Understanding Video Latency

What is video latency and why do we care about it?

By Pete Eberlein, Sensoray Company, Inc.

When choosing components for a video system, it is important to understand how the system is designed and how each piece affects the overall *latency* in the system – how long it takes a single frame of video to go from the camera to the display. Overall latency is important because there is a delay between what is seen on the display and what is actually happening in front of the camera.

In most cases, users want latency to be as low as possible, which is why one sees so many products advertised as "low-latency." For example, an operator manipulating a remote device does not want a delay in seeing the results of his movements. Similarly, those participating in a two-way video conference do not want delays between speaking and hearing a response.

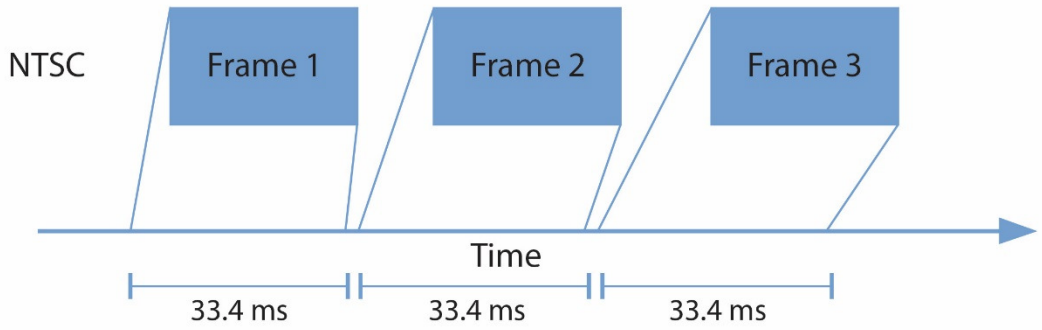
Achieving *low* latency, usually defined as less than 100 milliseconds (ms), is most important for operating remote devices, video conferencing, streaming live events, and computer vision. Recording and streaming recorded events can tolerate higher latency.

This article will explain how each piece of a video system contributes to latency, methods that some products use to achieve low-latency, and several methods for measuring a system's latency.

Understanding how video signals are displayed

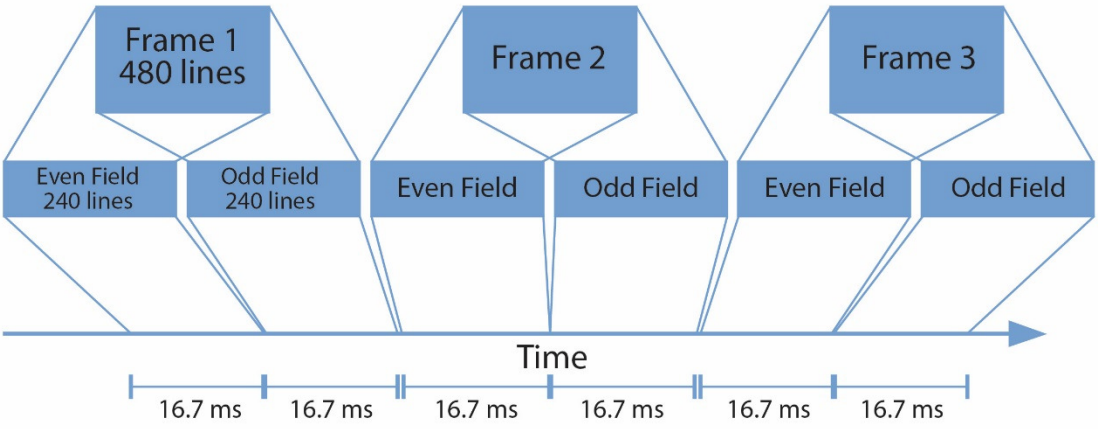
To understand latency it is important to know more about how video signals are displayed. Video is often described in terms of frames per second (FPS). Two different standard-definition analog video modes are used to describe how this works. North and South America use the NTSC (National Television System Committee) definition, which is 29.97 FPS. This means there are 29.97 frames displayed during a one-second period. The definition used in all other overseas locations, known as PAL (Phase Alternating Line), is usually 25 frames per second, although variants do exist. Therefore, when transmitting a video signal, each frame takes 1/29.97 seconds (about 33.4ms) to be transferred for NTSC, and 1/25 seconds (40ms) for PAL. See **Figure 1** for an illustration of this concept for NTSC.

Figure 1



Each frame is comprised of 480 lines of active video (525 including blanking) for NTSC, and 576 lines (625 including blanking) for PAL. Both NTSC and PAL video are interlaced, meaning there are two fields for each frame. One field stores the even lines and the other field stores the odd lines in the frame. A video camera does not capture both fields simultaneously; instead, the camera shutter captures each field at twice the frame rate, so the fields-per-second is 59.94, and in the video signal each field takes 1/59.94 seconds (about 16.7ms) to be transferred. For NTSC each field is 240 lines of active video, and for PAL, 288 lines. **Figure 2** illustrates this concept.

Figure 2



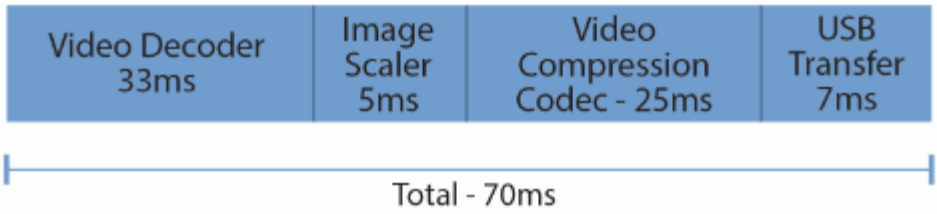
The way each stage in a system processes the video affects the latency in a video processing system. For example, suppose we have a video system composed of a *video decoder* (which translates an analog video to a digital signal) an *image scaler*, a *video compression codec*, and a *USB interface*.

The video decoder must process the whole frame (both fields) and then writes the interlaced image into memory. The image scaler must wait for the whole frame to be available before it can

resize the image, but the operation can be done fairly quickly. The video compression codec must wait for the scaled image to be available before compression; the compression operation is fairly lengthy. Once the video frame has been compressed, the USB interface transfers the compressed video frame data to the host computer. The total time taken for the video frame to be acquired by the system and transferred to the host computer is the sum total of all these stages.

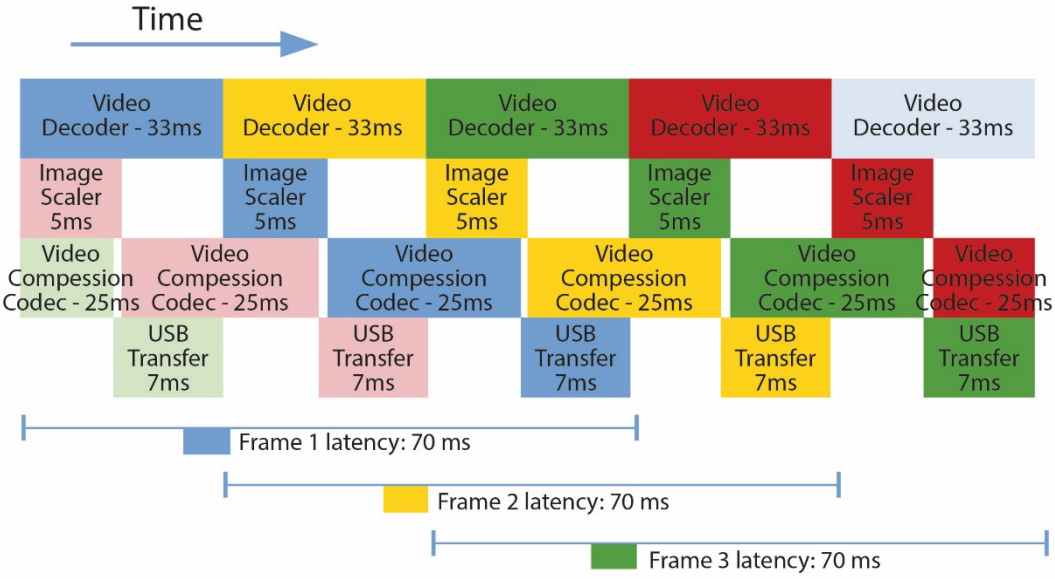
As shown in **Figure 3**, the video decoder requires 33ms per frame; the image scaler requires 5ms per frame; the video compression codec requires 25ms per frame; and the USB interface requires 7ms per compressed frame – for a total of 70ms.

Figure 3



Each stage works independently of the other stages, and while one frame is being processed by the image scaler, the next frame is being captured by the video. **Figure 4** shows each stage on a separate line to illustrate how the stages work in parallel. Each frame arrives on the host with a latency of 70ms, at 33ms intervals.

Figure 4 - Video stages in parallel



When choosing a video system, operators typically specify how much latency they are willing to tolerate. So let's suppose there is a requirement for reduced-latency display (also called reduced-latency preview) – meaning the operator needs to view the video at the same time as it is being recorded.

The device could bypass the image scaler and video compression codec, and instead send the uncompressed frame directly to the host. Sending the uncompressed frame over USB takes more time than a compressed frame, since the number of bytes is larger for an uncompressed frame. In this case the video decoder requires 33ms per frame and the USB interface requires 20ms per uncompressed frame, for a total of 53 ms.

Is there a way to improve this? One way is with a video decoder that allows us to know when each field is complete by sending an interrupt. This would allow the system to quickly reassemble two fields into a frame on the host. We would be able to start sending one field early while the video decoder is processing the next field. In this case the video decoder requires 33ms per frame, or 17ms per field. The USB interface requires 20ms per uncompressed frame, or 10ms per field, so the whole frame is now transferred – with a 10ms improvement. The latency is reduced by half.

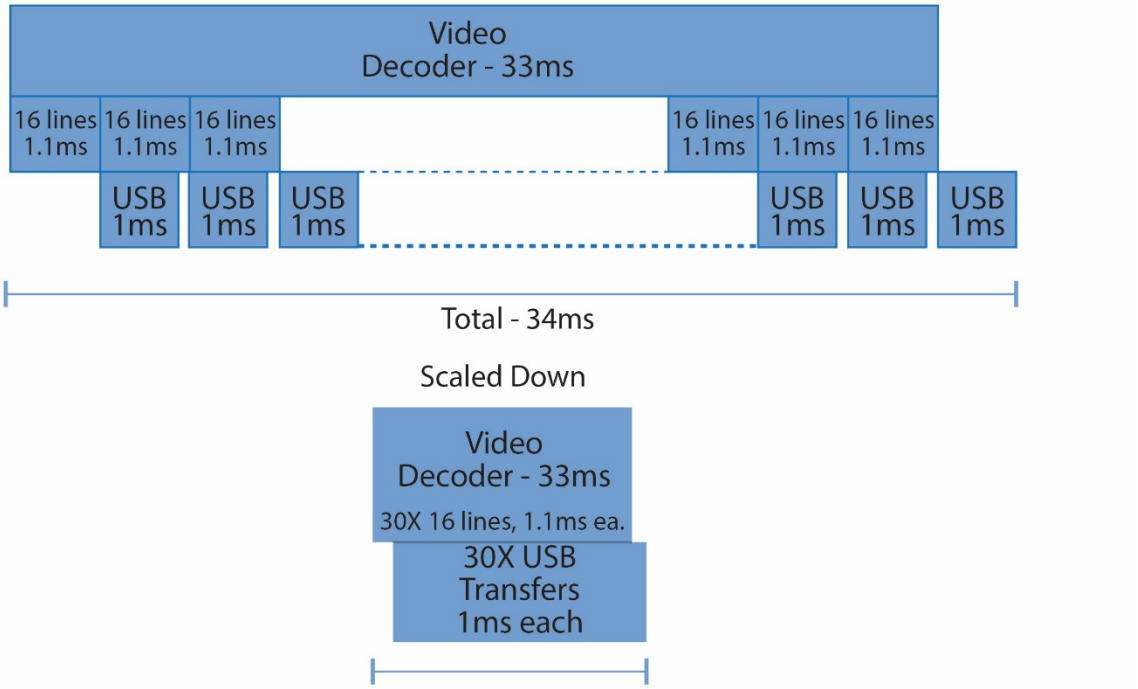
Some video decoders can be configured to interrupt at a specific line interval, and this would allow even lower latency by processing small chunks of lines instead of whole fields or frames. For example, were we to program the video decoder to deliver an interrupt every 16 lines, and transfer each chunk of lines separately over USB, latency can be reduced even further.

Figure 5 illustrates the concept. The video frame is divided into chunks of 16 lines, with a whole frame consisting of 30 chunks. Since the video frame is 33ms, each chunk of 16 lines would take about 1.1ms to process, and each 16-line chunk would take about 1ms to transfer over USB. Due to the increased frequency of starting a new USB transfer for each chunk of lines, the efficiency may be slightly reduced, and require more system processing power. The video decoder requires 1.1ms per 16 lines. The USB interface requires 1ms per 16 lines.

Now a single video frame can be transferred in 34ms. In general, latency can be reduced by processing content in smaller pieces. Despite drawing the 30X USB transfers 1ms each as a single block, the USB transfer duration per frame is still roughly 33ms, not 30ms, since there are .1ms gaps between each small transfer.

By choosing a video encoding that is limited to working at frame granularity, the latency cannot go below one frame duration. Latency can be reduced by choosing an encoding that can work at field- or line-granularity.

Figure 5



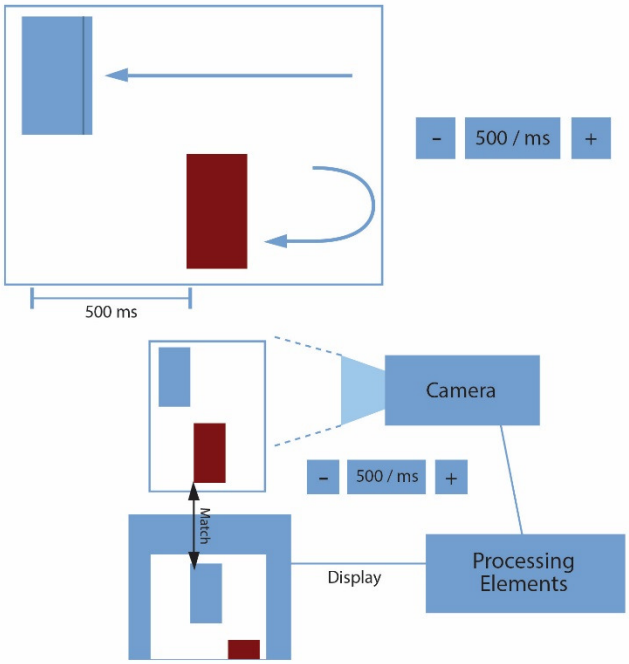
Measuring latency in a system

A variety of methods can be used to determine whether a system's latency meets requirements. Developed over time, such methods reflect advances made as researchers discovered better and more accurate ways to perform the measurement.

One method is a program that displays a pair of colored vertical bars that oscillate back and forth. The phase of the second bar is adjustable relative to the first, so the motion of the second bar is delayed by a certain amount of time, simulating latency.

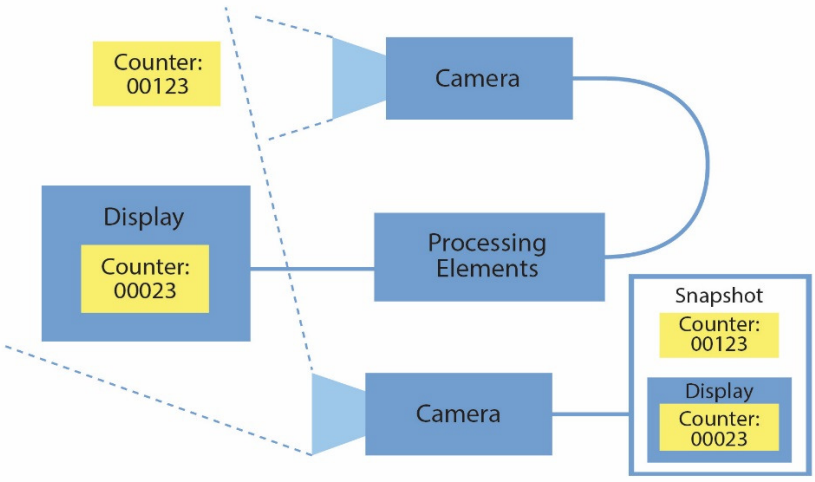
When the camera is pointed at the first blue bar, the display shows the same blue bar delayed by the system latency. The goal is to make the position of the red bar in the program window match the blue bar in the display, by adjusting the simulated latency value in the program. When the bars are perfectly in sync, the simulated latency gives an estimate of the system latency. The method is quite effective, however, tuning the latency value while comparing two oscillating bars can become tiresome. **Figure 6** illustrates this method.

Figure 6



Another method is to use a counter to calculate the time difference between the camera and the display. As shown on **Figure 7**, the camera is pointed at a 1kHz counter situated near the display. As the system is running, the operator takes a snapshot of both the counter value (A) and the counter value in the displayed video (B). To calculate the total latency, one looks at the snapshot and subtracts the counter value (B) from the counter value (A). Since the counters are changing quickly, the snapshot allows taking a sample of both capture and display counters simultaneously. The snapshot shows that by subtracting the Display Counter (B) from Counter (A) gives the latency, which is 100ms for this illustration.

Figure 7

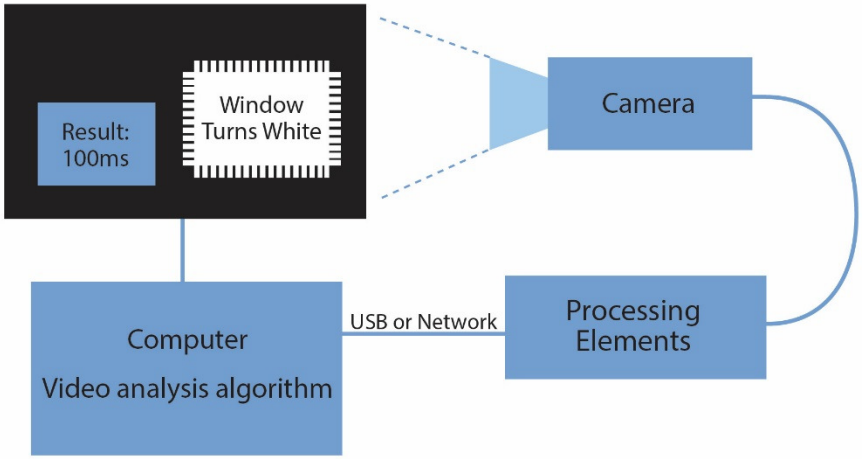


The method shown on **Figure 8** requires more control over the final stages in the processing pipeline, and a computer to run the measurement program. The camera source for the video processing pipeline is aimed at the computer monitor. The end of the pipeline must terminate at the computer, either through a network or USB/PCI/PCIe interface, and the program must be able to receive the video stream through a network socket interface (UDP or RTSP) or a video streaming interface (Video4Linux2 or Directshow.)

The computer program uses a low-latency API (such as the xv library on Linux) to display a window where it displays a black colored background normally. When the program initiates a latency measurement, it records the initial timestamp (A), and the display window turns white. The video pipeline camera is continuously capturing the window on the monitor, and the white image will quickly wind its way through the video processing pipeline and into the computer.

Meanwhile, the measurement program is capturing the video stream from the network or video interface, recording the timestamp (B) of each frame as it arrives. If the video stream is compressed, it is decompressed using the avcodec library into an uncompressed video image. The measurement program uses an algorithm to analyze the luminance (brightness) of the pixels in the image. When the algorithm detects a sharp increase of brightness, it can calculate the latency by subtracting the initial timestamp (A) of the window turning white, from timestamp (B) of the received video frame. This method is preferable to taking a manual snapshot, since the measurement is automated.

Figure 8



A variety of technology exists in the marketplace to help achieve lower latency. An example is the Sensoray Model 2253 compact USB-compatible audio/video codec, which has a low-latency preview mode, as well as the Model 2263, which supports a variety of analog and digital input formats and features a low latency uncompressed preview on a host and efficient H.264 video compression.