Integrated Software Architecture for Low-Latency OpenVR Haptic Synthesis via SAMD21 and DRV2605L Hardware

The advancement of immersion in virtual reality environments is increasingly dependent on the fidelity and latency of tactile feedback systems. While visual and auditory modalities have reached high levels of maturity, haptic interfaces often remain a secondary consideration, constrained by generic vibration motors and high-latency software stacks. The system under consideration utilizes the Adafruit QT Py SAMD21 as a bridge between the SteamVR ecosystem and the Texas Instruments DRV2605L haptic driver, providing a pathway for high-resolution tactile actuation.[1] This architecture necessitates a sophisticated multi-layered software stack implemented primarily in the Rust programming language, encompassing a Windows-based OpenVR driver, a robust host-device communication protocol over USB, and an embedded firmware environment designed for real-time responsiveness.[3]

Hardware Interfacing and Peripheral Configuration

The selection of the Adafruit QT Py SAMD21 provides a compact yet powerful foundation for haptic control. At its core, the ATSAMD21E18 features a 32-bit ARM Cortex-M0+ processor running at 48 MHz, which offers sufficient computational headroom for managing the simultaneous demands of the Universal Serial Bus (USB) and the Inter-Integrated Circuit (I2C) peripherals.[1] The board's physical design includes the STEMMA QT connector, a SparkFun Qwiic-compatible JST SH 4-pin interface that facilitates solderless connection to I2C sensors and drivers.[6]

The I2C bus on the SAMD21 is managed through the Serial Communication Interface (SERCOM) peripheral. In the QT Py's default configuration, SERCOM2 is utilized for I2C communication via pins PA08 (SDA) and PA09 (SCL).[7] These pins are routed to the STEMMA QT port, providing a 3.3V logic level that is directly compatible with the DRV2605L haptic driver.[2] While the SAMD21 pins do not feature internal pull-up resistors by default, the Adafruit STEMMA QT ecosystem mitigates this by integrating 2.2K-10K resistors on the breakout boards, ensuring signal integrity on the I2C lines.[7]

SAMD21 Pin	Signal	SERCOM Assignment	Physical Port
PA08	I2C SDA	SERCOM2 Pad 0	STEMMA QT / D4 [7]

PA09	I2C SCL	SERCOM2 Pad 1	STEMMA QT / D5 [7]
PA24	USB D-	Native USB Peripheral	USB-C Connector [6]
PA25	USB D+	Native USB Peripheral	USB-C Connector [6]
PA02	Analog A0	ADC / DAC	Header Pin A0 [7]

The DRV2605L acts as the haptic actuator controller, supporting both Eccentric Rotating Mass (ERM) and Linear Resonant Actuator (LRA) motors.[8] The device is addressable via I2C at a fixed 7-bit address of 0x5A.[2] Its internal architecture is centered around a sophisticated playback engine that can trigger pre-programmed waveforms or operate in a Real-Time Playback (RTP) mode.[2] For VR haptics, where responsiveness is paramount, the RTP mode allows the host to stream 8-bit intensity values directly to the motor, enabling the translation of complex in-game physics into tactile sensations.[10]

Embedded Systems Architecture: The Rust no_std Environment

Developing firmware for the SAMD21 in Rust requires a departure from the standard library (std) used in application-level programming. Instead, the project must utilize the core library, which provides a platform-independent subset of Rust's functionality.[5] This bare-metal approach is facilitated by the thumbv6m-none-eabi target triple, which identifies the ARM v6-M architecture and the absence of an underlying operating system.[5]

The software stack on the microcontroller is built upon a hierarchy of crates. The Peripheral Access Crate (PAC) provides the register-level definitions for the SAMD21, while the Hardware Abstraction Layer (HAL) crate, specifically atsamd-hal, offers higher-level traits and structures for managing SERCOM, GPIO, and the USB controller.[13] To satisfy the requirements of the project, the firmware must be configured with specific attributes and handlers.

The #![no_std] and #![no_main] attributes inform the compiler that it should not link the standard library or look for a traditional main function.[5] In its place, the cortex-m-rt crate provides a runtime that handles the reset handler and stack initialization, allowing the use of the #[entry] attribute to define the firmware's starting point.[5] Furthermore, a panic handler must be defined to manage system-level errors. The panic-halt crate is a common choice, which puts the processor into an infinite loop upon a panic, though panic-persist can be used for more advanced debugging scenarios by saving the panic message to a dedicated RAM

region.[5]

The build process is controlled through .cargo/config.toml, where the default target is set to thumbv6m-none-eabi and specific linker flags are provided.[5] For the SAMD21, the -C link-arg=-Tlink.x flag is required to use the memory layout defined by the cortex-m-rt crate.[5] This ensures that the code, data, and stack are correctly mapped to the 256KB of Flash and 32KB of SRAM available on the QT Py.[1]

The DRV2605L Haptic Engine: Mechanics and Logic

The DRV2605L is designed to abstract the complexities of haptic motor driving, particularly the back-EMF sensing required for high-performance LRAs.[2] In a standard implementation, an LRA has a narrow resonant frequency range, typically between 170 Hz and 200 Hz. Driving the LRA at this frequency is essential for maximizing efficiency and vibrational amplitude.[8] The DRV2605L's Smart-Loop architecture performs automatic resonance tracking, constantly adjusting the drive signal to match the physical state of the motor, which may shift due to temperature or mounting conditions.[2]

Feature	ERM Support	LRA Support
Drive Type	Unidirectional DC	Bidirectional AC (Resonant) [2]
Braking	Passive / Active	Active Phase Reversal [2]
Response Time	Moderate (Inertia-bound)	Very Fast (Instantaneous start/stop) [8]
Complexity	Low	High (Requires calibration) [11]

The software control of the DRV2605L is achieved through a sequence of I2C transactions. Upon power-up, the device enters a standby state to conserve power. The initialization sequence involves setting the MODE register (0x01) to exit standby and perform an auto-calibration.[2] Calibration is a critical step for LRAs, as the driver must determine the motor's resonant frequency and the appropriate overdrive and braking voltages.[11]

For the OpenVR integration, the Real-Time Playback (RTP) mode is the most applicable operating state. When the MODE register is set to 0x05, the driver continuously drives the motor based on the value in the RTP_INPUT register (0x02).[10] This value is an 8-bit integer that represents the desired intensity. In closed-loop LRA mode, the value determines the

swing amplitude of the AC signal, while in open-loop ERM mode, it correlates to the DC voltage applied to the motor.[10]

The frequency of haptics is a parameter frequently requested by VR applications via the OpenVR API. In the case of LRAs, the frequency is generally fixed to the hardware resonance; however, the DRV2605L allows for "drive time" adjustments. The drive time is effectively half the period of the LRA frequency, and it can be manually adjusted to tune the "feel" of the vibration.[11] The conversion from frequency in Hz to the internal register value is calculated as follows:

$$LRA\_Drive\_Time\_Value = \frac{1000}{2 \times f_{LRA} \times 0.1} - 0.5$$

This allows the embedded firmware to translate the frequency data received from the OpenVR driver into a register update that modifies the physical behavior of the motor in real time.[11]

Host-Device Communication: Latency and Protocol Selection

The communication between the PC and the SAMD21 is the most significant bottleneck for haptic latency. While the USB 2.0 interface on the SAMD21 is capable of high throughput, the choice of USB class profoundly affects the timing of data delivery.[6] Traditional serial communication via the Communications Device Class (CDC) is often favored for its simplicity; however, it is subject to non-deterministic delays introduced by the operating system's serial driver and the bulk transfer nature of CDC packets.[16]

For haptic applications, the Human Interface Device (HID) class is superior. HID utilizes interrupt transfers, which allow the device to specify a polling interval.[15] In Full-Speed USB mode (which the SAMD21 supports), the polling interval can be as low as 1 millisecond. This provides a guaranteed window in which the host can send haptic commands to the device.[15]

Communication Metric	USB CDC (Serial)	USB HID (Interrupt)
Transfer Type	Bulk (Asynchronous)	Interrupt (Synchronous Polling) [19]
Best Case Latency	< 1ms	1ms (fixed) [15]

Worst Case Latency	> 10ms (OS dependent)	2ms [15]
Implementation	usbd-serial crate	usbd-hid crate [20]
OS Driver	Vendor or Generic Serial	Native HID Driver [17]

Using the usb-device and usbd-hid crates in Rust, the SAMD21 firmware can be configured to expose a Raw HID endpoint. This endpoint accepts custom data packets containing haptic parameters.[20] A robust packet design might include a single-byte command identifier followed by the 8-bit RTP intensity and a 16-bit duration value. On the host side, the OpenVR driver utilizes the hidapi-rs library to open the device by its Vendor ID (VID) and Product ID (PID) and transmit these packets whenever a haptic event is generated by the VR runtime.[20]

The reliability of this link is further enhanced by the deterministic nature of interrupt transfers. Even if the host PC is under significant CPU load, the USB controller's hardware scheduler will prioritize the HID interrupt packets, ensuring that haptic feedback remains synchronized with the visual events in the headset.[15] This is particularly critical in VR, where sensory desynchronization can lead to discomfort or motion sickness.

The OpenVR Driver Framework: Bridging Software and Physicality

The integration of the haptic system into the SteamVR environment requires the development of an OpenVR driver. Unlike standard applications that use the OpenVR API as clients, a driver acts as a provider of hardware functionality.[23] The driver is implemented as a Windows dynamic link library (DLL) that is loaded by the SteamVR server (vrserver.exe) upon startup.[26]

The primary entry point for any OpenVR driver is the HmdDriverFactory function. This function is called by SteamVR to retrieve pointers to the driver's provider interfaces.[23] In Rust, this function must be declared with #[no_mangle] and extern "C" to ensure the exported name matches what SteamVR expects.[28]

Rust

```
#[no_mangle]
pub extern "C" fn HmdDriverFactory(
    p_interface_name: *const i8,
    p_return_code: *mut i32,
```

```
) -> *mut std::ffi::c_void {
    // Interface discovery logic
}
```

The driver must implement several key interfaces, most notably IServerTrackedDeviceProvider and ITrackedDeviceServerDriver.[23] The provider interface manages the lifecycle of the driver, including its initialization and the polling of events.[23] The device driver interface represents the haptic controller itself. During the device's activation, the driver calls IVRDriverInput::CreateHapticComponent to register a haptic output with SteamVR.[29]

Each haptic component is identified by a path, such as /output/haptic.[29] This path allows applications and the SteamVR binding system to route haptic signals to the specific device. When an application calls TriggerHapticVibration, the SteamVR runtime dispatches a VREvent_Input_HapticVibration event to the driver.[25] This event contains the amplitude, frequency, and duration of the requested vibration, which the driver must then transmit to the SAMD21.[29]

The Rust DLL Implementation: FFI and C++ Interface Emulation

Implementing an OpenVR driver in Rust presents a unique challenge: OpenVR is a C++ API designed around classes with virtual functions.[23] Rust does not have a native representation of C++ classes, so the driver must manually reconstruct the binary layout of these classes to maintain Application Binary Interface (ABI) compatibility.[23] This is achieved by creating structures in Rust that mirror the vtable (virtual function table) of the corresponding C++ interfaces.[23]

OpenVR Interface	Rust Equivalent Structure	Responsibility
IServerTrackedDeviceProvider	VTable_ServerProvider	Driver lifecycle and registration [23]
ITrackedDeviceServerDriver	VTable_DeviceDriver	Per-device activation and property management [23]
IVRServerDriverHost	VTable_DriverHost	Callback interface into SteamVR [25]
IVRDriverInput	VTable_DriverInput	Input/Output component

		creation [29]

The Rust compiler's cdylib crate type is used to produce the final .dll file.[28] To handle the strings and handles passed between Rust and the OpenVR C++ code, the driver uses the std::ffi module, specifically CString for passing Rust strings to C and CStr for reading strings from C.[28] Because OpenVR on Windows is compiled with the Microsoft Visual C++ (MSVC) toolchain, the Rust driver must also be built using the x86_64-pc-windows-msvc target to ensure that calling conventions and symbol resolutions are identical.[4]

Furthermore, the driver must link against the openvr_api.lib to access the utility functions provided by the SDK.[4] The openvr-sys crate can automate some of this process by providing raw FFI bindings; however, for driver development, a more manual approach to vtable construction is often required to satisfy the specific inheritance patterns of the vrserver.[4]

Haptic Signal Mapping: From OpenVR Events to Motor Actuation

Once the VREvent_Input_HapticVibration event is intercepted by the OpenVR driver, it must be translated into a series of commands for the DRV2605L. The event data provides three primary scalars: duration (in seconds), frequency (in Hz), and amplitude (0.0 to 1.0).[29]

The amplitude mapping is relatively straightforward: the 0.0 to 1.0 scalar is multiplied by 255 to create an 8-bit RTP value.[11] However, the human perception of vibration is non-linear; applying a logarithmic scaling to the amplitude can result in a more natural-feeling haptic response.[8]

Duration and frequency require more complex handling. If the duration is very short (e.g., less than 20ms), the driver can trigger a pre-programmed "click" waveform from the DRV2605L's ROM library.[3] For longer vibrations, the driver enters RTP mode and maintains the intensity for the requested time. The frequency parameter can be used to select between different library effects or to adjust the LRA drive time as previously discussed.[11]

Haptic Request Type	Duration	Frequency Mapping	Actuation Mode
Instantaneous Click	< 10ms	High (> 100Hz)	Library Effect ID 1 (Strong Click) [3]
Soft Bump	10-50ms	Low (< 50Hz)	Library Effect ID 7

			(Soft Bump) [10]
Sustained Rumble	> 100ms	0-200Hz	Real-Time Playback (RTP) [2]
Variable Pulse	Variable	Dynamic	Waveform Sequencer (up to 8 slots) [3]

For sustained haptic events, the OpenVR driver implements a software timer. Upon receiving the haptic event, it immediately sends a "Start RTP" command to the SAMD21. After the requested duration has elapsed, it sends a "Stop" command. This approach minimizes the complexity of the embedded firmware while centralizing the timing logic on the more powerful host processor.[10]

Toolchain Configuration and Deployment Methodology

The deployment of a custom haptic stack involves cross-compilation and specific registry configurations on the host system. The SAMD21 firmware is compiled using cargo build --target thumbv6m-none-eabi --release.[5] The resulting ELF binary is converted to a UF2 or BIN format for flashing onto the QT Py via its native USB bootloader.[1]

The OpenVR driver DLL is built using the standard Windows Rust toolchain.[28] To deploy the driver, a specific folder structure must be created within the SteamVR directory or a dedicated external path.[24] This structure includes the driver manifest file (driver.vrdrivermanifest), which provides metadata such as the driver name and whether it should always be activated.[25]

The vrpathreg.exe utility, located in the SteamVR bin directory, is used to register the driver path with the OpenVR system.[29] Once registered, SteamVR will search the bin/win64 subdirectory of the driver folder for a DLL named driver_<name>.dll.[24]

Bash

```
# Example driver registration command
"C:\Program Files (x86)\Steam\steamapps\common\SteamVR\bin\win64\vrpathreg.exe" adddriver "C:\path\to\my_haptic_driver"
```

In addition to the DLL, the driver folder must contain an input profile JSON file.[24] This file

defines the haptic components and their localized names, allowing them to appear in the SteamVR controller binding UI.[25] Proper configuration of the manifest and input profiles ensures that the haptic device is recognized as a first-class citizen within the VR ecosystem, enabling users to rebind haptic actions across different applications.

System Resilience and Real-Time Performance Tuning

Maintaining high performance in a haptic stack requires careful attention to the concurrency models used on both the host and the device. On the SAMD21, the embassy framework's asynchronous executor is ideal for handling the high-frequency interrupts generated by the USB and I2C peripherals.[3] By using async tasks, the firmware can wait for an I2C transaction to complete without blocking the USB controller's ability to receive the next haptic packet.[3]

On the host side, the OpenVR driver must avoid blocking the main SteamVR thread. The communication with the USB HID device should occur on a dedicated background thread.[15] This ensures that any delays in the USB stack do not impact the frame rate of the VR compositor.[22]

System resilience is further enhanced through robust error handling. The I2C bus is prone to "sticky" states if a device is disconnected mid-transaction. The SAMD21 firmware should monitor the SERCOM status registers and implement an automatic bus reset if a timeout occurs.[2] Similarly, the OpenVR driver should implement a reconnection logic that periodically attempts to reopen the HID device if the connection is lost, allowing for hot-plugging of the QT Py during a VR session.[19]

Reliability Layer	Component	Strategy
Embedded Hardware	I2C Bus	SERCOM Timeout and Peripheral Reset [2]
Embedded Software	Main Loop	Hardware Watchdog Timer (WDT) [25]
Communication	USB HID	Packet Checksums and Retry Logic [19]
Host Software	Driver DLL	Non-blocking I/O and Hot-plug support [19]

By implementing these layers of resilience, the haptic stack provides a seamless experience for the user. The combination of Rust's safety guarantees and the precision of the SAMD21 and DRV2605L creates a haptic interface that is not only high-performing but also inherently

stable.

Synthesis and Conclusion

The architecture presented herein provides a comprehensive blueprint for bridging the gap between virtual reality software and physical haptic actuation. By utilizing the Adafruit QT Py SAMD21 as a high-speed intermediary and the DRV2605L as a precision driver, the system achieves a level of tactile fidelity that far exceeds generic vibration solutions. The implementation of this stack in Rust ensures that memory safety and real-time performance are prioritized at every level, from the Windows DLL to the bare-metal firmware.

The transition from abstract OpenVR haptic events to physical motor movements is mediated by a low-latency USB HID protocol and a sophisticated mapping logic that accounts for the non-linearities of human perception and the resonant characteristics of haptic actuators. As VR continues to evolve, the ability to provide nuanced and responsive tactile feedback will remain a cornerstone of true immersion. This stack serves as a modular and extensible foundation for future innovations in the field of wearable haptics and motion-integrated VR interfaces.

Works cited

1. Adafruit QT Py - SAMD21 Dev Board with STEMMA QT - 3DMakerWorld, Inc., accessed February 16, 2026, https://3dmakerworld.com/products/adafruit-qt-py-samd21-dev-board-with-stemma-qt
2. DRV2605 Haptic Driver for ERM and LRA With Built-In Library and Smart-Loop Architecture datasheet (Rev. E) - Texas Instruments, accessed February 16, 2026, https://www.ti.com/lit/ds/symlink/drv2605.pdf
3. embassy-drv2605l - crates.io: Rust Package Registry, accessed February 16, 2026, https://crates.io/crates/embassy-drv2605l
4. OpenVR bindings for rust. - GitHub, accessed February 16, 2026, https://github.com/rust-openvr/rust-openvr
5. Building - MicroRust, accessed February 16, 2026, https://droogmic.github.io/microrust/getting-started/01.00.BUILD.html
6. Adafruit QT Py - SAMD21 Dev Board with STEMMA QT, accessed February 16, 2026, https://www.adafruit.com/product/4600
7. Pinouts | Adafruit QT Py SAMD21 | Adafruit Learning System, accessed February 16, 2026, https://learn.adafruit.com/adafruit-qt-py/pinouts
8. Integrating the TI DRV2605L for Audio-to-Haptic Feedback in a Gaming Handheld, accessed February 16, 2026, https://www.ti.com/lit/pdf/sloa358
9. DRV2605L Haptic Drivers - TI - Mouser Electronics Belgium, accessed February 16, 2026, https://www.mouser.be/new/texas-instruments/ti-drv2605l-driver/
10. API Reference — Adafruit DRV2605 Library 1.0 documentation - CircuitPython, accessed February 16, 2026, https://docs.circuitpython.org/projects/drv2605/en/latest/api.html

11. DRV2605 Haptic Motor Driver - ESP32 - — ESPP main documentation, accessed February 16, 2026, https://esp-cpp.github.io/espp/haptics/drv2605.html
12. arm-none-eabi - The rustc book - Rust Documentation, accessed February 16, 2026, https://doc.rust-lang.org/rustc/platform-support/arm-none-eabi.html
13. embedded_hal - Rust - Docs.rs, accessed February 16, 2026, https://docs.rs/embedded-hal
14. Haptic Motor Driver Hook-Up Guide - SparkFun Learn, accessed February 16, 2026, https://learn.sparkfun.com/tutorials/haptic-motor-driver-hook-up-guide/all
15. USB: low latency (< 1ms) with interrupt transfer and raw HID - Stack Overflow, accessed February 16, 2026, https://stackoverflow.com/questions/27976992/usb-low-latency-1ms-with-interrupt-transfer-and-raw-hid
16. Serial vs raw HID vs something else? - Teensy Forum - PJRC, accessed February 16, 2026, https://forum.pjrc.com/index.php?threads/serial-vs-raw-hid-vs-something-else.18423/
17. USB HID vs CDC : r/embedded - Reddit, accessed February 16, 2026, https://www.reddit.com/r/embedded/comments/18lxad1/usb_hid_vs_cdc/
18. How bad is the latency of a serial port? : r/embedded - Reddit, accessed February 16, 2026, https://www.reddit.com/r/embedded/comments/1p33w1k/how_bad_is_the_latency_of_a_serial_port/
19. USB: what are the advantages (or disadvantages) or using HID over serial-over-USB?, accessed February 16, 2026, https://electronics.stackexchange.com/questions/21383/usb-what-are-the-advantages-or-disadvantages-or-using-hid-over-serial-over-us
20. Intro to Embedded Rust Part 3: USB Serial Logging and Debugging - DigiKey, accessed February 16, 2026, https://www.digikey.com/en/maker/tutorials/2026/intro-to-embedded-rust-part-3-usb-serial-logging-and-debugging
21. USB vs HID USB (USB-Serial) - EEVblog, accessed February 16, 2026, https://www.eevblog.com/forum/projects/usb-vs-hid-usb-(usb-serial)/
22. Concurrency Patterns in Embedded Rust - Ferrous Systems, accessed February 16, 2026, https://ferrous-systems.com/blog/embedded-concurrency-patterns/
23. Trouble understanding OpenVR driver API - Stack Overflow, accessed February 16, 2026, https://stackoverflow.com/questions/65280391/trouble-understanding-openvr-driver-api
24. Driver Documentation · ValveSoftware/openvr Wiki · GitHub, accessed February 16, 2026, https://github.com/ValveSoftware/openvr/wiki/Driver-Documentation
25. openvr/docs/Driver_API_Documentation.md at master - GitHub, accessed February 16, 2026, https://github.com/ValveSoftware/openvr/blob/master/docs/Driver_API_Documentation.md
26. OpenVR custom driver - How to implement the driver factory function :: SteamVR

Developer Hardware General Discussions - Steam Community, accessed February 16, 2026, https://steamcommunity.com/app/358720/discussions/0/3572855624919670 63/
27. Driver Factory Function · ValveSoftware/openvr Wiki - GitHub, accessed February 16, 2026, https://github.com/ValveSoftware/openvr/wiki/Driver-Factory-Function
28. DLL loader in Rust. Going down the rabbit hole with Rust... | by Dan ..., accessed February 16, 2026, https://medium.com/@dangroner/dlls-in-rust-e1322da511da
29. IVRDriverInput Overview · ValveSoftware/openvr Wiki · GitHub, accessed February 16, 2026, https://github.com/ValveSoftware/openvr/wiki/IVRDriverInput-Overview
30. Moondust/Assets/SteamVR/Input/SteamVR_Action_Vibration.cs at master - GitHub, accessed February 16, 2026, https://github.com/ValveSoftware/Moondust/blob/master/Assets/SteamVR/Input/SteamVR_Action_Vibration.cs
31. Best way to access DLL functions in Rust? - Rust Users Forum, accessed February 16, 2026, https://users.rust-lang.org/t/best-way-to-access-dll-functions-in-rust/14548
32. vr::ITrackedDeviceServerDriver Overview · ValveSoftware/openvr Wiki - GitHub, accessed February 16, 2026, https://github.com/ValveSoftware/openvr/wiki/vr::ITrackedDeviceServerDriver-Overview
33. How to implement a tracked virtual controller? · Issue #1468 · ValveSoftware/openvr, accessed February 16, 2026, https://github.com/ValveSoftware/openvr/issues/1468
34. bindgen - crates.io: Rust Package Registry, accessed February 16, 2026, https://crates.io/crates/bindgen
35. lucidscape/openvr-rs: Rust Bindings for the OpenVR Library - GitHub, accessed February 16, 2026, https://github.com/lucidscape/openvr-rs
36. steamvr-library · GitHub Topics, accessed February 16, 2026, https://github.com/topics/steamvr-library
37. Introduction - The bindgen User Guide, accessed February 16, 2026, https://rust-lang.github.io/rust-bindgen/
38. OpenVR driver - Antilatency, accessed February 16, 2026, https://developers.antilatency.com/Software/OpenVR_Driver_en.html
39. SteamVR Input · ValveSoftware/openvr Wiki - GitHub, accessed February 16, 2026, https://github.com/ValveSoftware/openvr/wiki/SteamVR-Input
40. Render to Overlay - SteamVR Developers - Steam Community, accessed February 16, 2026, https://steamcommunity.com/app/250820/discussions/7/2296220207955334714/?l=latam