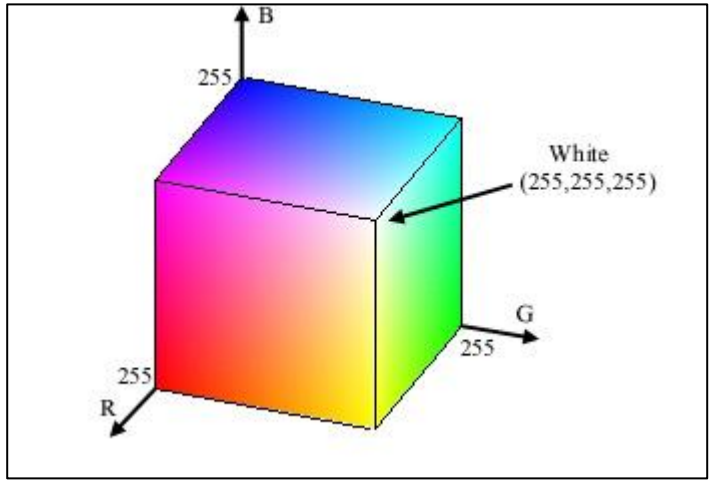
Plotting a Color Cube



5. Input Image

Plotting an image's pixel histogram in 3D





6. Image Transformation

Rotating the Color Space