Multi-Threaded Real-Time Video Grabber

Zdeněk Trávníček
DCGI, FEE
Czech Technical University in Prague
Czech Republic
zdenek.travnicek@fel.cvut.cz

Roman Berka
Institute of Intermedia, FEE
Czech Technical University in Prague
Czech Republic
berka@iim.cz

ABSTRACT

Communication in general incorporates technologies with increasing number of communication modes. Special applications are developed in the area of virtual reality, multimedia communications and others where combinations of audio, video, 3D data are sent between two (or more) distant users which can commonly interact with these data. A form of so exchanged information usually requires, among others, special forms of presentation. Thus stereoscopic and virtual reality visualization devices are used to present intricately structured information in multi-modal form.
There are situations where the presented information is to be rendered in real-time and transmitted to the remote user in form of a video-stream. In this case, the content is presented on a local visualization device (e.g. CAVE) being simultaneously sent to a remote device. Thus a method how to obtain rendered data from graphics hardware in real-time is necessary.
The problem is, how to obtain the rendered data for transmission with minimal impact on the rendering and visualization process. In this paper, we present a method how to retrieve video stream from an arbitrary running OpenGL application, capturing every frame with minimal impact on performance.

Keywords: OpenGL, real-time video grabber, streaming video, streamcast

1 INTRODUCTION

With the rise of 3D digital media, stereoscopic movies and upcoming 3D television, the need for a new sources of stereoscopic signal emerges. The usual sources of such a signal are cameras in stereoscopic setups or pre-rendered video sequences. There are many applications rendering 3D images, some of them even stereoscopic ones. Those could be great source for such a stream, but they usually does not support producing an video that could be directly used as a source of video signal for stream nor support saving video to a file.

In order to use such an application we need to be able to retrieve output of the running application in real-time (see fig. 1). From other point of view, we may simply want to record output of running application and store it locally for later, offline use. In order to get those, we could alter the application itself to produce such a video stream or file. We can also use some screen grabbing application (streamcast) or have a hardware solution.

As the graphics hardware and software technologies changes over the time, the problem is still actual and new approaches appear. The main problem is related to the cost of the grabbing process because the data source (typically a graphical subsystem) produces content in real time. Thus the grabber should obtain pictures with minimal impact on the rendering process.

We first describe known methods of the video grabbing which appeared during a period of the last decade. These methods are evaluated according to our criteria based on modification that needs to be done to the application itself, impact on performance of the application and possibility of grabbing stereoscopic images from quad buffer. We evaluate the performance loss for multi-core/multi-CPU systems. Next, our own asynchronous wrapper is described and compared with the already implemented solutions. Finally, some applications of the described wrapper are presented.

2 STATE OF THE ART

There exist several approaches to the solution of how to acquire a stream of graphical data from an application running on the system. These approaches are then often implemented for various purposes. We can classify them into four basic groups:

- alteration of the application which is the source of the data
- screen grabbing
- combination of previous two methods
- capturing output of the graphics hardware

These methods are know explained and compared in the next paragraphs.

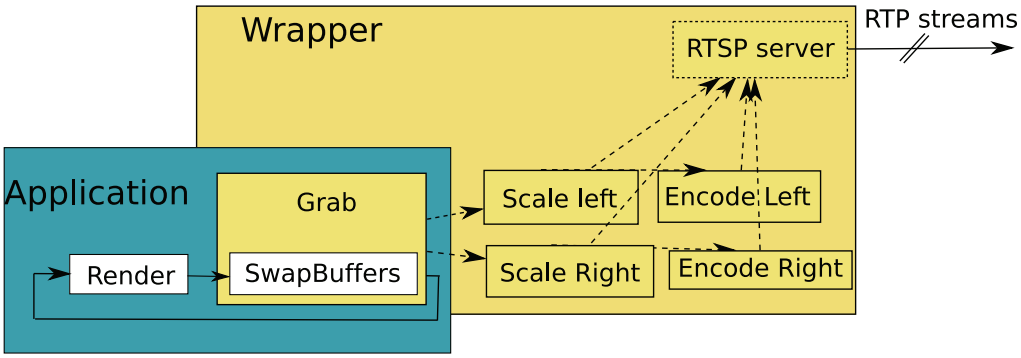

Figure 1: A general scheme of grabber.

2.1 Altering an Application

The method of altering an existing application has an obvious drawback in a need to have source codes for the application and also alteration of every application we use. This basically limits the usability of it to applications where we have source code (typically open-source). The solution is also complicated when we use many different applications.

Aside from that, this method has an advantage in knowing everything about the application to have full control over the grabbing process. Thus it can grab the images synchronously with the rendering speed. Also, for an application rendering stereoscopic images into quad-buffer, this method can grab images for both eyes. The implementation is specific to every application as well as the performance loss. This solution can be therefore seen in special applications (e.g., applications working as real-time video content generators for network projects or art performances).

2.2 Screen Grabber

The screen grabbing represent a next approach where the graphical information is obtained independently on the application code. Using a standalone screen grabber does not require any alteration of the application, but on many systems it has problems on accelerated windows. It won't be synchronized with the speed of an application as it does not have any information about architecture of the application. Asynchronous grabbing can introduce image distortions when the frame buffer is changed during read, it can miss frames when the application renders faster than the grabber grabs and can unnecessarily grab the same image multiple times, when the application stalls or is just slower than the grabber. Furthermore, this method would fail for quad-buffer stereo.

As an example of such an approach, there are applications like *scrot* and *xsnap* realized in GNU/Linux environment. The code of the grabber runs outside the context of the application, so the impact on the rendering speed should be quite small.

2.3 Combined Solution

Another solution would be combination of above mentioned two methods. Here, a separate grabber without modifying the application is used. This can be done using an wrapper to rendering library, i.e. OpenGL, which would inject some code to proper place of the rendering process and execute it there. Provided our code could get enough information about rendering window, we can grab the exact window, adjust the area being grabbed when the application window changes and we can start the grabbing exactly once per frame.

There is an opensource project *captury* using this solution. In this project, the code is executed in context of rendering thread of the application, effectively slowing down the rendering of every frame by grabbing, compressing and saving every frame, before it the buffers gets swapped.

2.4 Hardware Solution

A hardware solution means plugging some device into output of graphics card and process it on other computer or in the device itself. This solution needs separate hardware, it is quite expensive, and is not synchronized with the application's speed. The output signal needs to be cropped when rendering only into an window. In addition, the captured signal has given parameters, like resolution, which are not easily controllable during the grabbing process. On the other hand, it has absolutely no impact on the application itself, as there's no processing on the rendering machine.

As the acquisition of the video from graphics hardware in real-time is an interesting problem new solutions implemented directly in the graphics boards rises. In August 2009, nVIDIA released solution to record/output SDI uncompressed video directly to/from Quadro GPU's memory. As this information is too

much new, we had no chance to test it before submission of this paper.

3 MULTI-THREADED REAL-TIME VIDEO GRABBER

The solution we propose is a modifie approach to wrapping rendering library's calls and injecting our code there.

The key is in using a wrapper, that "hooks" onto few library calls in order to retrieve information about application's window and to grab the window in a right time.

The grabbing itself is done in the context of the rendering thread using standard methods to retrieve the content of framebuffer. This directly implies that, when rendering in quad-buffer mode for active stereoscopy, we can easily get both images as we can control the flow of the code. After getting the frame we send it to an other thread to next process. This ensures that the impact will be as small as possible, provided the machine has multi-core CPU or multiple CPUs. The processing itself can be done in multiple threads also, to use more available cores more effectively. In the processing threads, we can save the video to the local storage or stream it over network and optionally compress it.

The implementation we present was done under GNU/Linux environment, using an OpenGL applications and nVIDIA QUADRO FX cards to render active stereoscopic images in quad-buffered mode.

3.1 Wrapping

The wrapping is done by utilizing linux dynamic loader, which takes care of loading libraries and resolving symbols. Using `LD_PRELOAD` environmental variable recognized by the loader, we tell it to preload a shared object before an application and use it for symbol resolving with higher priority. In the shared object we provide hooks on few function that inject our code before the real call to the library function.

Namely we "hook" onto `glViewport` in order to get information about the window size and it's changes. We also use this as a point to initialize the processing threads. We also hook onto framework specific functions in order to swap buffers (`glxSwapBuffers`, `SDL_GL_SwapBuffers`). When the application calls swap buffers, it signalizes it has finished rendering the frame, so it's the right place for us to grab it and send it to the next process. It is also the place where we can drop frames if the application is rendering too fast. Our implementation also wraps `dlsym` call to catch symbol resolving done in real-time and not by dynamic loader.

3.2 Grabbing

During a rendering process the rendered images are stored in two (or four in case of stereoscopic output) frame buffers which are periodically swapped. On principle, there are two types of frame buffer reading:

- asynchronous – based on so called *Pixel Buffer Objects* [Biermann et al., 2004]
- synchronous – direct buffer reading

First, retrieving the image is the done by calling `glReadPixel` with correctly set read buffer in OpenGL context, optionally on initialized Pixel Buffer Object (PBO). PBO approach moves the reading into background so it does not block the rendering thread. But it introduces a delay of 1 frame, because we get the data on the next buffer swap.

The direct approach introduces delay into the rendering thread, which means a slowdown of the application, but we get the data sooner. We support both methods. By changing actual buffer and repeating the read, we can retrieve data for the other eye, if we have quad-buffer stereo.

3.3 Processing

The processing threads are doing color space conversions and re-sampling. Other threads can take care of possible video compression and others can stream it or save it locally. Processing of stereoscopic signals is done by pairs of threads to improve multi-threaded performance.

3.4 Summary

A scheme of the process is shown on figure 2. Original application is wrapped in it's call to Swap Buffers (usually `glxSwapBuffers`) is intercepted and instead of it, our code is executed. Content of the framebuffer is then grabbed as described in 3.2 and sent for processing to other threads. Then original `SwapBuffers` method is called and control is returned to the application. Meanwhile the data from framebuffer are being processed in other threads and eventually streamed out (or recorded).

The whole grabbing process is done in the context of the rendering thread, but the rest of the processing is done in other threads, not directly affecting the application's performance. So the impact to application is mostly defined by the slowdown that takes place in the grabbing functions. Of course, in case the application would do some CPU intensive operation the video (i.e. compression), it may place load to the CPU and indirectly slowing down the application.

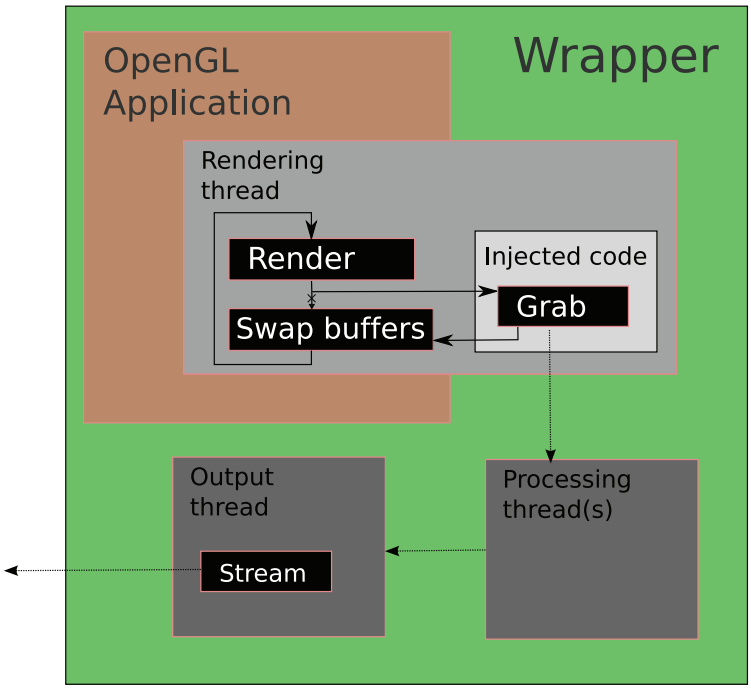

Figure 2: Scheme of the wrapped grabber

4 APPLICATIONS

The possibility to capture rendered video in real-time has lot of applications in wide area. As the problem described in this paper is part of another project, we can mention some applications which already use our grabber.

4.1 Project C2C

Described method is successfully used in project Cave2Cave (C2C) [Berka et al., 2009] to stream a stereoscopic video signal from applications running in CAVE-like system [Cruz-Neira et al., 1992] and to present it on remote site (see fig 3).

We use the the multi-threaded grabber to get video of the application, scale it, optionally compress it and stream it using standard protocol RTP [Schulzrinne et al., 1996]. The grabber also creates RTSP [Schulzrinne et al., 1998] server to provide SDP descriptions [Arkko et al., 2006] of the streams. This way we can (and we do) present applications from our CAVE system to distant viewers. The use of standard streaming protocols allows us to partially preserve possibility of receiving data by standard players used by remote user.

4.2 Prerendering

Another use of the method is to allow prerendering with applications that does not support it natively. For example, application rendering complex model which can not be rendered in real-time could be used to render it as fast as it could while having it's whole run recorded. Then we simply playback the recorded video at the requested speed. This allows us to present output of any application even in cases, when the application itself can not do it in real-time. We successfully used this method for presenting walks through very complex VRML models to public.

4.3 Industrial Applications

As the grabber can wrap theoretically any OpenGL application (it depend on correctness of application implementation in relation to OpenGL library), it offers itself in such situations where some industrial product (like an architectural model or model of a car) is to be, probably interactively, presented to a remote user without necessity to send these data to his/her computer. It is important when there is not possible to move real data or software, e.g. due to license limitations. Using systems like CAVE, running our grabber on each wall, as a source of content, an application then allows to mediate immersive environment remotely using standards described in already referenced RFC documents.

5 CONCLUSION

The proposed method allows real-time retrieval of rendered stereoscopic images from arbitrary OpenGL application without a need to modify the application itself. It can be used as base for a system to record an output of an application to local storage for offline use or to stream the content over network in real-time.

The solution has potentially lot of applications in wide area of remote visualizations also on immersive devices or in the area of collaborative environments. As it has been already mentioned above the problem

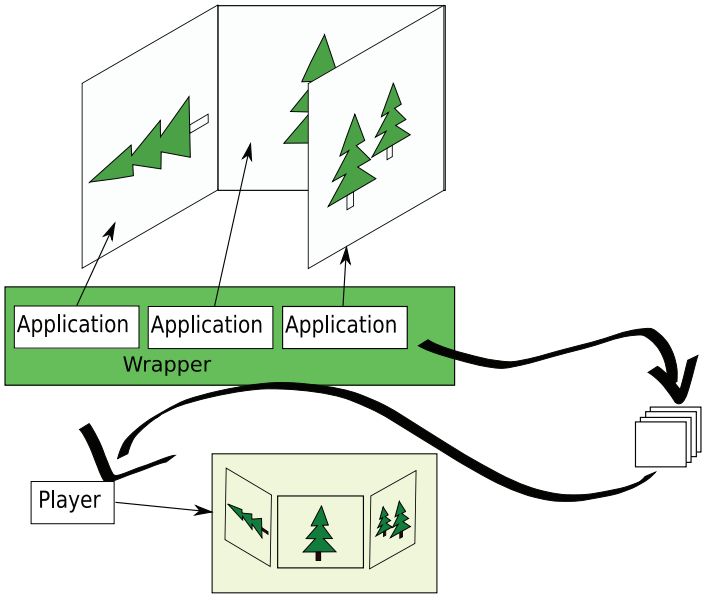

Figure 3: Scheme of the multi-projection screen based configuration. A scene rendered in the resource device with 3 projection walls is grabbed and the resulting video is transmitted to the remote device where it is presented on remote projection wall.

with grabbing methods is in continuous development and follows possibilities of contemporary technologies. For know, we can expect that the support of hardware solutions will be probably accessible for wider area of applications.

6 ACKNOWLEDGMENTS

This work has been partially supported by:

CESNET, association of legal entities,
Prague, Czech Republic
under the research program MSM 6383917201

Czech Technical University in Prague
Institute of Intermedia

Center for Computer Graphics
under the research program LC-06008

REFERENCES

[Arkko et al., 2006] Arkko, J., Lindholm, F., Naslund, M., Norrman, K., and Carrara, E. (2006). Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP). RFC 4567 (Proposed Standard).

[Berka et al., 2009] Berka, R., Trávníček, Z., Havran, V., Bittner, J., Žára, J., Slavík, P., and Navrátil, J. (2009). *Networking studies III, Selected Technical Reports*, chapter CAVE to CAVE: Communication in a Distributed Virtual Environment, pages 161–174. CESNET, 1ts edition. ISBN: 978-80-904173-4-2.

[Biermann et al., 2004] Biermann, R., Carter, N., Cornish, D., Craighead, M., Kilgard, M., Kirkland, D., Leech, J., Paul, B., Roell, T., Romanick, I., and Sandmel, J. (2004). ARB_pixel_buffer_object specificatio
http://www.opengl.org/registry/specs/arb/pixel_buffer_object.txt. Web page. Downloaded in October 2009.

[Cruz-Neira et al., 1992] Cruz-Neira, C., Sandin, D., Defanti, T., Kenyon, R., and Hart, J. (1992). The cave: Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, 35(6):65–72.

[Schulzrinne et al., 1996] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. (1996). RTP: A Transport Protocol for Real-Time Applications. RFC 1889 (Proposed Standard). Obsoleted by RFC 3550.

[Schulzrinne et al., 1998] Schulzrinne, H., Rao, A., and Lanphier, R. (1998). Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard).