RICE

RICE UNIVERSITY

MECH 408: END-OF-SEMESTER DESIGN REPORT

SPRING 2025

TUESDAY 6TH MAY 2025

TEAM NAME:

AutoArts

TEAM MEMBERS:

Abdullah Alsinan

Aidan Colon

Saba Feleke

Yurie Han

Jorge Hinojosa

Alice Owens

Zach Woodley

Maaz Zuberi

# Table Of Contents

# Project Summary

As a team of Rice Engineering students, we aim to blend technical skills with creative design by building an innovative art car that showcases the intersection of engineering and art. Our project will engage with and contribute to the Houston community by participating in the Houston Art Car Parade, the city's largest public event, with an estimated 250 art cars and over 250,000 spectators.

# Problem Statement

Our project will elevate the level of art car engineering by merging aesthetics with functionality, safety, and cutting-edge technology, offering an interactive experience that demonstrates both our technical and artistic skills.

## *Overview*

We are designing and constructing an art car for the Orange Show's 38th Annual Houston Art Car Parade[1], scheduled for April 12, 2025. As the largest free event in Houston, with an audience of over 250,000 annually, the parade is the world's premier celebration of art cars. Our goal is to create a unique and unconventional vehicle that reflects Rice University's spirit of innovation, captivating the audience with an engaging blend of artistic expression and technological design.

The Art Car Parade, described by the City of Houston as a celebration of the inherent creativity within everyone, provides a platform to showcase artistic and engineering ingenuity on a grand scale. Our multidisciplinary team of artists, mechanical engineers, and electrical engineers aims to build an interactive experience that is both visually striking and technologically immersive, enhancing the parade's vibrant atmosphere.

## *Historical Context*

The Houston Art Car Parade traces its origins to 1984[2], when the Orange Show Foundation auctioned a 1967 Ford Station Wagon to artist Jackie Harris, who transformed it into the iconic "Fruit Mobile." Its popularity led to its reacquisition by the community and eventual return to the Orange Show Foundation, sparking the rise of art cars across Houston. In 1986, the New Music Parade, organized by Rachel Hecker and Trish Herrera during the New Music

[1] https://www.thehoustonartcarparade.com/
[2] https://www.thehoustonartcarparade.com/history-of-the-houston-art-car-para

America Festival, combined art cars and floats, attracting growing participation and audiences. By the 1990s, the Houston Art Car Parade had officially emerged as an annual event.

***Figure 1.*** *"First" Art Car: Fruit Mobile (1984)*
*Artist Jackie Harris who transformed the car into the "Fruit Mobile", includes sculptures of fruits and a painting exterior, along with small grill interventions to simulate fruits coming out of the grill.*

Both the Orange Show and the Art Car Parade represent Houston's unique folk art tradition, celebrating creativity that exists within communities and often outside traditional galleries and museums. Art cars embody this spirit by welcoming artists of all skill levels—from beginners to seasoned professionals—into a shared celebration of artistic exploration and community engagement.

The current state of art car design often lacks advanced interactive and dynamic elements, leaving room for innovation in integrating kinetic art and advanced technology. In previous years of the Houston Art Car Parade, many winning designs were flashy, colorful, and extravagant but predominantly featured static sculptures with limited use of interactive features. This highlights a clear opportunity for us to differentiate our car by introducing advanced engineering skills to interact with the audience and environment. Our team plans to push the boundaries by designing a beautiful, functional kinetic sculpture mounted on the vehicle and enhanced by laser displays and music synchronization, creating a cohesive, artistic presentation. This combination of

elements will set our car apart by actively engaging spectators and highlighting engineering as an art form in itself.

## *Prior Art*

Over 250 cars typically participate in the Houston Art Car Parade, presenting a variety of styles, from DIY modifications to professionally engineered vehicles. Competitive designs range from static sculptures to cars with integrated kinetic elements or light displays, but very few blend advanced technology with artistic creativity at the level we propose. Innovation will come from combining aesthetic beauty, dynamic motion, and cutting-edge technology to complete a vehicle that is competitively placed among its contenders.

***Figure 2.*** *2022 1st Place Art Car*
*"Rising Strong," exhibiting a stationary sculpture and phoenix-themed paint job*

***Figure 3.*** *2023 1st Place Art Car*
*"Gretapocalypse," An environmental theme , the design having smaller decorations and a double decker built on top of the truck, along with a trailer, decorated with a "Mad Max" apocalyptic aesthetic*

# Problem Description

Our primary customer is the Orange Show[3], the organizer of the Houston Art Car Parade. Particularly, our vehicle's unique design must not only entertain spectators but also meet strict safety and performance standards in compliance with the parade's regulations. Additionally, we aim to impress both the judges and spectators with a creative and technically advanced design that reflects our engineering skills. As a team, we also want to challenge ourselves and take full advantage of the creative freedom to build something tangible that showcases our abilities and stretches our imaginations.

Other requirements for this project include those noted in a user agreement with Rice Housing & Dining describing our team's terms of use for the E-VAN, the vehicle we intend to decorate for the parade. Lastly, we also consider the team's specific artistic goals and individual needs for our art car as requirements to meet.

## *Customer Requirements*

The Orange Show's Houston Art Car Parade has specific regulations to ensure safety and enjoyment for all participants. All entries must be decorated, functional, and family-friendly. Vehicles must meet height restrictions (under 13') and have proper registration, insurance, and a licensed driver. A fire extinguisher (minimum 10BC rating) is required in motorized vehicles. Parade vehicles must be able to idle in the sun for extended periods and are prohibited from launching or throwing objects. Any wheels must have rubber (no metal in contact with the street), and safety measures, like harnesses, are mandatory for individuals standing on vehicles. Open container laws are strictly enforced, and drinking and driving is illegal. Non-street-legal entries may be towed or trailered to the event. Prior approval is required for pyrotechnics or fire effects.

### Summary of Key Vehicle and Driving Regulations

- Entries must be decorated, functional, and under 13' tall.
- Licensed drivers are required for motorized vehicles; proper registration and insurance are mandatory.
- Vehicles need a 10BC-rated fire extinguisher.
- No launching or throwing objects from entries.
- Wheels must be rubber; no metal-to-street contact allowed.
- Drinking and driving, as well as open containers, are prohibited.
- Non-street-legal entries may be trailered to the event.

[3] https://orangeshow.org/

- Fire/pyrotechnics require prior approval.
- Vehicles must idle for long periods in the sun during the slow-moving parade.

The **E-VAN User Agreement** outlines the conditions for the Art Car Parade Group's use of the E-VAN van, owned by Housing and Dining (H&D), for their art car parade project. The agreement ensures responsible usage, establishes liability, and defines key procedures to protect both parties and the vehicle. The E-VAN may only be used for project-related activities and must be returned in its original condition, barring normal wear and tear.

## Summary of E-VAN User Agreement

- **Authorization**: H&D approves the E-VAN's use solely for art car parade project purposes.
- **Key Check-Out**: A designated point of contact manages keys; all users must sign a key log.
- **Use Conditions**: The E-VAN must be returned in good condition and may not be used for personal activities.
- **Insurance & Liability**: Users must provide proof of insurance or driver's license details; H&D is not liable for injuries, damages, or losses.
- **Duration**: Usage is limited to specified dates, with extensions requiring written approval.
- **Termination**: H&D may terminate the agreement for breaches or misuse.
- **Signatures**: Both H&D and the Art Car Parade Group must sign to confirm their agreement to these terms.

## Summary of Artistic Concept

The artistic concept for our art car is "Engineering Eras: An Art Car Through Time" blending the ingenuity of the past with the vision of the future. The design divides the van into two distinct thematic halves: the front embraces a **steampunk** aesthetic, while the back explores the high-tech world of **cyberpunk**.

The **steampunk-themed front half** will showcase Victorian-era engineering with ornate panels, gears, and cogs. It will incorporate self-contained machines demonstrating fundamental engineering principles, alongside 3D-printed designs and kinetic sculptures.

The **cyberpunk-themed back half** will feature sleek, futuristic panels with sharp lines and interactive LED displays that respond to crowd noise. These panels will project dynamic designs, creating an immersive and engaging experience. Additionally, the back will host a DJ, *Maaz*, along with a speaker and microphone setup to energize the crowd and amplify the celebration.

## *Engineering Requirements*

Customer requirements for this project emphasize both functional performance and an engaging user experience. The structural integrity of the system is paramount, as the frame must support all kinetic sculptures and technological components without failure. Power systems need to be robust enough to drive motors, displays, and audio systems reliably. Control systems must synchronize the movement of kinetic sculptures with lighting and sound, ensuring cohesive and dynamic operation. Interactivity is a key element, requiring the system to respond to audience inputs or environmental factors such as light and sound, creating an immersive and engaging experience.

### Operating temperature

To ensure our art car performs reliably under high temperatures and variable weather conditions, we have carefully selected materials, adhesives, and electronic components suited for extreme environments. Research indicates that vehicles exposed to direct sunlight in Texas can reach interior temperatures exceeding 120°F, with surface components like dashboards reaching up to 200°F. These findings, supported by Pediatrics[4] and the National Weather Service[5], emphasize the importance of heat-resistant materials and designs.

The kinetic sculptures will use either 6061-T6 aluminum or 316 stainless steel. Aluminum offers lightweight and corrosion-resistant properties but has lower fatigue resistance under cyclic loads, which could affect durability. In contrast, 316 stainless steel provides superior strength, fatigue resistance, and excellent protection against corrosion in humid or rainy conditions. For bonding, we propose using J-B Weld Cold Weld Epoxy[6], which can withstand temperatures up to 550°F and offers a tensile strength of 5,020 psi, ensuring reliable performance under both mechanical and thermal stresses.

The panels on the van's exterior will be fabricated from acrylic, chosen for its durability, lightweight properties, and ability to maintain its structural integrity at temperatures up to 180°F[7]. Acrylic is also easy to cut and shape, allowing for intricate designs consistent with our steampunk and cyberpunk themes.

For the electronics, we are incorporating RGB LED Matrix Panels from Adafruit[8], which will feature interactive lighting. These panels are capable of operating in high-temperature

---

[4] McLaren C, Null J, Quinn J. Heat stress from enclosed vehicles: moderate ambient temperatures cause significant temperature rise in enclosed vehicles. Pediatrics. 2005 Jul;116(1):e109-12. doi: 10.1542/peds.2004-2368. PMID: 15995010.

[5] https://www.weather.gov/lsx/excessiveheat-automobiles

[6]https://www.jbweld.com/product/j-b-weld-twin-tube?srsltid=AfmBOorlPEUcofsqaILZbWANcZdXEsXecklYGzisEtBGQcxmQsJbIbSC

[7]https://www.acrylite.co/resources/knowledge-base/article/what-are-the-recommended-continuous-service-temperatures-for-acrylite-acrylic-sheet?category=working-with-acrylite-r

[8] https://www.adafruit.com/product/1484

environments, but additional cooling measures, such as ventilation or heat sinks, may be necessary to prevent overheating during prolonged use. Careful wiring and insulation will also ensure safe and efficient operation in the event's challenging conditions.

By selecting robust materials, adhesives, and electronic components, and addressing environmental factors such as heat and humidity, we can ensure the functionality, safety, and visual impact of our art car throughout the competition.

### Waterproofing

To address the possibility that inclement weather may be present during the parade or any of the other events, we looked toward the international standard IEC 60529[9], which classifies the degrees of protection by enclosures, such as the protection of enclosures of electronic elements from water. This standard identifies quantitative ratings classifying degrees of waterproofing, which were used to inform the need to waterproof elements of our entry that need to be protected from water. As a requirement for the exterior elements of the design, we aim for an IPX-4 rating, which is protection from splashing water at any angle.

### Graphics performance

The LED graphics display is a critical component, with specific requirements for brightness and resolution to ensure clear visuals in outdoor environments. Meeting a minimum refresh rate of at least 30 frames per second is essential for maintaining high-quality visuals that are vibrant and perceived as compelling, continuous motion.

The platform for animating graphics must provide flexibility and versatility. The LED matrix displays must be compatible with open-source solutions that offer comprehensive documentation, a large repository of community examples, and seamless integration with Raspberry Pi systems. These criteria ensure ease of development and robust functionality while supporting tools for video and sound interaction. Essential engineering requirements include structural integrity, power system reliability, control system synchronization, and LED display performance.

### Power capacity

Our art car has various different interactive elements that will require to be powered. Our team came up with power estimates for the various components we are considering adding to our design.

- Adafruit P6 32x32 Panels (15) ~ **300W**
    - Each panel can take up to 5V, 4A of power, so 15 panels is 60 amps.

[9] https://www.wewontech.com/180619001.pdf

    - Panels will be powered by 300W 60A, 5V power supply
- Large PA Speakers (2)[10] **~1000W**
    - Need to test loudness versus power
    - Power ratings for those large PA's are usually 500W
- Laptops (2)[11] **~140W**
    - 70W Charging brick, but does not need to be hypothetically sustained the whole time
    - One laptop running DJ booth, another running the visualizations
- Mini Projector[12] **~100W**
    - Running visualizations inside the van accompanying the music
- Panasonic TH-42PHD7UY TV[13] (1) **~350W**
    - More visualizations inside the van.

The most essential piece to power is our exterior LED panels. To power the LED panels, we determined that a DC-to-AC converter capable of handling high current draw is required. Each LED panel requires 5V at 4A, resulting in a total current draw of approximately 48A for 12 panels. To accommodate this, we plan to use a 5V/60A power supply, drawing about 300 Watts, ensuring a reliable margin above the estimated load. We also calculated the laptop and tv usage based on what we currently have, but these are subject to change. The Mini projector is based on a model we are considering buying, and the PA speakers are based on the specifications of speakers we have available for us to use through KTRU.

After reviewing the power requirements for the other components of our art car—including LED panels, speakers, laptops, and a projector—we estimate a total power need of approximately 1,890 watts (or 9450 Wh for a 5-hour runtime).

# Engineering Standards

Throughout the development of our art car, adherence to recognized engineering standards has been a key priority.

**1. Road Legality**

The art car design complies with the Orange Show's Houston Art Car Parade regulations: Vehicle Dimensions: The height of the vehicle, including all added components, is kept under 13 feet, as stated in the parade guidelines. Safety Equipment: A 10BC-rated fire extinguisher will be

[10] https://www.guitarcenter.com/Electro-Voice/ZLX-12P-G2-12-2-Way-Powered-Speaker-1500000418076.gc

[11] https://www.anker.com/blogs/chargers/charger-wattage-for-macbook-pro

[12] https://www.amazon.com/270%C2%B0Adjustable-projector-Bluetooth-WiMiUS-S27/dp/B0CCSF3VSY?source=ps-sl-shoppingads-lpcontext&ref_=fplfs&psc=1&smid=A16BRZ2F2CC5AR&gQT=1

[13] https://www.hdtvsolutions.com/Panasonic-TH-42PHD7UY.htm

included, as required for all motorized parade vehicles. The vehicle will be designed to idle for extended periods in the sun without overheating, a necessary feature for participation in the slow-moving parade.

**2. Waterproofing**

Waterproofing is critical to ensure the reliability of electronics and components during outdoor use. All selected materials, adhesives, and electronics are suitable for operation in humid and rainy conditions. Acrylic panels and J-B Weld Cold Weld Epoxy are included for their weather-resistant properties and ability to withstand high temperatures (up to 550°F).

**3. Materials Standards**

The materials used in the construction of the art car meet specific functional and aesthetic requirements. The frame is constructed using 1x1” 4130-grade steel tubing, selected for its strength, durability, and corrosion resistance. Panels: Exterior side panels are made from polycarbonate and copper-painted acrylic, chosen for their lightweight and durable properties.

**4. Power and Electrical Systems**

Power for the LED panels is supplied through a 5V/60A power supply, calculated to meet the estimated load of the system, as outlined in the "Power" section. For our electrical and electronic systems, we relied on IEC, ISO, and UL guidelines when selecting wiring, connectors, and power supplies. These standards helped us ensure that cables, insulation, and terminations were appropriate for expected current loads and environmental conditions.

The structural frame is a modular, exoskeleton-style design, made to avoid permanent modifications to the E-VAN and support all mounted components. It is divided into sections for ease of assembly and disassembly. ASME and SAE guidelines influenced the design of our frame and kinetic sculptures. By referencing these standards, we confirmed that our chosen materials, fabrication methods, and joint configurations would be capable of withstanding anticipated loads and vibrational forces encountered during the parade.

# Design Components

## *Steel Tubing Frame*

### Design Criteria

The frame was designed with several key requirements in mind to ensure it could successfully support the art car panels and any additional loads while being modular, easy to assemble, and stable during use. The frame needed to support a total of 300 lbs, with 150 lbs on each side, while maintaining its stability and structural integrity under dynamic conditions. The design also prioritized modularity, allowing the frame to be easily assembled and disassembled for adjustments, fabrication, and maintenance. Strength and stability were critical to ensure that the frame could support the panels and withstand external forces, such as vibrations during movement, without deformation or failure. Additionally, the design aimed to be simple enough for fabrication without requiring specialized equipment.

### Material Selection

A36 Steel was selected as the primary material for the frame due to its excellent tensile strength, which made it ideal for supporting the 300 lbs total load while ensuring structural integrity. Steel is also easy to weld, cut, and fabricate, which streamlined the construction process. Furthermore, it was more cost-effective compared to other high-strength materials like aluminum, making it the best choice for this project. The 1x1 square tubing used for the frame was chosen for its ease of storage. Unlike bulkier materials like 2x4 lumber, the 1x1 square tubing is thin and long, which allows for more efficient storage. The choice of 1x1 square tubing saved 93.7% in volume and 54.8% in weight compared to using 2x4 lumber, providing significant space and weight savings without compromising the frame's strength.

### Final Design

*Figure 4* illustrates the final design of the frame, which is mounted at five different locations on the van: two rear mounts, two front mounts, and one top resting mount. These mounting points were strategically chosen to connect directly to the van’s existing frame at strong points, ensuring maximum strength and stability.

A key feature of the final design is its modularity. Every connection in the frame was designed to be bolted on, allowing for easy assembly and disassembly. This modular design allows for quick adjustments, repairs, or modifications without the need for extensive rework.

The use of bolted connections ensures that the frame remains reliable and structurally sound while maintaining the flexibility to be taken apart as needed.

The choice of mounting points and beam design was made to ensure that all bolts are placed in tension, rather than shear. Bolts are stronger in tension because they resist forces that pull the material apart. By orienting the bolts to handle tensile forces, the frame achieves greater overall stability and safety, reducing the likelihood of failure under dynamic loads. *Figure 5* provides a detailed view of the bolt placement and shows how the design prioritizes tensile load on the bolts to enhance the frame's performance and maintain its modularity.

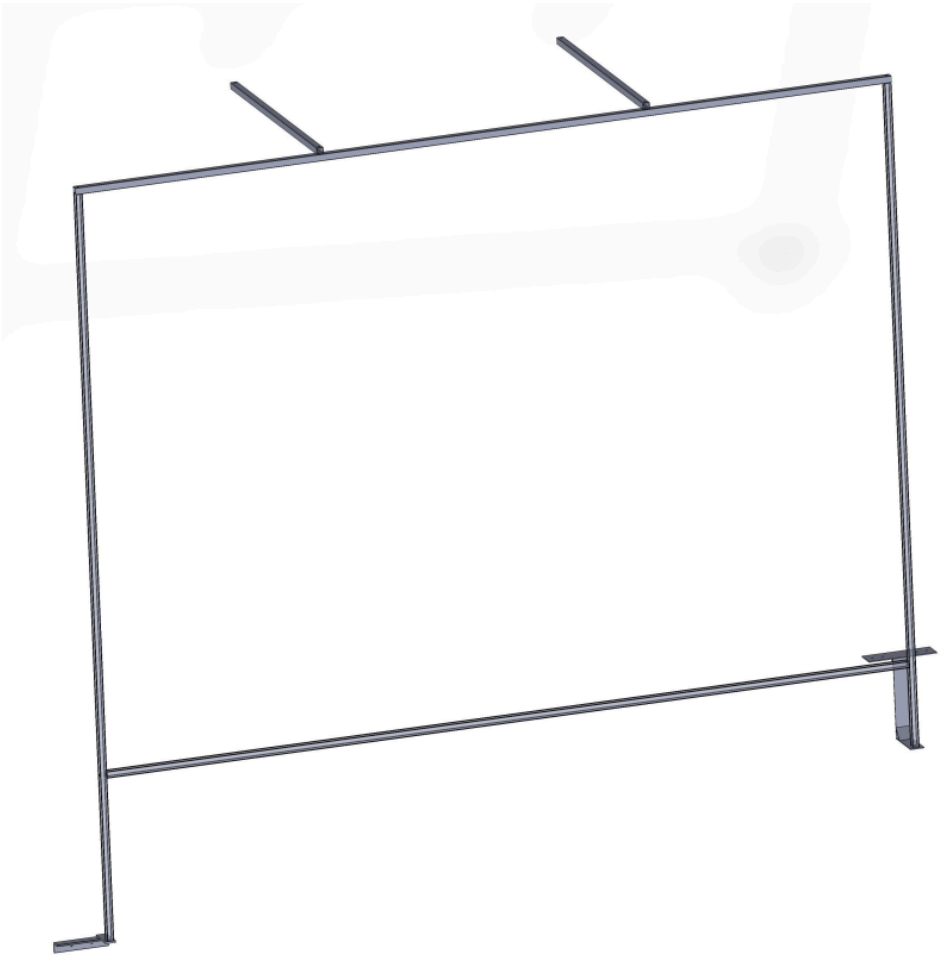

***Figure 4.*** *One side of EVan frame showing the the 2 side mounts and top mount*

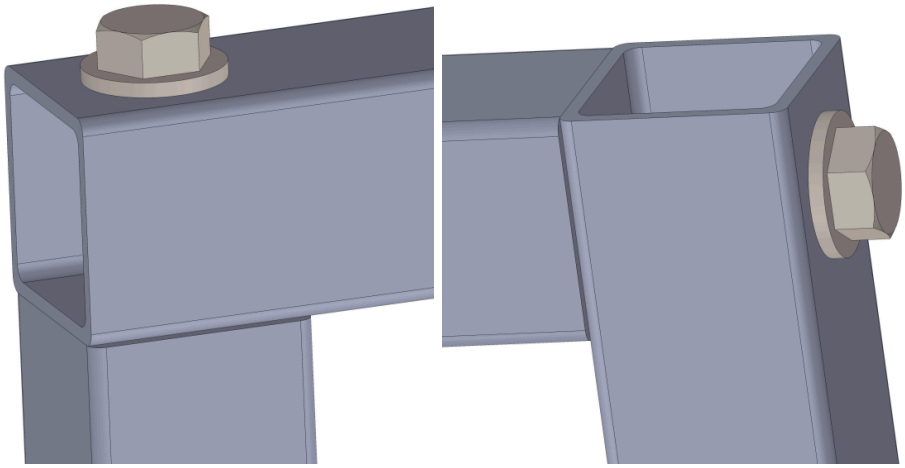

***Figure 5.*** *Chosen design: Tension load(Left); Shear load (Right)*

## Frame Components:

### Beam-to-Beam Connection

The beam-to-beam connection is reinforced using a 1/2-inch ASTM A53 black steel pipe coupler. This coupler serves as both a reinforcement and alignment tool, ensuring the beams are securely connected and aligned properly. The use of the pipe coupler helps prevent weld failure by reducing the stress concentrations that would otherwise occur at the welds. Additionally, it speeds up the fabrication process by simplifying the connection alignment, providing extra security that the welds will hold under load. This design ensures that the frame can bear the required 300 lbs total load without risk of failure.

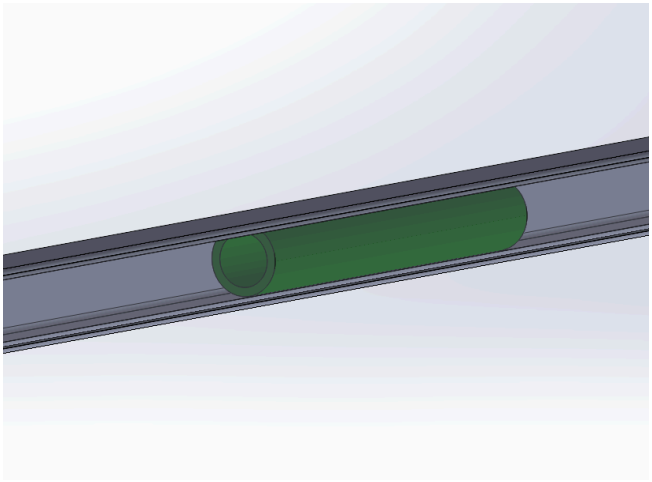

***Figure 6.*** *Steel Cylinder coupler (Green) in 1x1” Beam*

**Rear Connection**

The rear connection uses A36 steel angle iron (8 inches wide, ¼-inch thick) and is bolted directly onto the Tommy gate, which is mounted securely to the van. This connection was chosen because the Tommy gate provides an easily accessible and strong mounting point, ensuring a secure attachment of the frame to the van. The rear connection provides stability and structural integrity while being easy to install and remove if needed for maintenance.

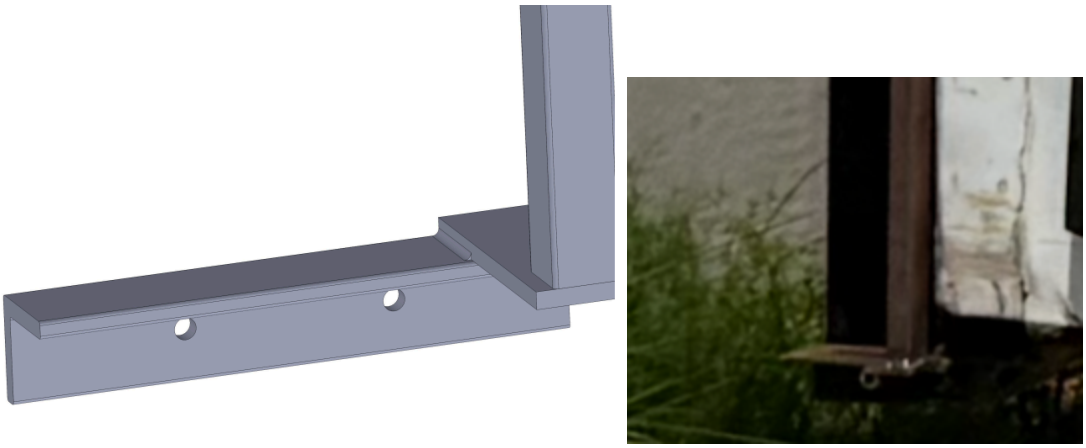

***Figure 7.*** *Rear connection: CAD Model(Left); Fabricated Part (Right)*

**Top Connection**

The top connection provides both vertical and horizontal support to the frame. A ¼-inch plywood layer is used between the frame and the van to distribute forces and protect the van's surface from damage. This design helps to prevent any lateral movement of the frame and ensures that the frame remains stable while maintaining protection for the van. The top connection plays a critical role in ensuring that the frame maintains its structural stability and does not shift during operation.

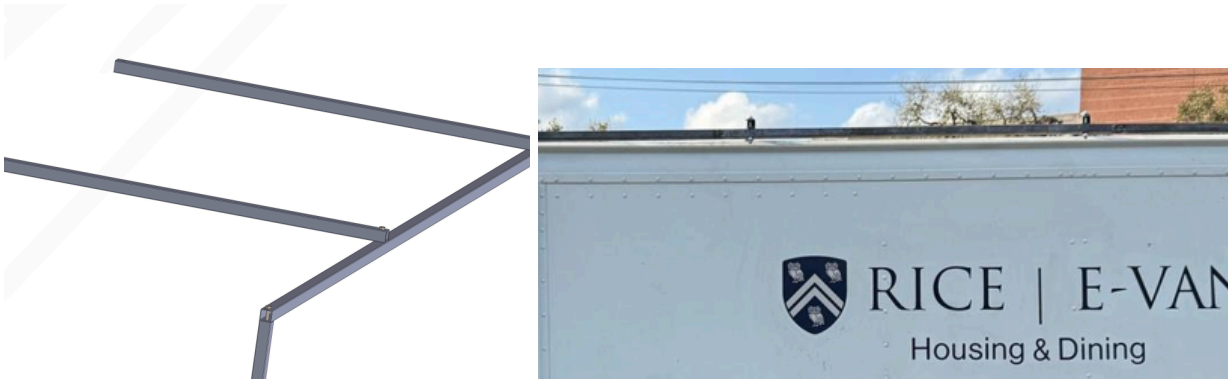


***Figure 8.*** *Top section: CAD Model (Left); Fabricated top support(Right)*

**Front Connection**

The front connection is constructed from 1/8-inch steel and is bolted to the frame using 5 partially threaded bolts (1/4-inch diameter). The front connection was specifically designed to be angular, providing increased rigidity and ensuring that the frame is positioned at the correct angle. This angular design also helps distribute forces more evenly across the frame. The front connection is crucial for providing the necessary structural support at the front end of the frame while allowing for easy assembly and disassembly.

***Figure 9.*** *Front Section Bracket: CAD Model (LEFT); Fabricated (Right)*

**Panel Attachment**

The panel attachment uses ¼-inch bolts to securely fasten the wooden panels to the frame. The panels help add rigidity to the overall structure and can be easily removed or replaced if necessary. This design ensures that the panels stay in place under load while providing a quick and simple method for adjusting or replacing panels as needed. The attachment points were strategically placed to distribute the load evenly and prevent stress concentration in any one area.

***Figure 10.*** *Panel Assembly: Back attachment (Left); Front attachment (Right)*

## Analysis

### Full Frame Analysis

The FEA results for the entire frame showed that the design successfully supports the required 300 lbs load with minimal deformation. The frame remained structurally stable under dynamic loading conditions, with all components performing within safe limits. Only one side of the frame was analyzed since it was symmetrical to the other side. *Figure 11* shows the model with the applied loads and boundary conditions.

### Front Connection: Weakest Part

While the frame as a whole performed well, the front connection was identified as the weakest part of the design, experiencing the highest stress during analysis. This connection, made from 1/8-inch steel, was designed to handle forces from the front end of the frame. Despite experiencing higher stress levels, the FOS for the front connection was 4, indicating that it remains safe under the expected load.

The high stress at the front connection is a result of the design's need to support using a unique s style bracket, which concentrates forces in this region. However, the FOS of 4 confirms that the front connection, while under greater stress than other parts of the frame, will not fail under normal operational conditions.

### Overall Performance

The FEA results demonstrate that the frame is structurally sound, with the front connection being the only area of higher stress. Despite this, the high factor of safety across all key components ensures that the frame can safely support the required load, providing reliability and stability during use.

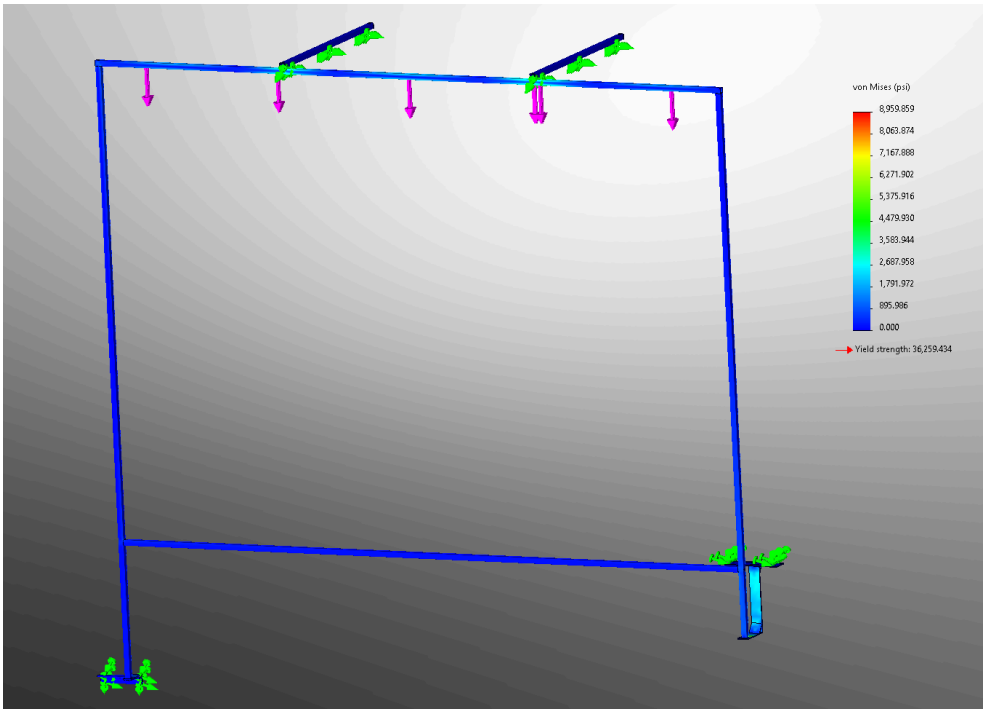


***Figure 11.*** *Frame Von Mises Contour Plot*

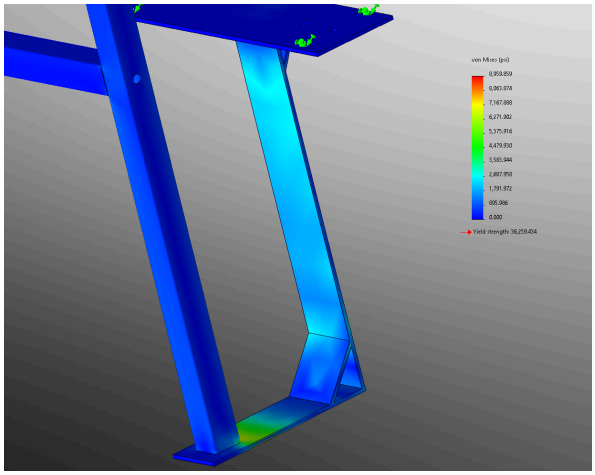

***Figure 12.*** *Front Bracket Von Mises Contour Plot*

## *Custom Panels for a Modular Canvas*

The custom built, modular, lattice panels to cover the sides of the EVan are one of the most defining elements of our Art Car. They represent the thematic vision and engineering creativity of our team. The panels serve not only as decorative canvases but also as structural components, mounting directly to the steel frame of the EVan and allowing for additional attachments to be made to their front–building up a layered three-dimensional collage as the base of our Art Car.

### Theme Development and Design Process

Steampunk draws inspiration from Victorian industrialism, retro-futuristic imagination, and the works of Victorian science fiction authors such as Jules Verne and H.G. Wells. Cyberpunk draws inspiration from visual cues of neon-lit dystopias and urban futurism. We explored reference images and visual motifs of both genres and translated these influences into distinctive lattice patterns to create the intricate panelling.

We used Adobe Illustrator to create vector files for the panels, with thematic cutouts—clockwork mechanisms and mechanical shapes for Steampunk, and circuit-based designs and geometric overlays for Cyberpunk. They were crafted to balance structure with aesthetics, as they would be mounted for display, and exposed to the outdoors at the Houston Art Car Parade.

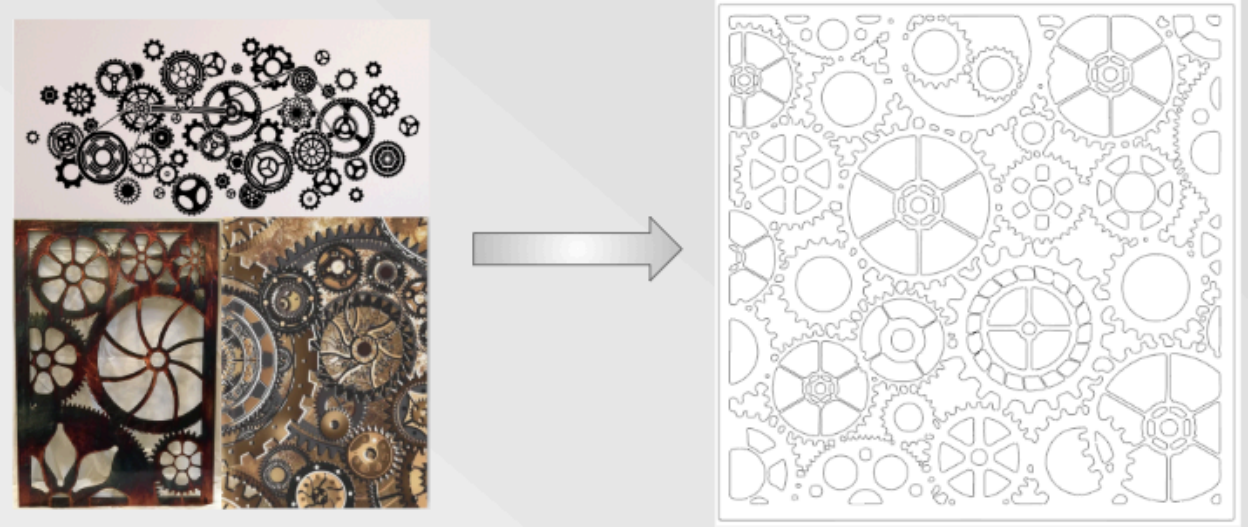

***Figure 13.*** *Steampunk Panel conceptualisation and initial Adobe Illustrator design*

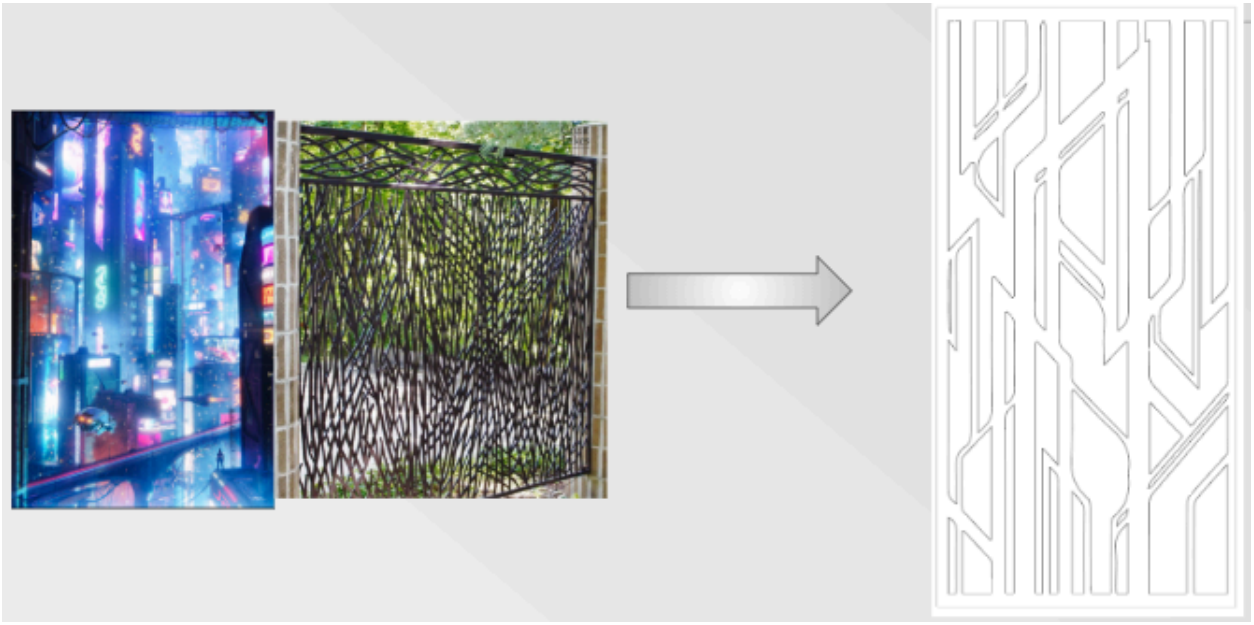

***Figure 14.*** *Cyberpunk Panel conceptualisation and initial Adobe Illustrator design*

We created a 1:10 scale model of the EVan using Adobe Illustrator, and laser-cut material on an Epilog cutter. The model allowed us to try out panel proportions and visualise how motifs would read at a distance.

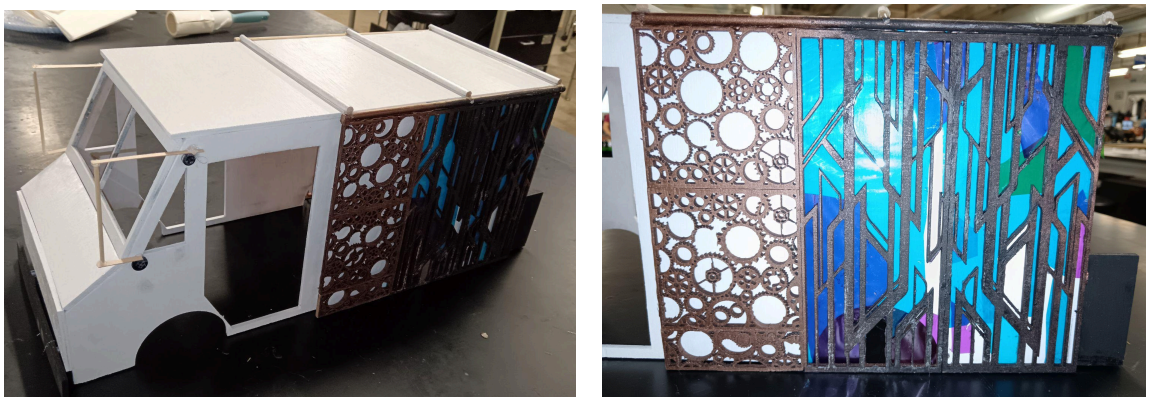

***Figure 15.*** *1:10 Scale Model of EVan with lattice panels*

## Material Exploration and Iterative Testing

We initially explored the idea of cutting the panels from metal such as aluminium or steel, with a waterjet or plasma cutter. This technique, while potentially yielding a more dramatic visual effect and greater durability and toughness, eventually turned out to be impractical due to material cost, added fabrication time, weight concerns, and the need for labour intensive post-processing: angle grinding, deburring and polishing.

We thus investigated 1/8" plywood and 1/8" acrylic as more affordable options. Both offered better workability when laser cut, but early prototypes were fragile—thin connecting

features and small details in the gear patterns would easily break in or after cutting. Acrylic was particularly susceptible to crack, especially under mounting stress.

To address this, we tidied up our Illustrator files, making line weights more robust in breakable areas, rationalising some of the decorative details, and adding structural bonds to prevent breakage when handled.

We evaluated materials based on four key criteria: structural integrity, aesthetic compatibility, workability, and cost. The following table summarises our evaluations:

**Table 1.** Material evaluations and decision

| **Material** | **Pros** | **Cons** | **Decision** |
|---|---|---|---|
| Aluminium | Strong, durable<br>Clean finish<br>Lighter than Steel<br>Theme-Compatible<br>Waterproof<br>Can be polished to reflect sun in parade | Heavy, expensive<br>Required manually intensive post-processing | Rejected |
| Plywood (1/8") | Lightweight, cheap, easy to cut | Fragile in thin areas; poor longevity outdoors | Used with caution |
| Acrylic (1/8") | Clean finish, vibrant colors, theme-compatible | Brittle, especially at mounting points | Used with caution |
| Polycarbonate | Tough, flexible, glossy finish | Slightly more expensive than acrylic | Rejected |
| Plywood (¼") (backboards) | Rigid, paintable, structurally stable | Heavy, prone to swelling if wet | Selected with sealing |

Our final material selections for the lattice panels were **plywood** and **acrylic**, chosen for both their structural properties and their symbolic alignment with our project theme. The transition from plywood to acrylic was a deliberate design decision to represent the evolution of engineering: plywood evokes the mechanical and handcrafted qualities of the steampunk era, while acrylic reflects the sleek, modern aesthetics of cyberpunk.

This conceptual progression is further emphasised by the changing dimensions of the panels, which evolved from uniform 1.5 ft² squares that could be cut on either of the laser cutters we had access to at the OEDK. As we progress to the cyberpunk panels, we cut vertically oriented formats that increase in size. The smaller panel can be cut in the Epilog M2 cutter, the

middle size panel needs the larger bed size and more sophisticated Epilog Fusion Pro, whilst the largest panels that we cut were outsourced to Axiom Space who generously volunteered their 8 by 4 foot CNC machine. Cyberpunk panels progress: 1 ft wide × 2.5 ft tall; 1 ft wide × 3.75 ft tall; 3 ft wide × 7.5 ft tall

To ensure both materials were suitable for laser cutting and long-term durability, we made several crucial design refinements in Illustrator before proceeding with the final cuts. These included:

- Increasing the minimum feature width to 3 mm in fragile areas to reduce the risk of snapping.
- Rounding sharp inside corners to minimise stress concentrations during and after cutting.
- Adding structural bridges between delicate motifs, such as gear spokes, to enhance overall integrity.
- Adding pre cut corner holes to reduce cracking when affixing panels to the backboards.

These modifications preserved the intricate visual language of the lattice designs while ensuring they could be successfully manufactured and remain durable in their final installed forms.

**Table 2.** Iterative prototyping and testing procedure

| **Test Material** | **Issue Observed** | **Action Taken** |
|---|---|---|
| Acrylic | Cracking at corners; Corners snapping off; Damage to intricate design when drilling holes in corner | Increased radius at corners;<br>Thickened elements;<br>Simplified patterns;<br>Added pre cut holes in corners for self tapping screws |
| Acrylic | Laser did not cut all the way through material thickness | Scores design edges to pop out sections that are not cut all the way through;<br>Adjust laser speed and frequency for next cut;<br>Refocus laser |
| Acrylic | Paint did not adhere evenly | Prefab the panels so paint would coat evenly;<br>Ensured application of spray paint in correct conditions to dry |
| Plywood | Fine gear teeth broke off | Increased minimum thickness of major design elements to 3mm;<br>Added pre cut holes in CAD |

## Finish Testing and Aesthetic Treatments

We experimented with a variety of finishes for the panels, including black stain, chalk paint, bronze spray paint, and metallic coppers. We evaluated finishes based on colour vibrancy, adherence to the material, and compatibility with our thematic goals.

**Table 3.** Testing and Evaluating an array of paints and finishes

| **Finish Tested** | **Resulting Look** | **Adhesion & Durability** | **Theme Reflected** | **Final Choice** |
|---|---|---|---|---|
| Black Stain | Flat, grain visible | Poor on acrylic | Steampunk/Cyberpunk | Rejected |
| Chalk Paint (black) | Matte, chalkboard style | Good on backboard | Steampunk | Used (background) |
| Copper Spray Paint | Reflective, metallic | Excellent on acrylic | Cyberpunk | Used (panel) |
| Bronze Spray Paint | Aged-metal look | Good | Steampunk | Considered |
| Black Gloss Spray Paint | Sleek, reflective | High-quality surface | Cyberpunk | Used (panel) |

***Figure 16.*** *Test cuts 1 and 2, using plywood (steampunk) and ⅛”black acrylic (cyberpunk)*

***Figure 17.*** *Full size panels cut from ¼” thick, clear acrylic Finish testing: Rustoleum Spray Paint in Shiny Copper and Gloss Black*

## Backboard Creation and Layering Visual Elements

To elevate the panels' visual depth and modularity, we built three large backboards for each side of the EVan, which the panels would mount onto. Each backboard measured approximately 3 ft x 7.5 ft, and had panels collated on it.

**Table 4.** Backboard design decisions

| **Side** | **Base Finish** | **Visual Overlay** |
|---|---|---|
| Steampunk | Painted matte chalkboard black | White paint; engineering equations |
| Cyberpunk | Metallic silver spray paint | Cellophane (blue/green/purple) for glowing effect |

***Figure 18.*** *Assembling the panels for one side of the EVan*

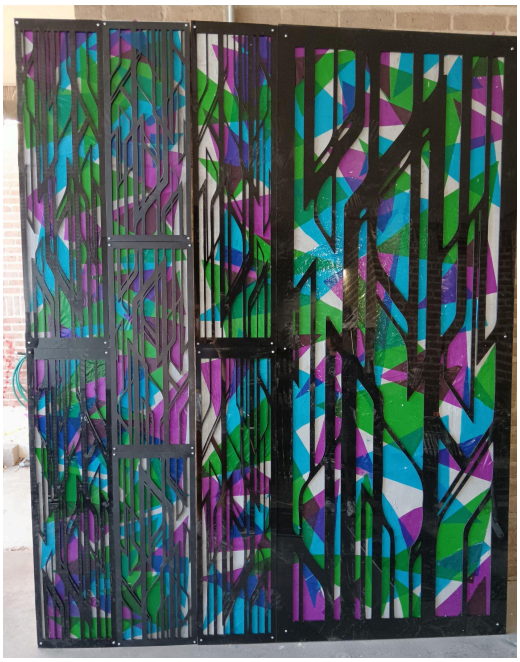

***Figure 19.*** *Cyberpunk Panels after painting and securing to backboard, but before frame attachment and modifications*

## Mounting the Panels and Final Assembly

Mounting the panels presented additional design challenges. Drilling into acrylic after cutting led to cracking, so we revised the Illustrator files to include laser-cut mounting holes. The panels were then secured to the backboards using bolts and coat hooks, which allowed them to hang cleanly over the EVan's steel frame and be removed or replaced as needed. This system of

hooks and bolts ensure stability during movement and the attached panels underwent a driving test.

The final installed panels covered areas measuring 10.5 ft x 7.5 ft on one side and 7.5 ft x 9 ft on the other, creating an immersive, high-impact display that showcased both sides of our theme in harmony.

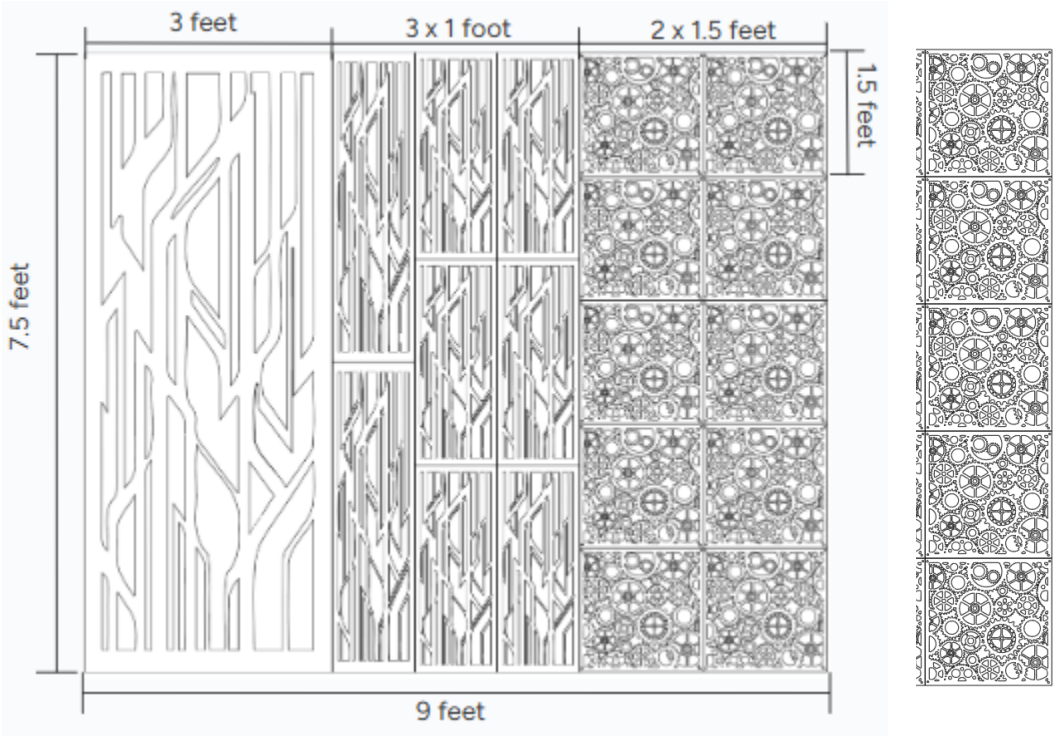


***Figure 20.*** *Final Panel arrangement for passenger side (right-hand-side) of the EVan. Driver side of the van has an additional column of steampunk panels to create a total length of 10.5 ft of panelling.*

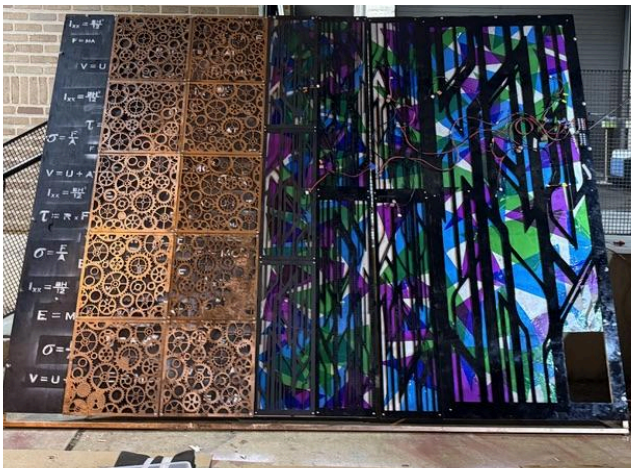

***Figure 21.*** *Completed construction of driver side (left-side) of the panels and frame. Rectangular hole is cut in the bottom right to allow for the panels to sit over the rear-end sensors, without modifying the van.*

## *Audio-reactive LED Display*

### Wiring

The RGB panels use a Hub75 interface, which consists of two connectors per panel: an input and an output. This design allows multiple panels to be chained together, with the input receiving data and the output passing it along to the next panel in the chain. An IDC cable is used to connect the Raspberry Pi to the input connector of the first panel in each chain.

For each LED panel chain, 13 GPIO lines are required, which conveniently fit within the GPIO header available on the Raspberry Pi 4. Up to three separate chains can be supported. The following signals must be connected across all chains in parallel: GND, STROBE, CLOCK, OE (Output Enable), A, B, C, D. These shared lines transmit the same data to all chains simultaneously. Since two chains are used, the A output pin, for instance, on the Raspberry Pi must be connected to both chains' A input pins via separate wires, as well as all other shared signals.

In addition to the shared control signals, each chain requires a dedicated set of six color data lines for the first panel: R1, G1, B1 for the upper half of the LED screen, and R2, G2, B2 for the lower half. These lines must be connected to the corresponding labeled inputs on the panel (marked as [1] and [2]).

| Connection | Pin | Pin | Connection |
|---|---|---|---|
| - | 1 | 2 | - |
| - | 3 | 4 | - |
| - | 5 | 6 | GND |
| STROBE | 7 | 8 | - |
| - | 9 | 10 | E |
| CLOCK | 11 | 12 | OE- |
| [1] G1 | 13 | 14 | - |
| A | 15 | 16 | B |
| - | 17 | 18 | C |
| [1] B2 | 19 | 20 | - |
| [1] G2 | 21 | 22 | D |
| [1] R1 | 23 | 24 | [1] R2 |
| - | 25 | 26 | [1] B1 |
| - | 27 | 28 | - |
| [2] G1 | 29 | 30 | - |
| [2] B1 | 31 | 32 | [2] R1 |
| [2] G2 | 33 | 34 | - |
| [2] R2 | 35 | 36 | - |
| - | 37 | 38 | [2] B2 |
| - | 39 | 40 | - |

***Figure 22.*** *Mapping of Raspberry Pi pins to the Hub75 inputs*

## Core Modules and Dataflow

This section describes the structure, design, and usage of the following three Python modules powering the LED matrix audio-reactive Simplex noise visualization:

- **`samplebase.py`**: Base class for LED matrix configuration and setup (Henner Zeller RGBMatrix wrapper) *add footnote citation with link to the open-source github*.
- **`sound.py`**: Real-time audio capture and analysis (AudioStream & AudioAnalyzer).
- **`simplex.py`**: Main animation logic combining Simplex noise, color mapping, and audio data.

The table below details the core responsibilities of each of the three Python modules.

| Module | Primary Role |
|---|---|
| `samplebase.py` | Parses LED matrix CLI options, initializes `RGBMatrix` hardware. |
| `sound.py` | Captures audio via PyAudio, computes FFT spectrum & beat events. |
| `simplex.py` | Generates noise frames, maps to colors, drives LED display loop. |

### `samplebase.py`

**Module Summary**: Implements a reusable base class (`SampleBase`) for any py5- or Processing-style sketch targeting the Henner Zeller RGBMatrix library. It centralizes parsing of LED matrix options (rows, columns, chain length, brightness, GPIO mapping), constructs the corresponding `RGBMatrixOptions`, and instantiates the `RGBMatrix` hardware interface. Utility methods such as `usleep()` support precise timing in render loops.

- **Class `SampleBase`**:
  - Defines CLI arguments for matrix geometry, PWM settings, and performance tuning.

- `process()`: Translates parsed arguments into `RGBMatrixOptions` and creates the `RGBMatrix` instance.

### `sound.py`

This module provides end-to-end audio I/O and analysis for visualization. `AudioStream` wraps PyAudio to capture audio input into a fixed-length circular buffer. `AudioAnalyzer` consumes that buffer to produce a normalized, log-frequency spectrum and detect rhythmic beats based on energy thresholds and cooldown logic.

- **Class `AudioStream`**:
  - Opens default audio input via PyAudio.
  - Maintains a `deque` buffer of the last `fft_size` samples.
    Methods: `start()`, `stop()`, `update()` (reads audio chunk and appends to buffer).

- **Class `AudioAnalyzer`**:
  - Configurable FFT size, number of frequency bins, dB normalization range, window function.
  - Computes log-spaced frequency bins via `create_log_frequency_bins()`.
  - `update(buffer)` performs windowing, FFT, power spectrum, dB conversion, normalization, smoothing.
  - Beat detection on a user-defined frequency band, with cooldown logic and RMS thresholding.

### `simplex.py`

This module hosts the core animation logic. It defines:

- **`ColorMap`**: Creates a bilinear color gradient in Oklch space between four corner colors, outputting an RGB lookup table.
- **`NoiseGenerator`**: Samples 3D OpenSimplex noise over a static (x,y) grid with a time dimension to produce smoothly evolving patterns.
- **`Display`**: Subclasses `SampleBase` to push numpy frames to the LED matrix.

It glues audio and noise into visuals via `create_frame()`, which:

1. Reads `analyzer.smoothed_spectrum` to build an energy map.
2. Samples `noise_generator.get_frame()` for spatial variation.
3. Uses `ColorMap.get_frame()` to translate (noise, energy) into RGB pixels.
4. Optionally applies chromatic aberration when `analyzer.beat_detected`.

Standard `py5` functions `setup()` and `draw()` drive initialization and per-frame updates, ending in `Display.update()` to render on hardware.

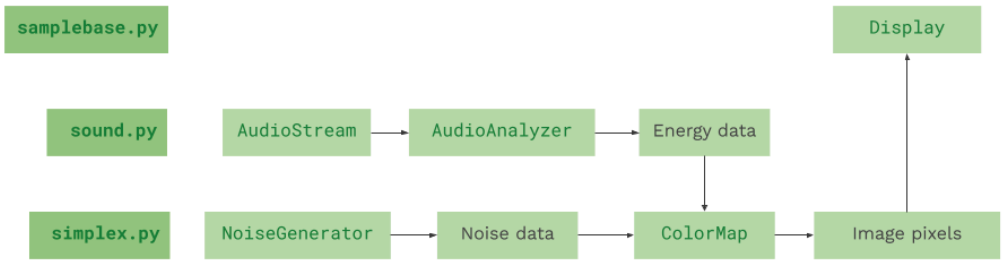


***Figure 23.*** *Data Flow Diagram of Core Python Modules*

## Frequency Response Evaluation

To evaluate the frequency response of the custom `AudioAnalyzer` data structure, several test signals were fed into the `AudioAnalyzer` object and the response was plotted as a spectrogram. The figure below depicts the spectrogram produced from a logarithmic chirp signal that increases from 20 Hz to 20 kHz over 2 seconds.

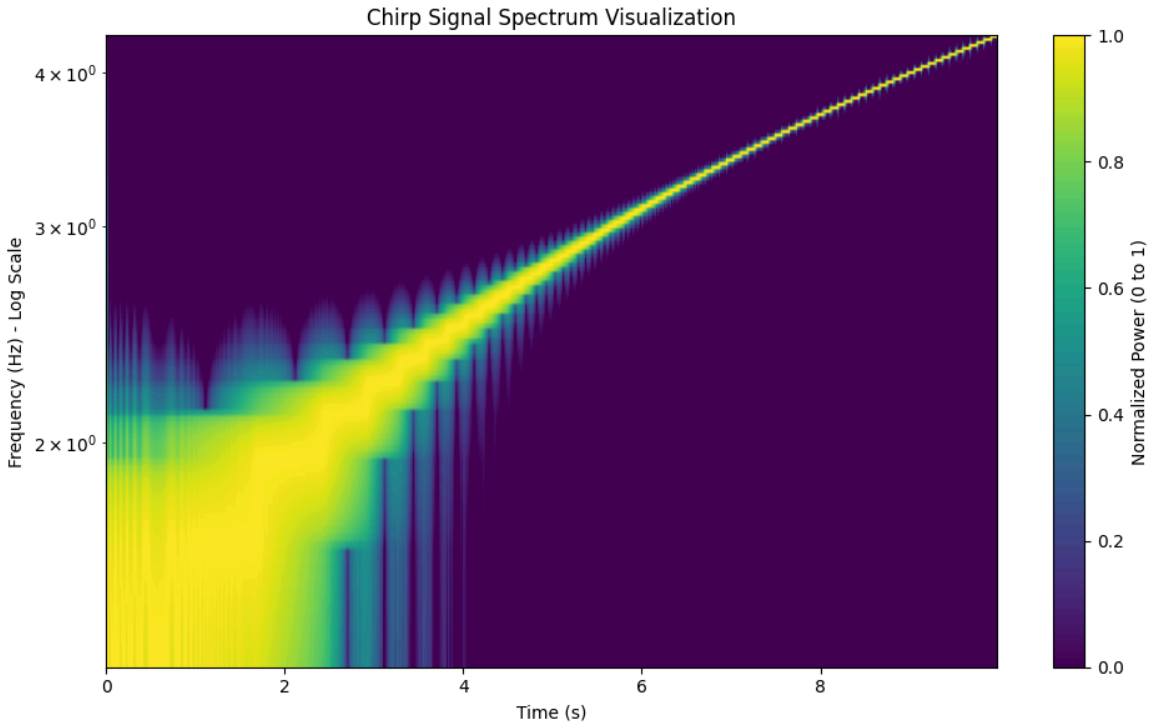


***Figure 24.*** *Spectrogram response of a linear chirp signal*

## Frame Rate Synchronization and Circular Audio Buffer

To synchronize the audio processing with the target frame rate of the animation, an appropriate audio chunk size must be chosen. The audio chunk is the queue of raw audio samples that are processed at each frame of the animation. The audio chunk size is set to be `sample_rate / frame_rate` so that each time we draw a new animation frame, there is exactly one fresh block of audio samples corresponding to that frame's duration. This keeps audio analysis and visual rendering in perfect lock-step, minimizing latency and avoiding drift. In this instance, the sampling rate of the audio stream is 48 kHz, and the target frame rate is 60 fps, so the audio chunk size is 800 raw audio samples.

At each frame, the audio chunk is pushed onto the circular audio buffer encapsulated in the AudioStream object. This buffer size is chosen to be 1024 samples for efficient FFT processing and to promote smoothness in the animation by processing some information from the previous frame.

## Performance Evaluation

To assess the graphical performance of the Raspberry Pi-powered RGB LED matrix and verify compliance with the specified requirements for smooth visual output, a performance test was

conducted. The test involved displaying a full-screen static white image on the LED matrix for a duration of 10 minutes. This configuration represents a worst-case scenario, as it drives all red, green, and blue channels of each LED at maximum intensity, thereby imposing the highest possible demand on both power delivery and data load.

Throughout the test, the system's frame rate was recorded at one-second intervals and subsequently plotted to evaluate temporal consistency. The results demonstrated that the display maintained a stable frame rate of 60 frames per second (FPS) over the entire test duration. Given that 60 FPS is a widely accepted benchmark for smooth and fluid animation, this confirms the system's capability to support real-time visual content reliably under peak load conditions.

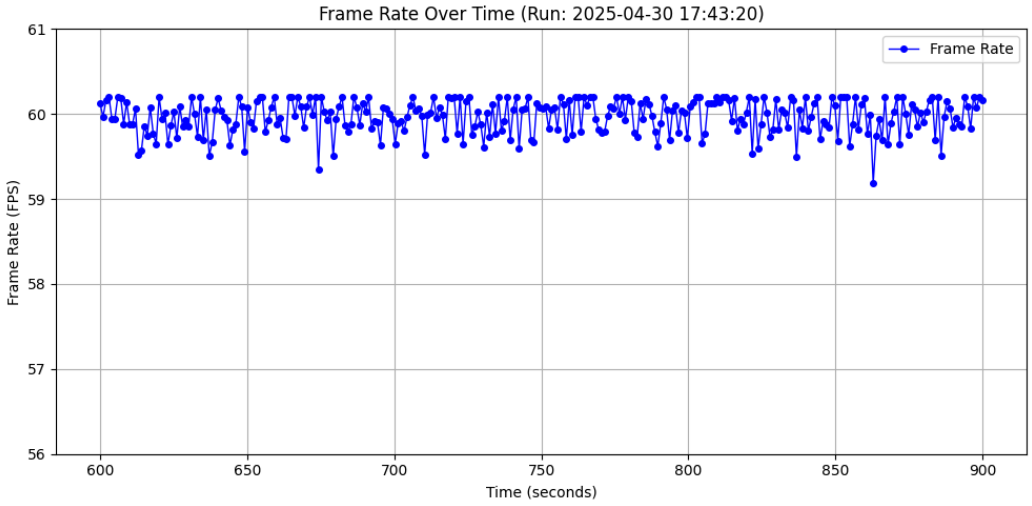


***Figure 25.*** *Frame rate performance test*

## Power Capacity Evaluation

To validate that our electronics could operate untethered on battery power for the entire five‑hour parade we focused on the three loads that will actually be onboard on parade day: the daisy‑chained RGB LED panels, one Raspberry Pi controller, and a small cluster of servo motors, and sized the battery accordingly. First we calculated our total load:

| Load | Qty | Operating (V) | Max I (per) | Power (per) | Sub‑total |
|---|---|---|---|---|---|
| Adafruit P6 32 × 32 LED panels | 15 | 5 V | 4 A | 20 W | **300 W** |

| | | | | | |
|---|---|---|---|---|---|
| Raspberry Pi 4 | 1 | 5 V | 1.2 A | 6 W | 6 W |
| Micro Servos | 6 | 5 V | 0.5 A* | 2.5 W | 15 W |
| **Total worst-case draw** | – | – | – | – | **≈ 321 W** |

Once we understood the amount of power we needed, we wanted a solution that would be both affordable as well as with enough capacity. Considering we needed to run for 5 hours, we needed at least 1.61 kWh of capacity. We chose a 12 V 300 Ah $LiFePO_4$ pack with ≈ 3.6 kWh usable because it offers more than double the calculated energy requirement, giving >50 % reserve even after inverter losses. The battery is the sweet spot between cost and capacity, at roughly $0.35 / Wh it is far cheaper than commercial "power stations" yet half the weight of an equivalent AGM lead-acid pack. The battery however is 12 volts while our panels require 5v of supply. Furthermore, we may still need AC style power to charge any laptops with our extra capacity so we can provide an audio signal from the DJ booth setup. Our below diagram shows the setup we chose. The 12 volt battery is attached to a 500W DC-AC converter that gives us wall outlet style power for our Raspberry Pi, laptop chargers, and 350W 5V DC Supply. The DC Supply then provides power to the 16 LED panels.

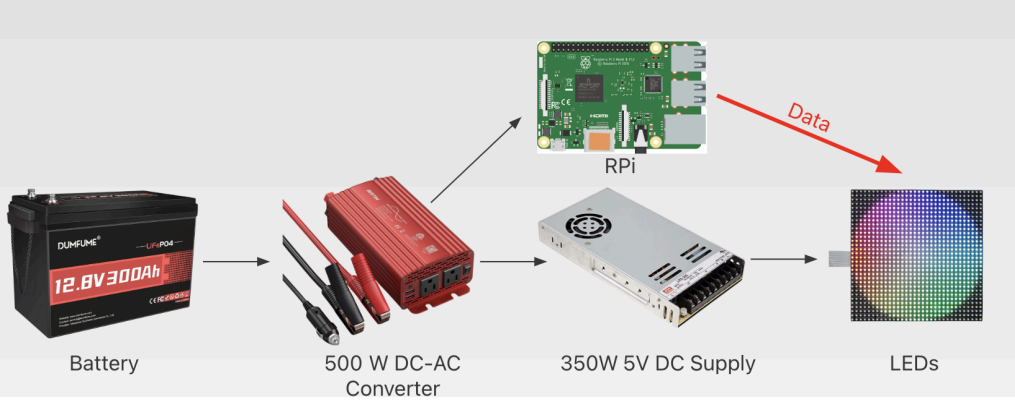


***Figure 26.*** *Power Diagram*

The selected $LiFePO_4$ battery comfortably satisfies capacity, safety, and budget constraints while leaving headroom for unanticipated loads or parade delays, ensuring uninterrupted operation of the LED display, control electronics, and kinetic elements.

## Waterproofing Evaluation

**Why we cared:**
The parade rolls rain-or-shine, and our LEDs all sit only a few feet above the street. Even a short sprinkle could short the 5 V bus or corrode signal headers. We therefore set the minimum target at **IPX4**, the IEC 60529 rating that guarantees an enclosure can withstand water splashes from any direction.

**What IPX4 means:**

- The device is sprayed by an oscillating nozzle delivering **≈ 10 L min$^{-1}$** for **10 min**, while being rotated so every face is exposed.

- No harmful ingress (no malfunction or safety hazard) is permitted during or after the test.

To ensure nothing would be damaged during the test, we wanted to provide extra security by shielding the various electronic components on the back. Our team designed a 3mm acrylic piece for the back of the panel, with the appropriate cut outs for the HUB75 connectors and power connector. We then could seal the connectors using a silicone adhesive such as caulk. We also provided an aesthetically pleasing front clear acrylic cover for the LEDs. We sanded the acrylic to provide a smooth matte finish in front of the LEDs so the light will be diffused more evenly.

**Testing the panels:**

We sprayed one of our panels for 10 minutes, consistently providing high pressure of water from a sink in the OEDK. We rotated the panel as well so that every face is exposed. We then wiped the excess water and powered it on to ensure the panel continued to work. No LEDs failed, the Pi did not reboot, and the current draw remained the same. While not a certified lab test, the outcome gave us high confidence that the design meets or exceeds IPX4.

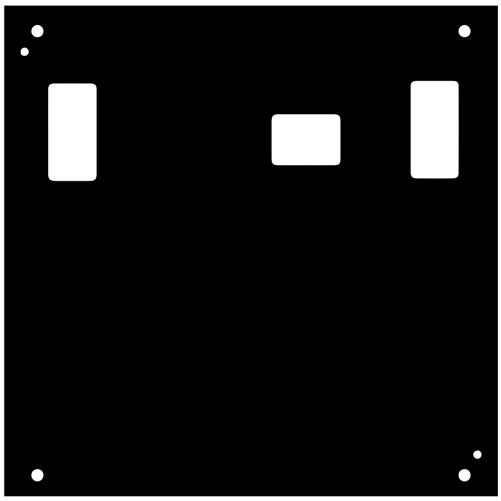

***Figure 27.*** *CAD Model of Acrylic Back Plate*

# Mechanical Elements

A distinctive feature of our Steampunk-themed side of the EVan is the series of functional wooden gears that animate the aesthetic vision with mechanical motion. These gears serve as both symbolic and kinetic elements, merging Victorian industrial motifs with engineering precision. The gear system enhances the interactive quality of the panel design and demonstrates the integration of motion into static art.

Design and Fabrication Process
We began by selecting a series of gear profiles generated using an online gear generator tool, using standard parameters: a diametral pitch of 2, a pressure angle of 18°, and consistent tooth geometry. The generated vector files were exported and used to laser cut the gears from wood. The decision to use wood was both thematic—evoking handcrafted Victorian machinery—and practical, as it allowed easy fabrication and aesthetic customization via staining and painting. Each gear was finished to match the Steampunk aesthetic with dark stains and metallic paints to simulate aged copper and iron.

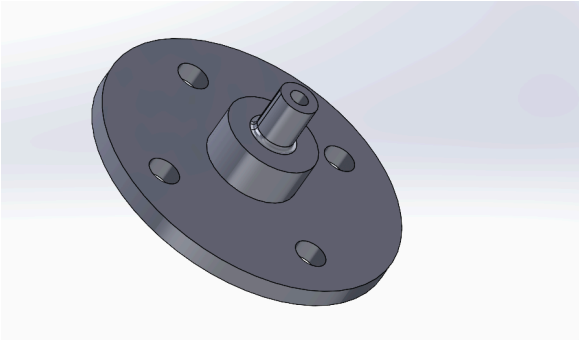

**Figure 28.** Gear Attachment

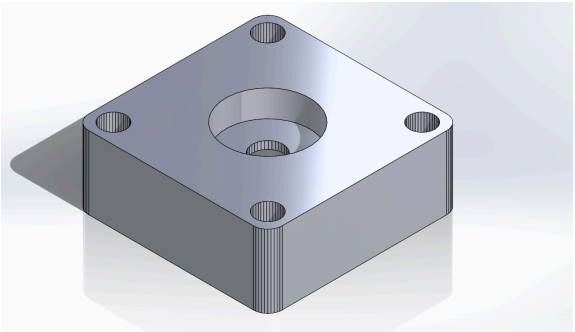

**Figure 29.** Wooden Panel Attachment

The gear attachment was a critical subsystem requiring multiple design iterations. Using SolidWorks, we modeled a custom central hub (shown in Figures 28 and 29) that would both connect to the wooden gear and mount securely to the panel via a flush wooden attachment. This hub was attached to the gear via four ¼" bolts. From the center of the gear hub, we incorporated a heatsunk bolt mechanism that allowed for secure anchoring to the mounting panel while maintaining free rotation.

To ensure smooth motion and structural stability, we embedded two 688-RS skateboard bearings into the wooden attachment, which was bolted through the panel. These bearings allowed the gear to rotate around a central axle while a shoulder on the gear attachment interfaced precisely with the bearing, preventing lateral movement. This assembly allowed for precise, aligned, and freely rotating gears mounted flush with the surface.

**Mechanical Integration and Testing**

This design aimed for a balanced interface: the gears needed to rotate with minimal resistance while maintaining enough stability to survive vibration and transit. Achieving this balance required precise tolerancing and alignment during both design and assembly. We faced several engineering challenges:

**Misalignment Issues**

Gear placement required high positional accuracy to ensure clean meshing and avoid excess friction.

Variation in component thickness led to vertical misalignments, causing partial disengagement or binding.

Slight angular deviation during mounting caused edge rubbing, reducing efficiency and increasing wear.

**Material and Structural Limitations**
PLA components, used in early prototyping, suffered from brittleness and cracking under stress or overtightening.

Thermal expansion during driving tests led to warping and screw loosening.

Over time, PLA degraded under repeated mechanical stress and environmental exposure, particularly in humid conditions.

**Frictional and Rotational Challenges**
Inexpensive skateboard bearings lacked permanent lubrication, leading to inconsistent rotational smoothness.

Friction at gear interfaces and bearing shoulders was worsened by imperfect alignment.

Without grease or dry lubrication, the gears occasionally jammed under sustained motion.

**Manufacturing and Iterative Refinement**
Throughout the build, we experienced significant variability in part performance due to limitations in 3D print quality and tolerance control. This resulted in inconsistent dimensions between otherwise identical parts, necessitating post-processing such as sanding or resizing holes.

Key improvements made across iterations included:

Widening bolt clearances to avoid overtightening-induced cracking.

Adding internal shoulders in gear hubs to better seat bearing faces.

Laser cutting pre-drilled holes in the wood to reduce stress concentrations during bolt insertion.

Switching from PLA to wood-and-metal hybrid assemblies to enhance durability and reduce heat sensitivity.

Despite these setbacks, the final gear assemblies operated smoothly and contributed dynamic, theme-consistent motion to the EVan's steampunk display.

# Conclusion

## *A Brief History of the Project*

In the initial months of the project, the primary challenges posed were the agreement on an artistic theme for the art car and the need to source a vehicle for the team's parade entry. During these months, we met with many people in the Houston art car community, including Rebecca Bass from the Orange Show Foundation, Greg Marshall from Rice University, and various leadership and members of the Houston Art Car Klub (HACK). Talking with these members of the community and attending HACK events during the fall semester helped to provide us guidance with creating a coherent theme for our parade entry and advice on how to begin building our art car.

The team explored several avenues to source a vehicle for the project, including outreach outside of the Rice community with local dealerships and car donation organizations, and outreach within the Rice community, including the Rice Engineering Alumni (REA), the Rice Automotive Club, and Rice Housing & Dining (H&D). Without a direct donation of a vehicle, many of the options for purchasing a cheap used vehicle for the team's parade entry presented a heavy budget constraint as this would have required the majority of the original $2500 budget to go to the purchase of a vehicle, leaving little room for additional budget to be used on the creation of artistic engineering components to decorate our parade entry. To mitigate these challenges, the team submitted additional applications for scholarships from REA and the Moody Foundation, for which we received an additional $1500 in funding, and by the end of the fall semester, we had arranged an agreement with H&D to make use of the E-VAN for the art car parade on the condition that our team made no permanent alterations to the vehicle.

As the team settled on the artistic theme of *Engineering Eras* at the midpoint of the fall semester, we began to come up with more detailed plans for the various design components to include in our parade entry. We decided on creating kinetic mechanical sculptures and a modular LED display as independent components of our design that would encapsulate the two engineering aesthetics–steampunk and cyberpunk–that we hoped to portray. From there, we divided responsibilities among the team to begin building these components and create a frame that could function as a modular canvas which could be easily assembled and disassembled on the E-VAN.

Beginning from the fall semester, the team faced many scheduling challenges and failures of time management. Without the knowledge of the exact vehicle that would be used, the team struggled to properly conceptualize a goal for the project that could be communicated according to engineering standards. This became a factor that reduced the team's motivation to settle on a

detailed plan for the art car and wasted lots of time early in the project that could have been spent on prototyping a solution. This issue could have been mitigated by instead focusing on art car design that was modular, extensible, and flexible to any type of vehicle that the team may have used for the parade. Moreover, due to differences in the "artistic motivation" of individual members of the team, even the selection of an artistic theme that would be interesting and motivating for all team members proved to be a difficult task, especially for those that did not consider themselves "artistic" or "creative." For example, the team spent a few weeks between October and November attempting to conceptualize a Rube Goldberg-inpsired marble run that would function as a complete kinetic sculpture on the art car; however, this idea was eventually scrapped due to insufficient team motivation to see the idea through. It was clear that more time could have been saved if team members, including the team leads, were honest about their interest in ideas proposed in early stages of the project.

By the end of the fall semester, the team had fallen behind schedule in several regards. An original goal was to have a complete steel tubing frame for at least one side of the E-VAN to demonstrate the team's idea for mounting sculpture elements onto our vehicle. Moreover, as the team struggled to settle on clear objectives for the design of the art car, early prototypes were not explored in sufficient depth, which also encouraged team members to work in isolation from each other to develop independent components rather than working together on a cohesive product.

In the spring semester, there was an attempt to address these challenges by choosing to operate the LED display as its own independent installation while allowing other kinetic sculptures to serve as their own installations on the art car. Moreover, a stronger focus was placed on construction of the steel tubing frame and the modular panels to provide a platform for prototyping and mounting installation work. However, the team severely underestimated the time and physical labor required to complete the frame for the van and did not make the necessary effort to organize team members to work together to complete the frame early enough in the project's timeline. The complete steel tubing frame was not finished until the week of the parade; although, its progress was dramatically improved by reassigning a different team member to lead its construction. Additionally, there was a lack of focus on developing the artistic installations that were originally planned for the art car, such as the LED display and the proposed kinetic sculptures, as team members spent necessary time on the physical labor of assembling the modular plywood panels, which were to serve as the basis and canvas for the design, and assisting with construction of the steel tubing frame.

Over the weekend before the parade, a coolant leak from one of the battery housings was discovered as driving tests were being conducted with the E-VAN. To address this problem, the team contacted the E-VAN project's original lead, who instructed members of our team on how to properly disassemble the steel box housing the EV batteries and identify and seal the coolant

leak so that the vehicle would be safe to drive during the parade. Although the team was successfully able to seal the coolant leak in the week leading up to the parade, the EV batteries collapsed on top of each other within the aluminum racks that were designed to house them. Upon consultation, it was determined that the EV batteries presented a serious fire and electrical shock hazard that would be too dangerous for team members to attempt to address without the proper safety equipment and training for managing high voltage electronics.

### *Final State of the Project*

In the final week of the project, the frame and canvas that was originally designed and tested for use on the E-VAN was adapted into an independent standing frame to display on its own at the 2025 Engineering Showcase. This was done after discovering the fire hazard posed by the EV batteries and determining that the E-VAN was unsafe to continue driving. This self-standing frame demonstrated that the steel tubing design was capable of supporting the weight of all artistic pieces, including the modular canvas panels. Additionally, a complete audio-reactive LED display was mounted onto the canvas and demoed at the Engineering Showcase with both musical audio and microphone inputs.

### *Future Work*

If this project were to be proposed as a senior design project in the future, it is of the team's opinion that a vehicle should be sourced and provided from the start of the project, possibly from an actual client. Along these lines, it would be helpful to have the project directed by an "artistic lead" (effectively the client) that can communicate an objective for the design of the desired art car. Presenting the project in this way would address the challenge of conceptualizing a clear goal for which engineering requirements can be communicated and would mitigate the damaging effects that differences in individual motivation–artistic or otherwise–could have on the development of an engaging project that can be accomplished using proper engineering standards.

# Appendices

### *A1: Bill of Materials*

| Transaction ($) | Vendor | Item/Reason | Date |
|---|---|---|---|
| 2,500.00 | OEDK Budget | Senior Project Budget Allocation | Semester Begin |

| | | | |
|---|---|---|---|
| -80.00 | Mouser Electronics | Raspberry Pi | 10/10 |
| -12.28 | Mouser Electronics | SDCS2 | 10/10 |
| -10.00 | OEDK Store | Birchwood Ply for modeling | 31/10 |
| -43.11 | Adafruit | Adafruit Panel | 31/10 |
| -80.00 | OEDK Order | General Materials | 31/10 |
| -10.00 | OEDK Store | Birch Plywood | 14/11 |
| -55.00 | Digi-Key | Raspberry Pi | 06/12 |
| -34.20 | Digi-Key | Meanwell Power Supply | 06/12 |
| -39.95 | Adafruit | LED Panel | 06/12 |
| -47.36 | Amazon | Power Cords | 22/01 |
| -428.44 | Amazon | Electronics and Parts | 27/01 |
| -123.08 | OEDK Store | Acrylic and Plywood | 29/01 |
| -154.62 | OEDK Store | Acrylic | 31/01 |
| -162.25 | Adafruit | LED Panels | 03/02 |
| -78.56 | Amazon | Cellophane, Spray Paint | 03/02 |
| -7.99 | Amazon | Ball Bearings | 06/02 |
| -347.58 | Adafruit | 8 Panels | 12/02 |
| -17.15 | Amazon | Cables | 12/02 |
| -362.00 | A&C Plastics | Acrylic Panels for Axiom | 20/02 |
| -72.69 | Amazon | Paints and Microphones | 20/02 |
| -51.85 | Home Depot | Wood Panelling | 24/02 |
| -51.85 | Home Depot | Wood Panelling | 24/02 |
| -86.16 | Onlinemetals.com | 42 ft Steel Tubing | 04/03 |

| | | | |
|---|---|---|---|
| -40.95 | Amazon | Audio Cables | 04/03 |
| -33.44 | Amazon | Paint Cans | 06/03 |
| -64.50 | Home Depot | Steel Junction and Couplers | 06/03 |
| -22.98 | Amazon | Power Cables, Wire Levers | 07/03 |
| -22.90 | OSH Park | Custom PCBs for RPi Hub | 14/03 |
| -9.99 | Amazon | 3x DC Motor | 01/04 |
| -48.68 | Home Depot | Titanium Drill Bits | 07/04 |
| -25.98 | Select | Spitfire Bearings | 08/04 |

| **Transaction ($)** | **Vendor** | **Item/Reason** | **Date** |
|---|---|---|---|
| 1,500.00 | Moody Fund | Opportunity Fund Allocation | 30/10/2024 |
| -210.00 | — | Tax/Adjustment | — |
| -95.14 | Electromaker.io | 2x Adafruit Panels | 12/11 |
| -384.55 | Online Metal Supply | 36 ft 1x1" Steel Tubing | 12/11 |
| -173.89 | Online Metal Supply | 18 ft 1x1" Steel Tubing | 21/11 |
| -113.22 | McMaster Carr | 100 in² Sheet Steel | 21/11 |
| -75.70 | Amazon | 20 ft Steel Tubing | 20/11 |
| -16.23 | Amazon | 1/8" Aluminum Sheet | 20/11 |
| -31.38 | Amazon | Acrylic Tube | 20/11 |
| -76.80 | Amazon | 15 ft Mild Steel Tubing | 20/11 |
| -16.54 | Amazon | Cellophane | 26/11 |
| -105.20 | Amazon | 8 Large Acrylic Sheets | 26/11 |
| -32.03 | Amazon | 10 Small Acrylic Sheets | 26/11 |

| | | | |
|---|---|---|---|
| -16.44 | Amazon | Cellophane | 26/11 |
| -21.93 | Amazon | Spray Paint | 28/11 |
| -51.94 | Amazon | Servo Motors, Arduino UNO | 02/12 |
| -72.99 | Amazon | Black Plexiglass | 25/01 |

## *A2: Instructions for Product Assembly*

The product frame is designed using a modular steel cage constructed from 1"x1" square steel tubing, with plywood and acrylic panels mounted to form side enclosures. The system integrates thematic backboard aesthetics—steampunk and cyberpunk—through decorative overlays. This section details how subsystems (frame, panels, and aesthetics) fit together, supported by CAD assemblies.

### A2.1: Frame Assembly (Steel Tubing)

The frame is composed of beams, brackets, and beam-bracket fasteners, all selected for their strength, stability, and ease of assembly. The construction process required standard metalworking tools, including a welder, cutting tools, and grinders, for fabrication and assembly.

1/2-inch ASTM A53 black steel pipe couplers were cut to 6 inches and welded into the beams to reinforce the beam-to-beam connections.  shows the coupler already welded inside one of the beams.

End cap fasteners were made from 1/16-inch steel, with a hole drilled in the center and a nut for the 5/16-inch bolts used in the design. These fasteners were welded to the ends of the beams to enable easy connection between the beams. *Figure 1a* illustrates an attachment fastener at the end of a beam.

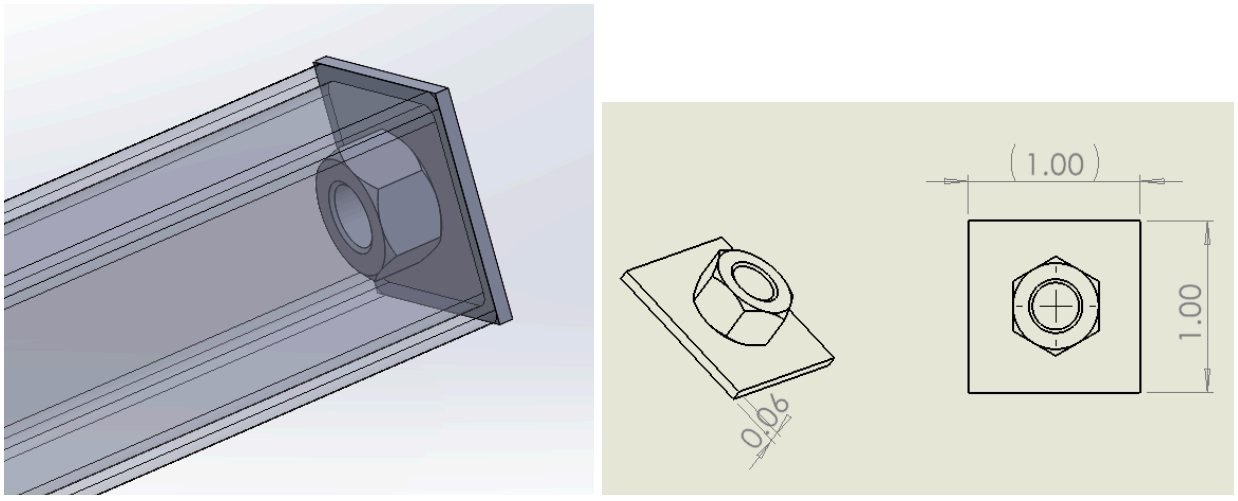


***Figure 1a.*** *Fasting End Caps for beam. Visual CAD (Left); Dimensions (Right)*

The cage consists of structural tubing segments welded or bolted together, forming a rectangular cage structure affixed to the base of the van using custom built brackets detailed in *Figures 2a, 3a.*

## Rear Bracket

The rear bracket consists of ¼-inch angle iron, 8 inches long, with two 5/16-inch holes for mounting to the frame. A 4x1.5x1/4-inch steel plate is welded onto the bracket for accurate beam placement. *Figure 2a* illustrates the dimensions of the rear bracket.

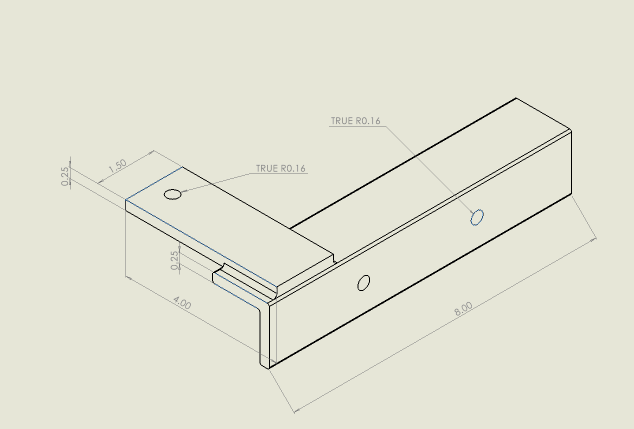


***Figure 2a.*** *Rear Bracket Dimensions*

## Front Bracket

The front bracket is composed of ⅛-inch steel, which is welded together. *Figure 3a* provides the dimensions of the front bracket. The front bracket plays a crucial role in supporting the front portion of the frame, ensuring the structure's stability.

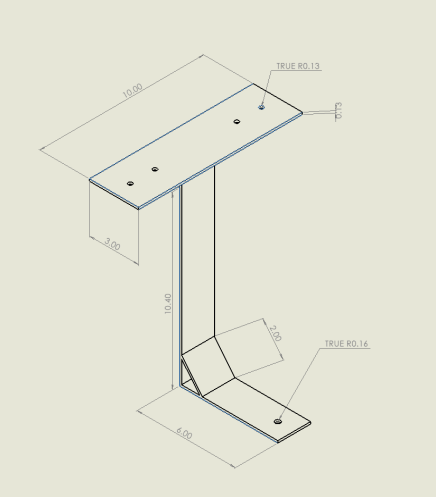


***Figure 3a.*** *Front Bracket Dimensions*

**Beams**

The frame is primarily constructed using 1x1 square tubing, reinforced with various components to ensure structural integrity. *Figure 4a* shows the arrangement of the beams and brackets, along with the frame's dimensions.

**Table 1a.** Steel Tubing Quantity and Layout

| Position | Quantity | Length (ft) | Dimensions |
|---|---|---|---|
| Top Right Horizontal | 1 | 9.0 | 1"x1" Square |
| Middle Right Horizontal | 1 | 9.0 | 1"x1" Square |
| Bottom Right Horizontal | 1 | 9.0 | 1"x1" Square |
| Top Left Horizontal | 1 | 10.5 | 1"x1" Square |
| Middle Left Horizontal | 1 | 10.5 | 1"x1" Square |
| Bottom Left Horizontal | 1 | 10.5 | 1"x1" Square |
| Vertical Corner Posts | 4 | 7.5 | 1"x1" Square |
| Top Cross Members | 3 | 6.0 | 1"x1" Square |

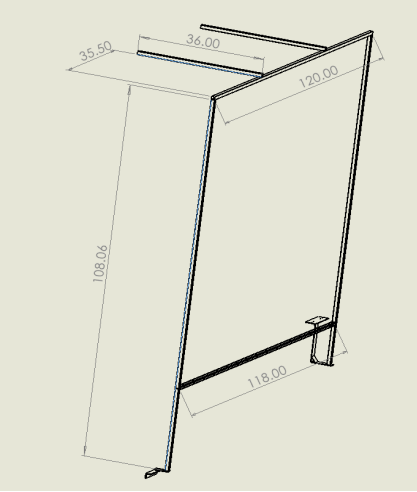


***Figure 4a.*** *Full Frame Dimensions and Assembly*

**Total Steel Tubing Required:**
= (3 × 9) + (3 × 10.5) + (4 × 7.5) + (3 × 6)
= 27 + 31.5 + 30 + 18
= **106.5 ft of 1"x1" square steel tubing**

**Assembly Notes:**

- Use corner joints (preferably gusseted or welded) for structural rigidity.
- Anchor vertical posts securely to the base via plates or through-bolting.
- CAD models show positioning for load distribution and aesthetic alignment.

### A2.2: Panels CAD and Assembly Diagram

Panels mount to both long sides of the cage. Six main backboards are cut from plywood (½" - ¾" thick recommended) and painted/decorated before being fastened to the steel frame.

**Panel Breakdown:**

| Side | Panel Dimensions (ft) | Surface Treatment |
|---|---|---|
| Passenger | (3) panels: 3 × 7.5 | 2 gloss/silver/cellophane<br>1 chalk black steampunk |
| Driver | (3) panels: 3 × 7.5, 3 × 7.5, 4.5 × 7.5 | 2 gloss/silver/cellophane<br>1 chalk black steampunk (large) |

**Surface Preparation:**

Steampunk Panels (x2):

- Sand and prepare backboard surface
- Paint with black chalkboard paint.
- Decorate using white chalk markers with engineering-style equations and vector diagrams.
- Acrylic steampunk lattice panels mount  with screws.

Cyberpunk/Gloss Panels (x4):

- Sand and prepare backboard surface
- Silver gloss spray paint finish.
- Seal with a clear topcoat.
- Colour overlays applied using cellophane film under clear acrylic sheets.

**Driver Side Panel Installation (from front to back):**

- 15 Steampunk Squares
    i. Layout: 5 (tall) × 3 (wide) matrix

ii. Each: 1.5' × 1.5'
iii. Mounted over one 3 × 7.5' backboard
iv. Screwed into 1/2” plywood backboard behind with 1” self-tapping screws

- Cyberpunk Panels (smaller)
  i. Layout: 3 (tall) × 2 (wide)
  ii. Each: 1' wide × 2.5' tall
  iii. Layout: 2 (tall) x 1 (wide)
  iv. Each: 1’ wide x 3.75’ tall
  v. Mounted on second 3 × 7.5' backboard
- Cyberpunk Panels (large)
  i. One large: 3' wide × 7.5' tall (rear-most section)

**Panel Mounting Notes:**

- Ensure acrylic decorations are pre-drilled to prevent cracking.
- Use rubber spacers if mounting clear overlays to reduce pressure points.

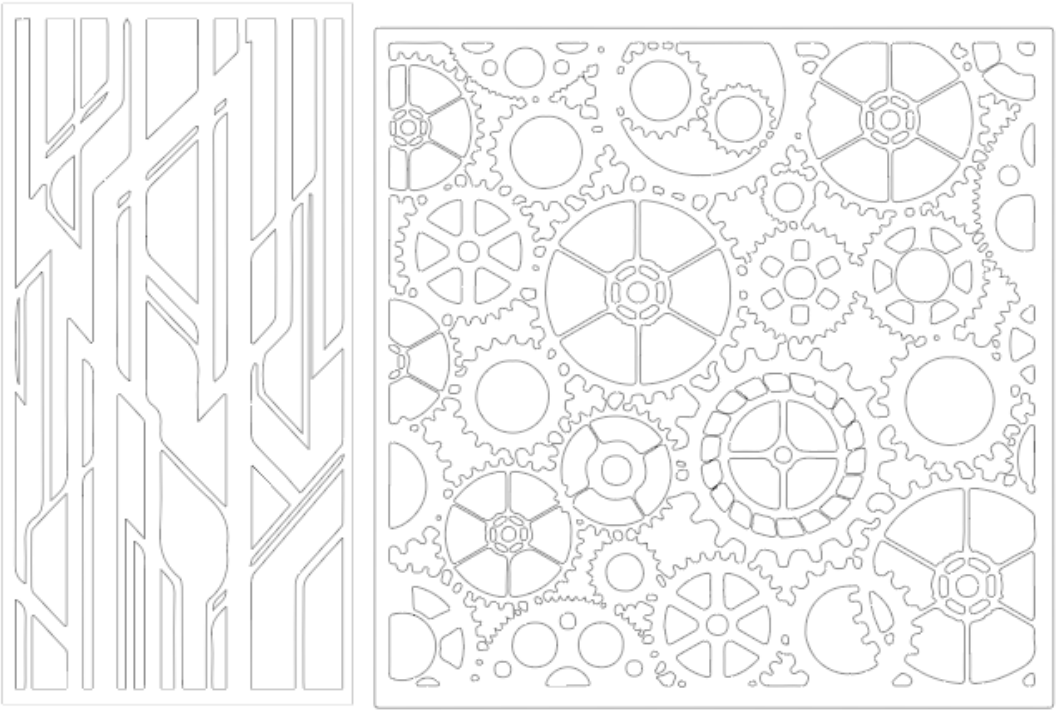

***Figure 5a:*** *Lattice Panel designs. Cyberpunk (left); Steampunk (right)*
*Adjust size accordingly to cover backboards appropriately*

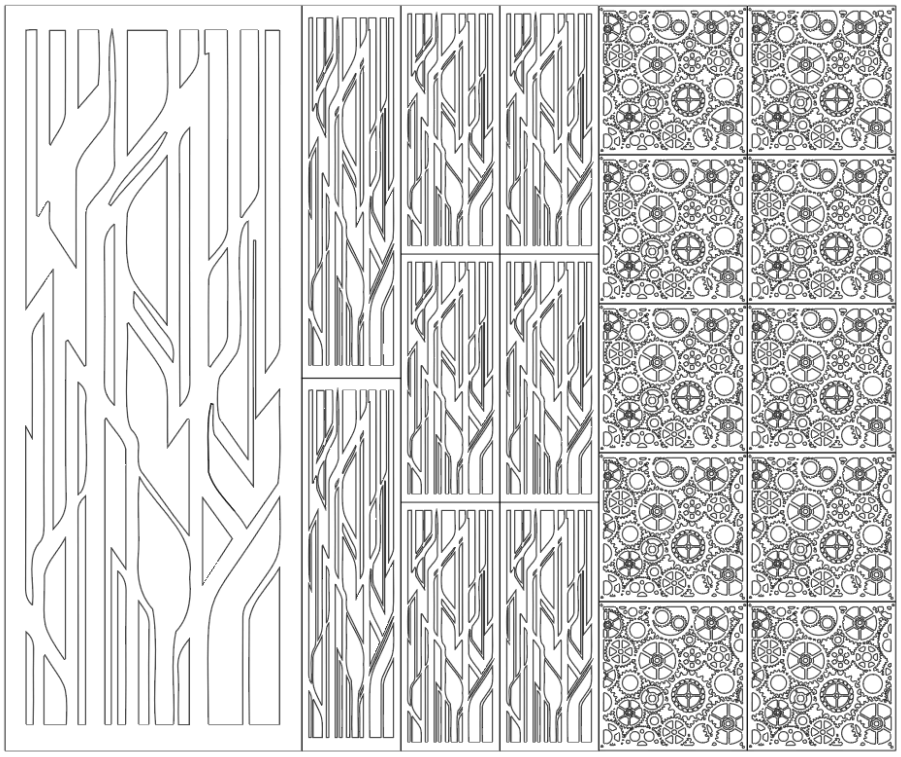

***Figure 6a.*** *Final lattice panel configuration*

**Panel Attachment**

*Figure 10* shows how ¼-inch bolts were used to fasten the wooden panels to the frame. These bolts help maintain rigidity in the overall structure, ensuring the panels stay securely in place.

| Component | Fastener Type | Size | Notes |
|---|---|---|---|
| Acrylic Panels | Machine Screw + Washer | #8-32, 3/4" | With rubber spacer for acrylic |
| Plywood Backboard | Wood Screws | #10, 1" | Use with pilot holes |
| Backboard to Frame | Bolts + Locknuts | 1/4” × 1.5” | Drill through steel + plywood. Place approximately 1ft apart |
| Optional Mount Brackets | Steel L-Brackets | Varies | Pre-drilled, 90° offset |

# *A3: User's Manual*

# *A4: Python Modules Source Code*

## samplebase.py

File: samplebase.py is adapted from
https://github.com/hzeller/rpi-rgb-led-matrix/tree/master/bindings/python/samples

## simplex.py

File: simplex.py

```
001: """
002: An animation using Simplex noise and real-time audio input.
003:
004: This module creates an LED matrix animation driven by OpenSimplex noise and
005: modulated by live audio input using the `AudioStream` and `AudioAnalyzer`
006: classes. Each frame combines Perlin-like noise with frequency bin data mapped
007: through a ColorMap, and may include visual effects like chromatic aberration
008: on beat detection.
009:
010: py5 for Processing is used as the primary graphics package for the animation.
011: The Display object is a wrapper around the RGBMatrix object from the (c) Henner
012: Zeller h.zeller@acm.org LED-matrix library. At each frame, the Simplex noise
013: function is sampled and displayed in the Processing sketch. The numpy array
014: representation of the Processing sketch is then used to display the frame on
015: the LED matrices.
016: """
017:
018: import sys
019:
020: import py5
021: import pyaudio
022: import numpy as np
023: import cv2
024: from PIL import Image
025: from coloraide import Color
026:
027: # from rgbmatrix import RGBMatrix, RGBMatrixOptions
028: from samplebase import SampleBase
029: from sound import AudioStream, AudioAnalyzer
030:
031:
032: # Dimensions for the LED matrix setup
033: LED_ROWS = 32
034: LED_COLS = 32
035: LED_CHAIN_LENGTH = 8
036: LED_PARALLEL = 2
037:
038: # Parameters for the Processing sketch
```

```
039: SKETCH_WIDTH = LED_COLS * LED_CHAIN_LENGTH
040: SKETCH_HEIGHT = LED_ROWS * LED_PARALLEL
041: FRAME_RATE = 60
042: STEP_SIZE = 0.02
043:
044: # Audio processing parameters
045: SAMPLE_RATE = 48000             # Hz
046: FORMAT = pyaudio.paFloat32      # 32-bit floating point
047: CHANNELS = 1                    # Mono
048: CHUNK_SIZE = SAMPLE_RATE // 60  # Synchronized for 60fps
049: FFT_SIZE = 1024                 # FFT size for spectral analysis
050:
051:
052: class ColorMap:
053:     """
054:     Represents a color palette for mapping 2D coordinates to RGB colors
055:     via bilinear interpolation in Oklch color space.
056:
057:     This class constructs a color gradient over a unit square using
058:     four corner colors. It supports efficient lookup of colors at any (y, x)
059:     index, and can be used to colorize frames based on 2D mappings like noise
060:     or audio data.
061:     """
062:
063:     def __init__(self, corners, size):
064:         """Initialize a ColorMap with the given corner colors and size.
065:
066:         Corner colors are specified as a list of four colors in Oklch
067:         color space. The input list should be ordered as follows:
068:             [top_left, top_right, bottom_left, bottom_right]
069:
070:         Args:
071:             colors: A list of four colors, one for each corner of the unit
072:                 square.
073:             size: A tuple (height, width) specifying the size of the color
074:                 map.
075:
076:         """
077:         self.top_left = Color('oklch', corners[0])
078:         self.top_right = Color('oklch', corners[1])
079:         self.bottom_left = Color('oklch', corners[2])
080:         self.bottom_right = Color('oklch', corners[3])
081:
082:         # Initialize the color map
083:         self.cm = self.init_map(size[0], size[1])
084:
085:     @staticmethod
086:     def default():
087:         """
088:         Create a default ColorMap with a predefined color gradient.
089:
090:         Returns:
091:             ColorMap: A ColorMap instance sized for the sketch dimensions.
```

```
        """
        corners = [
            [0.6102, 0.233, 292.61],
            [0.4496, 0.233, 292.61],
            [0.6102, 0.233, 323.65],
            [0.7443, 0.1785, 61.14],
        ] # defined using Oklch color space
        return ColorMap(corners, (SKETCH_HEIGHT, SKETCH_WIDTH))

    def init_map(self, height, width):
        """
        Generate a full RGB color map by interpolating between corner colors.

        Colors are first interpolated along the vertical edges in Oklch space,
        then horizontally for each row. The result is a smooth bilinear blend
        across all four corners.

        Args:
            height (int): Number of rows in the output color map.
            width (int): Number of columns in the output color map.

        Returns:
            np.ndarray: A (height x width x 3) uint8 array of RGB values.
        """
        colors = np.zeros((height, width, 3), dtype=np.uint8)

        # Interpolate along vertical edges
        left = Color.interpolate(
            [self.top_left, self.bottom_left],
            space='oklch',
            length=height
        )
        right = Color.interpolate(
            [self.top_right, self.bottom_right],
            space='oklch',
            length=height
        )

        # Interpolate along the horizontal axis for each row
        for y in range(height):
            y_norm = y / (height - 1)

            row = Color.interpolate(
                [left(y_norm), right(y_norm)],
                space='oklch',
                length=width
            )

            for x in range(width):
                x_norm = x / (width - 1)
                color = row(x_norm).convert('srgb').to_dict()
                r, g, b = [int(np.clip(c, 0, 1) * 255) for c in color['coords']]
                colors[y, x] = [r, g, b]
```

```
145:
146:         return colors
147:
148:     def get_color(self, y, x):
149:         """
150:         Retrieve a single RGB color at the specified coordinates.
151:
152:         Args:
153:             y (int): Row index.
154:             x (int): Column index.
155:
156:         Returns:
157:             np.ndarray: RGB color as a 3-element uint8 array.
158:         """
159:         return self.cm[y, x]
160:
161:     def get_frame(self, y_indices, x_indices):
162:         """
163:         Retrieve a frame (batch of colors) from the color map given index arrays.
164:
165:         This function supports bulk retrieval of color values using 2D numpy
166:         arrays of y and x indices (e.g., from remapped noise or audio data).
167:
168:         Args:
169:             y_indices (np.ndarray): 2D array of row indices.
170:             x_indices (np.ndarray): 2D array of column indices.
171:
172:         Returns:
173:             np.ndarray: RGB image (height x width x 3).
174:         """
175:         return self.cm[y_indices, x_indices]
176:
177:
178: class NoiseGenerator:
179:      """
180:     Generates 3D OpenSimplex noise frames for animation.
181:
182:     This class precomputes a 2D grid of spatial coordinates and animates
183:     through time by incrementing a third dimension (t) to produce smoothly
184:     evolving noise fields. These noise fields are typically used to drive
185:     visuals like color lookups or displacement maps in animations.
186:
187:     Attributes:
188:         xx (np.ndarray): X-coordinates of the noise grid.
189:         yy (np.ndarray): Y-coordinates of the noise grid.
190:         tt (np.ndarray): Time component (1D array) used for 3D noise sampling.
191:         time (float): Current time value (increments each frame).
192:         default_step_size (float): Initial speed of time evolution.
193:         step_size (float): Current frame-to-frame increment in time.
194:     """
195:
196:     def __init__(self, height, width, range=2, step_size=0.05):
197:         """
```

```
198:         Initialize the NoiseGenerator with grid dimensions and motion parameters.
199:
200:         Args:
201:             height (int): Number of rows in the spatial noise grid.
202:             width (int): Number of columns in the spatial noise grid.
203:             range (float): Spatial extent of the noise field (controls frequency).
204:             step_size (float): Temporal step size for each animation frame.
205:         """
206:         self.xx, self.yy = np.meshgrid(
207:             np.linspace(0, range, num=width, dtype=np.float32),
208:             np.linspace(0, range, num=height, dtype=np.float32)
209:         )
210:         self.tt = np.zeros((1,), dtype=np.float32)
211:         self.time = 0.0
212:         self.default_step_size = step_size
213:         self.step_size = step_size
214:
215:     def get_frame(self):
216:         """
217:         Generate the next frame of 2D noise based on the current time.
218:
219:         Increments the internal time value, then samples a 3D OpenSimplex
220:         noise field at (x, y, t) where t evolves linearly.
221:
222:         Returns:
223:             np.ndarray: 2D array of noise values in range [-1, 1].
224:         """
225:         self.time += self.step_size
226:         self.tt[0] = self.time
227:
228:         return py5.os_noise(
229:             self.xx, self.yy, self.tt.reshape(1,1,1)
230:         ).squeeze()
231:
232:
233: class Display(SampleBase):
234:     """A class representing the LED matrix setup.
235:
236:     Attributes:
237:         matrix: an RGBMatrix object used to control the LEDs
238:         double_buffer: a FrameCanvas object, which is a buffer used to
239:             read/write image frames to display on the LEDs
240:     """
241:
242:     def __init__(self, *args, **kwargs):
243:         """Initializes the instance using the parameters in RGBMatrixOptions.
244:
245:         Args:
246:             options: an RGBMatrixOptions object, which contains parameters
247:                 that specify the LED matrix configuration
248:         """
249:         self.matrix = None
250:         super(Display, self).__init__(*args, **kwargs)
```

```
        self.process()
        self.double_buffer = self.matrix.CreateFrameCanvas()

    def update(self, pixels):
        """Write and display an image frame on the LEDs.

        Args:
            pixels (nd.array): an array representing the image frame
        """
        rgb_array = pixels[..., 1:]  # drop the A channel
        image = Image.fromarray(rgb_array, mode='RGB')

        self.double_buffer.SetImage(image, unsafe=False)
        self.double_buffer = self.matrix.SwapOnVSync(self.double_buffer)


# Configure LED matrix
# --led-rows=32 --led-cols=32 --led-slowdown-gpio=4 --led-no-hardware-pulse
LED_NO_HARDWARE_PULSE --led-no-drop-privs
matrix = Display()

# Simplex noise parameters
t = 0.0
colormap = ColorMap.default()
noise_generator = NoiseGenerator(SKETCH_HEIGHT,
                                 SKETCH_WIDTH,
                                 step_size=STEP_SIZE)

# Audio
audio_stream = AudioStream(sample_rate=SAMPLE_RATE,
                           chunk_size=CHUNK_SIZE,
                           fft_size=FFT_SIZE,
                           format=FORMAT,
                           channels=CHANNELS,)
analyzer = AudioAnalyzer(n_bins=SKETCH_WIDTH,
                         sample_rate=SAMPLE_RATE,
                         fft_size=FFT_SIZE,
                         min_db=-60,
                         max_db=30,)


def shift_channel(img, dx, dy):
    """
    Shift a single color channel in an image by a given x and y offset.

    Args:
        img (np.ndarray): 2D array representing a color channel.
        dx (float): Horizontal shift.
        dy (float): Vertical shift.

    Returns:
        np.ndarray: Shifted image channel.
    """
```

```
303:     rows, cols = img.shape
304:     M = np.float32([[1, 0, dx], [0, 1, dy]])
305:     return cv2.warpAffine(img, M, (cols, rows), borderMode=cv2.BORDER_REFLECT)
306: 
307: 
308: def chromatic_aberration_cv2(img, shift_r=(2, 0), shift_g=(0, 0), shift_b=(-2, 0)):
309:     """
310:     Apply a chromatic aberration effect to an image by shifting color channels.
311: 
312:     Args:
313:         img (np.ndarray): RGB image.
314:         shift_r (tuple): (dx, dy) shift for red channel.
315:         shift_g (tuple): (dx, dy) shift for green channel.
316:         shift_b (tuple): (dx, dy) shift for blue channel.
317: 
318:     Returns:
319:         np.ndarray: Image with chromatic aberration applied.
320:     """
321:     r = shift_channel(img[..., 0], *shift_r)
322:     g = shift_channel(img[..., 1], *shift_g)
323:     b = shift_channel(img[..., 2], *shift_b)
324:     return np.stack([r, g, b], axis=-1)
325: 
326: 
327: def create_frame(scale_factor=3, sigmaY=1):
328:     """
329:     Create a single frame of the animation by combining noise and audio data.
330: 
331:     This function samples Simplex noise and uses the smoothed audio spectrum
332:     to drive the color mapping. It optionally applies chromatic aberration
333:     when a beat is detected.
334: 
335:     Args:
336:         scale_factor (float): Controls dynamic range of the audio energy influence.
337:         sigmaY (float): Currently unused, placeholder for potential Y-blur effects.
338: 
339:     Returns:
340:         np.ndarray: RGB frame to be displayed on the LED matrix.
341:     """
342:     global t
343:     t += STEP_SIZE
344: 
345:     # Get audio data
346:     analyzer.update(audio_stream.buffer)
347:     energy = np.ones((SKETCH_HEIGHT, SKETCH_WIDTH)) * analyzer.smoothed_spectrum[np.newaxis,
:]
348:     energy = np.power(energy, scale_factor)
349: 
350:     # Get noise data
351:     noise = noise_generator.get_frame()
352: 
353:     # Map noise and energy to color
354:     y_indices = py5.remap(noise, -1, 1, 0, SKETCH_HEIGHT - 1).astype(int)
```

```
    x_indices = py5.remap(energy, 0, 1, 0, SKETCH_WIDTH - 1).astype(int)
    out = colormap.get_frame(y_indices, x_indices)

    strength = analyzer.chrom_aberration_strength
    if strength > 0.01:
        print(f'{strength}')
        max_shift = 50  # max pixels of chromatic aberration
        shift_x = int(strength * max_shift)
        shift_y = int(strength * max_shift)

        out = chromatic_aberration_cv2(out, shift_r=(shift_x, shift_y), shift_b=(-shift_x,
-shift_y))

    return out


def setup():
    """
    py5 setup function, called once before the animation loop begins.

    Initializes canvas size and frame rate and starts the audio stream.
    """
    py5.size(SKETCH_WIDTH, SKETCH_HEIGHT)
    py5.frame_rate(FRAME_RATE)
    audio_stream.start()
    print("Running animation...")


def draw():
    """
    py5 draw function, called once per frame.

    Updates audio stream, generates a new frame, and displays it on both
    the py5 sketch and the physical LED matrix.
    """
    # Update audio buffer
    audio_stream.update()

    # Update display
    out = create_frame()
    py5.set_np_pixels(out, bands='RGB')
    py5.load_np_pixels()
    matrix.update(py5.np_pixels)


if __name__ == '__main__':
    print(f'{np.__version__=}')

    # Run the animation
    try:
        print('Press CTRL-C to stop.')
        py5.run_sketch()
    except KeyboardInterrupt:
```

```
407:         sys.exit(0)
408:
```

## sound.py

```
File: sound.py
001: """
002: Real-time audio processing for visualization and beat detection.
003:
004: This module provides tools for capturing and analyzing real-time audio input,
005: primarily for use in LED-based or generative art visualizations. It includes two
006: main classes:
007:
008: - `AudioStream`: Interfaces with a microphone (or other default audio input)
009:   using PyAudio to continuously read audio samples into a rolling buffer.
010:
011: - `AudioAnalyzer`: Applies windowed FFT analysis on buffered audio data, maps
012:   it to a logarithmic frequency scale, and exposes a normalized spectrum for
013:   visualization. It also includes simple beat detection logic based on energy
014:   surges in a selected frequency band.
015:
016: Typical usage involves:
017: 1. Creating and starting an `AudioStream`.
018: 2. Periodically calling `AudioStream.update()` to fill the buffer.
019: 3. Passing the buffer to `AudioAnalyzer.update()` to compute spectral data.
020: 4. Using `AudioAnalyzer.smoothed_spectrum` or `beat_detected` in a visual output.
021: """
022:
023: from collections import deque
024:
025: import pyaudio
026: import numpy as np
027: from scipy import signal
028:
029:
030: class AudioStream:
031:     """
032:     A class to manage real-time audio input for visualization.
033:
034:     Captures audio data from the default input device using PyAudio,
035:     stores it in a circular buffer, and exposes methods to start, stop,
036:     and update the audio stream.
037:
038:     Attributes:
039:         sample_rate (int): Sampling rate in Hz.
040:         chunk_size (int): Number of audio frames per buffer read.
041:         fft_size (int): Size of the cicular buffer for FFT processing.
042:         format (int): PyAudio format (e.g., pyaudio.paFloat32).
043:         channels (int): Number of audio channels (1 = mono, 2 = stereo).
044:         pa (pyaudio.PyAudio): PyAudio interface instance.
045:         stream (pyaudio.Stream): Active audio stream.
046:         buffer (deque): Circular buffer of recent audio samples.
```

```
047:     """
048:     def __init__(self, sample_rate=44100, chunk_size=1024, fft_size=1024,
format=pyaudio.paFloat32, channels=1):
049:         """
050:         Initialize the audio stream with specified parameters.
051:
052:         Args:
053:             sample_rate (int): Audio sample rate in Hz.
054:             chunk_size (int): Size of audio chunks to read.
055:             fft_size (int): Size of the internal audio buffer for analysis.
056:             format (int): Audio format from PyAudio.
057:             channels (int): Number of audio channels to use.
058:         """
059:         self.sample_rate = sample_rate
060:         self.chunk_size = chunk_size
061:         self.format = format
062:         self.channels = channels
063:
064:         self.pa = pyaudio.PyAudio()
065:         self.stream = None
066:         self.buffer = deque(np.zeros(fft_size, dtype=np.float32), maxlen=fft_size)
067:
068:     def start(self):
069:         """
070:         Start the audio input stream using the system's default input device.
071:
072:         Attempts to open the audio stream and prints the input device parameters.
073:         """
074:         try:
075:             host_params = self.pa.get_default_input_device_info()
076:         except IOError as e:
077:             print(f'{e}')
078:
079:         print(f'Default input device parameters: {host_params}')
080:
081:         self.stream = self.pa.open(
082:             format=self.format,
083:             channels=self.channels,
084:             rate=self.sample_rate,
085:             input=True,
086:             frames_per_buffer=self.chunk_size,
087:         )
088:
089:     def stop(self):
090:         """
091:         Stop and close the audio input stream, and terminate the PyAudio instance.
092:         """
093:         if self.stream is not None:
094:             self.stream.stop_stream()
095:             self.stream.close()
096:         self.pa.terminate()
097:
098:     def update(self):
```

```
099:         """
100:         Read a chunk of audio data from the stream and update the rolling buffer.
101:
102:         Returns:
103:             float: A fallback value (0.5) if an error occurs during reading.
104:         """
105:         try:
106:             audio_data = np.frombuffer(
107:                 self.stream.read(self.chunk_size, exception_on_overflow=False),
108:                 dtype=np.float32
109:             )
110:             self.buffer.extend(audio_data)
111:         except Exception as e:
112:             print(f"Audio error: {e}")
113:             return 0.5
114:
115:
116: class AudioAnalyzer:
117:      """
118:     Logarithmic audio spectrum analyzer for LED visualizations and beat detection.
119:
120:     This class processes real-time audio input using FFT and maps it onto a
121:     logarithmic frequency scale. It supports beat detection and provides smoothed
122:     spectral data suitable for driving LED matrix visualizations.
123:
124:     Attributes:
125:         n_bins (int): Number of bins in the output spectrum.
126:         min_freq (float): Minimum frequency for analysis (Hz).
127:         max_freq (float): Maximum frequency for analysis (Hz).
128:         sample_rate (int): Audio sample rate (Hz).
129:         fft_size (int): FFT size (number of samples per transform).
130:         window (np.ndarray): Window function to apply before FFT.
131:         min_db (float): Minimum dB level for normalization.
132:         max_db (float): Maximum dB level for normalization.
133:         fft_freqs (np.ndarray): Linear FFT frequency bins.
134:         log_freqs (np.ndarray): Logarithmically spaced frequency bins.
135:         spectrum (np.ndarray): Current spectrum values (normalized).
136:         alpha (float): Smoothing factor for exponential moving average.
137:         smoothed_spectrum (np.ndarray): Smoothed spectrum values.
138:         min_rms_threshold (float): Threshold to skip beat detection when audio is too quiet.
139:         beat_band (np.ndarray): Indices of frequency bins used for beat detection.
140:         beat_history (deque): History of beat band energy values.
141:         beat_cooldown_frames (int): Number of frames to wait before detecting another beat.
142:         beat_cooldown_counter (int): Current cooldown frame count.
143:         beat_detected (bool): Flag indicating whether a beat was detected.
144:         chrom_aberration_strength (float): Value to control visual effect intensity on beat.
145:     """
146:     def __init__(self, n_bins=256, min_freq=20, max_freq=20000, sample_rate=44100,
fft_size=1024, window='hann', min_db=-60, max_db=30):
147:         """
148:         Initialize the AudioAnalyzer with frequency and analysis parameters.
149:
150:         Args:
```

```
151:             n_bins (int): Number of frequency bins in the output.
152:             min_freq (float): Minimum frequency to analyze.
153:             max_freq (float): Maximum frequency to analyze.
154:             sample_rate (int): Sampling rate of the audio input.
155:             fft_size (int): Number of samples per FFT window.
156:             window (str): Window function name to reduce spectral leakage.
157:             min_db (float): Minimum decibel level for normalization.
158:             max_db (float): Maximum decibel level for normalization.
159:         """
160:         self.n_bins = n_bins
161:         self.min_freq = min_freq
162:         self.max_freq = max_freq
163:         self.sample_rate = sample_rate
164:         self.fft_size = fft_size
165:         self.window = signal.get_window(window, fft_size)
166:         self.min_db = min_db
167:         self.max_db = max_db
168: 
169:         self.fft_freqs = np.fft.rfftfreq(fft_size, d=1/sample_rate)
170:         self.log_freqs = self.create_log_frequency_bins(
171:             n_bins // 2, min_freq, max_freq, sample_rate, fft_size
172:         )
173: 
174:         # Initialize buffer for storing frequency data
175:         self.spectrum = np.zeros(n_bins)
176: 
177:         # Animation parameters
178:         self.alpha = 0.5
179:         self.smoothed_spectrum = np.zeros(n_bins)
180:         self.chrom_aberration_strength = 0.0
181: 
182:         # Beat detection parameters
183:         self.min_rms_threshold = 0.6
184:         self.beat_band = self.get_band_indices(200, 1000)
185:         self.beat_history = deque(maxlen=60)
186:         self.beat_cooldown_frames = 20
187:         self.beat_cooldown_counter = 0
188:         self.beat_detected = False
189: 
190:     def create_log_frequency_bins(self, n_bins, min_freq=20, max_freq=20000,
sample_rate=44100, fft_size=1024):
191:         """
192:         Create logarithmically spaced frequency bins suitable for audio analysis.
193: 
194:         Args:
195:             n_bins (int) : Number of frequency bins to create
196:             min_freq (float) : Minimum frequency in Hz, default 20 Hz
197:             max_freq (float) : Maximum frequency in Hz, default 20 kHz
198:             sample_rate (int) : Audio sample rate in Hz, default 44.1 kHz
199: 
200:         Returns:
201:             - center_frequencies: Array of bin center frequencies
202:         """
```

```
        # Compute number of octaves and bins per octave
        n_octaves = np.log2(max_freq / min_freq)
        bins_per_octave = n_bins / n_octaves

        # Create logarithmically spaced frequencies
        n_edges = int(np.ceil(bins_per_octave * n_octaves)) + 1
        log_bin_edges = min_freq * (2 ** (np.arange(n_edges) / bins_per_octave))
        center_frequencies = np.sqrt(log_bin_edges[:-1] * log_bin_edges[1:])

        return center_frequencies

    def get_band_indices(self, low, high):
        """
        Get indices of frequency bins that fall within a given frequency range.

        Args:
            low (float): Lower bound of the frequency band (Hz).
            high (float): Upper bound of the frequency band (Hz).

        Returns:
            np.ndarray: Array of indices corresponding to the frequency band.
        """
        return np.where((self.log_freqs >= low) & (self.log_freqs <= high))[0]

    def compute_band_energy(self, spectrum, indices):
        """
        Compute the average energy of a given frequency band in the spectrum.

        Args:
            spectrum (np.ndarray): The full spectrum array.
            indices (np.ndarray): Indices of the frequency band.

        Returns:
            float: Average energy in the specified band.
        """
        return np.mean(spectrum[indices]) if len(indices) > 0 else 0

    def detect_beat(self, energy, history, threshold):
        """
        Determine whether a beat is detected based on energy in a frequency band.

        Args:
            energy (float): Current energy value in the beat band.
            history (deque): Rolling history of energy values.
            threshold (float): Beat detection threshold multiplier.

        Returns:
            bool: True if beat is detected, False otherwise.
        """
        if len(history) < history.maxlen:
            history.append(energy)
            return False
        avg_energy = np.mean(history)
```

```
256:         history.append(energy)
257:         return energy > threshold * avg_energy
258: 
259:     def update(self, audio_data):
260:         """
261:         Update the spectrum and beat detection using a new block of audio data.
262: 
263:         Args:
264:             audio_data (np.ndarray): 1D array of audio samples.
265: 
266:         Effects:
267:             - Updates internal spectrum and smoothed_spectrum buffers.
268:             - Sets `beat_detected` flag if a beat is detected.
269:             - Updates `chrom_aberration_strength` for visual effects.
270:         """
271:         # Zero-padding
272:         if len(audio_data) < self.fft_size:
273:             audio_data = np.pad(
274:                 audio_data,
275:                 (0, self.fft_size - len(audio_data)),
276:                 mode='constant',
277:                 constant_values=0
278:         )
279: 
280:         # Apply window function and compute FFT
281:         windowed = audio_data * self.window
282:         fft_magnitude = np.abs(np.fft.rfft(windowed, n=self.fft_size))
283:         fft_power = fft_magnitude ** 2
284: 
285:         # Interpolate to log frequency scale
286:         log_power = np.interp(self.log_freqs, self.fft_freqs, fft_power)
287: 
288:         # Convert magnitude to dB scale
289:         log_db = 10 * np.log10(log_power + 1e-10)
290:         log_db = np.clip(log_db, self.min_db, self.max_db)
291: 
292:         # Reflect across y-axis for symmetry
293:         log_db = np.concatenate([log_db[::-1], log_db])
294: 
295:         # Normalize and update spectrum buffer
296:         self.spectrum = (log_db - self.min_db) / (self.max_db - self.min_db)
297:         self.spectrum = np.clip(self.spectrum, 0, 1)
298:         self.smoothed_spectrum = (
299:             self.alpha * self.smoothed_spectrum
300:             + (1 - self.alpha) * self.spectrum
301:         )
302: 
303:         # Beat detection
304:         beat_detected = False
305:         overall_rms = np.sqrt(np.mean(self.spectrum ** 2))
306:         if overall_rms > self.min_rms_threshold:
307:             print(f'overall_rms: {overall_rms:.4f}')
308:             beat_energy = self.compute_band_energy(self.spectrum, self.beat_band)
```

```
309:             energy_history = np.mean(self.beat_history)
310:             beat_threshold = energy_history * 1.8
311:             beat_detected = self.detect_beat(beat_energy, self.beat_history, beat_threshold)
312:
313:         if self.beat_cooldown_counter > 0:
314:             self.beat_cooldown_counter -= 1
315:
316:         self.beat_detected = False
317:
318:         if self.beat_cooldown_counter == 0 and beat_detected:
319:             self.beat_detected = True
320:             self.beat_cooldown_counter = self.beat_cooldown_frames
321:             self.chrom_aberration_strength = 1.0
322:
323:         self.chrom_aberration_strength *= 0.8
324:
```