The Journal of Open Source Software

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

Tomas Chlubna [1]

1 Faculty of Information Technology, Brno University of Technology, Czech Republic

DOI: 10.21105/joss.08871

Software

- Review
- Repository
- Archive

Editor: Christoph Junghans

Reviewers:

- @boonth
- @dhelmrich

Submitted: 07 August 2025
Published: 17 January 2026

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 (general-purpose computing on graphics processing units) (Hu et al., 2016) 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). 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 vendor-free multiplatform API with a lot of features that are usually not fully supported by drivers. These APIs are only for GPGPU computations and lack access to GPU rendering features. The subsequent APIs can use 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 verbosity of Vulkan makes it difficult to use for quick experiments. 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 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. Users do not need to study complex GPU APIs to conduct scientific experiments. This addresses a frequent issue in science, where implementing an experimental

concept often requires a considerable amount of time due to technical difficulties.

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 vkCompViz library does not require 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), and VComputeBench for benchmarking purposes (Mammeri & Juurlink, 2018). The vkCompViz library aims to be a general GPGPU scientific framework. Kokkos (Trott et al., 2022) and RAJA (Beckingsale et al., 2019) are C++ abstractions for performance-portable parallel computation across CPUs and GPUs. In contrast, vkCompViz is GPU-oriented. Its goal is not only to simplify parallel computing but also visualization and data loading and provide access to additional GPU capabilities, such as rendering and other features exposed by Vulkan.

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.

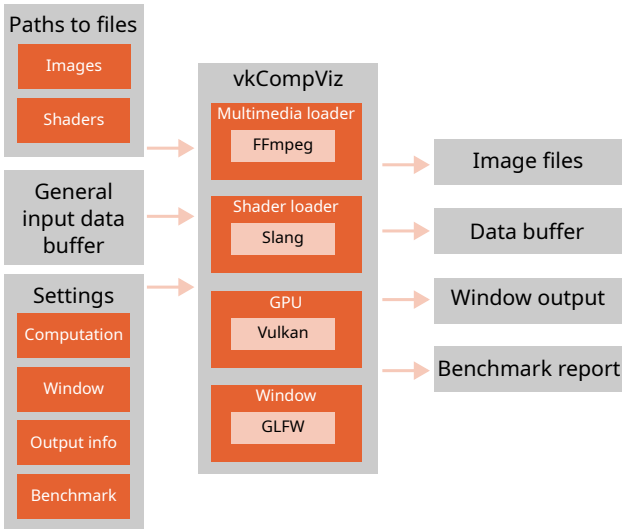

Figure 1: The figure describes the architecture of vkCompViz library.

Usage

The project uses CMake, the 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 with a given factor. The 3D Viewer also shows how the rendering pipeline can be customized to render an input 3D model file. The Parallel Reduction example shows how the library can be used to accelerate a summation of a big array of numbers. This example demonstrates a

simple use case of an experimental evaluation of the GPU accelerated algorithm (Jradi et al., 2020), compared to its CPU variant. The experiment was carried out on a machine equipped with a NVIDIA GeForce RTX 3060 Ti and Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz CPU, running Arch Linux. The output of several runs of this example is shown in Figure 2. Figure 3 shows the detailed runs where the GPU started to be faster than CPU. The results show that computational time is shorter with massively parallel GPU architecture. However, the data transfer delay increases the total time and shows that using GPU for this use case is viable only when the data is already generated on GPU or later used there or when the data size reaches certain amount. This fact is in alignment with previously published findings (Dinkelbach et al., 2012).

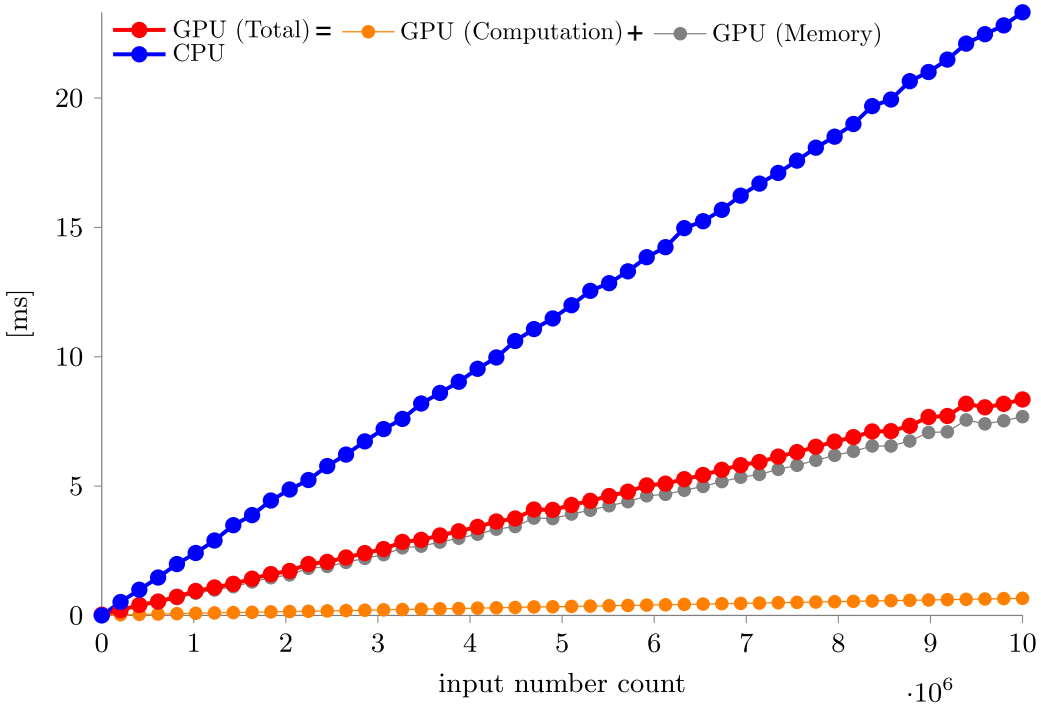

Figure 2: Comparison of GPU and CPU array summation is shown in the chart.

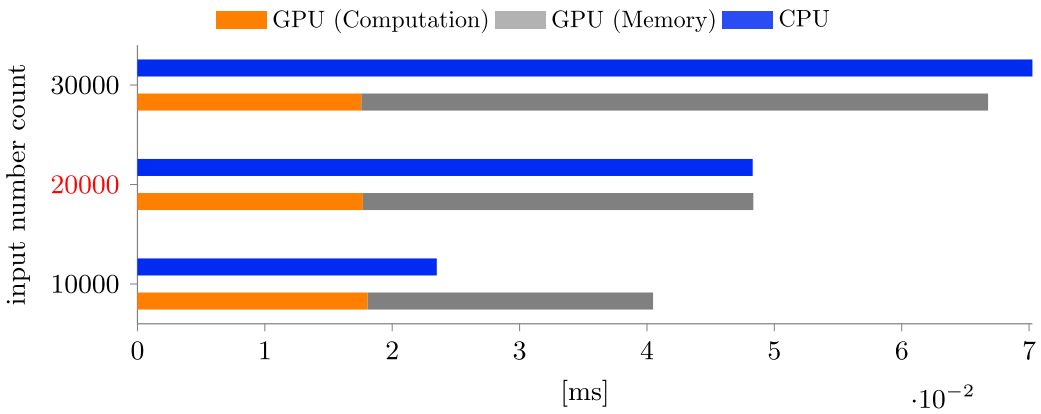

Figure 3: The turning point where GPU starts to be faster than CPU for array summation is shown with two neighboring measurements.

References

Bailey, M. (2019). Introduction to the Vulkan® computer graphics API. *SIGGRAPH Asia 2019 Courses*. https://doi.org/10.1145/3355047.3359405

Beckingsale, D. A., Burmark, J., Hornung, R., Jones, H., Killian, W., Kunen, A. J., Pearce, O., Robinson, P., Ryujin, B. S., & Scogland, T. R. (2019). RAJA: Portable performance for large-scale scientific applications. *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 71–81. https://doi.org/10.1109/P3HPC49587.2019.00012

Dinkelbach, H., Vitay, J., Beuth, F., & Hamker, F. (2012). Comparison of GPU- and CPU-implementations of mean-firing rate neural networks on parallel hardware. *Network (Bristol, England)*, *23*. https://doi.org/10.3109/0954898X.2012.739292

Henriksen, T. (2024). A comparison of OpenCL, CUDA, and HIP as compilation targets for a functional array language. *Proceedings of the 1st ACM SIGPLAN International Workshop on Functional Programming for Productivity and Performance*, 1–9. https://doi.org/10.1145/3677997.3678226

Hu, L., Che, X., & Zheng, S.-Q. (2016). *A closer look at GPGPU*. *48*(4). https://doi.org/10.1145/2873053

Jia, S., Tian, Z., Ma, Y., Sun, C., Zhang, Y., & Zhang, Y. (2021). A survey of GPGPU parallel processing architecture performance optimization. *2021 IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall)*, 75–82. https://doi.org/10.1109/ICISFall51598.2021.9627400

Jradi, W. A. R., Nascimento, H. A. D. do, & Martins, W. S. (2020). A GPU-based parallel reduction implementation. In C. Bianchini, C. Osthoff, P. Souza, & R. Ferreira (Eds.), *High performance computing systems* (pp. 168–182). Springer International Publishing. https://doi.org/10.1007/978-3-030-41050-6_11

Lavrič, P., Bohak, C., & Marolt, M. (2018). Vulkan abstraction layer for large data remote rendering system. In L. T. De Paolis & P. Bourdot (Eds.), *Augmented reality, virtual reality, and computer graphics* (pp. 480–488). Springer International Publishing. https://doi.org/10.1007/978-3-319-95270-3_40

Mammeri, N., & Juurlink, B. (2018). VComputeBench: A Vulkan benchmark suite for GPGPU on mobile and embedded GPUs. *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 25–35. https://doi.org/10.1109/IISWC.2018.8573477

Mittal, S., & Vaishay, S. (2019). A survey of techniques for optimizing deep learning on GPUs. *Journal of Systems Architecture*, *99*, 101635. https://doi.org/10.1016/j.sysarc.2019.101635

Muñoz Lopez, J. E., & Winston, S. (2025). Hands-on Vulkan® ray tracing with dynamic rendering. *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Labs*. https://doi.org/10.1145/3721251.3742861

Nguyen, H. (2007). *GPU gems 3*. Addison-Wesley Professional. ISBN: 9780321515261

Plebański, P., Kelm, A., & Hajder, M. (2025). *Efficiency and development effort of OpenCL interoperability in Vulkan and OpenGL environments: A comparative case study*. 111–119. https://doi.org/10.5220/0013529000003964

Rossant, C., & Rougier, N. P. (2021). High-performance interactive scientific visualization with Datoviz via the Vulkan low-level GPU API. *Computing in Science & Engineering*, *23*(4), 85–90. https://doi.org/10.1109/MCSE.2021.3078345

Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D.,

Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., & Wilke, J. (2022). Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, *33*(4), 805–817. https://doi.org/10.1109/TPDS.2021.3097283

Unterguggenberger, J., Kerbl, B., & Wimmer, M. (2023). Vulkan all the way: Transitioning to a modern low-level graphics API in academia. *Computers & Graphics*, *111*, 155–165. https://doi.org/10.1016/j.cag.2023.02.001