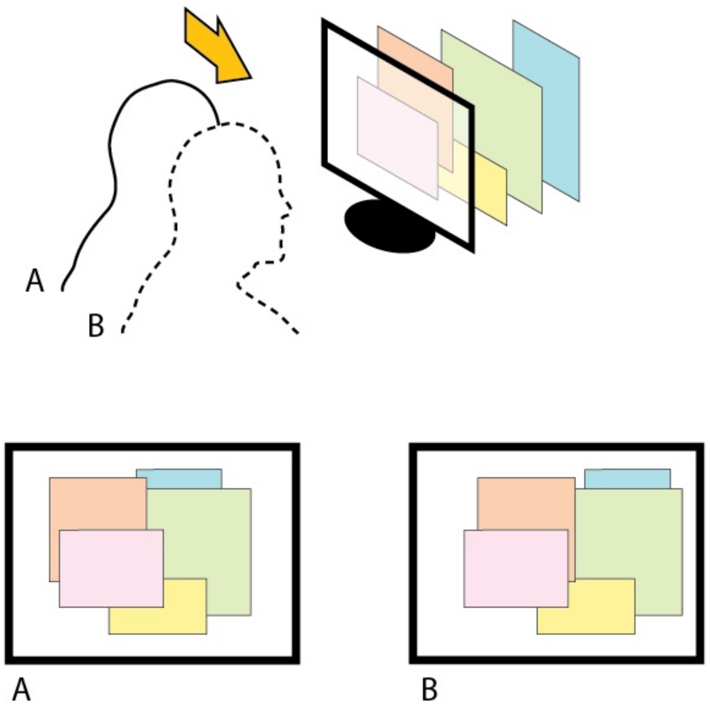
Simulating Windowed UI Depth using Parallax Motion In Response to Head Movement

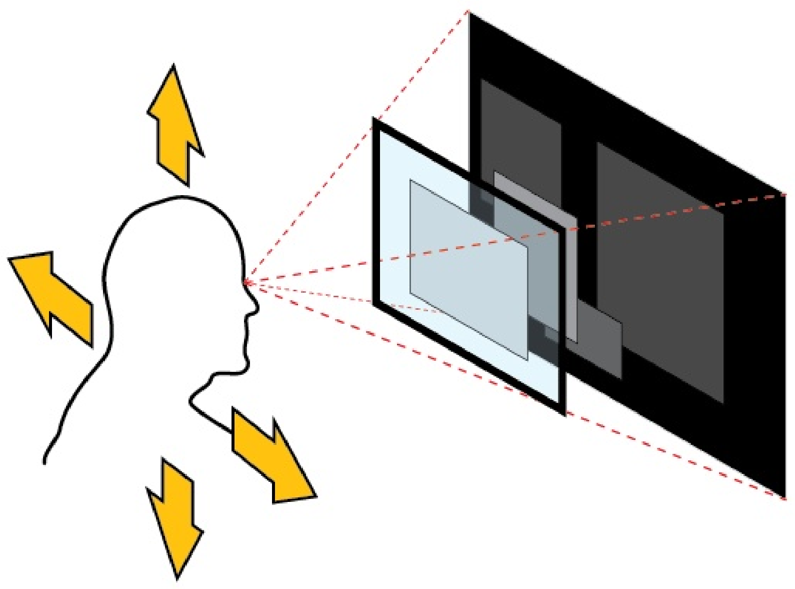


HCI 575 Project Proposal

Matt Montag
Chad Rosenbohm

Introduction

Imagine your monitor is a box one foot deep, and instead of all your applications being compressed to the front plane of the monitor, they are suspended at various depths inside this box. If you move your head to the side, you can peek around the application in front, and see a window that was hidden behind it. This is what I want to simulate with my project. The objective is to evoke a real sense of depth without resorting to stereoscopic display devices.



Perceiving Depth without Stereoscopy

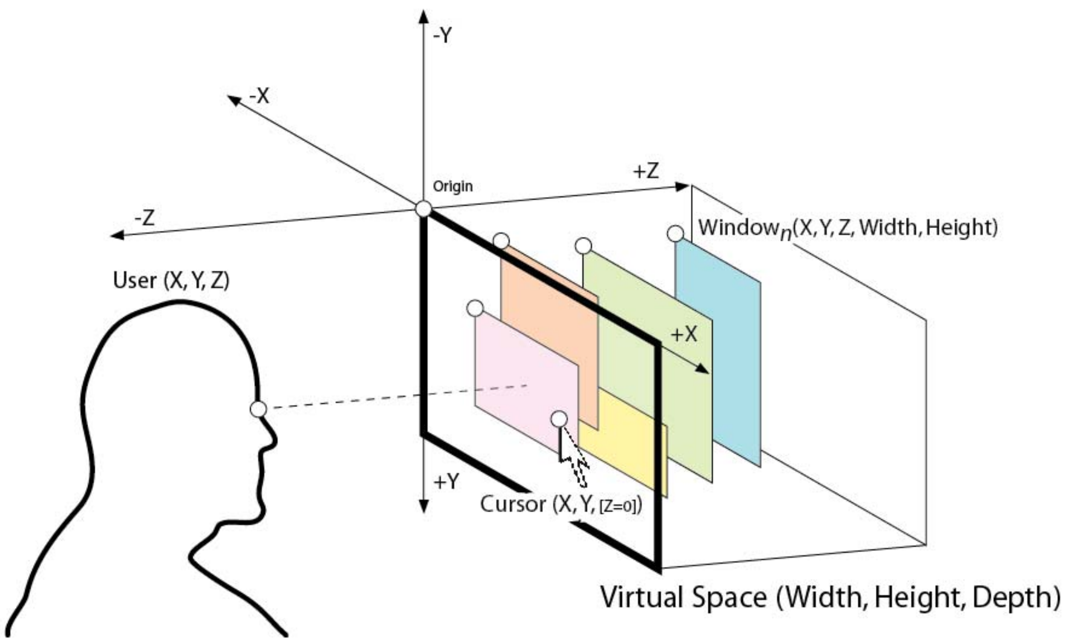
There are many existing technologies that help simulate 3D space: VR goggles, anaglyph (red/cyan) glasses, LCD shutter glasses, lenticular lenses, and LCD displays with a striped barrier. All of these work on the principle of sending a different image to each eye. We perceive depth primarily through this difference. But we can perceive depth with one eye closed because there are several other cues to depth besides binocular vision. The lens action of the human eye automatically adjusts to focus on the subject, which means a solitary eye knows how far away the subject is, and in some sense we can perceive depth through awareness of this physiological response. We also perceive depth through the parallax motion that occurs between visually overlapping near and far objects when we move our head and our viewpoint changes. Near objects appear to move relative to far objects. This is the feature I want to exploit.

Consumer stereoscopic display technologies – in particular 3D shutter glasses, head-mounted displays, and parallax barrier LCD displays – are not well suited to prolonged usage or productivity applications where resolution is important. Users have to put up with things like clunky headgear, decreased resolution, flicker, and headaches when dealing with 3D display technology as it stands today. I want to bypass these inconveniences and provide a depth experience practical for everyday use.

Platform and Implementation Details

My application requires Windows Vista and a webcam. It will make use of the Desktop Window Manager composition engine introduced in Vista. You can paint a live snapshot of the client area of any open application with a Desktop Window Manager API call. This is a central requirement for simulating parallax motion as window areas must be translated and scaled continuously. Users need a DirectX 9 capable video accelerator to use the DWM. The DWM is similar to Quartz in Mac OS X in that applications do not draw directly to the on screen display buffer. The best example of how the DWM works is when using Windows Flip 3D. Since windows are 2D entities by design, in order for them to be shown in 3D properly, the 2D rendering needs to be transformed into 3D space. Using off-screen buffers means that the window is only rendered once it has been positioned in the 3D environment, saving excessive processing. This high level OS management of desktop composition is essential to parallax motion simulation. The concept could eventually be extended to other platforms that use managed desktop composition.

A webcam on top of the screen will be used to monitor the user’s face position and update the display state. The application will maintain an internal representation of the user’s position, a 3D coordinate in real-world pixel units. In addition to having an XY position, each application will adopt a Z value, described as its distance in pixels from the front plane of the display.

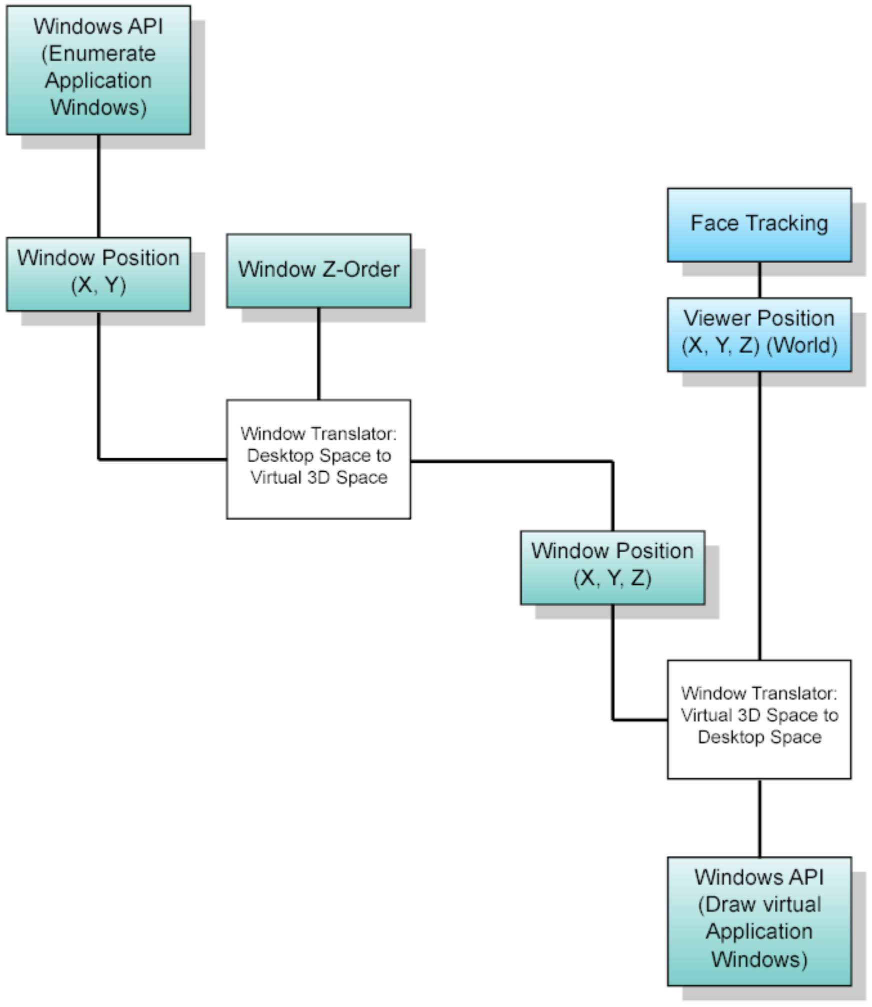

-Y
-X
Origin
+Z
-Z
Window$_n$(X, Y, Z, Width, Height)
User (X, Y, Z)
+X
+Y
Cursor (X, Y, [Z=0])
Virtual Space (Width, Height, Depth)

The XY Position of the user will be continuously tracked. I will try to track Z-position of the user’s head and model an appropriate display response, but if the viewer Z is too difficult to track accurately I will model it as a constant distance. In other words, I will assume the user's face is always 20 inches away from the display plane. This could be a configuration option.

Elementary visual techniques will be used to provide the illusion of depth.

1. Application windows will respond interactively to the user's head position with parallax motion.
2. Application windows will recede at a distance (decrease in size).
3. And possibly, application windows will appear blurry at a distance.
4. Atmospheric effect could also be incorporated to tint application windows at a distance.

Data Flow


Windows API (Enumerate Application Windows)
Window Position (X, Y)
Window Z-Order
Face Tracking
Viewer Position (X, Y, Z) (World)
Window Translator: Desktop Space to Virtual 3D Space
Window Position (X, Y, Z)
Window Translator: Virtual 3D Space to Desktop Space
Windows API (Draw virtual Application Windows)

Existing Products and Related Work

There are many research papers, software applications, and hardware devices (SmartNAV, Tracker Pro, Headmouse Extreme) devoted to controlling the user environment through head movements – moving a mouse cursor, clicking a button, etc.

However, my application is not meant to drive the mouse cursor. In virtual reality game applications such as Microsoft Flight Simulator, head *rotation* is used to control the game environment, i.e., to allow the player to look around the cockpit using head movement. Aside from use along with a head mounted display, I believe this behavior does not at all facilitate immersion, and it is only an alternative control mechanism. See http://www.naturalpoint.com/trackir/02-products/product-TrackIR-3-Pro.html. Head rotation response is only appropriate in HMDs and should not be employed in fixed display systems. Programs where changes in head *position* (XYZ movement/translation) are used to create an immersive response are much more closely related to my application. See http://www.gesturecentral.com/useyourhead/specs.html.

The issues involved in face tracking for a perceptual user interface are well explained in an Intel research paper by Gary Bradski titled *Computer Vision Face Tracking For Use in a Perceptual User Interface*. The paper discusses the CAMSHIFT algorithm, introduced by Intel and included in OpenCV. CAMSHIFT is a speedy algorithm that is ideal for face tracking for user interfaces. The basic principle underlying the algorithm is to track a region of color. Other more precise face tracking methods, such as those employing Eigenfaces are too slow.

Previous Experience

I received graphic design training at ISU and I have gained web design and Flash interface design experience in the professional field. I experimented with stereoscopic imaging and became familiar with the parallax effect while working on a design project during my senior year at Iowa State for which I created synthetic stereo image pairs for Viewmaster reels. I originally came up with the idea for parallax shift in Windows UI while a friend was taking HCI 575 last year.

Chad has years of professional experience programming Windows applications. While I have limited experience with Windows API programming, especially in regard to new Windows Vista DWM functions, I have a good friend at Microsoft who works on the desktop composition team. He has already provided useful insights about Vista's capabilities.

Usability

Accuracy

The application needs to track the head position with a low response time, high update rate, and with a fine resolution. Low response time is the challenge with the most opportunity for improvement through clever programming, since resolution challenges are bound to camera device resolution. The application should maintain a highly responsive frame rate with a goal of 30 updates per second so that it does not reduce usability.

Some kind of smoothing will have to take place so that the display does not jitter while the user is completely still. Examples of this kind of smoothing include noise gate, deadzone (threshold the position delta), moving average (average the user position over the past one second), etc.

Windows Environment

Many behaviors of the Windows operating system are not easily translated to a 3D space, where windows must behave as if they were sheets of paper floating in space, so there are several immediate practical concerns raised by the float windows application. I will try to address these concerns.

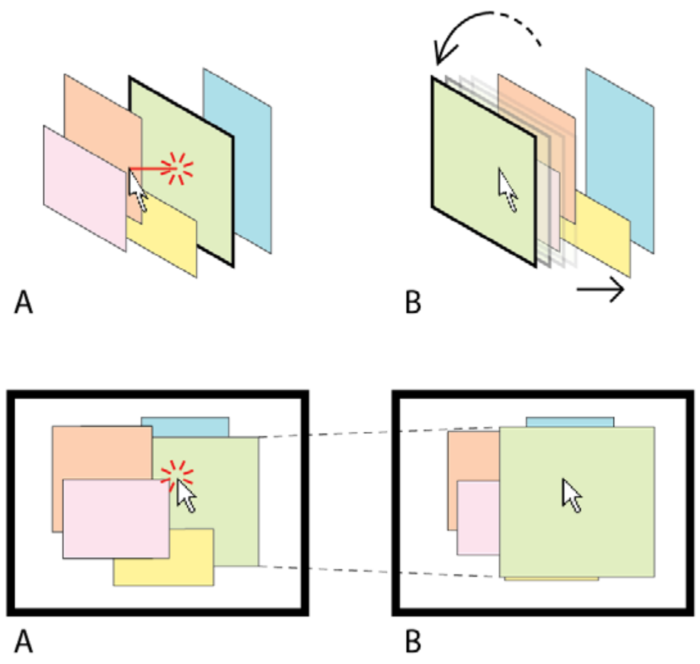
The application will not significantly change the behavior of the Windows environment. It should be suitable for continuous use in a productive setting. The idea of applications moving in response to the user's head position might seem unusable at first. However, what I have in mind will keep the active application completely stationary in its native pixel resolution. No application window virtual position will come closer than the screen plane. (No window will appear to float in front of the monitor. All applications will appear to float behind the screen.)

Task Switching

Task switching will be accomplished in the usual way; clicking an inactive application will bring it to the foreground.

The mouse cursor will appear as normal and thus remain at the zero z-depth (at screen distance). Task switching will take place with normal mouse click action as if a ray was cast from the user's eye and through the mouse cursor and into the first intersecting application window. The process is not as complicated as it sounds and will need no explanation in practice.

The activated window will be pulled out of its current level in the stack and move smoothly to a Z-position of 0, maintaining its XY coordinates. Other applications will recede to occupy the empty slot.



Drag & Drop

Drag and drop will be a problem since the actual background applications will be masked by the float window interface. Windows rendered in the float window interface are proxy images of the underlying applications. If a user attempts to drag a file from the foreground window to a background window, appropriate windows messages will not be sent to receiving applications unless drag and drop message forwarding is addressed in my application.

What happens with the Windows taskbar?

It remains in its normal position and appearance, and remains always on top, fixed at a virtual z-depth of 0 (at screen distance) so that it will never shift in response to user position. In other words, it doesn’t change at all.

What happens when applications are minimized or maximized?

Minimized applications disappear as usual. Nothing about maximized applications makes them incompatible with the 3D system - the window size is set to match the dimensions of the screen, and the position is locked. In my application this would behave as usual when the maximized application is in the foreground. When the maximized application loses focus, it would recede in space and appear to be reduced in size, but the user would not be able to see around it since it would continue *fill* the XY plane of the virtual space. Think of the maximized application moving back in the Z direction like the face of a piston receding down a cylinder. It fills the space. It would return to its maximized state when it receives focus again.

What happens when a user wants to read clear text from two applications at once?

The user will be able to pin applications to a z-depth of 0 (so that they don't recede when they lose focus) by shift-clicking or some other method, similar to the "Always on top" window behavior.

How are the boundaries of the virtual area defined and enforced?

The virtual space will extend backward from the screen in perpendicular extrusion. XY position of the mouse cursor in the virtual space will be confined to the dimensions of the display resolution; i.e. 1024x768. The maximum z-depth will be a user option.

Evaluation Criteria

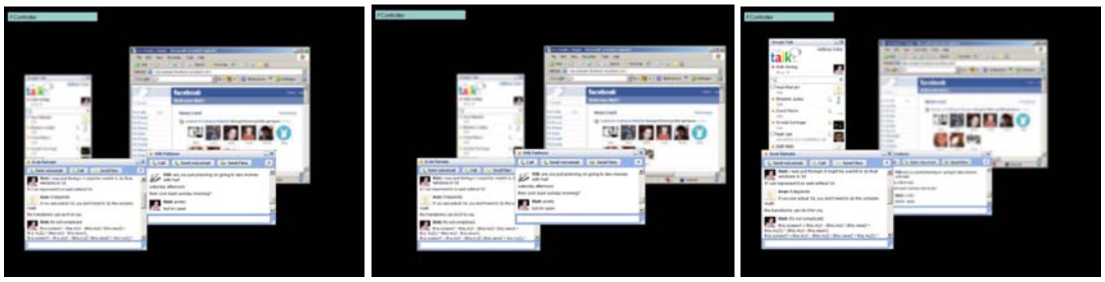
The success of the application will be determined by subjective analysis of how well it creates the illusion of depth. This will in turn depend on basic performance characteristics and the accuracy of the tracking algorithm. I would like to know if the depth effect makes the desktop workspace seem larger. I will ask testers if they felt that their desktop workspace was less cluttered while using the application.

A key question will be to determine whether the application is usable in an everyday working context or if users consider it a novelty. I will ask testers to rate how transparent the application is to their normal activities.

The application will be tested on at least 4 Windows Vista workstations of a variety of configurations, and we will try to use 4 different webcams, although this may not be possible. I believe the high requirements of Windows Vista Aero will remove most of the concern about processor speed and performance issues. If a computer is capable of running Aero it should be very capable of running the CAMSHIFT algorithm in real-time for a 30 fps video stream.

Prototype

I made a Flash prototype to demonstrate the essential behavior of the application. It is available online at http://www.mattmontag.com/hci575/project/. The prototype uses static images to show how the applications would shift on screen in response to head movement. The windows respond as if the user's head moved directly in front of the mouse cursor. Move the mouse from side to side and try to keep your nose directly in front of the pointer. You will see that far-away applications are moving from side to side as if they were floating a few inches behind your LCD screen.   Try to stare at the "foreground application" (the sharp image that doesn't move) instead of the mouse pointer as you move your head. This is just for demonstration purposes – in lieu of a webcam actually tracking your head position. Do not let the role of the mouse cursor in this demonstration confuse you: mouse movement will not cause application windows to shift in the real application.



References

[1] G. R. Bradski. Computer video face tracking for use in a perceptual user interface. *Intel Technology Journal*, Q2 1998.

[2] M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, Vol. 3, No.1, 1991, pp.71-86.

[3] M. Hunke and A. Waibel. Face locating for tracking and for human-computer interaction. 28th Asilomar Conference on Signals, Systems & Computers, Pacific Grove, Calif. 1994

[4] Kay Talmi and Jin Liu, "Eye and gaze tracking for visually controlled interactive stereoscopic displays," *Signal Processing Image Communication*, vol. 14, no. 10, pp. 799–810, 1999.

[5] A. Azarbayejani, T. Starner, B. Horowitz, and A. Pentland, "Visually controlled graphics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 6, pp. 602–605, 1993.

[6] Siegmund Pastoor, Jin Liu, and Sylvain Renault, "An experimental multimedia system allowing 3-D visualization and eye-controlled interaction without user-worn devices," *IEEE Transactions on Multimedia*, vol. 1, no. 1, pp. 41–52, 1999.