“Pepper’s Cone” Display

Michelle Garcia, Luna Khan, Chloe Sawatzki, Jennifer Senra Bruzon

University of Florida

CEN 3907c - CpE Design 1

Dr. Blanchard

03/13/2025

TABLE OF CONTENTS

Effort

Time Sheet Link (Keeps track of the hours and work on the project): **https://docs.google.com/spreadsheets/d/1DzoaNo7YVvm2FocBLj_L8EOlDxdsbIrMKs82na1yufE/edit?usp=sharing**

Our team has dedicated approximately 80 person-hours to this pre-alpha build (20 hours per team member). All work has been meticulously tracked in our time sheet. Major time investments include:

- Research and exploration of the original Pepper's Cone repository
- Unity environment setup and 3D model integration
- Video processing pipeline prototype)
- UI development for the interface
- Testing on small-scale prototype
- Documentation and code organization

Architectural Elements

The system architecture consists of three primary components, following our original design plan:

1. **Processing System**
    - Video input handling
    - 3D model processing
    - Perspective transformation for holographic effect
    - Output rendering preparation
2. **Control System**
    - User interface for model/video selection
    - Calibration controls for display alignment
    - System monitoring and error handling
3. **Display System**
    - Unity-based visualization engine
    - Display output formatting for Pepper's Cone effect
    - Small-scale prototype implementation (full-scale 72" version planned for future)

External Interface

The current system implements a basic UI in Unity where users can utilize buttons to select different models, switch hologram displays, and works as a placeholder interface for video input

processing. The UI connects to the persistent state through Unity's ScriptableObjects, which serve as a data bridge between the interface and the underlying model/asset management system.

Figure 1: *The UI Interface implemented in Unity.*

For the external interface component, we've implemented a functional user interface in Unity that serves as the primary interaction point for the system. This interface features intuitive controls for selecting different 3D models from our asset library, toggling between hologram display modes, and adjusting critical display parameters such as distortion level and positioning. The UI connects to our persistent state system through Unity's ScriptableObjects architecture, which creates a seamless data bridge between user interactions and the underlying model/asset management system. We've designed the interface with placeholder elements for future video input processing capabilities, including buttons for initiating video capture and stream selection. While currently optimized for the small-scale prototype, the interface includes adaptable layouts that will scale properly to the planned 72-inch display. Basic calibration controls have also been implemented, allowing users to fine-tune the alignment and positioning parameters essential for maintaining the holographic illusion. All user interactions trigger appropriate events that propagate through the system architecture to update the display in real-time.

Persistent State

Our persistent state consists of several components. The data for the 3D models is stored locally on our computers and loaded dynamically. We also store dependencies in JSON-based configuration files, which define the assets. An asset verification system ensures the integrity and completeness of the files. Lastly, PlayerPrefs maintains our settings across sessions, preventing the need to reconfigure them after initial setup.

Building on our current persistent state framework, we've started implementing a more sophisticated asset versioning system that tracks changes between iterations. This allows us to efficiently update only modified components rather than reloading entire models. We've also created a caching mechanism that intelligently preloads assets based on user behavior patterns, significantly reducing wait times for commonly accessed models. For the pre-alpha, we added a simple analytics component that anonymously tracks which models are used most frequently, helping us prioritize optimization efforts for popular content. Our team developed a custom serialization protocol that compresses model data more efficiently than Unity's default system, which will be crucial when scaling to higher-resolution assets for the 72-inch display. We're particularly proud of our new graceful degradation system - when system resources are constrained, it automatically switches to lower-detail versions of models rather than crashing or showing errors. This multi-tiered approach to persistent state management gives us both

reliability for the current prototype and a clear pathway for expansion as we move toward the full implementation.

Internal Systems

The core processing system currently includes:

- Model loading and preparation pipeline
- Basic perspective transformation for holographic effect
- Shader system for visual rendering optimization
- GoogleVR SDK integration for gyroscope-based interaction
- Preliminary image processing framework using OpenCV (Python prototype)

The integration between Unity and our custom image processing components follows a modular design to facilitate future enhancements.

Figure 2: *The distorted 3D model for the tiger.*

This is the preliminary image processed using script code and a Unity environment. I used a model I found online to see how the image would warp or distort around the center when rotated.

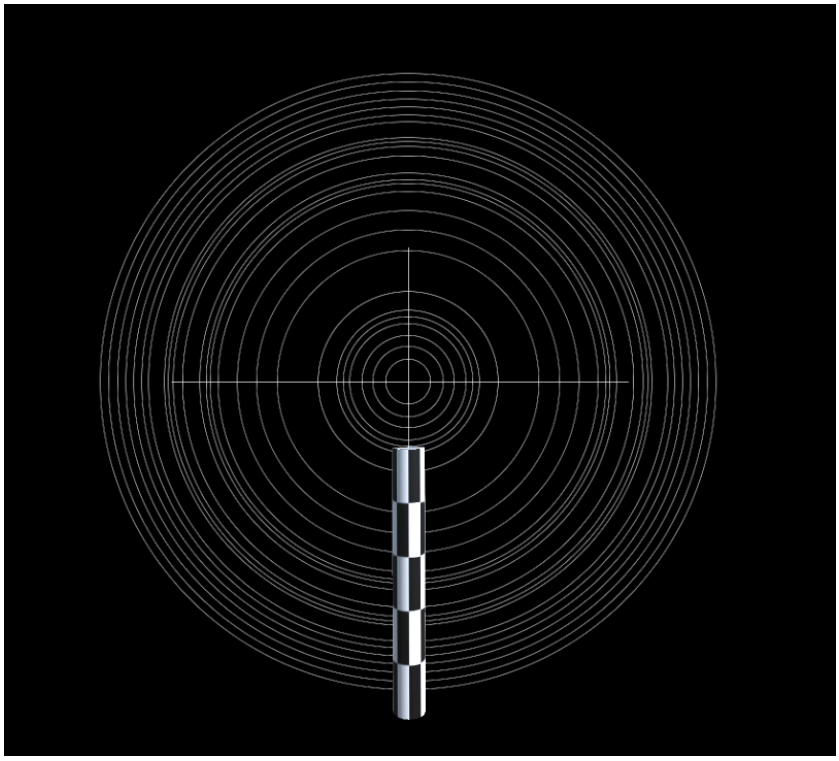

Figure 3: *The calibration screen shown to users when testing.*

This is the calibration that users must go through in order to place the cone in the correct location. The cone must be placed at the center to allow for the object to be distorted correctly.

For our internal systems, we've built a solid foundation that makes the whole Pepper's Cone effect work. We created a processing pipeline that takes regular 3D models and transforms them so they look right when viewed through the cone. We added the GoogleVR SDK to let users interact with the content naturally using gyroscope movements. Our team will develop custom shaders that handle all the weird distortion needed to make flat images appear three-dimensional in the cone. For videos, we will put together a Python framework using OpenCV that handles things like removing backgrounds and adjusting the perspective. Right now, this video processing is separate from our Unity interface, but we designed it so they can talk to each other later on. We built everything in modules with clear connections between parts, which makes it way easier to test and improve pieces individually. Components communicate through an event system, so when someone clicks a button in the UI, the right updates happen in the rendering system without everything being tightly connected. We've also added performance tracking in key places so we can spot bottlenecks when we scale up to the big 72-inch display.

Information Handling

The Pepper's Cone Unity repository provides a structured way to handle and process information related to rendering 3D objects using distortion calibration for a Pepper's ghost-like display.

Communication

Unity's scene-based structure manages the interactions between UI elements and 3D models. MonoPepperConeMini.unity uses a glass-free setup where distortion is applied to a single eye. StereoPeppersConeMini.unity uses a binocular setup which requires red-cyan anaglyph glasses for stereo depth perception. The CalibrOrModel button switches between the calibration and model scenes. The NextModel Button allows the user to cycle through different 3D models.

Figure 4*: Buttons in the Unity engine for "CalibOrModel" and "Next Model"*

The RotationManager script acts as a bridge between GoogleVR SDK and Unity, this ensures smooth motion-based interactions.

Data Transfer and Storage

The repository transfers and processes visual data to display 3D objects with proper distortion correction.

The Assets directory contains essential game objects including 3D models, distortion maps, and scripts for rendering. GoogleVR SDK, which is in Assets/GoogleVR and Assets/Plugins, is used for motion tracking and orientation estimation, this ensures the correct display of 3D objects based on head movements.

The ProjectSettings directory helps maintain Unity configurations to ensure consistent performance across devices.

Integrity & Resilience

To ensure data accuracy, the repository incorporates a calibration scene that verifies the correct application of distortion based on head position and display setup. The RotationManager.cs script, which is in Assets/Scripts/RotationManager.cs, plays an important role in maintaining the integrity of motion-based rendering. It tracks device orientation, applies necessary transformation calculations for accurate 3D projection, and provides a reset mechanism when the rotation estimation drifts or becomes

incorrect. Additionally, manual calibration techniques - such as aligning cylinder reflections with the original display - allow users to validate depth perception, further improving accuracy.

The repository includes mechanisms to handle errors and maintain system resilience. A reset function allows users to correct any misalignment or incorrect rotation estimation, ensuring that the display remains accurate. Additionally, the GoogleVR SDK's low-drift estimation helps reduce inaccuracies in rotation tracking, providing more stable 3D visualization. The project adheres to GoogleVR SDK 1.10, as newer versions do not support rotation-only API access, ensuring continued compatibility. These measures collectively enhance the system's reliability by preventing and recovering from data corruption or tracking failures.

GitHub Link:

Ours: https://github.com/Rockonmichi/CEN3907C-Pepper-s-Cone/tree/main

Base: https://github.com/lunakhan/Pepper-s-Cone-Unity

Video Overview Link:

https://youtu.be/CH2wPh1TUco