WEB BASED CONFIGURATION

Capture and delivery

TNO innovation for life

This is a basic web-based configuration employed in VR-Together. For the capturing process, the Kinect V2 or the Realsense D415 camera is used. For the streaming process, the WebRTC framework is used.

The foreground/background segmentation is based on the depth image. A reference image (RGB + depth) is captured before a user is present in the scene. Next, each frame of the captured video is processed. The depth image of each frame is compared to the reference frame to determine the foreground, i.e. the image of the user. This depth map is cleaned up, and applied to the RGB image to retrieve the users RGB foreground image. The sequence of images, retrieved in this way, is offered as a virtual webcam to the WebRTC framework.

The delivery is currently based on the SimpleWebRTC framework. The Media Capture API is used to retrieve both the virtual webcam and the HMDs microphone. Sessions are set up between the various users under supervision of the Orchestration component. The media is exchanged in a peer-to-peer fashion between the clients involved in the session.

Additional developments in this configuration involve:

Applying the captured RGB+D into a Point Cloud that is overlaid on the captured users position, to show a self view;

Streaming the captured depth images to other clients to allow a 3D image reconstruction at the receiving side;

Play-out

TNO innovation for life

For rendering, the A-Frame framework is used, which is based on WebVR. In this framework, various objects can be combined into a single VR experience. In the basic scenario, a 360 degree photo is used as a virtual environment. In this environment, 2D planes are placed at the other users' positions, and their video (containing only their foreground) is displayed on these 2D planes. The accompanying audio, which is synchronized with the video by using the WebRTC framework, is also placed at the users' positions.

Object-based audio is supported through the integration of the Google Resonance framework. For display of 3D objects, i.e. self-representation of a user or representation of other users, custom shaders that support the employed RGB + depth format for display in a 3D fashion have been developed while the 3D room model object is supported by A-Frame by default.

European Commission

This project has been funded by the European Commission as part of the H2020 program, under the grant agreement 762111

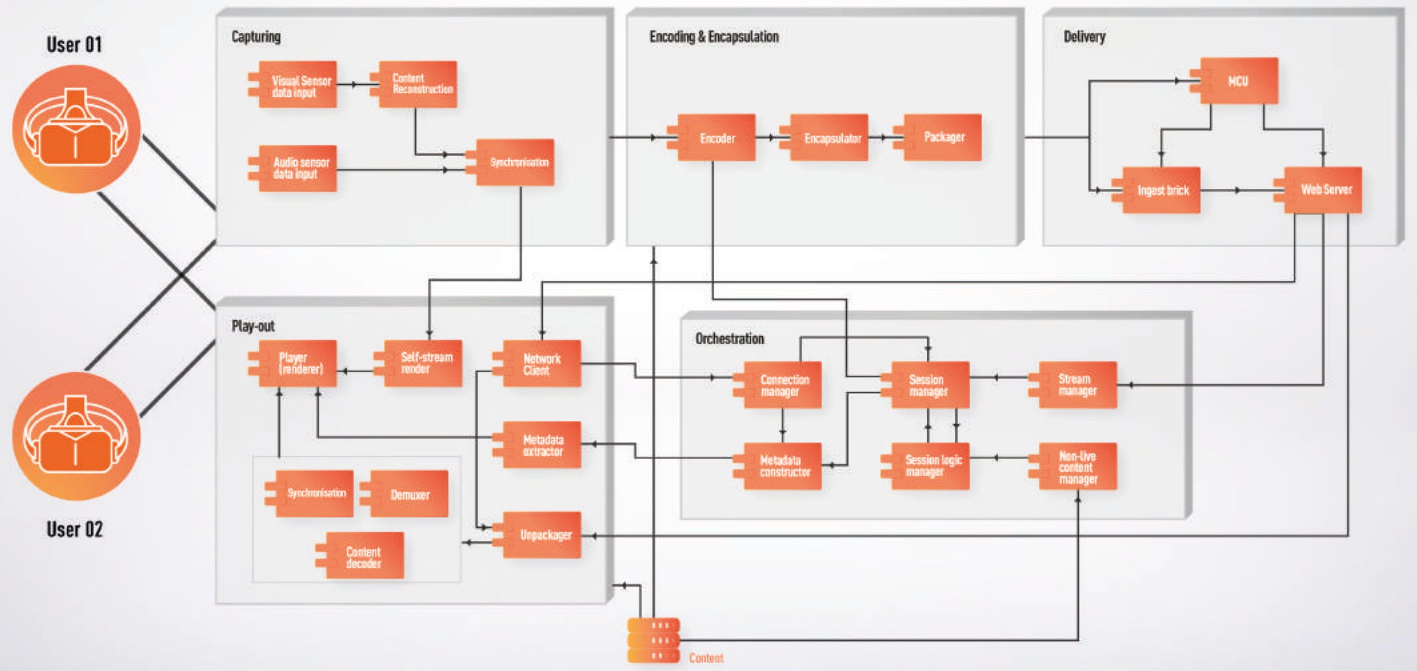
FOLLOW US @VRTogether_EU www.vrtogether.eu

VRTogether

[7]https://www.simplewebrtc.com/
[8]https://aframe.io/
[9]https:///developers.google.com/resonance-audio/

PLATFORM ARCHITECTURE

VRTogether



COMPONENTS DESCRIPTION

The software architecture of the end-to-end VR-Together (i.e. from capture to consumption chain) is comprised of 5 main components. These components serve as a basis of the integrated platform structure, and each one of them is a conceptual entity related to a general task within the end-to-end communication system.

The Capturing component captures / produces the visual and audio data, performs content reconstruction tasks and synchronizes the captures audio and video signals.

The Encoding & Encapsulation component encodes, encapsulates and packages the audio and video signals received from the previous component into one single stream.

The Delivery component makes this content available for consumption on the network.

The Orchestration component provides end-users with the information necessary to initiate a communication session over the VR-Together platform.

The Play-Out component is responsible for rendering and presentation of the immersive contents. It includes the contents for the shared virtual scenario and end-users' representation. It includes modules for unpackaging, demuxing and decoding and synchronizing the audio-visual content.

TIME VARYING MESH CONFIGURATION

Capture and delivery

A full-body 3D user representation is reconstructed (real-time) in a 3D mesh form, i.e. a set of connected vertices and polygons applying UV texturing. The capturing of color and depth information is realized by multiple calibrated RGB-D cameras including the processing of spatial mapping and alignment of the captured 2.5D data into 3D, resulting in a watertight 3D TVM which is reconstructed in real-time. TVMs contain multiple RGB textures, one per RGB-D device, as well as the 3D geometrical information of the mesh, i.e. the vertices, polygons and the corresponding normal vectors.

Due to the data intensive nature of TVMs, compression for fast and efficient transmission via web-based scenarios is a necessity. TVMs are compressed both on the level of textures as well as in that of the 3D mesh. 3D data is compressed employing state-of-the-art methods (e.g. Draco, Corto) while for the textures the TurboJPEG library is used. Subsequently, the compressed data are serialized (packaged) and distributed by a RabbitMQ server, allowing for the VR-Together player to connect and retrieve the transferred information in real-time.

Play-out

Following the information retrieval at the player's side, the packaged TVMs are deserialized and decoded prior to the final step, namely the TVM rendering. To this end, the transferred RGB textures are blended to colorize the 3D mesh within a unity 3D scene. In particular, the textures are weighted based on the confidence given by the angle difference between the normal of the corresponding polygons and the camera view axis, thus, higher angles correspond to lower confidence.

The audio is mixed locally to reflect the position of every audio object, and is affected by the position and rotation of each user and element in the virtual environment.

Such a practise provides a photo-realistic 3D environment where 3D assets, stereoscopic billboards and 3D user representations are seamlessly presented in a synchronized manner.

Each user starts a session where he or she can socially interact with the other person, being each one of them able to experience a different part of the story via their Head Mounted Displays (HMDs).

[1]https://github.com/google/draco
[2]https://github.com/cnr-isti-vclab/corto
[3]https://libjpeg-turbo.org/

VRTogether

POINT CLOUD CONFIGURATION

Capture and delivery

A volumetric representation of the user is captured in real-time as a Point Cloud, i.e., a set of independent points in space, each associated to its spatial coordinates and an RGB triplet indicating the color of the point. The capture is performed by a signal acquisition module, developed by CWI, that merges the data captured by multiple texture + depth Intel RealSense sensors, positioned around the user. The capture module outputs a raw Point Cloud frame and the corresponding capture timestamp at a variable frame rate.

The raw Point Cloud would require extremely high bandwidth to be transmitted over bandwidth-limited networks. Therefore, in order to enable delivery of the Point Cloud signal, each Point Cloud frame is delivered as input to a lossy encoder, developed by CWI, which reduces the bitrate needed to represent the volumetric signal, by removing visual redundancy. The output of the encoding module is an encoded Point Cloud frame. The capture timestamp is also included in the encoded stream to enable their synchronized presentation.

Finally, each encoded Point Cloud frame is packed into a DASH segment (i.e., m4s file), by an application based on the Signals framework, developed by Motion Spell. The m4s files corresponding to temporally consecutive Point Cloud frames are stored on a DASH server, ready to be delivered via HTTP.

Play-out

The incoming packaged Point Cloud frames are handled at the receiver side by using GStreamer framework. An HTTP connection is established to first obtain the MPD and then request the DASH segments composing the stream. From the DASH segments, the Point Cloud structure is extracted and sent to the Point Cloud decoder. The inserted timestamps at the capture side are extracted and used by a developed synchronization module to allow the presentation of all incoming data in a synchronized manner.

A native Point Cloud renderer module was developed to enable user representation in that format. It is based on the wrapper for Unity of Intel RealSense SDK 2.0, which transforms the raw data from the RGB Camera and the Depth Sensor into a Unity 3D mesh.

[4]https://github.com/gpac/signals
[5]https://gstreamer.freedesktop.org/
[6]https://github.com/IntelRealSense/librealsense/tree/master/wrappers/unity

VRTogether