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*

D3.2 Report on the initial software versions

Revision: 10

Authors: Omar Niamut (TNO), Argyris Chatzitofis (CERTH), Shishir Subramanyam (CWI), Romain Bouqeau (Motion Spell), Simon Gunkel, Martin Prins, Frank ter Haar (TNO), Pascal Perrot (Viaccess)

Delivery date: M3 (31-12-17)

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

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

Abstract: Report on the initial software versions, including their applicability for use in the pilots

Revision	Date	Author	Organisation	Description
0.1	07-12-17	Omar Niamut	TNO	TOC and first contributions from partners
0.2	10-01-2018	Omar Niamut	TNO	intermediate contributions from partners
0.3	15-01-2018	Omar Niamut	TNO	final contributions from partners
1.0	22-01-2018	Omar Niamut	TNO	Ready for approval

REVISION HISTORY

EXECUTIVE SUMMARY

In deliverable **D3.2**, we report on the initial software versions, including their applicability for use in the pilots. This information should help WP2 partners to start the assessment of compatibility aspects that they may encounter when integrating components for the first pilot; WP3 partners with a clear view on the initial developments of the components towards software version 0; and WP4 partners with the information to make a technical assessment of the foreseen work towards realizing the first pilot. This info is to be reported as part of T3.5 and ideally, should guarantee the WP2 partners an easy integration of the modules, allow the WP4 partners to anticipate bottlenecks for pilot deployment, as well as provide a basis for testing and evaluation guidelines.

CONTRIBUTORS

First Name	Last Name	Company	e-Mail
Martin	Prins	TNO	Martin.Prins@tno.nl
Simon	Gunkel	TNO	Martin.Prins@tno.nl
Frank	Ter Haar	TNO	Frank.TerHaar@tno.nl
Romain	Bouqueau	Motion Spell	romain.bouqueau@motionspell.com
Argyris	Chatzitofis	CERTH	tofis@iti.gr
Shishir	Subramanyam	CWI	S.Subramanyam@cwi.nl
Juan Antonio	Nuñez	I2CAT	juan.antonio.nunez@i2cat.net

CONTENTS

TABLES OF FIGURES

LIST OF ACRONYMS

Acronym	Description
AAC	Advanced Audio Coding
API	Application Programming Interface
BIFS	BInary Format for Scenes
CSS	Companion Screen Synchronization
CUDA	Compute Unified Device Architecture
DASH	Dynamic Adaptive Streaming over HTTP
DVB	Digital Video Broadcasting
FGBG	Foreground-Background
HD	High Definition
HEVC	High Efficiency Video Coding
HLS	HTTP Live Streaming
HMD	Head-Mounted Display
ICP	Iterative Closest Points
IR	Infrared
JPEG	Joint Photographic Experts Group
LoD	Level of Detail
MPEG	Moving Picture Experts Group
PCL	Point Cloud Library
PSNR	Peak Signal-to-Noise-Ratio
RTC	Real-Time Communications
RDA	Remote Data Access
RGB	Red-Green-Blue
RGBD	Red-Green-Blue and Depth

RTMP	Real-Time Messaging Protocol
RTP	Real time Transport Protocol
SVG	Scalable Vector Graphics
VR	Virtual Reality

1. INTRODUCTION.

1.1. Purpose of this document

The purpose of this document is to inform the consortium about the initial software that is made available from WP3 partners at the start of the project.

1.2. Scope of this document

In D3.2, we report on the initial software versions, including their applicability for use in the pilots. This information should provide WP2 partners with a starting point for assessing compatibility aspects that they may encounter when integrating components for the first pilot; WP3 partners with a clear view on the initial developments of the components towards software version 0; and WP4 partners with the information to make a technical assessment of the foreseen work towards realizing the first pilot.

1.3. Status of this document

The first version of D3.2 is delivered in M3, at the beginning of the project, capturing as much information as possible for the system in development coming from the different actors within VR-Together.

1.4. Relation with other VR-Together activities

This info is to be reported as part of T3.5 and ideally, should guarantee the WP2 partners an easy integration of the modules, allow the WP4 partners to anticipate bottlenecks for pilot deployment, as well as provide a basis for testing and evaluation guidelines.

2. OVERVIEW OF INITIAL SOFTWARE VERSION FOR CAPTURE AND RECONSTRUCTION COMPONENTS

In this section, information regarding the initial software versions of the components related to the Task 3.1 “Capture and Reconstruction” is given. In particular, a short component overview is provided, features are described, some performance indications are given and integration aspects are reported.

2.1.People 3D Capture

The people 3D capture component has a distributed capturing and centralized processing nature, comprising of multiple Kinect V2 sensors, multiple PCs (one for each sensor) and one central server collecting and processing the data (user point cloud generation). Each client streams RGBD data to the central PC at real-time rates through a local Ethernet interface. Aiming at high exchange rates, the locally captured data at the client side are encoded before transmission, following an intra-compression scheme to minimize transfer latency. In particular, the HD RGB data are compressed using standard JPEG compression, while a lossless algorithm is used for the depth data.

Features:

- RGB-D + Infrared streaming in real-time.
- Synchronization and spatial calibration information extraction for post usage.
- RGB-D + Infrared storage in compressed file format (.scnz)
    - Frame index
    - Color data
    - Color data timestamp
    - Color compressed buffer
    - Depth data
    - Depth data timestamp
    - Depth compressed buffer
    - IR data (optional)
    - IR data timestamp (optional)
    - IR compressed buffer (optional)
- Foreground Extraction
- Coloured 3D Point cloud

Performance: Real-time at 30 frames per second (depending on the frame rate of the RGB-D sensors)

Integration Aspects: C++ Static Library

2.2. People live 3D reconstruction

This component achieves real-time full-3D geometry reconstruction of people. The component methods have been implemented using CUDA, performing in real-time. Each sensor of a multi-RGBD-sensor setup produces a stream of spatially and temporally aligned color and depth frames, collected by a central processing server to reconstruct the user’s geometry and appearance on a per-frame basis. Moreover, a multi-texturing approach embeds appearance to the reconstructed geometry.

Features:

- Real-time time varying mesh production.
- Quality, thus production rate, parameterization.

Performance: Real-time at ~15 frames per second (depending on the visual quality we target)

Integration Aspects: C++ Static Library

2.3. People post (offline) 3D reconstruction

An end-to-end component for post reconstruction of human actor performances into 3D mesh sequences has been implemented, using the input from a multi-RGBD-sensor setup. This component, by offline pre-reconstructing and employing a deformable actor's 3D model to constrain the on-line reconstruction process, implicitly tracks the human motion. Handling non-rigid deformation of the 3D surface and applying appropriate texture mapping using the input from the multi-RGBD-sensor setup, it produces a dynamic sequence of temporally-coherent textured meshes.

Features:

- Quick-Post dynamic mesh production.
- Quality, thus production rate, parameterization.
- Output:
    - **Character Animation** using the template and motion capture data. (low quality, fast transmission rate)
    - **Enhanced Character Animation** using the template, motion capture data and dynamic texturing. (medium quality, medium transmission rate)
    - **Dynamic Mesh.** Animated mesh using the template and motion capture data deformed per frame based on 3D data. (medium quality, medium transmission rate)
    - **Enhanced Dynamic Mesh.** Animated mesh using the template, motion capture data and dynamic texturing deformed per frame based on 3D data. (high quality, low transmission rate)

Performance: Quick-Post procedure.

Integration Aspects: C++ Static Library

2.4. Video background removal component

The video background removal component consist of two modules. The first module does the real-time scene capturing using the Kinect V2 sensor that maps the color and depth images to shared memory on a client PC. The second module is an algorithm that performs the real-time foreground-background (FGBG) extraction using color and depth images from shared memory. The algorithm takes a single shot background and computes in each new frame the foreground objects that are in front of the initial background, such as persons and other objects. To integrate the output of this module with WebRTC, the background of the color image is removed and set to a specified color (green currently), and the resulting color images are piped to a RGB video stream.

Feature	Performance and support
Client device	Powerful PC or Laptop; We use MSI VR Ready gaming laptops with at least NVidia 980M graphics card and Intel i7 processor.
Target OS	Windows; For the Kinect V2 sensor capturing module
Resolution	960x540 images at 25 fps are preferred; the FGBG then drops about 1 frame per second on a low CPU load.
Static experience	Yes; users are standing or sitting. The FGBG module requires a static placement of the Kinect V2 sensor with a background and foreground object within 5 meters distance. After a replacement of the sensor, a new background image must be captured.

Integration aspects: Currently the real-time scene capturing and the Kinect V2 sensor run on Windows using RDA software (TNO Remote Data Access) to handle shared access. However, the FGBG module is a stand-alone platform independent python script that could also be integrated with other capture components either on client site or on the central PC in case of lossless compression for depth data. Furthermore, the background removal is not limited to the Kinect V2, but also applies to similar RGBD sensors.

3. OVERVIEW OF INITIAL SOFTWARE VERSION FOR ENCODING

In this section, information regarding the initial software versions of the components related to the Task 3.2 “Encoding” is given. The components described in this section aim to provide real-time encoding that is capable of significantly reducing the bandwidth requirements of transmitting data reconstructed from task T3.1. The overall objectives are efficient data reduction, low delay processing and high level of quality. The components available from different partners are the point cloud compression framework, end user real-time mesh encoding and signals modular platform.

3.1. Point Cloud Compression Framework

A framework for the compression of static and dynamic point clouds is available. The framework is open source and is available at (https://github.com/cwi-dis/cwi-pcl-codec). This framework includes an objective quality metric and currently supports three lossy codecs for point cloud compression. In addition it is integrated with the open source Point Cloud Library (PCL) (https://github.com/PointCloudLibrary), this makes a large number of point cloud processing functions available. The codecs provide feature lossy colour attribute coding using JPEG compression, inter prediction to exploit temporal redundancies, progressive decoding and a parallelized implementation.

The codecs available in the framework are all based on the octree data structure. This has been explained in Appendix I.

PCL: The codec in PCL created by Julius Kammerl et al [1] features an inter frame XOR based codec. PCL contains functions to perform the octree subdivision, traversal and search. They use a double buffered octree to store successive frames in order to perform inter prediction. The buffers are switched every time a new frame is loaded, as a result the current frame and the previous frame are always available in memory. The geometry or point coordinates and the attributes like colour are encoded separately while maintaining the bond between them. The codec in PCL features an inter prediction algorithm for point cloud geometry. This algorithm is based on the double buffered octree and the occupancy codes of octree nodes as shown in figure 1. An XOR operation is performed across the double buffer in order to reduce the entropy of the final bitstream and make statistical compression more efficient. In addition, at the leaf nodes an origin location is calculated at the lowest (smallest x, y and z) coordinates in the voxel, then the distances between each voxel local point and the origin is calculated. These distances are then discretized and coded using positive integers based on the specified precision. This quantization of point coordinates is the last step in lossy compression and a bitstream is generated. Next, a lossless statistical compression algorithm can be applied to the resulting bitstream to further compress it. As the voxel local point coordinates are stored as integers a range coder can be used on the bitstream. This algorithm is the integer arithmetic version of arithmetic coding.

This approach only compresses point cloud geometry and does not cover attributes like colour.

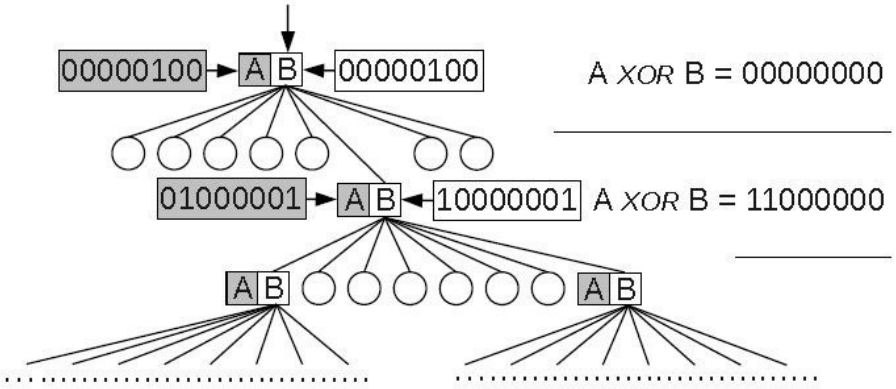

Figure 1: Differential occupancy coding of the double buffered octree [1]

CWI codec:

The CWI codec includes an intra frame coder that encodes geometry at fixed levels of detail by differentially encoding voxel local points from the PCL codec. This process is repeated at different pre-specified levels of detail so that the decoding process can be progressive and a final level of detail can be chosen by the decoder based on prevailing network conditions. A carry less byte range coder is then used on all supported LoDs for lossless entropy coding of the final bitstream. The range coder has been preferred over arithmetic coding as it operates at a byte level making it much faster on general microprocessors. The intra frame coder also encodes colours by performing a depth first traversal of the octree and scanning the colours to a structured JPEG image grid using a zig zag pattern to further exploit the correlations among co-located points. The overall schematic is shown in figure 2.

The CWI inter frame coder reuses the intra frame coder in combination with a lossy prediction scheme based on the iterative closest points (ICP) algorithm from PCL. This is shown in figure 3. Macroblocks are defined at a predefined number of levels above the final level of detail or voxel size. The inter frame coder identifies shared macroblocks in the previous frame and the current frame that are comparable and eligible for prediction. This eligibility is based on the macroblock point count and colour contrast changes. The ICP algorithm then identifies a rigid transform between shared macroblocks in the both frames and if the algorithm converges the transform is used as a predictor. The transform is stored as a rotation quaternion and three quantized translation components that occupy 16 bits each. The framework can optionally be used to calculate a colour offset between corresponding macroblocks across frames. Macroblocks can be uniquely identified and randomly accessed using a key k(x,y,z) containing three 16 bit integer values. Macroblocks that are coded using inter prediction are stored and transmitted as a structure with 18 bytes as shown in figure 4. An additional 3 bytes can be used to store the optional colour offset.

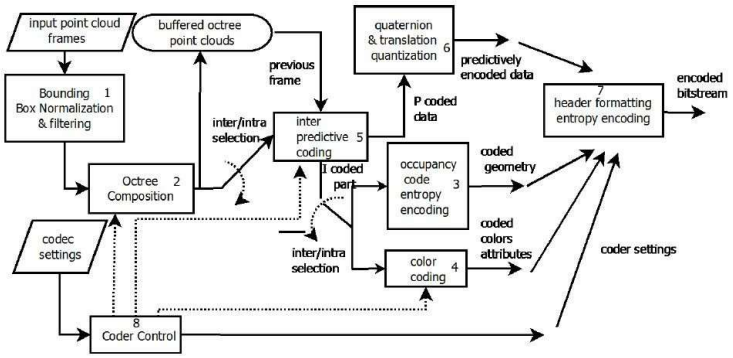

Figure 2: Schematic of the time varying point cloud compression codec [2]

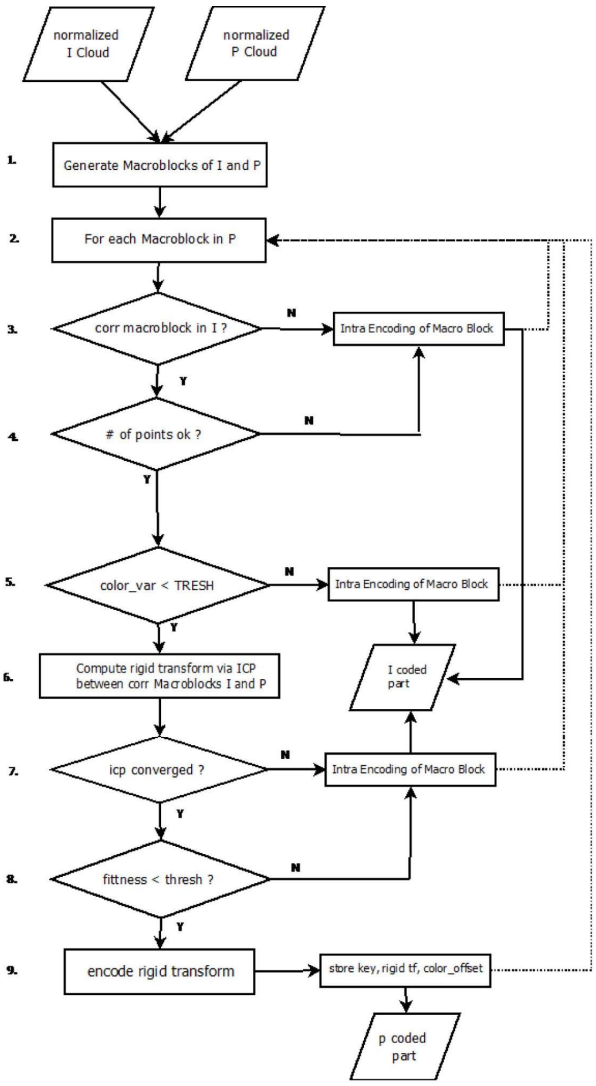

Figure 3: Inter frame prediction in the CWI codec [2]

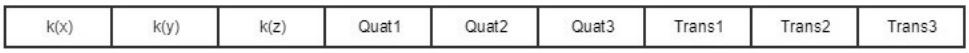

Figure 4: Memory Structure of a predicted macroblock in blocks of 2 bytes each.

The framework contains three codecs and a k-neighbour radius outlier filter. There are options to specify the point cloud coordinate resolution, voxel size, level of detail, colour resolution bits, macroblock size for inter frame prediction, JPEG quality and number of parallel threads to use. In addition the framework contains objective quality metrics (based on PSNR) to measure geometry and colour distortions.

3.2. End-user real-time mesh encoding

An encoding component has been developed with respect to the compression of live 3D reconstruction data. The geometry information (vertex 3D positions, normals and attributes, as well as connectivity) is compressed using OpenCTM (http://openctm.sourceforge.net/?page=about), allowing thus the 3D reconstructed data to be scalable to network conditions. An intra-frame static mesh codec, such as CTM, has been selected since the reconstructed 3D meshes are "time-varying" meshes (i.e. with variable number of vertices and connectivity along frames). Standard JPEG compression is employed for textures, due to its simplicity and very-fast performance.

Features: Time Varying Mesh compression/decompression

Performance: Real-time at 10-15 frames per second. From KB/frame: 100-300. Compression time: 60-120ms.

Integration Aspects: C++ Static Library

3.3. Traditional audio and video encoding

Traditional audio and video codecs rely on two mechanisms, i.e. discarding perceptually irrelevant components; and eliminating redundancies. These codecs better perform in lossy mode (i.e. the encoded signal is not identical to the source). Main used codecs in this category include the AAC codec family (2000-2010) for audio, and H264 (2003) and HEVC (2013) for video. These codecs are developed within the MPEG ISO standardization group.

The Signals toolkit offers encoding bricks for traditional audio and video codecs.

- In the project we plan to encode content using our Signals modular platform (http://signals.gpac-licensing.com).
- Currently we support many codecs including H264 and H265 encoders (x264, x265, Kvazaar, SocioNext, Nvidia, QuickSync).
- We plan to integrate some (Point Cloud or anything useful) encoders in the scope of the project (Point Cloud or anything useful).
- The Signals platforms allows to define "modules" that can transform the information in the proper way (in particular pre-processing or preparing the content for distribution).

- The global aim is to provide a common Signals tool for T3.2 and T.3.3 that would encode and distribute the content produced by T3.1 components.

Features: in the scope of VR Together we plan to focus on the following features:

- Audio: AAC codec (FFmpeg native, fdkaac or voaacenc) or any object-based codec that might be usable to serve our purpose.
- Video: AVC/H264 or HEVC/265 codecs. This list may be extended as new standards and encoder implementations are made available.

Performance:

- AAC usually offer a compression ratio of 20 (i.e. the compressed file is 20 times smaller than the source). Encoding is usually dozens of times faster than real-time, decoding is usually hundreds of times faster than real-time.
- H264 offers a compression ratio in the range of 20 to 50. Encoding is usually a bit faster than real-time, and decoding is several times faster than real-time.
- HEVC offers a compression ratio of 100. Encoding is usually slower than real-time and decoding is a bit faster than real-time.

Integration: the integration depends on:

1. Licensing consideration (for both the source code and royalty/license considerations).
2. Hardware considerations when relying on accelerators (GPU, CPU capabilities, etc.).

4. OVERVIEW OF INITIAL SOFTWARE VERSION FOR DISTRIBUTION AND ORCHESTRATION

In this section, information regarding the initial software versions of the components related to the Task 3.3 “Distribution and Orchestration” is given. The components described in this section aim to provide real-time distribution and orchestration of the encoded hybrid media. The components available from different partners are the GPAC and Signals distribution infrastructures, the Orchest.js media orchestration framework, the real-time 2-player interaction platform and the synchronization server.

4.1. GPAC for distribution

GPAC Multimedia Open Source Framework is a powerful worldwide project, established in 1999 in NYC, dedicated to the development of rich-media, video encoding and broadcast technologies. For distribution, GPAC has a extensible support of the MPEG-4 set of standards. This includes support for object-based media. The GPAC packager (mainly MP4Box, but also MP42TS) allows to capture a set of media, puts them in a single MP4 file, and distributes them with different formats (such as RTP, Apple HLS or MPEG-DASH). See http://gpac.io

Motion Spell plans to use and extend the GPAC open-source software for non-live use-cases.

4.2. Signals for distribution

The Signals module toolkit allows to re-use components and connect them together. Signals has some composition capabilities at the system level (e.g. synchronizing finely different media and forwarding them to the next module) but it doesn’t offer the level of control of GPAC. Signals supports the following outputs in the frame of the project: MPEG-DASH, Apple HLS, and real-time protocols such as RTP and RTMP.

Signals was designed for high-performance and controlled runtime environments. It can package hundreds of megabytes of data per second.

The integration in Signals is done through an easy C++ API. The API offers different levels of interaction whether one wishes to develop a module or an application.

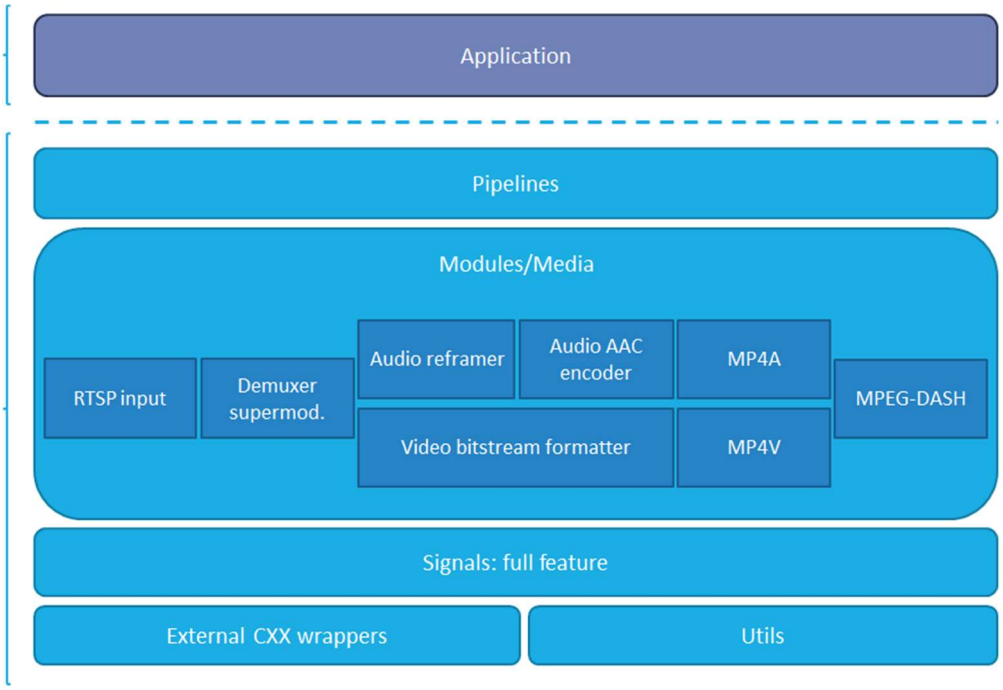

Figure 5: Example architecture application using Signals.

```
class Module : public IModule, public InputCap, public OutputCap {
   public:
      Module() = default;
      virtual ~Module() noexcept(false) {}
      virtual void flush() {}
   private:
      Module(Module const&) = delete;
      Module const& operator=(Module const&) = delete;
};
```

Figure 6: Module interface

```
VideoGenerator::VideoGenerator() {
  output = addOutput(new OutputPicture);
}

void VideoGenerator::process(Data /*data*/) {
  auto const dim = VIDEO_RESOLUTION;
  auto pic = DataPicture::create(output, dim, YUV420P);

  // generate video
  auto const p = pic->data();
  auto const val = (m_numFrames % FLASH_PERIOD) == 0 ? 0xCC :
0x80;
  memset(p, val, pic->getSize());

  auto const framePeriodIn180k = IClock::Rate / FRAMERATE;
  assert(IClock::Rate % FRAMERATE == 0);
  pic->setTime(m_numFrames * framePeriodIn180k);

  output->emit(pic);

  ++m_numFrames;
}
```

Create output pin

Get a picture from the allocator

Set timestamp

Send data to anyone that is connected

Figure 7: Video generator module code example (C++)

```
auto demux = pipeline.addModule(new Demux::LibavDemux(opt.url));
auto dasher = pipeline.addModule(new Modules::Stream::MPEG_DASH("dashcastx.mpd", params);

int numDashInputs = 0;
for (size_t i = 0; i < demux->getNumOutputs(); ++i) {
  auto const metadata = getMetadataFromOutput<MetadataPktLibav>(demux->getOutput(i));
  auto decode = pipeline.addModule(new Decode::LibavDecode(*metadata));
  pipeline.connect(demux, i, decode, 0);

  if (metadata->isVideo()) {
    auto webcamPreview = pipeline.addModule(new Render::SDLVideo());
    connect(decode, webcamPreview);
  }

  auto const numRes = metadata->isVideo() ? opt.v.size() : 1;
  for (size_t r = 0; r < numRes; ++r) {
    auto converter = pipeline.addModule(createConverter(metadata, opt.v[r].res));
    connect(decode, converter);

    auto encoder = pipeline.addModule(createEncoder(metadata, opt, r));
    connect(converter, encoder);

    auto muxer = pipeline.addModule(new Mux::GPACMuxMP4(filename, opt.segmentDuration));
    connect(encoder, muxer);

    pipeline.connect(muxer, 0, dasher, numDashInputs++);
}
```

Figure 8: Transcoder application code (C++): adding a preview in Red

Motion Spell plans to use Signals for live use-cases.

4.3. GPAC for orchestration

GPAC offers interesting capabilities for orchestration. GPAC has a very extensible support of the MPEG-4 set of standards. This includes the MPEG-4 Systems (OD - object-based) as a model and MPEG-4 BIFS for the Presentation layer (i.e. equivalent to Flash or HTML5). GPAC also has support for SVG. The MP4Box packager can import a lot of media in a single MP4 file (ISOBMFF container). Then, the presence of a presentation explains how to use these media in time and space. GPAC was used (both on packaging and playback) to demonstrate sub-frame accuracy resynchronization. This can be useful in use-cases where media are assembled at rendering time and need perfect synchronization (scalable codecs, picture-in-picture, hybrid broadcast/broadband delivery, …)

The GPAC packaging process is lightweight. It can easily handle tens of megabytes of data per second and is usually limited by the I/Os of the underlying computer.

The integration of GPAC can be done in four main ways:

- Using existing open-source tools.
- Using the GPAC C API (using the libgpac.so/dylib/dll).
- Building custom applications from scratch.
- Build custom applications using the Signal toolkit.

Motion Spell plans to assist other partners with GPAC and related R&D works.

4.4. Orchest.js

The TNO Media Orchestration Framework consists of Orchest.js and SignalMaster, both together offer a live communication orchestration server for multi-user VR experiences, with support for 2D and 3D environments and streams. Orchest.js takes care of session control, room

configurations and media synchronization (multi-user synchronized playback, shared playlists, playout control). SignalMaster is dealing with user discovery and session initialization for WebRTC. Both components work independent from each other, but are both necessary to support the full orchestration and audio/video communication functionalities in the client. More information about these components and how they operate in specific social VR experiences can be found here [8][9].

Feature	Performance and support
Orchest.js	
Target device	Linux, Mac or Windows instance; Uses NPM/Node.JS. Does not require powerful machine.
Parallel sessions	Yes, we tested multiple concurrent sessions.
Server side A/V rendering	No, current initial version uses only client side rendering. Currently only signalling and control.
Administration	Rudimentary admin interface which shows the current sessions, connected users. Allows for content playback control (skipping content, mute/unmute,etc)
User management	No; Each client is treated with its randomly generated client ID. WebRTC id is tied to the client id. No user login/registration etc.
Room management	Currently semi hardcoded, semi configurable
SignalMaster	Our implementation is based on https://github.com/andyet/signalmaster
Target device	Linux, Mac or Windows instance, uses NPM/Node.JS
End-to-end user discovery	Yes, SignalMaster does this
Stream setup and control	Yes, users are directly addressable via IP in a local network. For users behind NAT or firewall, a STUN/TURN server is needed. For users connected via a gateway, a WebRTC gateway (e.g. JANUS) is needed.

Integration: It is to be expected that integration with other software will require refactoring and new API calls. Currently the Orchest.js does not offer open API calls besides read access to session information. However technically it utilizes Socket.IO, thus offers easy connection to other clients and system components. SignalMaster operates with the open WebRTC standard and is thus compatible with any WebRTC compatible client.

4.5. Real-time 2-player interaction

A RabbitMQ-based implementation for real-time 2-player interaction, supporting 3D reconstruction live streaming for both players in real- time. For this technology, a server-based networking scheme is preferred over a P2P network, to offer scalability and centralized simulation capabilities. Each local 3D reconstruction setup transmits the data to the server, where synchronization takes place, following then the global state transmission to the

connected users. To this end, the RabbitMQ framework is used as the messaging layer, with each remote setup sending its messages to the server. The server then synchronizes the game state and in turn sends a state message to each client. As in most internet-based multi-user components, client side prediction has been implemented to account for transfer latency. The employed centralized architecture allows for non-participant users to also connect to the platform as spectators, allowing them to observe the immersive environment scene along with the interacting participants.

Features: Streaming data to multiple clients and streaming queue configuration

Integration Aspects: C# API

4.6. Synchronization

A synchronization server based solution connected to the same local network as the players to coordinate the distributed playback experience. Its main task is to make sure that all players are synchronized to the same clock and get selected content. The sync manager can be a standalone application independent from other players, running on a separate Windows or Linux PC (for test and demo scenarios). For the synchronization, the player implements emerging broadcast standard DVB-CSS (Companion Screen Synchronization), that is going to be a part of HbbTV 2.0. Especially it uses the following protocols defined by the standard:

- Discovery – for finding and connecting to session manager in local network,
- Wallclock (DVB-CSS-WC) – to synchronize clocks of all connected devices, to have common reference time,
- Content Information and Identification (DVB-CSS-CII) – for sharing the information about currently played content,
- Timeline Synchronization (DVB-CSS-TS) – for coordinating playout of the content.

Features:
Synchronizes and integrates different video and audio streams in a consistent experience.
Synchronizes content at the frame level with other displays in the local network.
Can distribute messages in the network between devices.

Integration aspects: Serves times as master, can be integrated in a client, so there's no need to deploy another service in the network. The times are defined per session, for live scenario NTP time can be used.

5. OVERVIEW OF INITIAL SOFTWARE VERSION FOR RENDERING AND DISPLAY

In this section, information regarding the initial software versions of the components related to the Task 3.4 “Rendering and Display” is given. The components described in this section aim to provide real-time rendering of the encoded hybrid media that is distributed over the network. The components available from different partners are the web-based TogetherVR client, the Unity3D native player, the GPAC and Signals player infrastructure and the time-varying mesh rendering tool.

5.1. TogetherVR Client

The TogetherVR Client is an A-Frame based client for multi-user photorealistic 360 degree experiences, in a WebVR capable web-browser. The front-end supports playback of video-streams, showing panoramic backgrounds, and showing other participants as 2D-video objects in the VR environment via WebRTC integration. The framework includes synchronized media playback, positional audio, capturing a user via a webcam, screen sharing and interaction with the VR environment.

Feature	Performance and support
Client device	Powerful PC or Laptop; We use MSI VR Ready gaming laptops with 1070/1080 GTX graphics card and Intel i7 processor.
Target OS, browser, headset	Windows, Firefox, Oculus Rift CV1; Other browsers do support WebVR/A-Frame, but we have seen issues with: headset support; video playback; performance. Chrome on Android is also an option but not sufficiently tested. Oculus Rift does not work on non-windows machines.
# of simultaneous participants in a shared virtual environment	Up to three users at 960x540 and 25 fps. We tested with 4 people, but with severe performance penalties. Performance is a function of the resolution, frame rate, etc.
3-degrees of freedom Static experience (tethered)	Yes; users are standing or sitting. The 360 camera to capture the room needs to reflect the user position, including the correct height. E.g. standing: the lens should be at eye-height of someone who is standing. Sitting: the lens should be at eye-height of someone who is sitting
6 degrees of freedom (untethered)	No; currently we only have a 2D-360 degree background. In the case of a rendered room we could support 6 degrees of freedom, depending on performance.
2D video of buddy	Yes; A WebRTC capture of other user(s) is positioned in the room. Supported codecs: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats
3D/stereoscopic video of user	Not implemented or tested. Stereoscopic is supported in A-Frame, but no experience with this yet.

Positional audio	We currently have a monaural recording of the user (mono-mic). The signal is transmitted and spatially aligned at the receiver.
Full spatial audio	Not implemented.
Video playback	MP4 files, WebM files, MPEG-DASH streams, in any resolution up to full HD. Dash playback using dash.js library. Supported formats depend on browser. We use Firefox. We may support higher resolution videos but this depends on: codec/cpu performance and browser support. Supported formats: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats
Synchronized video playback	Yes; we use a proprietary cooperative sync mechanism using NTP and HTML5 player timestamps. Clients periodically broadcast their current playback position. The clients lagging behind a certain threshold are forwarded
User capture and transmission	Yes, using WebRTC
A/V participant communication	Yes, using WebRTC. We use the SimpleWebRTC library for real-time communications
Bandwidth requirements per buddy stream	Typically 1 Mbps per stream, but note that WebRTC automatically adapts to network conditions.
End-to-end audio/video latency	Approx. 300 ms, Tested on Lan. Bound to typical WebRTC latency
Background	2D Image: Equirectangular 2D 360-degree, up to 4K 3D image: not tested 2D 360-degree video: not tested 3D 360-degree video: not tested Fully rendered backgrounds: not tested
Headset removal	Not implemented
User placement	Using alpha blending and spatial alignment; Note that in the current component version, the user feed contains some visual artifacts due to the blending, but also background removal (user capture).

Integration: The TogetherVR Client is directly bound to the Orchest.js server, thus significant refactoring and adjustments need to be made to integrate it in any other system architecture. Thus, as a mean of easy integration, we are currently planning to release our client as an open (cloud-based) service in Q1 2018.

5.2. Unity3D native player

Multiplatform player based on Unity3d engine enables intradevice and interdevice synchronized display of omnidirectional content with additional video portals on HMD, phones, tablets and TV sets through DVBCSS standard, supports stereo and ambisonics audio. Display contents based on a descriptive xml file, the contents are composed by Sequences, Scenes, Shapes and Hotspots (for interactivity).

The player can create different shapes with videos and images with transparency. The objects are: spheres, rectangles and hotspots. The spheres are used for omnidirectional videos or static images. The rectangles are used for directional videos and images and can be interactive. The hotspots are pointing elements inside the scene for transitions, sequence swap or showing images.

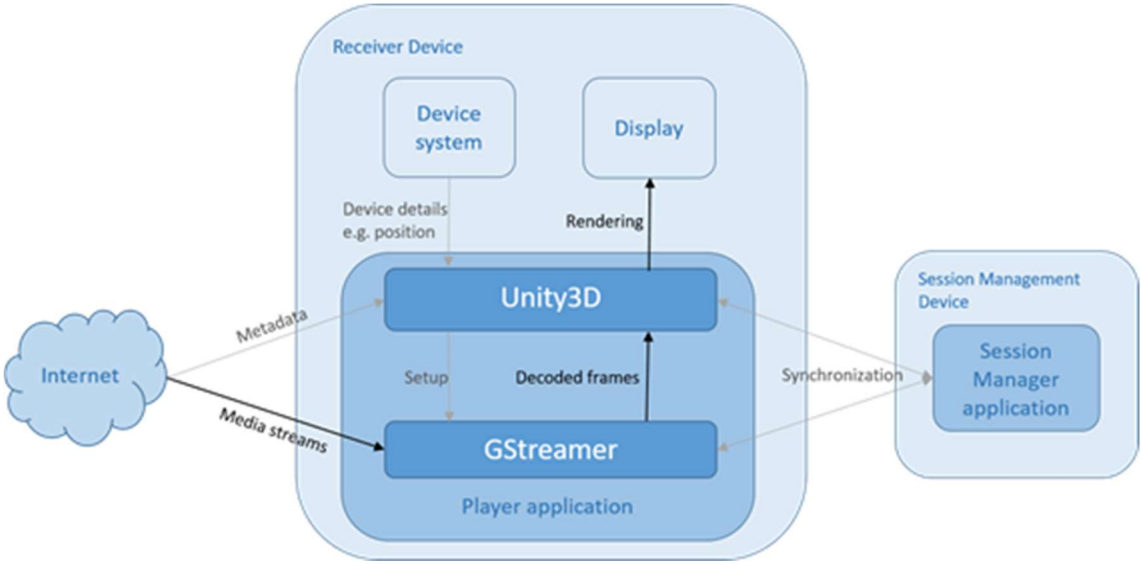

Figure 9: Unity3D native player.

The basic functionality of player includes:

- Content selection – player presents list of content available on server.
- Playout of VRTogether content:
  - Selecting object timeline based on device type.
  - Presenting 360° scene on devices.
  - Scene can be composed from multiple media streams.
  - Possible interactivity with the user.
  - Video transitions on user actions.
- Support for video streams:
  - Encoding: AVC/HEVC
  - Streaming protocols supported: MPEG-DASH, RTSP/RTMP
- Support for images. (PNG, JPEG)
- Support for 3D Meshes.
- Support for Point Clouds
- Synchronization with other devices.
- Logging QoE data (device parameters, displayed content id, delays, looking direction).
- Support for spatial audio

5.3. GPAC and Signals player infrastructure

We have two player infrastructures that work on most platforms:

- The GPAC open-source player (http://gpac.io) with compositing capabilities similar to Flash or HTML5 (using internally SVG and MPEG4-BIFS extended with Javascript).
- A quite traditional A/V pipeline based on our Signals platform (http://signals.gpac-licensing.com). These player infrastructures bring a significant experience in playing back and re-synchronizing content at the player level. That is where Motion Spell plans to help.

Features: a list of features is available at http://gpac.io/player/features. Below, some screenshots are provided.

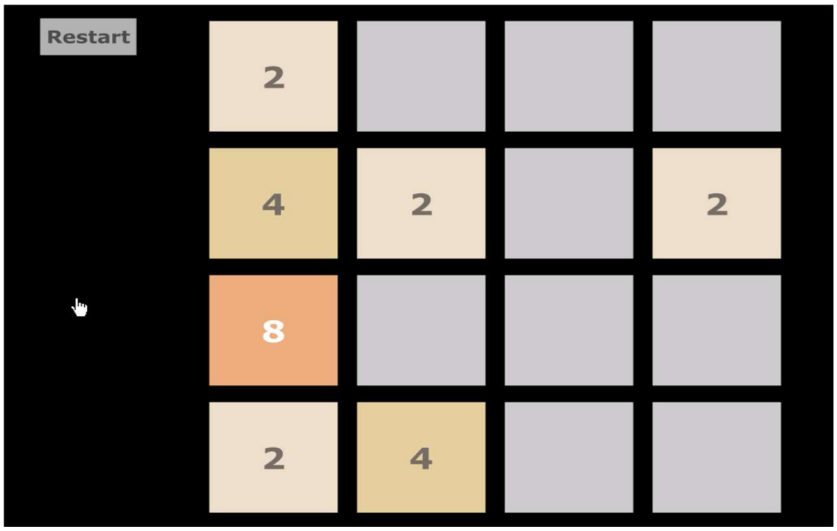

Figure 10: Interactive 2048 game within MP4Client

Figure 11: Interactivity within a movie.

Figure 12: Magnifier effect while playing a video.

Figure 13: Multi-view frame-accurate playback and switches.

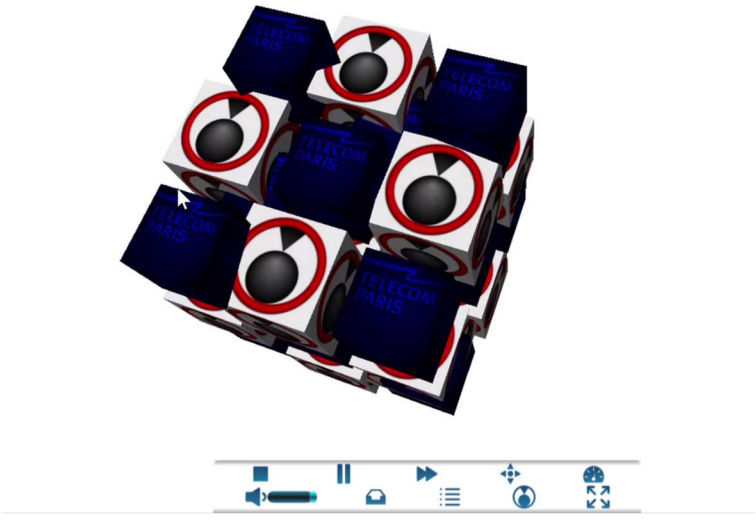

Figure 14: Mixing 2D, 3D and interaction within MP4Client.

Performance: comparable to what modern Web browsers achieve (but with a reduced set of features).

Integration: GPAC and Signals are integrable using C or C++ APIs.

5.4. Time-Varying Mesh Rendering

A multi-texturing approach, including one RGB texture from each RGBD sensor, is used to "paint" the 3D reconstructed geometry. In particular, each mesh triangle is assigned two textures, which are then blended in a weighted manner improving the visual quality of the overall texturing. According to that method, the texture mapping weights depend on: a) the "viewing" angle of the captured surface, i.e. on the angle between the line-of-sight vector and the vertex normal, and b) the 2D distance of the 2D projection pixel from the foreground human's 2D silhouette. On top of that, a custom shader has been developed that takes into account the extracted weights in order to appropriately render the textured mesh.

Features: The component enables appropriate texturing of time varying meshes using multiple RGB textures.

Integration Aspects: The component is used as an OpenGL shader in order to texture properly the time varying mesh using the multiple RGB textures.

6. OVERVIEW OF RELEVANT 3RD PARTY TOOLS

Some of the initial software components depend on 3rd party tools. Below a list of these tools is provided.

- SignalMaster: Simple socket.io server for webrtc signaling. Can be downloaded from: https://github.com/andyet/signalmaster
- RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. Can be downloaded from: https://www.rabbitmq.com/

7. CONCLUSIONS AND OUTLOOK

This deliverable provides the consortium partners with a description of the initial software components, allowing them to start both i) the further development of individual components and ii) the integration of the initial components towards the first project pilot. For each WP3 task T3.1-T3.4, an initial set of components has been described, along with its feature, a performance indication and information on integration.

As a follow-up action, the project partners need to elaborate on per-component and per-task experiments, to build a WP3 component development roadmap; and to discuss joint source code management (WP2) and testing (T3.5) for facilitating integration.

8. REFERENCES

[1] J Kammerl, N Blodow, R B Rusu, S Gedikli, E Steinbach, and M Beetz. Real-time compression of point cloud streams. Robotics and Automation (ICRA), 2012 IEEE International Conference, pages 778–785, May 2012.

[2] Rufael Mekuria, Kees Blom, and Pablo Cesar. Design, implementation and evaluation of a point cloud codec for tele-immersive video. IEEE Transactions on Circuits and Systems for Video Technology, January 2016.

[3] Octree data structure. https://commons.wikimedia.org/wiki/File:Octree2.svg. Accessed: 2016-10-28.

[4] Ricardo De Queiroz and Philip A. Chou. Compression of 3d point clouds using a region-adaptive hierarchical transform. IEEE Transactions on Image Processing 25, June 2016.

[5] Ruwen Schnabel and Reinhard Klein. Octree-based point-cloud compression. Proceedings of Symposium on Point-Based Graphics 2006, Eurographics, July 2006.

[6] Dorina Thanou, Philip A. Chou, and Pascal Frossard. Graph-based motion estimation and compensation for dynamic 3d point cloud compression. Image Processing (ICIP), 2015 IEEE International Conference on, September 2015.

[7] Cha Zhang, Dinei Florencio, and Charles Loop. Point cloud attribute compression with graph transform. Image Processing (ICIP), 2014 IEEE International Conference on, October 2014.

[8] Simon N.B. Gunkel, Martin Prins, Hans Stokking, and Omar Niamut. 2017. Social VR Platform: Building 360-degree Shared VR Spaces. In Adjunct Publication of the 2017 ACM International Conference on Interactive Experiences for TV and Online Video (TVX '17 Adjunct). ACM, New York, NY, USA, 83-84. DOI: https://doi.org/10.1145/3084289.3089914

[9] S. Gunkel, M. Prins, H. Stokking, and O. Niamut, "WebVR Meets WebRTC: Towards 360-Degree Social VR Experiences", Proc. IEEE VR 2017, p. 457--458 DOI: 10.1109/VR.2017.7892377

ANNEX I – OCTREE DATA STRUCTURE

The most popular data structure used to code point clouds is the octree. An octree is used to partition 3D space by recursively subdividing it into 8 octants as shown in figure 5. For point clouds this recursive subdivision is done for all regions that are occupied. This data structure is the 3D analog of the 2D quadtree used in video compression. The primary reason octrees have been used for point cloud compression [2, 1, 5] is because they enable a more efficient representation of the point cloud geometry by regularizing the structure and using occupancy codes for partitioned spaces. An example of occupancy coding is shown in figure 6. This is done during the subdivision process where a code of occupied octants can be generated at each level of the tree. These occupancy codes together describe the entire geometry of the point cloud. The leaf nodes of the tree are voxels and the geometry of points within the voxel can be differentially encoded (for example using coordinates relative to the centroid of each voxel). The attributes such as color are compressed separately for each leaf node either by mapping to an image grid [2, 1]

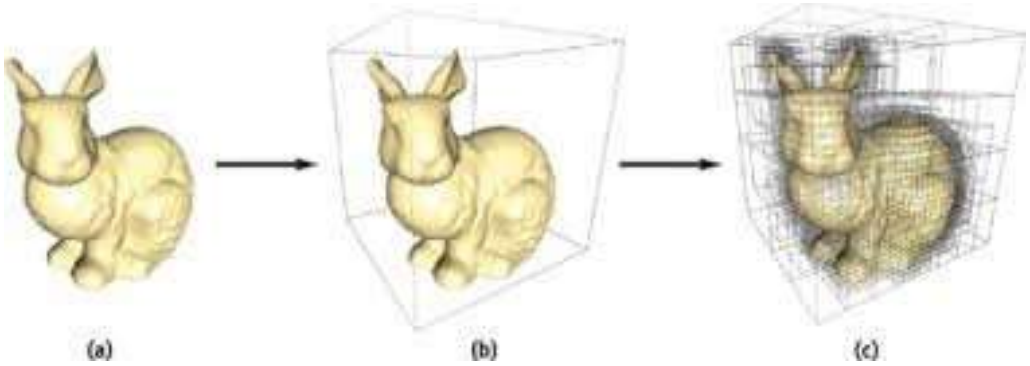

Figure 15: An example of recursive subdivision in octrees.

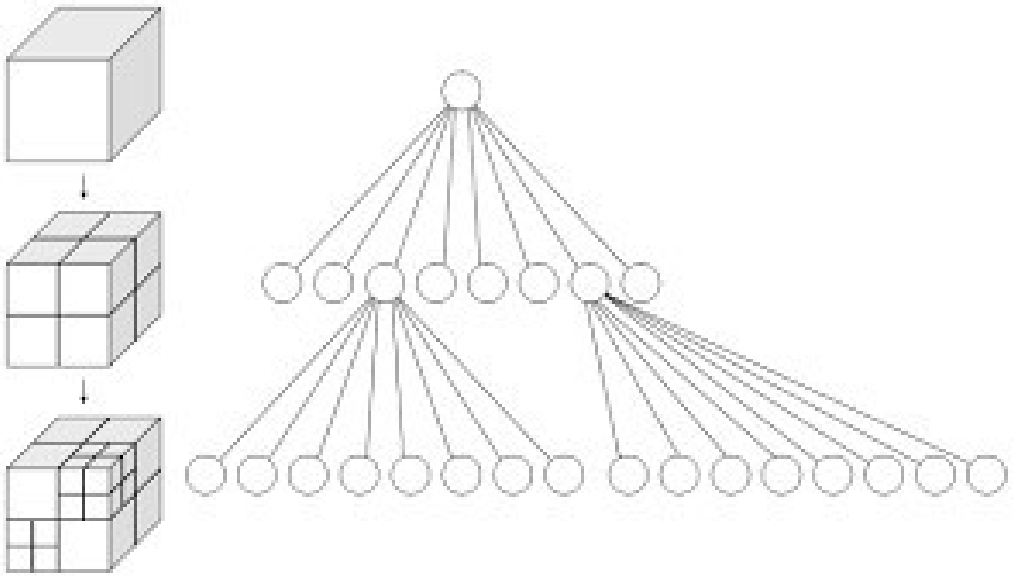

Figure 16: Occupancy coding after subdivision [3]

or by graph transform [6, 7, 4]. The graph transform method is not suited for dynamic scenes as the computational overhead becomes prohibitive but they are more suited for

static point clouds. For dynamic scenes directly encoding colors by placing them on a 2D image grid has been shown to be more effective [2].

The attribute compression process is repeated for each of the leaf nodes in the octree. The number of leaf nodes in an octree are related to the tree depth exponentially. For a tree of depth n the number of leaf nodes can be up to 8^n depending on how they occupy the 3D space (empty regions are not subdivided further). The computational complexity is therefore exponential with respect to the octree depth or level of detail, as the number of leaf nodes that need to be coded increases by a factor of 8 each time subdivision occurs.

<END OF DOCUMENT>