# 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

The following sections describe the concepts for our four environments and the interactions that can occur in each environment.

### 2.1.1 Sumner — Snowflakes

Snowflakes will be a winter-themed environment where users can create their own structures by manipulating snow blocks. The environment will have "snow" on the ground and winter themed items in the environment such as snowmen. The user will be able to create new snow blocks by pointing both controllers at the ground and pulling both of the triggers. This action will create a snow block being held between the users hands. When the user releases the triggers, the snow block will fall in a physically-accurate manner with collision detection.

### 2.1.2 Robbie — Let's Get Physical

An environment where you can throw lots of objects around and see them interact in a physically-accurate manner.

### 2.1.3 Sam — Workbench

### 2.1.4 Team — VRsh

A VR shell where users can interact with a variety of widgets as well as open other programs (environments).

More than just widgets, vrsh is a graphical way of sending commands. My thought is that we actually create some sort of templeting engine, that allows us to wrap traditional, UNIX-y commands with vrsh widget systems.

# 3 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.

10/18 **Individual Assignments** — Demo 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.

## 3.1 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.

# 4 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`.