Virtual Reality on a Linux Desktop

Robert van Liere
Jurriaan D. Mulder
Center for Mathematics and Computer Science CWI,
Amsterdam, Netherlands
email{robertl,mullie}@cwi.nl

Abstract

In this paper we discuss a Linux desktop implementation of a near-field virtual environment, the Personal Space Station (PSS), We evaluate different hardware platforms by discussing implications each platform has on optical tracking latencies.

The PSS consists of a mirror in which stereoscopic images are reflected. The user reaches under the mirror to interact with the virtual world. Two cameras are used to track the space in which the interaction takes place. A fully configured PSS will consist of a high end stereo enabled graphics running at 1280x1024 @ 120Hz (for graphics rendering), two high end frame grabbers with cameras @ 60 Hz in PAL resolution (for interaction tracking), an acoustic tracker (for head tracking), and three foot pedals connected to the parallel port (for button clicks).

1 Introduction.

Virtual reality shows great promise as a research tool in computational science and engineering. The analysis of 3D data sets (read from disk or computed on the fly) may benefit from the interface styles provided by virtual reality. By providing additional depth and viewing cues, the virtual reality interface can aid the unambiguous display of 3D structures. In addition, virtual reality interaction styles allow for direct and intuitive exploration of the data.

In this paper we discuss the implementation of a near-field virtual environment, the Personal Space Station (PSS) [1], on a desktop PC running Linux. Near-field virtual reality allows users to interact with virtual objects within arm's reach of the user. Interactive tasks in the PSS are done in a direct and intuitive way using task specific input devices.

Reducing end-to-end system latency, the total time required for the displayed image to change in response to a user interaction or head movement, is one of the most challenging technical problems of a VR system. When latency is too high the user may have a feeling of moving an object, but actually see their virtual pointer move at a significantly later time. In the PSS, interaction is based on optical tracking using two cameras to track input devices. End-to-end optical tracking latency is computed as the sum of various latencies in the data pipeline associated with image acquisition, image processing, and displaying data. We will show that acceptable latency levels can be realized when using only commodity desktop components.

The format of the paper is as follows: In sections 2 and 3 we review the concept of the PSS and briefly discuss the PVR, a multi-threaded software environment which drives the PSS. In section 4, we report on the end-to-end latencies when using a VRML viewer. Finally, in section 5 we discuss some pros and cons of implementing the PSS on PC platforms.

2 Personal Space Station.

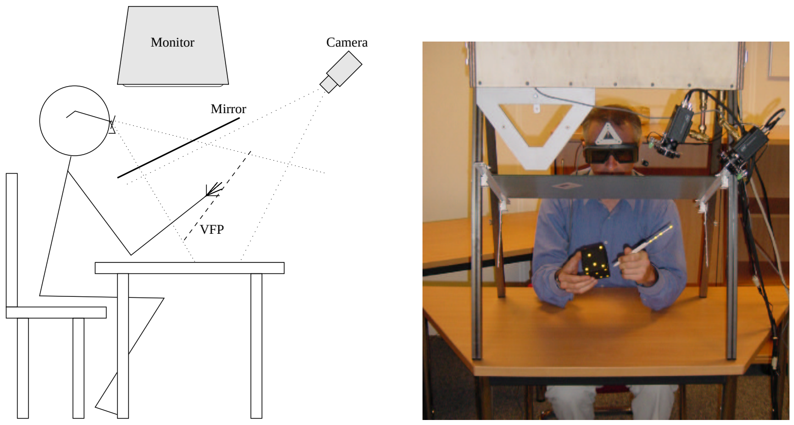

Figure 1: The Personal Space Station prototype. Left: concept. Right: prototype.

The design of the PSS and a prototype implementation is shown in Figure 1. The design distinguishes between three spaces: the *visual space* (defined as the virtual space that the user can visually perceive), the *interaction space* (the area in which the user performs 3D interaction), the *tracking space* (the area covered by the cameras).

Visual Space The mirror reflects the display surface of the CRT monitor into a *virtual focus plane* in front of the user. Depending on the application requirements, the CRT monitor and the mirror can be mounted at a different positions and orientations in the chassis. Changing these positions and orientations will result in a different position and orientation of the virtual focus plane with respect to the chassis.

A Logitech acoustic head tracker is used for head tracking. The ultrasound emitter is mounted in the chassis above the mirror and the receiver is mounted on the user's shutter glasses.

Interaction Space The interaction space is restricted to the area that the user can reach with his hands or with the input devices. The design goal is to position the interaction workspace such that 3D interaction can be realized comfortably, i.e. the user is seated behind a desk, his elbows are rested on the desk top, and he should not need to over-reach into the virtual world to perform 3D interaction. Important parameters in this respect are the position and height of the chair and table in combination with the user's physical characteristics such as his upper and lower arm length.

Tracking Space Two cameras are used to track the space in which the interaction takes place. A camera's field of view is determined by its extrinsic parameters (position and orientation) and intrinsic parameters (internal geometry and optical characteristics). The tracking space is defined as the intersection volume of both cameras' field of view. The tracking space is illuminated by rings of IR LEDs mounted closely around the camera lenses. IR-pass filters in front of the camera lenses are used to cut-off the light below a chosen wavelength. Retro-reflective markers are applied to all objects to be tracked. IR light from the LEDs is reflected by the markers into the lens such that, after thresholding, blobs of white pixels occur in the acquired image.

The steps performed in reconstructing a 3D position from the marker reflections in the images consists of 2D blob position detection, correction, rectification, corresponding, and 3D re-projection. To facilitate the correspondence problem, retro-reflective markers are placed in specific patterns.

3 Portable Virtual Reality.

In this section we review PVR, the software which drives the PSS (see Figure 2) [2]. The PVR environment consists of three parts, which are implemented as separate UNIX processes: (i) one or more device drivers that translate physical device properties into logical events that describe the state of the device, (ii) one or more PVR allocations, (iii) SSD, a device daemon that routes logical device events to PVR allocations. In the case of the PSS there are three device process, the optical tracker, the head tracker, and the pedals process.

A PVR application consists of one or more threads attached to a bus. Threads communicate using an event-based mechanism. This mechanism is conceptually easy to understand: after a thread attaches to the bus, it registers patterns with the bus that describe the events it is interested in. This allows each thread to set an *event filter*. When an event is posted on the bus, it is dispatched to the interested threads. Each thread is parameterized with a user supplied callback. The bus will invoke the callback whenever an event occurs that the thread is subscribed to.

Application threads are categorized in one of four types. The render thread is responsible for rendering. An application defined callback will be executed when a redraw is necessary. Device threads are responsible for device handling. An event will be posted each time the state of a device changes (i.e. button press, sensor motion, etc). The event report will contain the state of the device. File threads are responsible for all file I/O and I/O transactions with external services. For example, performing transactions with data base management systems, CORBA servers, or external simulations. Compute threads are responsible for all other required computation. This may be the

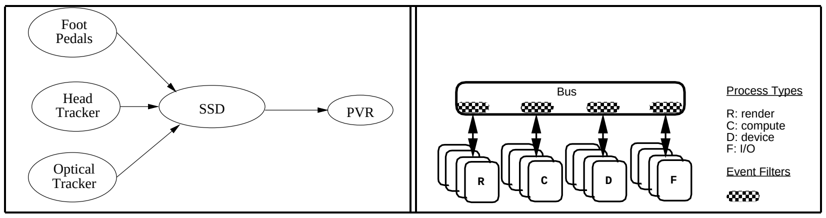

Figure 2: Left: The PVR environment. The the case of the PSS the environment consists of a PVR application and three processes that manage the optical tracking, the Logitech head tracker, and foot pedals. A device daemon (SSD) is used to route device events to the PVR application. Right: A PVR application consists of one or more threads attached to a bus. Each thread subscribes to events through event filters. A thread posts events to the bus, which in turn publishes these events to interested threads.

complete simulation itself, or the computation of visualization techniques (streamlines, iso-surfaces, etc).

A shared data facility allows threads to share data structures. This facility allows threads to allocate and lock shared data structures in a simple and flexible way. Pointers to shared data structures can be stored in events.

4 PSS latencies.

In this section we report experimental measurements of the end-to-end latencies of the PSS for a simple viewer of VRML objects. With this viewer a head-tracked user can use the cube device to position and orient the VRML object (see Figure 1). The VRML object used in the experiments consists of 26218 triangles.

Table 1 lists the components of two desktop platforms that have been used. The graphics engine is configured in full-screen quad-buffered stereo mode. The display is a 22 inch CRT monitor set to a resolution of 1280x1024 @ 120hz. The tracking engine consists of two Leutron Vision PictPort H4D dual channel frame grabbers and two Leutron Vision LV-7500 progressive scan CCD-cameras. Computar H0612FI lenses with a focal length of 6 mm and an F number of 1.2 are fitted on the cameras.

CPU	graphics engine	tracking engine
Pentium4 @ 2.25Ghz	NVidea Quadro4	dual Leutron Pictport frame grabbers
dual Athalon @ 1.6Ghz	ATI FireGL4	dual Leutron Pictport frame grabbers

Table 1: Two desktops used for measuring optical tracking latencies.

End-to-end optical tracking latency is defined as the delay from the moment of an cube device motion until the moment a frame is redrawn. The three processing steps involved are summarized as: (i) grab two stereo images, identify corresponding blobs in the images, compute the pose of the input device from blob patterns, (ii) send event report to SSD, SSD receives and broadcasts event report to PVR application, receive event report from SSD, (iii) compute new application state, send redraw event to render thread, redraw geometry.

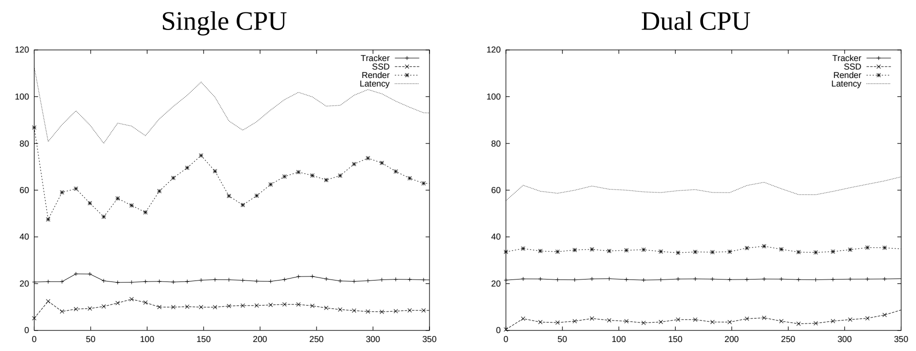

Figure 3: End-to-end optical tracking latencies on a single and dual CPU PC. The y-axis are milliseconds. The x-axis are seconds of the interactive session. Rendering, optical tracking, SSD transport times are graphed as dotted, dashed and crossed lines. The total end-to-end latency is graphed as a solid line.

Figure 3 shows the end-to-end optical tracking latencies during an interactive session with the virtual object for the single CPU (left) and the dual CPU case (right). The times used in each processing step (dot, dash, star lines) and the total end-to-end latency (solid line) is plotted. The overhead involved in sending/receiving events to/from the SSD is included in the processing time of the SSD.

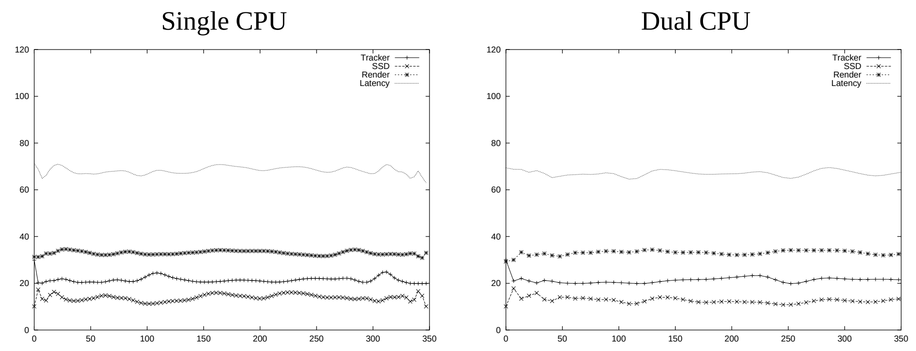

Figure 4: End-to-end acoustic tracking latencies.

Figure 4 show the end-to-end acoustic tracking latencies for each hardware platform. The Logitech acoustic head tracker is connected to the serial port. End-to-end acoustic tracking latency is defined as the delay from the moment a of head movement until the moment a new frame is redrawn. The three processing steps involved are summarized as: (i) read event report from acoustic tracking device, (ii) send event report to SSD, the SSD broadcasts event report to PVR application, (iii) compute new application state, send redraw event to render thread, redraw geometry.

A number of observations can be made from this data:

- The end-to-end interaction latency (optical tracking) is in the range of 100 ms for the single CPU PC and 60 ms for the dual PC, The end-to-end head tracking latency (acoustic tracking) is in the range of 60 ms for both platforms.

- In the acoustic tracking case where no additional processing is involved, the single and dual platforms have similar latencies. For optical tracking, where additional image processing is required, the single and dual platforms differ. The end-to-end latencies in the dual case are lower and have smaller fluctuations than on the single CPU platform. This is probably due to the multi-threaded environment in which the tracking and rendering threads can be assigned to different processors.

- The rendering latency scales linearly with the number of polygons that are drawn. In contrast, is optical tracking latency is primarily determined by the time consumed in scanning the image during 2D blob detection. The time used to determine blob correspondence and compute the pose of the cube is negligible to the time spent in the blob detection itself. Hence, although optical tracking is seemingly a complicated and time consuming process, the pose of a few cube devices can be computed within 25 ms.

- The bandwidth needed to run the VRML viewer can be computed from the frame rates. The frame grabbers can acquire 30 images per second on the platforms. This amounts to 2 * 30 * (640 * 480) bytes per second that need to be transported over the PCI bus to the cache. The acoustic tracker can generate 50 reports per second, resulting in 50 * 20 floats that are read from the serial port. The graphics frame rate is approximately 45 frames per second (22 for the left and 22 for the right eye). In the case of the virtual object used in the experiment this amounts to 45 * 26K polygons per second.

- The amount of time required by the SSD to receive event reports from the device processes and send these reports to the PVR application is relatively high (approx 20 ms). Event reports are sent as messages over TCP/IP sockets using reliable, two-way, connection-based byte streams.

  Running the optical tracker on a remote Linux PC will increase the latencies introduced by the SSD but not tracking or rendering latencies. When used in a collaborative virtual environment the SSD will broadcast event reports to all PVR applications subscribed to the event.

5 Discussion

In the previous sections we have discussed some issues of implementing the PSS on a Linux PC desktop. We have demonstrated that end-to-end latencies are reasonable on both single and dual CPU platforms.

Many of the state-of-the-art virtual environments require large, expensive and cumbersome equipment. For example, the popular CAVE environment requires at least 4 large Barco-like projectors driven by a multi-processor SGI Onyx with at least 4 graphics pipes. Interaction is mostly realized with wire-based 6 degrees-of-freedom input devices, such as a wand, equipped with a magnetic tracker. Interaction techniques that use these devices are often indirect, difficult to use, and lack precision. Furthermore, the CAVE will need to be placed a large darkened room.

In contrast, the motivation for using desktop PCs for the PSS was to address the issues of ergonomics and costs. With respect to ergonomics, it has been our desire to make the design ‘user and office-friendly’. This includes keeping the system compact and portable, being able to use it in normal office conditions. With respect to costs, the hardware of the prototype amounts to approximately 13 kEuro. Although still expensive, this is a fraction of what special purpose VR environments cost.

With respect to latency it can be argued the experiments given in section 4 only report the system latency of processing data in the system, and not the total time required for the displayed image to change in response to a user interaction. Indeed, our measurements do not report the precise moment a device changes a pose. Rather, our measurements approximate the time that the CCDs of the cameras are filled with information to be processed. Similarly, our measurements do not report the precise moment that the monitor is refreshed. Rather, the time that the GLUT call ’glutSwapBuffers()’ returns is used. In both cases an additional temporal latency of a few milliseconds may occur. To take these cases into account, a technique that visually measures system latency must be implemented. For example, Liang et al. built a pendulum and use a video camera to record the periodic motion of the pendulum in the real and virtual world. In this way they could compute the total end-to-end latency by comparing the pendulum’s position in the real world and the position in the virtual world, [3]. Variations of this technique have also been implemented by others, eg. [4].

Linux has proved to be an excellent environment for the development of the PSS. We use out-of-the-box the Redhat 7.X Linux distribution with standard released device drivers from Leutron, NVidea and ATI. The PVR environment requires the GLUT and OpenGL libraries for rendering support, POSIX for multi-threading support, libC and libm for miscellaneous support such as memory management, socket I/O and math routines. Linux has allowed us to experiment with different commodity components by providing the flexibility of interchanging and upgrading individual hardware components.

Future work includes replacing the acoustic head tracker and high-end Leutron framegrabbers with multiple Firewire cameras. It is not yet clear what frame rate and system latencies will be achieved when using four Firewire cameras. In addition, a half-silvered mirror will be mounted in the chassis. This will require a more accurate calibration of the cameras.

References

[1] J.D. Mulder and R. van Liere. The personal space station: Bringing interaction within reach. In S. Richer, P. Richard, and B. Taravel, editors, *Proceedings of the Virtual Reality International Conference, VRIC 2002*, pages 73–81, 2002.

[2] R. van Liere and J.D. Mulder. PVR - an architecture for portable VR applications. In M. Gervautz, A. Hildebrand, and D. Schmalstieg, editors, *Virtual Environments ’99, Proceedings of the Virtual Environments Conference & 5th Eurographics Workshop*, pages 125–135. Springer Verlag, 1999.

[3] J. Liang, C. Shaw, and M. Green. On temporal-spatial realism in the virutal reality environment. In *Proceedings of the Symposium on User Interface Software and Technology*, pages 19–25. ACM/SIGGRAPH, 1991.

[4] C. Swindells, J. Dill, and K. Booth. System lag tests for augmented and virtual environments. In *Proceedings of the Symposium on User Interface Software and Technology*, pages 161–170. ACM/SIGGRAPH, 2000.