The Omniscope - Multimedia Streaming and Computer Vision for Applications in the Virtuality Continuum

Gerald Melles[1]

Abstract: Researching applications within the Virtuality Continuum (VC) is a process involving combinations of many different technologies. Media streaming and computer vision in particular are important aspects of many VC applications. This paper introduces the Omniscope library as a way to integrate both in an efficient, user-friendly and extensible manner. It achieves this by combining GStreamer and OpenCV in a C/C++ library as well as a plugin for the Unity IDE.

Keywords: virtuality continuum; VR; streaming; computer vision; OpenCV; GStreamer; Unity

1 Introduction

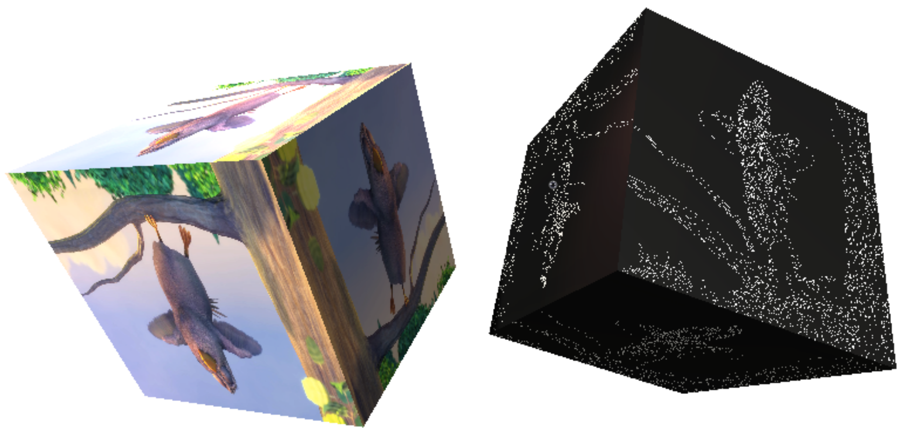
Fig. 1: Omniscope Plugin rendering and transforming a Full-HD video using edge detection

Multimedia streaming is a common requirement in systems within the virtuality continuum (VC) which encompasses Augmented, Virtual, Mixed and Blended Reality (cf. [MK94]). Among other aspects, media streams are often used to transport the required data for visual and auditory stimuli to and from peripheral devices.

[1] Hamburg University Of Applied Sciences, Fakultät TI-I, Berliner Tor 5, 20099 Hamburg, Deutschland contact@geraldmelles.com

VC applications also make extensive use of computer vision algorithms, such as in camera-based tracking and motion capture systems. In [Ng17] for instance, Nguyen et al. propose a new and less obtrusive design for markers similar to QR-Codes and suggest their use in VR applications.
VC research projects often need combinations of both of these aspects, for example to take the user's gaze into account to render in a foveated manner, such as in [Lu17]. There is no lack of projects and proposals in this intersection, but their tooling tends to be either purpose-built for a specific use case or proprietary.
Generic multimedia streaming and computer vision libraries exist, in both proprietary and open source variants. The main challenge lies in their integration into the game or simulation engine at the core of the VC application. This is exacerbated by requirements concerning speed, efficiency and extensibility, the maintenance of platform independence and the support for a range of data formats and devices. This may go some way toward explaining why there is no generic and open solution. Computer vision algorithms in particular often remain inaccessibly embedded within proprietary software and hardware products. Rapidly prototyping a VC application combining the two as well as maintaining the ability to change their internal workings is not supported by software solutions that are available today.
The Creative Space for Technical Innovations (CSTI) at the Hamburg University of Applied Sciences (HAW) primarily uses the Unity IDE as the simulation engine for developing their applications. This proprietary IDE is one of only a few viable options for the development of state-of-the-art VC systems, especially when developing a new simulation engine is not an option (i.e. due to the time and effort required). It can be extended through scripting and plugins in a variety of programming languages. At the time of writing, Unity is one of the most commonly used IDEs for VR and AR simulations, with its main competitors being other proprietary products like the Unreal Engine and Cryengine. In part, this dominance of proprietary engines may be related to the need to support the peripherals and platforms unique to VC systems, such as head-mounted displays, wands and motion tracking systems, which are also proprietary themselves.
Both the IDE's built-in media streaming support and its open source plugins lack a way to efficiently interface them with computer vision libraries, in addition to providing only very limited support for formats and encodings. There are also no comprehensive open source integrations of computer vision libraries themselves.
This paper presents the Omniscope library and plugin for Unity as a means of making the research and development of media streaming and computer vision based applications in the virtuality continuum faster, easier and more efficient. It was developed as a link between the two fields: A C/C++ library combining the GStreamer and OpenCV libraries. It is intended to make media streaming and computer vision features as accessible and customizable as possible for researchers into VC systems at the CSTI and elsewhere and can be integrated into Unity using the IDEs native plugin architecture. Plans exist to adapt it for other VC development environments, such as the Unreal Engine.
Fig. 1 shows the Plugin rendering an MP4 video[Bi08] onto cubes in a Unity scene. On the right cube, an edge detection algorithm was applied before rendering (an implementation of the algorithm proposed by John Canny in [Ca86]).

2 Architecture

Audio and video streams flow back and forth between servers, the VC simulation system and its various peripheral devices. A large number of standards, formats and encodings are in use, both free and proprietary. A few streaming frameworks seek to support as many of these as possible, both by endeavoring to be agnostic to the nature of the data they handle and by utilizing plugin architectures for components like encoders/decoders to ease their integration. One such framework is GStreamer, which is largely platform-independent and included in most Linux distributions.
OpenCV is a well-established computer vision library. Its open source nature has made it a popular basis for research into computer vision, particularly of feature detection algorithms. While it does already offer an integration with GStreamer, the Omniscope library instead offers its own in order to allow for better extensibility (i.e. other streaming framework integrations) and more complete playback support.
The Omniscope plugin links with both GStreamer and OpenCV dynamically. This is largely to accommodate the possibility of the user requiring all of the libraries' modules to have a specific (e.g. non-proprietary) licensing model and to enable its use in applications without licensing issues.[2]
Similarly, the plugin's functionality is exported to Unity using C-style bindings in a dynamically loaded library.[3]
The plugin's pipeline architecture with elements, sources and sinks is similar to that of GStreamer. Like GStreamer, the Omniscope library also largely abstracts from its elements' implementations. This way, elements are also more easily added to or removed from a project, regardless of their dependencies.
The Omniscope project currently consists of six C/C++ subprojects: Five internal modules and a standalone application. These subprojects are intended to be built using the GNU GCC Compiler (cf. [GN18]) for Linux-based systems or cross-compiled for Microsoft Windows using MinGW-w64 (cf. [Mi18]). Additionally, the project contains C# scripts for easier access to the library from within the IDE.

- Omniscope Common Module
- Omniscope GStreamer Module
- Omniscope OpenCV Module
- Omniscope Core Module
- Omniscope Unity Module
- Omniscope Standalone application

[2] Note that all GStreamer plugins used by the Omniscope plugin are licensed under the LGPL and OpenCV under the 3-clause BSD license.
[3] The Omniscope Unity module itself contains some code by Unity Inc., under the MIT license.

Omniscope Common contains header files with the pipeline elements' base interfaces and a custom thread pool implementation.
Omniscope GStreamer contains element specializations for GStreamer pipelines and elements. It provides support for GStreamer's gst-launch command line syntax, which is the recommended way for the user to extend the Omniscope plugin's streaming functionality without having to resort to working with its source code. It is required that the user supply at least one appsink or appsrc element in the pipeline definition as these are used for the exchange of samples between GStreamer and other pipeline elements.
Omniscope OpenCV consists of elements offering stream capture and processing implementations using OpenCV. It can be extended with custom sample analysis and processing modules. In its simplest form, a new sample processor can be implemented by providing a new override to a single function taking and returning a generic media sample instance (which may contain several frames and additional information). By default, sample processing functions are automatically executed asynchronously using a thread pool.
The Omniscope Core provides the pipeline itself: an API for instantiating and connecting sources, sample processors and sinks, manipulating their state (e.g. playing, pausing and seeking) and accessing the resulting media samples and analysis results.
The Omniscope Unity module adapts Unity's low-level native rendering plugin support (in C++) to permit rendering captured and optionally processed frames directly to existing target textures. This low-level access to Unity's rendering APIs makes the Omniscope much faster than passing frames up to Unity's C# scripts for rendering would be. It is also capable of interfacing with different graphics architectures and maintains the same platform independence as the rest of Omniscope's modules. In addition, any analysis results of sample processors can also be accessed through it. It is nevertheless recommended to use the provided C# scripts to facilitate communication between the Omniscope library and Unity. These use the Unity IDE's component system to provide a graphical UI (cf. 'Usage').
The project also contains a simple standalone application built upon the other modules (excluding the Omniscope Unity module). This primarily serves as a means to ease development and testing of new features but could also be used for the development of standalone streaming and computer vision applications.

3 Usage

Most of Omniscope's complexity can be hidden behind a graphical UI. The user may interact with the plugin in four ways (ascending by flexibility and descending by ease of use):

- Using Omniscope's UI (in Unity)
- Using Omniscope's C# API (within Unity's scripting system)
- Using Omniscope's 'C' style linkages
- Extending Omniscope's modular architecture in C/C++

The first option consists of an extension to Unity's component inspector. It offers input fields and buttons (no programming is required) and is intended to be used primarily for simple audio and video playback and the use of pre-built frame processors. The use of GStreamer's gst-launch syntax adds additional flexibility.
The second option is intended for developers who either want to have programmatic control over the plugin's behavior at runtime (e.g. for pausing playback or seeking) or to use the built-in computer vision algorithms for frame-by-frame analyses. For example, it could be used to apply a feature detection algorithm as a sample processor and receive the relative positions of detected features (such as eyes and faces) once per render cycle.
If the C# API does not suffice or if the plugin is to be used outside the Unity IDE, the user may also use the plugin's 'C' style linkages.
Lastly, as mentioned above, the Omniscope's architecture has been designed with extensibility in mind. Its central features are abstracted through interfaces and abstract base classes, easing the development of new elements, such as frame processing or stream capture implementations.
The Omniscope's focus lies on video and vision, but it also offers limited support for other media such as audio streams and subtitles to accompany video playback.

4 Conclusion

The need for well-integrated, efficient, open and extensible streaming and computer vision support for the research and development of VC applications has been established. The Omniscope project has been introduced as a solution, combining the GStreamer and OpenCV libraries and optionally integrating them with the Unity IDE. Its uses in VC applications include playing back multimedia streams from a variety of sources (both video and audio), transforming the visual frames - for example to show detected edges in a stylized way - and extracting other data through computer vision algorithms, such as the location of detected faces or tracking markers. It is being used in the CSTI and steadily improved and extended.

5 Further Research

While the Omniscope is already in use by researchers in the CSTI, formal and exhaustive analyses of its speed and efficiency have yet to be performed.
The Omniscope could also serve as a means of integrating other media based features with VC applications, such as three-dimensional audio processing based on the acoustics of a virtual space. Related work is being done at the HAW's wave field synthesis lab, for example exploring the use of acoustics in redirected walking (cf. [NF16]).
The capture of multimedia streams and their processing by computer vision algorithms are also only some of the aspects of networked VC systems. These systems' communication with various peripherals, local software and hardware landscapes and remote services provides many other challenges worthy of research. Of particular interest to researchers in the CSTI

is the integration of embedded systems and smart environments in VC systems. To this end, a number of projects have sprung up, such as the CSTI middleware (cf. [Ei17]).

References

[Bi08] Big Buck Bunny, original by the Blender Foundation (peach.blender.org), official mirror by Janus B. Kristensen (http://bbb3d.renderfarming.net), last accessed 01.07.2018).

[Ca86] Canny, J.: A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6):679–698, Nov 1986.

[Ei17] Eichler, Tobias; Draheim, Susanne; Grecos, Christos; Wang, Qi; von Luck, Kai: Scalable Context-Aware Development Infrastructure for Interactive Systems in Smart Environments. In: Fifth International Workshop on Pervasive and Context-Aware Middleware 2017 (Per-CAM'17). Rome, Italy, October 2017.

[GN18] GNU Compiler Collection (official site): https://gcc.gnu.org/.

[Lu17] Lungaro, Pietro; Tollmar, Konrad; Mittal, Ashutosh; Valero, Alfredo Fanghella: Gaze- and Qoe-aware Video Streaming Solutions for Mobile VR. In: Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology. VRST '17, ACM, New York, NY, USA, pp. 85:1–85:2, 2017.

[Mi18] MinGW-w64 - GCC for Windows 64 & 32 bits (official site): https://mingw-w64.org.

[MK94] Milgram, Paul; Kishino, Fumio: A Taxonomy of Mixed Reality Visual Displays. In: IEICE Trans. Information Systems. volume vol. E77-D, no. 12, pp. 1321–1329, 12 1994.

[NF16] Nogalski, M.; Fohl, W.: Acoustic redirected walking with auditory cues by means of wave field synthesis. In: 2016 IEEE Virtual Reality (VR). pp. 245–246, March 2016.

[Ng17] Nguyen, Minh; Tran, Huy; Le, Huy; Yan, Wei Qi: A Tile Based Colour Picture with Hidden QR Code for Augmented Reality and Beyond. In: Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology. VRST '17, ACM, New York, NY, USA, pp. 8:1–8:4, 2017.