CESNET Technical Report 2/2010

Implementing Video-Based, Remotely Accessible Virtual Environment System

Roman Berka[1], Zdeněk Trávníček[1], Vlastimil Havran[2], Jiří Bittner[2], Jiří Žára[2], Pavel Slavík[2], Peter Borovský[3], Jiří Navrátil[4]

[1] *Institute of Intermedia, Faculty of Electrical Engineering, Czech Technical University in Prague*
[2] *Department of Computer Graphics and Interaction, Faculty of Electrical Engineering, Czech Technical University in Prague*
[3] *Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava*
[4] *CESNET, z. s. p. o.*

Received 04.03.2010

Abstract

Communication in general incorporates technologies with increasing number of communication modes (visual mode, audio mode, gestures etc.). 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. We describe implementation details of important components of whole communication chain containing video-grabber and special multichannel player with user-interactive interface in this report. We first show whole chain, then the video-grabber is described, the multichannel player is presented as next and finally we present ideas concerning remote interactions of user with a virtual world. The last part of this report is devoted to applications and future development activities.

Keywords: OpenGL, real-time video grabber, streaming video, streamcast, distributed virtual environments, interactions, collaborative environments, immersion.

1 Introduction

Virtual Collaborative Environments (VCE) and their development is still actual problem. It concerns increasing accessibility of technologies and growing level of HCI (Human Computer Interaction). It all makes possible development of applications where new methods of human-computer communication are applied [1].

Development of applications in the area of VCE makes possible to connect users which are located at distant places and to work together as in case when being both in one room. This idea is usually applied in the area of visualizations where two or more teams interact with a model (e.g. model of a building), all teams are situated in different locations but all of them work with the same shared data.

The core of such system is a basic chain consisting of server application which plays role of data source for other – client applications. The client has to collect

all types of data and present them in suitable form to the remote users. The users can also interact with the remote scene/data and thus the client application must be able to transport information about user's interactions to the server to update its state.

Usually, this scheme supposes that each team is equipped with the same (or similar) device for visualization of the given data and all these devices run under an unique software. Transmitting the visual information in the form of standard video-stream we can get possibility to deliver audiovisual output from an interactive application running in the VR (*Virtual Reality*) system to the user equipped just only with a standard PC or notebook nor a special VR device.

In cases when the presented information is to be rendered in real-time and transmitted to the remote user in form of a video-stream, 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 the first part of the report, we present a method how to retrieve video stream from an arbitrary running OpenGL application, capturing every frame with minimal impact on performance.

The transmitted audiovisual information presented to the remote user is in general rendered on device which has different technical parameters and capabilities in comparison with a device for which the data stream has been originally prepared. Thus any kind of content adaptation is necessary to prepare it for presentation on remote device. The problem concerning the methods of content adaptation is standalone subject for research but in order to complete whole basic communication chain we have implemented a special player which can receive the stream of data and present it to the remote user according to capabilities of its device. The player represents a special solution but there should be a more complex methodology making possible to retrieve just only simple part of the content (e.g. single video channel instead of all available content) which can be processed using common application (like VLC player as video stream receiver). The second part of the report is devoted to the player and to the interactions which also represent one part of the transmitted content between client and server.

The presented approach has potentially wide range of applications. We tried to apply our two-point communication chain in two basic classes of applications. First, the remote presentation of inaccessible or remote spaces makes possible to deliver any degree of immersion to group of remote people which can interact with the presented scene. Second, the visualization of 3D (e.g. industrial) models is applicable in cases where it is not allowed to give the presented data to the remote user but it is necessary to present these data him/her interactively. Our approach makes possible to solve this problem. The third part of the report describes applications of our approach.

1.1 Video Acquisition

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 real-time rendering applications, some of them even stereoscopic ones. Those could be great source for such a stream, but they usually do not support producing a video that could be directly used as a source of video signal for stream nor support saving video to a file.

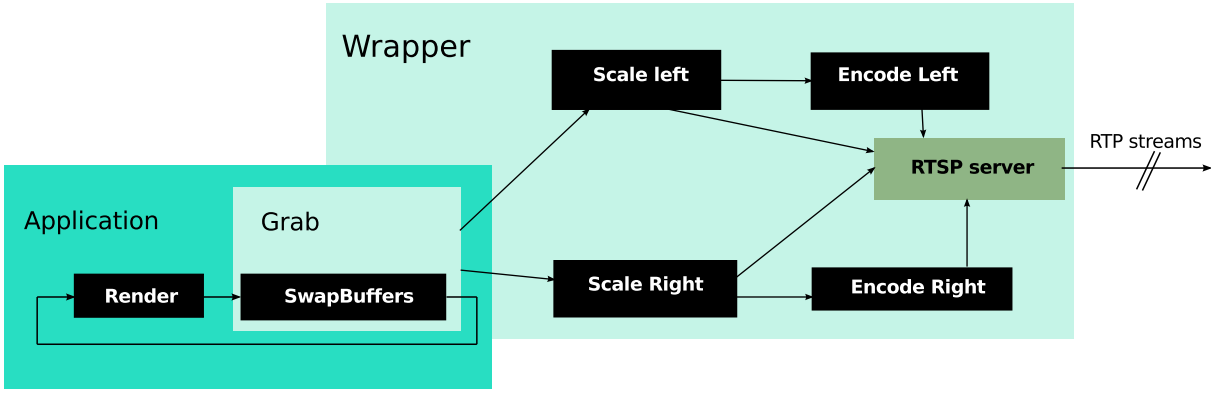

Figure 1. A general scheme of grabber.

In order to use such an application we need to be able to retrieve output of the running application in real-time (see Figure 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 modify 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 change 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. Our goal is that the grabber obtains 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 modifications that need to be incorporated to the application, 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.

1.2 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 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 explained and compared in the next paragraphs.

1.2.1 Modifying an Application

The method of modifying an existing application has an obvious drawback in a need to have source codes for the application and also modifying every application we use. This basically limits the usability of it to applications where we have source code (typically opensource). 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 quadbuffer, 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 seen in special applications (e.g., applications working as real-time video content generators for network projects or art performances).

1.2.2 Screen Grabber

The screen grabbing represents a next approach where the graphical information is obtained independently on the application code. Using a standalone screen grabber does not require any modification of the application, but on many systems it has problems on accelerated windows. It is not 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[1] and xsnap[2] realized in the 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.

1.2.3 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 a 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,

[1] `http://freshmeat.net/projects/scrot/`

[2] `ftp://ftp.ac-grenoble.fr/ge/xutils/`

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[3] 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.

1.2.4 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 a 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 fiinalizing the text.

1.3 Communication Chain Architecture

Various configuration of communication chain have been described in [3]. The common characteristics of the chain is in generalized form presented by scheme in Figure 2. The scheme represents the actual state of implementation consisting of components which will be described in next parts of this report in more detail.

The main subjects of interest are black boxes representing already mentioned *video-grabbing component* which is in our case implemented as software video grabber. The second is the *multichannel video player* which can receive all available data channels (up to this time multiple channel video stream and interaction command stream). Both of these components are now described in the next sections.

2 Multi-Threaded Real-Time Video Grabber

The solution we propose is a modified 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 quadbuffer 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,

[3] `http://gitorious.org/captury`

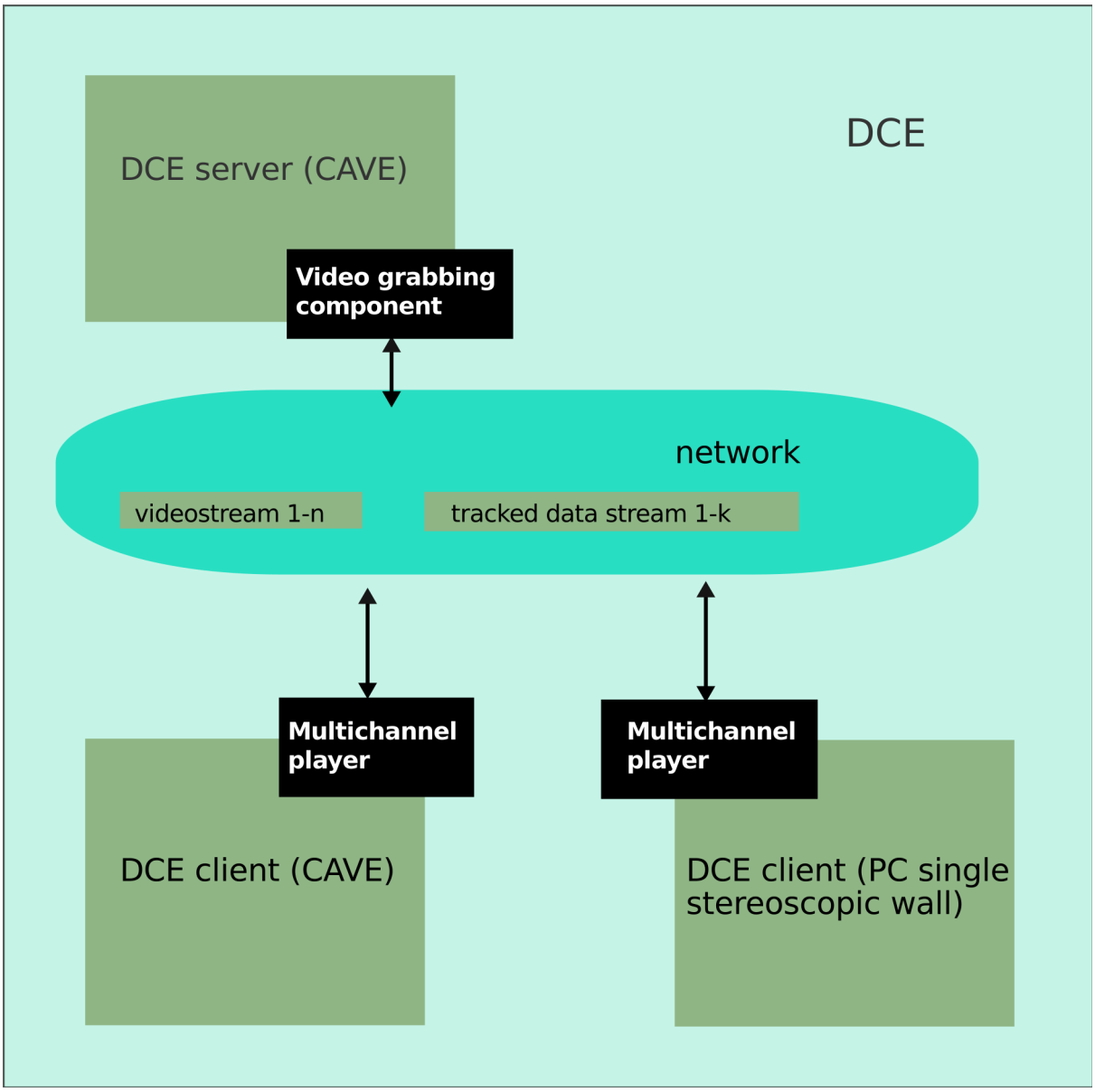

Figure 2. The general scheme of the communication chain inside architecture of experimental *Distributed Collaborative Environment* (DCE). More client devices present content (consisting of more data types - video, 3D data, tracked data) served by remote source device.

provided the machine has multi-core CPU or multiple CPUs. The processing itself can also be done in multiple threads 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.

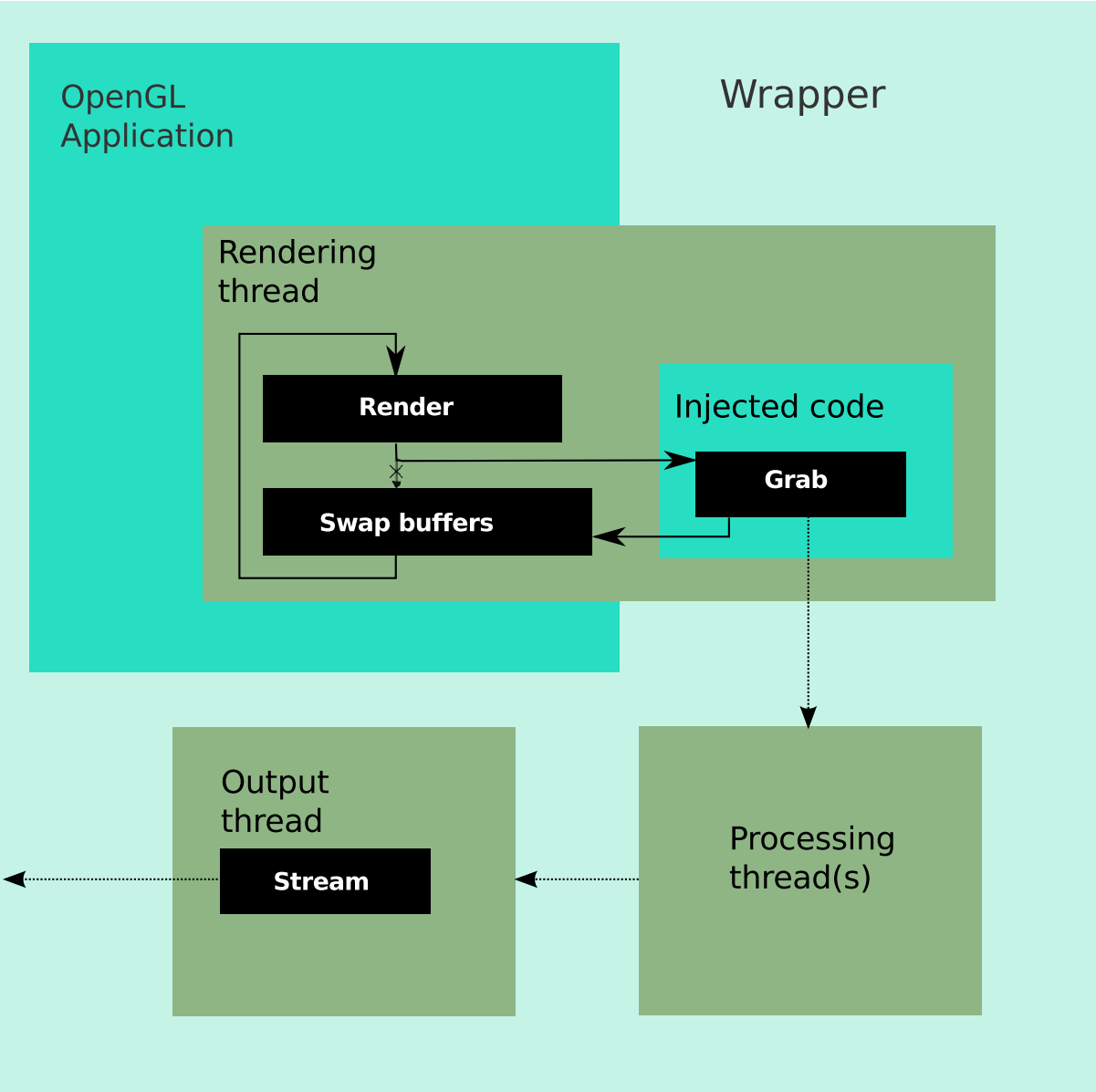

Figure 3. Scheme of the wrapped grabber.

2.1 Wrapping

The wrapping utilizes 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 its 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 a frame, so it is the right place for us to grab the frame 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 realtime and not by dynamic loader.

2.2 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[4],
— synchronous – direct buffer reading.

First, retrieving the image is implemented 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 quadbuffer stereo.

2.3 Processing

The processing threads perform color space conversions and re-sampling. Other threads take care of possible video compression and others stream it or save it locally. The performance of processing stereoscopic signals is increased by using pairs of threads.

2.4 Summary

A scheme of the process is shown in Figure 3. Original application is wrapped and its 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 Section 2.2 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 processed in other threads and eventually streamed out (or recorded).

The whole grabbing process is carried out in the context of the rendering thread, but the rest of the processing is 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.

[4] `http://www.opengl.org/registry/specs/ARB/pixel_buffer_object.txt`

3 Multichannel Interactive Player

Our grabber is based on widely used standard protocol RTP [8], so virtually any common player can act as passive receiver (provided it can decode the codec than we use in that particular case). However to simultaneously display more than one stream and to be able to reproduce stereoscopic effect, we need specialized player. Moreover to use the remote control capabilities of our solution, the special player is also needed.

The requirements for such a player are:

— ability to receive many streams simultaneously,
— configurable stereoscopic projection of stream pairs,
— mapping of the streams into virtual scene (i.e. onto CAVE like scene),
— support for local movement in the virtual scene,
— support to control the remote device,
— support for uncompressed video,
— as low latency as possible.

In this section we present our proof-of-concept solution that meet all of the above requirements.

3.1 Architecture overview

The player is based on library *libyuri* version 0.3. This library, which is developed simultaneously with the C2C project in IIM (Institute of Intermedia), was also greatly expanded during development of this application. *Libyuri* gives us a basis for multi-threaded application for mainly video processing, interface for configuration files and some prepared objects for video processing.

The player uses many distinct threads for receiving data, rendering and for user interaction. Based on the configuration file, the player open windows on specified displays and render appropriate parts of the scene into them. Each window is rendered in separate thread. The RTP reception is done separately in one or more threads, the received streams are decoded (one decoder thread per stream) and sent to main thread. When any of the rendering threads requests data for new frame, the input queues are traversed and only the latest images are returned.

User input is handled independently in input thread (spawned only when requested by configuration and when the specified device is available) and the state is made available to every renderer upon request.

In the basic CAVE mode, the connected control device moves avatar in the local virtual scene, unless specified button is pressed, which forces the player to records events for remote control (only when requested by configuration). The current implementation simply outputs the events to a file. More details follow in Section 4.

3.2 Features

Current implementation can handle RTP streams with uncompressed video packed as defined in RFC4175[5] (support is limited to RGB/RGBA color spaces) and compressed streams in DV format [7]. Support for other formats will be implemented soon. It can retrieves partial information about stream from it's SDP description,

though the support is not complete yet. At the moment we support either flat projection (one stream over one full window) or CAVE-like projection, when the streams are mapped onto walls of virtual cube.

Support for stereoscopic projection is handled by having two identical projections using windows on different screens which are expected to be projected to different eyes. Adjustments of interocular distance are supported. Each of the projections can consist of several windows. As basic support for user avatars we have implemented "faces" - tiles floating in the scene (meant for "backward" streams from cameras installed in the cave, but the player can use streams from any source for this purpose). Support for movement in the local scene - in predefined range to simplify control. If output events are generated, they are written to a file specified by configuration.

3.3 Future Enhancements

In short-term future work, we would like to address problems already mentioned - Complete support for SDP [6] description (this would simplify the configuration of incoming streams), external models of a virtual scene (currently there is only hard-coded CAVE-like cube) and support for active stereoscopy. Also we want to have the application to be distributed over several computers.

4 Interaction

When we want to turn the passive receiver into an active participant, we need to give him chance to control the remote application. This section describes our proof-of-concept solution. As most of the applications, we use now, have their inputs based on trackd[5] daemon, our solution is targeted to applications using *trackd*. Our goal is to give the user ability to control the remote application, without need to modify the application.

4.1 Implementation

As the development platform for our project is Linux based operating system, we describe some details using terminology originating from this environment. But the principles of main ideas are generally independent of used platform even if the implementation on other platforms can be more complicated. The implementation has three main parts:

— new driver for *trackd*,
— kernel module for virtual input device,
— player capable of generating the events.

[5] http://www.mechdyne.com/integratedSolutions/software/products/trackd/trackd.htm

4.1.1 Trackd Driver

The idea for the driver is to make it receive events from several input devices and to present events from any of the devices equally. Using this approach we can have the configuration unchanged for the normal single controller operation, but we can have several input devices that controls the scene. When we add some kind of virtual device as one of those, we can insert our own events very simply and cleanly.

4.1.2 Virtual Event Device

To get the virtual device mentioned in Section 4.1.1, we implemented linux kernel module simulating an input device. It is controlled by an interface in `/proc`[6] filesystem. Basically it receives events in the form of `input_event`[7] structure and simply replicates them in kernel.

4.1.3 Capable Player

Our player is capable of generating the event structure recognized by the device described in Section 4.1.2. It outputs them into a file specified by configuration. In order to give those to the remote virtual event device, we dump them into a pipe instead of file and then redirect the pipe through a `ssh` tunnel to the remote machines `/proc` interface.

4.2 Data Transmission

The transmission of tracked data in our plans should be handled by a transparent methods defined in context of all other data types transport management. For now, the proof-of-concept implementation uses ssh for data transmission.

5 Applications

The possibility to capture rendered video in real-time and present it on a distant place has lot of applications in wide area. After we described some implementation details we can mention some applications which already use our grabber and player.

5.1 Project C2C

The methods introduced above in Section 1.3 have been successfully used in the Cave2Cave project (C2C) [3] to stream a stereoscopic video signal from applications running in CAVE-like system (CAVE visualization projection device used in the area of virtual reality applications [4]) and to present it on remote site (see Figure 4) using multichannel video player (Section 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 [8]. The grabber also creates RTSP [9] server to provide SDP descriptions [6] of the streams. This way we can (and we do) present applications from our CAVE system to distant

[6] Dynamically generated pseudo filesystem used to access process information from the UNIX kernels

[7] C-language structure used by linux kernel to send notification about events from input devices

viewers. The use of standard streaming protocols allows us to partially preserve possibility of receiving data by standard players used by remote user.

5.2 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 (Virtual Reality Modeling Language) models to public.

5.3 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.

6 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 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.

7 Future Work

The work described in this report is part of project *CAVE to CAVE* (C2C) which addresses to development and application of new approaches in the area of distributed collaborative environments. The report describes already implemented parts which has been used to get a proof-of-concept. The next activities in framework of the project will tend to extension of actual state according to ideas already mentioned in [2]. These extensions should be based mostly on technical improve-

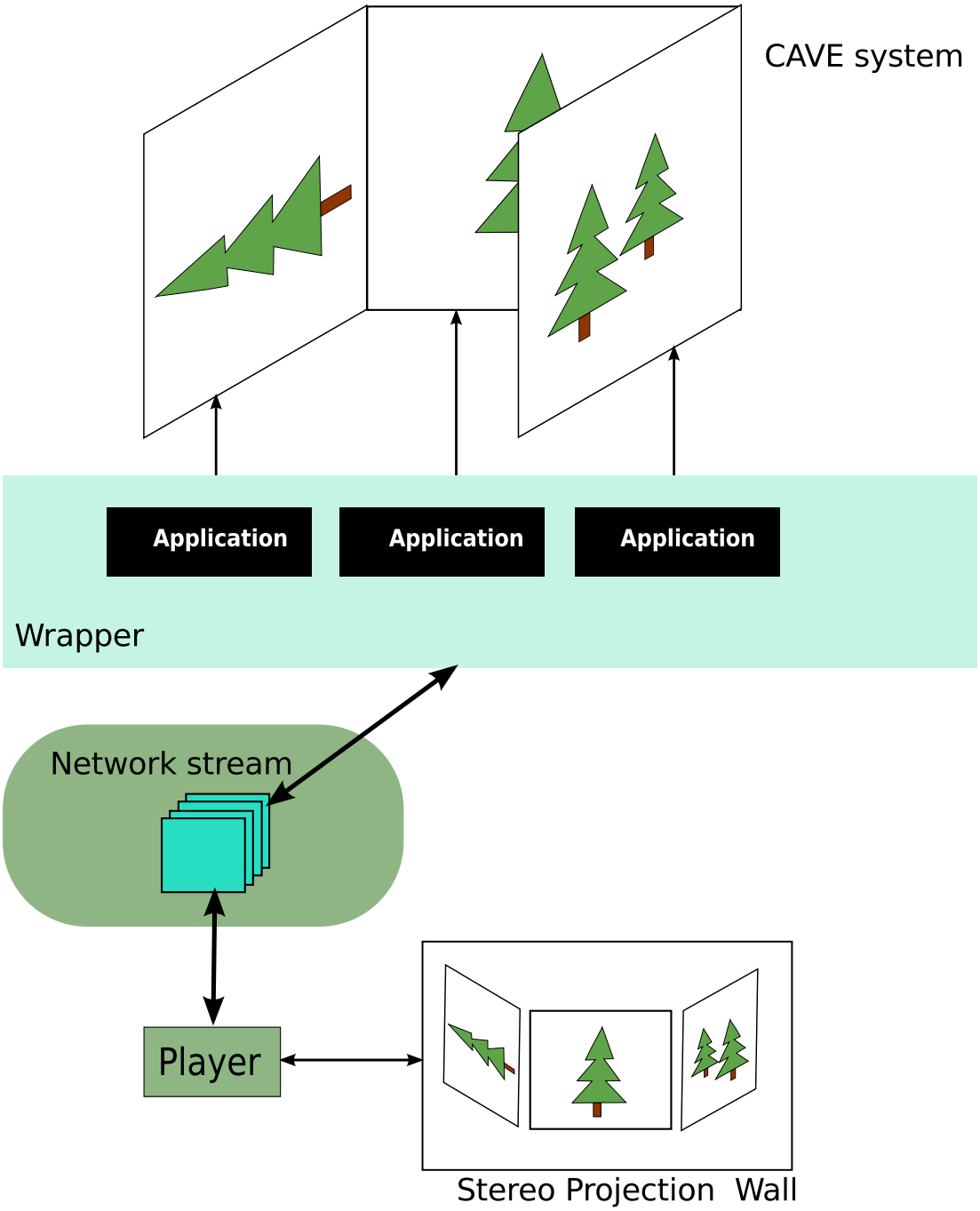

Figure 4. Scheme of the configuration based on multi-projection screen. A scene rendered in the resource device with 3 projection walls is grabbed and the resulting video is transmitted to the remote device where it is presented on remote projection wall.

ment of several parts in whole communication chain presented in Section 1.3 (Figure 2). This includes more flexible processing of tracked data to allow higher variability of tracking devices. Next, a "back" video channel (with pictures of remote users) and its composition with the presented scene directly on screen is standalone problem which is now solved by authors.

The next significant topic for the C2C project is a definition of new generation of concept which will be based on higher level of communication control among

VR devices. The new level should be characterized by the ability to realize adaptation of multimedia content according to end-point device taking into account bandwidth, number of information channels (e.g. mono vs. stereo video), security or/and permissions of each endpoint-device. Such communication platform offer its possibilities for more applications then only for collaborative distributed environments. The ideas about this level of communication forms new visions where multi-modal information does not overload the user but improves his/her creative potential.

8 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

[1] BAIK, S.; BALA, J.; JO, Y. *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 3681/2005, chapter Distributed Visual Interfaces for Collaborative Exploration of Data Spaces, p. 887–892. Berlin: Springer, 2005. ISBN: 978-3-540-28894-7.

[2] BERKA, R.; TRÁVNÍČEK, Z.; HAVRAN, V.; BITTNER, J.; ŽÁRA, J.; SLAVÍK, P.; NAVRÁTIL, J. *CAVE to CAVE: Communication in a Distributed Virtual Environment*. Technical report 23/2008[8], Praha: CESNET, 2008.

[3] BERKA, R.; TRÁVNÍČEK, Z.; HAVRAN, V.; BITTNER, J.; ŽÁRA, J.; SLAVÍK, P.; NAVRÁTIL, J. CAVE to CAVE: Communication in a Distributed Virtual Environment. In LHOTKA, L.; SATRAPA, P. (ed.). *Networking Studies III: Selected Technical Reports*, Praha: CESNET, 2009, p. 161–174. ISBN: 978-80-904173-4-2.

[4] CRUZ-NEIRA, C.; SANDIN, D.; DEFANTI, T.; KENYON, R.; HART, J. The CAVE: Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, vol. 35, no. 6, 1992, p. 65–72.

[5] GHARAI, L.; PERKINS, C. *RTP Payload Format for Uncompressed Video*. RFC 4175[9], IETF, September 2005.

[6] HANDLEY, M.; JACOBSON, V.; PERKINS, C. *SDP: Session Description Protocol*. RFC 4566[10], IETF, July 2006.

[7] KOBAYASHI, K.; OGAWA, A.; CASNER, S.; BORMANN, C. *RTP Payload Format for DV (IEC 61834) Video*. RFC 3189[11], IETF, January 2002.

[8] http://www.cesnet.cz/doc/techzpravy/2008/cave-to-cave/

[9] http://tools.ietf.org/rfc/rfc4175

[10] http://tools.ietf.org/rfc/rfc4566

[11] http://tools.ietf.org/rfc/rfc3189

[8] SCHULZRINNE, H.; CASNER, S.; FREDERICK, R.; JACOBSON, V. *RTP: A Transport Protocol for Real-Time Applications.* RFC 3550[12], IETF, July 2003.

[9] SCHULZRINNE, H.; RAO, A.; LANPHIER, R. *Real Time Streaming Protocol (RTSP).* RFC 2326[13], IETF, April 1998.

[12] http://tools.ietf.org/rfc/rfc3550

[13] http://tools.ietf.org/rfc/rfc2326