Final Project Proposal

Sumner Evans, Robbie Merillat, Sam Sartor

October 11, 2017

1 Introduction

VR Technology has been developing rapidly in the 21st century. Current solutions such as Unity attempt to use old programming languages and paradigms to implement VR environments and thus limit developers' abilities to create new and unique environments. With every cutting edge technology, new paradigms and design patterns must be invented. This project intends to explore and implement these paradigms and design patterns.

2 Objectives

The goal of this project is to experiment with a variety of paradigms and design patterns to see which work best for implementing VR environments. To do this, we will implement four different VR environments where users can move around and interact with a variety of virtual objects. Each team member will build one environment individually and the fourth will be built by the whole team and will connect the individually-built environments.

2.1 Environments and Interactions

These are the concepts for our four environments and the interactions that can occur in each environment:

Sumner	"Snowflakes" — a winter-themed environment where users can create their own structures by manipulating snow blocks.
Robbie	"Lets get Physical" — an environment where you can throw lots of objects around and see them interact in a physically-accurate manner.
Sam	"Workbench" — The goal of this environment is to find a general pattern for VR element design that can apply to both diegetic (semi-realistic) and spacial (purely abstract) items. Specifically, it includes logic for moving, placing, snapping, and recombining arbitrary VR elements. These elements are split into 2 categories. Some are purely abstract user interface items (spacial), others are blocks that appear more concrete (diegetic). The user has the ability to recombine blocks into moving objects such as miniature vehicles. More interestingly, the user can recombine interface elements into static control "devices" for these objects. For example, the user could attach 4 rocket engines together, and control it by combining the 4 thrust controls into a crude joystick.
Team	"VRsh" — a VR shell where users can interact with a variety of widgets as well as open other programs (environments).

2.2 C-MAPP Event Readiness

Our goal is to have this project ready for the C-MAPP event in January. To do this, we must have a finished product and report by the end of this semester.

3 Proposed Implementation

In order for vrsh to be a useful shell, it must be capable of performing the same actions as existing shells such as bash. As such, this project is implemented partially as a bash wrapper. Although many commands such as `cd` and `ls` are replaced entirely, more complex programs (e.g. `git`) are executed through command templates. These templates take the outputs from specified input widgets and combine them into a valid bash command to be executed.

TODO: Robbie and Sumner take a crack at this

4 Plan of Action

This project will have four main stages that correlate with the due dates for the individual assignments and the final project milestones.

10/03	**Individual Assignments** — Building the individual environments
11/03	**Milestone I** — Building the VRsh environment
11/17	**Milestone II** — Integrating the VRsh and individual environments
12/08	**Final Code Submission** — Final touches to the VRsh environment.

In addition to the above code submissions, we will also produce a final report along with the Final Code Submission describing the work that we did and the lessons we learned while implementing the project.

5 References and Dependencies

Because of the performance requirements and complexity of virtual reality software, we decided to use the Rust programming language for all the projects in this independent study.

Our individual and final projects will all be built using our own shared, open-source library called "flight", which will be continually updated to reflect what we learn while developing these projects. As of writing, flight incorporates our own VR hardware interface, several custom rendering environments, mesh manipulation tools, and a few asset loading utilities.

Although most of the planned features of our shell can be implemented using the Rust standard libraries, there are several other tools that we will be depending on. The most notable are gfx (a GPU graphics wrapper), nalgebra (vector and matrix math), ncollide (geometric operations such as collisions and intersections), nphysics (physics engine), OpenVR (Valve's VR hardware SDK), image (raster image loader), and futures (asynchronous events). With the exception of OpenVR, all these libraries can be found on `crates.io`.