# Final Project Report

Jonathan Sumner Evans*, Robinson Merillat†, and Sam Sartor‡
Department of Computer Science, Colorado School of Mines
Golden, Colorado
Email: *jonathanevans@mines.edu, †rdmerillat@mines.edu, ‡ssartor@mines.edu

***Abstract*—Virtual Reality (VR) Technology has been developing rapidly over the past decade. 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. In this paper, we discuss a deferred immediate mode (DIM) application architecture suitable for implementing large virtual reality applications. We present a library which utilizes this architecture and a few case studies of this library in use.**

## I. Introduction

The Virtual Reality (VR) market is growing rapidly. The International Data Corporation (IDC) projects that revenues for the combined Augmented Reality (AR) and Virtual Reality markets will grow from $5.2 billion in 2016 to more than $162 billion in 2020 [1]. This flourishing new industry has created an exciting new field of Software Engineering with great potential for revolutionary new design paradigms and program architectures.

Most current frameworks and libraries attempt to apply old design paradigms and program architectures that are well-suited for 2D user interfaces and rendering 3D environments to a flat screen to virtual reality. Although these endeavors have been successful, they do not fully explore potential new design paradigms and program architectures.

Our goal is to find a system that provides a modern, fast, and practical approach to virtual reality development. Specifically, we require that this system meets the following criteria.

### A. Performant

Virtual Reality requires a very high frame rate. A virtual reality program must be capable of generating 90 frames per second (fps); thus the effective required frame rate is 180fps. Achieving this frame rate is resource intensive and requires highly efficient and optimized code. Additionally, multithreading is imperative so that long-running processes can occur without blocking the user interface. This is unlike a traditional desktop user interface where blocking the UI process for a second does not have a major affect on the usability of the program. These factors require a VR framework that is highly performant and multi-threadable.

### B. Natural User Interface

A *natural user interface* is an interface that can be used without the need for a controller [2]. Although current VR systems utilize hand-held controllers, they emulate this goal much better than desktop, mobile, and web applications. For our discussion of these natural user interfaces, we define the following terms.

**Definition 1.** *A* planar UI *is a user interface where components are organized along a 2D surface.*

**Definition 2.** *A* spacial UI *is a user interface where components are organized within a 3D space.*

**Definition 3.** *A* uniform UI *element is a user interface element where the functionality of the element depends on at most 2 dimensions.*

**Definition 4.** *A* volumetric UI *element is a user interface element where the functionality of the element depends on 3 dimensions.*

All of these user interfaces and user interface elements can be natural user interfaces. For example, a physical-word planar UI would be a painter's palate. Spacial UI is found in <TODO>. Uniform UIs are found in the physical world in the form of levers and sliders. An example of volumetric UI in the physical world is <TODO>.

Because Virtual Reality environments are inherently 3D, they make spacial UI and volumetric UI elements convenient and practical for the first time. Thus, the ability to create spacial UI and volumetric UI elements rather than merely to planar UI and uniform UI is a high priority.

### C. Programmable

We need a framework that prioritizes application development over game development—a software development kit first and foremost. Features such as character rigging, pathfinding, and high-fidelity materials are not necessary for non-entertainment software. For example, existing frameworks for classical application development (e.g. HTML and GTK) do not include any of these features. The easy programmatic definition and use of user interface components must take priority.

### D. Flexible

As evidenced by the massive breadth of desktop, mobile, and web applications, the number of software applications is nearly infinite. The virtual reality field is no different, and VR application architectures must be scalable, adaptable, and <TODO> in order to enable this diversity. In other words, an architecture must not be opinionated about which types of application should be created with it.

### E. Modular

A direct result of flexibility is modularity—a flexible architecture cannot be a monolith. Many current VR architectures have features such as pathfinding and character rigging baked in which must be included even if they are not used. We want an architecture which has a minimal feature set baked in while still allowing extensibility via addition of self-containing modules.

***

Given these criteria, we evaluate current program architectures including entity component systems, React-like architectures, and event tree architectures in Section II. We also describe how these architectures influenced our deferred immediate mode architecture which we formally present in Section III. In Section IV we describe Flight—an implementation of a VR UI library using the referred immediate mode architecture. Then we present three case studies of Flight and the deferred immediate mode architecture in use in real applications in Section V and conclude in Section VI.

## II. Evolution of Deferred Immediate Mode

### A. Entity Component System

### B. React-Like Architecture

### C. Event Tree

## III. Deferred Immediate Mode

### A. Deferred

### B. Immediate Mode

## IV. Flight

Flight is our implementation of a VR UI library using the DIM architecture. It is written in Rust and is designed to be highly modular. It uses Rust `FnOnce` closures to implement the deferred aspect of the DIM architecture and resolves state using a *guru* system.

### A. Language

### B. Dependencies

The Rust dependency manager, Cargo, allows programmers to easy include third-party libraries from https://crates.io. To avoid duplicating work by other programmers, we utilized a few external libraries for Flight. The main dependencies are listed below.

- **`WebVR`**: an OpenVR wrapper built by Mozilla for the Servo browser.
- **`nalgebra`**: library for geometric/volumetric operations and queries.
- **`gfx`**:

## References

[1] IDC. *Worldwide Revenues for Augmented and Virtual Reality Forecast to Reach $162 Billion in 2020, According to IDC*. Aug. 15, 2016. URL: https://www.idc.com/getdoc.jsp?containerId=prUS41676216.

[2] Alexandre Wimmers et al. *Natural User Interface and Virtual Reality Integration in Video Games*. Mar. 31, 2015. URL: http://csiflabs.cs.ucdavis.edu/~cs193/cs193_2015/23_design.pdf.