Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

WinSGL: Software Genlocking for Cost-Effective Display Synchronization under Microsoft Windows

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

Computer Graphics Laboratory, ETH Zurich, Switzerland

Abstract

This paper presents the first 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: First, an external synchronization signal is transmitted over interrupt lines to a dedicated driver. Second, a user-space application performs the synchronization by inserting or removing lines to the invisible part of the image. Robustness to potential frame losses is achieved through continuous consistent timestamping. Tests yield an accuracy of up to ± ½ 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 and spatially immersive displays using commodity PC clusters.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Distributed/network graphics H.5.1 [Information Interfaces and Presentation]: Artificial, augmented, and virtual realities

1. Introduction

Many VR installations combine video signals from multiple graphics engines to build large tiled or spatially immersive displays such as CAVE™ installations [CNSD93] or the Varrier™ display [SMG*05]. To achieve a high visual quality, all involved video output devices require exact frame synchronization, i.e. genlocking. 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. Because timing is critical, such software is to date only available for the Linux operating system which can be easily extended with real-time capabilities.

This paper 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 [AGL*02] which already provide an application-level frame synchronization. By additionally synchronizing the video signals, genlocking improves the resulting display quality. Besides the aforementioned areas which have output synchronization as their major requirement, other projects like structured light scanning systems [CNGF04, WWC*05] require synchronization between video output devices and cameras. Again, our software can be used there as an inexpensive alternative to specialized hardware.

The software architecture comprising a system driver and a synchronization process permits transparent extension of any application with genlocking capabilities at run time. A hardware abstraction layer makes our system applicable with 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 newer digital output devices like LCDs

or DLP projectors which are quite sensitive to timing adjustments.

The paper 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 problems specific to digital output devices. Those build the basis for our implementation including both software and hardware design. Implementation issues are described in detail as we plan to make our software publicly available as open source. We evaluate our approach on a variety of hardware platforms and provide comparisons with existing solutions. Finally, we summarize our achievements and give an outlook to future work.

2. Related work

Genlocking has been in use in video studios for a long time in order to allow for clean cutting, blending and other editing operations. Apart from specialized hardware, home computers like the Amiga supported an external video clock signal and provided cheap alternatives. Some are still in use today.

Traditionally, VR installations have been built using high-end graphics mainframes such as those produced by SGI. Apart from 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 thus come at a high price.

As the graphics performance of conventional PCs is rapidly increasing, expensive graphics mainframes are successively replaced by powerful low-cost PC clusters. Software systems like Net Juggler [AGL*02], Syzygy [SG03] or Chromium [HHN*02] provide platforms for distributed execution of virtual reality applications. See Streit et al. [SCB04] for an extensive overview.

Designed as high-level middleware platforms, they do not include sophisticated display synchronization methods. Net Juggler [AGL*02] 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. [BBS*01] or Scheffer et al. [Sch02] 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.

A cheap alternative are software genlocking solutions which try to synchronize the video timing as close as possible to a reference clock signal, usually fed through the parallel port. SoftGenLock [AGL*03] 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. [WA], also for Linux only. Besides their restriction to the operating system, both systems only support a limited set of graphics hardware: either NVIDIA cards only [WA] or boards compatible to the VGA standard on the register level [AGL*03], both excluding for example modern ATI boards. Moreover, their synchronization strategies are incompatible with most LCD and DLP display devices yielding distorted images.

Recently, some novel applications requiring video synchronization have been developed. Waschbüsch et al. [WWC*05] 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. [CNGF04] who use imperceptible high-speed structured light projections for alignment of and interaction with projected virtual displays. The genlocking hardware those systems are using can be transparently replaced by our software solution.

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.

3.1. Image generation

Throughout this paper 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. graphics card, and the display device, e.g. monitor or projector.

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. Figure 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—referred to as the *active period*—is surrounded by an invisible area which is made up by the *front porch*, the *back porch* and the *retrace* (or *sync width* when referring to the length of 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 Figure 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 thus effec-

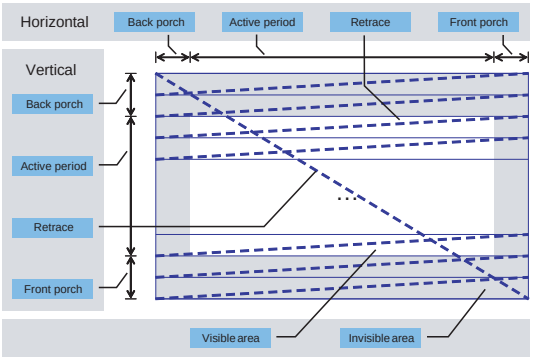

Figure 1: *Video timing.*

tively 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, thus avoiding smears.
- During the **retrace** period the sync signal is active a certain amount of time (sync width) and the ray moves to the other side of the screen.
- 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 front porch, retrace and back porch, they are often used to transmit non-image information. Teletext for example is transmitted during the vertical retrace of a television signal.

Horizontally, the length of these periods is measured in pixels. Each pixel has a length (the reciprocal of the pixel clock) and, thus, the length of a horizontal period is the number of pixels multiplied with the length of one pixel. One full line consists of one horizontal active period, one horizontal front porch, one horizontal retrace and one horizontal back porch. Equation (1) shows how the length t_{line} of one full line can be computed, where $p_{<index>}$ is the number of pixels of the corresponding horizontal timing component. The duration of a line is the reciprocal of the scan rate:

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

Vertically, the length is measured in lines. The length of a vertical period is the number of lines it consists of multiplied with the length 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 length t_{frame} of one frame, thus, can be computed as shown in equation (2), where $l_{<index>}$ is the number of lines of the corresponding vertical timing component. The duration of a frame is the reciprocal of the refresh rate:

$$t_{frame} = t_{line} \cdot (l_{active} + l_{front} + l_{retrace} + l_{back}). \quad (2)$$

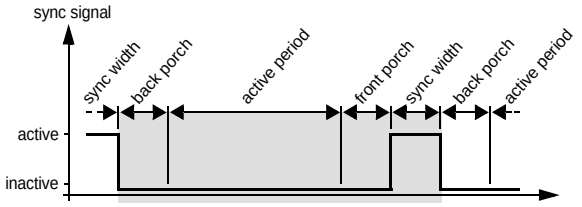

Figure 2: *Sequence of timing components.*

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

3.2. Synchronizing multiple video cards

From the properties of 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 front porch, back porch et cetera). Then, in theory, the shift between the cards should be eliminated and they should be in sync. However, this approach fails. The pixel clock is derived from the video card's 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 (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 seconds.

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. Secondly, 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 which might arise with software genlocking and digital out-

put 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 Hz to 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 low, the parallel port still found in most computers is an optimal choice. Its pins can be accessed directly at a certain memory mapped address thus making it fast and easy 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.

Interrupt vs. polling. SoftGenLock for Linux [AGL*03] 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 expected clock signal. However, the software then 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 pin such as the parallel port's ACK line, allows for low response times and thus provides an efficient and simple way to read the signal with a high precision.

Signal propagation topology. The existing SoftGenLock for Linux uses a setup with one master sending its internal sync signal to multiple slaves. Though sending the signal itself is easy, 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, causing problems on their side. Since software genlocking itself already poses a multitude of challenges, the decision was made to use a dedicated clock generator as reference and run all computers as slaves.

4.2. Measuring the local timing

Measuring the local timing basically comes down to detecting 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 external 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 timestamping algorithms.

4.3. Modifying 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 length 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 board.

According to our experience, digital display devices do not tolerate 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. On 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 its reactions, the arrows indicate the direction of the shift. The granularity for these changes were eight pixels (horizontal) and one line (vertical). The reactions could already be noticed after increasing or decreasing one step of this granularity. Similar effects could be 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.

Table 1: *Reaction of a NEC LT 240K DLP projector to different changes to invisible timing components. The direction of the arrow indicates the observed shift of the picture.*

	horizontal		vertical	
	increase	decrease	increase	decrease
front porch	←	→	stable	stable
back porch	→	←	↓	↑
sync width	→	←	↓	↑

Modification of the pixel clock did not work at all with digital display devices. They reacted with jitter and distortion of the whole image.

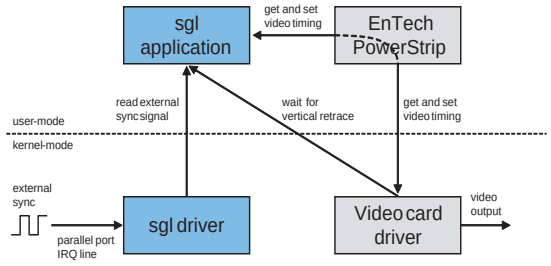

Figure 3: *WinSGL architecture.*

Thus, it is obvious that the only suitable method for changing the local video timing is modification of the vertical front porch. It is a very clean way, since the change occurs after the visible area has been sampled, thus 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.

5. Implementation

According to the previous considerations we implemented a robust software genlock solution running on unmodified Windows 2000 or Windows XP systems. Figure 3 shows the architecture of the solution and the interaction between all components. The shaded 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 the video card driver and does the necessary adjustments by programming the graphics board. The graphics hardware is accessed through the third-party utility EnTech PowerStrip 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 its parallel port's interrupt line at pin 10 on the commonly used 25 pin D-SUB connector. Such a 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.

5.2. Hardware Abstraction Layer

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

DirectDraw. The hardware-independent DirectDraw function *WaitForVerticalBlank* is used by our solution to detect vertical retraces.

PowerStrip. PowerStrip [EnT] 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 GPU manufacturer's graphics driver. 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 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 computer's parallel port. It is compliant with the Windows Driver Model (WDM) including plug-and-play and power management support and thus can be loaded and unloaded during runtime without rebooting.

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 8 timestamps when the interrupt occurred and a ring buffer of the last 256 interval lengths between two interrupts. The former is needed 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 external clock's period length which is represented by the sum of all 256 values. At each interrupt call the sum is updated by adding the current interval length and substracting the oldest one, yielding a time complexity of O(1).

Time is measured in ticks of the computer's 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.

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 it performs three main steps: *Calibrate*, *BringInSync* and

KeepInSync. 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*.

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's period length. They are computed directly from the currently active timing by modifying its front porch. The number of lines l_{front} in the front porch of the faster timing is computed as

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

with t_{line} from equation (1). d_{ext} is the period length of the external signal in clock ticks and $f_{perfcount}$ 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 video card's clock generator as well as the computer's clock may have a small tolerance. We cope with those inaccuracies in the next steps using appropriate timing thresholds.

BringInSync. This function brings the computer's video signal in sync with the external reference signal. To accomplish this, one of the two reference timings computed by *Calibrate* is activated 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 8 reference timestamps from the driver are fetched after an additional delay of half a period length. This delay is necessary as the closest external clock tick might occur after the local retrace. Finally, the deviation is computed using the reference timestamp with the minimum absolute distance to the local timestamp. The measurement process is illustrated on the right side of Figure 4. Additionally, on the left side, it 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.

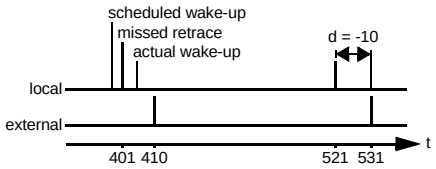

Figure 4: *Deviation (d) between local video timing and external reference signal.*

KeepInSync. After the local video signal has been synchronized once, this function loops forever to keep it in sync as long as no error occurs. Like *BringInSync* it continuously measures the deviation between the local and the external signal. As soon as its sign changes, the reference timings are swapped, 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. Figure 5 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.

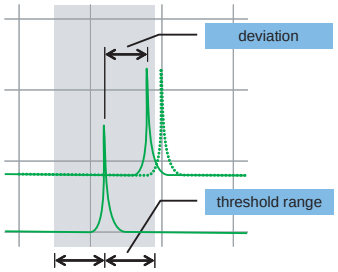

Figure 5: *Threshold for deviation.*

The deviation is measured like in *BringInSync*. However, the algorithm here 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. Hence, reactions can be taken even faster.

6. Results

We evaluate our software genlock using various graphics boards in different PC systems by measuring the maximum amount of deviation between the reference clock and the synchronized vertical retrace. Additionally we quantify the CPU load caused by our software. All tests were conducted at a resolution of 1024 by 768 pixels and a refresh rate of 60 Hz. To test the stability of our systems 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 2 shows results using various graphics boards in a state-of-the-art PC running Windows XP SP2. Except for an older NVIDIA GeForxe4 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 system's performance. Even more important is the fact that genlocking precision does not decrease

on a busy CPU, leaving the system available for running VR applications.

Table 2: *Different video cards on a Pentium 4 with 1.8 GHz and Windows XP SP2.*

	CPU idle		CPU busy	
Graphics card	precision	load	precision	load
NVIDIA GeForce4 MX 420	± 30 µs+	2%	± 33 µs+	2%
NVIDIA GeForce FX 5200	± 34 µs	2%	± 32 µs	2%
ATI Radeon 8500*	± 27 µs	2-3%	± 27 µs	2-3%
ATI Radeon X800*	± 27 µs	2-3%	± 27 µs	2-3%

+ signal sporadically out of sync
* additional CPU load of 8-11% caused by PowerStrip, see text

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 use when changing video timing of this hardware. 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 to 120 seconds, abruptly cancelling the currently drawn frame that and starting a new one. This is presumably a problem of PowerStrip or the NVIDIA display driver arising after too frequent changes of the video timing. 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 3 and 4. 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.

Table 3: *Different video cards on a Pentium 3 with 800 MHz and Windows 2000 SP4.*

	CPU idle		CPU busy	
Graphics card	precision	load	precision	load
NVIDIA GeForce3 Ti 200	± 30 µs	10%	± 36 µs	10%
Matrox MGA-G200 AGP	± 10 µs	10%	± 10 µs	10%

We compare our solution with SoftGenLock for Linux version 2.0a3 which in spite of its alpha status appeared to be more performant then the last stable version 1.0. The tests were conducted on a 1.8 GHz Pentium 4 running Mandrake Linux 10.0 and a NVIDIA GeForce FX 5200 graphics board using the official NVIDIA drivers for Linux. We used

Table 4: *Two notebooks with ATI and NVIDIA graphics chips.*

	CPU idle		CPU busy	
Notebook model	precision	load	precision	load
IBM T42 Pentium M 1.7 GHz ATI Mobility Radeon 9600*	± 25 µs	7%	± 25 µs	7%
DELL Precision M60 Pentium M 1.6 GHz NVIDIA Quadro FX Go700	± 34 µs	10%	± 31 µs	10%

* additional CPU load of 7-11% caused by PowerStrip, see text

the Linux kernel version 2.4.25 with RTAI 3.0r4 real-time extensions [RTA] 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 computer's real-time clock with a rate of 1024 ticks per second.

Using pixel clock adjustment as video timing adaptation strategy, SoftGenLock achieves a higher precision than our method, as can be seen in table 5. 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 containing less effective scheduling algorithms did not yield a stable synchronization.

Notice, however, that the employed adaptation strategy is proprietary for NVIDIA cards only and thus does not provide the flexibility of our approach. 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. Moreover, both timing modes are incompatible with most digital output devices.

Table 5: *Comparison of WinSGL and SoftGenLock for Linux on a 1.8 GHz Pentium 4 with NVIDIA GeForce FX 5200 graphics board.*

	CPU idle		CPU busy	
Operating System	precision	load	precision	load
Windows XP SP2	± 34 µs	2%	± 32 µs	2%
Linux 2.4.25 & RTAI 3.0r4	± 7 µs	2%	± 7 µs	2%
Linux 2.4.25	± 8 µs+	20%	± 8 µs+	20%
Linux 2.6.3	± 7 µs	10%	± 7 µs	10%

+ signal sporadically out of sync

The presented results show that our software solution is running reliably on unmodified Windows systems with very low impact on the system's performance and thus can be a

feasible alternative to expensive hardware if no hard synchronization is required. Additional tests with resolutions up to 2048 by 1536 pixels showed a similar behavior. The achieved precision is suitable for most display applications. In combination with framelocking middleware like Net Juggler [AGL*02], it could be used to build VR installations with active stereo displays. First tests have shown that it also works in synchronized projection and acquisition systems [CNGF04, WWC*05].

7. 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 any VGA graphics board and cooperates smoothly with digital output devices. The implemented software performs accurately and reliably while only requiring a low CPU load. We plan to publish our code as open source. Thus, our solution can be used as an inexpensive alternative to hardware genlocking when no pixel-accurate synchronization is needed.

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 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 port for example—would be desirable but imposes some difficult technical challenges to guarantee an accurate synchronization.

References

[AGL*02] Allard J., Gouranton V., Lecointre L., Melin E., Raffin B.: Net Juggler: running VR Juggler with multiple displays on a commodity component cluster. In *Proc. IEEE VR '02* (2002), pp. 273–274.

[AGL*03] Allard J., Gouranton V., Lamarque G., Melin E., Raffin B.: SoftGenLock: active stereo and genlock for PC cluster. In *Proc. EGVE '03* (2003), pp. 255–260.

[BBS*01] Bues M., Blach R., Stegmaier S., Hafner U., Hoffmann H., , Haselberger F.: Towards a scalable high performance application platform for immersive virtual environements. In *Immersive Projection Technology and Virtual Environements 2001*. Springer, 2001, pp. 165–174.

[CNGF04] Cotting D., Naef M., Gross M., Fuchs H.: Embedding imperceptible patterns into projected images for simultaneous acquisition and display. In *Proc. ISMAR '04* (2004), IEEE Computer Society Press, pp. 100–109.

[CNSD93] Cruz-Neira C., Sandin D. J., DeFanti T. A.: Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *Proc. SIGGRAPH '93* (1993), pp. 135–142.

[EnT] EnTech Taiwan: PowerStrip. http://www.entechtaiwan.net/util/ps.shtm.

[HHN*02] Humphreys G., Houston M., Ng R., Frank R., Ahern S., Kirchner P. D., Klosowski J. T.: Chromium: a stream-processing framework for interactive rendering on clusters. In *Proc. SIGGRAPH '02* (2002), pp. 693–702.

[RTA] Real-Time Application Interface for Linux. http://www.rtai.org.

[SCB04] Streit A., Christie R., Boud A.: Understanding next-generation VR: classifying commodity clusters for immersive virtual reality. In *Proc. GRAPHITE '04* (2004), pp. 222–229.

[Sch02] Schaeffer B.: Networking and management frameworks for cluster-based graphics. In *Proc. Virtual Environment on a PC Cluster Workshop* (Protvino, Russia, 2002).

[SG03] Schaeffer B., Goudeseune C.: Syzygy: native PC cluster VR. In *Proc. IEEE VR '03* (2003), pp. 15–22.

[SMG*05] Sandin D., Margolis T., Ge J., Girado J., Peterka T., DeFanti T. A.: The VarrierTM autostereoscopic virtual reality display. *ACM Transactions on Graphics 24, 3* (2005), 894–903.

[WA] Wössner U., Aumüller M.: Software-based genlock for active stereo NVIDIA cards. http://www.hlrs.de/organization/vis/people/aumueller/genlock.

[WWC*05] Waschbüsch M., Würmlin S., Cotting D., Sadlo F., Gross M.: Scalable 3D video of dynamic scenes. *The Visual Computer 21, 8–10* (2005), 629–638.