Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 33 (2007) 420–437
PARALLEL COMPUTING
www.elsevier.com/locate/parco

WinSGL: synchronizing displays in parallel graphics using cost-effective software genlocking

D. Cotting *, M. Waschbüsch, M. Duller, M. Gross

Computer Graphics Laboratory, ETH Zurich, Switzerland

Received 19 July 2006; received in revised form 4 December 2006; accepted 12 February 2007
Available online 1 March 2007

Abstract

This article presents a software genlocking approach for unmodified Microsoft Windows systems, requiring no specialized graphics boards but only a low-cost signal generator as additional hardware. Compared to existing solutions for other operating systems, it does not rely on any real-time extensions or kernel modifications. Its novel design can be divided into two parts: a dedicated driver reads an external synchronization signal via interrupt lines. A user-space application performs the synchronization using EnTech PowerStrip [EnTech Taiwan, PowerStrip Version 3.61, http://www.entechtaiwan.net/util/ps.shtm] by inserting or removing lines to the invisible part of the images output by the graphics board. Robustness to potential frame losses is achieved through continuous consistent timestamping. Tests yield an accuracy of up to $\pm 1/2$ line deviation from the external signal and a low CPU load of 2% on current PC systems. Our system has been designed to be compatible with off-the-shelf graphics hardware and digital output devices based on LCD or DLP technology. Our solution can be employed to build cost-effective VR installations such as large tiled displays and spatially immersive environments using commodity PC clusters. Furthermore, displays synchronized to camera acquisitions allow for novel and convenient systems in the area of 3D scanning and smart adaptive displays, two application areas we will present in the second part of this article.

Keywords: Genlocking; Display synchronization; Parallel graphics; Distributed display; Network graphics

1. Introduction

Many VR installations combine video signals from multiple graphics engines to build large tiled or spatially immersive displays such as CAVE™ installations [1] or the Varrier™ display [2]. To achieve a high visual quality, all involved video output devices require exact frame synchronization, i.e. genlocking. Besides the aforementioned areas which have output synchronization as their major requirement, other projects like structured light scanning systems [3,4] require synchronization between video output devices and cameras.

* Corresponding author.
E-mail address: dcotting@inf.ethz.ch (D. Cotting).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2007.02.014

Recently, commodity PC clusters were introduced as an alternative to specialized graphics mainframes, like SGI Onyx systems. Genlocking in those systems is usually achieved with specialized graphics boards, available, e.g. from NVIDIA or 3Dlabs, which are unfortunately rather expensive. Software genlocking relying on very little, inexpensive hardware can provide a cost-effective alternative. Since timing is critical, to date reliable solutions are only available for real-time Linux.

This article explores the possibilities of using software genlocking on unmodified Microsoft Windows operating systems having no support for real-time applications. We provide a robust implementation for synchronizing the video timings of all PCs in a cluster with a reference clock signal. Our software is intended to be used in combination with existing high-level VR middlewares like Net Juggler [5] which already provide an application-level frame synchronization. By additionally synchronizing the video signals, genlocking improves the resulting display quality.

The software architecture, which comprises a system driver and a synchronization process, permits transparent extension of any application with genlocking capabilities at run time. A hardware abstraction layer based on PowerStrip [14] makes our system suited for any standard graphics board. The only special hardware requirement is a low-cost TTL signal generator distributing the reference clock to the cluster nodes. The employed synchronization strategy has been carefully developed to work smoothly with state-of-the-art digital output devices like LCDs or DLP projectors which are quite sensitive to timing adjustments.

This article is an extended version of a previous publication [6]. It provides additional technical details on our implementation and a more elaborate presentation of potential applications. It is organized as follows: after surveying related work, we provide a short overview of the basics of video timing and present the challenges of synchronizing multiple video cards without direct hardware support. Based on those insights, we develop a set of design decisions by evaluating alternatives to synchronize video cards in software and by considering problems specific to digital output devices. Those decisions build the basis for our implementation, including both software and hardware design. Implementation issues are described in detail, as we have made our software publicly available as open source (see http://graphics.ethz.ch/winsgl). We evaluate our approach on a variety of hardware platforms and provide comparisons with existing solutions. Potential applications of our novel approach illustrate the flexibility and versatility of software genlocking. Finally, we summarize our achievements and give an outlook on possible future extensions.

2. Related work

Genlocking has been used in video studios for a long time. There, synchronization of different video sources like cameras, title generators and computers is required in order to allow for clean cutting, blending and other editing operations. Apart from costly specialized hardware, inexpensive multimedia computers like the Amiga supported an external video clock signal, which provided genlocking capabilities. Even though some of these vintage computers are still in use today, their development and support has been discontinued.

Traditionally, VR installations have been built using high-end graphics mainframes such as those produced by SGI. Besides high graphics performance they usually offer a custom hardware genlocking option enabling for tiled or active stereo display technologies. However, they are targeted at a specialized high-end market and come at a high price.

As the graphics performance of conventional PCs is rapidly increasing, expensive graphics mainframes are successively being replaced by powerful low-cost PC clusters. Software systems like Net Juggler [5], Syzygy [7] or Chromium [8] provide platforms for distributed execution of virtual reality applications. See Streit et al. [9] for an extensive overview. Designed as high-level middleware platforms, they do not include sophisticated display synchronization methods. Net Juggler [5] at least provides software swaplocking to synchronize framebuffer swaps between the cluster nodes, using synchronization barriers over Ethernet connections. The other systems can be complemented by methods such as those by Bues et al. [10] or Scheffer et al. [11] which also implement software swaplocking.

If higher synchronization accuracy is desired, the cluster nodes can be equipped with high-end graphics cards, e.g. from NVIDIA or 3Dlabs, which are available with genlocking options. However, the prices of such specialized boards still cannot compete with commodity PC hardware. Furthermore, some applications do not need the rendering performance of the high-end graphics boards at all.

A cheap alternative is provided by software genlocking, which tries to synchronize the video timing as close as possible to a reference clock signal by only using a minimum amount of additional hardware. The reference signal is usually fed through the parallel port, and synchronization is carried out by modifying one or more parameters of the video timing in software. SoftGenLock [12] is an open source implementation for Linux designed as an extension for the Net Juggler platform. An improved implementation is available from Wössner et al. [13], also for Linux only. Besides their restriction to a specific operating system, both solutions only support a limited set of graphics hardware because they directly modify the registers of the graphics chips: either NVIDIA cards only [13], or boards compatible to the VGA standard on the register level [12], both excluding for example modern ATI boards. Moreover, their synchronization strategies are incompatible with most LCD and DLP display devices, yielding distorted images.

3. Video timing

This section provides a short introduction to video timing and presents the challenges of synchronizing multiple video cards without direct hardware support. The information provided in this section serves as a basis for the conclusions taken in the next section.

3.1. Image generation

Throughout this article, the term synchronization will mainly be used to refer to the synchronization of multiple video cards, respectively their output signals. The most basic nature of synchronization in terms of video signals, however, is the synchronization between the video source, e.g. a graphics card, and the display device, e.g. a monitor or a projector. Hence, we go into the details of this synchronization by examining how video frames are encoded and transmitted from the video source to the display device.

A frame consists not only of visible but also of invisible pixels. The latter are a legacy from the times of cathode ray tube (CRT) monitors. Fig. 1 shows how a frame is composed of various video timing areas in a single frame and indicates the path the cathode ray follows when drawing an image. The visible area, where the ray is effectively drawing the image onto the screen is referred to as the active period. It is surrounded by an invisible area which is made up by the front porch, the back porch and the retrace. While passing this area the ray is disabled.

Those four timing components appear horizontally as well as vertically and always in the same sequence, as illustrated in Fig. 2. The gray area marks one full sequence comprising a full line (horizontally) or a full frame (vertically). Furthermore, the state of the synchronization signal is indicated. It is active during the retrace and

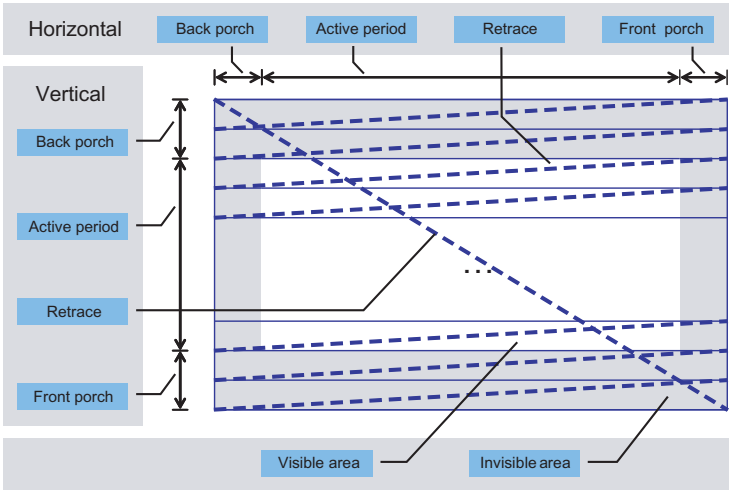

Fig. 1. Video timing.

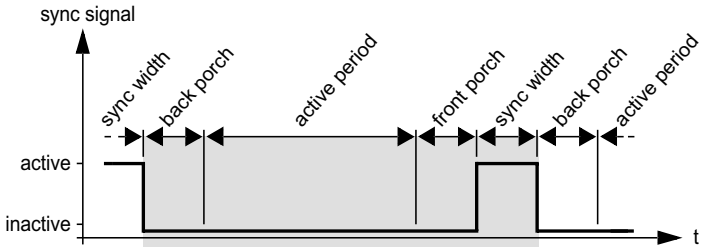

Fig. 2. Sequence of timing components.

thus effectively causes the ray to return. The purpose of those timing components for CRT monitors are as follows.

- During the active period, the actual image is drawn.
- The front porch follows the active period. It gives the ray time to go to black before retracing across the screen.
- During the retrace period, the sync signal is active and the ray moves to the other side of the screen. The duration of the retrace is referred to as the sync width.
- After the ray has retraced to the other side of the screen, the back porch allows the ray to stabilize again before drawing the next active period.

Since nothing is displayed during the front porch, the retrace and the back porch, these timing components are often used to transmit non-image information. For example, during the vertical retrace teletext is transmitted in PAL, or closed caption in NTSC.

Horizontally, the duration of these periods is measured in pixels. Each pixel has a duration (the reciprocal of the pixel clock) and, thus, the duration of a horizontal period is the number of pixels multiplied with the duration of one pixel. One full line consists of one horizontal active period, one horizontal front porch, one horizontal retrace and one horizontal back porch. The duration t_{line} of one full line can be computed as

$$t_{\text{line}} = \frac{1}{f_{\text{pixelclock}}} \cdot (p_{\text{active}} + p_{\text{front}} + p_{\text{retrace}} + p_{\text{back}}), \tag{1}$$

where $p_{\langle \text{index} \rangle}$ is the number of pixels of the corresponding horizontal timing component. Note that in data sheets, one usually specifies the scan rate, which is the number of lines transmitted per second. Therefore, the scan rate is the reciprocal of the duration of one line.

Vertically, the duration is measured in lines. The duration of a vertical period is the number of lines it consists of multiplied with the duration of one full line. One full frame consists of one vertical active period, one vertical front porch, one vertical retrace and one vertical back porch. The duration t_{frame} of one frame, thus, can be computed as

$$t_{\text{frame}} = t_{\text{line}} \cdot (l_{\text{active}} + l_{\text{front}} + l_{\text{retrace}} + l_{\text{back}}), \tag{2}$$

where $l_{\langle \text{index} \rangle}$ is the number of lines of the corresponding vertical timing component. Note that in data sheets, one usually specifies the refresh rate, which is the number of frames transmitted per second. Therefore, the refresh rate is the reciprocal of the duration of one frame.

The combination of Eqs. (1) and (2) shows that the duration of a frame solely depends on the number of all pixels and the duration of one pixel.

3.2. Synchronizing multiple video cards

From the properties of the video timing it is obvious that synchronizing two or more video cards is primarily a question of either continuously synchronizing their pixel clock or at least synchronizing the point in time

when they start drawing their lines. Hardware genlocking solutions have the ability to use an external clock as their pixel clock or at least as a trigger for the vertical retrace.

Without this support in hardware, the most obvious approach is to set the video cards to exactly the same timing (pixel clock, size of the front porch, back porch, et cetera). Then, in theory, once the shift between the cards has been eliminated, the signals should be in sync. However, this approach fails due to slight disparities of the pixel clocks of the cards. The pixel clock is derived from the video card clock generator whose core component is a quartz crystal. Though quartz crystals are very precise and have a very low tolerance limit, it is still not possible to keep two or more cards synchronized. Tests at a refresh rate of 60 Hz with two identical graphics cards (NVIDIA GeForce 4 MX) in two identical computers set to identical timings yielded a discrepancy of 0.000086 Hz between both clocks. This resulted in one card being one full frame ahead of the other after approximately 194 s.

Hence, synchronizing multiple video cards is not as easy as bringing the cards in sync once since it requires continuous control and intervention to keep them synchronized. Information about the reference signal and about their own timing therefore has to be known and a method must be available to change the local video timing. The following section presents several solutions that fulfill these requirements.

4. Design decisions

Synchronizing video cards in software requires first of all information about the reference and local timing. Second, one needs to modify the local timing in order to catch up or slow down and thus stay as close as possible to the reference signal. This section presents and discusses different solutions fulfilling these requirements. Furthermore, problems that might arise with software genlocking and digital output devices are discussed. Since we impose proper operation with LCD and DLP devices, these problems are of particular importance and have to be solved.

4.1. Exchanging the reference clock signal

The reference signal used by a software genlocking solution generally equals the refresh rate, which means it is usually within the range of 50–100 Hz. In practice, it has to be exchanged over an I/O port of the computer. Since timing precision is crucial for genlocking and the costs of additional hardware should be kept minimal, the parallel port is an optimal choice. It can still be found in many current computers or it can be added with inexpensive expansion cards. The pins of the parallel port can be accessed directly at a certain memory-mapped address thus making it fast and easy to use from the software side as well as from the hardware side. Other ports would either require more sophisticated hardware or an overhead in processing, thus increasing costs and decreasing precision. In the following sections, we investigate different signal acquisition and signal propagation variants.

4.1.1. Interrupt vs. polling

SoftGenLock for Linux [12] uses polling to read the reference signal from the parallel port. To avoid a continuous loop of busy waiting, the software releases the CPU for most of the time and only wakes up shortly before the next clock signal is expected. However, as a consequence, the software is exposed to the good will of the scheduler of the operating system and thus only works reliably under a real-time kernel. A far better approach is the use of hardware interrupts which have a very high priority also in non-real-time operating systems. Feeding the external clock through an interrupt, e.g. the ACK-line of the parallel port, allows for very low response times. When an interrupt occurs, the kernel will immediately preempt all other processes running at a lower priority and call the interrupt service routing (ISR) associated with the interrupt of the parallel port. Only other hardware interrupts that happen at the same time or that are in their ISR already can delay the call of the ISR associated with the parallel port. However, even if this happens, the delay will be minimal, as ISRs have to be very short. They only do the work that is absolutely necessary in the moment the interrupt occurs and then schedule a so-called deferred procedure call that does the main work later. Using the interrupt of the parallel port thus is a very efficient and simple way to read the external clock signal with a very high precision in a non-real-time operating system.

4.1.2. Signal propagation topology

The existing SoftGenLock for Linux solution uses a setup with one master sending its internal sync signal to multiple slaves. Sending the signal itself is easy and reliable, since it is basically a write access to a memory address. However, detecting the own local timing, and thus knowing when to send, is not very reliable. If the master misses its own retrace from time to time, it will send out an inaccurate clock signal to the slaves. Therefore, the decision was made to use a dedicated clock generator as a reference and run all computers as slaves.

4.1.3. External synchronization via FireWire

Task scheduling on the local CPUs is the biggest timing critical issue, especially when the genlocking software is running concurrently to computationally expensive display applications. An alternative to the aforementioned approaches would be the offloading of the complete synchronization task from the rendering nodes to a dedicated PC. This could be achieved using FireWire links. FireWire controllers have the ability to directly access the computer memory with no interaction of the main CPU. This ability is commonly used for debugging, but could also provide direct memory access to the address spaces of the display nodes from an external computer. If an approach like SoftGenLock for Linux is chosen, where the actual genlocking is carried out by raw memory accesses, i.e. by modifying memory-mapped graphics cards registers, a synchronization PC could spend its full CPU time with polling and writing to registers of multiple graphics cards in multiple display nodes connected over the FireWire bus. As a drawback, such an approach would again be constrained to a very specific set of graphics hardware which is why we did no further investigation. Our implementation yields an easier but nevertheless reliable solution which runs autonomously on each display node.

4.2. Measuring the local timing

When measuring the local timing, one basically has to detect the point in time when the vertical retrace occurs on the local video card. The Microsoft DirectDraw API provides a function WaitForVerticalBlank that will block until the vertical retrace has occurred. However, in most drivers, this function is implemented as busy waiting. Having the CPU blocked the whole time renders the computer useless and thus is not an option. Hence, the process has to release the CPU for a certain amount of time using a wait call. The time to wait will be chosen as the period length of the sync signal minus some safety margin. On a non-real-time operating system, however, it is not guaranteed that the scheduler will reactivate the process in time and thus it is possible that one or more local retraces are missed. Those cases have do be detected using appropriate time-stamping algorithms.

4.3. Modifying the local timing

To synchronize the local video card, it is necessary to modify its timing, so it can slow down or catch up compared to the reference signal. This can be done by changing the invisible area of the image. By removing or inserting some pixels (horizontal) or lines (vertical) into one of the timing components that comprise the invisible area (front porch, back porch and retrace) it is possible to change the duration of a frame without changing the visible area. Another way to achieve this is to modify the pixel clock. These methods were evaluated with the targeted output device, an NEC LT 240K DLP projector. Additionally, two DELL LCD flat panel displays (models FP1700 and FP1701) were tested as well. All devices have been connected to the analog VGA output of the graphics boards.

According to our experience, digital display devices do not tolerate even small changes in most of the timing components, because they have to resample the video signal into an internal framebuffer. Changes to the different components show different distortions in the displayed image. When increasing or decreasing each of the six invisible components in a full frame, we observed horizontal or vertical shifts of the picture generated by the DLP projector. Table 1 lists the observed reactions, where the arrows indicate the direction of the shift. The granularities for these changes were eight pixels horizontally and one line vertically. The reactions could already be noticed after increasing or decreasing one step of these granularities. Similar effects could be

Table 1
Reaction of an NEC LT 240K DLP projector to different changes in the invisible timing components

	Horizontal		Vertical	
	Increase	Decrease	Increase	Decrease
Front porch	←	→	Stable	Stable
Back porch	→	←	↓	↑
Sync width	→	←	↓	↑

The direction of the arrow indicates the observed shift of the picture.

observed on the other digital displays. Depending on the device, changes to the vertical front porch of up to ten lines did not show any reaction.

These reactions make sense when considering how these display devices work. Looking at the vertical front porch for example, which occurs below the visible area, between the active period and the retrace (see Fig. 1), one can expect that changes to this timing component can be ignored. The visible area of the image is already drawn and sampled to the frame buffer of the device, and the sampling circuit is waiting for the vertical retrace, which indicates the upcoming start of a new frame and is transmitted through a dedicated sync line. Hence the vertical front porch can safely be ignored, as long as the overall timing does not vary too much from the timing onto which the sampling circuit has been calibrated. As an example where the image changes, the reaction of the device to an increase of the horizontal back porch also makes sense. The back porch appears before the visible area of that line and when it is increased, the sampling circuit already samples pixels because it thinks the video signal is already in the active period. These pixels are all black during the back porch and, hence, the image appears to be shifted to the right.

Another method to change the video timing is to change the pixel clock. Digital display devices however do not appreciate these changes at all, reacting with jitter and distortion of the whole image. Thus, it is obvious that the only suitable method for changing the local video timing is the modification of the vertical front porch. It is a very clean way, since the change occurs after the visible area has been sampled, making it possible to operate the device with a timing it has not been fully calibrated to, but to still have a properly sampled image. Using this method, we also could observe a similar robust behavior for displays connected to the DVI output of the graphics board and with different resolutions.

5. Implementation

According to the previous considerations, we implemented a robust software genlocking solution running on unmodified Windows 2000 or Windows XP systems. Fig. 3 shows the architecture of the solution and the interaction between all components. The boxes on the left side are our custom-built software modules. The kernel-mode sgl driver measures the external clock signal via an interrupt service routine and generates appropriate timestamps. Those are received by the user-mode sgl application which measures the local timing from

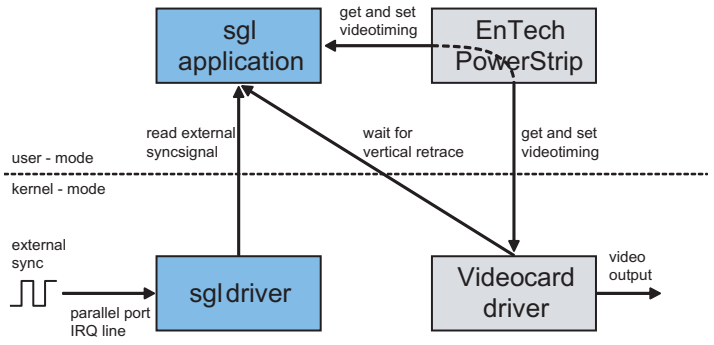

Fig. 3. WinSGL architecture.

the video card driver and performs the necessary adjustments by programming the graphics board. The graphics hardware is accessed through the third party utility EnTech PowerStrip [14] and the DirectDraw API of the video driver, both providing a hardware abstraction layer, making our system applicable for a variety of graphics boards. A detailed description of all system components is presented in the following sections.

5.1. Hardware

The system requires a reference signal to synchronize with. A square-shaped TTL signal with the frequency according to the targeted refresh rate is fed into every PC through the parallel port interrupt line at pin 10 on the commonly used 25 pin D-SUB connector. An appropriate signal can be generated by an inexpensive off-the-shelf signal generator or a programmable microcontroller-based solution, the approach we have chosen for our implementation.

5.1.1. Clock generator

The WinSGL software requires an external signal to synchronize with. The frequency of this signal has to be equal to the targeted refresh rate, and its shape ideally is a square wave. Other shapes might also work as long as they have a clear high–low and low–high transition, and both the low and the high parts have reasonable durations. Low and high signals should comply with the signal levels for IEEE 1248 compliant standard parallel ports which are 0 V and 5 V (TTL).

We used a Toshiba TMP92FD54 microcontroller on a developer board (approx $50, see Fig. 4), running a program that generates a clock signal by looping a certain number of iterations and thus executing a certain number of processor cycles of known duration. Such a generator provides a stable clock signal that is appropriate for software genlocking.

During implementation, we also evaluated an inexpensive off-the-shelf signal generator. The quality of this device, however, was so poor that the frequency changed significantly over time, probably due to thermal effects. Though these changes did not disturb the operation of the software genlocking, which adapted smoothly to the changes, a stable reference signal was desired, and thus the microcontroller was used as a clock generator for the project.

5.1.2. Parallel port cable

The reference signal provided by the clock generator is fed to the computers through a custom cable that provides a BNC connector for the clock generator and several male 25 pin D-SUB connectors for the parallel ports. On a D-SUB connector, the clock signal is connected to pin 10 (ACK-line), which can be used to trigger the interrupt line of the parallel port, and ground is connected to one of the many ground pins (pins 18–25). Since parallel ports are very sensitive and some clock generators can generate negative peaks when being powered off, a quenching diode is added to the cable as a protection.

5.2. Hardware abstraction layer

To support a variety of different graphics boards, we are using a hardware abstraction layer, which comprises the DirectDraw API included in Windows as well as the third party utility PowerStrip.

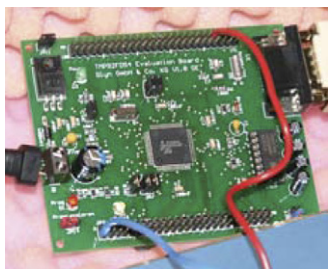

Fig. 4. Microcontroller for generating clock signal.

5.2.1. DirectDraw

The hardware-independent DirectDraw function WaitForVerticalBlank is used by our solution to detect vertical retraces. Virtually every video driver available for recent versions of Windows is DirectDraw compatible and thus provides an appropriate implementation of this function.

5.2.2. PowerStrip

PowerStrip [14] from EnTech Taiwan is a shareware utility that allows to change every aspect of video timing. It supports all native display modes implemented in the graphics driver of the GPU manufacturer. Besides the graphical user interface, PowerStrip features an API that can be used to access its functionality from other programs. Since PowerStrip supports a variety of graphics boards, we use it as a hardware-independent way to read and modify the timing of the video card.

5.3. SGL driver

Like most modern operating systems, Windows 2000/XP does not allow user-space applications to directly access hardware. Therefore, we developed a kernel-mode driver to measure the reference clock signal at the parallel port.

The driver installs its own interrupt service routine to keep track of the reference signal via parallel port interrupts. It maintains a ring buffer of the last eight timestamps when the interrupt occurred and a ring buffer of the last 256 interval lengths between two interrupts. The former is required by the application to measure the deviation between the local video timing and the reference signal. The latter is used to get a smooth average measurement of the period length of the external clock which is represented by the sum of all 256 values. At each call to the interrupt service routine, the sum is updated by adding the current interval length and subtracting the oldest one, yielding a time complexity of O(1). Time is measured in ticks of the computer timer. Under Windows, this time is referred to as the *performance counter*. In recent systems, the resolution of the timer equals the clock of the CPU. On older systems, this is not the case and the timer can have a different (usually lower) resolution. Many early Pentium 4 computers for example have a timer with a resolution of 3.579545 MHz.

The following sections describe the driver in more detail providing insight into its implementation and operation.

5.3.1. Initialization

When being loaded, the driver creates a device object *\\Device\sglreads0*, attaches it to the driver stack and creates a symbolic link *\\DosDevices\sglreads0*, which makes it easier to access the driver from the application. The driver will only take further initialization steps when it is actually used. With the first access to the driver, which must be an I/O control operation with the control code `IOCTL_INITIALIZE`, the driver initializes and prepares for operation. In the registry, it looks up the parallel port device to attach to, opens it and connects the interrupt service routine to the interrupt of the parallel port. Finally, it waits for the ring buffers to fill, before the I/O control operation returns. If not enough samples arrive through the interrupt line in reasonable time, the I/O control operation returns with an error.

5.3.2. Interrupt service routine

On every interrupt of the parallel port, the interrupt service routine is called. The sole purpose of this routine is to update the two ring buffers and to compute the average period length. When called, the interrupt service routine first stores the current timestamp. Afterwards, it computes the interval length between this measurement and the one from the last call to the routine (stored in the ring buffer containing the last eight timestamps). The next step is to subtract the oldest interval in the time interval ring buffer from the variable containing the average period length times 256. Then, the new interval is added to this variable and stored in the ring buffer. Finally, the current timestamp is stored in the timestamp ring buffer. This way, computing the average period length of the external clock signal is always of complexity O(1), independent from the number of measurements used.

5.3.3. I/O control operations

The application can request information from the driver through various I/O control operations. Table 2 lists the operations supported by the driver. `IOCTL_GET_INTERRUPT_COUNTER` and `IOCTL_GET_PARPORT_STATE` are for statistical purposes only and not strictly needed by the application during regular operation.

5.3.4. Deployment

Besides the aforementioned functionality required by software genlocking, the driver has some additional, technical properties. It is compliant with the Windows Driver Model (WDM) including plug-and-play and power management support. These enable the driver to be installed, removed, loaded and unloaded during runtime without rebooting.

5.4. SGL application

The main application contains the actual genlocking logic. It interacts with the driver, PowerStrip and the DirectDraw API to achieve software genlocking. After initialization during startup, it performs three main steps: Calibrate, BringInSync and KeepInSync (see Fig. 5). During normal operation, KeepInSync will finally loop forever. If, however, for any reason KeepInSync cannot proceed, it will return and the main loop will start over again with Calibrate. The initialization as well as the three main steps are described in the following sections.

5.4.1. Start

The initialization phase consists of several steps, which are executed sequentially in the following order:

1. Query performance counter to retrieve its precision.
2. Get handle to PowerStrip.
3. Create an instance of DirectDraw.
4. Get handle for sgl driver.
5. Initialize the driver.
6. Get average period length of external reference clock from driver.
7. Fetch modeline that is currently active (using PowerStrip).

If the steps mentioned above were executed successfully, the application priority is increased to the maximum possible. Under Windows the priority of a program consists of a process priority class and the priority of its threads. The highest process priority class is `REALTIME_PRIORITY_CLASS`, and the highest thread priority is `THREAD_PRIORITY_TIME_CRITICAL`. After increasing the priority, the main loop is entered, executing the three steps explained in the subsequent sections.

5.4.2. Calibrate

During calibration, two reference video timings are identified that are closest to the external reference signal, one slower and one faster than the signal period length. They are computed directly from the currently

Table 2
I/O control operations supported by the sgl driver

I/O control request	Purpose
`IOCTL_INITIALIZE`	Initializes operation of the driver, returns an error if initialization fails
`IOCTL_GET_AVERAGE_DIFF`	Returns the value containing the average period length of the external clock times 256 as *long long*
`IOCTL_GET_LAST_TIMESTAMPS`	Returns the last eight timestamps when the ISR was called in an array of *long long*
`IOCTL_GET_INTERRUPT_COUNTER`	Returns how often the ISR has been called since initialization as *unsigned long*
`IOCTL_GET_PARPORT_STATE`	Returns the three eight-bit registers of the parallel port in one 32-bit *unsigned long*

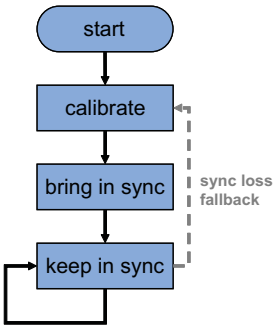

Fig. 5. SGL application control flow.

active timing/modeline by modifying its front porch. The number of lines l_{front} in the front porch of the faster timing is computed as

$$l_{\text{front}} = \left\lfloor \frac{d_{\text{ext}}}{f_{\text{perfcount}}} \cdot \frac{1}{t_{\text{line}}} - (l_{\text{active}} + l_{\text{retrace}} + l_{\text{back}}) \right\rfloor, \tag{3}$$

with t_{line} from Eq. (1). d_{ext} is the period length of the external signal in clock ticks, and $f_{\text{perfcount}}$ the resolution of the performance counter. The slow timing will have exactly one line more.

However, the measurements of the external signal as well as the timing values provided by the driver might not be totally precise. Furthermore, the clock generator of the video card as well as the clock of the computer may have a small tolerance. We cope with those inaccuracies in the next steps using appropriate timing thresholds.

5.4.3. BringInSync

This function brings the video signal of the computer in sync with the external reference signal. To accomplish this, one of the two reference timings computed by Calibrate is activated using PowerStrip, and the deviation between the video signal and the external clock is continuously measured. If the deviation drops below a threshold, BringInSync exits and passes control to KeepInSync.

The deviation is the signed distance between the local video signal and the closest reference signal. The application first waits for the vertical retrace using the DirectDraw function WaitForVerticalBlank. As soon as it returns, the timestamp of the performance counter is stored. Second, the last eight reference timestamps from the driver are fetched after an additional delay of half a period length. This delay is necessary because the closest external clock tick might occur after the local retrace. Finally, the deviation is computed as the difference between the local timestamp and the reference timestamp that has the smallest absolute distance to the local timestamp. The measurement process is illustrated on the right side of Fig. 6. Additionally, on the left side, the figure shows a situation where a local retrace is missed (at timestamp 401) because the scheduler has not reactivated the process in time. The algorithm still works correctly in that case, since it just waits for the next retrace.

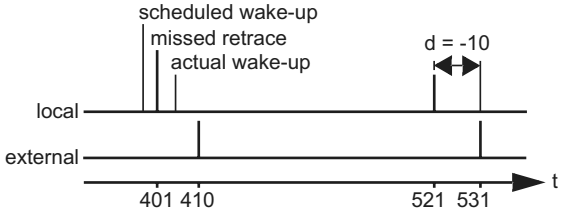

Fig. 6. Deviation (d) between the local video timing and the external reference signal.

5.4.4. KeepInSync

After the local video signal has been synchronized once, the KeepInSync function loops forever to keep the signal in sync as long as no error occurs. Like BringlnSync, it continuously measures the deviation between the local and the external signal. As soon as the sign of the deviation changes, the reference timings are swapped using PowerStrip, i.e. if the fast timing was active, the slow will become active and vice versa. In order to allow for smoother operation, a threshold can be specified which implements a hysteresis. Fig. 7 shows the two timings measured on an oscilloscope and the threshold range around the reference signal (lower signal). As long as the local video signal is within this range no actions will be taken by the algorithm. When the local signal leaves the threshold range and thus the absolute value of the deviation exceeds the threshold, the timings are swapped and the local signal will move towards the reference signal again.

The deviation is measured the same way as in BringInSync. However, here the algorithm does not wait half a period length after WaitForVerticalBlank returned, but just as long as it is necessary for the next external clock tick to arrive. Due to this optimization, reactions can be taken even faster.

6. Results

We evaluate our software genlocking approach using various graphics boards in different PC systems by measuring the maximum amount of deviation between the reference clock signal and the synchronized vertical retrace using an oscilloscope. In a common VGA connector, the vertical sync signal is directly accessible on a dedicated pin (pin 14). We built a small box with both a female and a male VGA connector and two BNC connectors, providing access to the vertical sync as well as the horizontal sync (pin 13).

Additionally, we also quantify the CPU load caused by our software. Where the resolution is not explicitly stated, the tests are conducted at a resolution of 1024×768 pixels and a refresh rate of 60 Hz. To test the stability of our system, we perform the measurements on both an idle and a busy CPU. We simulate computationally expensive applications running on the same PC by running a process consuming all available CPU time at default process priority.

Table 3 shows results using various graphics boards in a state-of-the-art PC running Windows XP SP2. Except for an older NVIDIA GeForce4 MX 420 board, genlocking is stable in all cases with a precision of ± 30 μs which is approximately ± 1.5 line with regard to the used video timing. Given the low CPU load of 2–3%, our software has a very low impact on the performance of the system. Even more important is the fact that genlocking precision does not decrease on a busy CPU, leaving the system available for running VR and other parallel graphics applications.

Note, however, that the PowerStrip tool needs an additional CPU load of up to 11% on ATI cards. This presumably results from the manner PowerStrip has to act when changing the video timing on these cards. When running on other graphics cards, PowerStrip consumes virtually no CPU time.

With the GeForce4 board, the signal gets out of sync approximately every 5–120 s, abruptly cancelling the currently drawn frame and starting a new one. This is presumably a problem of PowerStrip or the NVIDIA

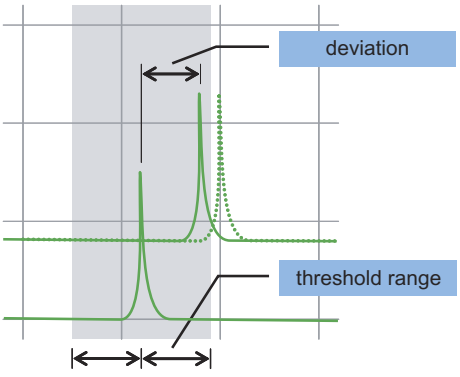

Fig. 7. Threshold for deviation.

Table 3
Different video cards on a Pentium 4 with 1.8 GHz and Windows XP SP2

Graphics card	CPU idle		CPU busy	
	Precision (μs)	Load (%)	Precision (μs)	Load (%)
NVIDIA GeForce4 MX 420	±30[a]	2	±33[a]	2
NVIDIA GeForce FX 5200	±34	2	±32	2
ATI Radeon 8500[b]	±27	2–3	±27	2–3
ATI Radeon X800[b]	±27	2–3	±27	2–3

[a] Signal sporadically out of sync.
[b] Additional CPU load of 8–11% caused by PowerStrip, see text.

display driver, arising after too many changes of the video timings. However, the signal returns back into sync showing that our algorithm effectively handles delayed wake-ups and missed timings.

To emphasize that our system really works with off-the-shelf hardware, we have done some further tests with a legacy 800 MHz Pentium 3 PC running Windows 2000 SP4 and two different Windows XP notebooks. Results are given in Tables 4 and 5. The CPU load increases due to the weaker processors, but the synchronization precision stays as good as before. We are even able to reach an accuracy of ±10 μs on a Matrox board.

We compare our solution with SoftGenLock for Linux version 2.0a3 which in spite of its alpha status appeared to be more performant than the last stable version 1.0. The tests were conducted on a 1.8 GHz Pentium 4 running Mandrake Linux 10.0 and an NVIDIA GeForce FX 5200 graphics board using the official NVIDIA drivers for Linux. We used the Linux kernel version 2.4.25 with RTAI 3.0r4 real-time extensions [15] as well as the vanilla kernels 2.4.25 and 2.6.3. Under the non-real-time kernels, best results have been achieved by using FIFO scheduling, i.e. real-time priority, and the real-time clock of the computer with a rate of 1024 ticks per second.

Using the pixel clock adjustment as a video timing adaptation strategy, SoftGenLock achieves a higher precision than our method, as can be seen in Table 6. As could be expected, the process is stable with barely no CPU usage under the real-time kernel. Similar results could be achieved under the non-real-time kernel 2.6.3 at the cost of a higher CPU load compared to our software. Kernel version 2.4.25, which contains less effective scheduling algorithms, did not yield a stable synchronization. The more flexible, alternative strategy of inserting invisible pixels has been less robust in our experiments. We could only achieve a precision of ±70 μs under the real-time kernel and just frame synchronization accuracy using the vanilla kernels.

SoftGenLock for Linux performs better than our solution when using pixel clock adjustment. However, this adaptation strategy is limited to NVIDIA cards and is incompatible with digital output devices. Also the less efficient adaptation strategy of inserting invisible pixels is incompatible with digital output devices.

Table 4
Different video cards on a Pentium 3 with 800 MHz and Windows 2000 SP4

Graphics card	CPU idle		CPU busy	
	Precision (μs)	Load (%)	Precision (μs)	Load (%)
NVIDIA GeForce3 Ti 200	±30	10%	±36	10%
Matrox MGA-G200 AGP	±10	10%	±10	10%

Table 5
Two notebooks with ATI and NVIDIA graphics chips

Notebook model	CPU idle		CPU busy	
	Precision (μs)	Load (%)	Precision (μs)	Load (%)
IBM T42 Pentium M 1.7 GHz ATI Mobility Radeon 9600[a]	±25	7	±25	7
DELL Precision M60 Pentium M 1.6 GHz NVIDIA Quadra FX Go700	±34	10	±31	10

[a] Additional CPU load of 7–11% caused by PowerStrip, see text.

Table 6
Comparison of WinSGL (for Windows) and SoftGenLock (for Linux) on a 1.8 GHz Pentium 4 PC with an NVIDIA GeForce FX 5200 graphics board

Operating system	CPU idle		CPU busy	
	Precision (μs)	Load (%)	Precision (μs)	Load (%)
Windows XP SP2	±34	2	±32	2
Linux 2.4.25 & RTAI 3.0r4	±7	2	±7	2
Linux 2.4.25	±8[a]	20	±8[a]	20
Linux 2.6.3	±7	10	±7	10

[a] Signal sporadically out of sync.

Furthermore, good performance with very low CPU load is only achieved on a real-time kernel. Therefore, SoftGenLock for Linux does not provide the flexibility of our approach as it is only suited for a very limited range of setups, consisting of a real-time enabled kernel, NVIDIA graphics cards and cathode ray tube monitors or projectors.

The presented results show that our software solution is running reliably on unmodified Windows systems with very low impact on the performance of the system and thus can be a feasible alternative to expensive hardware if no hard synchronization is required. Due to the signal propagation topology and software architecture, our approach scales to virtually any number of nodes containing a heterogeneous setup of graphics cards. It is compliant with digital output devices like DLP projectors or TFT panels and works with the VGA interface as well as with the DVI interface. Additional tests with resolutions up to 2048×1536 pixels showed that it supports higher resolutions without degradation in genlocking performance or increased CPU load.

7. Applications

The precision achieved by our software-based genlocking approach is suitable for a wide range of display applications. In combination with framelocking middleware like Net Juggler [5], it can be used to build VR installations with active stereo displays in CAVE-like immersive environments or on large tiled displays (see Fig. 8).

Recently, novel applications that require video synchronization have been developed. Waschbuesch et al. [4] synchronize projectors and cameras to acquire three-dimensional geometry of dynamic scenes using structured light patterns. Even more accurate timing is required by Cotting et al. [3] who use imperceptible high-speed structured light projections for realizing smart displays sensing the environment. WinSGL can be used there as a cost-effective alternative to genlocking hardware. In the following sections, we present these two research areas in more detail.

7.1. Structured light scanning with concurrent texture acquisition

We present the concept of our 3D video acquisition system for capturing textures and depth maps of a moving scene [4]. This data can be used to reconstruct a dynamic three-dimensional model of the scene which allows for re-rendering the video from novel views of a virtual camera as depicted in Fig. 9.

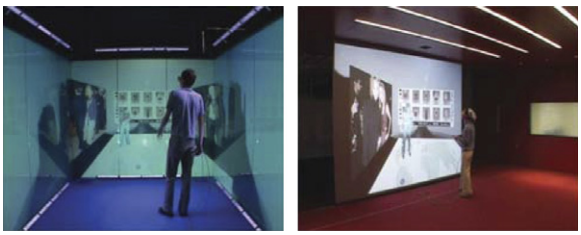

Fig. 8. Active stereo projection in immersive environments.

Fig. 9. Our 3D video system acquires color images of moving scenes (left) with corresponding depth maps (middle) which are used for re-rendering the video from novel views (right).

The basic building blocks of the setup are movable 3D video acquisition bricks, each consisting of a standard PC, several calibrated cameras and a projector. In our current setup, we are using four bricks. A single brick prototype is shown in Fig. 10. All hardware components of the system are synchronized to a common clock signal: the cameras support hardware triggering at 12 frames per second via a proprietary signal connector, the projectors are genlocked at 60 Hz using WinSGL.

Each brick acquires texture information with a color camera and depth information using stereo vision on a pair of greyscale cameras. To increase robustness and accuracy of the stereo matching, a projector is used to illuminate the scene with structured light patterns, as originally proposed by Kang et al. [16]. To avoid untexturized shadows, the scene is illuminated by patterns from all bricks at the same time. In contrast to pure structured light methods, our stereo vision approach has the advantage of being insensitive to interferences between the different projections and does not need a projector calibration.

Color textures are captured concurrently to structured light illumination by alternating between projections of a pattern and its corresponding inverse, and using an appropriately synchronized texture cameras, as illustrated in Fig. 11. The shutters of the color cameras are open during the projection phases of pairs of inverse patterns. This way, those cameras do not see the patterns emanating from the projector but only a constant white light which serves as a scene illumination preserving the original textures (see Fig. 10). Since the patterns are changing at a limited rate of 60 Hz, flickering is slightly visible to the human eye. Alternative solutions using imperceptible structured light [3] do not show any flickering, but require faster, more sensitive and therefore more expensive cameras for reliable stereo depth extraction.

Fig. 10. 3D video brick with cameras and projector (left), simultaneously acquiring textures (middle) and structured light patterns (right).

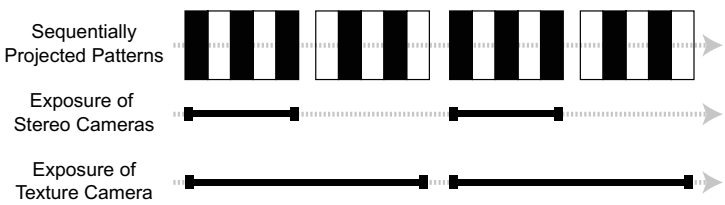

Fig. 11. Camera exposure with inverse pattern projection.

7.2. Smart displays using imperceptible embedding of patterns

As an enabling technology for smart displays, we use a method to invisibly embed arbitrary binary patterns into ordinary color images displayed by unmodified, off-the-shelf Digital Light Processing™ (DLP™) projectors (see Fig. 12). In DLP projectors, each displayed pixel is generated by a tiny micro-mirror, tilting towards the screen to project light and orienting towards an absorber to keep the pixel dark. Gradations of intensity values are created by flipping the mirror in a fast modulation sequence (see Fig. 13), while a synchronized filter wheel rotates in the optical path to generate colors. By carefully selecting the projected intensities, one can control whether the mirrors for the corresponding pixels project light onto the scene during a predefined exposure time slot of a synchronized camera. The core idea of the imperceptible pattern embedding is a dithering of the projected images using color sets appearing either bright or dark in the triggered camera, depending on the chosen pattern. Such color sets can be obtained for conventional DLP projectors by analyzing their intensity patterns using a synchronized camera. For more details refer to [3].

This imperceptible embedding technique [3] allows to control the appearance of the projection surface during a camera exposure in a setting, where the projectors are genlocked and synchronized to the camera-based acquisition using WinSGL. This control is done at the scale of individual projector pixels and in an imperceptible way, thus allowing structured light approaches not noticeable by the user. Therefore, during operation of our smart displays, the environment can be imperceptibly scanned and novel environment-aware display functionality can be achieved, enhancing existing display technology in the areas of user-friendliness, flexibility, scalability and interaction.

By introducing vision-based structured light scanning imperceptibly into office environments and human-inhabited mixed and augmented reality settings, we can explore various application scenarios and achieve novel display opportunities like undistorted views on arbitrary geometries [3], self-calibrating displays dynamically adapting to setup changes [17], encumbrance-free tracking techniques (see Fig. 14) and environment-aware adaptivity, as shown in Figs. 15 and 16 [18].

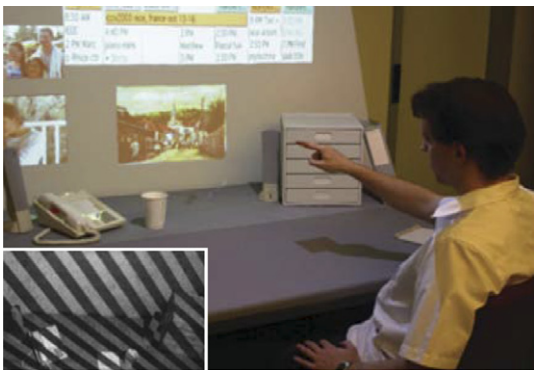

Fig. 12. Projected display containing imperceptible pattern (see inset).

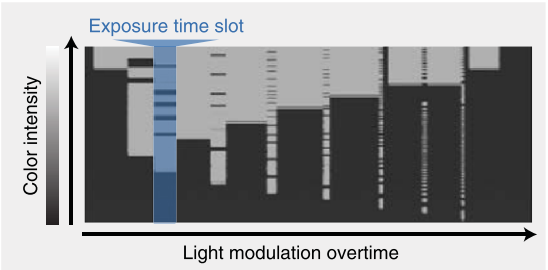

Fig. 13. Light modulation of a DLP projector.

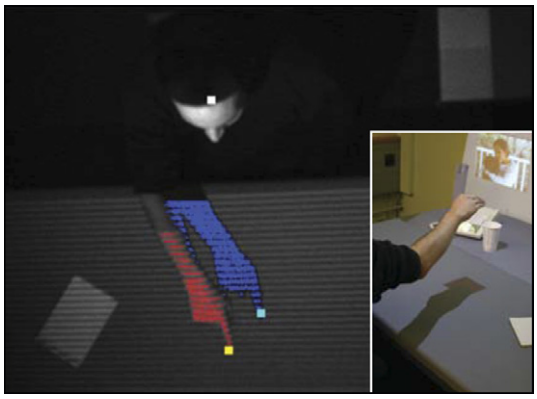

Fig. 14. Encumbrance-free interaction using imperceptible structured light (inset illustrates what the user sees).

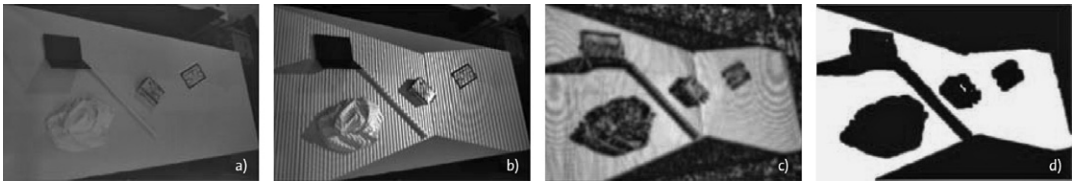

Fig. 15. Analysis of the projection surface properties for the environment-aware display adaptivity: (a) Snapshot of the scene taken from a camera (visible to human). (b) Captured pattern image (imperceptible to human). Note that in this setting the projector frusta do not cover the entire table. (c) Magnitude of the filter response. (d) Parts of the environment suited for projection (shown in white).

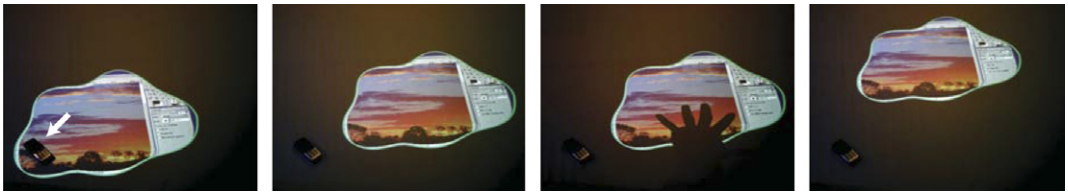

Fig. 16. Environment-aware display adaptivity based on imperceptible structured light scanning enabled by synchronized projection and capture. The display reacts to the colliding cell phone and subsequently avoids the shadow cast by the hand by dynamically relocating to an adequate area determined by the scanning of the projection surface.

8. Conclusion and future work

We presented a novel software genlocking scheme for Microsoft Windows operating systems imposing no additional requirements like real-time extensions. Our system is compatible with most graphics boards and cooperates smoothly with digital output devices. The implemented software performs accurately and reliably while only requiring a low CPU load. Thus, our solution can be used as an inexpensive alternative to hardware genlocking when no pixel-accurate synchronization is needed. For the benefit of the parallel graphics community, we have published our code as open source (see http://graphics.ethz.ch/winsgl).

Additional improvements could be accomplished in the following areas:

Frame- and swaplocking: instead of depending on third party software, we would like to natively provide frame- and swaplocking mechanisms based on network barriers in our software.

Multi-head support: We would like to support multi-head systems which include multi-head graphics cards as well as systems with more than one graphics card. PowerStrip has support for multi-head configurations which leaves the detection of the vertical retrace as a last obstacle to implement multi-head support.

Parallel port alternatives: the standard parallel port is becoming legacy and is already missing in some new computers. Porting the driver to another interface, like the USB or FireWire port for example, would be desirable but imposes some difficult technical challenges to guarantee an accurate synchronization.

References

[1] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti, Surround-screen projection-based virtual reality: the design and implementation of the CAVE, in: SIGGRAPH'93, 1993, pp. 135–142.
[2] D. Sandin, T. Margolis, J. Ge, J. Girado, T. Peterka, T.A. DeFanti, The VarrierTM autostereoscopic virtual reality display, ACM Transactions on Graphics 24 (3) (2005) 894–903.
[3] D. Cotting, M. Naef, M. Gross, H. Fuchs, Embedding imperceptible patterns into projected images for simultaneous acquisition and display, in: ISMAR'04, IEEE Computer Society Press, 2004, pp. 100–109.
[4] M. Waschbuesch, S. Wuermlin, D. Cotting, F. Sadlo, M. Gross, Scalable 3D video of dynamic scenes, The Visual Computer (Special Issues for Pacific Graphics 2005), Springer, pp. 629–638.
[5] J. Allard, V. Gouranton, L. Lecointre, E. Melin, B. Raffin, Net Juggler: running VR Juggler with multiple displays on a commodity component cluster, in: IEEE VR'02, 2002, pp. 273–274.
[6] M. Waschbuesch, D. Cotting, M. Duller, M. Gross, WinSGL: software genlocking for cost-effective display synchronization under Microsoft Windows, in: Proceedings of the Sixth Eurographics Symposium on Parallel Graphics and Visualization, 2006, pp. 111–118.
[7] B. Schaeffer, C. Goudeseune, Syzygy: native PC cluster VR, in: IEEE VR'03, 2003, pp. 15–22.
[8] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, J.T. Klosowski, Chromium: a stream-processing framework for interactive rendering on clusters, in: of SIGGRAPH'02, 2002, pp. 693–702.
[9] A. Streit, R. Christie, A. Boud, Understanding next-generation VR: classifying commodity clusters for immersive virtual reality, in: GRAPHITE'04, 2004, pp. 222–229.
[10] M. Bues, R. Blach, S. Stegmaier, U. Hafner, H. Hoffmann, F. Haselberger, Towards a scalable high performance application platform for immersive virtual environements, in: Immersive Projection Technology and Virtual Environements 2001, Springer, 2001, pp. 165–174.
[11] B. Schaeffer, Networking and management frameworks for cluster-based graphics, in: Virtual Environment on a PC Cluster Workshop, Protvino, Russia, 2002.
[12] J. Allard, V. Gouranton, G. Lamarque, E. Melin, B. Raffin, SoftGenLock: active stereo and genlock for PC cluster, in: EGVE'03, 2003, pp. 255–260.
[13] U. Wössner, M. Aumüller, Software-based genlock for active stereo NVIDIA cards, http://www.hlrs.de/organization/vis/people/aumueller/genlock.
[14] EnTech Taiwan, PowerStrip Version 3.61, http://www.entechtaiwan.net/util/ps.shtm.
[15] Real-Time Application Interface for Linux, http://www.rtai.org.
[16] S. Kang, J. Webb, L. Zitnick, T. Kanade, A multi-baseline stereo system with active illumination and real-time image acquisition, in: ICCV'95, 1995, pp. 88–93.
[17] D. Cotting, R. Ziegler, M. Gross, H. Fuchs, Adaptive instant displays: continuously calibrated projections using per-pixel light control, in: Proceedings of Eurographics 2005, 2005, pp. 705–714.
[18] D. Cotting, M. Gross, Interactive environment-aware display bubbles, in: Proceedings of UIST'06, 2006.