Deliverable

Project Acronym:	**VRTogether**
Grant Agreement number:	762111
Project Title:	*An end-to-end system for the production and delivery of photorealistic social immersive virtual reality experiences*

D3.8 Software Components final version

Revision: 1.3

Authors: Simon Gunkel (TNO), Argyris Chatzitofis (CERTH), Shishir Subramanyam (CWI), Romain Bouqueau (Motion Spell), Karim El Assal (TNO), Rick Hindriks (TNO), Frank ter Haar (TNO), Vincent Lepec (Viaccess), Dimitris Zarpalas (CERTH), Prodromos Boutis (CERTH), Gianluca Cernigliaro (I2CAT), Marc Martos (I2CAT), Jack Jansen (CWI), Patrice Angot (Viaccess), Francesca De Simone (CWI), Fernando Pérez (TheMo), Guillermo Calahorra (TheMo), Ana Revilla (TheMo)

Delivery date: M39 (24-12-2020)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 762111

Dissemination Level		
P	Public	P
C	Confidential, only for members of the consortium and the Commission Services	

Abstract: Final Software release for capture, encoding, orchestration and rendering. This includes testing towards facilitating integration and use in the pilots, as well as software description, technical evaluations and including technical documentation report targeting its use beyond the scope of the VR-Together project.

REVISION HISTORY

Revision	Date	Author	Organisation	Description
0.1	09-11-2020	Simon Gunkel	TNO	First draft
0.2	23-11-2020	Simon Gunkel	TNO	Allocation of input requests and pre-filling of document with existing text from the web
0.3	02-12-2020	Argyris Chatzitofis	CERTH	input CERTH
0.4	02-12-2020	Gianluca Cernigliaro	I2CAT	input I2CAT
0.5	04-12-2020	Vincent Lepec	VO	input VO
0.6	02-12-2020	Prodromos Boutis	CERTH	input CERTH
0.7	11-12-2020	Jack Jansen	CWI	input CWI
0.8	17-12-2020	-	CERTH, I2CAT, VO, MO	Various inputs
0.9	18-12-2020	Simon Gunkel	TNO	Editorial Formatting of document
0.9.1	18-12-2020	Guillermo Calahorra	TheMo	final input before review
1.0	22-12-2020	Jack Jansen Simon Gunkel	CWI / TNO	Review comments
1.1	22-12-2020	Simon Gunkel	TNO	Request for final input
1.2	23-12-2020	Simon Gunkel	TNO	Final Input from VO/MS/CWI/TNO
1.3	24-12-2020	Simon Gunkel	TNO	Final review and formatting

EXECUTIVE SUMMARY

D3.8 (Software Components final version) provides the final software release for capture, encoding, orchestration and rendering. This includes testing towards facilitating integration and use in the pilots, as well as software description, technical evaluations and including technical documentation. D3.8 consists of the software modules, as well as documentation, describing i) what modules are provided; ii) the functionality of the modules; iii) the expected input and output of the modules; iv) instructions on how to install, deploy and operate the modules; v) any additional information that may be relevant to partners interested in using the components. Related tasks are T3.1-T3.5.

CONTRIBUTORS

First Name	Last Name	Company	e-Mail
Simon	Gunkel	TNO	Simon.Gunkel@tno.nl
Rick	Hindriks	TNO	Rick.Hindriks@tno.nl
Frank	Ter Haar	TNO	Frank.TerHaar@tno.nl
Roelof	van Dijk	TNO	roelof.vandijk@tno.nl
Marten	van Antwerpen	TNO	marten.vanantwerpen@tno.nl
Karim	El Assal	TNO	karim.elassal@tno.nl
Romain	Bouqueau	Motion Spell	romain.bouqueau@motionspell.com
Argyris	Chatzitofis	CERTH	tofis@iti.gr
Shishir	Subramanyam	CWI	S.Subramanyam@cwi.nl
Francesca	De Simone	CWI	Francesca.De.Simone@cwi.nl
Juan Antonio	Nuñez	I2CAT	juan.antonio.nunez@i2cat.net
Dimitris	Zarpalas	CERTH	zarpalas@iti.gr
Marc	Brelot	Motion Spell	marc.brelot@gpac-licensing.com
Gianluca	Cernigliaro	I2CAT	gianluca.cernigliaro@i2cat.net
Marc	Martos	I2CAT	marc.martos@i2cat.net
Vincent	Lepec	Viaccess-Orca	vincent.lepec@viaccess-orca.com
Patrice	Angot	Viaccess-Orca	patrice.angot@viaccess-orca.com
Jack	Jansen	CWI	jack.jansen@cwi.nl
Prodromos	Boutis	CERTH	prod@iti.g
Fernando	Pérez	TheMo	fernando.perez@themoderncultural.com
Guillermo	Calahorra	TheMo	guillermo@themoderncultural.com
Ana	Revilla	TheMo	anarevilla@themoderncultural.com

CONTENTS

Revision History 1

Executive Summary 2

Contributors 3

Contents 4

Tables 6

Figures 7

1. Introduction 8

1.1. Purpose of this document 8

1.2. Scope of this document 8

1.3. Status of this document 8

1.4. Relation with other VR-Together activities 8

2. Components Overview 9

2.1. List of components 10

2.1.1. Task 3.1 Capture 10

2.1.1.1. 3D Capture 10

2.1.1.2. RGBD Capture 11

2.1.1.3. Simple PC Capture Modules 11

2.1.2. Task 3.2 Encoding 12

2.1.2.1. TVM encoding & transmission 12

2.1.2.2. Point Cloud (PC) encoding & decoding 13

2.1.3. Task 3.3 Orchestration & Delivery 14

2.1.3.1. Media/Session Orchestrator 14

2.1.3.2. RTMP Live Video Transmission - GUB 14

2.1.3.3. (PC/Audio/Live) DASH Sender - bin2dash 14

2.1.3.4. (PC/Audio/Live) DASH Distributor - evanescent 15

2.1.3.5. (PC/Audio/Live) DASH Receiver - SUB 15

2.1.3.6. (PC/Audio/Live) Live presenter 15

2.1.3.7. WEBRTC-VR-MCU 16

2.1.3.8. PC-MCU 16

2.1.4. Task 3.4 Rendering and Display 17

2.1.4.1. Web-based player 17

2.1.4.2 Unity based player 17

2.2 Component Progress Year 0 – 3 18

3. Software for Task 3.1 Capture 27
3.1. 3D CERTH Capture 27
3.2. RGBD Capture 32
3.3. Simple PC Capture 35
4. Software for Task 3.2 Encoding 37
4.1. TVM encoding & transmission 38
4.2. Point Cloud (PC) encoding & decoding 41
5. Software for Task 3.3 Orchestration & Delivery 43
5.1. Media/Session Orchestrator 44
5.2. RTMP Live Video Transmission - Gstreamer Unity Bridge 48
5.3. (PC/Audio/Live) DASH Sender - bin2dash 50
5.4. (PC/Audio/Live) DASH Distributor - evanescent 51
5.5. (PC/Audio/Live) DASH Receiver - Signals Unity Bridge (SUB) 52
5.6. (PC/Audio/Live) Live presenter 53
5.7. WEBRTC-VR-MCU 54
5.8. PC-MCU 58
6. Software for Task 3.4 Rendering and Display 60
6.1. Web-based player 61
6.2 Unity-based player 64
7. Integration & Software packaging 66
7.1. Integration Overview 66
7.1.1. Native pipeline 66
7.1.2. Web Pipeline 70
7.2. Packaging & Deployment Overview 71
7.2.1. Native pipeline 71
7.2.2. Web Pipeline 74
8. Conclusion 75

TABLES

Table 1 List of Component 9
Table 2 Component Progress Delta Table 18
Table 3. Detailed Delta Table - 3D Capture 28
Table 4. Detailed Delta Table - RGBD Capture 32
Table 5. Detailed Delta Table - Simple PC Capture 36
Table 6. Detailed Delta Table – TVM encoding and transmission 38
Table 7. TVM Performance – Draco 39
Table 8. TVM Performance – Corto 39
Table 9. TVM Performance – Jpeg 40
Table 10. Detailed Delta Table – Point Cloud encoding and decoding 42
Table 11. Detailed Delta Table – Orchestrator 45
Table 12. Detailed Delta Table – RTMP Live Video Transmission 49
Table 13. Detailed Delta Table – DASH Sender 50
Table 14. Detailed Delta Table – DASH Distributor 51
Table 15. Detailed Delta Table – DASH Receiver 52
Table 16 . Detailed Delta Table – Live presenter 53
Table 17. Detailed Delta Table – WebRTC VR MCU 55
Table 18. Detailed Delta Table – PC MCU 59
Table 19. PC-MCU Performance 60
Table 20. Detailed Delta Table – Web-based Player 61
Table 21 Full System end to end (glass-to-glass) delays: 62
Table 22. Detailed Delta Table – Unity-based Player 64
Table 23 Performance of VRT Unity Client 65

FIGURES

Figure 1. RGBD Capture performance measurement setup 34

Figure 2. RGBD Capture performance results 34

Figure 3. WebRTC MCU performance of 3D conference room with users rendered as 2D sprite (with green chroma background in 540x800px resolution) 56

Figure 4. WebRTC MCU client performance of 3D conference room with users rendered as 3D point cloud (from RGBD video with total of 1080x800px resolution) 57

Figure 5. WebRTC MCU client performance of Oculus Quest running Firefox reality browser with room as 360-degree image background and users rendered as 2D sprite (with green chroma background in 540x800px resolution) 57

Figure 6. Native pipeline reference layout [D5.5] 68

Figure 7. Low-level integration diagram of PC and TVM components in native pipeline [D2.5] 69

Figure 8 Installer of Native Client – Setup 72

Figure 9 Installer of Native Client – Completion 72

Figure 10. Overview of the Web Client Architecture and Setup 74

1. INTRODUCTION

1.1. Purpose of this document

The purpose of this document is to provide a comprehensive view on the software components that are developed in the VR Together project. This should aid any interested party including the consortium partners a clear view on the final status of components and any potential technical use beyond the scope of the project.

1.2. Scope of this document

D3.8 (Software Components final version) provides the final software release for capture, encoding, orchestration and rendering. This includes testing towards facilitating integration and use in the pilots, as well as software description, technical evaluations and technical documentation. D3.8 consists of the software modules, as well as documentation, describing i) what modules are provided; ii) the functionality of the modules; iii) the expected input and output of the modules; iv) instructions on how to install, deploy and operate the modules; v) any additional information that may be relevant to using the components.

1.3. Status of this document

Following on the continuous development presented in D3.6, D3.8 extends the previous document with the current status of components for pilot 3 and thus reports on the overall developments within the project and targeting its use beyond the scope of the project.

1.4. Relation with other VR-Together activities

Related tasks are T3.1-T3.5, in which software components for the VR-Together platform are developed.

2. COMPONENTS OVERVIEW

While in this document the focus is on the development of individual components the link between components and explanations of the integrated software system can be found in the deliverable D2.7. The development of individual component of VRT WP3 follows four major tasks (Capture, Orchestration & Delivery, Encoding and Rendering). In this chapter we give an outline and short introduction to all components developed in the VRTogether project (see Table 1 for a list of all components) and its subcomponents. Each component is explained in more detail in the subsequent chapters of the document.

Table 1 List of Component

Component	Short description Section	Detailed description Section	Pipeline
Task 3.1 Capture			
3D Capture	2.1.1.1.	3.1.	Native pipeline
RGBD Capture	2.1.1.2.	3.2.	Web pipeline
Simple PC Capture	2.1.1.3.	3.3.	Native pipeline
Task 3.2 Encoding			
TVM Encoding & Transmission	2.1.2.1.	4.1.	Native pipeline
Point Cloud Coding	2.1.2.2.	4.2.	Native pipeline
Task 3.3 Orchestration & Delivery			
Media / Session Orchestrator	2.1.3.1.	5.1.	Native pipeline
RTMP Live Video Transmission GUB	2.1.3.2.	5.2.	Native pipeline (opt.)
DASH Sender - bin2dash	2.1.3.3.	5.3.	Native pipeline
DASH Distributor - evanescent	2.1.3.4.	5.4.	Native pipeline
DASH Receiver - SUB	2.1.3.5.	5.5.	Native pipeline
Live presenter	2.1.3.6.	5.6.	Native pipeline (opt.)
WEBRTC-VR-MCU	2.1.3.7.	5.7.	Web pipeline (opt.)
PC-MCU	2.1.3.8.	5.8.	Native pipeline (opt.)
2.1.4. Task 3.4 Rendering and Display			
Web-based player	2.1.4.1.	6.1.	Web pipeline
Unity based player	2.1.4.2.	6.2.	Native pipeline

2.1. List of components

This section summarizes the technology components described in this deliverable and its packing for integration and exploitation. Further each component states its current status of IPR, which is affected licensing due to the use of external (3rd party) libraries and potential IPR conditions to just the component in the future.

2.1.1. Task 3.1 Capture

2.1.1.1. 3D Capture

3D Capture is a component designed to orchestrate the capturing, streaming and recording of the data acquired from a multi-sensor infrastructure. It allows multi-stream synchronization and data-driven and global optimized volumetric alignment (calibration), in order to distribute live volumetric video in the form of point-clouds or time varying meshes (TVMs) to other applications via a broker. A client application can retrieve this 3D information by downloading the stream directly by the broker. Our native and Unity API offers real-time receiving and rendering of volumetric video data inside different virtual environments. We can distinguish the 3D capture into 2 modes:

1. **Point-based Capture:** CERTH/VCL's 3D capture component (Volumetric Capture, VolCap) provides real-time volumetric data acquired from a set of Intel RealSense v2.0 D415 series, Azure Kinect or a mix of those RGBD sensors. A number of processing units each manage and collect data from a single sensor using a headless application called Eye. A set of sensors is orchestrated by a centralized UI application, VolCap, that is also the delivery point of the connected sensor streams. As a result, a real-time fusion from multiple RGB-D streams constructs a 360o colorized point-cloud representation. VolCap offers an extended variety of real-time RGBD data parameterization from the user interface regarding the image's resolutions, compression parameters, sensor presets and so on. The 3D reconstructed point-based volumetric video is accessible from various applications via a broker.
2. **Mesh-based Capture:** A separate tool, called Volumetric Reconstruction (VolReco), consumes the 3D information along with multiple textures from the aforementioned broker and constructs a time varying mesh (TVM) self-representation in real-time. This particular software provides real-time tunable mesh shading, voxel grid resolution and bounding box for the generated mesh. The produced TVM is also accessible from various applications via the same broker.

CERTH/VCL's point-based capture software (VolCap) is open and publicly available, however multiple opportunities are given to raise economical exploitation and boost the process of creating a market-ready solution. Already multiple interested corporations, broadcasters and XR content creators have approached CERTH/VCL expressing interest in the volumetric capturing system. The exploitation plan of CERTH/VCL after the end of the project is to create an open source version of its software and follow a business plan that will create revenue opportunities, while taking advantage of the system's openness to obtain a constant stream of fast paced innovation. Often users of open-source technologies are willing to

purchase additional software features under proprietary licenses, or purchase other services or elements of value that complement the open-source software. Since CERTH/VCL is a non-profit organization, alternatives and ways that could make it possible to commercially exploit the system are discussed, possibly through D-cube, which is a CERTH's spin-off company that aims at producing and distributing innovative high technology products.

As far as the mesh-based capturing tool (VolReco) goes, it can be further developed after individual agreements. Regarding our market oriented plan, CERTH/VCL is open to new opportunities and possible synergies with companies, organizations and individuals. CERTH/VCL is frequently in contact with interested parties that want to learn more about the system and its specifications. Answering to those signs, CERTH/VCL is considering creating a license to further exploit the reconstruction tool. Currently, serious considerations about undertaking IP protection measures are being discussed as it is a prerequisite to commercially exploit the system.

2.1.1.2. RGBD Capture

The capture module is responsible for a photo-realistic capture of the user based on one RGBD sensor (i.e. Kinect / RealSense). Therefore the module has a flexible interface to connect to different RGBD drivers in order to get the colour image and depth image from the hardware sensor. The image is further processed (as of now this may include foreground-background removal and replacement of the HMD with a pre-captured representation of the human face) and finally converted to an 2D RGB + grayscale depth image. The final image is displayed on the screen for capture in the WEB browser. The camera calibration is sent to the browser for geometrically correct 3D rendering using WebGL shaders. The capture is rendered as 3D selfview for self-presence and as other view for the other participants for shared presence all in real-time. All modules are optimized for capturing and rendering users in real-time in a 3D virtual environment.

The RGBD sensor capture is written in C++ using the Kinect2.0-SDK (Microsoft End User Agreement) in combination with TNO's RDA C++ module (IP-owned, proprietary license). The processing modules are written in Python, the main dependencies are: Python (PSF licence) and OpenCV (BSD License).

2.1.1.3. Simple PC Capture Modules

The simple point cloud capture module captures RGB and Depth images from RGBD cameras attached to a single high-end workstation with USB 3.2 connections. The simple in the name we have given it applies to ease of deployment: this module is easier to deploy than the RGBD capturer from the previous section, with the tradeoff of requiring a more capable workstation (independent USB 3.2 ports for each camera, more capable CPU) than the RGBD capturer, which offloads part of the processing to the GPU and to independent small computers attached to each camera. The modules can technically support 8 cameras, but practically have only been deployed with 1-4 cameras.

The module comes in two flavors (and two experimental versions, see below) that are fully API-compatible: *cwipc_realsense2* supports Intel Realsense D400 cameras and *cwipc_kinect* supports Microsoft Azure Kinect cameras. The cameras are calibrated and aligned manually using an accompanying cwipc_align utility. The captured images are fused into a single point cloud in cwipc representation (see T3.2 below) that contains not only X, Y and Z coordinates and RGB data for each point but also information about the camera number from which the point was captured.

The modules have an API that is usable from different languages, such as C++, C#, C and Python. The module is compatible with Linux, MacOS (cwipc_realsense2 only) and Windows.

The modules will be made available under a permissive BSD-style or Apache-style open source license. External dependencies are the Intel librealsense library (Apache license, for cwipc_realsense2 only), the Microsoft Azure Kinect DK (MIT license, cwipc_kinect only) and the PCL point cloud library (BSD license).

Additional PC Capture Functionality

Two additional point cloud capturers are available in the *cwipc_util* module (described in the PC encoding and decoding section) with the same API as the Simple PC Capture modules, but these should be considered experimental (although both have been used in near-production scenarios during the project):

- The *cwipc_certh* module is a small layer around the RGBD capture module that allows selecting either one of the simple capturers or the RGBD capturer at runtime, by providing the same API,
- The *cwipc_proxy* module allows use of an RGBD camera that is connected to the workstation over the Internet (instead of directly connected through USB). There is an accompanying Android app that streams RGBD images over a 5G mobile connection or a WiFi connection. This allows capturing point clouds from mobile devices.

2.1.2. Task 3.2 Encoding

2.1.2.1. TVM encoding & transmission

Our software creates real-time dynamic 3D self-representations. As a result, encoding and transmission are of great importance, when exploiting an entirely dynamic asset, which is transmitted via the network. The software for real-time encoding and transmission of such dynamic assets is provided through its integration in the volumetric capture and reconstruction components. Any external application can receive and decode 3D representation data from our platform in real-time.

As far as the encoding goes, we are exploiting two state of the art native compression libraries. The supported 3D representations, being 3D fused colorized point-cloud and time-varying mesh, are encoded/decoded using Corto and GoogleDraco libraries respectively. This comes as a result of testing and benchmarking between different compression algorithms by CERTH (including OpenCTM among the aforementioned) and these choices take into account the pipeline's overall performance, as well as the bandwidth consumption. The encoding and

decoding modules are tunable and compatible with the different parametrization of the 3D data in both capture scenarios.

Regarding the transmission, by exploiting our platform's network infrastructure real-time streaming, we stream 3D data representations to a RabbitMQ server, serving as a broker, using the Amqp Cpp library.

As we mentioned above, this component has been developed using open-source libraries. Linking to licences of these libraries can be found below:

- Corto Cpp library (corto): https://github.com/cnr-isti-vclab/corto
- GoogleDraco Cpp library (draco): https://github.com/google/draco
- RabbitMQ Cpp library (amqpcpp): https://github.com/CopernicaMarketingSoftware/AMQP-CPP

2.1.2.2. Point Cloud (PC) encoding & decoding

The point cloud encoding and decoding module, *cwipc_codec*, compresses and linearizes point clouds in cwipc representation for transmission or storage and subsequent decompression back to cwipc representation. The module also has the option to reduce the number of points (LOD, level of detail) through voxelization (allowing tradeoff between quality and data size) and to split a point cloud into tiles based on contributing camera (allowing tradeoff between quality and orientation of the point cloud surface). The compressed data can be transmitted and received through the DASH Sender/Distributor/Receiver chain (see T3.3) or through other means.

The accompanying module *cwipc_util* implements the infrastructure for managing point cloud storage while the (rather voluminous!) point clouds are transferred between different modules and possibly different implementation languages, and for converting between point cloud data formats applicable to those modules while maintaining maximum efficiency and minimal copying. This module implements the cwipc format, which is shared between cwipc_codec, cwipc_realsense2 and cwipc_kinect. Included are utilities to convert point clouds between cwipc format and the industry-standard PLY format and to view streams of point clouds coming from capturers or over the network.

The modules have an API that is usable from different languages, such as C++, C#, C and Python. They are compatible with Linux, MacOS and Windows.

The modules will be made available under a permissive BSD-style or Apache-style open source license. External dependencies are the PCL point cloud library (BSD license) and libjpeg-turbo (BSD license).

2.1.3. Task 3.3 Orchestration & Delivery

2.1.3.1. Media/Session Orchestrator

Core features:

The VRT orchestrator, as a centralised server component, is primarily in charge of the management of a set of end-user profiles which are connected into a platform session. The orchestrator is able to control the various connection events in order to grant a unified experience and ensure that whole media pipelines are properly switched across the various connected nodes. It is also the reference point used for the synchronisation of the media pipelines. Then, from the backend viewpoint, the orchestrator is in charge of managing and supervising the transmission and the delivery of all of the media stream pipelines.

Regarding the management of the platform sessions, the orchestrator provides a set of exposed API to handle the multiple aspects of a collaborative and scripted experience. From the sessions and rooms dynamically built across scenarios, which are stored in dedicated databases, to the users' profiles which include specific rights roles and attributes, as well as time clocks reference point, the orchestrator framework provides a complete toolset to build and deploy a smart and unified collaborative platform.

Then the second role of the orchestrator is to manage and supervise the components in charge of the transmission and distribution of the media streams shared between end-users. The framework provides a dedicated exposed API to pilot and monitor the various components associated with the transmission of the media streams (on-demand content, live video, volumetric and audio streaming).

2.1.3.2. RTMP Live Video Transmission - GUB

GStreamer Unity Bridge (GUB) allows Unity Player to receive an RTMP stream content using GStreamer (LGPL) libraries. It handles the receiving module, grabbing the content from the RTMP url and, the decoding module that decodes this RTMP using GStreamer to do all this tasks and finally render an H264 video inside Unity Player.

2.1.3.3. (PC/Audio/Live) DASH Sender - bin2dash

This component allows to package any raw streams into standard MPEG-DASH over CMAF and then deliver to a server component (like an SFU or MCU). It can be directly integrated into the native client (which is producing raw streams for self-view). This component enables to handle all (encoded) legacy media like video, audio, subtitles, metadata (including timed metadata) but it has been adapted also to handle any kind of volumetric data from Point Clouds (V-PCC, G-PCC), Mesh/TVM, MPEG 3DoF+, or any personalized data. It offers capabilities included by DASH which are mainly scalability and low-latency.

This component is written in C++. It has dependencies on Motion Spell's Signals (C++, IP-owned, proprietary license), GPAC (C, LGPLv2+ license), cURL (C, MIT/X license)

2.1.3.4. (PC/Audio/Live) DASH Distributor - evanescent

Evanescent is a server component that allows to receive any kind of DASH stream and directly forward them to other receiver clients. In this sense, Evanescent is an SFU (Stream Forward Unit). More technically, it receives low latency dash chunks and serves them back to a low latency dash receiver using http chunked transfers. This component is low footprint in terms of CPU, memory and network overhead. It can be improved into an MCU (Media Composition Unit) that will allow adaptive filtering, transcoding, compression and remuxing of input streams (which much more CPU intensive).

This component is written in C++. It has no dependency.

2.1.3.5. (PC/Audio/Live) DASH Receiver - SUB

This component is the receiving end of the pipeline. It allows to unpackage any standard MPEG-DASH streams into raw streams to be rendered by the receiver client. It also contains preliminary algorithms with respect to tiling support. It has been extended to read files and RTMP streams for debugging and interoperability purposes. More technically, it receives segments from the DASH Distributor (or low latency DASH server) using HTTP chunked requests, and demuxers them to be presented to the receiver client (that is usually a player).

This component in written in C++. It has dependencies on Motion Spell's Signals (C++, IP-owned, proprietary license), GPAC (C, LGPLv2+ license), FFmpeg (C, LGPLv2+ license), cURL (C, MIT/X license).

2.1.3.6. (PC/Audio/Live) Live presenter

The Live Presenter component involves both the capturing, encapsulation and transmission of a stereoscopic video content (also considered as the Live recording component), and the rendering of the content in the VR-Together player including background removal feature.

The first step (capturing, encapsulation and transmission) is performed by the Live Presenter machine, through the Z-Cam Wonderlive application[1], that encapsulates and transmits the raw video stream through RTMP protocol to the centralized Transcoder service (i.e DashCastX).

The rendering is performed by the player after receiving the video encoded and encapsulated (H264, Dash), decoding it using the FFMpeg libraries[2] and cropping the content to facilitate the green screen-based background removal.

Overall, regarding the management of the Live presenter pipeline, the LRTS (aka Live-Recording-Test-Simulator) is in charge of managing the start and stop commands coming from the Core Orchestrator service to pilot the Dockerised transcoding service.

[1] https://www.z-cam.com/wonderlive/

[2] https://ffmpeg.org/

This component is written using Docker and bash scripts with the current dependencies: Motion Spell's Signals (C++, IP-owned, proprietary license), Motion Spell's Evanescent (C++, IP-owned, proprietary license), GPAC (C, LGPLv2+ license), FFmpeg (C, LGPLv2+ license), cURL (C, MIT/X license), nginx (C, 2-clause BSD license).

2.1.3.7. WEBRTC-VR-MCU

The WebRTC VR MCU system is capable of ingesting any number of WebRTC client inputs. The MCU system synchronises and composes its inputs into a single output stream which is then published using WebRTC. As a result, clients are able to retrieve all relevant streams together instead of separately. This optimizes the network bandwidth due to more efficient routing (each client only sends its video stream to the MCU, and no longer to all other clients), as well as the decoding resources (clients have a limited amount of hardware decoders, which can now be dedicated to decoding all streams instead of just one) of said clients.

Incoming WebRTC streams are demuxed into RTP streams and are fed into a pluggable media pipeline. The media pipeline performs the compositing operation and outputs a single RTP stream, which is broadcast using multiple WebRTC return streams. The media pipeline of the MCU system has been designed as a containerized service such that given enough hardware, it is horizontally scalable over multiple parallel sessions. These services are managed using Docker Swarm and the orchestrator component.

The WebRTC-MCU's main dependencies are: Janus (GPLv3), gstreamer (LGPL) Python (PSF licence), NodeJS (MIT), Docker Swarm (Apache 2.0).

2.1.3.8. PC-MCU

The PC-MCU is a tool able to perform a set of volumetric video processing routines, to deliver the best possible representation of the content with the minimum amount of data to each user according specific session parameters like position (distance) of users in the 3D environment, bandwidth and computing capacity.

The PC-MCU will provide the following features:

- Level of Detail (LOD) selection for the volumetric video
- Occluded areas removal
- Fusion of two, or more volumetric videos

I2CAT will publish the PC-MCU application as open source code, released under LGPL licensing. To modify and work with PCs, the MCU uses PCL point cloud library (BSD license).

2.1.4. Task 3.4 Rendering and Display

2.1.4.1. Web-based player

The entry point for the TogetherVR Web client is offered by the web server back-end, which is based on Node.js, React and Aframe. This allows any modern WebVR enabled browser to display the VR content on a screen or an open VR hardware enabled VR-HMD (e.g. Oculus Rift cv1). Further the client can access the image produced from the TNO RGBD Capture module to be displayed as self-view to the user itself or to be sent via WebRTC to one or multiple other clients. In order to support such a multiuser connection, the client is connected to a second WEB server that is handling all sorts of multimedia orchestration. The rendering of users in the VR environment is done via custom WebGL shaders that alpha-blend user representations into the surroundings for a natural visual representation. Audio is also captured, transmitted and made auditable as spatial audio with the help of the Google Resonance APIs.

The full list of dependencies is rather long and might vary on different features being enabled or disabled in the different modules of the pipeline. This is we have a highly modalized platform that can be further adapted to specific productization needs. As of now the list of dependencies covers fall under the following licenses: Apache-2.0, BSD-2-Clause, CC-BY-3.0, CC0-1.0, ISC, MIT.

2.1.4.2 Unity based player

The Unity Player is the final element of the pipeline, and it is in charge of integrating and merging all the components mentioned above. It enables the player to display the VR content on a screen or an open VR hardware enabled VR-HMD (e.g. Oculus Rift, HTC Vive, Oculus Quest). The player can receive the user representations from the TVM, or PC capturing system or a webcam and render them in the three-dimensional scenario. Moreover, the player can send the audio captured from the HMD to the other clients and consequently receive and reproduce the audio of the rest of clients. During the development of this Unity based player, several tasks and issues have been solved in order to present the final version of this Unity based player:

- Creation of a multitasking system used by all components of the project
- Creation of a queuing system to distribute information between tasks.
- Access system to C # components (necessary to access the components that had been developed, the sub, the bin2dash, CWI libraries, etc.). This was modified according to the needs of the club
- Encoding and decoding of webcam audio and video
- Shaders, materials, pilot flow programming
- Modification of the socket.io libraries for multitask support
- Reading and playing webcam audio and video (independent tasks)

The full list of dependencies is a combination of almost all the other components and depends on which of them are used on every specific case and pipeline. So, the player is a highly modular platform that can be further adapted to specifically productize the technology. The fixed list of dependencies covers the following licenses: Apache, BSD, C, LGPL, LGPLv2+.

2.2 Component Progress Year 0 – 3

In Table 1 we outline the development of technical components through the lifetime of the project from the start of the project in year 0, to their final status the end of year 3. Table 2, shows the delta of different features and improvements in the different stages of the project.

EU definition of TRL: https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf

Table 2 Component Progress Delta Table

Component	Status Year 0 (M0)	Status Year 1 (M12)	Status Year 2 (M24)	Year 3 (M36) Final outcome of Project
Task 3.1 Capture				
3D Capture (CERTH)	TRL5: Existing multi-view 3D Capturing and Reconstruction platform (with KinectV2) 4 KinectV2 sensors: - Color res: 1280x720 - Depth res: 512x424 Reconstruction rate: 5fps	TRL6: Updates on the existing multi-view 3D Capturing and Reconstruction platform (with KinectV2) for VRT end-product, supporting skeleton tracking and performance improvements. 4 KinectV2 sensors: • Color res: 1280x720 • Depth res: 320x180 Reconstruction rate: 9fps	TRL7: Integrated synchronization and streaming utilities for 3D data in the new 3D Capturing Platform (with Intel RealSense). TRL7: New Reconstruction software (with Intel RealSense) 4 Intel Realsense D415 sensors: • Color res: 1280x720 • Depth res: 320x180 Reconstruction rate: 22 fps	Support of Azure Kinect. TRL7: 3D Capturing Platform supports both Azure Kinect and Intel RealSense RGBD sensors. TRL7: Reconstruction pipeline supports both Azure Kinect and Intel RealSense RGBD sensors. Performance optimizations based on the reconstruction rate, targeting real-time performance. Performance optimizations are related with the increased bandwidth needs and the 3D data

				processing due to the increased depth resolution.
RGBD Capture (TNO)	Overall Component TRL5: TRL5: RGB with Chroma background removal. TRL2: One KinectV2 as RGBD capture device with depth-based background removal. TRL4: 3D shaders to real-time reconstruct RGBD to true scale rendering in browser-based VR.	Overall Component TRL6: TRL6: RGB with Chroma background removal. Lowered processing CPU (i7) usage from 20% to 15%. TRL6: Integrated KinectV2 and RealSense as RGBD devices. TRL6: 3D shaders for self-view and other-view capture and rendering. TRL4: Sensor calibration of Oculus Rift and KinectV2 for RGBD-based HMD removal.	Overall Component TRL6: TRL5: HMD removal fully integrated in browser-based VR and Unity-VR for user experiments. Overall performance CPU usage: Kinect v2 capture 13.2%, Rift capture 1.0% CPU, processing 14.0%. Capture to screen delay ~150ms. Overall capture (non HMD removal) processing improvement from 15% CPU usage to 10% CPU usage. Capture to screen delay ~140ms.	TRL6: Integrated one executable capture with different modes (Kinect v2, RealSense, Azure, HMD replacement, fgbg) and toggle to switch between different modes (i.e. 2D and 3D capture mode).
Simple Point Cloud Capture (CWI)	No TRL at M0. The component was originally not planned.	TRL4: demonstrated in the lab during the review. Point Clouds could be captured from a single RealSense RGBD camera at about 5fps.	TRL5: Point Clouds could be captured from multiple cameras and aligned and fused. Capture frame rate increased to above 25fps for 1 camera, 15fps for 4 cameras. Integration into operational pipeline. End-to-end latency (from camera on sender to HMD on receiver) decreased to approximately 400ms. Self-	TRL7: Point Clouds are captured and assigned relevant metadata to allow for user adaptive streaming using tiles. Based on testing, the exact target frame rate and quality level will have to be decided (finding a tradeoff between latency, visual quality and resource consumption). The

			view latency (from camera to own HMD) decreased to approximately 300ms.	target is that the pipeline will be able to provide either around 20-25fps or a visual quality that is 4 out of 5 on "no perceptible difference with uncompressed self-view", but it is explicitly not a target to provide both of those at the same time.
Task 3.2 Encoding				
TVM encoding & transmission (CERTH)	TRL5: OpenCTM encoding and simple RMQ transmission in existing multi-view 3D platform *OpenCTM profile, 64x128x64*: - Encoding: ~65ms TVM Bandwidth: ~5Mbps TVM pipeline end-to-end delay: ~225ms	TRL7: Integration of SoA algorithms (Corto, Draco) for mesh compression. *libdraco profile,* 64x128x64: - Encoding:~17ms *libcorto profile,* 64x128x64: - Encoding:~10ms TVM Bandwidth: ~2Mbps TVM pipeline end-to-end delay: ~183ms	TRL7: Integration of mesh compression new features / updates and optimized RMQ using efficient task management (RxCpp) to handle messages with the most appropriate way for TVM transmission *libdraco profile*, 64x128x64: - Encoding: ~15ms *libcorto profile*, 64x128x64: - Encoding: ~1ms TVM Bandwidth: ~3Mbps TVM pipeline end-to-end delay: 55ms	TRL7: Integration of mesh compression new features / updates and optimized RMQ using efficient task management (RxCpp) to handle messages with the most appropriate way for TVM transmission

Point Cloud (PC) encoding & decoding (CWI)	TRL6**:** Core of encoder and decoder existed. Performance numbers are available in *Mekuria et al DOI=10.1109/TCSVT.2016.2543039,* but they are not that useful for comparison purposes since they used an old and different dataset. Encode times for the Dimitris-2-Zippering Sequence (~320k points) using JPEG compression for the color attributes: OctreeDepth 8 (~120k points): ~80 ms Octree Depth 9 (~260k points): ~160 ms Octree Depth 10 (~300k points): ~260ms Octree Depth 11 (~310k points): ~380ms	TRL6: Ad-hoc integration of encoder and decoder into Gstreamer-based delivery pipeline.	TRL7: integration to operational pipeline. Optimization of memory use. Latencies decreased to usable levels. No formal before/after measurements have been taken at the time, but memory consumption is approximately halved for both encoder and decoder, and encoder latency has also approximately halved. Together with the simple capturer, improvements the end-to-end point cloud pipeline has gone from 1 camera at 5fps to 4 cameras at 15fps.	TRL7: Coding module is integrated into the operational pipeline. The module is optimized for multiple encoding to create a tiled Point Cloud adaptation set. The Encoder is parallelized to enable using multiple quality levels and multiple tiles without an increase in end-to-end latency (assuming enough CPU cores are available). Unfortunately only minimal performance testing has been done during the project.
Task 3.3 Orchestration & Delivery				
Media/Session Orchestrator (MS / VO)	No TRL at M0 (TRL0): Non existing component at initial time.	Overall Component TRL2: ● 2 users managed statically ● Static Platform configuration	Overall Component TRL4: ● 4 users per session managed dynamically	Overall Component TRL6: ● 10 users per session managed dynamically ● PC & TVM pipelines supervision

			• PC & TVM pipelines supervision • Live presenter supervision	• Live presenter supervision • Interaction manager Virtualisation pre-packaging
RTMP Live Video Transmission - GUB (I2CAT)	TRL6 (Component developed in EU H2020 ImmersiaTV project) Ingest of DASH and RTMP streams (or other media URIs) from GStreamer into Unity KPI: • Used for stored DASH streams • Smooth playout • No delays tests conducted so far	TRL6: Ingest of DASH and RTMP streams (or other media URIs) from GStreamer into Unity KPI: • Used for stored DASH streams • Smooth playout • No delays tests conducted so far	TRL6: Same features. Updating the component to interact with newer versions of GStreamer. KPI: • End-to-End latency for a live RTMP stream: ~1.5s	TRL6: Same features. KPI: End-to-End latency for a live RTMP stream: ~1.5s
(PC/Audio/Live) DASH Sender - bin2dash (MS)	TRL4: Integrated component (plugged with capture and encoding) Limitation: 1 visual quality + 1 audio (to be provided using the same interface) Latency: 2 frames + internal threading	TRL6: New modular structure and new Metadata handling Audio and visual content can be provided separately to match with hardware capture needs. Latency: 1 frame + internal threading	TRL7: Improved with more data types, more testing use-cases Unlimited number of streams. However a high number increases concurrency on the system with a risk of OS kernel saturation (currently measured at 64 streams on Windows and 256 on Linux). Latency: 1 frame (a complete frame is needed for copy)	TRL7: Tiled Point Clouds, and Support for TVMs. Tiling to use the MPEG-DASH SRD standard descriptors. No limitation on the number of streams or tiles. However a high number increases concurrency on the system with a risk of OS Kernel saturation to be measured and optimized accordingly. Latency: 1 frame (a complete frame is needed for copy).

(PC/Audio/Live) DASH Distributor - evanescent (MS)	No TRL at M0 (TRL0): Absent component at initial time	TRL4: Basic system with HTTP and HTTPS/SSL Latency: 1 HTTP chunk request	TRL5: SFU testing with the associate components Latency: 1 HTTP chunk request Scaling: Evanescent can take a lot of CPU and saturate the kernel when handling more than 30 streams.	TRL6: SFU testing with the associated components Latency: 1 HTTP chunk request Scaling: improve to more than 30 streams.
(PC/Audio/Live) DASH Receiver - SUB (MS)	TRL4: Basic DASH reception **Latency:** 2 ISOBMFF/MP4 fragments	TRL5: RTMP input **Latency:** 1 ISOBMFF/MP4 fragment or 1 frame (RTMP)+network delays	TRL6: Support for other streamers (e.g. not only bin2dash) Latency (est.): 1 ISOBMFF/MP4 fragment or 1 frame (RTMP)+network delays	TRL6: Support for tiling and TVM. Switching delay: one frame + one fragment (usually one frame) + one segment (usually 2-4 seconds). Latency (est.): 1 ISOBMFF/MP4 fragment or 1 frame (RTMP) + network delays
(PC/Audio/Live) Live presenter (MS)	No TRL at M0 (TRL0): Absent component at initial time	Overall Component TRL5: ● RTMP input (audio+video) ● Cropping ● DASH output Latency: 2 frames + 1 GOP (approximately 1-2s depending on the capture material)	Overall Component TRL6: Container 1 (transcoder mode): ● RTMP input ● Cropping (now optional) ● RTMP or DASH output Container 2 (RTMP relay mode):	Overall Component TRL6: No core modifications expected. This module heavily depends on the capture material capabilities.

			• RTMP input • Cropping (now optional) • RTMP output **Latency:** 2 frames + 1 GOP when activating cropping (approximately 1-2s depending on the capture material) + "1 frame + 50ms" when transcoding *NOTE: An experiment to determine the delays in operational (cross-national) environments is planned*	
WEBRTC-VR-MCU (TNO)	Overall Component TRL2: • Only baseline code • No MCU (Full P2P mesh) *NOTE: An experiment to determine the delays is planned*	First code version of Component, TRL3: Component Integration surface	Overall Component TRL5: • Horizontally scalable MCU • Orchestration • Integration with web Player • Number of users: 8 *NOTE: An experiment to determine the delays is planned*	Overall TRL 7: • Number of users 16 • Fully dockerized deployment • Support for low-powered (mobile) devices
PC-MCU (I2CAT)	No TRL at M0: Absent component at initial time	Overall Component TRL0: • Analysis of the SoA in that area (e.g. traditional	Overall Component TRL1: • Initial proof of concept	Overall Component TRL3: <u>Phase 1 (Start of Year 3)</u> • Integration with PC

		MCUs for 2D conferencing, viewport aware streaming ...) ● Preliminary core implementation		pipeline ● Sequential implementation ● Multiple Volumetric Video de/coding ● Reception and delivery of multiple streams ● Level of Detail (LOD) adjustment ● Removal of non visible volumetric video ● Fusion of volumetric videos ● Parallel Operations KPIs: ● CPU saving for the client: >50% ● Bandwidth saving for the client: >50% ● Users supported in a Holoconferencing session: 4+ ● Supported Resolution per user: Up to 50k voxels ● Latency: <500 ms ● PC-MCU Core functionalities KPIs: ● Fusion: 4+ users supported in real time ● LOD Selection: 3+

				resolutions supported
T3.4 Rendering and Display				
Web-based player (TNO)	Overall Component TRL6: ● 360-degree only ● P2P streaming	Overall Component TRL7: ● Pilot 1 ● 3D volumetric environment and users ● Streaming via MCU ● Self-view ● Spatial audio ● Admin panel	Overall Component TRL7: ● Refactored client ● Pilot 2 ● Optimized MCU pipeline ● improved 3D volumetric user shader to display users as mesh or point cloud rendering ● Use capture (hardware) parameters for improved rendering	TRL7: ● Integrated technical measurements (profiling of web client performance and WebRTC stats) ● New virtual room(s) with a focus on communication ● Support for low-powered (mobile) devices
Unity player (I2CAT)	No TRL at M0 (TRL0): Absent component at initial time N/A	Overall Component TRL6: ● TVM 1.0 Rendering ● 2 Users support ● P2P streaming	Overall Component TRL7: ● TVM 2.0 Rendering ● 4 Users support ● Orchestrator based streaming ● Stereoscopic video live stream ● Multi-threaded pipelines ● GUB Integration	Overall Component TRL7: ● 5+ Users support ● SUB Integration ● Socket.io data streams ● PC pipeline ● WebCam pipeline ● Spectator mode ● PC-MCU Integration ● Multi-threading improvements to enable multiple encoding/decoding ● Networked interactions ● Voice commands

3. SOFTWARE FOR TASK 3.1 CAPTURE

In this chapter, we are going to describe the software related to the capturing process that was created and improved during the development of the VRTogether project and its platform. The term capturing refers to the acquisition of data (3D in this case), as well as their processing and serialization in order for them to be transmitted or used by other applications. Following the acquisition, which is frequently achieved by another service or application and based on the sensor's software development kit, 3D data have to be volumetrically aligned depending on the sensor's specifics, a procedure that is called calibration. The processed data can be either directly used or, firstly, serialized in order to become the input for another application, often after being transmitted via a broker. Below, we are going to describe the capturing pipelines that were created with respect to the needs of this project and exhibit their evolution throughout the years, along with their dependencies and features.

3.1. 3D CERTH Capture

Description:

Volumetric Capture (VolCap) is a toolset designed to orchestrate the capturing, streaming and recording of the data acquired from a multi-sensor infrastructure. A number of processing units each manage and collect data from a single sensor, being either an Intel RealSense 2.0 D415 or a Azure Kinect, using a headless application called Eye. This set of sensors is orchestrated by a centralized UI application, VolCap, that is also the delivery point of the connected sensor streams. VolCap allows multi-stream synchronization and data-driven and global optimized volumetric alignment (calibration), in order for the volumetric data and their respective metadata to be finally streamed through a RabbitMQ server, serving as a broker. These data streams can be obtained and used by external tools to construct point-based volumetric representations called point-clouds. VolCap also offers an extended variety of real-time RGBD data parameterization from the user interface regarding the images' resolutions, compression parameters, sensor presets and so on.

The Volumetric Reconstruction (VolReco) tool can, also, receive the aforementioned data and metadata in order to locally reconstruct 3D meshes (time varying meshes, TVMs) and, eventually, encode and transmit them through a RabbitMQ server, serving as a broker. Furthermore, it allows real-time tunable mesh shading, voxel grid resolution and bounding box for the generated mesh. It also provides a timestamp for each TVM in order to calculate the end-to-end delay.

Links:

- Capturing: https://github.com/VCL3D/VolumetricCapture
- Reconstruction: http://vcl.iti.gr/vr-together-eu/guides/Volumetric_Reconstruction_Documentation_v2.pdf

Evolution:

Table 3. Detailed Delta Table - 3D Capture

	Status Year 0	**Status Year 1**	**Status Year 2**	**Status Year 3**
TRL	TRL5: Existing multi-view 3D Capturing and Reconstruction platform (with KinectV2)	TRL6: Updates on the existing multi-view 3D Capturing and Reconstruction platform (with KinectV2) for VRT end- product for performance improvements	TRL7: Integrated synchronization and streaming utilities for 3D data in the new 3D Capturing Platform (with Intel RealSense). TRL7: New Reconstruction software (with Intel RealSense)	TRL7: Integration of new Azure Kinect 4 Azure RGBD Sensors in new 3D Capturing Platform. Capturing Platform supports both Azure Kinect and Intel RealSense RGBD sensors. TRL7: Reconstruction pipeline with new Azure Kinect sensors. Reconstruction pipeline supports both Azure Kinect and Intel RealSense RGBD sensors. Performance optimizations based on the reconstruction rate, targeting real-time performance. Performance optimizations are related with the increased bandwidth needs and the 3D data processing due to the increased depth resolution.
Features	Use of 4 KinectV2 sensors with: - Color resolution: 1280x720 - Depth resolution: 512x424	Use of 4 KinectV2 sensors with: - Color resolution: 1280x720 - Depth resolution: 512x424 Skeleton tracking	Use of 4 IntelRealsense D415 sensors with: - Color resolution: 1280x720 - Depth resolution: 320x180	Use of 4 Azure Kinect sensors: - Color resolution: 1280x720 - Depth resolution: 320x288
KPIs	Reconstruction rate: 5fps	Reconstruction rate: 9fps	Reconstruction rate: 22 fps	Reconstruction rate: 25 fps

Features:

- Headless acquisition application's (Eye) main feature is:
  - management and collection of data from a single RGBD sensor.
- VolCap 's features include:
  - data collection from multiple Eye applications;
  - multi-stream synchronization;
  - data-driven and global optimized volumetric alignment (calibration);
  - data recording;
  - control over rgb and depth resolution as well as the compression rate of the rgb data;
  - control over other camera and capturing related parameters (further details can be found in the provided documentation);
  - streaming volumetric data and the respective metadata through a RabbitMQ server, serving as a broker.
- VolReco's features include:
  - reception of volumetric data and metadata provided by VolCap;
  - reconstruction of time-varying meshes (TVMs) locally;
  - data compression and transmission through a RabbitMq server, serving as a broker;
  - timestamp for end-to-end delay calculation, real-time tunable mesh shading, voxel grid resolution and bounding box for the generated mesh.

Performance:

- Volumetric Capture (VolCap) and headless data acquisition application (Eye):
  - Capturing rate at 30 FPS (equal to sensors' streaming)
  - Synchronization duration lower than 3 seconds
  - Calibration (with 4 or more device) duration lower than 1 min
  - Resources
    - RAM: ~400 MBs
    - CPU (Intel i7): ~25%
    - Bandwidth: ~20 MBps (per device, capture client to central PC)
- Volumetric Reconstruction (VolReco):
  - Reconstruction rate at 27-29 fps (depending on the sensors' synchronization)
  - Resources:
    - RAM: ~400 MBs
    - CPU (Intel i7): ~17%
    - GPU (Nvidia GeForce 1070): ~25%
    - Bandwidth: ~85 MBps (middle color resolution)

Licensing:

- Volumetric Capture (VolCap) and headless data acquisition (Eye) applications:
  - Apache License, Version 2.0: https://www.apache.org/licenses/LICENSE-2.0.txt
  - Blosc Cpp library (c-blosc): https://github.com/Blosc/c-blosc/blob/master/LICENSES/BLOSC.txt
  - Boost Cpp library (boost): https://www.boost.org/users/license.html
  - High Performance Serialization Cpp library (hps): https://github.com/jl2922/hps/blob/master/LICENSE
  - RabbitMQ Cpp library (amqpcpp): https://github.com/CopernicaMarketingSoftware/AMQP-CPP/blob/master/LICENSE
  - Reactive Extensions (ReactiveX) Cpp library (RxCpp): https://github.com/ReactiveX/RxCpp/blob/master/license.md
  - Turbojpeg Cpp library (libjpeg-turbo): https://github.com/libjpeg-turbo/libjpeg-turbo/blob/master/LICENSE.md
- Volumetric Capture (VolCap) application:
  - Corto Cpp library (corto): https://www.gnu.org/licenses/gpl-3.0.html
  - SIMD Cpp library (libsimdpp): https://github.com/p12tic/libsimdpp/blob/master/LICENSE_1_0.txt
- Headless data acquisition application (Eye):
  - Intel RealSense SDK license: https://software.intel.com/content/dam/develop/external/us/en/documents/intel-realsense-sdk-license-843462.pdf
  - Kinect SDK v2 License: https://download.microsoft.com/download/0/D/C/0DC5308E-36A7-4DCD-B299-B01CDFC8E345/Kinect-SDK2.0-EULA_en-US.pdf
- Volumetric Reconstruction (VolReco) application:
  - CUDA license: http://docs.nvidia.com/cuda/eula/index.html
  - Blosc Cpp library (c-blosc): https://github.com/Blosc/c-blosc/blob/master/LICENSES/BLOSC.txt
  - Boost Cpp library (boost): http://www.boost.org/LICENSE_1_0.txt
  - Flann Cpp library (flann): https://github.com/mariusmuja/flann/blob/master/COPYING
  - GoogleDraco Cpp library (draco): https://github.com/google/draco/blob/master/LICENSE
  - High Performance Serialization Cpp library (hps): https://github.com/jl2922/hps/blob/master/LICENSE
  - MIT License (rendering): https://opensource.org/licenses/MIT
  - OpenCV Cpp library (opencv): https://opencv.org/license.html
  - RabbitMQ Cpp library (amqpcpp): https://github.com/CopernicaMarketingSoftware/AMQP-CPP/blob/master/LICENSE
  - Reactive Extensions (ReactiveX) cpp library (RxCpp): https://github.com/ReactiveX/RxCpp/blob/master/license.md
  - Turbojpeg Cpp library (libjpeg-turbo): https://github.com/libjpeg-turbo/libjpeg-turbo/blob/master/LICENSE.md

Deployment Documentation:

- **Source Code Location:**

  VolCap's source code is not provided, but its binaries can be found under https://github.com/VCL3D/VolumetricCapture/releases. VolReco is closed source and its binaries were only provided to VRTOGETHER's partners.

- **Installation instructions:**
  VolCap and VolReco are provided through their binaries and, as such, there is no installation needed.
- **External dependencies:**
  None
- **Deployment steps:**

  Regarding the deployment, for each sensor processing unit:

  1. Create the directory: C:\Capturer.
  2. When you have downloaded the remote.zip file from the release, extract its content in C:\Capturer.
  3. You should have a \bin folder which comprises the Eye application executables and in \remote_eye_service the service files.
  4. To install the Monitor service, you should run the install_remote_eye_service.bat in C:\Capturer\remote_eye_service\ with Administrator rights (right-click and choose "Run as administrator").
  5. If you ever need to uninstall the Monitor service, again you must run uninstall_remote_eye_service.bat with Administrator rights.

  To achieve automatic sensor connection between the VolCap and Eye applications, we use the Monitor service. This service runs in the background of the sensor processing unit, and listens to UDP port 11234 in the network's broadcast channel. Thus, you have to follow the steps above for creating inbound and outbound Windows Firewall rules, for all the processing units (workstation and sensor) for UDP port 11234. Then, a new service called remote_eye_service is going to be automatically run. Another machine, working as the central machine acquiring the data provided by the installed services, has to be used to run the VolCap application. For the application to be used, the steps below must be followed:

  1. Download the latest release of Volumetric Capture from the Releases section.
  2. Extract the main.zip file in a directory of your choosing (e.g. C:/). Avoid using paths that include "spaces".
  3. Create a shortcut of volcap.exe on your desktop if you want.
  4. If RabbitMQ has been installed on the same machine, just double click the volcap.exe (or the shortcut if you created one), and you are ready to go.

  The former steps can be also found in VolCap's documentation (Chapter 3, Paragraph 3.1, Links section). Assuming CERTH has given the license and provided the binaries, the Volumetric Reconstruction tool's deployment is pretty straightforward, as the user just has to run the application.

API (function(input) -> output: description):

For VolCap, the input comes from a number of processing units which collect data from RGBD sensors through a headless application (Eye) and transmit those to a corresponding RabbitMQ server. The output is a set of synchronized RGB frames along with volumetrically aligned geometry data and their corresponding metadata, which are compressed and transmitted through a RabbitMQ server, serving as a broker.

VolReco receives the former data and metadata as input in order to produce locally a volumetric 3D mesh (time varying mesh, TVM), which is, eventually, transmitted through a RabbitMQ server, serving as a broker.

3.2. RGBD Capture

Description:

The main aim of the RGBD Capture module is to provide a lightweight, simple to setup sensing solution that is easy to deploy (e.g. in peoples homes). Thus the capture module is responsible for a photo-realistic capture of the user, based on a single RGBD sensor (i.e. Kinect / RealSense). The module has an adaptable interface to connect to different RGBD drivers in order to get the colour image and depth image from the hardware sensor. As of now this may include foreground+background removal and replacement of the HMD with a pre-captured representation of the human face (HMD removal). The image is further processed and finally converted to a 2D RGB + grayscale depth image. The final image is displayed on the screen for capture in the WEB browser. The camera calibration is sent to the browser for geometrically correct 3D rendering using WebGL shaders. The capture is rendered as:

- 3D self-view for self-presence and
- view for the other participants for shared presence

all in real-time. All modules are optimized for capturing and rendering users in real-time in a 3D virtual environment.

Links:

- Capture and Transmit RGBD Data for 3D rendering (paper): https://dl.acm.org/doi/10.1145/3204949.3208115
- HMD removal (video): https://www.youtube.com/watch?v=fwhJKJ_Nm4E

Evolution:

Table 4. Detailed Delta Table - RGBD Capture

	Status Year 0	**Status Year 1**	**Status Year 2**	**Status Year 3**
TRL	5	6	6	7
Features	- RGB with Chroma background removal. - support of KinectV2 - shaders to	- improved background removal - added support for RealSense sensor	- HMD removal fully integrated - improvements on capture conversion performance	- added support for Kinect4Azure - one executable with different modes (i.e. 2D

	real-time reconstruct & render in browser-based VR	- 3D shaders for self-view and user rendering. - Sensor calibration of Oculus Rift and KinectV2 for RGBD-based HMD removal		and 3D capture mode)
KPI	n/a	n/a	n/a	See below

Features:

- **Different depth camera support (Kinect v2, Realsense, Azure Kinect)**
- **Video Background Removal:** Based on the depth and RGB raw data from the capture sensor the user is "cut out" from his / her background to allow only to transmit the relevant user information as well as to be able to blend users into the the virtual environment while rendering
- **HMD Replacement:** The HMD Replacement module consists of an open source available ArUco marker detection implemented using python-opencv applied to the RGB-D. When the marker is attached to the HMD of the subject, the HMD can be detected in real-time and replaced with a pre-recorded face image of the user.
- **Different output representations (2D with Chroma background, 3D RGBD):** The final capture module allows to switch in runtime between two types of representation to render users as 2D sprites (alpha blending users into the virtual environment by rendering the background transparent) and 3D volumetric (as point cloud or mesh).
- **Extended metadata and calibration**
- **Upload only client**
    - **Allows stand-alone capture to support untethered clients**
    - **directly uploads video stream to WEB-MCU**
    - **full control of bitrate / quality settings**

Performance / KPI:

In the following we outline the final performance of the RGBD capture module under the different supported sensor configurations. Figure 1, shows the setup of the performance measurements we conducted to evaluate the performance. CPU and GPU was measured with the Resources Consumption Metrics (RCM) measurement tool (https://github.com/ETSE-UV/RCM-UV) in sessions of 15 minutes. Delays were measured with VideoLat (https://videolat.org/) with at least 1000 samples. Figure 2, shows the final performance values of the RGBD capture module. With an overall CPU usage of max. ~30% we achieve a reasonable overhead to run the capture together with the web client on one end device. Further it is to mention that the processing difference between Kinect V2 / Realsense and Azure Kinect can be explained with different versions of the FGBG removal functions. The newer and simplified background removal manages to reduce the CPU load significantly. Further with an overall max. delay of ~342ms we in a range suitable for real-time communication.

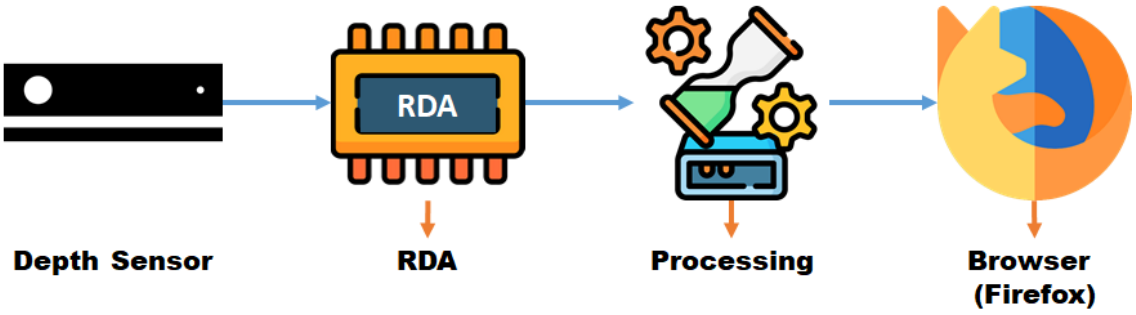

Figure 1. RGBD Capture performance measurement setup

Capture Sensor Setup	Capture (CPU)/(GPU)	Processing (CPU)/(GPU)	RDA Delay	Processing Delay	Full Delay
Kinect v2	C 10.54% / G 6.63%	C 19.08%	78.72 (SD 15.87)	138.19 (SD 18.71)	342.01 (SD 39.31)
RealSense D415	C 3.93%	C 21.81%	79.54 (SD 15.79)	152.18 (SD 18.89)	190.16 (SD 19.56)
Kinect Azure	C 4.43% / G 4.36 %	C 9.64%	112.29 (SD 15.51)	155.99 (SD 16.90)	261.74 (SD 24.92)

Figure 2. RGBD Capture performance results

Licensing:

- tno-capture (including hmd-removal): IP-owned, proprietary License
- RDA (TNO, Remote Data Access): IP-owned, proprietary License

The RGBD Capture software follows a proprietary license that can be made available to consortium members or to any customer on an ad-hoc basis. Further information and contact details are provided here:

- https://www.tno.nl/en/focus-areas/information-communication-technology/roadmaps/fast-open-infrastructures/social-xr-extended-reality/

Source Code / executable Location:

- https://ci.tno.nl/gitlab/vrtogether/RDA-Kinect-setup/-/tree/master/ERPCapture

Deployment Documentation:

To explain the technical functionality of the RGBD capture we can break this module into 3 sub-modules:

a. Sensor capture: The first capture module does real-time scene and user capture by reading the raw RGBD data from the capture sensor and mapping the colour and depth images to a shared memory location on a client PC.
b. Background Removal Processing: The raw RGBD sensor needs some further processing in a second capture module. The main aim of this module is to improve the image quality and perform real-time foreground-background (FGBG) extraction using colour and depth images from shared memory.
c. HMD Replacement: this module will detect an VR HMD with the help of a visible marker attached to the HMD. This image detection in the RGB stream is combined with the depth image to acquire an accurate 3D position and orientation of the HMD. With the position of the HMD the 2D image of the HMD will be replaced with an re-captured representation of the user's face (both on the RGB and depth data).

The capture module is deployed as a one click executable including all dependencies (besides the Sensor SDK and RDA that needs to be installed prior to execution.).

A detailed explanation of installing the source code is described here:

- https://ci.tno.nl/gitlab/vrtogether/RDA-Kinect-setup/-/blob/master/README.pdf

Dependencies:

Depending on the version of the capture and capture sensor used different dependencies are needed.

- Minimal requirement (for binary and source code execution)
  - Sensor SDK (Kinect v2, Realsense or Azure Kinect)
  - RDA (TNO, Remote Data Access)
- Requirements for source code execution only
  - Python 3
  - Python dependencies:
    - opencv-contrib-python>=4.2.0.34,<4.3
    - scipy>=1.4.1,<1.5
    - numpy>=1.18,<2
    - psutil>=5.7,<6
    - pywin32==227
    - tornado>=6.0.4,<7
    - matplotlib

3.3. Simple PC Capture

Description:

This component offers live capture and reconstruction of 3D point clouds using IntelRealSense D400 devices and/or Azure Kinect devices. The component can run with zero or more sensors. If no sensors are found a synthetic point cloud is generated, if multiple sensors are found the point clouds from each sensor are transformed and merged together.

Links:

- **https://github.com/cwi-dis/VRTogether-capture**

Evolution:

During year 3 it became clear that integration of the RGBD capturer (section 3.2) with sufficient visual quality and performance would be challenging, so more effort was spent on the simple capture component. Visual quality was increased by adding support for Azure Kinect cameras, by optimizing the fusion of RGBD data from multiple cameras and generally by optimizing performance and quality through code review. Deployment of the capturer to single-camera systems has been streamlined.

Table 5. Detailed Delta Table - Simple PC Capture

	Status Year 0	**Status Year 1**	**Status Year 2**	**Status Year 3**
TRL	N/A The component was not originally planned	TRL4: Prototype implementation using a single sensor	TRL5: Integration to operational pipeline. Optimization of memory use	TRL7: Optimization for viewport adaptive streaming
Features		Support for one Intel RealSense D400 sensor Synthetic point cloud is generated if no sensors are detected	Support for zero or more sensors. Reference Implementation for testing with 4 sensors	Point clouds captured will be segmented into spatial tiles along with relevant metadata Optimization of the trade-off between latency, visual quality and resource consumption (experimental) Support for Azure Kinect added
KPIs		Together with the compression module end to end pipeline is able to operate at 5 fps with 1 camera on commodity hardware	Together with the compression module end to end pipeline is able to operate at 15fps with 4 cameras on commodity hardware Self view latency of 300ms	Formally: same as year 2, but now also supporting Azure Kinect devices (the other new features implemented this year have not been tested in a deployment situation)

Features:

- Kinect support
- Tiling support
- Improvement of visual quality
- Improvement of calibration and alignment process

Performance:

Latency and bandwidth are unchanged from last year. Unfortunately we have not been able to test the performance of the segmentation and tiling implemented this year in a production setting.

Licensing:

Permissive open source (BSD, Apache, MIT or similar)

Source Code / executable Location:

- https://github.com/cwi-dis/cwipc_util
- https://github.com/cwi-dis/cwipc_kinect
- https://github.com/cwi-dis/cwipc_realsense2

Deployment Documentation:

- **Installation instructions:**

  As of this writing installation has to be done from source (for users outside the VRTogether consortium). Preliminary installation instructions are available in the top-level *readme.md* documents of the three repositories.

- **External dependencies:**

  External dependencies are managed automatically through *cmake*, and are all under permissive open source licenses.

- **Deployment steps:**

  See installation instructions.

- **API (function(input) -> output: description)**

  API documentation is available (*Doxygen* style) in the *cwipc_util* repository.

4. Software for Task 3.2 Encoding

After the volumetric data has been captured and possibly displayed locally for self-view it has to be transmitted to the other parties in the experience. But before the data can be transmitted it must be linearized into a format that can be decoded again on the receiving end. Moreover, because volumetric data is very large (at least 10 times larger than video data at a comparable quality level) it needs to be compressed and decompressed again at the receiver side.

There are many compression schemes for “flat” video data and some for 360 video data, but no standard compression schemes for volumetric data. In this task we have built upon the prior art contributed by two of the partners (Certh and CWI) and produced two compression and encapsulation schemes for volumetric video: one for TVM transmission and another for point cloud transmission.

4.1. TVM encoding & transmission

Description:

The encoding component is responsible for the compression and decompression of the transmitted data. Over the years, CERTH/VCL has experimented with a variety of techniques and algorithms for this purpose (OpenCTM, Draco, Corto), but eventually ended up using Corto Cpp library in VolCap for performance reasons and GoogleDraco in VolReco for performance and bandwidth related issues.
Regarding the transmission, both VolCap and VolReco are using RabbitMQ, an open source message broker software. RabbitMQ servers are used as brokers: they accept messages from producers, and deliver them to consumers. They act like a middleman which can be used to reduce loads and delivery times taken by web application servers.

Links:

TVM encoding/decoding and transmission processes are not provided as external tools, but as integrated parts of VolCap and VolReco.

Evolution:

Table 6. Detailed Delta Table – TVM encoding and transmission

	Status Year 0	**Status Year 1**	**Status Year 2**	**Status Year 3**
TRL	TRL5: OpenCTM encoding and simple RMQ transmission in existing multi-view 3D platform	TRL7: Integration of SoA algorithms (Corto, Draco) for mesh compression.	TRL7: Integration of mesh compression new features / updates and optimized RMQ using efficient task management (RxCpp) to handle messages with the most appropriate way for TVM transmission	TRL7: Integration of mesh compression new features / updates and optimized RMQ using efficient task management (RxCpp) to handle messages with the most appropriate way for TVM transmission
Features	Supporting *OpenCTM profile 64x128x64*	Supporting two new encoding profiles: *libcorto* and *libdraco*	Message handling with RxCpp	Message handling with RxCpp
KPIs	*OpenCTM profile 64x128x64*:	*libdraco profile* 64x128x64: -Encoding: ~17ms	*libdraco profile*: 64x128x64: - Encoding: ~15ms	*libdraco profile*: 64x128x64: - Encoding: ~15ms

	- Encoding: ~65ms TVM Bandwidth: ~5Mbps TVM pipeline end-to-end delay: ~225ms	*libcorto profile* 64x128x64: - Encoding: ~10ms TVM Bandwidth: ~2Mbps TVM pipeline end-to-end delay: ~183ms	*libcorto profile*: 64x128x64: - Encoding: ~1ms TVM Bandwidth: ~3Mbps TVM pipeline end-to-end delay: 55ms	*libcorto profile*: 64x128x64: - Encoding: ~1ms TVM Bandwidth: ~3Mbps TVM pipeline end-to-end delay: 55ms

Features:

- Encoding/Decoding's main feature is the compression/decompression of serialized data and metadata of a 3D representation.
- Transmission's main feature is the transmission of serialized data through a RabbitMQ server used as a broker, in order for another application to receive them.

Both of them are handled using efficient task management (RxCpp), resulting in an improved overall performance.

Performance:

Below, we are exhibiting the benchmarking results for the encoding and decoding. As far as the TVM's transmission goes, the bandwidth was about 3MBps while using a voxel grid size of 64x128x64 and HD (1280x720x3) color resolution. With the former parameterization, the pipeline's end-to-end delay was computed about 55 milliseconds.

Table 7. TVM Performance – Draco

Draco	Metrics	Encoding (ms)	Decoding (ms)	Number of Vertices
32x64x32	Average	2.76	1.57	6750
	Standard deviation	0.44	0.45	525
64x128x64	Average	15.47	6.40	29570
	Standard deviation	1.72	2.31	2514

Table 8. TVM Performance – Corto

Corto	Metrics	Encoding (ms)	Decoding (ms)	Number of Vertices
32x64x32	Average	1.05	0.16	6750
	Standard deviation	0.24	0.37	525

64x128x64	Average	13.43	3.40	29570
	Standard deviation	4.81	1.42	2514

Table 9. TVM Performance – Jpeg

Jpeg	Encoding per image (ms)	Decoding per image (ms)	Quality
1280 x 720 x 3	2.7	2.0	95

Licensing:

- Encoding/Decoding:
  - Volumetric Capture (VolCap) application:
    - Corto Cpp library (corto): https://www.gnu.org/licenses/gpl-3.0.html
  - Volumetric Reconstruction (VolReco) application:
    - GoogleDraco Cpp library (draco): https://github.com/google/draco/blob/master/LICENSE
  - Volumetric Capture (VolCap) and Volumetric Reconstruction (VolReco) applications:
    - Apache License, Version 2.0: https://www.apache.org/licenses/LICENSE-2.0.txt
    - SIMD Cpp library (libsimdpp): https://github.com/p12tic/libsimdpp/blob/master/LICENSE_1_0.txt
    - Turbojpeg Cpp library (libjpeg-turbo): https://github.com/libjpeg-turbo/libjpeg-turbo/blob/master/LICENSE.md
- Transmission:
  - Volumetric Capture (VolCap) and Volumetric Reconstruction (VolReco) applications:
    - Apache License, Version 2.0: https://www.apache.org/licenses/LICENSE-2.0.txt
    - RabbitMQ Cpp library (amqpcpp): https://github.com/CopernicaMarketingSoftware/AMQP-CPP/blob/master/LICENSE
    - Blosc Cpp library (c-blosc): https://github.com/Blosc/c-blosc/blob/master/LICENSES/BLOSC.txt

Deployment Documentation:

- **Source Code Location:**

  The encoding/decoding and transmission processes are not provided as external tools, as they are integrated inside VolCap and VolReco. VolCap's source code is not provided, but its binaries can be found under https://github.com/VCL3D/VolumetricCapture/releases. VolReco is closed source and its binaries were only provided to VRTOGETHER's partners.

- **Installation instructions:**

  There are no external tools provided for the encoding/decoding and transmission processes, but, as integrated parts of VolCap and VolReco, an installation is not needed.

- **External dependencies:**

  None.

- **Deployment steps:**

  TVM encoding/decoding and transmission processes are not provided as external tools, but as integrated parts of VolCap and VolReco. As a result, there are no deployment steps to be documented, except from the ones regarding the RabbitMQ server's deployment:

  1. Download and install the Erlang Compiler, by choosing Windows x64 Binary file.
  2. Download the RabbitMQ installation file from RabbitMQ webpage and install.
  3. In order to establish communication between the different parts of the system, you will have to open port: 5672 on Windows Firewall.
     a. Open Windows Start Menu and search for Control Panel
     b. In Control Panel open Windows Defender Firewall
     c. In the left panel select Advanced Settings
     d. In the left panel of the window that popped-up select Inbound Rules
     e. In the right panel that appears select New Rule...
     f. In the window that appeared select Port and hit Next
     g. In Specific local ports text box type 5672 and hit next
     h. In the next window leave the Allow Connection option selected and hit next
     i. In the next window leave everything selected and hit next
     j. In the next window you can add a name for the rule and a description, so as to know that you created that rule, and it's not a Windows default rule.
     k. Repeat all the steps in order to create an Outbound Rule too, for the same port
     l. The same rules for port 5672 must be applied to each capturer processing unit (e.g. Intel-NUC) hosting the Remote Eyes also.

  The former steps can be also found in VolCap's documentation (Chapter 3, Paragraph 3.1, Links section).

- **API (function(input) -> output: description)**

  Regarding the encoding/decoding, a serialized buffer containing the proper data and metadata is compressed/decompressed into a smaller one for performance and bandwidth related reasons.

  As far as the transmission goes, a compressed buffer is transmitted using a RabbitMQ server as a broker and gets received by another application intact.

4.2. Point Cloud (PC) encoding & decoding

Description:

This component offers a generic real-time dynamic point cloud codec for 3D immersive video. The component features low delay encoding and decoding at multiple levels of

detail. Geometry coding is based on octree occupancy and attribute compression is based on existing image coding standards.

Links:

- https://github.com/cwi-dis/cwi-pcl-codec

Evolution:

Table 10. Detailed Delta Table – Point Cloud encoding and decoding

	Status Year 0	**Status Year 1**	**Status Year 2**	**Status Year 3**
TRL	TRL6: Reference implementation of core modules for encoding and decoding	TRL6: Ad-hoc integration of encoder and decoder into GStreamer based delivery pipeline	TRL7: Integration to operational pipeline. Optimization of memory use	TRL7: Optimization for parallelized multiple encoding
Features	Low delay encoding and decoding Lossy geometry coding using octree occupancy Lossy attribute coding using existing image coding standards	Integration for use with a GStreamer based delivery pipeline	Optimization of memory use and performance for dense point clouds captured with 4 cameras	Optimization of codec to support multiple levels of detail for network adaptive streaming. Parallelized Implementation to code point clouds segmented spatially into tiles for viewport adaptive streaming
KPIs	Encode times for Dimitris-2-Zippering Sequence (~320k points) OctreeDepth8: ~80ms OctreeDepth9: ~160ms OctreeDepth10: ~260ms	Together with simple capturer end to end pipeline is able to operate at 5 fps with 1 camera on commodity hardware	Together with simple capturer end to end pipeline is able to operate at 15fps with 4 cameras on commodity hardware. End to end latency of 400 ms	

Features:

Parallel encoding for multiple tiles and multiple quality levels.

Performance:

Unfortunately we have not been able to test the performance of the parallel encoder in a production setting, so we can not make a formal statement about it. Lab experiments have shown that the performance of tiled parallel encoders is at least as good as the performance of a single (untiled) encoder.

Licensing:

Permissive open source license (BSD, Apache, MIT or similar)

Source Code / executable Location:

Eventually https://github.com/cwi-dis/cwipc_kinect, as of this writing https://github.com/cwi-dis/cwi-pcl-codec .

Deployment Documentation:

- **Installation instructions:**

  Toplevel *readme.md* file in the repository of the component, plus the documentation in the *cwipc_util* component.

- **External dependencies:**

  Handled automatically through *cmake*. All external dependencies are distributed under a permissive open source license.

- **Deployment steps:**

  See installation instructions

- **API (function(input) -> output: description)**

  Included in API description of *cwipc_util* component.

5. Software for Task 3.3 Orchestration & Delivery

This task includes several software components. These components are in charge of distribution of media and of media orchestration. The media orchestration component creates one single unified experience based on the different data formats distributed by the platform. This component is at the core of the platform and deals with major challenges like bandwidth adaptation, media synchronization and scene orchestration. The distribution of media is covered by three components: the sender (bin2dash), the distributor (Evanescent), the receiver (Signals Unity Bridge (SUB)). These three components and the DASH transmission system are stable. The live presenter content goes through some specific component (RTMP Live Video Transmission - Gstreamer Unity Bridge). The content can be intelligently distributed by a MCU (Multipoint Conferencing Unit, both WEBRTC-VR-MCU and PC-MCU): these components provide an innovative VR conferencing platform, which is suitable for at least up to 10 participants. They perform the mixing of a multitude of content into a single output for each user in a central (cloud-based) platform, controlled by the media orchestration components.

5.1. Media/Session Orchestrator

Description:

The VRT orchestrator, as a centralized server component, is primarily in charge of the management of a set of end-user profiles that are connected into a platform session. The orchestrator is able to control the various connection events in order to grant a unified experience and ensure that whole media pipelines are properly switched across the various connected nodes. It is also the reference point used for the synchronization of the media pipelines. Then, from the backend viewpoint, the orchestrator is in charge of managing and supervising the transmission and the delivery of all of the media stream pipelines.

Experimental features:

Aside from the implementations developed to meet platform requirements; a framework extension has been integrated as an experimental work to provide further media transmission capabilities. The idea is to evaluate the orchestrator exchange protocol allocated in this perimeter. Thus, it is possible to carry-on audio and volumetric streams through an additional module of the orchestrator, which forward incoming packets within all connected users.

Technologies:

In terms of technologies, the central Core Orchestrator component relies on the Node.js open-source platform where its API is implemented in JavaScript. The protocol exchange takes benefits from the socket.io API, distributed under MIT licensing model, which enables real-time communication over connected nodes.

Finally, the orchestrator framework is integrated on the client side with the UMTS (User Manager Test Simulator) package. This module developed within the Unity Game Engine is developed in C# to provide the necessary client-side API to be wrapped in order to connect the Orchestrator with the User node (i.e Capture, Encoding and Play-out components). The UMTS framework is available as a licensed software module and relies on the BestHTTP Unity framework dependency to support the socket.io communication. The necessary modifications have been made to BestHTTP to support our multitask system.

Links:

Stable service location: https://vrt-orch-ms-vo.viaccess-orca.com/socket.io/
Sandbox testing channel location: https://vrt-orch-sandbox.viaccess-orca.com/socket.io/
(will not be maintained after the end of the project)

Evolution:

Table 11. Detailed Delta Table – Orchestrator

	Status Year 0	Status Year 1	Status Year 2	Status Year 3
TRL	TRL0	TRL2	TRL5	TRL6
Features	Non existing component at initial time	Static platform configuration	- Native pipeline cloud components supervision - PC & TVM pipelines supervision - Live presenter supervision - Users sessions management - Socket.io audio transmission	- Year 2 + - Interactive events transmission and supervision - Virtualization pre-packaging
KPIs	NA	- 2 users managed statically for each session - One session at a time	- 4 users managed dynamically - Up to 5 sessions can run simultaneously - 300 ms audio latency	- 10 users managed dynamically - Up to 10 sessions can run simultaneously

Features:

The main functionalities offered by the final orchestrator are:

- User sessions management: the orchestrator handles the whole lifecycle of a user session (connect/disconnect, login / logout, join/leave VR session, join/leave VR room)
- VR scenes descriptions: the orchestrator can ingest high-level scenes description:
  - Description of a Scenario
  - Description of any number of rooms within the scenario
  - Description of timed scene events in the scenario. The orchestrator can then dispatch scene events on time to users playing the scenario.
- VR sessions management: the orchestrator allows users to create VR sessions or to join existing sessions, and to communicate with other users within the same session (send/receive messages or audio/video/PCC data).
- Generic data streams framework: the orchestrator provides an API that allows users to share any type of binary data streams with other users within the same session:
  - Users can declare any number of data streams, with an associated stream type

    - Users can register for listening to any number of data streams from any number of other users
    - The orchestrator handles routing of the streams data to the registered users
- Third party modules management: the orchestrator can also be configured to manage the lifecycle of third-party applications (e.g. Live Presenter)
- NTP synchronization: the orchestrator can be used as the reference clock for the VR sessions.
- Interactive events transmission and supervision
- Virtualization pre-packaging

Performance:

The final release of the Core Orchestration service can support;

- up to 100 of registered users accounts,
- up to 10 users managed dynamically,
- up to 10 simultaneous sessions,
- real-time event messaging system,
- socket.io based PCC packet streaming,
- socket.io based Audio packet streaming (about 300 ms latency)

Licensing:

- Core: MIT
- UMTS: VOSA (Viaccess-Orca Software Agreement) 1.0 License
    - (Available per request at https://www.viaccess-orca.com/contact)
- LRTS: Permissive open source license, such as MIT, BSD or Apache

Source Code / executable Location:

The Core Orchestrator package sources can be found at:
https://baltig.viaccess-orca.com:8443/VRT/orchestration-group/Orchestration

(Will be migrated once the public forge will get open)

The Core Orchestrator releases can be found at:
https://baltig.viaccess-orca.com:8443/VRT/orchestration-group/Orchestration/releases

(Will be migrated once the public forge will get open)

Deployment Documentation:

Installation instructions:

- Download and install NodeJS for your system: https://nodejs.org/en/download/
- Download the last core Orchestrator release (ZIP) at: https://baltig.viaccess-orca.com:8443/VRT/orchestration-group/Orchestration/releases
- Extract the Orchestrator-Core-vX.X.X-x64.zip
- Open a command line prompt into the Orchestrator-Core-vx.x.x-x64/Core folder
- Run command npm install to download or update NodeJS dependencies (this will create or update node_modules folder)

Internal Project dependencies:

- LRTS (Live Recording test Simulator): The LRTS module is a set of predefined commands framework working with 2 Python scripts to:
  - Start the Traditional Audio and Video Encoding software, which is packaged in a Docker image with an nginx-rtmp server as well as an Evanescent instance.
  - Start an FFMPEG stream session when the Live camera is not operational (developer option).
  - Stop the Docker image as well as the FFMPEG running session.

  https://baltig.viaccess-orca.com:8443/VRT/orchestration-group/Orchestration/-/releases (Will be migrated once the public forge will be opened).
- UMTS (User Manager Test Simulator): C# integration framework developed within the Unity Game Engine to provide the necessary client-side API to be wrapped in order to connect the Orchestrator server with the Unity client.
- https://baltig.viaccess-orca.com:8443/VRT/orchestration-group/Orchestration/-/releases (Will be migrated once the public forge will be opened).

External dependencies:

- NodeJS: https://nodejs.org/en/

Deployment steps:

- The Core/config folder contains four configuration samples (*.json.samples) files for the Orchestrator.
- Rename each sample configuration file by removing trailing .sample suffix.
- Modify any configuration file as needed to conform to your integration environment:
  - app-config.json describes the host/port for the Orchestrator service:
  - hostname: the host name.
  - port: the port the Orchestrator service will be bound to.
  - sfu-config.json describes the SFU configuration:
    - tls: indicates whether TLS will be used by instantiated SFU modules. Please note that this is only informative; TLS must be configured in SFU command line arguments to be effective.
    - portMapping declares a list of [port, sfuData] pairs, described as follows:
      - port: one port number available for SFU instantiation.
      - sfuData: a set of [key, value] pairs that will be transparently sent to users when calling GetSfuData() or GetUserInfo() commands. Three special values can be used in the value field:
        - %SFU_PORT% will be automatically replaced with the port the SFU is bound to.
        - %SESSION_ID% will be replaced with the sessionId of the session the user belongs to (if any).
        - %USER_ID% will be replaced with the userId of the user calling for sfuData.
    - autorestart: declares whether each SFU instance must automatically restart when stopped or crashed.
    - log, logFilePfx, logFileSfx:

        - log: indicates whether the SFU instances output must be logged.
        - logFilePfx, logFileSfx: indicates which prefix/suffix must be used for the log file name. The log file name is then created as logFilePfx + port + logFileSfx, where port is the port the SFU is bound to.
    - commandLine: declares the command line to be used for SFU instantiation. Each argument must be set in a separate string. The special %SFU_PORT% value will automatically be replaced by the SFU chosen port.
- users.json declares the users. Each user is defined by:
    - userId: UUID that uniquely identifies the user.
    - userName: the login username for the user.
    - userPassword: the login password for the user.
    - userAdmin: states whether the user is an admin user (only admin users can add/remove users / scenarios and query for orchestrator full state).
    - userData: a map of [key,value] pairs that users can manage with getUserData(), updateUserDataXXX() and ClearUserData() API calls.
- scenarios.json declares all the available scenarios. Each scenario is defined by:
    - scenarioId: UUID that uniquely identifies the scenario.
    - scenarioName: a label for the scenario.
    - scenarioDescription: a description for the scenario.
    - scenarioRooms: a list of rooms for the scenario. Each room is identified by:
        - roomId: UUID that uniquely identifies the room.
        - roomName: a label for the room.
        - roomDescription: a description for the room.
        - roomCapacity: the maximum number of users that can sit in the room.
    - scenarioGltf: the GLTF description for the scenario.

The Orchestrator service can be launched with the following procedure:

- Open a command line prompt into the Core folder.
- Run node app.js command

API

https://baltig.viaccess-orca.com:8443/VRT/orchestration-group/Orchestration/-/blob/master/README.md (will be migrated once the public forge will get open).

5.2. RTMP Live Video Transmission - Gstreamer Unity Bridge

Description:

Software component that connects GStreamer with Unity. It is able to play any media URI provided by GStreamer (1.X) pipelines into Unity 3D textures

[editor note i.e. from web: Key Components of Native VR-Together PlatformU1Zp4UoV2WLJ0/edit]

Links:

- Github

Evolution:

Table 12. Detailed Delta Table – RTMP Live Video Transmission

	Status Year 0	Status Year 1	Status Year 2	Status Year 3
TRL	TRL6 (Component developed in EU H2020 ImmersiaTV project)	(Not Used in the Project during Year 1)	TRL6	(Not Used in the Project during Year 3)
Features	- Ingest of DASH and RTMP streams (or other media URIs) from GStreamer into Unity		- Same features, but component updated to interact with newer versions of GStreamer.	
KPIs	- Used for stored DASH streams - Smooth playout - No delays tests conducted so far		- End-to-End latency for a live RTMP stream: ~1.5s	

Features:

The GStreamer Unity Bridge (GUB) allows playing media (Audio and Video) as a texture on Unity, using the GStreamer framework (https://gstreamer.freedesktop.org), which is contained in the Plugins folder.

Main Feature: Ingest of DASH and RTMP streams (or other media URIs) from GStreamer into Unity

Performance:

Live Streaming of stereoscopic video with latency <2 seconds

Licensing:

- Core: LGPL Licence
- Depts:
  - GStreamer: LGPL Licence

Source Code / executable Location: https://github.com/ua-i2cat/gst-unity-bridge

Deployment Documentation:

- **Installation instructions:** https://github.com/ua-i2cat/gst-unity-bridge/blob/master/README.md

- **External dependencies:** GStreamer
- **Deployment steps:** https://github.com/ua-i2cat/gst-unity-bridge/tree/master/Plugin
- **API (function(input) -> output: description):** The system is composed of a set of C# scripts to be used inside Unity, which interact with a native plugin (GstUnityBridge.dll or libGstUnityBridge.so).

5.3. (PC/Audio/Live) DASH Sender - bin2dash

Description:

bin2dash is an open API allowing to package any volumetric data in the industry-standard MP4and then stream it using MPEG-DASH.

Links:

https://baltig.viaccess-orca.com:8443/VRT/nativeclient-group/EncodingEncapsulation

Evolution:

Table 13. Detailed Delta Table – DASH Sender

	Status Year 0	Status Year 1	Status Year 2	Status Year 3
TRL	TRL4	TRL6	TRL7	TRL7
Features	Integrated component (plugged with capture and encoding)	New modular structure and new Metadata handling	Improved with more data types, more testing use-cases	Tiled Point Clouds. Support for TVMs
KPIs	Latency: 2 frames + internal threading	Latency: 1 frame + internal threading	Latency: 1 frame (a complete frame is needed for copy)	Latency: 1 frame (a complete frame is needed for copy)

Features:

This component enables to handle all (encoded) legacy media like video, audio, subtitles, metadata (including timed metadata) but it has been adapted also to handle any kind of volumetric data from Point Clouds (V-PCC, G-PCC), Mesh/TVM, MPEG 3DoF+, or any personalized data. It offers capabilities included by DASH which are mainly scalability and low-latency.

Performance:

This component has reached the lowest possible latency of one frame for any type of data. Latency may of course be higher if a dependency pattern is induced between sent samples.

Licensing:

proprietary

Source Code / executable Location:

Source Code Location:

https://baltig.viaccess-orca.com:8443/VRT/nativeclient-group/EncodingEncapsulation

Deployment Documentation:

- **Installation instructions:** extract the archive in the desired folder. Then follow the instruction to build the sample or build your own application.
- **External dependencies:** dependencies are included in the binary release.
- **Deployment steps:** deployment instructions are provided by the components integrating bin2dash (to avoid environment collision leading to a wrong loading)
- **API (function(input) -> output: description):** provided in the bin2dash.hpp file (that is part of every release)

5.4. (PC/Audio/Live) DASH Distributor - evanescent

Description:

Evanescent is a server component that allows to receive any kind of DASH stream and simply forward them to other receiver clients. In this sense, Evanescent is an SFU (Stream Forward Unit). More technically, it receives low latency dash chunks and serves them back to a low latency dash receiver using http chunked transfers. Evanescent is implemented in C++ based on Signals sub-modules.

Links:

- **Source Code Location**: https://git.gpac-licensing.com/alaiwans/Evanescent
- **Documentation:** LD_LIBRARY_PATH=$DIR $DIR/evanescent.exe [--tls] [--port port_number]
- **Installation guide:** just copy the files, see details at https://baltig.viaccess-orca.com:8443/VRT/deliverymcu-group/DeliveryMCU/blob/master/README.md.

Evolution:

Table 14. Detailed Delta Table – DASH Distributor

	Status Year 0	Status Year 1	Status Year 2	Status Year 3
TRL	TRL0	TRL4	TRL5	TRL5
Features	NA	Basic system with HTTP and HTTPS/SSL	SFU testing toward MCU	Intelligence in processing tiled content.
KPIs	NA	Latency: 1 HTTP chunk request	Latency: 1 HTTP chunk request	Latency: 1 HTTP chunk request with tiled content

Features:

Evanescent is a server component that allows to receive any kind of DASH stream and directsimply forward them to other receiver clients. In this sense, Evanescent is an SFU (Stream Forward Unit). More technically, it receives low latency dash chunks and serves them back to a low latency dash receiver using http chunked transfers.

Performance:

This component is low footprint in terms of CPU, memory and network overhead. It offers the best possible latency of one chunk (typically one fMP4 "fragment").

Licensing:

Proprietary.

Source Code / executable Location:

- https://baltig.viaccess-orca.com:8443/VRT/deliverymcu-group/DeliveryMCU/-/releases

Deployment Documentation:

- **Installation instructions:** extract the archive at the desired location.
- **External dependencies:** optionally libssl/libcrypto. Either the version from a system or the version provided in the archive may be used.
- **Deployment steps:** launch the executable to launch a new server SFU.

5.5. (PC/Audio/Live) DASH Receiver - Signals Unity Bridge (SUB)

Description:

This component is the receiving end of the pipeline. It allows to unpackage any standard MPEG-DASH streams into raw streams to be used by the receiver client. It also contains a preliminary algorithm with respect to tiling support. It has been extended to read files and RTMP streams for debugging and interoperability purposes.

Links:

- *https://baltig.viaccess-orca.com:8443/VRT/nativeclient-group/SUB/releases*

Evolution:

Table 15. Detailed Delta Table – DASH Receiver

	Status Year 0	Status Year 1	Status Year 2	Status Year 3
TRL	TRL4	TRL5	TRL6	TRL6
Features	Basic DASH reception	RTMP input	Support for more streamers (e.g. not only bin2dash)	Support for tiling and TVM.

KPIs	Latency: 2 ISOBMFF/MP4 fragments	Latency: 1 ISOBMFF/MP4 fragment or 1 frame (RTMP)+network delays	Latency (est.): 1 ISOBMFF/MP4 fragment or 1 frame (RTMP)+network	Latency: 1 HTTP chunk request with tiled content

Features:

This component is the receiving end of the pipeline. It allows to unpackage any standard MPEG-DASH streams into raw streams to be renderedused (e.g. displayed) by the receiver client. It also contains preliminary algorithms with respect to tiling support. It has been extended to read files and RTMP streams for debugging and interoperability purposes.

Performance:

This component has reached the lowest possible latency. The tiling switching delay is of maximum one GOP (or equivalent), which is sufficient regarding the TRL of this subpart of the component.

Licensing:

Proprietary.

Source Code / executable Location:

- https://baltig.viaccess-orca.com:8443/VRT/nativeclient-group/SUB

Deployment Documentation:

- **Installation instructions: extract the archive in the desired folder. Then follow the instruction to build the sample or build your own application.**
- **External dependencies: dependencies are included in the binary release.**
- **Deployment steps: deployment instructions are provided by the components integrating bin2dash (to avoid environment collision leading to a wrong loading)**
- **API (function(input) -> output: description): provided in the signals_unity_bridge.hpp file (that is part of every release)**

5.6. (PC/Audio/Live) Live presenter

Description:

The component receives a live RTMP stream from a 3D stereoscopic camera, crops both eyes, transcodes it to an appropriate quality, and restream it to the VRTogether session participants.

Links:

https://www.gpac-licensing.com /downloads/VRTogether/vrt.tar.gz

Evolution:

Table 16 . Detailed Delta Table – Live presenter

	Status Year O	Status Year 1	Status Year 2	Status Year 3
TRL	NA	TRL5	TRL6	TRL6

Features	NA	Overall Component · RTMP input · DASH output	Overall Component · RTMP input/output	Overall Component No core modifications expected. This module heavily depends on the capture material capabilities.
KPIs	NA	Latency <= 2s	Latency <= 1s	Latency <= 1s

Features:

The RTMP reception uses nginx and nginx-rtmp, splits both mono streams, crops the left and right eye to fit the presenter, recombine both eyes, and transcodes the video into a final stream (using FFmpeg) which can be processed as any other media in VRTogether, then fetched by the player using RTMP or DASH. The whole pipeline is low latency and runs in a docker.

Performance:

This component has reached an acceptable level of performance. The configuratibility is static, which means that the characteristics of the stereoscopic video input stream must be known in advance. The component transcodes the video which requires some CPU ressources.

Licensing:

MIT

Source Code / executable Location:

https://www.gpac-licensing.com /downloads/VRTogether/vrt.tar.gz

Deployment Documentation:

- **Installation instructions:**
  `wget https://www.gpac-licensing.com/downloads/VRTogether/vrt.tar.gz && gunzip vrt.tar.gz`
- **External dependencies:** *N/A*
- **Deployment steps:**
  `docker load < vrt.tar && docker run -p 1935:1935 --name vrt --rm -t vrt`
- **API (function(input) -> output: description):** *port 1935 is open to receive the input, which is then pushed to the location passed as a parameter*

5.7. WEBRTC-VR-MCU

Description:

The WebRTC VR MCU system is capable of ingesting any number of WebRTC client inputs. The MCU system synchronises and composes its inputs into a single output stream which is then published using WebRTC. As a result, clients are able to retrieve all relevant streams together instead of separately. This optimizes the network bandwidth due to more efficient routing (each client only sends its video stream to the MCU, and no longer to all other clients), as well as the decoding resources (clients have a limited amount of hardware decoders, which can now be dedicated to decoding all streams instead of just one) of said clients.

Incoming WebRTC streams are demuxed into RTP streams and are fed into a pluggable media pipeline. The media pipeline performs the compositing operation and outputs a single RTP stream which is broadcast to each client using WebRTC return streams. The media pipeline of the MCU system has been designed as a containerized service such that given enough hardware, it is horizontally scalable over multiple parallel sessions. These services are managed using Docker Swarm and the orchestrator component.

Links:

- MMSys Demo Paper: https://dl.acm.org/doi/abs/10.1145/3304109.3323838

Evolution:

Table 17. Detailed Delta Table – WebRTC VR MCU

	Status Year 0	Status Year 1	Status Year 2	Status Year 3
TRL	2	3	5	6
Features	- Only baseline code - No MCU (Full P2P mesh)	- initial concept of component stubs	- Horizontally scalable MCU - Orchestration - Integration with web Player	- Number of users 16 - Fully containerized deployment - Support for low-powered (mobile) devices
KPIs	Number of simultaneous users 3	Number of simultaneous users 4	Number of simultaneous users 8	Number of simultaneous users 16

Features:

- Fully containerized deployment
- Utilization of full mosaic (horizontal and vertical matrix)
- support for 8 users in 3D - RGBD
- support for 16 users in 2D - RGB +Chroma
- support for 10 users in 2D on low-powered (mobile) devices (i.e. Oculus Quest)

Performance:

In order to evaluate the final state of the system and overall benefit of the MCU we run a set of simulated measurements with a different set of user streams (2-16 streams) comparing both the performance of MCU and peer-to-peer transmission. Further we run a test with a Oculus Quest non-tethered HMD show the performance on a mobile HMD.

The regular web client evaluation had the following configuration:

- Device: Laptop MSI GS65 8RF Stealth Thin (CPU: Intel Core i7-8750H CPU @ 2.20GHz (6 cores / 12LPUs; GPU: NVIDIA GeForce GTX 1070 Max-Q; Memory 32 GB)
- Network connection: wired Internet access with (30Mbit/s up and 50 Mbit/s down)
- Room: 3D GLTF model of a conference room

- User representation:
  - 2D sprites (RGB+Chroma background in 540x800px resolution blend into the environment)
  - 3D volumetric (RGB+Depth in 1080x800px resolution)
- Browser: Google Chrome
- Measurements: Resources Consumption Metrics (RCM) measurement tool (https://github.com/ETSE-UV/RCM-UV)

The Oculus Quest evaluation had the following configuration:

- Device: Oculus Quest
- Network connection: Internet over USB connection (gnirehtet; 30Mbit/s up and 50 Mbit/s down)
- Room: 360-degree 2D image of a conference room
- User representation: 2D sprites (RGB+Chroma background in 540x800px resolution blend into the environment)
- Browser: Firefox reality browser
- Measurements: OVRMetricsTool_v1.4 (https://developer.oculus.com/downloads/package/ovr-metrics-tool/)

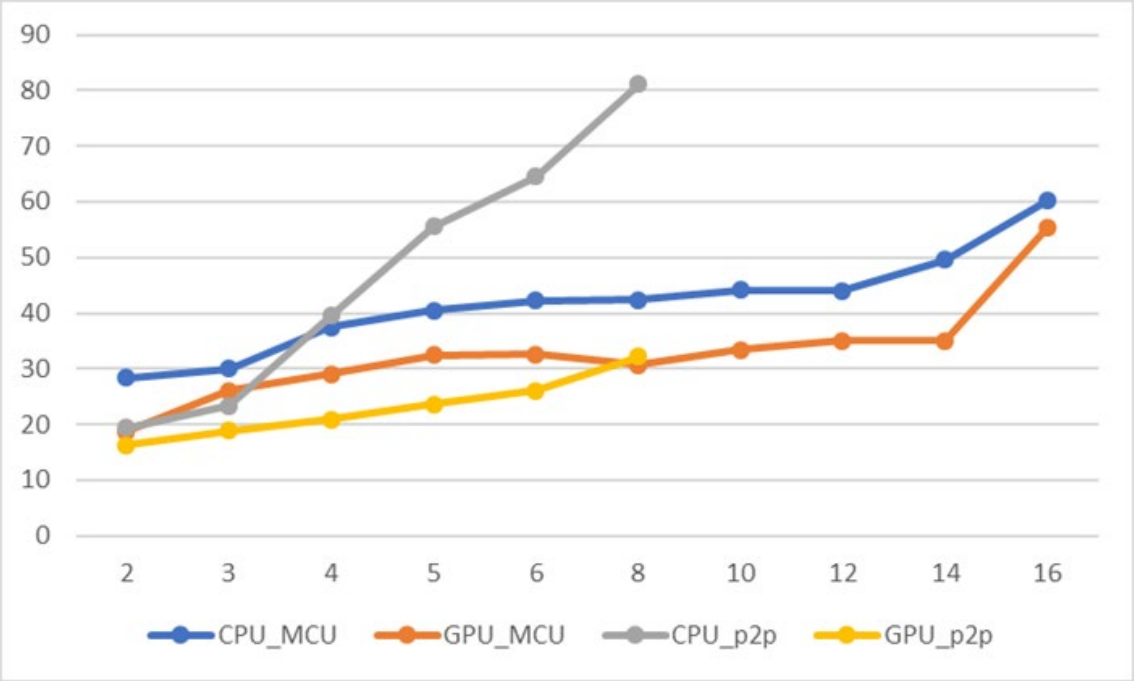

Figure 3. WebRTC MCU performance of 3D conference room with users rendered as 2D sprite (with green chroma background in 540x800px resolution)

While in peer-to-peer a client follows a steep increase in CPU utilization, the simulation results show an overall suitable utilization of resources (see Figure 3). This is the client and MCU can support up to 16 simultaneous users in 2D. Further the test show that from 4 users the usage of an MCU is recommended.

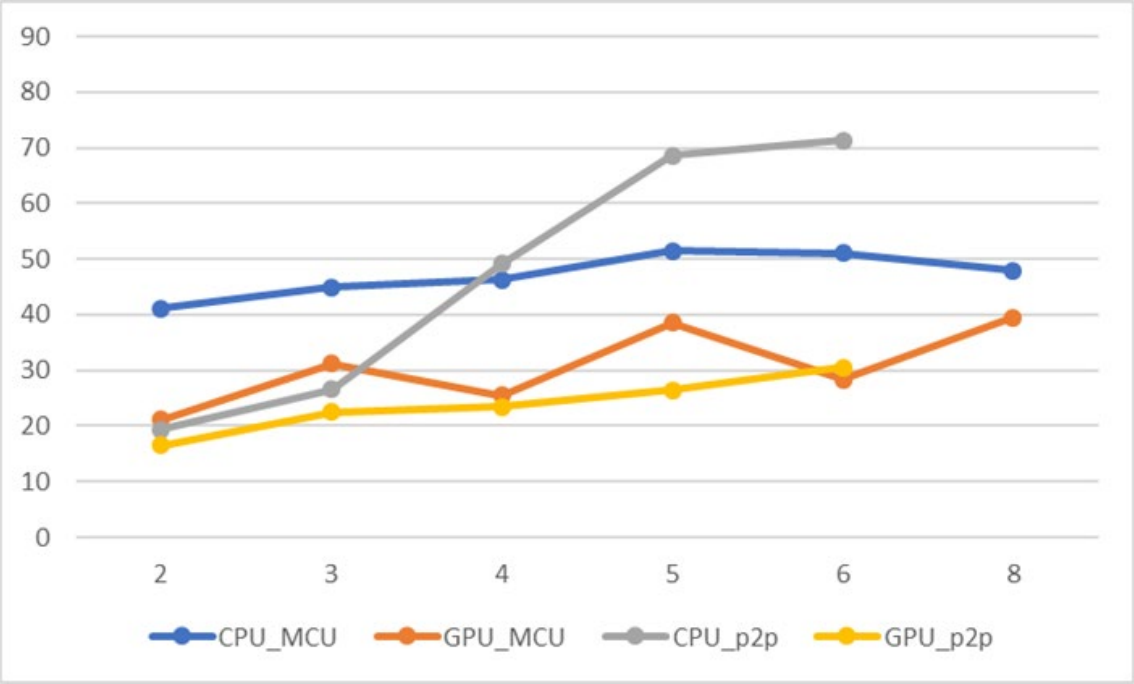

Figure 4. WebRTC MCU client performance of 3D conference room with users rendered as 3D point cloud (from RGBD video with total of 1080x800px resolution)

The behavior of the system is similar for 3D user representations (Figure 4) then for 2D users. This is the CPU is rapidly increasing in the peer-to-peer transmission vs. MCU-based transmission. Further we can see an overall increase in performance usage in the 3D condition allowing for 8 simultaneous users.

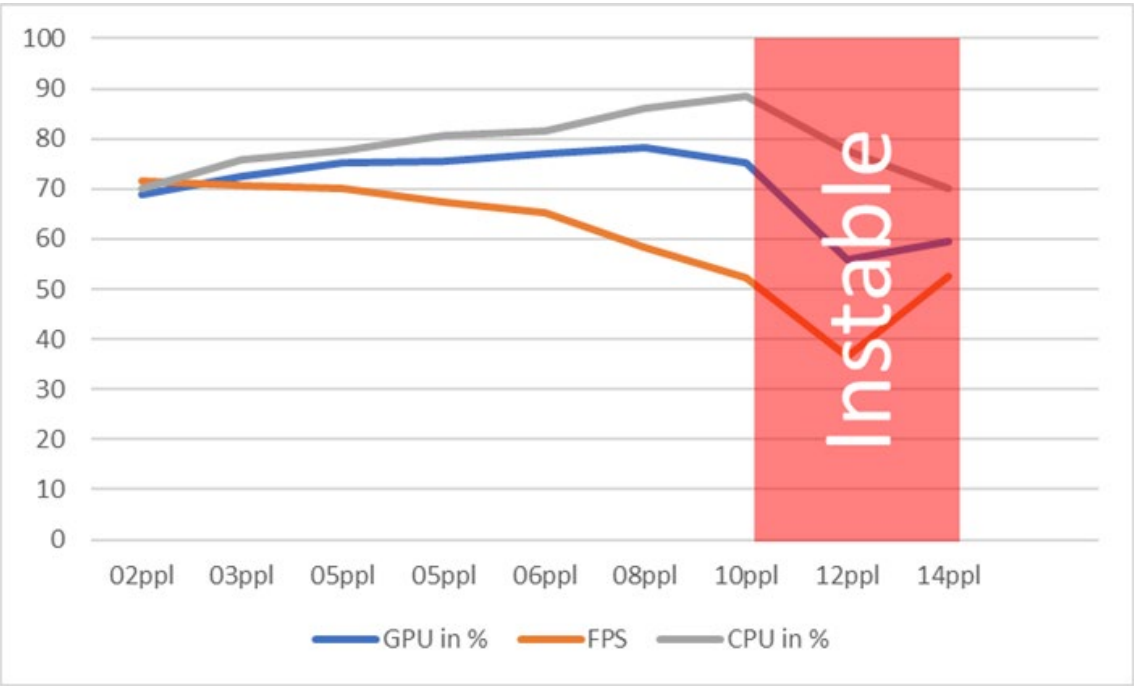

Figure 5. WebRTC MCU client performance of Oculus Quest running Firefox reality browser with room as 360-degree image background and users rendered as 2D sprite (with green chroma background in 540x800px resolution)

Running the web-client on a Oculus Quest device shows a much higher resource utilization (see Figure 5). In particular, we can see instabilities from above 10 simultaneous users. This is the device cannot keep up the rending of frames and we can consider this unusable. Thus, in total our current client and MCU combination can support up to 10 simultaneous users in 2D.

Licensing:

The WebRTC-VR-MCU follows a proprietary license that can be made available to consortium members or to any customer on an ad-hoc basis. Further information and contact details are provided here:

- https://www.tno.nl/en/focus-areas/information-communication-technology/roadmaps/fast-open-infrastructures/social-xr-extended-reality/

Source Code / executable Location:

- *https://ci.tno.nl/gitlab/vrtogether/mcu-supervisor*
- *https://ci.tno.nl/gitlab/vrtogether/vrt-janus*
- *https://ci.tno.nl/gitlab/vrtogether/vrt-mcu*

The code is available in a private repository of TNO on request, you can find more details here.

Deployment Documentation:

- To install simply follow the install script of the different git repo
  - cd ~/git/vrt-janus && sudo ./build.sh
  - cd ~/git/vrt-mcu && sudo ./build.sh
  - cd ~/git/mcu-supervisor && npm install
- Then the mcu-supervisor service can simply be started
  - cd ~/git/mcu-supervisor && sudo node index.js

Minimum server's system requirements:

- Minimum of 4GB RAM (approximately 4GB for 4K output)
- At least one CPU (4-8 cores), but multiple CPUs are supported for performance gains.
- For 6 users: estimated 18Mbps uplink, 108Mbps downlink

Software requirements:

- **Ubuntu OS** - (min version 18.04, https://www.ubuntu.com/)
- **NodeJS** - (min version 12.16.2, https://nodejs.org/en/)
- **Docker** - (min version 18.09.5, https://docs.docker.com/)
- **yarn** (https://yarnpkg.com/)
- ***gstreamer*** (https://gstreamer.freedesktop.org/)
- **python 3** (https://www.python.org/)

5.8. PC-MCU

Description:

(Virtualized) Holoconferencing cloud-based component that aims at reducing the end-user client computational resources and bandwidth usage, providing the following key features: fusion of different volumetric videos, Level of Detail (LoD) adjustment and Field of View (FoV) aware delivery.

Links:

- Demo video: https://www.youtube.com/watch?v=XqLOtSKHzFE
- NOSSDAV 2020 paper: *https://dl.acm.org/doi/abs/10.1145/3386290.3396936*

Evolution:

Table 18. Detailed Delta Table – PC MCU

	Status Year 0	Status Year 1	Status Year 2	Status Year 3
TRL	N/A (Not available at the start of the project)	TRL0	TRL1	TRL 3
Features	N/A	N/A Analysis of state-of-the-art Preliminary core implementation	Preliminary PoC	- Integration with PC pipeline - Sequential implementation – Multiple Volumetric Video de/coding – Reception and delivery of multiple streams – Level of Detail (LOD) adjustment – Removal of non visible volumetric video – Fusion of volumetric videos – Parallel Operations
KPIs	N/A	N/A	N/A	Overall KPI: - CPU saving for the client: >50% - Bandwidth saving for the client: >50% - Users supported in a Holoconferencing session: 4+ - Supported Resolution per user: Up to 50k voxels - Latency: <500 ms PC-MCU Core functionalities KPIs: - Fusion: 4+ users supported in real time - LOD Selection: 3+ resolutions supported

Features:

- Multiple Volumetric Video de/coding, being compatible to the most common Point Cloud compression strategies.
- Reception and delivery of multiple MPEG Dynamic Adaptive
- Streaming over HTTP (DASH) streams.
- Level of Detail (LOD) adjustment: the Point Cloud representation of the participants can be down-sampled, providing the most appropriate resolution based on the relative users' position, activity and underlying context.

- Removal of non visible volumetric video: the non visible parts of the volumetric environment can be removed and never received.
- Fusion of volumetric videos: the incoming videos are decoded, processed accordingly, and converted into as a fused volumetric video for the scene, thus providing a single stream to the client devices.

Performance:

In order to validate the goal KPI stated in the evolution matrix, we have performed a set of experiments in simulated environments (detailed in the work of Cernigliaro et al.[3]). The results of the computational and bandwidth savings over perform the initial stated goals:

Table 19. PC-MCU Performance

-	No PC-MCU	PC-MCU	Δ Reduction
CPU (%)	25	7.8	68.7%
GPU (%)	27.2	25.4	6.6%
RAM (MBs)	445.5	366.3	17.85%
BW (Mbps)	92.3	14.9	83.9%

Licensing:

- Core: Proprietary Code, proprietary License (TBD)
- Depts:
  - GPAC: LGPLv2+[4] or proprietary license https://github.com/gpac/gpac/blob/master/COPYING
  - cwipc_codec: Permissive open source license, such as MIT, BSD or Apache
  - PCL: BSD license

Source Code / executable Location: The code will be eventually published in a https://github.com/ repository

Deployment Documentation: The documentation will be published, together with the code, in a https://github.com/ repository

6. Software for Task 3.4 Rendering and Display

In this chapter, we are going to describe the software related to the Task 3.4 “Rendering and Display”. The intention is to provide two different client software components in order to address different configurations and setups at home. This is a native client implemented in unity and a web-based client implemented in node.js/react. The clients incorporate different aspects for data capture, data rendering and processing to address different aspects of the platform on the user end devices, like point-cloud data, meshes, videos, and interactive characters. The intention is to provide two different versions of the rendering.

[3] Gianluca Cernigliaro, Marc Martos, Mario Montagud, Amir Ansari, and Sergi Fernandez. 2020. PC-MCU: point cloud multipoint control unit for multi-user holoconferencing systems. In Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '20).

[4] https://www.gnu.org/licenses/old-licenses/lgpl-2.1.txt

6.1. Web-based player

Description:

The entry point for the TogetherVR Web client is offered by the web server back-end, which is based on Node.js, React and Aframe. This allows any modern WebVR enabled browser to display the VR content on a screen or an OpenVR (https://github.com/ValveSoftware/openvr) enabled VR-HMD (a.o. the Oculus Rift CV1 or the Valve Index). Furthermore the client can access the image produced by the TNO RGBD Capture module to be displayed as self-view to the user itself or to be sent via WebRTC to one or multiple other clients. In order to support such a multiuser connection, the client is connected to a second WEB server that is handling all sorts of multimedia orchestration. The rendering of users in the VR environment is done via custom WebGL shaders that alpha-blend user representations into the surroundings for a natural visual representation. Audio is also captured, transmitted and made audible as spatial audio with the help of the Google Resonance APIs.

Links:

- Explanation Video
- Overview Paper,
- Look & Feel of 1st 360-degree prototype (video)

Evolution:

Table 20. Detailed Delta Table – Web-based Player

	Status Year 0	**Status Year 1**	**Status Year 2**	**Status Year 3**
TRL	6	7	7	7
Features	- 360-degree only - Peer to peer streaming	- Pilot 1 content - Volumetric - Streaming via MCU - Self-view - Spatial audio - Admin panel	- comptel client Refactor based on redux/reflux - Pilot 2 content - Optimized MCU pipeline - Use capture (hardware) parameters for improved rendering	- Integrated technical measurements - New virtual room(s) with a focus on communication - Support for low-powered (mobile) devices
KPIs	MOS score for 360-degree exp. (5-point): - overall exp. 4.01 - video quality 3.59	MOS score for 360-degree exp. (5-point): - overall exp. 4.35 - video quality 3.65	MOS score for 3D exp. (9-point): - overall exp. 6.92	Full System end to end (glass-to-glass) delays: - RGB delay -- p2p 396ms -- MCU 564ms - RGBD delay -- p2p 384ms -- MCU 622ms

Features:

- Simply (No-install) web client (to allow easy access for users to VR communication)
- Support for different HMDs based on OpenXR
- Central Experience Control
- Improved self-view shader (geometry)
- Support Non-tethered HMD (i.e. Quest)
- integrated real-time measurement (WebRTC and AFrame Stats)
- More social features
    - Screen sharing
    - Object interaction
    - Synchronized video watching

Performance:

To evaluate the web client, we conducted full end-to-end (capture to display) delay measurements connecting the clients via peer-to-peer connections and the MCU. Measurements were done with VideoLat (https://videolat.org/) at at least 1000 samples.

Table 21 Full System end to end (glass-to-glass) delays:

	RGB delay (540x800px)	RGBD delay (1080x800px)
peer-to-peer	396ms	384ms
MCU	564ms	622ms

Further to outline the user experience the web client was tested by multiple users in different settings with the following result in Mean Opinions Scores (MOS):

- https://show.ibc.org/__media/Files/Tech%20Papers%202019/-Simon-Gunkel.pdf
- MOS score for 360-degree exp. (5-point):
    - overall exp. 4.35
    - video quality 3.65
- MOS score for 3D exp. (9-point):
    - overall exp. 6.92

Licensing:

The web-based player follows a proprietary licence that can be made available to consortium members or to any customer on an ad-hoc basis. Further information and contact details here:

- https://www.tno.nl/en/focus-areas/information-communication-technology/roadmaps/fast-open-infrastructures/social-xr-extended-reality/

Source Code / executable Location:

- https://ci.tno.nl/gitlab/vrtogether/orchest.js/
- https://ci.tno.nl/gitlab/vrtogether/signalmaster
- https://ci.tno.nl/gitlab/vrtogether/tvr-turn
- https://ci.tno.nl/gitlab/vrtogether/togethervr2

The code is available in a private repository of TNO on request, you can find more details here.

Deployment Documentation:

- Once a server is installed a user can simply access the service via a weblink in the WebXR enabled browser (like Google Chrome or Firefox)

External software dependencies:

- **Other software components running on the same or different server:** Orchest.js (see 5.4) and the MCU (see 4.9).
- **Ubuntu OS** - (min version 18.04, https://www.ubuntu.com/)
- **NodeJS** - (min version 12.16.2, https://nodejs.org/en/)
- ***yarn*** (https://yarnpkg.com/)

Installation guide Server:

- After installing all dependencies each server component has its own respective install routine:
  - cd ~/git/togethervr2 && yarn install
  - cd ~/git/signalmaster && npm install
  - cd ~/git/orchest.js && npm install
- Each server component can simply be started via node.js:
  - cd ~/git/signalmaster && export NODE_ENV=production && node server.js
  - cd ~/git/orchest.js && npm run dev
  - cd ~/git/togethervr2 && sudo npm start

Installation guide Client: The TogetherVR client is provided as a web service and a web app. No installation of the software itself is needed as it runs in the web browser. It runs in combination with peripherals: Oculus Rift, Webcam, microphone, which need to be installed (see the respective vendor install documentation, if needed).
To see if a webcam and/or a microphone is supported, see if it is shown in the following page: https://webrtc.github.io/samples/src/content/devices/input-output/.
Even though the client can be used as-is, the capture module should be installed on the user's machine to ensure the proper format is streamed to the MCU. See 2.4 for more details.

Client dependencies:
TogetherVR further requires a powerful laptop which runs Firefox and preferably Windows (to run any OpenXR enabled VR HMD).

Inputs, outputs, configuration: Users can create/join a TogetherVR session via the web portal. TogetherVR is designed to automatically work with the TogetherVR client, via a Socket.IO/websocket interface.

Logging, metadata: Messages exchanged between external sources and within the web client are logged to the browser's developer console. This includes the state of the application at the moment those messages are exchanged and how they change the state.
Furthermore the Firefox browser allows for extensive Webrtc logging:
https://wiki.mozilla.org/Media/WebRTC/Logging
To instantiate this logging in firefox you can follow this procedure:
Open cmd.exe – windows console
set MOZ_LOG=webrtc_trace:5,timestamp,sync
set MOZ_LOG_FILE=C:\temp\log.txt
"C:\Program Files\Mozilla Firefox\firefox.exe" (start firefox)

6.2 Unity-based player

Description:

Unity-based player capable in charge of the reception, integration and presentation of all available streams and VR content for the envisioned VR-Together scenarios. It also supports various types of interaction and synchronization features.

Links:

- Demo Video: https://www.youtube.com/watch?v=8lGo1NPYwuA

Evolution:

Table 22. Detailed Delta Table – Unity-based Player

	Status Year 0	**Status Year 1**	**Status Year 2**	**Status Year 3**
TRL	TRL0 (Not available at the start of the project)	TRL6	TRL7	TRL7
Features	N/A	- TVM 1.0 Rendering - P2P streaming	- TVM 2.0 Rendering - Orchestrator based streaming - Stereoscopic video live stream - Multi-threaded pipelines - GUB Integration	- SUB Integration - Socket.io data streams - PC pipeline - WebCam pipeline - Spectator mode - PC-MCU Integration - Multi-threading improvements to enable multiple encoding/decoding - Networked interactions - Voice commands
KPIs	N/A	- 2 Users support	- 4 Users support	- 5+ Users support

Features:

- Support for different HMD in the market, either based on OpenXR or proprietary technologies.
- User sessions management.
- Different kinds of experiences based on scenes. (HoloMeeting, Broadcasting, Gaming, etc)
- Multi-User support.
- Spatialized Audio 3D.
- Multiple user representations: 3D Avatar, 2D WebCam Video, VolumetricMesh (TVM), Volumetric Point Cloud.
- Spectator mode.
- Voice commands.
- Audio chat communications between users in a session.

- Interactions with the scenario.
- Interactions between users.
- Capability to implement new scenarios and experiences easily.
- Possibility to use different communication protocols: Dash or Socket.io.

Performance:

The table below represents an example of a performance table with different combinations of representations in the same session. The performance data was recorded with a computer with the following specifications:

- Intel(R) Core(TM) i7-10750H @ 2.60GHz 2.59 GHz
- NVIDIA GeForce RTX 2070
- RAM 16.0 GB

The tests were executed with a RealSense (RS) sensor for the single point cloud representations. The computer used to grab the performance data acted as a "No representation" user and as a 2D Video user using the integrated webcam.

Table 23 Performance of VRT Unity Client

Users	Representation									CPU (%)	GPU (%)	RAM (MBs)
	No Repre.	2D Video	3D Avatar	TVM	S. PC (RS)	S. PC (K4A)	Synth. PC	Full PC	Spectator			
2	1				1					8,63	63,35	801,79
3	1				2					9,84	64,55	806,16
3	1				2					10,78	66,64	807,61
4	1				3					13,70	66,41	822,60
4		1			3					15,23	66,35	838,03
5		2			3					18,31	65,82	812,08
5	1				4					13,23	63,21	833,45

Licensing:

The Unity Player is going to be a Proprietary Code, proprietary License (TBD)

The dependencies of it could be encapsulated:

- Simple Point Cloud Capture: Permissive open source license, such as MIT, BSD or Apache.
- TVM Transmission: Apache License v2.0
- Point Cloud Encoding & Decoding: Permissive open source license, such as MIT, BSD or Apache.
- UMTS - User Manager Test Simulator: IP-owned, proprietary License.
- Dash Pipeline: IP-owned, proprietary License.
- FFmpeg.Autogen: GNU GPLv3

Source Code / executable Location:

The Unity Player repository with all the source code and components can be found at https://baltig.viaccess-orca.com:8443/VRT/nativeclient-group/Testbed (will be migrated once the public forge will get open).

Deployment Documentation:

Installation & Deployment instructions:

The Unity Player documentation of how to install and deploy it can be found at https://baltig.viaccess-orca.com:8443/VRT/nativeclient-group/Testbed

Internal Project dependencies:

The full list of dependencies is a combination of almost all the other components and depends on which of them are used on every specific case and pipeline. So we can inherit the dependencies from each other component:

- Simple Point Cloud Capture
- TVM Transmission
- Point Cloud Encoding & Decoding
- UMTS - User Manager Test Simulator
- Dash Pipeline

External dependencies:

- FFmpeg.Autogen: https://github.com/Ruslan-B/FFmpeg.AutoGen
- SPEEX: GNU project

7. INTEGRATION & SOFTWARE PACKAGING

We follow a 2-pipeline architecture for Native (TVM+PC) and WEB. This chapter outlines the integration and packaging of components within those pipelines.

7.1. Integration Overview

7.1.1. Native pipeline

The native pipeline integrates all the components required to build the collaborative platform which, in the end, allows users to participate in the VR-Together experience. End-users are captured in real-time and are represented realistically in 3D through volumetric video streaming, in a fully operational virtual environment. The native pipeline scenario allows to consume video in different formats (volumetric, traditional 2D, stereoscopic, 360) which are delivered On-Demand or Live, which for this last one offers the possibility for the users to interact with the actors.

In order to witness the social experience, the native pipeline platform uses a set of hardware devices and components for which a high-level description is provided in Figure 7.1.1, also provided as a reference layout in deliverable D5.5. The diagram is divided into three main set of hardware and software component entities:

- The user
- The live presenter
- The orchestrator
- The transmission servers

The user node includes i) the hardware needed to capture the volumetric video; composed from 1 to 4 RGBD cameras, with their corresponding (optional) reconstruction machines (NUCs units) and the “Volumetric Video Processing Component” which is in charge of generating, processing, encoding and transmitting the user representation to the server, and ii) the “Rendering Machine” with the player in charge of displaying the virtual environment to the HMD.

The live presenter node includes a set of hardware devices required to transmit a live stream to the end user rendering machine. It is composed of a stereoscopic 180 camera, a microphone and speakers which are specifically used for the interaction between users and the presenter. Finally, a “Live Presenter Processing Station” running a dedicated Unity-based interface allows the live presenter to receive the volumetric video of the end users (with the corresponding audio) and to allow the end users to visualize, listen and interact with the live presenter.

The orchestrator is in charge of managing the communication sessions over the VR-Together platform by creating the virtual rooms, dispatching session events, setting-up the connections and handling the data flow entry-points between all connected end-users as well as the live presenter. It also serves as the reference time clock necessary to make media synchronization consistent across all connections.

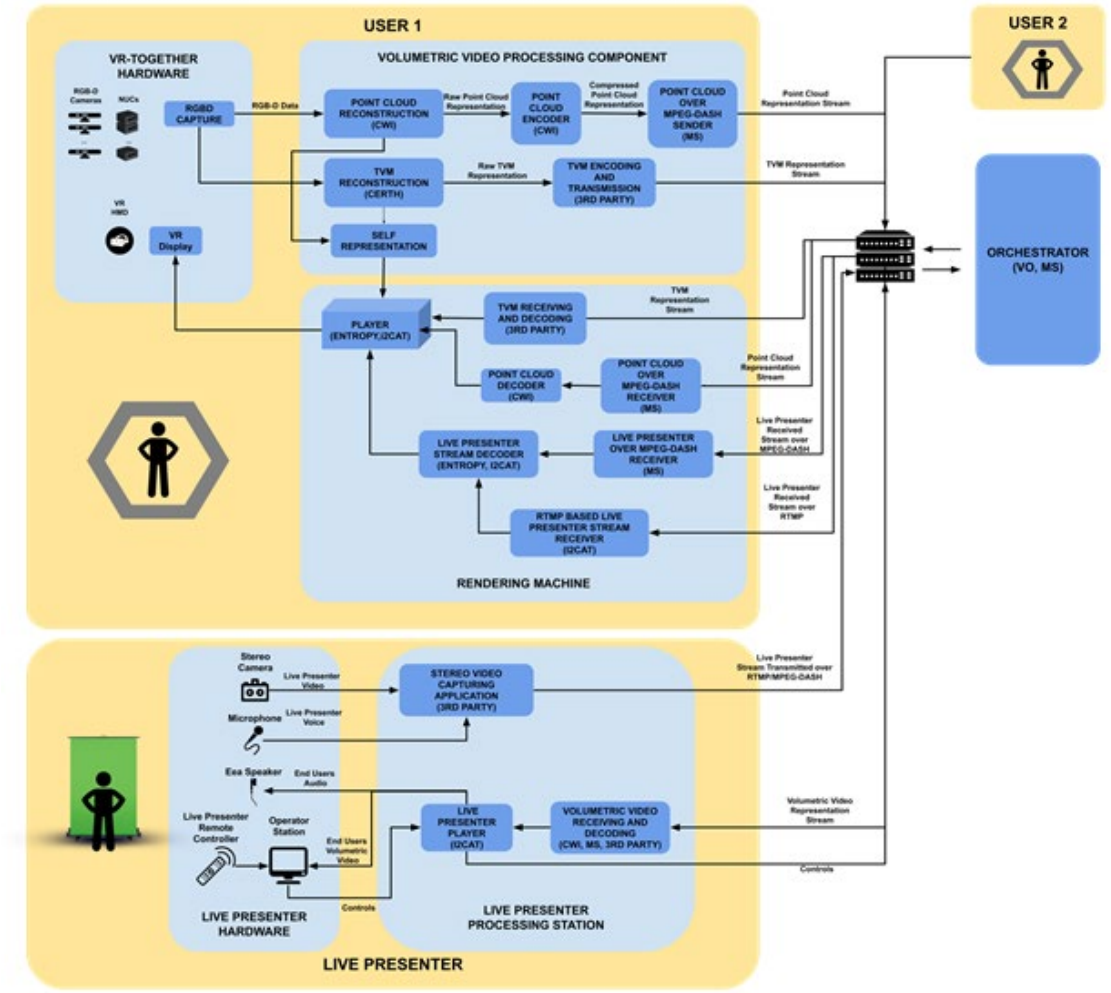

Figure 6. Native pipeline reference layout [D5.5]

The integration of the components included in the native pipeline has followed a continuous and iterative integration process described in deliverable D2.7 in order to build a modular integrated native pipeline which supports both capturing variants (Point Cloud and TVM).

Considering the current TRL of the technical components, and considering continuous development improvements, the Native pipeline can be considered as a complete E2E software platform, which could be considered a potential exploitable product. All the components used in the platform have been integrated with each other according to the scheme shown in Figure 6 and Figure 7, based on the high-level component architecture definition of the platform which is published in the D2.3 deliverable. Figure 7, shows a low-level diagram describing the integration between components.

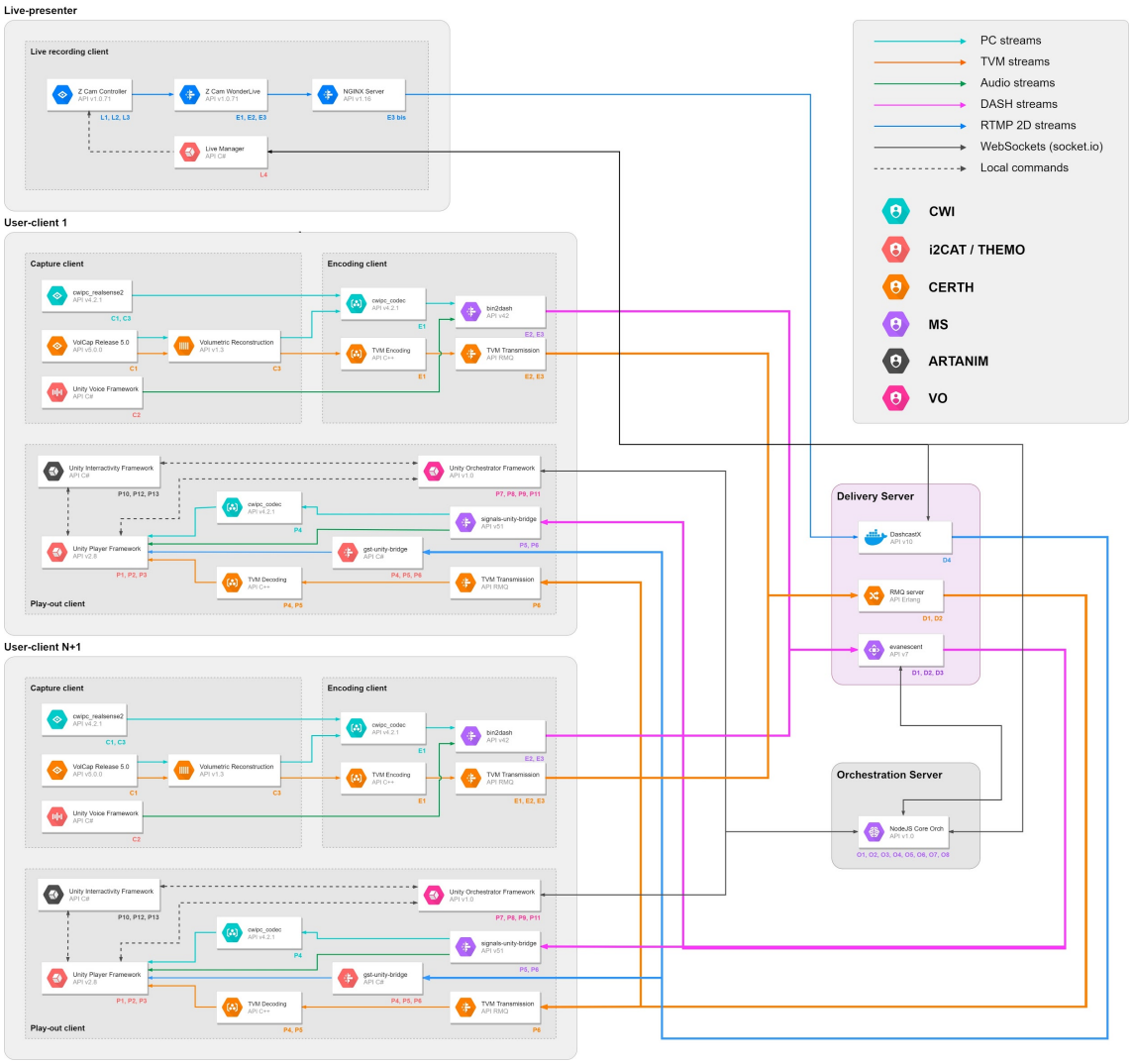

Figure 7. Low-level integration diagram of PC and TVM components in native pipeline [D2.5]

TVM Pipeline

The end-product of the native TVM pipeline is the distribution of a real-time 3D time varying mesh inside a graphics environment like Unity, in order to be consumable from third party software. This particular pipeline consists of the following software components/packages:

Software components:

- **Capturer:** Parameterizable acquisition of RGBD data from Intel RealSense D415 sensors and streaming these data to a server. The configuration of the parameters is realized from the GUI of a different software package (Volumetric Capture software). Capturer also uses a built-in module to trigger the sensor via synchronization cables, ensuring the simultaneous RGBD capturing between different Capturers.
- **Reconstruction:** Parameterizable real-time 3D reconstruction. Responsible for receiving the fused 3D information of Volumetric Capture and generating the TVM. There is also the utility of uploading the TVM to a server, making it accessible from third-party software. Offers three different voxel grid resolutions and different shaders for the rendering of the TVM.

PC Pipeline

The end-product of the native PC pipeline distributes point cloud representations of users to the various instances of the Unity playback engine. This particular pipeline consists of the following software components/packages:

Software Components:

- *Cwipc_util*: Infrastructure to operate on point clouds in different components and across different implementation languages without having to cater for memory management, representation issues and such.
- *Cwipc_realsense2*: simple point cloud capturer, can capture and fuse point clouds from RGBD data obtained from up to 4 Intel RealSense D4xx cameras attached to a single high-end reconstruction computer.
- Cwipc_codec: compression and decompression of point clouds for transmission or storage. Allows downsampling of point clouds for size reduction.

7.1.2. Web Pipeline

The web pipeline consists of multiple server components and a capture module. We can split the web pipeline into 3 core components: capture client, web server and MCU server (see section 7). Together the three components form a coherent end-to-end pipeline over the web. As with any web service the heart of the system is the central server instance. In detail we can separate the server into 3 individual web server instances, namely:

- TogetherVR web service: The TogetherVR web service is the main entry point for a client. It offers a web page to be accessible for a client to download all necessary resources and logic to create, display and facilitate the VR experience to the user. This also includes the connection to the orchestrator and Signalmaster.
- TogetherVR orchestrator: The orchestrator is mainly a message forwarding entity that allows for the client to get all session related information and metadata necessary to execute and maintain a running VR communication experience.
- WebRTC Signalmaster: The Signalmaster is a standard WebRTC component (open source and not developed in the context of VRTogether , https://github.com/simplewebrtc/signalmaster), that is necessary to connect multiple clients via dedicated peer-to-peer audio and video connections.

In the following we explain how the different components interact and integrate.

- A client is running the dedicated web capture module (currently this results in the captured RGB+depth data to be displayed on the screen and grabbed via screen capture by the web client for further processing and transmission).
- Further the client is running a modern VR-enabled web browser (i.e. Firefox or Chrome) that accesses the TogetherVR web service.
- Based on a list of existing sessions the user has the choice to select or create a new VR communication experience session.
- On selecting a session, the browser will load all necessary resources from the web server and executes the client JavaScript code.
- Following the code, the client will create a constant socketc.io connection to the TogetherVR orchestrator which will on first connect send all relevant data related to the communication experience session and send updates to the client on any change.

- With the help of the information of the orchestrator (e.g. about other clients in the session) and the Signalmaster, the client will set-up a WebRTC peer-to-peer connection with all other clients.
- After all WebRTC connections have been made, the user can enter the VR mode (given he/she has connected the VR HMD to the computer) and fully enjoy the VR communication experience.

Alternatively given an MCU is provided, the client will not set-up WebRTC peer-to-peer connection with all clients but only an upload and download WebRTC connection with the MCU (facilitated by the TogetherVR orchestrator).

7.2. Packaging & Deployment Overview

7.2.1. Native pipeline

The native pipeline platform is built on several components that consist of a large set of modules. All these entities are correctly defined and projected by the definition of the high-level component software architecture published in the deliverable D2.3, which gives the design of the reference model of the VR-Together platform.

On top of that, a continuous effort in terms of software packaging has been followed to simplify and ensure the integration of the technology. This is in fact characterized by the Native pipeline integration architecture providing the software package layout.

The next goal of this integration follow-up consisted in automatizing the deployment of all of the technologies involved in the native pipeline with a specific deployment framework (which is extensively documented in the D2.7 deliverable section 4). In fact, it allows to easily handle the technology packaging of the native client (which is made up of the Capture, encoding and playback components) and provides a conformal process to build a wWndows installer embedding the whole VRT native client software and dependencies.

As a result, for each major release of the native-client software, an end-user Windows installer is built so that it allows installing the project executable and its dependencies in a very simple way. The use of the deployment framework of the project requires the respect of several stages that are precisely described in D2.7:

The windows machine preparing to receive the project should have git and python 3 installed.

The environment variables redirecting to the programs necessary for the proper functioning of the project must be properly configured.

The project is currently hosted on the Viaccess-Orca intenral GitLab; its migration to a public forge is underway. For the moment, it is therefore necessary to have a GitLab account to access the folder: https: //baltig.viaccess-orca.com: 8443 / VRT (see D2.7). It gives access to the main deployment script that will automatically install the necessary libraries. A script to verify the installed software is also available in order to monitor operations.

Once the deployment framework is properly installed, it is possible to generate the Windows installer (see Figure 8 and Figure 9).

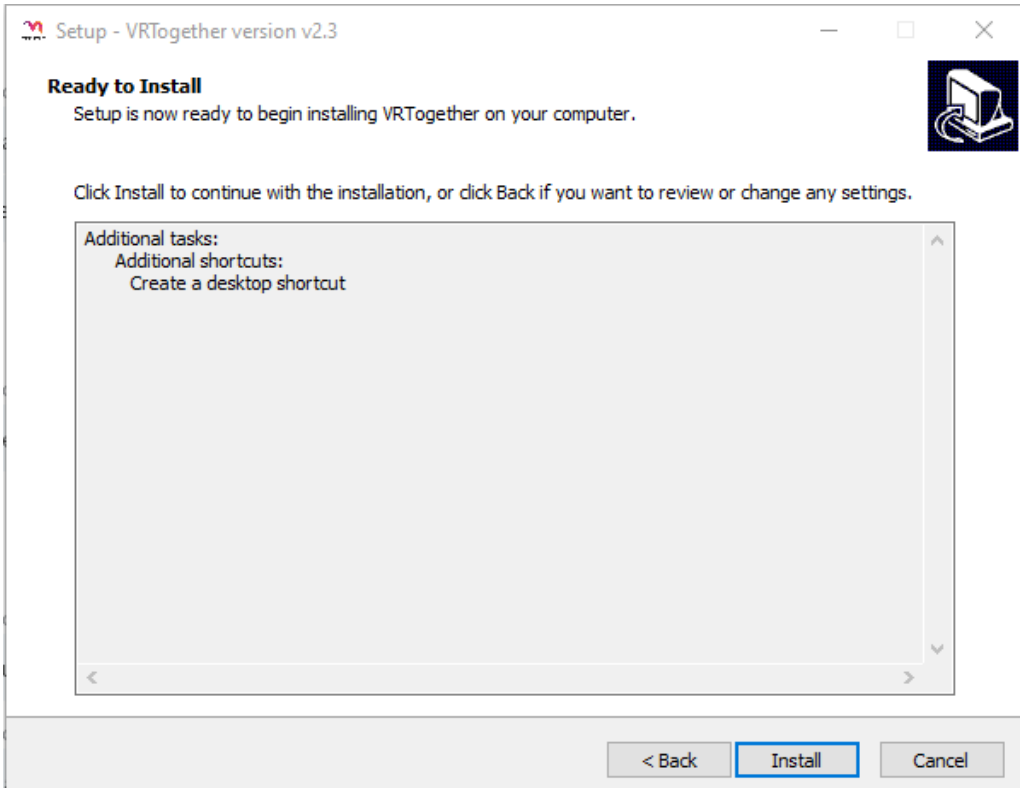

Figure 8 Installer of Native Client – Setup

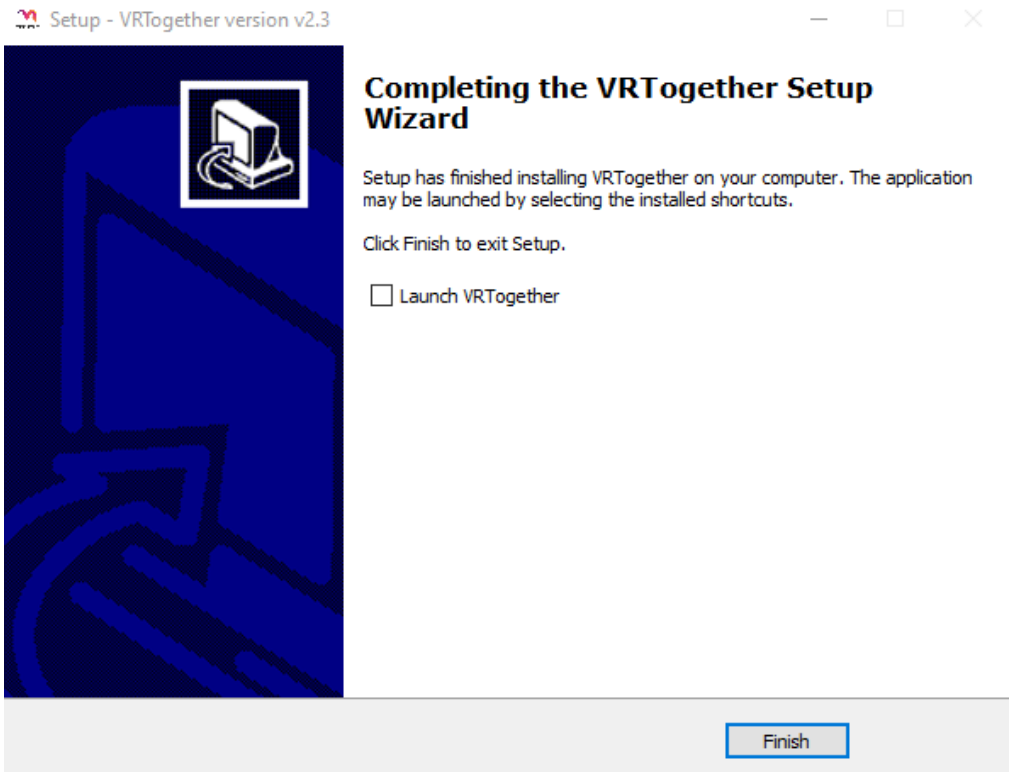

Figure 9 Installer of Native Client – Completion

Hardware requirements:

The deployment of the complete native pipeline requires a "capable" Windows computer, with at least four independent USB 3.1 channels. Additionally, for the renderer a powerful graphics card may be required, and for the HMD more USB interfaces may be required. However, a scaled down version of the pipeline (single camera, webcam capture, no HMD) can be run on commodity hardware.

TVM Pipeline:

Although the Windows installer is embedding the necessary client software dependencies and configuration, the TVM pipeline remains not fully handled, as multiple manual configuration steps are necessary to setup an operational TVM node.

Software packages:

- Volumetric Capture: This particular software is pivotal for the entire pipeline. It handles the receiving and the synchronization of multiple Capturers. After the fusion and the generation of the new 3D data, a different module uploads these data to a server. This software package tunes the capturing parameters of the Capturers via a GUI at runtime.

By exploiting all the aforementioned software packages/components we create a unified real-time TVM generation pipeline. In our case, the basic consumer of this pipeline is a managed application inside a virtual environment.

Deployment:

The deployment of the native TVM pipeline is dependent on the different software packages/modules of the pipeline. The different steps for the deployment of the native TVM pipeline are listed below:

Supposing we have multiple NUCs and we connect an Intel RealSense D415 to each NUC. Install the required Windows Service along with its resources that handles the utilities of the sensor. We do not need to start the Capturer manually as the software is managed from a Windows Service running in the background.

- Install the Volumetric Capture package on a different computer. This package controls the acquisition and the streaming of the Capturers.
- Install Volumetric Reconstruction on a different computer.
- Install a RabbitMQ server on a computer. The simplest solution in order to decrease the number of the computers is to install the server on the same computer as Volumetric Capture.

For all the aforementioned software to communicate via the network, an extra port configuration is needed.

A step-by-step detailed description of the multi-view capturing platform is provided on CERTH's' GitHub page: https://github.com/VCL3D/VolumetricCapture/wiki

7.2.2. Web Pipeline

The Web Pipeline offers a lightweight Social VR service as a real-time E2E communication system. Based on web technologies, such as HTML5 and JavaScript as well as WebRTC and WebGL APIs, combined with video coding and distribution standards like DASH, it provides an efficient VR based conferencing system. The setup is light and affordable, as the capturing component uses a single consumer-grade depth camera for the real-time capture system. It can be easily installed and configured at the end-user level. Combined with other components developed in the project (stream encoding and transport, orchestration …) this Lightweight Social VR service is particularly adapted for the deployment of VR conferencing, training and Social VR experiences.

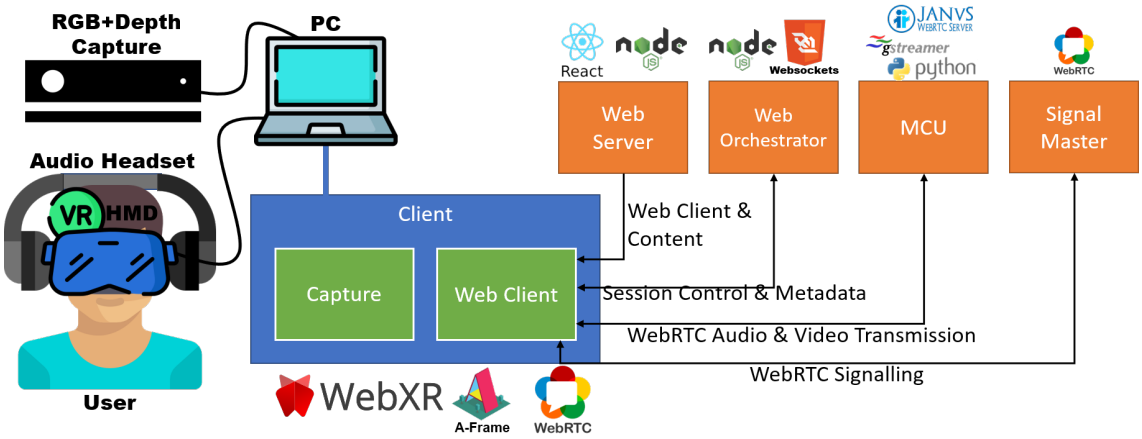

Figure 10. Overview of the Web Client Architecture and Setup

Currently the Web Pipeline follows the following packaging & deployment:

- Capture client
  - Simple Windows executables (.exe files) including all dependencies and python environment to run the client (only requires pre-installed RDA software)
- TogetherVR web service
  - Node.js JavaScript code package
  - Needs installing of node server and comes with predefined scripts to install and execute
- TogetherVR orchestrator
  - Node.js JavaScript code package
  - Needs installing of node server and comes with predefined scripts to install and execute
- WebRTC Signalmaster
  - Node.js JavaScript code package
  - Needs installing of node server and comes with predefined scripts to install and execute
- WebVR-MCU
  - Delivered as fully Dockerized image

8. CONCLUSION

This deliverable provides a detailed description of the final version of software components of the VR Together project. For each WP3 task, a set of components has been provided, along with concrete instructions, testing opportunities, and notes on further integration.

<END OF DOCUMENT>