Implementation of rendering system in Rust

Eduard Lavuš

Faculty of Electrical Engineering

Czech Technical University in Prague

2020-06-01

Motivation and aim

- Motivation
  - Abandoned open-source projects with similar aim
  - Rust safety features exploration
- Aim
  - Designing and implementing flexible and transparent high-level Vulkan API wrapper
  - Comparing design to previous attempts and measuring development and performance cost
  - Creating the “core” for future work to build upon

Structure of the thesis

- Introduction to Vulkan
- Overview of existing projects in both Rust and C++
- Design principles and Rust features
- Implementation details and difficulties
- Evaluation
    - Developer experience
    - Performance
    - Safety

Rust and Vulkan

- Rust
  - Fast
  - Flexible
  - Safe
  - Developer friendly
- Vulkan
  - Fast
  - Flexible
  - Unsafe
  - Developer unfriendly

Design and implementation

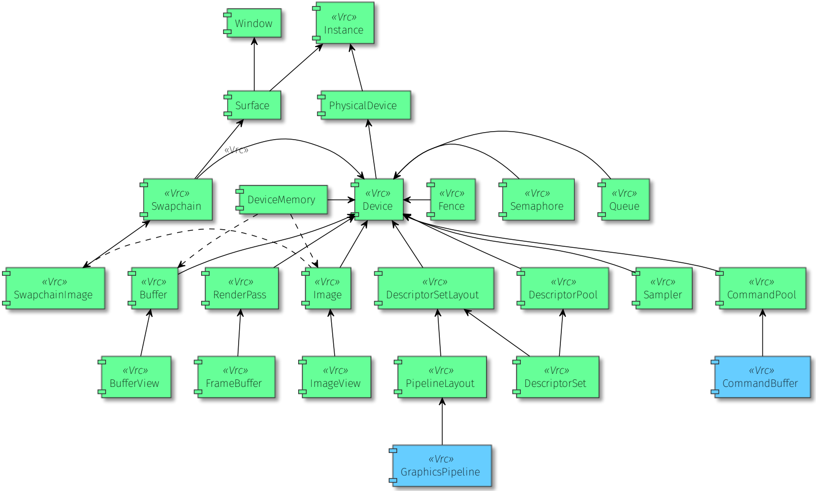

Figure 1: Object Dependency Graph of Vulkayes

Design and implementation

Project name: Vulkayes

- Design
  - Transparent
  - Minimal overhead
  - Statical safety
- Implementation
  - Cargo features
  - Vrc, Deref, generics
  - Flexibility

Results and evaluation

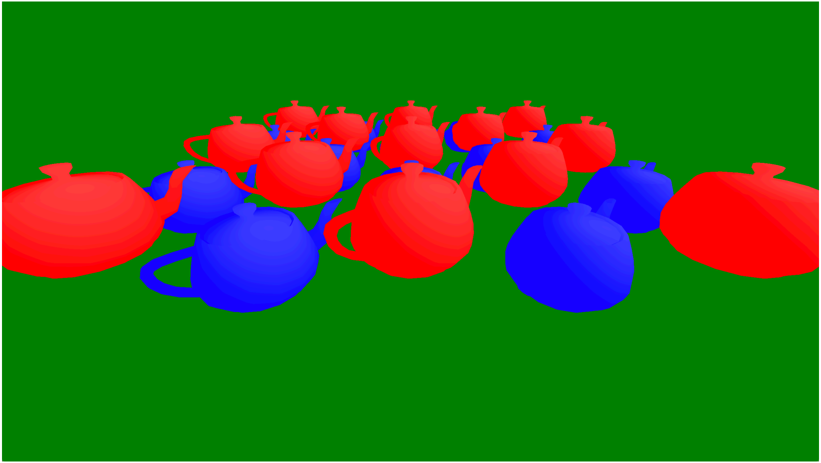

Figure 2: Benchmarking application output

Results and evaluation

- Developer experience
  - In selected code snipptes, Vulkayes code is three times shorter than equivalent ash code
  - The benchmarking program code in Vulkayes is 33% shorter than in ash
- Performance
  - Vulkayes was evaluated against ash, the bindings to Vulkan API that are used, as a baseline
  - `vy_ST` represents single-threaded version of Vulkayes
  - `vy_MT` represents Vulkayes with multi-threading enabled
- Safety
  - All but two Vulkan implicit validations were solved: 317 statically and 28 dynamically
  - 21% (120) of explicit validations were solved statically as a byproduct of good API design

Results and evaluation

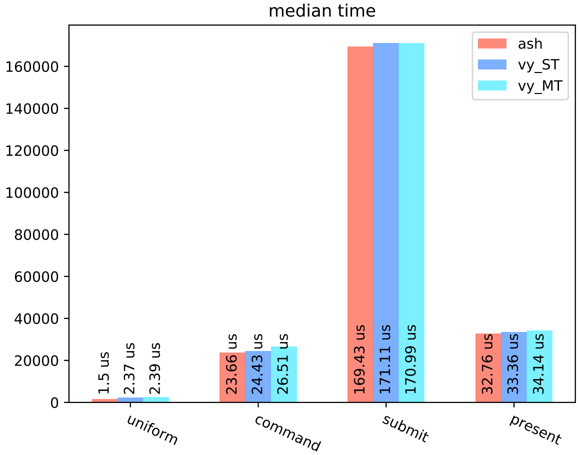

Figure 3: *Average median time (n = 99000): macOS 10.15.3 (19D76), Quad-Core Intel Core i5, Intel Iris Plus Graphics 655, Vulkan 1.2.135*

Results and evaluation

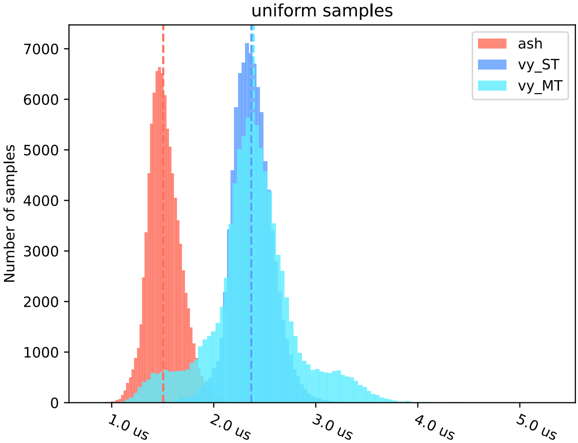

Figure 4: *Histogram of uniform stage of the benchmarks (n = 99000). It is clear that ash is faster than both single- and multi-threaded Vulkayes. However, the overhead is constant.*

Results and evaluation

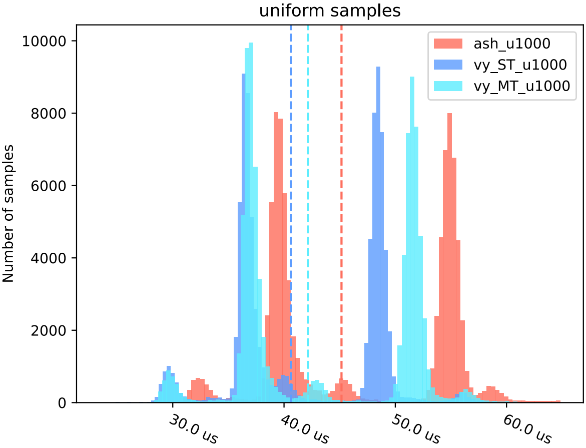

Figure 5: *Histogram of uniform stage of the benchmarks (n = 99000) with 1000 writes instead of 1. The overhead displayed in previous bench is overshadowed by the gains of proper writing strategy.*

Conclusion

- Code written using Vulkayes is shorter but still flexible
- Vulkayes performs as fast as ash
- Safety is greatly increased thanks to both Rust and API design
- Vulkayes is a good step towards a more complex modular solution