

vkCompViz: Universal C++ Library for GPU-Based Experiments

Tomas Chlubna ^{1*}

¹ Faculty of Information Technology Brno University of Technology, Czech Republic * These authors contributed equally.

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

GPUs are gaining popularity due to their massive computational parallelism (Jia et al., 2021) and usage in interactive graphics (Nguyen, 2007) or machine learning applications (Mittal & Vaishay, 2019). Prototyping GPU-executed experiments is often time-consuming due to the complexity of GPU-related APIs. Multiplatform and multi-vendor support is also not guaranteed with all existing APIs. The vkCompViz C++ library offers a simple way to write a GPGPU (Hu et al., 2016) (general-purpose computing on graphics processing units) program. Only the paths to files with GPU code (shader/kernel), paths to input images, or a buffer of arbitrary input data need to be provided by the host application. The library is capable of running a sequence of compute shaders, processing the input data, and storing the result or presenting it in a window. Memory usage and computational time can also be automatically measured.

Statement of need

Necessary data allocations and transfers, and the execution of shaders, require a special API that communicates with the GPU drivers (Henriksen, 2024; Plebański et al., 2025). However, using these APIs presents various problems. Vendor-specific GPGPU API CUDA by NVIDIA and HIP by AMD quickly provide access to new GPU features but can be used only with the given vendor's GPUs. OpenCL is a multiplatform and vendor-free API that contains a lot of features, but, in practice, GPU drivers usually do not implement its entire specification. The APIs referenced earlier are tailored only for GPGPU computations and lack direct access to GPU rendering features. In contrast, subsequent APIs are capable of using rendering pipelines and GPGPU functions. DirectX does not depend on the vendor, but is developed by Microsoft and can be used only on Windows. Similarly, Metal works only on Apple devices. OpenGL is multiplatform and vendor-free, but it does not support new features and its development is discontinued. Vulkan is multi-platform, vendor-free, supports new features quickly, and offers low-level optimization settings. The main disadvantage of Vulkan is its verbosity, which makes it difficult to use for quick experiments, as a substantial portion of the code is necessary even for basic functionality. All the APIs also require the host application to allocate data in GPU memory, load and decode resources like images, transfer the data, create a window, etc.

The proposed vkCompViz library offers a high-level API that significantly simplifies GPU programming. The library only requires the paths to the code files to be executed on GPU, paths to the input images, a buffer with arbitrary input data, and a set of parameters. The library then allocates the necessary memory, creates the GPU-related objects, transfers the data on the GPU, runs the computation (or rendering) pipeline, and returns or stores the results. The library can visualize the results in a window, where the parameters can be interactively adjusted at run-time, or can be run in headless mode on machines with no window systems.

The memory usage and time performance of the data transfer and shader execution reporting mechanism is also implemented. With this library, researchers do not need to learn about complex GPU-related APIs and can quickly conduct scientific experiments on GPU. This tackles a frequent issue in the scientific domain, where bringing an experimental concept to life usually requires a considerable amount of time owing to the need to acquire novel technological skills and address numerous small, implementation-related challenges.

Compared to existing frameworks used in science that simplify the work with Vulkan, such as vk-bootstrap (Muñoz Lopez & Winston, 2025), or Auto-Vk-Toolkit (Unterguggenberger et al., 2023), the proposed library does not require an in-depth work with the GPU-related structures. The library is specially designed for quick experimental prototyping. Other frameworks often focus on specific tasks, such as Datoviz (Rossant & Rougier, 2021) for scientific data visualization, the framework for remote rendering of large data (Lavrič et al., 2018), or VComputeBench for benchmarking purposes (Mammeri & Juurlink, 2018). The proposed library aims to be a general GPGPU scientific framework. The goal is to reduce the time required for GPU programming, particularly since numerous scientific experiments may not produce significant results, and it is inefficient to spend excessive time learning about existing APIs.

Architecture

The library uses Vulkan (Bailey, 2019), which ensures support for modern operating systems and GPUs. Vulkan is expected to be further supported in the future and also quickly adopts novel GPU features which can be possibly used in the library. The input shaders are expected to be written in the Slang language, which is a universal modern language designed for GPU shaders. The input images are loaded with the FFmpeg library, which supports a wide range of multimedia formats. The window is created by the GLFW multi-platform library. The library workflow and architecture are described in Figure 1.

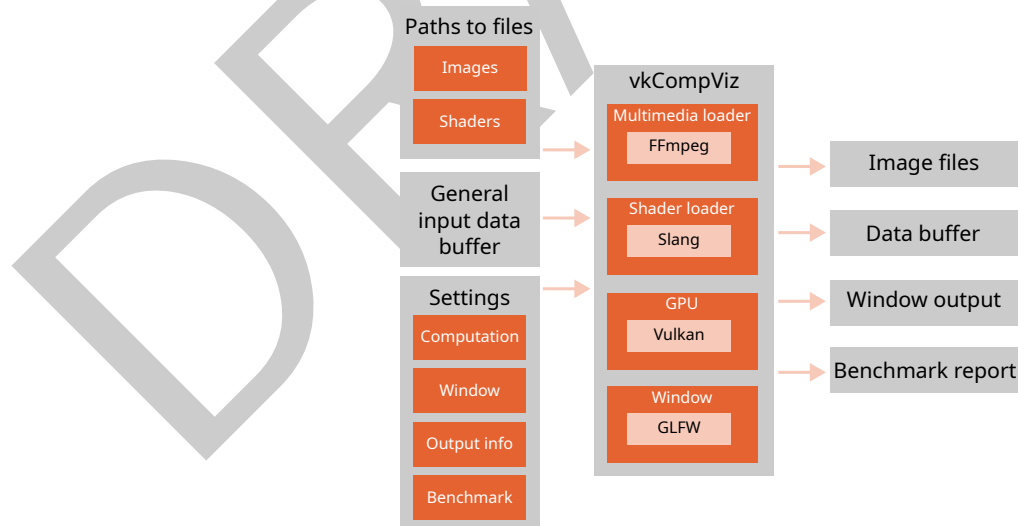


Figure 1: The figure describes the architecture of vkCompViz library.

Usage

The project uses CMake, C++26 standard, and C++20 modules. The usage of the library is demonstrated and documented in three basic example subprojects. The Simple Blending example demonstrates operations with images in which two images can be blended together

71 with a given factor. The 3D Viewer also shows how the rendering pipeline can be customized
72 to render an input 3D model file. The Parallel Reduction example shows how the library can
73 be used to accelerate a summation of a big array of numbers. This example demonstrates a
74 simple use case of an experimental evaluation of the GPU accelerated algorithm (Jradi et al.,
75 2020), compared to its CPU variant. The experiment was carried out on a machine equipped
76 with a NVIDIA GeForce RTX 3060 Ti and Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz CPU,
77 running Arch Linux. The output of several runs of this example is shown in Fig. Figure 2. Fig.
78 Figure 3 shows the detailed runs where the GPU started to be faster than CPU. The results
79 show that computational time is shorter with massively parallel GPU architecture. However,
80 the data transfer delay increases the total time and shows that using GPU for this use case
81 is viable only when the data is already generated on GPU or later used there or when the
82 data size reaches certain amount. This fact is in alignment with previously published findings
83 (Dinkelbach et al., 2012).

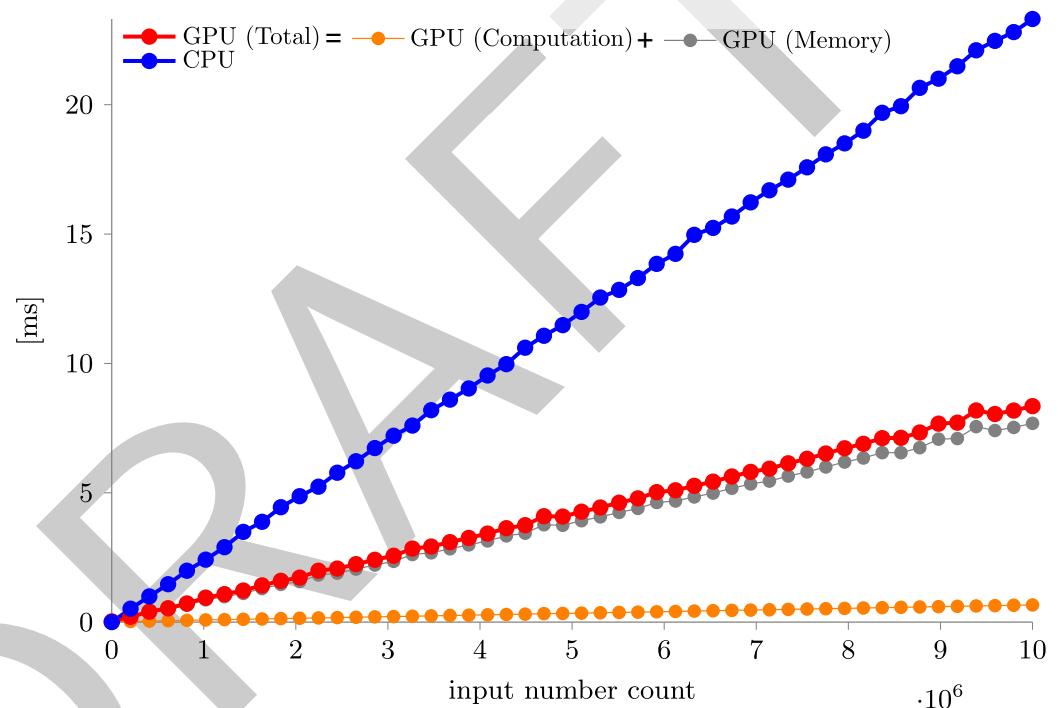


Figure 2: Comparison of GPU and CPU array summation is shown in the chart.

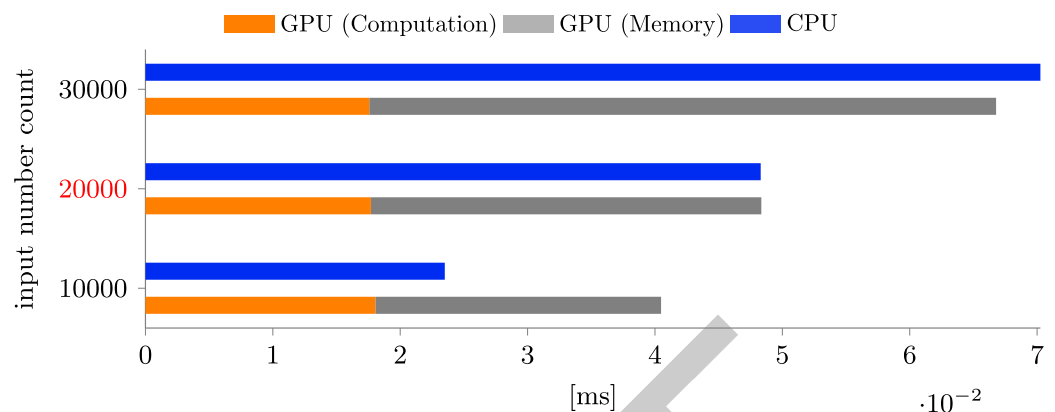


Figure 3: The turning point where GPU starts to be faster than CPU for array summation is shown with two neighboring measurements.

References

- 84
- 85 Bailey, M. (2019). Introduction to the vulkan® computer graphics API. *SIGGRAPH Asia 2019*
- 86 *Courses*. <https://doi.org/10.1145/3355047.3359405>
- 87 Dinkelbach, H., Vitay, J., Beuth, F., & Hamker, F. (2012). Comparison of GPU- and CPU-
- 88 implementations of mean-firing rate neural networks on parallel hardware. *Network (Bristol,*
- 89 *England)*, 23. <https://doi.org/10.3109/0954898X.2012.739292>
- 90 Henriksen, T. (2024). A comparison of OpenCL, CUDA, and HIP as compilation targets
- 91 for a functional array language. *Proceedings of the 1st ACM SIGPLAN International*
- 92 *Workshop on Functional Programming for Productivity and Performance*, 1–9. <https://doi.org/10.1145/3677997.3678226>
- 93
- 94 Hu, L., Che, X., & Zheng, S.-Q. (2016). A closer look at GPGPU. 48(4). <https://doi.org/10.1145/2873053>
- 95
- 96 Jia, S., Tian, Z., Ma, Y., Sun, C., Zhang, Y., & Zhang, Y. (2021). A survey of GPGPU parallel
- 97 processing architecture performance optimization. *2021 IEEE/ACIS 20th International Fall*
- 98 *Conference on Computer and Information Science (ICIS Fall)*, 75–82. <https://doi.org/10.1109/ICISFall51598.2021.9627400>
- 99
- 100 Jradi, W. A. R., Nascimento, H. A. D. do, & Martins, W. S. (2020). A GPU-based parallel
- 101 reduction implementation. In C. Bianchini, C. Osthoff, P. Souza, & R. Ferreira (Eds.),
- 102 *High performance computing systems* (pp. 168–182). Springer International Publishing.
- 103 ISBN: 978-3-030-41050-6
- 104 Lavrič, P., Bohak, C., & Marolt, M. (2018). Vulkan abstraction layer for large data remote
- 105 rendering system. In L. T. De Paolis & P. Bourdot (Eds.), *Augmented reality, virtual reality,*
- 106 *and computer graphics* (pp. 480–488). Springer International Publishing. ISBN: 978-3-
- 107 319-95270-3
- 108 Mammeri, N., & Juurlink, B. (2018). VComputeBench: A vulkan benchmark suite for
- 109 GPGPU on mobile and embedded GPUs. *2018 IEEE International Symposium on Workload*
- 110 *Characterization (IISWC)*, 25–35. <https://doi.org/10.1109/IISWC.2018.8573477>
- 111 Mittal, S., & Vaishay, S. (2019). A survey of techniques for optimizing deep learning on GPUs.
- 112 *Journal of Systems Architecture*, 99, 101635. <https://doi.org/10.1016/j.sysarc.2019.101635>
- 113
- 114 Muñoz Lopez, J. E., & Winston, S. (2025). Hands-on vulkan® ray tracing with dynamic
- 115 rendering. *Proceedings of the Special Interest Group on Computer Graphics and Interactive*

- 116 *Techniques Conference Labs*. <https://doi.org/10.1145/3721251.3742861>
- 117 Nguyen, H. (2007). *GPU gems 3*. Addison-Wesley Professional. ISBN: 9780321515261
- 118 Plebański, P., Kelm, A., & Hajder, M. (2025). *Efficiency and development effort of OpenCL*
119 *interoperability in vulkan and OpenGL environments: A comparative case study*. 111–119.
120 <https://doi.org/10.5220/0013529000003964>
- 121 Rossant, C., & Rougier, N. P. (2021). High-performance interactive scientific visualization with
122 datoviz via the vulkan low-level GPU API. *Computing in Science & Engineering*, 23(4),
123 85–90. <https://doi.org/10.1109/MCSE.2021.3078345>
- 124 Unterguggenberger, J., Kerbl, B., & Wimmer, M. (2023). Vulkan all the way: Transitioning
125 to a modern low-level graphics API in academia. *Computers & Graphics*, 111, 155–165.
126 <https://doi.org/https://doi.org/10.1016/j.cag.2023.02.001>

DRAFT