PHYSICALIZED GAME INTERFACES

DEVELOPMENT LOG

MELODY HENRICH

AUGUST 21, 2024

Introduction

This document is the development log outlined in the Higher Level Design Document (HLDD) submitted along with the initial directed study request.[1] As such, this document contains the following:

- Week-by-week progress updates
- Meeting notes from update meetings with faculty supervisor
- Playtest notes from any playtests conducted during the project

[1] The plan of study is available in the same GitHub repository as this log.

The week-by-week progress updates will take the form of notes on what deliverables were and weren't completed during the week, and the plan for the following week.[2] Additionally, any changes in scope or project direction will be documented in the weekly logs, and later worked into the finalized HLDD.

[2] These weekly updates will begin from after the first meeting with my faculty advisor Prof. Murphy on 2024-07-12.

The meeting notes will be initialy taken in bullet form. The same process will be used for playtest notes.

Week 1

Deliverables completed

- Initial hardware setup
- Prelim. software architecture design
- Documentation layout
- Timeline established
- Set up software project

Deliverables to-do

- Redo software design for new deliverables
- Read camera input into software
- Detect AprilTags
- Drawing hard-coded shapes

Calibration moved from hand-calibration to hybrid.

Aspect ratios between the camera and projector match, but there is a slight fisheye in the camera and there will always be a slight offset. In order to achieve maximum accuracy, a vector offset must be *combined* with a skew-like projection tranformation to account for any distortion in the image.

Design change removing the editor.

Due to scope concerns, the editor side of the project will be scrapped in favor of manual authoring of data files. This is now a reach goal, allowing project goals to remain MVP-oriented and achievable within the timeframe of the directed study.

Week 2

Deliverables completed

- Set up development environment
- Start setting up Rust project & libraries

Deliverables to-do

- Make an initial project to test engine functionality
- Start work on code infrastructure

Development environment set up.

To maximize build reproducibility, I've opted to use a nix flake to set the development environment for the project. nix is a package manager that allows declarative instantiation of dependencies, meaning anyone on any machine with the nix package manager should be able to reproduce the development environment, down to the specific package versions used. Additionally, the project is planned to be built in Rust, which has many features for cross-platform and asynchronous coding.

Week 3

Deliverables completed

- Library compilation debugging
- Webcam library PR investigation
- Plan AprilTag library integration

Deliverables to-do

- Finish debugging libraries
- Pull webcam library from PR branch

Library compilation debugging.

There are several issues with some of the libraries required for this project. There are some issues getting the main engine/graphics framework to compile, although they're mostly errors from low-level build processes that just need a few added dependencies to get working again. The webcam library nokhwa, however, seems to have an upstream issue where they depend on a deprecated version of a critical package. There's a pull request open, and I plan on using that revision locally for the time being.

Week 4

Deliverables completed

- Library compilation debugging

- Experiment with Python testbed
- Testing with bevy_nokhwa plugin

Deliverables to-do

- Set up engine architecture for when libraries are working
- Start working on rendering side

Experiments with Python testbed.

Since there are still some lingering library issues that are preventing work on the main Rust repo from materially progressing, I experimented with a Python implementation I found on GitHub Gists for tracking AprilTags to try to look ahead and get a better sense of any issues I might deal with. In the process, I had to debug yet another library (this time OpenCV), which will probably come in handy for when I need to implement similar behavior in Rust, since OpenCV is a pretty ubiquitous library for computer vision.

Week 5

Deliverables completed

- Engine architecture setup
- Structure loading architecture
- Module separation for webcam/tracking/rendering

Deliverables to-do

- Get webcam and engine renderer to interop
- Implement tracking

Engine architecture.

The engine used for this project, bevy, is a well-known Rust-based game engine that is famous for its ECS.[3] The infrastructure provided by bevy made setting up the basic framework for the eventual product a breeze, and thinking about the functionality of the project in terms of systems and resources allowed for more plan-

[3] Entity component system

ning at this stage, even when certain things aren't yet complete/ working.

Structure loading architecture.

"Structures" are things like shapes or sprites that will be tracked to a tag's position. The setup in-engine for these consists of a resource-based list of StructureDescriptors that allows for meshes and sprites to be created and associated with identifiers ad-hoc in a startup system. This will make reach features like json loading much easier to implement, and increases the extensibility of the code overall.

Week 6

Deliverables completed

- Tag tracking
- ID -> Shape mapping
- **BONUS:** "Drawing"

Tag tracking.

Tag tracking itself turned out to be very simple, the real trouble was in converting the received webcam image's data into a format the AprilTag library could understand. This involved getting the raw data out of the webcam, converting it from sRGB to grayscale luma,[4] and funneling that raw data into the AprilTag library. Once an image is processed, the library gives a list of each tag ID found, where it is in the image, and how certain the model is that it correctly identified the tag.[5]

[4] This process was complex, since the size and format of the image had to be preserved during the conversion.

[5] Filtering based on this certainty value ended up being necessary to avoid false-positive tracks from spontaneously appearing.

Mapping tag IDs to structures.

Since the structures were already set up in-code to be associated with numeric IDs, tracking the structures was a simple matter of iterating through detected tags and modifying their associated structures' positions. There were a few other necessary quality-of-life tweaks, such as adding a delay between when a tag stops being visible and its associated structure becoming invisible to reduce flickering due to uncertain tracks.

Meeting Notes

Kickoff Meeting 2024-07-12

- Work begins!
- Most things are already pre-planned
- Scrapping editor, slimming down scope
  - Reach goal
- Think about 3D applications
- Timeline looks good

Meeting 2024-08-02

- Library issues w/ webcam & graphics libraries
- Handheld calibration tests went well
- Reach out and start asking around about issues / help with the project

Meeting 2024-08-20

- Library issues resolved
- Reached out to engine users/developers, got back a very promising solution
- All the infrastructure is set up for when the libraries do eventually work

Playtest Notes

Boston Indies

- Setup outdoors is difficult/impossible without propping up the support beam in a somewhat awkward position
- Tracking is *heavily* dependent on lighting conditions, especially when using a glossy phone screen.
    - Printing is definitely a better choice
- People immediately saw the potential for game design with PGI
- Significant “wow” factor since the projector gives an illusion of movement
- Look into Boston Tech Poetics for more tech/art related events