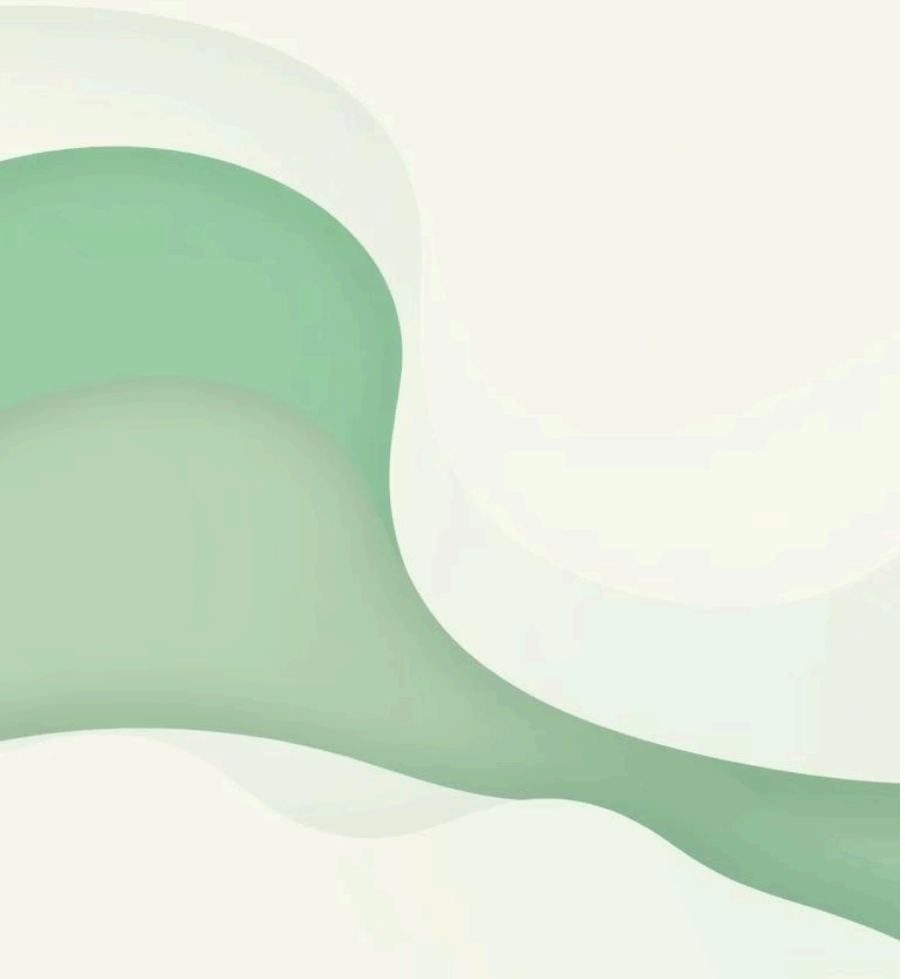

Setup Documentation

This document describes how to set up and build from scratch on Windows and Linux using Clang and the Vulkan SDK. It consolidates the steps shown in the tutorial videos into a clear technical reference.

1. Prerequisites Required Software

Git

Clang / LLVM

LLVM Builds (**Windows installer**)

On Linux: install via your package manager
(sudo apt-get install clang, pacman -S clang, etc.)

Visual Studio Code

Download Here

Install the C/C++ extension and optionally:

- Shader language support
- To-do highlight
- Themes (e.g., Gruvbox)

Vulkan SDK

Download Vulkan SDK

On Windows: run the installer and ensure
"Add LLVM to system path for all users" is selected.

On Linux:

- Download and extract the tarball
- Edit the shell configuration files for all users:

```
sudo nano /etc/zsh/zshrc
sudo nano /etc/bash.bashrc
```

Add at the end of each file:

```
# Vulkan SDK environment for all users

SETUP_ENV="Path_to_your_extracted_Vulkan_SDK/setup-env.sh"

if [ -z "$VULKAN_SDK" ]; then
    if [ -f "$SETUP_ENV" ]; then
        source "$SETUP_ENV"
        echo "Vulkan SDK environment loaded successfully."
    else
        echo "Warning: $SETUP_ENV not found!"
    fi
fi
```

Replace Path_to_your_extracted_Vulkan_SDK with the full path to your Vulkan SDK folder.

This ensures VULKAN_SDK is set for all users automatically.

Ensure VULKAN_SDK environment variable is set.

Additional Linux Dependencies

On Ubuntu/Debian-based distributions:

```
sudo apt-get install libx11-dev libxkbcommon-x11-dev libx11-xcb-dev
```

Other distros: install equivalent X11/XCB development libraries.

2. Repository Setup

Create a project folder:

```
git clone https://github.com/nourOkaram/Kaffi
```

Simplify folder structure if needed (move files up one level).

Ensure you have a code/ folder containing the repository.

3. Build Scripts Windows – engine/build.bat

Uses clang to compile engine sources into engine.dll and engine.lib.

Links against:

- user32.lib
- vulkan-1.lib

Defines:

- _DEBUG, KEXPORT, _CRT_SECURE_NO_WARNINGS.

Linux – engine/build.sh

Compiles into libengine.so.

Links against:

- vulkan, xcb, x11, x11-xcb, xkbcommon.

Defines: _DEBUG, KEXPORT.

⚠ ⚠️ **Note:** Make build scripts executable on Linux:

```
chmod +x engine/build.sh testbed/build.sh build-all.sh
```

Testbed – testbed/build.bat | build.sh

Builds the testbed application.

Links against engine.lib (Windows) or libengine.so (Linux).

Linux build script uses -Wl,-rpath,. so runtime libraries are found locally.

Build All – buildall.bat | buildall.sh

Runs both engine and testbed build scripts.

Stops execution if an error occurs.

4. Core Code engine/source/defines.h

Defines fixed-width integer types (u8, i32, f32, etc.).

Static assertions ensure type sizes match expectations.

Platform detection for Windows, Linux, Unix, Android, Apple.

KAPI macro for function exports (__declspec(dllexport) / __declspec(dllimport)).

5. VS Code Setup

Configuring Visual Studio Code is crucial for a streamlined development workflow. These essential JSON files provide IntelliSense, build tasks, debugging, and recommended extensions to enhance your coding experience.

c_cpp_properties.json

- Configures IntelliSense for both Windows and Linux environments, ensuring proper code completion and error checking.
- Specifies include paths, such as `engine/source` and the Vulkan SDK, allowing the compiler to locate necessary headers.
- Defines preprocessor macros like `_DEBUG` and `KEXPORT` for conditional compilation.
- Sets the compiler path to Clang, aligning with the project's build system.

tasks.json

- Defines custom build tasks for the project.
- Includes tasks for building the **Engine**, the **Testbed** application, and an overarching **Build All** task to compile everything.

launch.json

- Configures debugger settings for running and debugging the application.
- For Windows, it uses `cppvsdbg` to launch `testbed.exe`.
- For Linux, it utilizes `cppdbg` (GDB) to execute the `testbed` binary.

extensions.json

- Recommends project-specific VS Code extensions to ensure a consistent development environment across the team.
- Prompts users to install relevant tools like the C/C++ extension, shader language support, and more.

6. Building and Running

Once your VS Code environment is set up, you can easily compile and run the project using the defined tasks or directly via the build scripts.

Open Terminal

Navigate to your project root in the VS Code integrated terminal, or open an external terminal in the project directory.

Execute Build Script

Run the all-in-one build script:

```
./build-all.sh (Linux)
build-all.bat (Windows)
```

Alternatively, use the VS Code tasks (Ctrl+Shift+B) to select "Build All".

Launch Application

After a successful build, the executable will be in the `bin/` directory. Run it directly:

```
./bin/testbed (Linux)
bin/testbed.exe (Windows)
```

Or, use the debug configuration in VS Code (F5) to launch the `testbed` application.

Success! Upon successful compilation and execution, you should see the testbed application running, demonstrating the engine's capabilities.

7. Build Outputs

Inside bin/:

Windows:

- engine.dll → Engine library
- engine.lib → Import library
- engine.pdb → Debug symbols
- testbed.exe → Test application
- testbed.pdb → Debug symbols

Linux:

- libengine.so → Shared object library
- testbed → Test application

8. Doxygen Quick Setup

Windows

Download the installer from the official Doxygen website:

https://www.doxygen.nl/download.html

Run the installer and follow the on-screen steps. Once installed, open Command Prompt and execute the following command to generate documentation:

```
doxygen path\to\Doxyfile
```

Linux

Install Doxygen using your distribution's package manager:

```
sudo apt-get install doxygen  # Debian/Ubuntu
sudo yum install doxygen      # CentOS/Fedora
```

After installation, generate documentation by running Doxygen with your `Doxyfile`:

```
doxygen /path/to/Doxyfile
```

💡 **Note:** This quick setup focuses on downloading and running Doxygen with an existing `Doxyfile`. No additional configuration is required within Doxygen itself for basic operation.