Roger Olsson, Håkan Andersson, and Mårten Sjöström, A modular cross-platform GPU-based approach for flexible 3D video playback. In *Proceedings of Stereoscopic Displays and Applications XXII*. Burlingame, CA, USA, January 2011.

http://dx.doi.org/10.1117/12.872244

# A modular cross-platform GPU-based approach for flexible 3D video playback

Roger Olsson*, Håkan Andersson, and Mårten Sjöström

Dep. of Information Technology and Media, Mid Sweden University, Sundsvall, Sweden

## ABSTRACT

Different compression formats for stereo and multiview based 3D video is being standardized and software players capable of decoding and presenting these formats onto different display types is a vital part in the commercialization and evolution of 3D video. However, the number of publicly available software video players capable of decoding and playing multiview 3D video is still quite limited. This paper describes the design and implementation of a GPU-based real-time 3D video playback solution, built on top of cross-platform, open source libraries for video decoding and hardware accelerated graphics. A software architecture is presented that efficiently process and presents high definition 3D video in real-time and in a flexible manner support both current 3D video formats and emerging standards. Moreover, a set of bottlenecks in the processing of 3D video content in a GPU-based real-time 3D video playback solution is identified and discussed.

**Keywords:** 3D video, player, GPU-based, multiview, autostereoscopic, modular, cross-platform, FFmpeg, OpenGL

## 1. INTRODUCTION

Different compression formats for stereo and multiview based 3D video have been, or are in the process of being standardized [1]. Software capable of decoding the different compression formats and present 3D video on different display types using standardized players is a vital part in the commercialization and evolution of 3D video. Players for stereoscopic 3D (S3D) video are becoming more common both as software applications as well as hardware implementations in 3D enabled Blu-ray players, media players, and set-top boxes. However, the number of available players capable of decoding and playing multiview or autostereoscopic 3D (A3D) video is still only a handful [2, 3, 4]. This leaves room for constructing a flexible cross-platform playback solution that easily can be configured and extended to support:

- a wide range of 3D display technologies
- 3D video formats and compression standards
- the requirements of different applications
- research and development on 3D playback algorithms and tools

The work presented in this paper describes the design and implementation of a GPU-based real-time 3D video playback solution, built on top of cross-platform, open source libraries for video demultiplexing, decoding, and hardware accelerated graphics. A software architecture is proposed that efficiently process and playback high definition A3D video in real-time, which flexibly support various video formats and compression standards. Moreover, a set of bottlenecks in the processing of 3D video content is identified and discussed. The paper is organized as follows. Section 2 gives a summary of the general building blocks required for 3D video playback and the specific hardware-accelerated framework that is required for real-time performance. Section 3 presents the design and development strategies used in the proposed software architecture as well as implementation details vital for the 3D video playback solution. The performance of the playback solution is measured using a set of quality metrics and test video sequences defined in Section 4. The results from the performance evaluation is presented in Section 5 and Section 6 concludes the work, and list ideas for future work.

(*) Corresponding author. E-mail: Roger.Olsson@miun.se, Telephone: + (46) 60 14 86 98

## 2. PLAYBACK OF A3D VIDEO

A3D displays relies on multiplexing to present a set of different views to the observer(s) simultaneously. This set of 2D views is combined into a single 3D image that is perceived as having vital depth cues such as binocular- (depth) and motion parallax (look-around). A3D displays relying on lenticular lenses or parallax barriers for differentiating views utilize spatial multiplexing to combine the set of views into a joint intertwined 2D image that is presented on the A3D display's pixel panel. The process of intertwining views relies on a strong geometrical relationship between the display panel's subpixels (red, green, and blue) and the lenses or slits for lenticular and parallax barrier respectively. The geometrical correspondence differ between displays as this, together with display panel and lens-/slit pitch, control important A3D display properties such as available depth range, view spatial resolution, number of views, and view zone angle. Properties that have different importance and weight depending on application and user requirements.

Significant research effort has been put into constructing generic 3D formats that represent the set of 2D views constructing a 3D image in a display independent way, e.g. video-plus-depth, layered-depth-video etc. [1]. Decoupling the 3D format from the 3D display is vital for interoperability between different capture- and display technologies as well as between different equipment manufacturers.However, this flexibility comes at the cost of an increased requirement of computational complexity both in the transmitter and receiver end of the distribution chain from camera to display. For example, novel views must be synthesized from the formats' geometrical models, which translates to additional decoder operations that must be performed in real-time. Fortunately current video cards, with their dedicated high performance graphical processing units (GPU), allow for these decoder operations to be addressed efficiently. The increased programmability of the GPUs also enables a customized rendering process that can be used in a playback solution to modularly support different formats and display types.

The factors that limits a 3D video playback solution implemented on a PC desktop computer are bandwidth and/or processing power. In the term processing power we include both processing speed of CPU and GPU. In the term bandwidth we include system bus speed, memory access time, peripheral interface bandwidth and hard disk drive (HDD) data transfer rate. HDD data transfer rate in the particular case of this work targets the time it takes to transfer data from the stored media container file to main memory. For modern computers the HDD data transfer rate is significantly lower than the system bus, PCI Express interface and volatile memory speeds hence making it the most likely bottleneck. This HDD data transfer rate is a function of both internal speed (physical properties, buffers and mechanics of the HDD) and external speed (I/O interface) and can be divided into disk to buffer data rate and buffer to memory data rate. For HDDs the disk to buffer data rate is usually slower than the buffer to memory data rate since the mechanics of the HDD is slower than transferring data from buffer memory to system memory. For solid state disks (SSD) the relationship is different and instead the external speed of the I/O interface may be the limiting factor. As an example the standardized 1080p video format (also known as "Full HD") utilize a spatial resolution of 1920x1080 for each video frame that at 25 frames/s and 24 bit color depth translates to a required throughput for playback of at least 155.52 MB/s. Hence, a playback solution targeting A3D display that relies on a Full HD pixel panel as a subcomponent must at some point be able to handle this throughput.

## 3. DESIGN AND DEVELOPMENT STRATEGIES

Most standard 2D video player software is built upon one or several demultiplexer, decoder and hardware accelerated graphic libraries and application programming interfaces (APIs). Striving for a highly portable solution with minimal effort our design also follows this pattern and integrates the open-source demultiplexer and decoding library, in this case libavformat and libavcodec, of FFmpeg [5]. For hardware accelerated graphics we utilize the cross-language and cross-platform 2D/3D graphics API OpenGL and audio support is handled by the cross-platform 3D audio API OpenAL [6, 7].

When designing our 3D video playback solution using the support of these libraries and APIs we considered two alternative designs:

1. Virtual demultiplexer and decoder
   Integrating 3D video processing logic *within* the FFmpeg library enabling 3D playback in 2D video players.
2. Standalone 3D video player
   Creating a standalone 3D video player application built *on top of* FFmpeg.

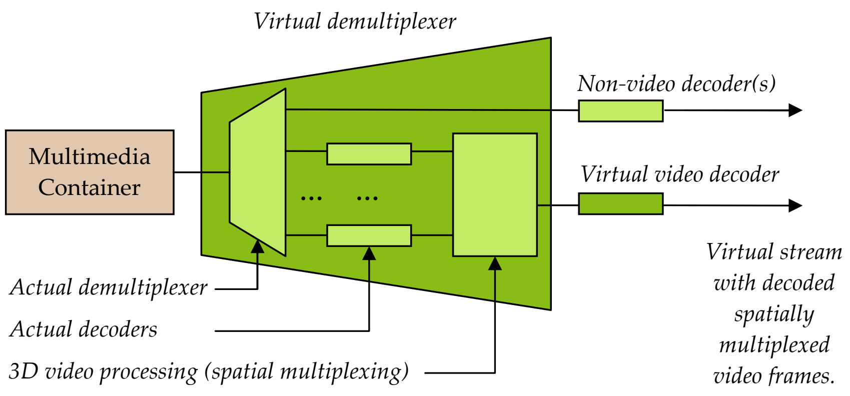

Figure 1. Overview of the components of the virtual demultiplexer

## 3.1 Virtual demultiplexer and decoder

Integrating 3D processing in the FFmpeg framework would enable 3D video playback in any standard 2D video player relying on FFmpeg for its demultiplexing and decoding. A virtual demultiplexer and decoder would then work as a wrapper around the actual media container demultiplexer of FFmpeg in a manner transparent to the video player. Figure 1 illustrates this relationship. This way the virtual demultiplexer manages both demultiplexing *and* decoding of 3D video streams. Logic for intertwining the views of the multi stream video is added as a final step in order to generate a single spatially multiplexed 3D video as output. Using this approach 2D video players interface directly to already decoded and spatially multiplexed 3D video (and non-video streams such as audio) without having to worry about the processing required for 3D video decoding. Although this seem to be a solid and attractive solution at first, a closer look reveals significant complexity and obscurity. Modern 2D video players such as e.g. VLC Video Player and others, relies on several different libraries for demultiplexing and decoding, and uses different combinations of these depending on the container type and encoding format [8]. In practice this prevents the virtual demultiplexer design to be adopted since all these different combinations must include the necessary 3D video processing operations. Even if this could be regarded as a solvable problem further obstacles exist for this design. These different libraries for demultiplexing and decoding does not manage image scaling nor has any information regarding the size of the video application window (scaling is typically done after the decoding stage). Knowledge about the final presentation resolution, the position of the video application window and its size, are strong prerequisites for the geometrical relationship mentioned previously. Unfortunately it does not help to define these to constants by adopting a fixed known video resolution, known display resolution, and full screen operation. Video data would still have to be processed within the virtual demultiplexer, decoded, processed within the GPU, and be sent back to the system RAM before being decoded again. Despite causing a significant loss of performance, this would also render the final solution inflexible and poor at handling different display technologies.

## 3.2 Standalone 3D video player

Creating a standalone 3D video player application is a more convenient and straightforward solution than the previously described virtual demultiplexer since it allows full control of demultiplexing, decoding and scaling as well as minimized overhead. However, this comes at the cost of requiring implementation of functionality such as stream synchronization, sound support and additional video player functions in order to create a working 3D video player. Despite this cost, the approach of designing a standalone player is the solution that was selected for the work presented in this paper. In the following subsections a number of important aspects of this approach will be presented.

## 3.3 Breakdown of modularity

To increase modularity and flexibility it is convenient to separate 3D video processing from video player functionality. Therefore the 3D video player application has been divided into two separate software components:

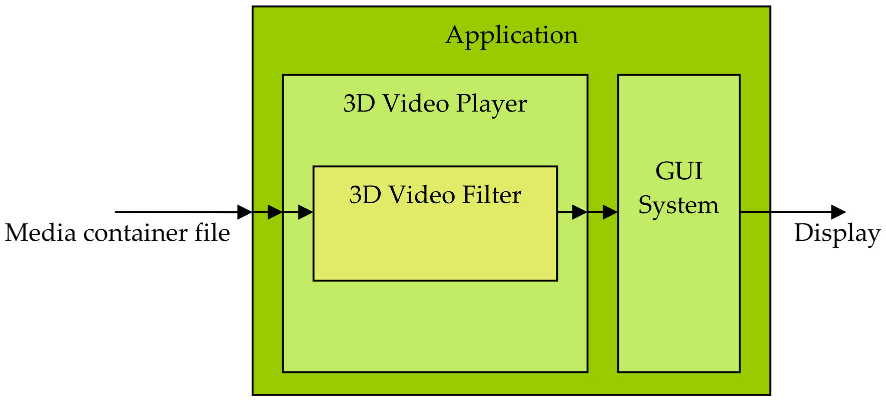

Figure 2. Conceptual composite structure of the 3D video player application

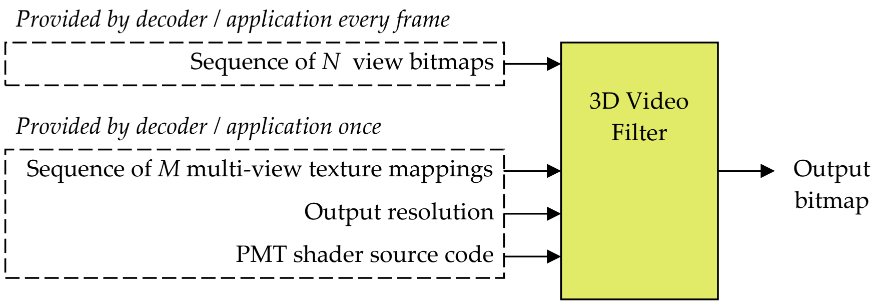
Provided by decoder / application every frame
Sequence of $N$ view bitmaps
Provided by decoder / application once
Sequence of $M$ multi-view texture mappings
Output resolution
PMT shader source code
3D Video Filter
Output bitmap

Figure 3. Overview of the 3D video filter input and output parameters

- Video player – containing high-level video player functionality such as media container demultiplexing, decoding, and synchronization.
- 3D video filter – containing 3D video processing logic in the context of spatial multiplexing, RAM to video memory pixel data copying, GPU processing in terms of shader management etc.

The video player has been designed to use the 3D video filter as an intermediate composite component in the video processing pipeline. This is conceptually illustrated in Figure 2. Separating the rendering of 3D video from general player functionality allows the 3D processing logic to also be extracted and used in other application contexts that require sending spatially multiplexed 3D video output. For example 3D video communication applications where a network sender and receiver is added to the distribution chain. In the following sections the 3D video filter will be further described.

## 3.4 3D video filter

The 3D video filter or processing component can abstractly be seen as a black box which takes as input a multi stream video signal and the most vital display properties, transforms the signal and produces an output video signal that is a valid spatially multiplexed 3D video signal for the particular display as illustrated in Figure 3. The 3D video filter is designed to be used as an output filter for a video or image decoder and constitutes the last processing stage in the video pipeline of the proposed 3D video player application. The input to the filter may vary from formats represented as tiled images as shown in Figure 4 to multiple video streams were each video stream corresponds to a disparate view. For consistency, the 3D video filter internally assembles multiple video streams into a tiled format as well as making this the common base from which further processing can be performed. The filter output however is fixed and limited to be a tiled image as that

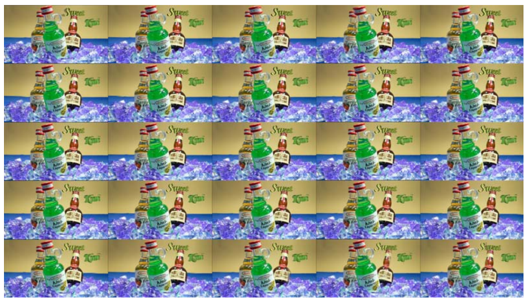

Figure 4. Example of a 5 x 5 tile, which represents 25 different views.

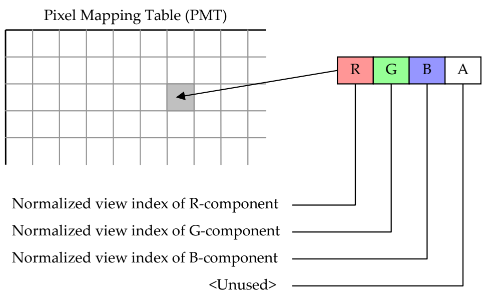
Pixel Mapping Table (PMT)
R
G
B
A
Normalized view index of R-component
Normalized view index of G-component
Normalized view index of B-component
<Unused>

Figure 5. View indices stored in the PMT for each subcomponent of the A3D display pixel panel.

of Figure 4. In addition to the pixel data of the multiple views a description of the tile structure is required in order to interpret and process the 3D video. For example, for a video with N disparate views, the filter must be fed with pixel data for all N views every frame together with N view mapping coordinates defining the location and size of each view in the tile. Except pixel data and format description data, it is also convenient to define the output resolution to the filter in order to enable scaling. This is needed if the native video resolution does not match the display resolution and resampling of the video for windowed or full screen mode is desired.

The function to multiplex N disparate views into a single spatially multiplexed texture is, as described, display dependent. However, it can be represented by a mathematical function which describes which pixel or pixel subcomponent (RGB) to map from a view texture to a certain pixel in the spatially multiplexed texture. The function will not change as long as the display is not interchanged, which allows for pre-calculation and caching of the results from such a function. The result can be represented as a pixel mapping table (PMT) which eliminates the need for re-computation and simplifies the mapping process to simple table lookups which is beneficial for performance. To further increase flexibility, the generation of a PMT is in this work accomplished by providing source code for a fragment shader program executing on the GPU, which calculates the PMT and stores it as a texture in video memory. This enables the shader program to be interchanged during run-time and compiled on the fly if the video format, display type or display resolution changes. Figure 5 illustrates how the mapping specified in the PMT is stored. The RGB-components of the pre-computed PMT specifies the index of the corresponding view to get the actual pixel's subcomponents from.

## 4. METHOD

The high system requirements of 3D video processing and the ongoing work of 3D video format standards requires flexible, yet efficient algorithms. In combination with requirements of cross-platform interoperability this affects both the design and implementation of 3D video playback software. To evaluate the proposed solution it is important to determine if it is efficient enough to handle throughput requirements. The following section describes methodology and measurements for evaluation of the proposed 3D video playback solution.

## 4.1 Performance criteria

The performance criteria used rely to different degree on accurate time measurements. This is accomplished by code injection of hardware timers in the playback system, which measures execution time with minimum overhead compared to that caused by e.g. profiling routines.

The throughput requirements are quantified in the number of presented frames per second ($f_{max}$), were the total processing time $T$ for the video is divided by the number of processed frames $N$. It might be tempting to measure the execution time of each frame in order to be able calculate the standard deviation of the processing time on a per-frame basis, but this is not practical since it would require time stamps to be saved into memory every frame implying greater overhead due to memory access time and dynamic memory allocation time.

By disabling video synchronization and display vertical synchronization (VSync) it is possible to measure the maximum throughput of the system in terms of average number of frames per second (FPS). The measured throughput should be compared to the native frame rate of the video in order to be evaluated. A throughput or update frequency of 27 frames/s for a video encoded in 30 frames/s is considered as inferior, while a throughput of 27 frames/s for a video encoded in 25 frames/s is considered as acceptable. The difference between the maximum average frame rate $f_{max}$ and the video's native frame rate $f_{native}$ is defined as

$$\Delta f = f_{max} - f_{native}. \tag{1}$$

Hence, if $\Delta f < 0$ , the system is not able to playback the video at the sustained frame rate. On the other hand if $\Delta f \geq 0$ , the system fulfills the performance requirements. The performance is highly dependent of both software implementation and the capabilities of the hardware, but the experiments within this work is limited to be performed on one machine only as further described.

To evaluate how the 3D video player solution is related to different limiting factors for 3D video playback, it is necessary to observe the average frame rate as a function of bitrate, resolution and number of views. In addition, measuring the load balance between CPU and GPU processing allows insight into how the playback solution can be further enhanced.

## 4.2 System properties

The system used for implementations and experimental measurements is a Dell PC with the following technical specification: Intel Xeon E5520 2.25 GHz CPU, 2.50 GB RAM, NVIDIA Quadro FX 580 graphics card and a Seagate Barracuda 7200.12 SATA 3Gb/s 250GB hard drive with a 125 MB/s sustained data rate. This system is considered as a standard high-end system to the current date and is likely to produce experimental results that are similar to the results of any high-end desktop computer today.

## 4.3 Test video sequences

A number of different 3D video sequences were used to find anomalies in performance dependent of compression format, resolution and number of views. An overview of the sequences is found in Table 1. A set of 12 standard and multiview video files were used and, gathered from freely available sources on the web as well as from marketing material supplied by A3D display manufacturers. Different codecs are used in the evaluated video sequences including MPEG-4 video, Windows Media Video 9, SMPTE VC1, and MPEG-2 video.

## 5. RESULTS

The experimental methodology and evaluation methods described in the previous section defines a number of experiments and heuristics for evaluation of the 3D video playback solution. A summary of the results obtained from these experiments is presented here.

The difference between average frame rate and native frame rate, $\Delta f$, as defined in Equation (1), is shown in Figure 6. Note that out of the evaluated video sequences only on fails to be decoded at its native frame rate. This video sequence is top-bottom Full HD S3D video (1920 x 2160 spatial resolution) coded with a variable bitrate of 30 Mbit/s and peaking at 60 Mbit/s using the SMPTE VC1 codec. Since this sequence exhibits the highest bitrate, as well as the highest resolution, it cannot be determined if the failure in decoding it is due to bandwidth or processing limitations.

The graph illustrated in Figure 7 illustrates the average measured frame rate for varying video bit rate for the 12 video files. Multi stream formats and tiled formats are represented by two different graphical symbols for clarity, squares and

Table 1. Range of values for the evaluated video sequences' vital properties

| Property | Value Range |
|---|---|
| Average compressed bitrate [Mbit/s] | [4, 30] |
| Peak compressed bitrate [Mbit/s] | [6, 60] |
| Number of views | [1, 8] |
| Number of streams | [1, 8] |
| Frame resolution [pixels] | [640x480, 1920x2160] |
| Frame rate [frames/s] | [24, 30] |
| Pixel rate [Mpixel/s] | [20.7, 124.4] |
| Raw bitrate [MByte/s] | [62.2, 373.3] |
| Duration [min:sec] | [0:08, 2:18] |

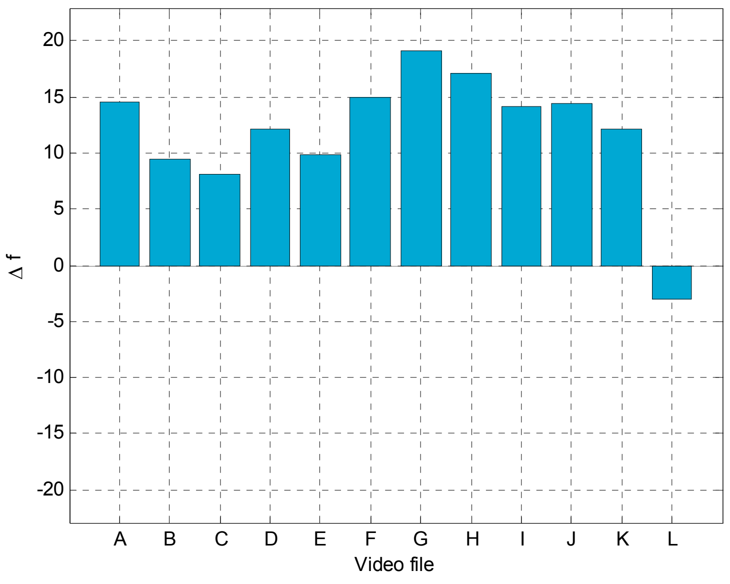

Figure 6. Difference between average and native frame rate.

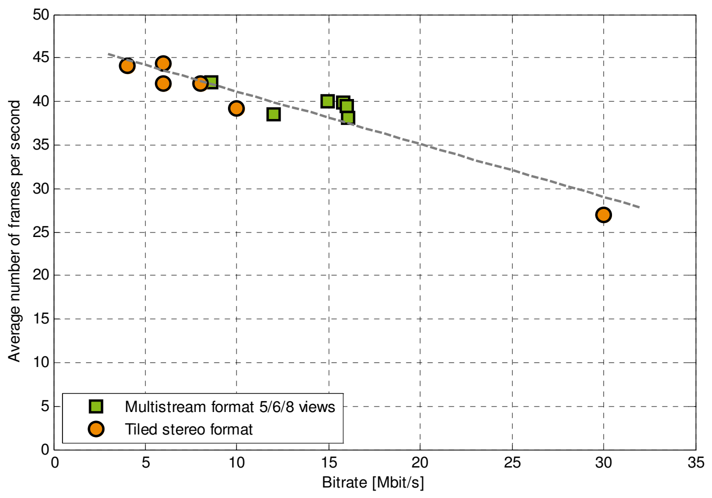

Figure 7. Evaluated throughput as a function of bitrate

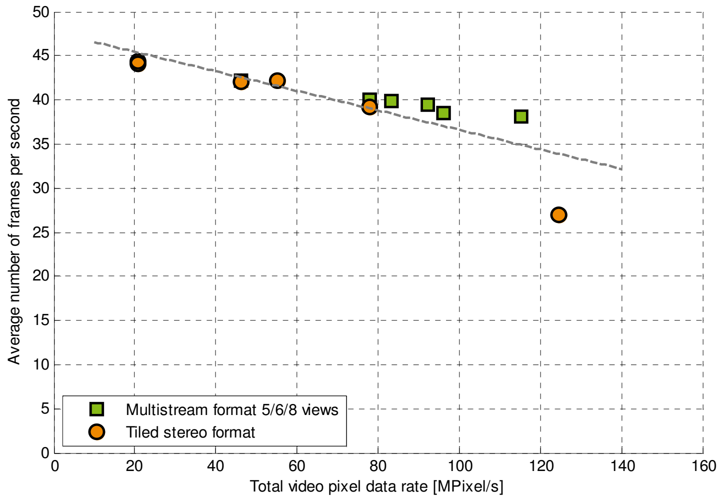

Figure 8. Evaluated throughput as a function of resolution

circles respectively. Figure 7 is complemented with a linear regression line plot in order to visualize the frame rate trend as bit rate increases. Throughput is approximately linearly dependent of the video bit rate. This is also the case for throughput as a function of resolution as can be seen in Figure 8, which illustrates the average measured frame rate for varying video resolution (pixel data rate) for the video files.

Figure 9 illustrates the average measured frame rate for varying number of views N, interpreting one of the video files as multiview video while rendering to a lenticular display. As can be seen, a second order best-fit polynomial approximation indicates a trend as the number of views N increases. This may be explained by the fact that GPU texture lookup is used to perform the spatial multiplexing. During this operation a texture cache is used on the GPU that is optimized for locality in two dimensions that thereby accelerates the lookup by pre-storing neighbouring texels (texture pixels) for the next lookup. The combination of the GPU cache format and a PMT adopted for a lenticular display results in an increased number of cache misses as the number of views increases. This as a result of how disparate view RGB sub-components are aligned in a lenticular display. A single view will cause high cache utilization where as a high number of views will cause a practically random GPU memory access pattern. Since the GPU cache is a square matrix, degradation is exponential that is

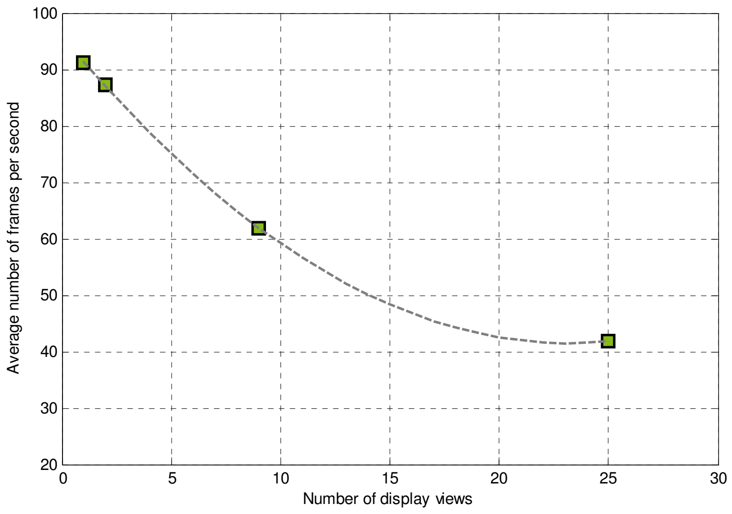

Figure 9. Evaluated throughput as a function of number of views

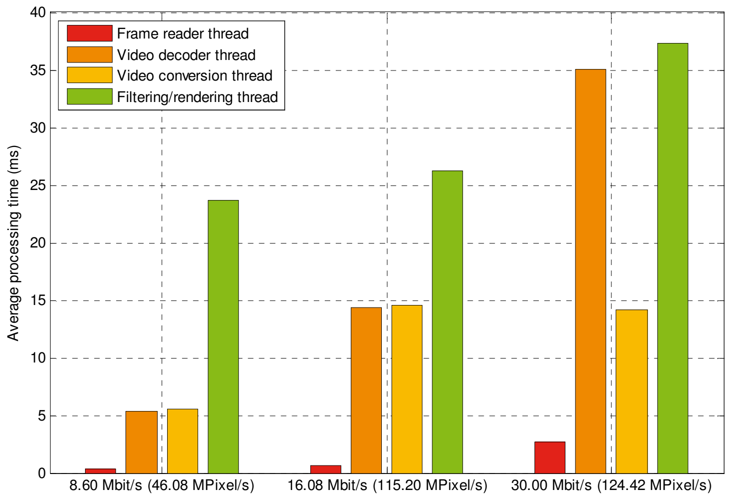

Figure 10. Load Balance

also described by the second order best-fit approximation. The minimum or floor throughput is given by the video memory access time for the particular video card being used.

The 3D video playback solution proposed relies on four different threads implementing the producer/consumer design pattern. Hence, the work load balance between these threads will directly affect performance. Figure 10 shows the average processing time per video frame and thread for three of the evaluated video sequences. Rendering and system bus seems to be limiting factors as video processing and rendering on the GPU has the highest average processing time per frame. Video decoding and pixel conversion threads requires approximately the same amount of processing time for relatively low bitrate and low resolution video, but as resolution and bitrate increases so does decoding processing time seem to increase and approaches the processing time needed to render video.

## 6. CONCLUSIONS

The proposed solution has proven that it is possible to create a working 3D video player built on top of cross-platform, open-source libraries, framework, and APIs while still providing high throughput and configuration flexibility. Future work includes incorporating view synthesis to provide a required number of views set by a A3D display that is higher than the available number of views given in a specific 3D video format, and the support of geometrically assisted video formats such as multiple video plus depth (MVD).

## ACKNOWLEDGEMENT

This work is supported by the EU Objective 2 - project Digital 3D Signage and the KK Foundation project 3D Video.

## References

[1] Smolic, A., Mueller, K., Merkle, P., Kauff, P., and Wiegand, T., “An overview of available and emerging 3d video formats and depth enhanced stereo as efficient generic solution,” in [*Picture Coding Symposium 2009*], 1 – 4 (May 2009).

[2] “Vim 3d movie center.” 3D International (December 2010).

[3] “Svi power player.” Spatial View Inc. (December 2010).

[4] “Stereoscopic player.” 3dtv.at (December 2010).

[5] “FFmpeg Multimedia System.” ffmpeg.org (December 2010).

[6] “Opengl.” opengl.org (December 2010).

[7] “Openal.” connect.creativelabs.com/openal (December 2010).

[8] “Vlc video player.” videolan.org (December 2010).