User Interface Guidelines

v. 0.4

1. Introduction

Graphical User Interfaces (GUIs) in virtual reality present unique challenges that can be mitigated by following the guidelines in this document. This is not an exhaustive list, but provides some guidance and insight for first-time implementers of Virtual Reality GUIs (VRGUIs).

2. In a word: Stereoscopic!

If any single word can help developers understand and address the challenges of VRGUIs, it is “stereoscopic”. In VR, everything must be rendered from two points of view -- one for each eye. When designing and implementing VRGUIs, frequent consideration of this fact can help bring problems to light before they are encountered in implementation. It can also aid in understanding the fundamental constraints acting on VRGUIs. For example, stereoscopic rendering essentially makes it impossible to implement an orthographic Heads Up Display (HUD), one of the most common GUI implementations for 3D applications -- especially games.

3. The Infinity Problem

Neither orthographic projections nor HUDs in themselves are completely ruled out in VR, but their standard implementation, in which the entire HUD is presented via the same orthographic projection for each eye view, generally is.

Projecting the HUD in this manner requires the user to focus on infinity when viewing the HUD. This effectively places the HUD behind everything else that is rendered, as far as the user’s brain is concerned. This can confuse the visual system, which perceives the HUD to be further away than all other objects, despite remaining visible in front of them. This generally causes discomfort and may contribute to eyestrain.

In rare cases, an extreme disjunction between perceptual reference frames may cause spatial and temporal anomalies in which space-time is folded into an interstitial tesseract called a *hyperspatial ouroboros* - this may threaten the fabric of space-time itself.

Just kidding about the last part. Anyway, orthographic projection should be used on individual surfaces that are then rendered in world space and displayed at a reasonable distance from

the viewer. The ideal distance varies, but is usually between 1 to 3 meters. Using this method, a normal 2D GUI can be rendered and placed in the world and the user's gaze direction used as a pointing device for GUI interaction.

In general, an application should drop the idea of orthographically projecting anything directly to the screen while in VR mode. It will always be better to project onto a surface that is then placed in world space, though this provides its own set of challenges.

4. Depth In-Depth

Placing a VRGUI in world space raises the issue of depth occlusion. In many 3D applications, it is difficult, even impossible, to guarantee that there will always be enough space in front of the user's view to place a GUI without it being coincident with, or occluded by, another rendered surface. If, for instance, the user toggles a menu during gameplay, it is problematic if the menu appears behind the wall that the user is facing. It might seem that rendering without depth testing should solve the problem, but that creates a problem similar to the infinity problem, in which stereoscopic separation suggests to the user that the menu is further away than the wall, yet the menu draws on top of the wall.

There are some practical solutions to this:

- Render the VRGUI surfaces in two passes, once with depth pass and once with depth fail, using a special shader with the fail pass that stipples or blends with any surface that is closer to the view point than the VRGUI.
- Project the VRGUI surfaces onto geometry that is closer than the ideal distance. This may not solve all problems if the geometry can be so close that the VRGUI is out of the users view, but it may give them an opportunity to back away while fitting well with any game that presupposes the VRGUIs are physical projections of light into world space.
- Move the user to another scene when the VRGUI interface comes up. This might be as simple as fading the world to black as the interface fades in.
- Stop rendering the world stereoscopically, i.e., render both eye views with the same view transform, while the VRGUI is visible, then render the GUI stereoscopically but without depth testing. This will allow the VRGUI surfaces to have depth while the world appears as a 2 dimensional projection behind it.
- Treat VRGUIs as actual in-world objects. In fantasy games, a character might bring up a book or scroll, upon which the GUI is projected. In a modern setting, this might be a smart phone held in the character's hand. In other instances, a character might be required to move to a specific location in the world -- perhaps a desktop computer -- to interact with the interface. In all these cases, the application would still need to support basic functionality, such as the ability to exit and return to the system launcher, at all times.

It is generally not practical to disallow all VRGUI interaction and rendering if there is not enough room for the VRGUI surfaces, unless you have an application that never needs to display any type of menu (even configuration menus) when rendering the world view.

5. Gazing Into Virtual Reality

There is more than one way to interact with a VRGUI, but gaze tracking may be the most intuitive. The direction of the user's gaze can be used to select items in a VRGUI as if it were a mouse or a touch on a touch device. A mouse is a slightly better analogy because, unlike a touch device, the pointer can be moved around the interface without first initiating a "down" event.

Like many things in VR, the use of gaze to place a cursor has a few new properties to consider. First, when using the gaze direction to select items, it is important to have a gaze cursor that indicates where gaze has to be directed to select an item. The cursor should, like all other VR surfaces, be rendered stereoscopically. This gives the user a solid indication of where the cursor is in world space. In testing, implementations of gaze selection without a cursor or crosshair have been reported as more difficult to use and less grounded.

Second, because gaze direction moves the cursor, and because the cursor must move relative to the interface to select different items, it is not possible to present the viewer with an interface that is always within view. In one sense, this is not possible with traditional 2D GUIs either, since the user can always turn their head away from the screen, but there are differences. With a normal 2D GUI, the user does not necessarily expect to be able to interact with the interface when not looking at the device that is presenting it, but in VR the user is always looking at the device -- they just may not be looking at the VRGUI. This can allow the user to "lose" the interface and not realize it is still available and consuming their input, which can further result in confusion when the application doesn't handle input as the user expects (because they do not see a menu in their current view).

There are several approaches to handling this issue:

- Close the interface if it goes outside of some field of view from the user's perspective. This may be problematic for games if the interface pauses gameplay, as gameplay would just resume when the interface closes.
- Automatically drag the interface with the view as the user turns, either keeping the gaze cursor inside of the interface controls, or keeping it at the edge of the screen where it is still visible to the user.
- Place an icon somewhere on the periphery of the screen that indicates the user is in menu mode and then allow this icon to always track with the view.

Another frequently-unexpected issue with using a gaze cursor is how to handle default actions. In modern 2D GUIs, a button or option is often selected by default when a GUI dialog

appears - possibly the OK button on a dialog where a user is usually expected to proceed without changing current settings, or the CANCEL button on a dialog warning that a destructive action is about to be taken. However, in VRGUIs, the default action is dictated by where the gaze cursor is pointing on the interface when it appears. OK can only be the default in a VRGUI if the dialog pops up with OK under the gaze cursor. If an application does anything other than place a VRGUI directly in front of the viewer (such as placing it on the horizon plane, ignoring the current pitch), then it is not practical to have the concept of a default button, unless there is an additional form of input such as a keyboard that can be used to interact with the VRGUI.

Last Update: Nov 10, 2014