

Hologram

Authors: Steven Guo, Kendric Terry, Andrew Yu

Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

Abstract—The goal of this project is to implement an interactive volumetric display using a swept-volume approach with rotating LED matrices. By combining a high-speed mechanical drivetrain, a distributed compute architecture split between an NVIDIA Jetson Orin Nano and a Raspberry Pi, and a real-time computer vision pipeline for hand gesture recognition, our system advances beyond existing passive volumetric displays to enable natural user interaction with rendered 3D content. The display targets a 128x64 resolution at 30 fps with under 200 ms of end-to-end latency, allowing users to manipulate 3D geometry through spatial hand movements without specialized eyewear.

Index Terms—Computer Vision, Gesture Recognition, HUB75 LED Matrix, Persistence of Vision, Raspberry Pi, Real-Time Rendering, Swept-Volume Display, Volumetric Display

1 INTRODUCTION

Traditional 2D screens rely on perspective tricks to simulate a sense of depth. A volumetric display illuminates physical points in 3D space, allowing multiple users to view a 3D object from different angles simultaneously without the need for specialized eyewear.

In the commercial sector, there have been successful industry-grade volumetric displays created through swept-volume displays, which rapidly rotate or oscillate a 2d physical screen. Companies like Voxon Photonics, for example, have developed highly precise, enclosed 3D renderers. However, these systems are prohibitively expensive, costing around \$6,000, and act primarily as a passive visualization tool. The display acts purely as an output monitor and requires a computer with proprietary software to control it.

On the hobbyist end, Persistence of Vision (POV) devices are widely prevalent [1]. These typically take the form of spinning LED fans or rotating cylindrical arrays that use human retinal persistence to form an image in the air. However, most of these solutions are computationally limited and can only cycle through pre-rendered content, making real-time interactivity impossible.

Both industry and hobbyist tiers share the limitation of treating volumetric technology as a passive medium. Our project addresses this gap by engineering a swept-volume display that utilizes real-time, natural user interaction. Like existing volumetric displays, our swept-volume will be implemented with a rotating LED matrix. Instead of just acting as a static monitor, our system couples a high-speed rotating LED matrix with a computer vision pipeline. By

utilizing a dedicated compute module to track hand gestures via a webcam, we can translate spatial human movements into real-time manipulation of our rendered content.

2 USE-CASE REQUIREMENTS

For a successful product with both visualization and interactive components, the system must satisfy physical limits for human perception as well as computational demands for real-time gesture tracking. Our target application is to render a generic 3D geometry from an obj file and allow user gestures to perform transformations on the geometry.

For resolution, our target is 128x64. This specific resolution has been used by existing projects like in figure 1 and seems visually detailed enough to have recognizable 3D geometry.

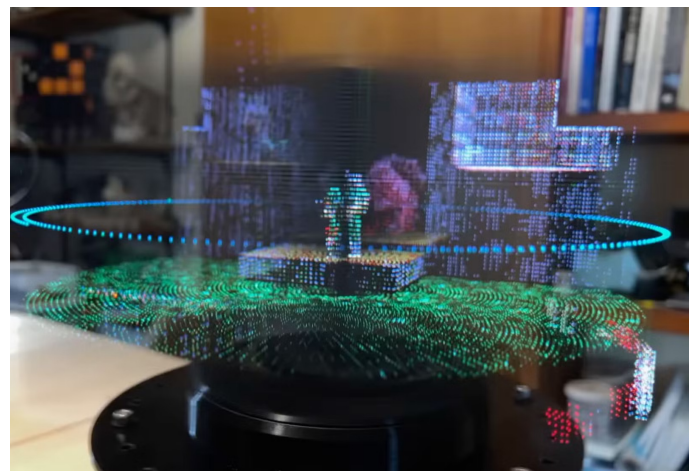


Figure 1: Existing volumetric display [2]

Our framerate target is 30 fps. This metric is based on POV [1], where the brain retains images for approximately $\frac{1}{30}$ of a second after viewing. This framerate would allow our rendered content to change and react smoothly.

To achieve the framerate target, our mechanism must spin at 900 rpm. This value is derived by calculating rotations necessary to display every angle 30 times per second. We can use half of the desired framerate, as we intend on having two displays back to back. Then:

$$\frac{30\text{fps}}{2} \cdot \frac{60\text{sec}}{1\text{min}} = 900\text{rpm} \quad (1)$$

From end to end, we aim for our system to have at most 200 ms of latency. This value is around the average human

reaction time, and we expect higher values to greatly decrease the level of immersion or perceived fluidity of the system.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

The central mechanism behind our volumetric display is a rotating LED matrix, which, when spun, generates a volume of pixels as shown in Figure 1. Thus, our design is partitioned into rotating and stationary components.

Figure 2 shows the main components necessary for operation. At a high-level, our goal is to rotate the LED panels (which also requires rotating a display controller for those LEDs). Then, for interactivity, we need stationary components for gesture control and content rendering. The system has 3 general subsystems to control the mechanical components for rotation, electrical systems for power, and software architecture for the actual data to pass around.

3.1 Mechanical Architecture

The mechanical subsystem is responsible for stably generating and transmitting rotational kinetic energy. In the block diagram, the main mechanical components are the electric speed controller (ESC) and the brushless motor.

The brushless motor is a pancake-style drone motor. Being brushless prevents the motor from wearing down over time, and since the motor is used for drones it has a high maximum speed.

There also will be necessary components not included in the block diagram like a chassis to house all components.

3.2 Electrical Architecture

The electrical architecture needs to route DC power to all components from a single AC source. Importantly, electricity needs to cross a rotating boundary, making power and signal constraints nontrivial.

A single 12V central power supply unit (PSU) is directly connected to stationary components like the ESC and Jetson Orin Nano. To prevent cable twisting, power must transfer to the rotating components through a slip ring. The slip ring can power the spinning Raspberry Pi and LED panels, but power must be stepped down to 5V for these parts.

3.3 Software Architecture

The flow of information through the system requires a distributed architecture, as the Pi which controls display logic is rotating, but we cannot capture any meaningful gesture data while rotating. Thus, the software is split across a compute node (a Jetson Orin Nano responsible for detecting gestures and rendering content) and the display controller (the Pi which sends data to LED panels).

Interaction begins with a USB webcam which is directly connected to the Jetson. The video stream is constantly processed so that hand landmarks can be extracted. From these landmarks, we can detect gestures based on relative finger positioning. This information needs to be translated into 3D transformation matrices in order to be applied on the 3D geometry we render in memory. The Jetson's main job is keeping this 3D data correctly updated in its memory.

To pass information from the Jetson to the Pi, the 3D geometry must be sampled with 2D slices to figure out what the LED panels would actually be displaying at any given time. Then, we can stream the 2D data over WiFi to the Pi.

For the Pi to communicate with the LED panels over HUB75 connectors, it needs some software to precisely control the timing of the output GPIO pins. There also needs to be synchronization, as the data received from the Jetson doesn't tell us when to display each slice. This is controlled using a hall effect sensor, which can give a reading every time the display rotates to a specific angle. These readings can be used to approximate angular velocity, which can then derive the rate at which to change between the 2D slices of data.

4 DESIGN REQUIREMENTS

Based on use case requirements described in section 2, we derived requirements for some of our components.

Our 128x64 resolution target dictates the rate at which our display controller needs to be able to output data. For example, to output data for two displays at 15 Hz for 360 degrees per rotation, we need to update each pixel with 24 bit color giving us a necessary data throughput of $2 \cdot (128 \cdot 64) \cdot 24\text{bits} \cdot 15\text{Hz} \cdot 360 = 2123366400 = 2.1\text{Gbps}$. Thus, our display controller needs to be able to output data very quickly, and it would be beneficial if we could parallelize data transfer to both LED displays to halve this requirement.

The framerate target of 30 fps is mainly bottlenecked by the data transfer rate between the Jetson and the Pi. This means our design needs to significantly optimize this connection. Possible solutions include wired data transfer through the slip ring or network compression techniques to speed up wireless transfer. Framerate also affects the necessary refresh rate of our LED displays. To achieve 30 fps with two displays, each display needs to update our entire volume at 15 Hz, and updating a volume requires 360 different images (for 360 degrees). In total, that means our display needs $15 \cdot 360 = 5400\text{Hz}$, making most higher quality displays insufficient.

Our physical rpm target of 900 determines our motor choice and drivetrain design. The rotating components sum up to a total of around 1 kg in weight (given two 400 g panels and the enclosure), so the drivetrain needs to be able to sustain rotation with this amount of mass. The slip ring also needs to be able to support rotation at this speed without degrading.

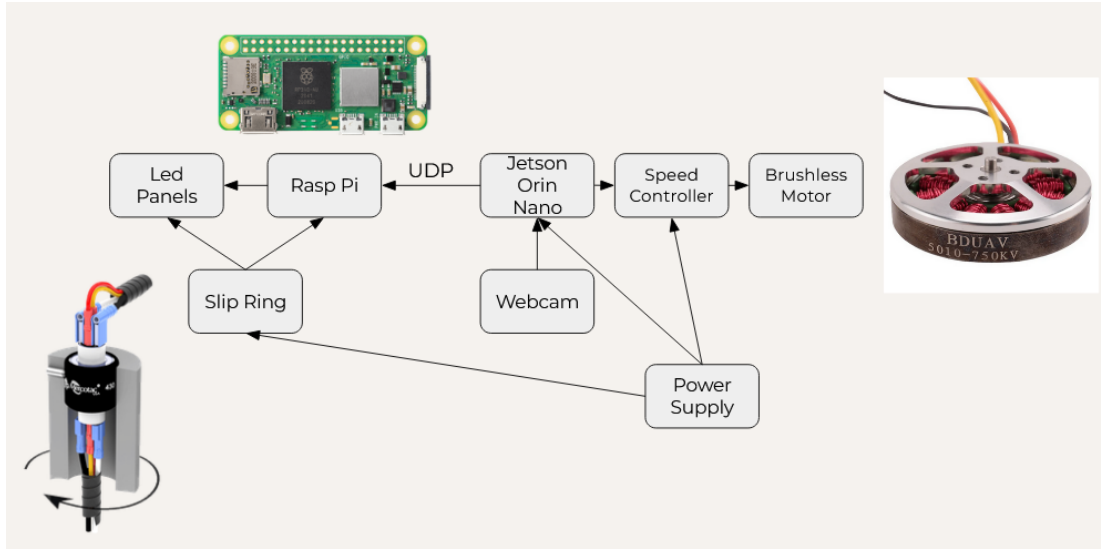


Figure 2: High-level Block Diagram

Finally, the design needs to optimize for latency to reach under 200 ms. To compute hand landmark detection, we will need a GPU. To minimize software execution speeds, we will need a low level language like C++ instead of Python for graphics computation. Networking will require using UDP instead of TCP for minimal latency as well.

Note that some of these design requirements do not directly quantify metrics that single components will need to meet. For example, it is not possible to source a motor that will immediately satisfy our requirement for 900 rpm with 1 kg of mass. Instead, we will elaborate on how specific design choices allow us to meet these requirements in section 5.

5 DESIGN TRADE STUDIES

To support component choice decisions, this section is divided into specific components and the reasoning behind why they were picked to meet some design or use-case requirement. For some decisions, tradeoffs between similar components were the main factor in the decision. Others required starting with specific options and working backwards to ensure they met design specifications.

5.1 Display

As described in section 4, our display needs a high refresh rate and a decent resolution to meet our target. With these constraints, the only options are LED display modules. We further filtered by scan rate, pixel density, and durability. The scan rate determines how many rows of the panel are lit at a time. For software simplicity when implementing the display controlling logic, it is important for this number to be a power of 2, so we chose a display with a scan rate of $\frac{1}{32}$. The pixel pitch, typically measured in mm, dictates the distance between neighboring pixels.

To have the highest density without drastically increasing cost, we chose a pixel pitch of 2.5 mm. Finally, our display was coated with a GOB (gel on board) surface, making it more durable to high speed winds.

5.2 Motor

For motor selection, the highest level of categorization was between brushed and brushless motors. Brushed motors rely on a physical connection and wear out more easily compared to brushless motors, so we choose the latter for robustness. We then filtered by speed, leading us to the general category of drone motors. We decided on a 5010 brushless motor. To confirm it would meet our requirements, we did quantitative analysis of torque and power consumption. In table 1, relevant variables and specifications for our chosen motor are presented.

Table 1: Variables and Motor Parameters

Symbol	Description	Units	Value
ρ	Air density at sea level	kg/m ³	1.225
C_d	Drag coefficient (flat plate)	–	1.2
h	Assembly height	m	0.16
R	Assembly radius	m	0.16
M	Assembly mass	kg	1.0
L	Total length ($2R$)	m	0.32
ω	Angular velocity	rad/s	94
t	Spin-up time	s	10
K_v	Motor velocity constant	RPM/V	750
K_t	Motor torque constant	Nm/A	0.0127

For analysis to be accurate, we also had to specify the drivetrain configuration. Given that we were using a drone motor, we knew the intended use case was a high speed, low torque application. Using our motor velocity constant, at 12 V our motor would aim for a target speed

of $K_v \cdot V = 750 \cdot 12 = 9000\text{RPM}$, and even with a lower efficiency ratio, this was too high for our system. We decided upon a pulley ratio of 8:1 to reduce this speed to our actual target for our display. Having pulleys in a belt drive also minimizes the risk of backlash breaking any components.

With our belt drive, the torque our system needs to produce needs to overcome the drag forces from the rectangular display moving around, as well as inertial forces from starting up the system. In (2) we show approximate drag forces by modeling our system as a rotating flat plate.

$$\tau_{\text{drag}} = \frac{1}{4} \rho C_d h \omega^2 R^4 N_{\text{sides}} \approx 0.68\text{Nm} \quad (2)$$

Beyond the primary drag forces from the spinning panel, we also have some inertial forces as we slowly spin our panels to target speed, shown in (3).

$$\tau_{\text{inertial}} = \left(\frac{1}{12} M L^2 \right) \frac{\omega}{t} \approx 0.08\text{Nm} \quad (3)$$

With these torque calculations we get a reasonable final torque total:

$$\tau_{\text{total}} = \tau_{\text{drag}} + \tau_{\text{inertial}} \approx 0.76\text{Nm} \quad (4)$$

This amount of torque would cause our motor to draw around 72 W of power, which is safe for our power supply. Based on this power consumption, the current drawn is also under 10 A (also within bounds of our system).

$$P = \left(\frac{\text{RPM}}{K_v} \right) \cdot \left(\frac{\tau_{\text{total}}}{K_t} \right) \approx 72\text{W} \quad (5)$$

Overall, our torque in (4) and power in (5) indicate reasonable numbers showing our motor choice is feasible.

5.3 Display Controller

Section 4 showed our need for our display controller to be able to output data quickly to both displays in parallel. Our original choice for the display driver was the Adafruit Matrix Portal S3, an ESP32 based development board with a single built-in HUB75 output. Unfortunately, the single output meant we would have to daisy chain the two displays, making data throughput a large constraint.

We ended up finding a bonnet for Raspberry Pi's that supported controlling up to 3 displays in parallel. However, because the Pi has a Unix operating system compared to RTOS on the ESP32, we initially thought there might be risk of image jitteriness. Fortunately, further research showed that previous implementations using Raspberry Pi's did not seem to suffer from this issue [2].

5.4 Slip Ring

Selecting the correct slip ring was a critical mechanical challenge, as this component is a necessary electrical interface for the rotating components to work at all. Most commercial slip rings operate by friction as metal contacts

stay in touch to pass through an electrical signal. Unfortunately, many of these options either fail to meet our current requirements or rpm requirements.

Liquid metal slip rings emerged as one of the only choices that solved both of these problems by operating without friction. We selected the model 230, which supports up to 250 V, 30 A, and 1800 rpm, making it a very safe choice.

5.5 Data Transfer

Establishing a reliable data link between the Jetson and Pi presented a pretty significant challenge. It was difficult to find a solution that had enough throughput as well as low enough latency. To limit latency, our initial approaches looked at wired data transfer options. The only way to support this was for our slip ring to support data transfer. Slip rings with Ethernet support exist, but cannot meet both data throughput and rpm requirements without becoming much more expensive than our budget.

With physical connections eliminated, contactless data transfer methods like an optical link were also evaluated. However, maintaining precise optical alignment and developing a prototype more robust than other wireless methods proved infeasible for the scope of this project.

Ultimately, a local WiFi network was decided as the only viable and cost-effective solution. To minimize latency we decided to use the UDP protocol and plan on constantly transmitting data without synchronization.

6 SYSTEM IMPLEMENTATION

This section contains concrete implementation details for the outlined architecture in section 3. Details are separated by our three subsystems below.

6.1 Mechanical Subsystem

As described in the architecture, the mechanical system includes the drivetrain for generating rotational energy and the chassis for containing all the components. Both of these components are shown in Figure 3. The drivetrain utilizes a belt drive to reduce backlash, while an 8:1 pulley ratio is used to increase the overall torque output. To ensure proper assembly and operation, the motor attaches to the chassis through oval holes, allowing the belt to be correctly tensioned before the motor is fastened tightly. Additionally, low-carbon steel was selected over stainless steel or aluminum for critical metallic components like the central shaft. Although all three materials block electrical interference, only low-carbon steel blocks magnetic interference as well.

The full wood enclosure is manufactured using laser cutting, which is much easier to scale compared to 3D printing. Because 6mm plywood is relatively thin, the chassis is built using a double-layered design. The outer layer utilizes T-slot joints so that structural forces rest on metal hardware

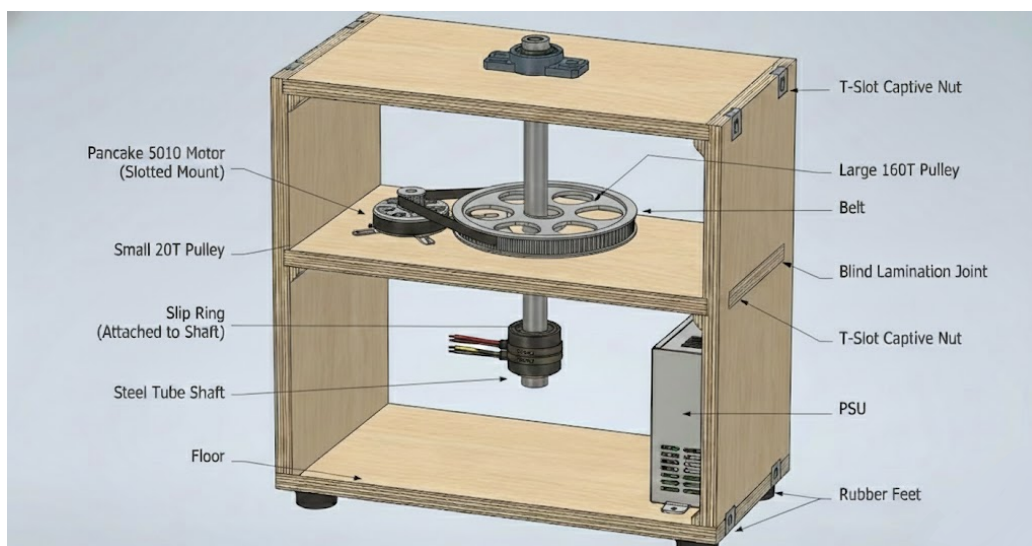


Figure 3: Mechanical Subsystem Diagram

rather than the wood. The inner layer is intentionally recessed, forming a blind lamination lap joint when glued to the outer layer for added rigidity. Structurally, the enclosure is divided into three distinct levels: a solid base, a middle shelf that houses a bearing and suspends the slip ring, and a roof that supports the upper bearing. Finally, rubber feet are installed on the base to increase friction and improve vibration resistance during high-speed rotation.

6.2 Electrical Subsystem

Figure 4 illustrates the distribution of power throughout the system. To convert 120V AC wall power to 12V DC, we selected a Mean Well LRS switching power supply capable of delivering 350W. Beginning at this 12V PSU, power flows directly to the stationary components: the 750 Kv motor and the Jetson Nano. As detailed in Figure 4, we expect the motor to consume a maximum of 100W to maintain our target output of 900 RPM, while the Jetson draws approximately 10W under normal operating conditions.

To transfer power to the rotating assembly, we utilize a Model 230 liquid metal slip ring. This slip ring is rated for up to 250V, 30A, and 1800 RPM, providing us with a comfortable safety margin for our operating conditions. The rotating assembly itself houses the Raspberry Pi and the two LED displays. Before passing through the slip ring, the 12V supply is stepped down to 5V using two buck converters to safely power these components.

We calculated that the two 5V LED displays will draw approximately 30W each, resulting in a current draw of 6A per panel. The Raspberry Pi also operates at 5V and draws an estimated 2W (0.4A). Therefore, the total current passing through the slip ring will be 12.4A, leaving a safety margin of over 50% against our 30A limit. Overall, the total power consumption of the entire system is estimated at 172W. This utilizes roughly half of our power supply's 350W capacity, comfortably ensuring design leeway for our

electrical system architecture.

6.3 Software Subsystem

Figure 5 shows the software packages needed on both the Jetson and Pi. The two devices are largely decoupled and only interface via data passed over the network, so this subsystem is further divided.

6.3.1 Jetson Orin Nano Software

The Jetson contains the vast majority of the software and has the more complex architecture. The overall system begins with the computer vision pipeline which was implemented in Python for library support. In Python, OpenCV is used to constantly read from the usb webcam stream before using the Mediapipe package to extract hand landmarks. The dependencies from these libraries introduce a bit of complexity as specific versions are required to get them working smoothly on the GPU and on the specific Linux flavor running on the Jetson. To solve this, we plan on deploying a Docker container with standardized dependencies based on existing images from the jetson-containers repository [3].

Since the rest of the C++ graphics pipeline does not require Docker, we will spawn a thread for the computer vision component to run in parallel. Inter-process communication between the hand detection and graphics will occur via shared memory. To synchronize between the process, a seqlock will be used to ensure data is consistent.

After spawning the computer vision pipeline, pure C++ is used for the gesture translation, 3D engine, slicer, and networking components in figure 5.

Gesture translation occurs with preset transformations that can be performed on our 3D object. We will map specific detected gestures like rightward rotation to the 3D

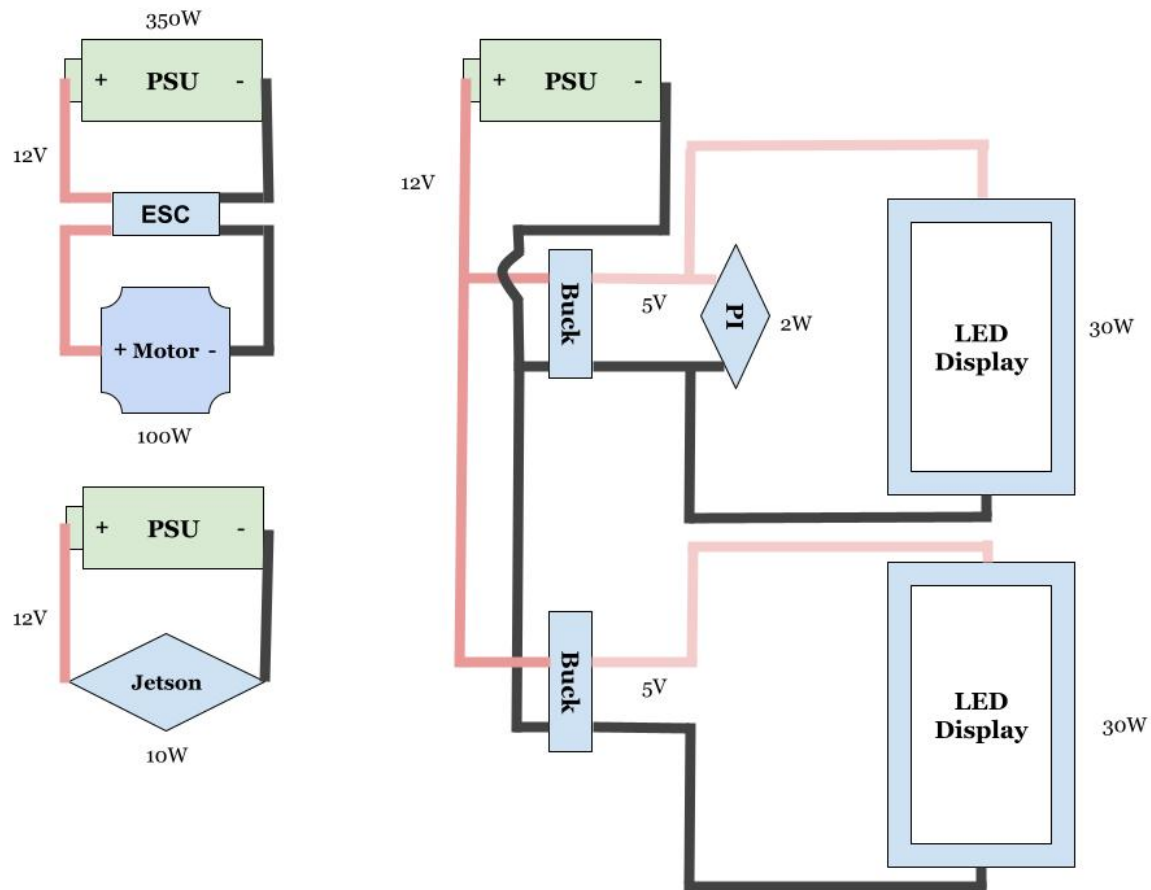


Figure 4: Electrical Subsystem Diagram

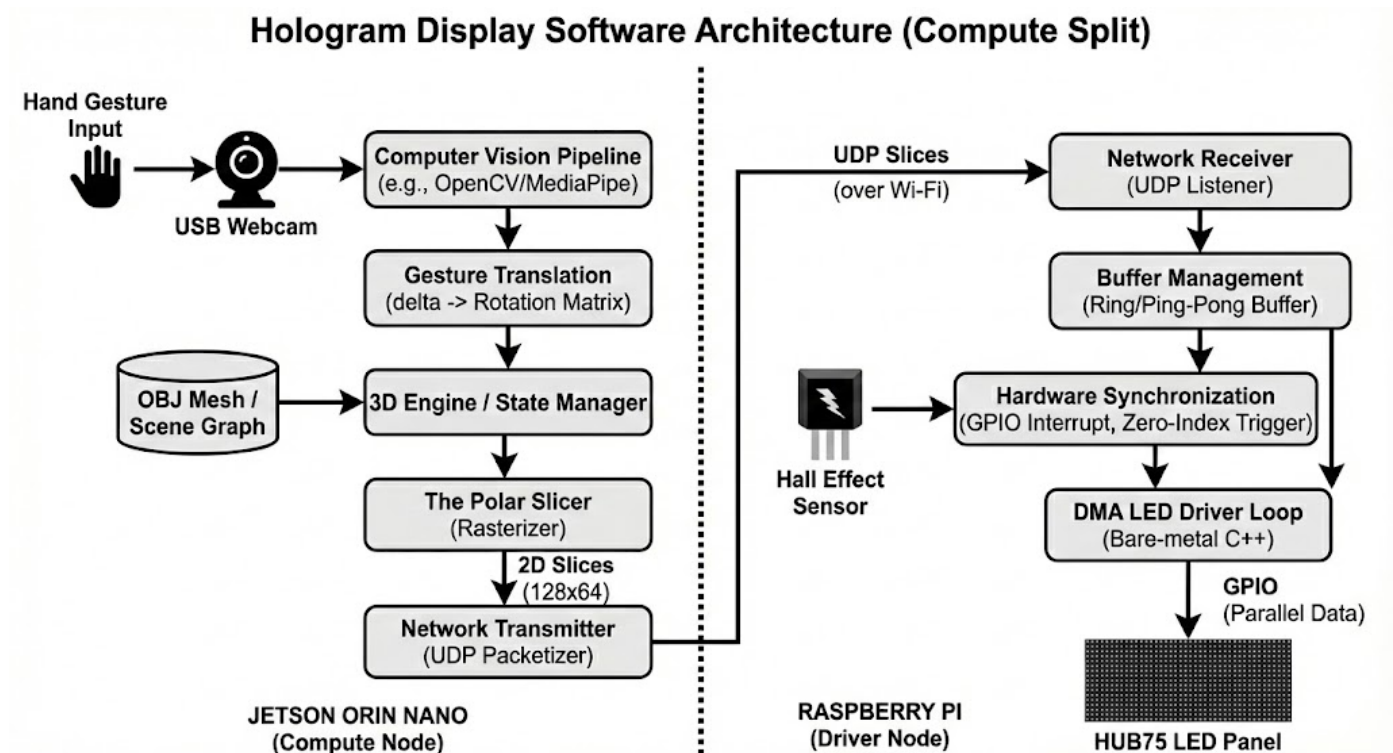


Figure 5: Software Subsystem Diagram

transformation matrix that corresponds to the mathematical operation we intend.

This matrix is multiplied onto the 3D object we've stored in memory. The management of this 3D geometry depends on the OpenGL library, which helps with rendering the object from an OBJ file initially. The OBJ format was chosen due to ease of parsing.

With the 3D geometry updated, the polar slicer component is responsible for sampling 2D slices that we would display on our LED panels. To increase runtime performance, we will precompute the mapping between specific pixels and how they would appear on 2D slices.

6.3.2 Networking Software

For network transmission to the Pi, the Jetson hosts its own access point to minimize interference from other devices. The Pi can connect to this network to receive data.

UDP packets are sent in a compressed format to minimize required data bandwidth. Our compression protocol relies mainly on RLE (run-length encoding). This encoding greatly reduces data when many consecutive pixels are the same, which is optimal for a use case of displaying wireframe 3D geometry with a lot of empty space.

To update as quickly as possible, data is sent constantly with metadata for the current version of the volume as well as the frame within the volume. The metadata allows the Pi to correctly construct the order in which to display frames and to identify stale data.

6.3.3 Raspberry Pi Software

Once the Raspberry Pi receives data, its job is to output signals to the LEDs as quickly as possible. Starting from the operating system, we install Raspberry Pi OS Lite to minimize any bloat. We also disable unnecessary components like the audio and video drivers.

Software is all implemented in C++ and includes a few main components as shown in figure 5. When receiving data, we use a double buffer to decouple the logic of reading frames over the network from writing frames to the LEDs. In this way, we can prevent possible race conditions like trying to display partially updated data.

Using the buffer with most recent complete data, the Pi figures out when to display frames based on angular velocity approximation derived from hall effect sensor readings. Finally, we use the rpi-rgb-led-matrix library [4] to efficiently write to the LED displays.

7 TEST & VALIDATION

To ensure we have met the use-case requirements of our project, we plan to extensively test each subsystem individually before moving into the final integration phase.

7.1 Tests for Design Specification: Frame Rate

- **Goal:** Achieve a perceived framerate of 30 fps from any 360-degree viewing angle.
- **Methodology:** Since our design utilizes two back-to-back LED panels, each panel must successfully update the entire volume at 15 Hz. We will implement a logging system directly on the Raspberry Pi to record timestamps. We will measure the time it takes the driver node to cycle through the 360 distinct 2D slices required for a full volumetric rotation.
- **Success Criteria:** The logs confirm the Raspberry Pi consistently outputs full volume updates at 15 Hz per panel without dropping frames or stalling.

7.2 Tests for Design Specification: Latency

- **Goal:** Maintain an end-to-end system latency of ≤ 200 ms to preserve user immersion and fluidity.
- **Methodology:** We will track a single interaction through the entire pipeline. We will log a timestamp on the Jetson Orin Nano the exact moment a specific hand gesture is parsed by the computer vision pipeline. A corresponding timestamp will be logged on the Raspberry Pi the moment the resulting transformed 2D slice is pushed via GPIO to the LED panels.
- **Success Criteria:** The time delta between the Jetson's initial gesture capture and the Pi's display output remains strictly under the 200 ms threshold.

7.3 Tests for Design Specification: Visual Stability

- **Goal:** Maintain a stable physical rotation with a maximum vertical tilt deviation of ≤ 2 degrees.
- **Methodology:** We will render a static, perfectly vertical straight line on the display. While the system operates at the target 900 rpm, we will use a camera with a high shutter speed to capture the rendered volume alongside a stationary physical reference tool (such as a framing square or plumb bob) placed directly behind the enclosure.
- **Success Criteria:** Image analysis of the captured frames confirms that any angular deviations in the rendered line are 2 degrees or less from the true vertical.

7.4 Tests for Design Specification: Power Draw

- **Goal:** Keep total system power consumption below 300 W, ensuring a comfortable safety margin for our 350 W Mean Well PSU.
- **Methodology:** Before full integration, we will use a multimeter to measure the isolated voltage and cur-

rent draw of individual components to verify they match our theoretical bounds: ~ 100 W for the motor, ~ 10 W for the Jetson, ~ 2 W for the Raspberry Pi, and ~ 30 W per LED panel. After verifying the subsystems, we will measure the aggregate draw at the PSU under maximum load conditions (motor spinning at 900 rpm with all LEDs displaying full white).

- **Success Criteria:** The isolated components sit within their theoretical bounds, and the peak integrated power draw remains strictly under 300 W.

8 PROJECT MANAGEMENT

8.1 Schedule

The detailed project schedule and major milestones are illustrated in the Gantt chart shown in figure 6. The timeline spans approximately twelve weeks, from early February to late April, and is structured to allow parallel development across the mechanical, electrical, and software domains before converging into integration phases. The university spring break is also scheduled as well to make work goals realistic.

8.2 Team Member Responsibilities

Since our system primarily divides into three categories, we chose to divide responsibilities based on these domains.

Kendric is mainly responsible for the physical architecture and drivetrain for the display. His core tasks include CAD for our custom pulleys and laser cut components.

Steven is responsible for electrical integration. This involves designing the power distribution system. Primary responsibilities include performing tests for the motor and LED display. He also has some secondary responsibility for implementing display driving logic on the Raspberry Pi.

Andrew is responsible for the software, which includes development of the computer vision pipeline, graphics rendering, networking logic, and display output. His primary goal is to implement all the application level logic.

All team members hold responsibility for integration of components. This includes physical assembly and end-to-end integration tests.

8.3 Bill of Materials and Budget

A complete breakdown of all purchased components, including descriptions, model numbers, manufacturers, quantities, and costs, is provided in Table 2. The total project cost is \$480.48, which is well within the course budget. Note that the Jetson Orin Nano and LED panels were borrowed and already owned, so they have been omitted from the total.

8.4 TechSpark Usage Plans

We plan to use TechSpark primarily for laser cutting the plywood panels that make up our chassis enclosure. This

includes cutting the double-layered walls with T-slot joints, the base, middle shelf, and roof components. We may also use TechSpark for general assembly and integration work as we bring the mechanical, electrical, and rotating subsystems together.

8.5 Risk Mitigation Plans

The following are the key risks we have identified and our plans to address them:

- **Mechanical failure at high RPM:** A structural failure while spinning at 900 rpm poses a serious safety hazard. To mitigate this, we are using a GOB-coated LED panel for durability, threadlocker on all rotating fasteners, and will perform incremental speed-up tests to check for vibration or wobble before reaching full speed. The wooden enclosure also acts as a protective barrier.
- **WiFi data throughput and latency:** No team member has previously implemented a high-throughput, low-latency UDP streaming pipeline between embedded devices. If raw frame data exceeds WiFi bandwidth, our RLE compression scheme may not be sufficient. As a fallback, we can reduce the angular resolution (fewer slices per rotation) or reduce color depth to lower bandwidth requirements.
- **Slip ring reliability:** The liquid metal slip ring is a single point of failure for powering all rotating components. If it degrades or introduces noise, the entire display fails. We selected a model rated well above our operating conditions (1800 rpm vs. 900 rpm, 30A vs. 12.4A) to provide a large safety margin, and we will monitor for any signs of electrical noise during testing.
- **Display synchronization:** Correctly timing which 2D slice to display based on hall effect sensor readings requires accurate angular velocity estimation. Errors could produce visible artifacts or ghosting. We plan to test synchronization early using simple test patterns before moving to full 3D content, and will implement interpolation logic to smooth over any sensor inconsistencies.
- **Computer vision pipeline latency:** Running Mediapipe hand detection on the Jetson within our latency budget is uncertain, as GPU acceleration depends on specific library versions and Docker configuration. We will benchmark the CV pipeline in isolation early and, if needed, reduce input resolution or detection frequency to stay within our 200 ms end-to-end target.

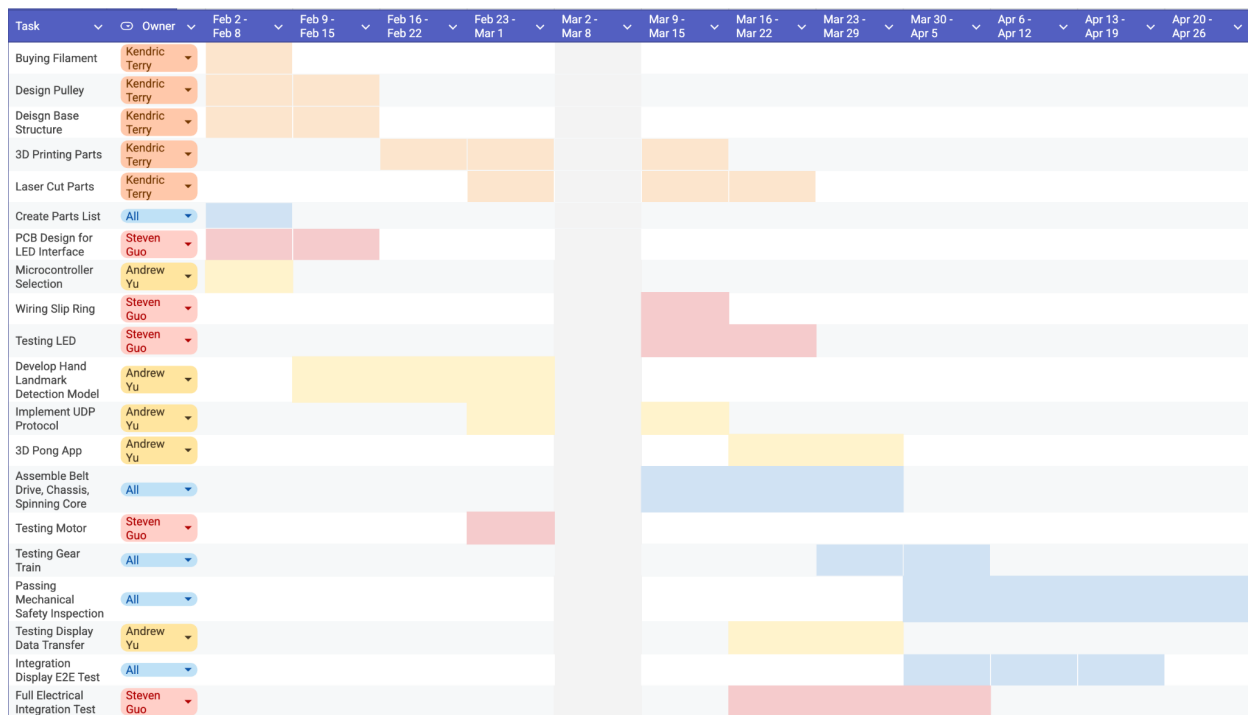


Figure 6: Gantt Chart

Table 2: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
5010 750KV Motor	TOQ3219951199275BR	GoolRC	1	\$27.89	\$27.89
ESC	43234-386		1	\$17.99	\$17.99
Servo Tester	124	Globact	1	\$12.99	\$12.99
Bearings	UCF204	uxcell	2	\$16.39	\$32.78
1 ft low carbon tube	6045N13	McMaster-Carr	1	\$23.05	\$23.05
Flange coupling	B0CPF7YYGM	YC-Yier	1	\$23.99	\$23.99
Timing Belt	7947K748	McMaster-Carr	1	\$8.92	\$8.92
Screw Kit	B0D3X5CT2J	Hapric	1	\$8.99	\$8.99
Threadlocker	B000I1RSNS	Loctite	1	\$5.97	\$5.97
Jetson Orin Nano		NVIDIA	1		
SSD	SP128GBP34A60M28	Silicon Power	1	\$42.97	\$42.97
Hall effect sensor	3-01-1193	HiLetgo	1	\$5.99	\$5.99
Triple HUB75 Bonnet	6358	Adafruit	1	\$9.95	\$9.95
Raspberry Pi Zero 2 W	Zero	Raspberry Pi	1	\$23.90	\$23.90
MicroSD	SDSQQNR-064G-GN6IA	SanDisk	1	\$19.99	\$19.99
Heatsink	B09QMBCXLB	Geekworm	1	\$7.00	\$7.00
2x20 40 Pin Stacking Header	4079	Adafruit	1	\$1.95	\$1.95
PSU	lrs-350-12	Mean Well	1	\$33.00	\$33.00
AC Power Cord	B0FNML9SRY	Vockjour	1	\$5.39	\$5.39
Socket	0712317301076	3Dman	1	\$7.99	\$7.99
Slip Ring	230	Mercotac	1	\$90.79	\$90.79
XT60 Female	B07BF8154S	Padarsey	1	\$6.99	\$6.99
5.5mm x 2.5mm DC Male Pigtail	GINTOOYUN-DC5525-90-3FT	GINTOOYUN	1	\$9.99	\$9.99
Fork Spade Terminal Connectors	SVS1.25-3.5	Baomain	1	\$6.59	\$6.59
Lever Nuts	221-2401	Peppy Products	1	\$15.45	\$15.45
Buck converter	W-08-13	WWZMDiB	2	\$9.99	\$19.98
VH 3.96mm Pitch 4 Pin	PZSCC005	pzsmocn	1	\$9.99	\$9.99
PETG-CF	06938936714606	Polymaker	1	\$19.99	\$19.99
					\$480.48

9 RELATED WORK

While commercial volumetric displays exist for specialized applications, the foundational proof-of-concept for our physical architecture draws heavily from the hobbyist engineering community, specifically the "Vortex" display developed by independent maker Ancient James [2]. The Vortex project successfully demonstrated the mechanical viability of a high-resolution persistence of vision (POV) envelope using commercial-off-the-shelf components. By utilizing back-to-back HUB75 LED matrices, a central rotating shaft, a high-speed slip ring, and a brushless motor, Vortex validated that heavy LED panels can be safely rotated at high speeds.

However, existing implementations like Vortex typically rely on tightly coupled, bare-metal microcontrollers mounted directly on the rotating core. Lacking the memory bandwidth and processing power for real-time 3D rendering, these displays operate strictly as passive kinetic sculptures, limited to mathematically generated patterns or pre-compiled voxel animations. Our capstone project builds upon this mechanical foundation but fundamentally shifts the software paradigm by utilizing a distributed, high-performance compute architecture. By decoupling the physical display driving (Raspberry Pi) from the spatial processing (NVIDIA Jetson Orin Nano), we introduce real-time computer vision into the rendering loop. This enables continuous hand landmark detection and the low-latency UDP data streaming necessary to run an interactive application like 3D Pong, advancing the swept-volume concept from a passive observation medium into a responsive human-computer interface.

10 SUMMARY

This report presented the design of an interactive swept-volume volumetric display that combines mechanical, electrical, and software subsystems into a cohesive system. The mechanical subsystem uses a brushless drone motor with an 8:1 belt-driven pulley ratio to rotate two back-to-back HUB75 LED panels at 900 rpm, housed in a laser-cut plywood enclosure. The electrical subsystem distributes power from a 350 W PSU through a liquid metal slip ring to both stationary and rotating components, with a total estimated draw of 172 W. The software subsystem employs a distributed architecture: an NVIDIA Jetson Orin Nano runs the computer vision and 3D rendering pipeline, while a Raspberry Pi on the rotating assembly drives the LED panels using angular position data from a hall effect sensor. Data is streamed between the two over a local WiFi link using UDP with run-length encoding compression.

Quantitative analysis of the motor torque, power consumption, and electrical margins confirms the feasibility of our component selections. Our validation plan includes targeted tests for framerate, latency, visual stability, and power draw to verify that the system meets its use-case requirements of 30 fps, sub-200 ms latency, and stable ro-

tation. The project builds on prior hobbyist swept-volume displays by introducing real-time gesture-based interactivity, shifting the volumetric display paradigm from passive visualization to a responsive human-computer interface.

Glossary of Acronyms

- 2D - 2 Dimensional
- 3D - 3 Dimensional
- POV - Persistence of Vision
- fps - Frames per Second
- rpm - Revolutions per Minute

References

- [1] Sydney Butler. *Those 3D Sci-Fi Displays Are Finally Real (And They Run Doom)*. How-To Geek. Sept. 10, 2024. URL: <https://www.howtogeek.com/those-3d-sci-fi-displays-are-finally-real-and-they-run-doom/> (visited on 02/27/2025).
- [2] Dustin Franklin. *Jetson Containers: Machine Learning Containers for Jetson and JetPack*. Version master branch. 2025. URL: <https://github.com/dusty-nv/jetson-containers>.
- [3] Wikipedia contributors. *Persistence of vision* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 27-February-2025]. 2024. URL: https://en.wikipedia.org/wiki/Persistence_of_vision.
- [4] Henner Zeller. *rpi-rgb-led-matrix: Controlling RGB LED displays with Raspberry Pi GPIO*. Version master branch. 2025. URL: <https://github.com/hzeller/rpi-rgb-led-matrix>.