Vision and Video in Virtuality

M.Sc. Computer Science

Gerald Melles
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg, Germany
contact@geraldmelles.com

ABSTRACT

In this paper, I am introducing my Omniscope Plugin for Unity as a useful tool for video playback as well as video analysis and processing in VR applications. It consists of two distinct modules: A GStreamer pipeline for receiving media streams from a variety of sources and an integration of the OpenCV computer vision library.

KEYWORDS

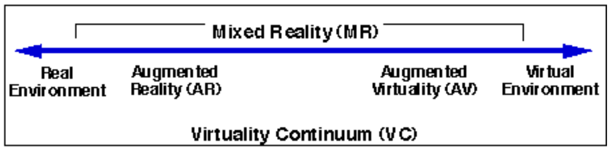
virtual worlds, VR, AR, Streaming, OpenCV, GStreamer, Unity

Disclaimer. This paper was created as part of a project in preparation for a master's thesis at the Hamburg University of Applied Sciences. Neither it nor any accompanying material are to be used or published for any purpose other than the HAW's regular examination and evaluation procedures.

1 (COMPUTER) VISION AND VIDEO IN THE VIRTUALITY CONTINUUM

Today's VR systems seek mainly to provide artificial visual and auditory stimuli in order to create the illusion of another - virtual - reality. All the way from Augmented Reality through Blended Reality and Augmented Virtuality to Virtual Reality (cf. [5]), the focus lies on vision above all other senses, with auditory senses coming second.
The capturing, streaming, processing and presentation of video and audio is, therefore, an important part of these technologies. Not all applications end up streaming media feeds directly to the user, either: Video feeds from cameras in the HMD (Head-Mounted Display) may just as well be used to facilitate inside-out motion tracking or feature detection based overlay of virtual entities in an otherwise real environment.
This may perhaps be the most important in Augmented Virtuality, where aspects of real perception are injected into an otherwise virtual experience. If, for example, an HMD with stereoscopic cameras (such as the HTC Vive Pro) was being used to immerse the user in a virtual environment, but using the cameras people in their surroundings were still being shown, computer vision algorithms would be at the center of that mechanism.
According to some researchers, the use of computer vision as a means of facilitating human-computer interaction is on the rise and may even surpass the use of the currently ubiquitous touch-interfaces (cf. [8]).
As such, media streaming and the processing of images and video are important parts of VR systems and their interconnection with other systems (such as peripheral devices and media providers). A number of proprietary frameworks offer streaming and computer vision capabilities to developers, but typically in the form of ready-made features. Since open source implementations for both do exist, we have created a lean plugin efficiently interfacing a media streaming framework (GStreamer), a computer vision library (OpenCV) and a VR development environment (Unity3D). All three of these are largely platform independent, as versions exist for Windows as well as Unix systems.[1] We will show how this was done and why it may prove useful for scientists studying aspects of systems in the Virtuality Continuum.

Figure 1: Simplified illustration of Milgram and Kishino's Virtuality Continuum [5]



This paper is structured into three parts:
Firstly, the importance of computer vision as a part of building VR systems is highlighted, together with an introduction into the OpenCV computer vision library and existing integrations with the Unity IDE. In the second part, the particular requirements of video streaming over networks are discussed, explaining the necessity to offer functionality beyond Unity's and OpenCV's built-in streaming support, using the GStreamer media framework.
Thirdly, the Omniscope plugin is introduced as a combined media framework and computer vision solution, efficiently integrating both GStreamer and OpenCV into Unity.

1.1 Unity

Unity (or Unity3D)[10] by Unity Technologies is an integrated development environment for the creation of multimedia applications, especially computer games. It is capable of producing applications for several platforms and currently the most often used framework for creating virtuality applications.

2 COMPUTER VISION

2.1 OpenCV

OpenCV is a free and open source program library for computer vision algorithms. Much of its code base and the underlying algorithmic concepts stem directly from the scientific community. Other reasons for its widespread adoption include its speed (it is written almost exclusively in native code), its extensive features,

[1] Though the implementation uses native C and C++ code for reasons of performance and interoperability, making it necessary to compile it separately for each platform.

platform independence and open (BSD) license.
OpenCV's algorithms are widely used in the processing and analysis of images and video, even in the field of AR and VR. Example for this can be found in [1] or [7].

2.2 OpenCV and Unity

At least one integration of OpenCV in Unity exists, but it is closed source and currently being sold via Unity's own asset store. As far as its limited documentation reveals, its functionality is limited to a small subset of OpenCV's capabilities. It is also not possible to extend the Plugin's functionality by using another variant of OpenCV.
Using EmguCV[3], the C# wrapper of OpenCV, in Unity's C# code scripts would present another alternative. This would come with a number of other problems, however. The largest one is that Unity's C# support is currently based on a legacy version of the Mono framework, which makes it seem unlikely that EmguCV would work as intended without further programming effort. Furthermore, EmguCV naturally trails behind OpenCV in terms of supported features and is not as fast as the original native code libraries.

2.3 Unity's Native Plugin System

The Unity Editor is able to use native code libraries to extend its own scripting capabilities. This simply requires placing the relevant libraries (e.g. .dll files) in the Assets/Plugins folder of a Unity project. [2] Native libraries can then be accessed by calling their functions from Unity's own scripts (e.g. C#, JavaScript) after specifying which DLL should be used.

3 MEDIA STREAMING

3.1 Streaming and Virtuality

Media streaming is another important part of the tool-set of any VR system developer. Examples for this include the ubiquitous streaming of 'traditional' videos with rectangular aspect ratios, but also 360 degree videos (such as OpenCV's algorithms could be used to 'stitch' together), stereoscopic 3D and surround sound. Since VR systems are often used to play back very bandwidth-hungry streams (such as 360 degree video), there has been a recent surge in research activity into facilitating this in an efficient manner. In [12] for example, Zare et al. propose to track the user's gaze and save bandwidth by adaptively reducing the quality of areas they are not actively looking at. In [2], Cheung et al. propose a similar solution.

3.2 Unity's own support for video streaming

As of its latest beta release (2017.1), the Unity3D Editor supports two different kinds of streaming features: The WebCamTexture and VideoPlayer components. WebCamTexture allows for local webcam feeds to be rendered onto a texture. VideoPlayer allows for rendering remote video streams onto a texture. However, both come with serious limitations and problems: Neither of them allows for efficient pre-processing of the received frames. WebCamTexture does not permit playing back video from remote cameras (e.g. so-called IP-Cameras). The VideoPlayer is incapable of playing back live streams with playback delays under a minute and in fact incapable of playing back most live streams at all. Even video files being streamed over the network proved to be problematic, as the VideoPlayer would refuse to play back some container format and encoding combinations altogether. It should be said that the VideoPlayer is currently only available as a beta feature and no claims have been made as to its state of completion.

3.3 OpenCV stream capture

OpenCV permits processing and changing video data quickly and efficiently, either through custom code routines or through a multitude of already available algorithms. Since it was not intended for playback, its capabilities in that regard are very limited. For example: While it is possible to receive a video stream or file using OpenCV, it will then normally attempt to process the video's frames as quickly as possible. Therefore, if OpenCV were to be used to play back a video file, the delay between frames would have to be implemented manually to ensure the intended playback speed. This and other missing common features of video playback such as accompanying media (subtitles, audio) or searching (finding a specific position in the time-line of a played-back media file) led to the decision of also integrating a dedicated multimedia framework. It should be noted that OpenCV does already include the possibility of an integration with GStreamer in its plethora of features (it needs to be re-compiled with the relevant options enabled), so this is by no means a novel concept. This integration is somewhat limited in some regards, however (cf. ch. 4), and since a separate GStreamer installation is required for it to work in any case, the decision was made to instead create an integration of GStreamer into Unity while keeping compatibility with OpenCV.

3.4 GStreamer media framework

GStreamer[4] is a multi-platform library for the creation of media-handling applications. One of its core capabilities is the streaming of multimedia content, both sending and receiving. Aside from being accessible though its core libraries (which are written in C), GStreamer provides access to its features through command-line programs. GStreamer is included in all major Linux distributions and in many ways equivalent to Microsoft's DirectShow filters. It can, however, be run on Windows and MacOS operating systems as well as on Linux (and derivates, such as Android).

3.5 GStreamer pipeline

GStreamer works by creating a *pipeline*, which is essentially a list of the modules which are to be used, each consuming data from its predecessor and providing data to its successor. A definition of a pipeline can be passed to GStreamer through the command line (see example pipeline definition in ch. 4) or the pipeline can be manually created in code. GStreamer categorizes the modules along its streaming pipeline into three types: Sources, Filters (and filter-like modules) and Sinks.
A Source is the initial element of a pipeline which provides a data stream which is then fed into a succession of Filters. The Filters transform and modify that stream, passing it on to other filters

[2]Care has to be taken, however, if these libraries dynamically link to other libraries, such as (in this case) Visual Studio redistributables and both GStreamer and OpenCV. If these are not found, Unity will (confusingly) claim that the native plugin itself does not exist at all.

and, finally, to a Sink. Sinks are the end of a streaming pipeline and typically pass the stream along to other programs, such as OpenCV or Unity in the case of this plugin.

3.6 GStreamer and Unity

There have been previous efforts to make GStreamer's streaming capabilities accessible from within the Unity IDE. Two (interrelated) Open Source native plugins for this purpose exist. One is Yamen Saraiji's GStreamer Unity Integration[6], which is still under active development. Another project based on this is gst-unity-bridge by the ImmersionTV project[11], funded by the European Union's horizon project. The latter seems to no longer be actively developed.

4 OMNISCOPE

The Omniscope Plugin for Unity has the following essential functions:

- Receiving and processing media streams with GStreamer, e.g. from webcams, window/screen captures, files or network streams
- Multi-threadable C++/C# interoperability of the OpenCV library with Unity
- Audio playback
- Video playback on surfaces managed by Unity by rendering video frames directly to textures on the GPU, from either GStreamer or OpenCV

4.0.1 Interoperability with Unity. The Plugin uses Unity's provided Low Level Native Plugin Interface for interoperability with the IDE, as well as Unity's Rendering API for accessing the necessary DirectX functionality to render OpenCV's output onto textures. The capture of video frames from streams is de-coupled from the rendering cycle responsible for rendering the texture. Full control over when capture frames and textures are updated - and on which thread - is available through the C#-API of the plugin rather. That way, performance can more easily be tweaked depending on each scenario's specific requirements.
The Plugin also offers the StreamPortal Component (see Fig. 2) as a graphical interface to aid non-programmers in its parametrization. For example, StreamPortal offers to dynamically rescale the object it is attached to relative to the resolution of the received video stream. This avoids cropping or rescaling the receiver texture, which could otherwise result in a diminished playback experience.

4.0.2 Reading video frames. Using OpenCV, capturing video from files and cameras is relatively easy. OpenCV provides extensive support and a high-level API for this in the VideoCapture class. Webcams are selected using their zero-based webcam device index, which is potentially problematic as OpenCV (surprisingly) does not have any convenience function for matching device names with indices, limiting users' options to iterating over the devices until the correct one has been identified using its device name. Unfortunately, this device name is also not unique, which means that if more than one device has the same (or no) device identifier it may be necessary to examine the actual video feed to make sure the right device is selected - which will likely require end-user effort.
Screen capture is somewhat more complex than webcam and file

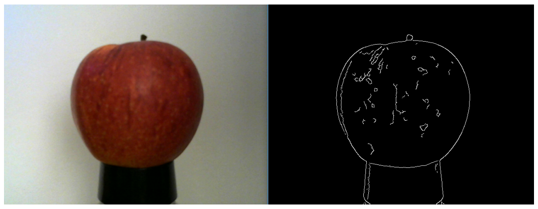
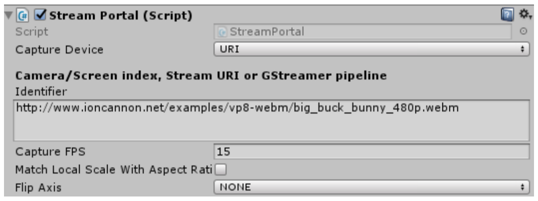


Figure 2: OpenCV applying an edge detection algorithm to a live webcam feed



Stream Portal (Script)

Script	StreamPortal
Capture Device	URI

Camera/Screen index, Stream URI or GStreamer pipeline

Identifier

http://www.ioncannon.net/examples/vp8-webm/big_buck_bunny_480p.webm

Capture FPS	15
Match Local Scale With Aspect Rati	☐
Flip Axis	NONE

Figure 3: The Unity Component serving as front end to the Native Plugin

capture, as OpenCV does not support it natively on Microsoft Windows. Instead, another class was implemented (based on [9]) to serve as a drop-in replacement for OpenCV's VideoCapture class in this eventuality.
VideoCapture does allow capturing from a file and even live streams through a network using a number of transfer protocols, however, its playback functions are very limited. Ultimately, OpenCV is intended for analysis and processing media rather than playback.

4.0.3 Image processing. OpenCVs primary function is to serve as a library of image processing functions. With the Omniscope Plugin for Unity, exposing any and all of these to Unity is relatively straightforward and only requires tying another API function to OpenCV's relevant function calls. Aside from modifying video frames, OpenCV also offers many functions which perform analyses and extract information from pictures and videos. The results of these functions can be any number of different data types and structures. Using the aforementioned means of communication between the native plugin and the superordinate Unity instance, these could also easily be passed on to be used in a virtual reality simulation. An example for this could be the extraction of the location of features (such as QR-Tags or human faces) from an HMD's camera feed and its subsequent display through Unity.

4.0.4 Rendering frames to the graphics card. Using Unity's Native Rendering API, an OpenCV frame (a.k.a. Material or Mat) can be rendered to graphics memory. For this, a texture is created in the Unity project's C# code and a pointer to the target texture's memory location is passed on through the Plugin's API to native code. The rendering procedure is executed on Unity's graphics thread (and thus separately from its script execution) for performance purposes.

4.1 Omniscope's GStreamer Integration

Omniscope takes two different approaches to integrate GStreamer: Firstly, it uses and extends integration modules already available

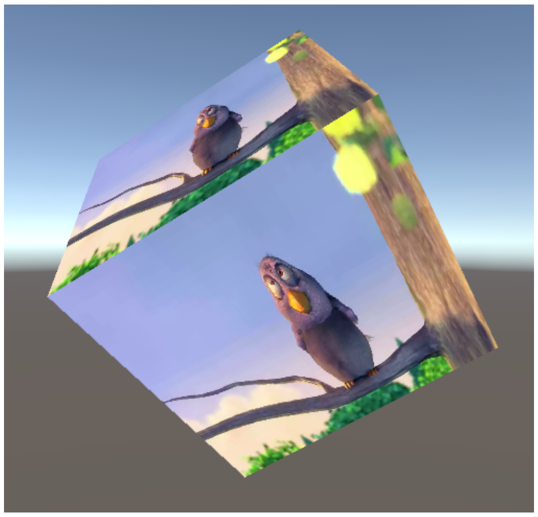


Figure 4: Video being streamed from a webserver (source!) and rendered onto cube

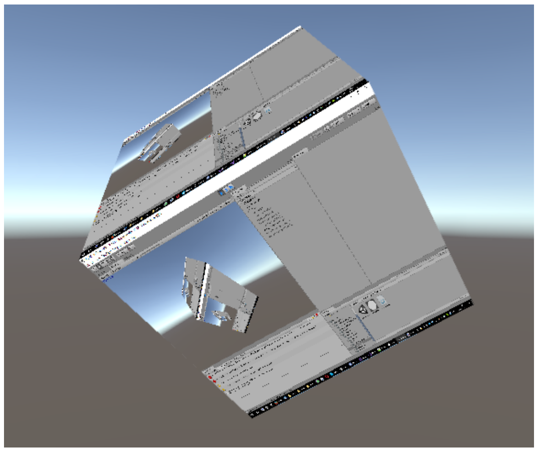


Figure 5: Current primary desktop being captured and rendered onto cube: note the infinity-mirror effect

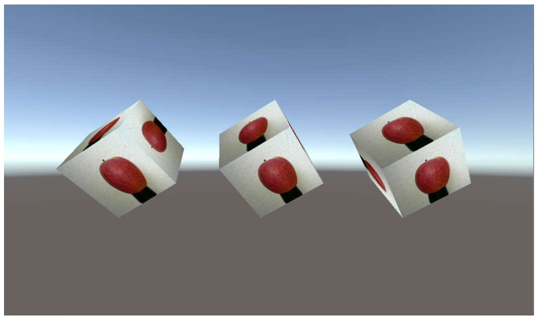


Figure 6: The same webcam capture being rendered onto three cubes simultaneously

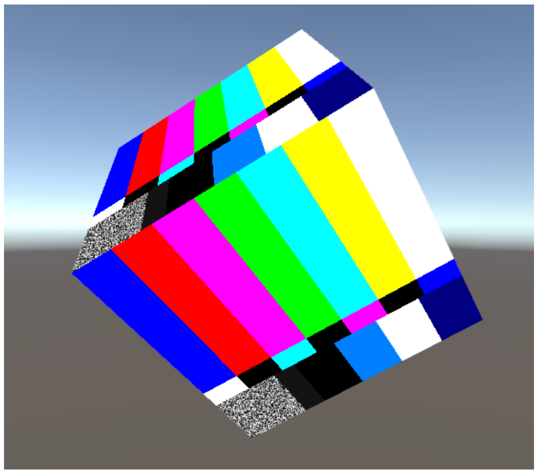


Figure 7: GStreamer pipeline example rendered onto cube

within OpenCV itself. The GStreamer integration modules required for this are only available if OpenCV is manually built from a recent source, in this case its git development branch.
The integration does not change the user interface within Unity either, instead accepting a GStreamer pipeline definition string as a valid URI. As a result, the Plugin can easily and flexibly be used to access the majority of GStreamer's functionality without any additional programming effort.
The GStreamer integration adds a lot of functionality, more than could be discussed here - interested readers should refer to GStreamer's own documentation instead. There are limitations, however. GStreamer capabilities (a.k.a. 'caps') programmed into OpenCV are, at this point, limited to video input to either 1 or 3 channel 8 bit raw video or bayer video format. Playback control and other features typically associated with media playback are also missing. This integration is therefore not intended to allow playback of media in VR applications, but rather to be used to tie media into computer vision algorithms and processing it. A good example for this could be feature detection, such as facial recognition or the tracking of a user's hands through the cameras on their HMD.
In addition to this, we have created an appsink component for GStreamer which allows for media to be rendered directly onto textures in Unity, bypassing OpenCV entirely. This is intended for media playback and is easily integrated into a custom GStreamer pipeline. It also surpasses the OpenCV integration in terms of performance by a large margin.

Example pipeline definition: 'videotestsrc ! videoconvert ! appsink' This example pipeline uses GStreamer's built-in video test source, converts it into the appsink's expected video format and pipes it into the Omniscope plugin (the destination sink has to be 'appsink' or 'opencvsink').

Tests. To ensure the correct and reliable operation of the receiving and rendering systems, the plugin was tested successfully for the following scenarios:

- Reading a video file from the local file system

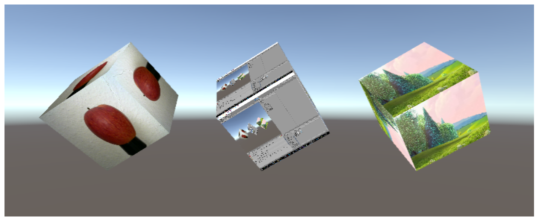


Figure 8: Three separate, simultaneous streams being rendered onto cubes. From left to right: a live webcam feed, a desktop capture and a video streamed from a web server

- Reading a video file from a web server using HTTP
- live-streaming a local GStreamer video test source
- live-streaming a local webcam capture
- live-streaming a local desktop capture
- live-streaming video from a web server using HTTP and WEBM

The plugin proved to be capable of rendering all of the above successfully with frame rates of 60 frames per second in FHD resolution (1920 by 1080 pixels) when using the OpenCV integration. It can handle multiple simultaneous inputs: In our tests up to three were possible without a drop in frame rate when using the direct GStreamer-Unity-Integration.

5 CONCLUSION

5.1 Summary

We have shown that an efficient and fast plugin integration of both a media fgramework and a computer vision library into Unity is possible. We have also highlighted the benefits of this integration being open source and as such modifiable and extensible to suit every application. In addition, we have described the most noteworthy challenges we have faced during its first prototypic implementation.

5.2 Future Works

Future works based on this paper and the accompanying code fall into two broad categories: improvements/extensions to the Omniscope Plugin on one hand and its application in VR research on the other.

In terms of improvements, the most obvious one would be to expose more of OpenCV and GStreamer's own functionality through Unity's component system in order to require as little programming effort on the side of the user as possible. This would make it easier esp. for non-computer scientists to work with.

In researching VR technologies, the plugin eases the implementation of scenarios integrating multimedia streams into VR applications. It offers easy interoperability with any system streaming multimedia data. It is also not limited to 'traditional' video streams, but could for example also stream two separate videos for stereoscopic video rendering. But its main strength lies in providing access to OpenCVs algorithms. As such, the use of computer vision in VR as well as the possibility of integration of the Omniscope plugin with other networking and messaging tools (such as the CSTI's Middleware and the mauzr framework) will be explored. The latter would be especially useful to systems in the virtuality continuum which rely heavily on smart environments, needing to send and receive both simple messages and multimedia streams.

REFERENCES

[1] Alba Amato, Salvatore Venticinque, and Beniamino Di Martino. 2013. Image Recognition and Augmented Reality in Cultural Heritage Using OpenCV. In *Proceedings of International Conference on Advances in Mobile Computing &#38; Multimedia (MoMM '13)*. ACM, New York, NY, USA, Article 53, 10 pages. https://doi.org/10.1145/2536853.2536878

[2] G. Cheung, Z. Liu, Z. Ma, and J. Z. G. Tan. 2017. Multi-stream switching for interactive virtual reality video streaming. In *2017 IEEE International Conference on Image Processing (ICIP)*. 2179–2183. https://doi.org/10.1109/ICIP.2017.8296668

[3] EmguCV. 2017. EmguCV .NET Wrapper for OpenCV (official website). (2017). http://www.emgu.com

[4] GStreamer. 2017. GStreamer media framework (official website). (2017). https://gstreamer.freedesktop.org

[5] Paul Milgram and Fumio Kishino. 1994. A Taxonomy of Mixed Reality Visual Displays. vol. E77-D, no. 12 (12 1994), 1321–1329.

[6] mrayy. 2017. GStreamer Unity integration. (2017). https://github.com/mrayy/mrayGStreamerUnity

[7] W. W. Oui, E. G. W. Ng, and R. U. Khan. 2011. An Augmented Reality's framework for mobile. In *ICIMU 2011 : Proceedings of the 5th international Conference on Information Technology Multimedia*. 1–4. https://doi.org/10.1109/ICIMU.2011.6122762

[8] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. 2012. Real-time Computer Vision with OpenCV. *Queue* 10, 4, Article 40 (April 2012), 17 pages. https://doi.org/10.1145/2181796.2206309

[9] StackExchange. 2017. Thread on ScreenCapture in OpenCV. (2017). https://codereview.stackexchange.com/questions/127667/opencv-basedwrapper-for-windows-screen-capture

[10] Unity Technologies. 2017. Unity IDE (official website). (2017). https://unity3d.com

[11] ua i2cat. 2017. GStreamer - Unity3D Bridge (GUB), a.k.a. Streamer Movie Texture (GMT). (2017). https://github.com/ua-i2cat/gst-unity-bridge,lastaccessed

[12] A. Zare, A. Aminlou, and M. M. Hannuksela. 2017. Virtual reality content streaming: Viewport-dependent projection and tile-based techniques. In *2017 IEEE International Conference on Image Processing (ICIP)*. 1432–1436. https://doi.org/10.1109/ICIP.2017.8296518