# Final Project Proposal v. 1.2

Sumner Evans, Robbie Merillat, Sam Sartor

October 20, 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 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) objects. Specifically, it includes logic for moving, placing, recombining, and programmatically updating 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).

Ideally, the developed design pattern could be very useful in creating CAD applications, such as the early "sketchpad" program. However, as the goal of this environment is simply to prove the practicality of the pattern, it gives far less granular control to the user. More like Lego than SolidWorks.

### 2.1.2 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 user's hands. When the user releases the triggers, the snow block will fall in a physically-accurate manner with collision detection (the blocks will stack on top of each other and the floor). To rearrange the blocks, the user can point to an existing block and pull both triggers. This will teleport the block to between the user's hands. Snowflakes will be able to load a saved environment including

block locations as dictated by VRsh. The user will also be able to create snowballs by pointing a single controller at the ground and pulling the trigger. They can "throw" the snowball at the blocks in the environment to knock the blocks down. One stretch goals is to include sound (e.g. Christmas music) to enhance the winter experience.

### 2.1.3 Robbie — Let's Get Physical

This project provides a physically "real" environment. That is an environment that applies some of the fundamental laws of physics as we know them. Simple gravitational pull, acceleration, and collisions which in turn includes momentum. To achieve this, the Rust `nphysics3d` library is used to provide physics utility functions to the program. Various objects will be placed around the environment that can be picked up, thrown, dropped, and/or interacted with in other ways. One example of such interactions would be shooting a bow and arrow.

# 3 Proposed Implementation — VRsh (Team)

For our final project, our goal is to reimagine file I/O for a VR environment. To do this, we will implement a VR "shell" (VRsh). (Note that, for our purposes, *state* refers to the current configuration of objects in an environment and any other environment metadata.) This environment will be a user-customizable environment with many widgets and allow the user to:

1. **Navigate to other environments using a "program picker".**
   This will allow the user to go to the environments built as individual projects.

2. **Persist the state of another environment using an intuitive UI specialized for VR.**
   This will be the file I/O equivalent of saving a file.

3. **Reopen the saved state of an environment using an intuitive UI specialized for VR.**
   This will be the file I/O equivalent of opening a saved file.

4. **Customize the environment state using design motifs that are specialized for VR.**
   This will allow users to manipulate the widgets and change environment state such as the background picture.

The primary design principle that VRsh, and the environments that interact with it, will employ is that *VRsh will store all environment state information.* VRsh will be the ultimate source of "truth" with regards to environment state. This means that programs do not themselves store their state, rather, they are passed state from VRsh and then modify it. To help facilitate this design pattern, programs will register their default state with VRsh. When the user wants to create a "clean slate" in an environment, they will grab the

environment from the program picker. VRsh will open the environment passing the default state.

One major stretch goal of VRsh will be to implement two modes: sitting and standing. These modes will be essentially the same, but the sitting mode will include the ability to teleport to different locations in the environment using a "hookshot"/"fishing pole" motif.

# 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.

10/18 **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. Polished individual projects.

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.

## 4.1 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.

# 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`.