Mewo: native shader playground

Charles Wang

Advisor: Adam Mally

University of Pennsylvania

ABSTRACT

Shader development and the tooling around it can be intimidating and time-intensive to set up without having to rely on large and heavy applications like game engines or digital content creation software (DCC). Projects like Shadertoy demonstrate the need and potential success for a platform that abstracts away all the infrastructure and setup required to start creative coding.

To that end, Mewo is a shader editor and playground for creativity and experimentation. It aims to offer a native, local-first, offline-first solution to programming shaders. Above all, it will feature an intuitive, fast, and customizable visual design and user experience that removes as much friction as possible.

Project website: https://mewo.czw.sh

GitHub: https://github.com/aczw/mewo

1. INTRODUCTION

When it comes to writing shaders, it is rare to be doing such an activity in an isolated environment. That is, the act of writing shaders is usually done as part of a larger project, such as a website's codebase, within a game (and therefore a game engine of some kind), or in a DCC. In all three of these cases, the focus is not on writing the particular shader per se, but rather a larger overarching goal: a website, a game, or artwork respectively.

However, there does exist a certain kind of work where writing the shader itself is the main task at hand. In the demoscene, the shader is the main focus. As a form of creative coding, it is not important whether the shader is being written in a game engine, DCC, or otherwise. Finally, it is entirely possible to write entire self-contained programs within a single shader, such as raytracers. It would be rather inefficient and roundabout to, for instance, open a game engine, create a scene, add geometry to it, add a texture to the geometry, apply a material, and then finally get to writing the shader.

Mewo is an application that aims to solve these problems. It offers a dedicated, isolated, and focused environment to write shaders (with a focus on some sort of visual output, usually changing over time). When nothing else matters except the shader program that is currently being written, Mewo shines the most; it fills the niche of allowing a user to start their computer, launch one single application, and immediately be able to start coding.

In terms of contributions, Mewo is best described using its feature list. These will be expanded on later. This project makes the following contributions:

- Single executable, native, cross-platform application
- Solves a single, nontrivial problem well, hopefully becoming a tool in an artist's or engineer's toolkit
- Creates an offline-first and local-first environment for creativity and experimentation, employing a file-over-app philosophy of storage and openness

1.1 Design Goals

Mewo is flexible in that it lets anyone who needs to write shaders successfully do their job.

For the game developers and technical artists that are writing shaders as part of a larger project, Mewo provides a quick and extremely accessible environment to test out changes or features. It will then be easy to copy-and-paste the shader. This process should theoretically be quicker and easier than alternative methods.

Creative coders and artists can also use the environment because for them, a stronger focus might be placed on the shader itself. In that way, Mewo provides a comfortable place to develop their project, allowing them to see their visual output and stay in the flow.

1.2 Projects Proposed Features and Functionality

Here, more details are provided for the contributions stated previously.

Single executable, native, cross-platform. Mewo will be distributed as a single executable that contains everything it needs to run except for system APIs. Care will be taken to introduce features and libraries that work across the major OSes, like macOS and Windows. Mewo will be compiled natively for these respective platforms. This helps support the stated project goal of removing friction: by staying self-dependent, no additional steps are required to use the application.

Solves a single, nontrivial problem. I believe the problem at hand is nontrivial because Mewo aims to provide a feature-rich user experience. It wouldn't take that long to create some simple interface that lets you write code and see the output side-by-side. However, Mewo will also feature hot reloading, ingesting external outputs like images, videos, and hardware media sources, and the

saving/loading of previous shaders, all included in order to make shader development more enjoyable.

Offline-first and local-first. Mewo will be more accessible because it does not require an internet connection. While certain features may require fetching data from online sources, these are not necessary in order to use the program. I also believe in letting users inspect the data that the application is outputting. To that end, Mewo will prioritize saving projects in an accessible, human readable, and non-proprietary manner, such that the user is not locked into using the program.

2. RELATED WORK

Shadertoy. This may be the elephant in the room. Shadertoy, accessible at https://www.shadertoy.com, is an online platform and community that focuses on allowing anyone to write shaders and share them. On paper, Mewo and Shadertoy share the same philosophy: let users write shaders as quickly as possible, and provide any tools necessary so they don't feel restricted. However, Mewo is able to do this without an internet connection, and will offer a much better user interface to go along with it. The hope is that "Turn on computer > Open Mewo" is more appealing than "Turn on computer > Open browser > Navigate to the Shadertoy website."

Processing / p5.js / Nannou. These platforms provide a set of functionality that lets users make art using code. The scope of these projects, however, is much too broad and subsumes the goals stated here. It is possible to use shaders to create effects, but it is only one of many, many choices available in these libraries. Mewo is much more restricted in functionality and direction, which is a benefit if you're only interested in shader development.

hydra. hydra, accessible at https://hydra.ojack.xyz, is, in their own words, a "live code-able video synth and coding environment." It provides an API surface to let you create procedural graphics and effects, and in that way serves the same function as Mewo. However, like Shadertoy, it is entirely browser-based, and uses a custom set of functions to create visual output. Mewo will focus on using standardized shader languages that will be more approachable and well known.

3. PROJECT PROPOSAL

A delicate balance between not reinventing the wheel and using flexible libraries, frameworks, and APIs will need to be maintained in order to achieve all of the goals in this project.

3.1 Anticipated Approach

At its core, Mewo will take the shader code that the user has written, attempt to compile it, and when successful, draw the result on a quad projected in world space. For more concrete details, I will now describe my approach for each of the three big features/contributions mentioned previously.

Single executable, native, cross-platform. C++23 will be used to program Mewo. Depending on the platform, either MSVC or Apple Clang will compile the code to a native binary, packaged either as a .app on macOS or a .exe on Windows. Preprocessor directives will be used to call OS-specific APIs in order to support hardware camera input and other system-specific quirks. The C++ Standard Library will be utilized as often as possible (for example, using <filesystem> as an abstract method of manipulating computer storage), but some functionality simply isn't available in the Standard or is better implemented somewhere else.

Potential third-party libraries that will be used include SDL3 for window management and ImGui for building and styling Mewo's user interface, including panes, buttons, tooltips, and the code editor itself (although I may need to pull in an additional library for the editor itself, as I want to support future features such as autocomplete or custom dialogs when typing).

The graphics API should also be discussed. Due to the cross-platform nature of the program, system-specific APIs like DirectX (for Windows) or Metal (for macOS) would not be appropriate. Instead, Mewo could use an API specifically designed for cross-platform like Vulkan or OpenGL, or a higher abstraction like NVIDIA's NVRHI, or WebGPU. Out of all these choices, WebGPU is the most appealing because it enables potential porting of the program to the web while still allowing the program to stay in C++ via Dawn, and features a slightly less wieldy API compared to Vulkan, with more up-to-date programming paradigms (unlike OpenGL).

Solves a single, nontrivial problem. One of the most important quality-of-life features a program like Mewo could support is hot reloading. The current planned approach is to have a separate thread act as a "watcher" that checks on a set timer whether the shader code has changed or not. Additional checks or attempts to compile the code should be debounced if further changes to the code are introduced.

A big, important question is what language should Mewo let users write their shaders in? The ideal answer to this is "any language" as that provides the most flexibility to the user. However, given the project's time constraints it is more realistic to stick to 1-2 languages. Since Mewo is using WebGPU, WGSL is naturally supported. Tint, a library being developed as part of Dawn, focuses on WGSL compilation, notably also supporting conversions to and from different shader languages. In theory, this would allow users to write shaders in more established languages like GLSL or HLSL, and Mewo would take care of converting it to WGSL via Tint. Another promising library is Slang, which is a superset of HLSL, and also supports compiling to WGSL. As development begins and as time permits, both approaches will be interesting to try.

One of the ways Shadertoy enables rapid shader development is its predefined shader uniforms, including viewport resolution, playback time, time delta, etc. This can easily be done in Mewo as well since it is fully in control of the WebGPU graphics pipeline and can simply manage these uniforms for the user.

Offline-first and local-first. Offline first is actually easier to implement because, well, it requires no additional work on my part (and in fact requires no work!) As previously stated, however, some features may require connectivity, including fetching images and videos from online. One approach I can take is using a third-party library to handle network requests, downloading the media to an appropriate location on disk.

This nicely leads into the next topic, project loading and saving. Mewo's usability would be severely degraded if users weren't able to save their work and return to it at a later date. For both development's sake and usability by the user, the project state can be stored in a format like JSON, for which there are many third-party libraries available. However, projects may also include external data like image files. Therefore, Mewo will use a method often used by other applications, in which the media and project state are stored in an archive file format such as ZIP, enabling both compression and the ability to store everything related to a project as a single file. (Mewo can also get fancy by using a custom file extension such as .mewoproj.)

3.2 Target Platforms

Mewo is targeting macOS and Windows as platforms. The operating system and graphics APIs will abstract away the graphics hardware available; therefore, no specific hardware or GPU will be required.

3.3 Evaluation Criteria

Due to this project's uncanny resemblance to Shadertoy, it might be worth it to port over a few famous Shadertoy examples to Mewo. Since this would only demonstrate that we're even, one potential advantage Mewo has is performance, as it's not restricted by browser capabilities and is compiled to run natively.

Performance might become a big difference factor when compared to other related projects. One of the many reasons that, for instance, Nannou became popular is because it uses Rust as its language, allowing it to run at native speeds. It would be interesting to see if Mewo can run any existing shader examples with some ridiculously high parameters (billions of objects, larger resolution, higher trace depth when raytracing, etc.)

4. RESEARCH TIMELINE

The milestones for Mewo will be based on the alpha, beta review, and final presentation dates set for this class. For each milestone, the following should be finished:

Alpha

- Finished setting up executable packaging for macOS and Windows, so you can open Mewo as a standalone window with content in it
- Ability to write code, ability to click a button to compile the code, and ability to see visual output of compiled code
- Shader environment with predefined uniform variables
- Saving of projects to disk, and at minimum loading of projects via command line

Beta

- External media support (for now, just local images and videos)
- Hot reloading
- More polished user interface
- Finish loading and saving of projects
- Simple on-page website offering a description of the program and download links
- Documentation within the codebase (in Markdown)

Project Final Deliverables

- Support camera from local computer and online media as external sources
- Final, polished, actually styled (aesthetically pleasing) user interface. It should not look like the default ImGui toggles and panes.
- Basic documentation on the website, maybe under a /docs subpage
- A complete program that you can download from either GitHub/the website, install via normal means, and then launch, and start using
- Everything from past milestones, just more refined and actually finished (if not)

Project Future Tasks

I have many ideas that can be considered stretch goals if I have more time or finish features early.

- Stateful application. By this I mean saving settings or states for Mewo, the application itself, to somewhere on disk (like %AppData% on Windows for instance). This lets users set application-wide settings, remember previous Mewo window sizes, and everything else below.
- Instead of launching a blank project, Mewo should feature a home screen displaying previously created projects at launch.
- Include commonly written/used functions like noise, RNGs, easing curves, etc.
- Ability to save custom snippets or macros.
- Some sort of "Shadertoy converter" that lets you copy code from a Shadertoy link and automatically make it run in Mewo, and potentially an exporter as well.
- Official support for Linux (everything I'm using should work on Linux, but I don't have time to do actual testing so I've only been mentioning macOS and Windows)
- Port Mewo to the web via the Emscripten Dawn library