Intercept Tags: Enhancing Intercept-based Systems

David J. Zielinski, Regis Kopper*
Duke immersive Virtual Environment
Duke University

Ryan P. McMahan†
Dept. of Computer Science
University of Texas at Dallas

Wenjie Lu, Silvia Ferrari‡
Dept. of Mechanical Engineering
Duke University

Abstract

In some virtual reality (VR) systems, OpenGL intercept methods are used to capture and render a desktop application's OpenGL calls within an immersive display. These systems often suffer from lower frame rates due to network bandwidth limitations, implementation of the intercept routine, and in some cases, the intercepted application's frame rate. To mitigate these issues and to enhance intercept-based systems in other ways, we present intercept tags, which are OpenGL geometries that are interpreted instead of rendered. We have identified and developed several uses for intercept tags, including hand-off interactions, display techniques, and visual enhancements. To demonstrate the value of intercept tags, we conducted a user study to compare a simple virtual hand technique implemented with and without intercept tags. Our results show that intercept tags significantly improve user performance and experience.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality;

Keywords: virtual reality, intercept tags, intercept-based systems

1 Introduction

In some VR systems, an OpenGL intercept method is used to capture and render a desktop application's OpenGL calls within an immersive display, such as a CAVE or a head-mounted display (HMD). Examples of such intercept-based VR systems include Chromium [Humphreys et al. 2002], TechViz [Raffin et al. 2006], and some Allosphere applications [Höllerer et al. 2007]. More recently, an open-source project called ML2VR has been used to interact with MATLAB visualizations in VR systems through OpenGL intercept [Zielinski et al. 2013].

A common problem for intercept-based VR systems is slower frame rates. In cluster-based systems, this issue most commonly stems from network bandwidth limitations and how the intercept routine is implemented to synchronize the OpenGL frames across the cluster. In other systems, lower frame rates are a result of the intercepted application's frame rate. For instance, MATLAB and other computational software packages are known to render visualizations slower than most graphical applications due to data-intensive computations [Choy and Edelman 2005].

In this paper, we present the concept of intercept tags, which are OpenGL geometry calls that are not rendered, but instead interpreted as scene information, cues, or even function calls. Intercept tags can be used in several ways to enhance intercept-based systems, including increasing the overall frame rate. We categorize these uses as hand-off techniques, display techniques, and visual enhancements. Hand-off techniques simply allow the intercepted application to hand off control of an interaction technique to the intercepting application, which can operate, through optimized rendering routines and lack of simulation computation load, at a faster frame rate then the intercepted application. The display techniques we discuss also improve frame rates by reducing the transmission of redundant or expected geometry data. Finally, we explain how intercept tags can be used to improve animations through interpolation and enhance visuals by providing advanced shaders.

To show the benefits of our new intercept tags, we conducted a user study comparing a virtual hand technique implemented in a MATLAB simulation with and without intercept tags. Our results show that intercept tags can significantly improve user performance and experience. We conclude by discussing the advantages and limitations of intercept tags.

2 Related Work

Using an OpenGL intercept technique is one common method for enabling cluster-based VR systems [Raffin et al. 2006]. A common intercept approach is to use a replacement driver to process calls before they are passed on to the true OpenGL driver. This allows a system to intercept and distribute an application's OpenGL calls to a cluster of computers for rendering.

One of the first intercept-based systems was WireGL, which used an intercept method to submit OpenGL commands to one or more pipeservers [Humphreys et al. 2001]. WireGL shortly evolved into the better-known Chromium, a software library for distributing streams of graphics calls in a cluster-based system [Humphreys et al. 2002]. Chromium has since been used to intercept the graphics calls of applications, including closed-source ones, to be displayed in immersive VR systems. One such example is the Allosphere at UCSB [Höllerer et al. 2007].

More-recent intercept-based systems have also been developed. One open-source project called ML2VR has been developed to allow users to easily port their MATLAB visualizations to a VR system and add 3D interactions [Zielinski et al. 2013]. In a research-oriented project, OpenGL interception was also used to create an image-parallel distribution system for more-efficient ray tracing [Brownlee et al. 2013].

In addition to duplicating the intercept OpenGL stream, one can also modify the contents of the stream. One example of this is HijackGL [Mohr and Gleicher 2002]. By modifying the incoming stream, HijackGL can create several stylized rendering effects (e.g., pencil sketch, blue print, cartoon).

Though intercept-based systems are still being used and developed, a common problem for many of these systems is slower frame rates. A major cause of this problem is the limitations of network bandwidths among nodes. This is sometimes further complicated by computationally intense applications that have low frame rates prior to being intercepted.

*e-mail:[djzielin,regis.kopper]@duke.edu
†e-mail:rymcmaha@utdallas.edu
‡e-mail:[wl72,sferrari]@duke.edu

3 Concept of Intercept Tags

The original inspiration for our development of intercept tags was the need to reduce the latency of interacting with a MATLAB-based robotics simulation that we were intercepting and displaying in a CAVE application. In addition to intercepting MATLAB's OpenGL calls, our system architecture allowed MATLAB to access the event data of the CAVE's 6-degree-of-freedom (DOF) wand. We used this data to implement a simple virtual hand technique for the user to move objects around within the simulation. A major limitation of this method was the simulation's slow frame rate, which caused noticeable latency in the virtual hand technique. Improving the simulation's frame rate was not feasible without significantly reworking its computations.

As an alternative solution, we determined that MATLAB should hand off control of the virtual hand technique to the intercepting CAVE application, which ran at a faster frame rate than the robotics simulation. One issue was that MATLAB could not easily inform the CAVE application which object to control without using complex messages that referenced the OpenGL stream. We determined that such complexity could be avoided if the control messages were embedded in the OpenGL stream. Thus intercept tags were born.

Intercept tags are predefined OpenGL geometry calls that are checked for during the decoding of an intercepted OpenGL stream and are interpreted as scene information, cues, or even function calls, instead of being rendered.

Because intercept tags are interpreted and not rendered when decoded, the OpenGL geometry chosen to define a tag must be carefully chosen. In practice, we have chosen geometries that would never appear in our intercepted applications, such as empty polygons (e.g., a triangle with all three vertices at (0, 0, 0)). We can also vary the specific values of the points in the empty polygon to encode different types of tags (e.g. (1,1,1)).

Like HTML or XML tags, intercept tags are normally used in pairs, with a start tag and an end tag enclosing a block of OpenGL calls. When the first tag is decoded, it provides additional information about the upcoming OpenGL calls or signifies that they should be specially handled. When the second tag is decoded, it is interpreted as the end of the information or special handling. As with XML tags, intercept tags can also be nested.

4 Uses of Intercept Tags

Through the course of our research, we have identified several potential uses of intercept tags. We discuss those uses here.

4.1 Hand-Off Techniques

Hand-off techniques use intercept tags to signal the intercepting application to use a particular interaction technique on a portion of the OpenGL scene. This method allows the interaction technique to function at the frame rate of the intercepting application, as opposed to the frame rate of the slower intercepted application.

4.1.1 Hand-Off Manipulation

The virtual hand technique, our inspiration for intercept tags, essentially consists of three phases: selection of the desired object, manipulation of its position and orientation, and object release. In our MATLAB-based simulation, selection and release of the object functioned well thanks to the event-based input data available from the wand. But, due to the slow frame rate of the simulation, manipulation of the object's position and orientation was noticeably affected by latency.

We have used intercept tags to hand off the manipulation of the object from our MATLAB simulation to the CAVE application intercepting the OpenGL calls. Once the simulation determines that an object is selected, a pair of intercept tags is used to enclose the OpenGL geometry of that object. Upon intercepting the first hand-off tag, the CAVE application uses a glPushMatrix and an appropriate glMultMatrix command to translate and rotate the upcoming geometry relative to the wand's current position. Once the second hand-off tag is intercepted, the CAVE application uses glPopMatrix to leave the remaining scene geometry unchanged. When the object is released, the simulation stops calling the intercept tags, effectively ending hand-off manipulation. While the object has been handed off, the MATLAB simulation can still update some object properties of the object (e.g. color, size, shape).

4.1.2 Hand-Off Slice Plane

A slice plane is a useful technique that enables the user to control their view of the structure and internal volume of geometric objects and shapes. Using intercept tags, we have implemented a slice plane technique. Like hand-off manipulation, a pair of slice-plane tags is used to enclose the intended OpenGL geometry, and the intercepting application employs a slice plane technique at its faster frame rate.

A major advantage of using intercept tags to implement the slice plane technique is the ability to control which geometry is sliceable. Most slice plane implementations normally affect the entire scene, which makes it difficult to slice one object without affecting the view of nearby objects. By using intercept tags to enclose the desired geometry, the entire scene or only particular objects can be designated as sliceable by the slice plane technique. Figure 1 demonstrates this unique capability of our technique.

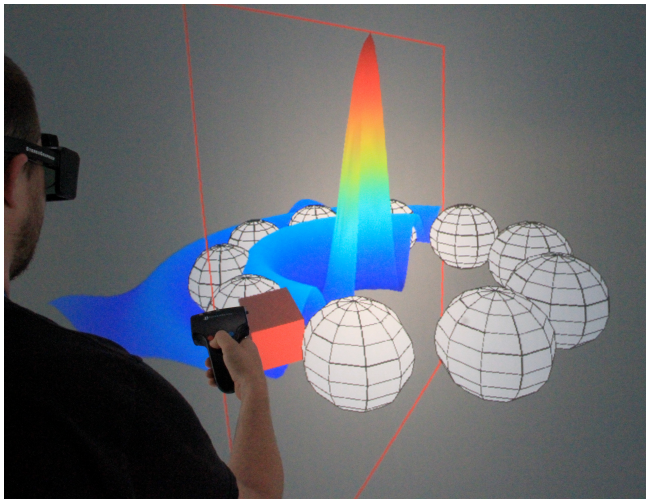

Figure 1: *Hand-off slice plane technique. The rainbow surface is the only object affected by the slice plane, which is handled by the intercepting application..*

4.2 Display Techniques

In addition to handing off control of interaction techniques to the intercepting application, intercept tags can also be used to specify how and when parts of the OpenGL scene should be displayed by the intercepting application.

4.2.1 Display Lists

In some applications, portions of the intercepted scene may be static from frame to frame. Using normal intercept techniques,

these static geometries are intercepted, encoded, and decoded every frame. This decreases the overall frame rate of the entire system due to transmitting the same geometries every frame.

Our solution is to use intercept tags to create display lists, which behave as geometry caches that are transmitted once. If a portion of the OpenGL scene is known to be static in upcoming frames, a unique pair of display-list tags can be used to enclose the scene portion. Upon interpreting the tags, the intercepting application caches and renders the enclosed geometry. In later frames, the intercepted application calls the display-list tag instead of the geometry calls associated with the static scene. When the intercepting application interprets the single display-list tag, it renders the associated geometries from its cache, thus negating the need to transmit the same geometries every frame and increasing the overall frame rate.

4.2.2 Level of Detail

Intercept tags can also be used to specify different levels of detail (LOD). Instead of requiring the intercepted application to determine the current level of detail every frame for each object, intercept tags can be used in the initial frame to pass all LOD instances of an object to the intercepting application to be cached. After the initial lengthy frame, the intercepting application then has the capability to adjust the LOD for an object based on the user's current position, given the cached LOD geometry.

4.3 Visual Enhancements

Finally, we have also determined that intercept tags can be used to visually enhance an intercepted application's graphics.

4.3.1 Interpolated Animations

A noticeable problem with our MATLAB simulations is that the slower frame rates cause time-based animations to appear clunky and not smooth. The positions of objects during these animations are correct when a frame is first intercepted, but until the next intercepted frame, the positions quickly become outdated.

A remedy to these clunky animations is to use intercept tags to interpolate animated movements. In the simulation, a pair of interpolation tags is used to enclose the moving geometry. When the tags are interpreted by the intercepting application, the current geometry and time are cached. A linear interpolation method is then used to render the animation based on the previously cached geometry, the most recently cached geometry, and the time that has passed since the last intercepted frame. Because the intercepting application runs faster than the simulation, the result is a smoother-looking animation.

4.3.2 Advanced Shaders

Another use of intercept tags for visual enhancement is to call advanced shaders. Some types of intercepted applications, such as MATLAB, do not support advanced shaders. As a solution, we propose using intercept tags to call for advanced shaders to be applied to the OpenGL scene by the intercepting application. This would allow for lighting, bump mapping, shadows, specular highlights, and other types of shaders to be incorporated into even the most basic of intercepted OpenGL applications.

5 Evaluation of Intercept Tags

To demonstrate how intercept tags can improve the quality of the user experience, we conducted a user study comparing our original virtual hand implementation without intercept tags to our current implementation that utilizes hand-off manipulations.

5.1 Experiment Overview

We evaluated the task of placing a solid cube completely inside a wireframe cube that varied in size (how much larger than the solid cube) on a graphically complex MATLAB simulation, which ran at about 5 frames per second (fps). The goal of the evaluation was to assess usability and performance benefit of hand-off manipulation over a traditional virtual hand technique.

5.1.1 Design

The study used a within-subjects design with two independent variables – interaction method (original virtual hand, hand-off manipulation) and target size (10%, 20%, and 40% wireframe cube larger than solid cube). Time to successfully complete the task and number of object drops (clutches) were measured. The order of interaction method was counterbalanced and each set of three target sizes was performed 18 times for each method. The order of target size within each set was predetermined randomly. Thus, each participant performed a total of 108 placement tasks.

5.1.2 Apparatus

We used a six-sided CAVE-like display to perform the experiment. Tracking was provided by an Intersense IS-900 tracker, which tracked the participant's head and wand input device. Active shutter glasses provided 3D stereoscopic graphics. The intercepted application was written in MATLAB, and ML2VR [Zielinski et al. 2013] was used as the OpenGL intercept library to render graphics in the CAVE. The MATLAB simulation ran at about 5fps. In the original virtual hand condition, the intercepting application in the CAVE ran at the same frame rate as the MATLAB simulation (~5fps). For the hand-off manipulation interaction method, the cube dragging updated at about 55fps. Selection and drop events were handled by MATLAB at 5fps.

5.1.3 Participants

Fourteen unpaid participants (3 female), with ages ranging from 22 to 60 years (M=38.5, SD=14.3) volunteered to participate in the study. Three participants were left-handed and all participants held the wand with their preferred hand.

5.1.4 Procedure

After signing the informed consent form, participants filled out a demographics and background questionnaire. After that, participants were introduced to the display and practiced in a standard application for 5 minutes to get used to the CAVE environment. Participants were then instructed to stand and remain on a piece of carpet placed in the center of the CAVE and practice one experimental task, after which data collection began. After completing each set of 54 tasks per interaction method, participants took a break and filled out presence, usability and simulator sickness questionnaires. At the end of the final session, participants were asked to determine their preferred technique and to provide additional feedback.

5.2 Results

Participants had at most 30 seconds to complete each placement task, after which the task would be deemed incomplete and the next task would begin. 4.8% of the tasks timed out and were removed from the analysis.

We performed a factorial ANOVA with repeated measures on both dependent variables: mean time to complete a task and mean number of clutches. For questionnaires, we conducted paired-samples t-tests on the sum scores for each interaction method.

5.2.1 Time

Hand-off manipulation (M=4.9, SE=.26) was significantly faster than the traditional virtual hand (M=9.76, SE=.77) overall ($F_{1,13}$=44.085, p<.0001). There was a main effect of target size overall for time ($F_{2,26}$=188.879, p<.0001). Pairwise comparisons showed that larger targets yielded significantly lower task times (means 11.8s, 6.1 and 4.2). A significant interaction effect between target size and interaction method was observed for time ($F_{2,26}$=12.07, p<.0001). Pairwise comparisons showed that hand-off manipulation was always faster than the traditional virtual hand, but the mean difference decreased as targets got larger (figure 2).

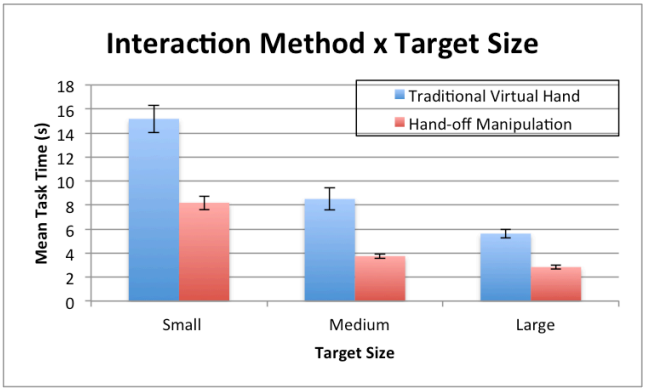

Figure 2: *Interaction between target size and interaction method.*

5.2.2 Clutches

Hand-off manipulation (M=.272, SE=.04) yielded significantly fewer clutches than the traditional virtual hand (M=.587, SE=.13) overall ($F_{1,13}$=7.509, p<.05). There was a main effect of target size overall for clutches ($F_{2,13}$=40.177, p<.0001). Pairwise comparisons showed that larger targets yielded significantly fewer clutches (means respectively .85, .28 and .15). No significant interaction effect between target size and interaction method was observed for clutches ($F_{2,26}$=2.164, p=.135).

5.2.3 Questionnaires

A main effect was found for usability (t_{13}=4.426, p<.005), with hand-off manipulation rated significantly more usable than the traditional technique. Hand-off manipulation caused significantly higher presence (t_{13}=−2.621, p<.05) and significantly lower simulator sickness (t_{13}=−2.104, p<.001) when compared to the traditional technique. No participant reported more than slight simulator sickness in the handoff manipulation. All but one participant preferred hand-off manipulation overall. The participant who preferred the traditional technique indicated he did so because he developed a strategy that minimized clutching.

6 Discussion

We evaluated an intercept tag that improved the performance and experience of an interaction task running at a low frame rate MATLAB simulation. The improved performance with hand-off manipulation indicates that, even with the button events being transmitted at the slow rate of the simulation, the high quality perceptual feedback loop provided by the smooth dragging allowed users to complete the task faster than when the whole application ran at the simulation rate. The artifact of simulator sickness caused by low frame rate immersive systems, verified by reports of moderate to high sickness in the traditional technique, was mitigated by hand-off manipulation, where no participants reported a high level of simulator sickness. Another important aspect of immersive systems is presence, which was positively affected by the intercept tag. Conversely, participants rated the hand-off manipulation technique as significantly more usable than the original virtual hand technique.

There are some limitations to the advantages of intercept tags. The most obvious limitation is that intercept tags only work in intercept-based systems. While these types of systems are not the most popular or used, there are some VR systems that require intercept-based methods to display an application immersively. Another limitation of intercept tags is that the order of OpenGL calls must be controllable. Intercept tags will not function properly if they do not enclose their affected geometry. In most low-level systems, such as MATLAB, it is easy to define the order of OpenGL commands. In higher-level systems, such as game engines, it may not be possible to define the order, or it may require additional functionality, such as Unity's subshader tags.

7 Conclusion

We have proposed the concept of intercept tags as a way to enhance intercept-based systems. We empirically demonstrated the benefits of a common application of intercept tags – handing off control of the interaction to the intercepting application. Apart from the empirical evaluation, we presented and discussed hand-off and display techniques and visual enhancements provided by intercept tags. The frame rate of the intercepting application can be increased by decoupling it from the original application (hand-off techniques). Performance of the intercepted application can be improved by using display techniques such as display lists and levels of detail. Further, visual enhancements, such as smoothed animations and advanced shaders can be implemented in the intercepting application through the use of intercept tags.

Acknowledgements

This work has been supported by the National Science Foundation, under IGERT Grant No. DGE-1068871.

References

BROWNLEE, C., IZE, T., AND HANSEN, C. D. 2013. Image-parallel ray tracing using opengl interception. In *Proc. EGPGV*, 65–72.

CHOY, R., AND EDELMAN, A. 2005. Parallel MATLAB: Doing it right. *Proc. IEEE 93*, 2, 331–341.

HÖLLERER, T., KUCHERA-MORIN, J., AND AMATRIAIN, X. 2007. The Allosphere: a large-scale immersive surround-view instrument. In *Proc. EDT*.

HUMPHREYS, G., ELDRIDGE, M., BUCK, I., STOLL, G., EVERETT, M., AND HANRAHAN, P. 2001. WireGL: a scalable graphics system for clusters. In *Proc. SIGGRAPH*, 129–140.

HUMPHREYS, G., HOUSTON, M., NG, R., FRANK, R., AHERN, S., KIRCHNER, P. D., AND KLOSOWSKI, J. T. 2002. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proc. SIGGRAPH*, 693–702.

MOHR, A., AND GLEICHER, M. 2002. HijackGL: reconstructing from streams for stylized rendering. In *Proc. NPAR*.

RAFFIN, B., SOARES, L., NI, T., BALL, R., SCHMIDT, G., LIVINGSTON, M., STAADT, O., AND MAY, R. 2006. PC clusters for virtual reality. In *Proc. IEEE VR*, 215–222.

ZIELINSKI, D. J., MCMAHAN, R. P., LU, W., AND FERRARI, S. 2013. ML2VR: Providing MATLAB users an easy transition to virtual reality and immersive interactivity. In *Proc. IEEE VR*.