ML2VR: Providing MATLAB Users an Easy Transition to Virtual Reality and Immersive Interactivity

David J. Zielinski[1] Duke University

Ryan P. McMahan[2] University of Texas at Dallas

Wenjie Lu[3] Duke University

Silvia Ferrari[4] Duke University

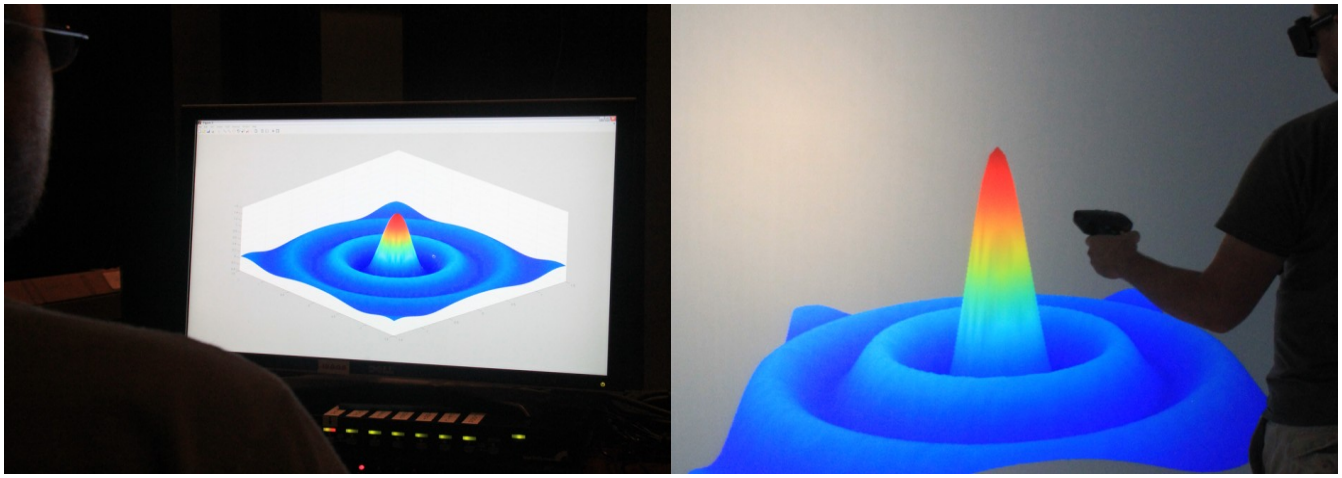

Fig. 1. Using ML2VR, a MATLAB user transitions from viewing a 3D surface plot on a desktop computer (left) to viewing and interacting with the plot in a virtual reality system (right).

ABSTRACT—MATLAB is a popular computational system and programming environment that is used in numerous engineering and science programs in the United States. One feature of MATLAB is the capability to generate 3D visualizations, which can be used to visualize scientific data or even to simulate engineering models and processes. Unfortunately, MATLAB provides only limited interactivity for these visualizations. As a solution to this problem, we have developed a software system that easily integrates with MATLAB scripts to provide the capability to view visualizations and interact with them in virtual reality (VR) systems. We call this system "ML2VR" and expect it will introduce more users to VR by enabling a large population of MATLAB programmers to easily transition to immersive systems. We will describe the system architecture of ML2VR and report on a successful case study involving the use of ML2VR.

Index Terms—Virtual reality, MATLAB, 3D visualizations, immersive interactivity.

1 INTRODUCTION

MATLAB, commercially produced by The MathWorks, Inc., is a computational software system and programming environment that allows for mathematical calculations, complex data processing, and visualizations of functions and data. We desired a way for MATLAB users to easily access head-mounted displays and CAVE-type systems, which often have a cluster-based architecture (e.g. one screen per computer). We first examined several existing solutions. *Simulink 3D Animation* is another product from MathWorks that provides an interface for linking MATLAB algorithms to 3D graphical objects [1]. However, the product is targeted at desktop systems and not cluster-based immersive VR systems. We also investigated prior research in porting graphical desktop applications to cluster-based systems. Two well-known projects are *WireGL* [2] and *Chromium*. Both of these software systems were designed to distribute an application's OpenGL calls to a cluster of computers for rendering by using an OpenGL intercept method (i.e., using a replacement OpenGL driver to process calls before they are passed on to the true OpenGL driver). However, neither of these libraries supported developing VR applications, as both lacked support for handling devices like six-degrees-of-freedom (6-DOF) trackers.

So we decided to develop a cross-platform, open-source software system that easily integrates with MATLAB. We are hoping our new system will introduce more users to VR by enabling those MATLAB users with access to VR systems (e.g., CAVEs and head-mounted displays) to easily transition to them from their desktops.

[1]email: djzielin@duke.edu
[2]email: rymcmaha@utdallas.edu
[3]email: wl72@duke.edu
[4]email: sferrari@duke.edu

2 METHOD

2.1 OpenGL Intercept

When designing ML2VR, we decided to utilize a preexisting VR software library, Syzygy [3], to provide the features of cluster-based rendering and access to VR input devices. ML2VR takes advantage of the fact that MATLAB can use OpenGL for rendering. By intercepting the OpenGL calls made by a running instance of

```
vr = vr_interface(ServerIP,ServerPort);

while 1
 [event,btn,pos]=vr.get_button_event();
 if event==1 && btn==0
  % modify data here
 end

 clf;
 surf(dataX,dataY,dataZ);
 drawnow;
end
```

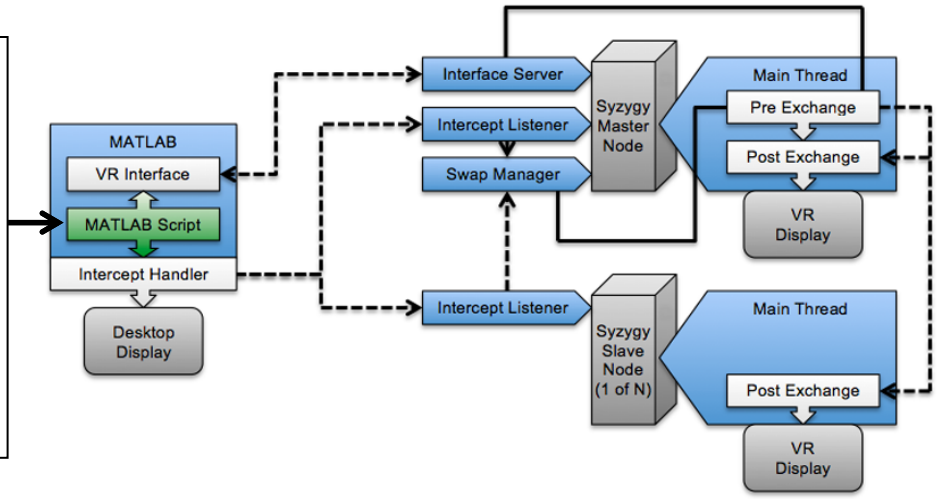

Fig. 2. A Matlab script (left) that uses ML2VR to the change the visualization based on wand button events. To the right, the System architecture of ML2VR: The Intercept Handler intercepts MATLAB's OpenGL calls and distributes content to all of the nodes in the Syzygy master/slave framework. On reception, the Intercept Listeners notify the Swap Manager, which synchronizes the swapping of content.

MATLAB, the system's "Intercept Handler" (a modified version of GLTrace) is able to communicate content data to "Intercept Listeners", which are threads running on each Syzygy node. After receiving and storing the data, a ready signal is sent to the "Swap Manager". The Swap Manager then utilizes Syzygy's variable distribution method to distribute the ready signal. Due to our careful construction of the content distribution and synchronization methods, we were able to have the head-based rendering of the Syzygy application run at a high frame rate and not be slowed down by MATLAB's lower frame rate. This is especially useful when dealing with larger scenes.

2.2 Interface Server

To provide more interactivity and high fidelity techniques for MATLAB, we designed the system architecture of ML2VR to provide access to the VR input devices supported by the Syzygy framework. This access is provided via the "Interface Server" and a MATLAB script that we call the "VR Interface". The MATLAB user can query the VR Interface to determine if any button events (i.e., presses and releases) have occurred. Every event also contains the position and orientation of the wand device when that event occurred. By tracking the state of the wand when they occur, events provide more-accurate interactions than simple polling, since the current wand position usually has changed since the button press. Overall this allows the MATLAB user to construct scripts as a loop: query user activity, update the simulation, and then render the resulting scene.

3 Case Study: Sensor Path Planning Simulation

As an initial case study of using ML2VR, we have been working with roboticists to extend their MATLAB applications beyond the desktop. One such application is a simulation involving sensor path planning to navigate a robotic sensor to a predefined target. The simulation uses an artificial potential function [4] to plan the motion of the robot. By utilizing ML2VR our colleagues were able to judge the dynamic capabilities of their sensor path-planning algorithm by manipulating the obstacles during simulation, while immersed in the VR system. We consider this case study of using ML2VR to have been a success.

4 Future Work

We would like to support more of the OpenGL geometries, lighting, and textures in our intercept code. Our source code will be made available at http://sourceforge.net/projects/ml2vr/

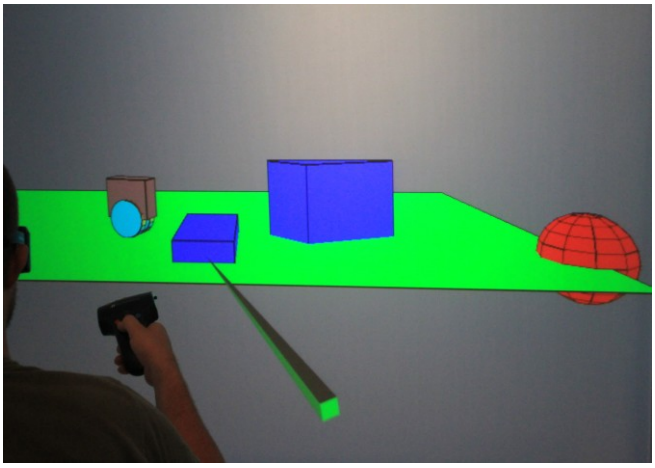

Fig. 3. ML2VR is used to view and dynamically manipulate the obstacles within the sensor path planning simulation.

Acknowledgments

This work has been supported by the National Science Foundation, under IGERT Grant No. DGE-1068871. The authors would like to thank Rachael B. Brady for her involvement in the early stages of the project, and also Sarah V. Goetz and Eric E. Monson for helping with the videos, figures, and diagrams presented with this project.

References

[1] "Simulink 3D Animation," 2012; http://www.mathworks.com/products/3d-animation/.

[2] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan, "WireGL: A Scalable Graphics System for Clusters," in *ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2001, pp. 129-140.

[3] B. Schaeffer, and C. Goudeseune, "Syzygy: Native PC Cluster VR," in *IEEE Virtual Reality (VR)*, 2003, pp. 15-22.

[4] G. Zhang, and S. Ferrari, "An Adaptive Artificial Potential Function Approach for Geometric Sensing," in *IEEE Decision and Control*, 2009, pp. 7903-7910.