HARVEST4D

HARVESTING DYNAMIC 3D WORLDS FROM COMMODITY SENSOR CLOUDS

# Final integrated prototype

## Deliverable 9.5

| | |
|---|---|
| Date: | 15.5.2016 |
| Grant Agreement number: | EU 323567 |
| Project acronym: | HARVEST4D |
| Project title: | Harvesting Dynamic 3D Worlds from Commodity Sensor Clouds |

## Document Information

| | |
|---|---|
| **Deliverable number** | D9.5 |
| **Deliverable name** | Final integrated prototype |
| **Version** | 1.0 |
| **Date** | 2016-05-15 |
| **WP Number** | 9 |
| **Lead Beneficiary** | CNR |
| **Nature** | P |
| **Dissemination level** | PU |
| **Status** | Final |
| **Author(s)** | CNR |

## Revision History

| Rev. | Date | Author | Org. | Description |
|---|---|---|---|---|
| 0.1 | 10/04/2016 | P.Cignoni, G.Palma | CNR | First draft |
| 0.2 | 08/05/2016 | All partners | | Second draft |
| 0.3 | 15/05/2016 | P.Cignoni, G.Palma | CNR | Final version |

## Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

# TABLE OF CONTENTS

# 1 EXECUTIVE SUMMARY

This document presents the status of the work in the Work Package 9 (WP9) - Integration and Evaluation - at the end of Month 34 of the project HARVEST4D.

The activities follow the original plan drafted in the project Description of Work (DoW).

The objective of this deliverable is to describe the final prototype defined in the deliverable 9.4. The final prototype is composed by a set of selected WP prototypes that exchange intermediate data using the common data interfaces and formats described in the deliverable 9.1. The used WP prototypes are subdivided in two categories: the WP prototypes involved in the data processing; the WP prototypes for the visualization of the intermediate and final data. The following Sections introduce each WP prototypes with a general description, its impact on the projects and its basic usage.

The binaries of all prototypes can be accessed of the Harvest4D webpage.

# 2 DATA PROCESSING PROTOTYPES

To make more easy the understanding of the interaction among the different WP prototypes involved in the data processing and how the data flow among them, we report in Figure 1 the integration scheme already presented in the deliverable 9.4. In the figure we use the following visual rules:

- The green boxes represent the different type of data involved in project
- The blue boxes represent the WP prototypes used in the final prototype;
- The arrows show how the data flow in and out the different WP prototypes.

The type of data handled by the various prototypes are:

- **images**
- **point clouds** with a set of possible per-vertex attributes defined in the Deliverable D9.1
- **triangular meshes** with a set of possible per-vertex attributes defined in the Deliverable D9.1
- **camera parameters** using the format described in the Deliverable D9.1
- **compressed point clouds**, using the format described in [Golla et al. 2015]
- **reflectance data** as Surface Light Field (SLF) using the format described in [Palma et al. 2013]

- **compressed Sparse Voxel Octree** (SVO) data structure using the format described in [Dado et al 2016]

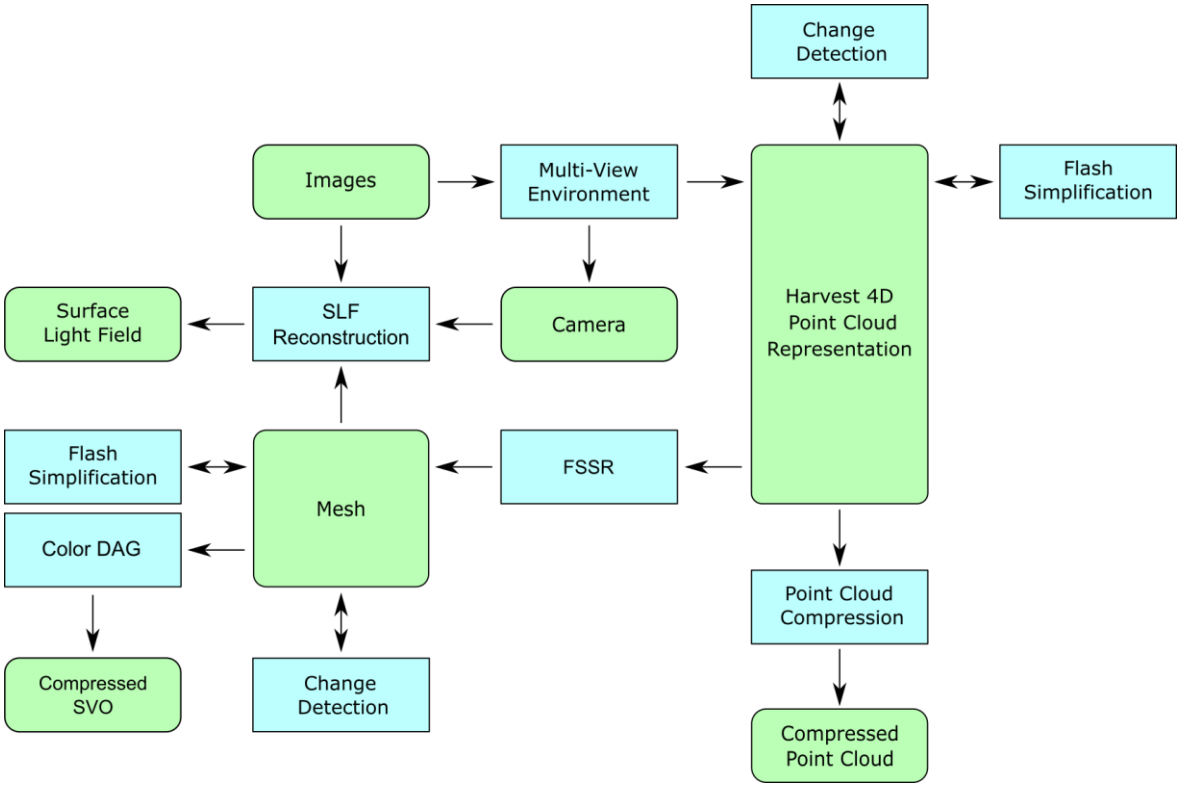


Figure 1. Integration scheme of the Work Package prototypes used by the final prototypes in the data processing.

## 2.1 MULTI-VIEW ENVIRONMENT

### 2.1.1 Description of Prototype

The software Multi-View Environment (MVE) from [Fuhrmann et al. 2014a] takes multiple photos of a scene as input. It includes a Structure-from-Motion (SfM) implementation to estimate a sparse point cloud of a captured scene as well as camera parameters of images during the capturing process, such as their rotations and positions.

Moreover, MVE contains a Multi-View Stereo (MVS) implementation to calculate a dense depth map for each registered camera and its respective image. The estimated dense depth maps can then be converted and merged into a large and dense 3D sample point cloud including positions, normals, scale values (sample sizes) and other optional attributes, such as colors and confidences. See also Section 3.1 for data visualization by means of MVE.

### 2.1.2 Impact for the Project

MVE has a large impact for the project since it is one of the basic building blocks of the Harvest4D data processing system. Importantly, it is designed to fit to the Harvest4D capture paradigm by being robust and working well for uncontrolled image datasets with challenging properties, such as strongly varying camera to object surface distances, changing lighting conditions and so on.

The output 3D point clouds produced by MVE are used by many other Harvest4D tools like FSSR, Flash Simplification and Change Detection to name a few of them.

### 2.1.3 Usage

First of all, the command line tool `makescene` imports images from an input directory and creates a scene directory with the given images and some meta data. For example:

```
makescene.exe -i -o StatueImages StatueScene
```

The usual next step is to use sfmrecon to compute SfM data for the created scene:

```
sfmrecon.exe StatueScene
```

Alternatively, MVE also supports import of SfM results from other tools, such as Noah Snavely's Bundler (http://www.cs.cornell.edu/~snavely/bundler/) or VisualSfM (http://ccwu.me/vsfm/).

Depth maps are computed via the command line tool `dmrecon` which accepts a scale parameter to control image resolution for depth estimations, for example:

```
dmrecon.exe -s2 StatueScene
```

Finally, the tool scene2pset converts the results of `dmrecon` to represent the scene by means of a 3D point cloud. It should be called with the same scale parameter as for `dmrecon`:

```
scene2pset.exe -F2 StatueScene StatueScene/scenePointCloudL2.ply
```

Each tool describes its required input when called without arguments. Additionally, a more detailed user guide can be found at the following link:

https://github.com/simonfuhrmann/mve/wiki/MVE-Users-Guide.

## 2.2 FLASH SIMPLIFICATION

### 2.2.1 Description of Prototype

This prototype is a high performance geometry simplification software that allows to reduce the size of 3D models made of millions of polygons/points in a few milliseconds. It builds upon a new approach to parallel error-driven adaptive clustering, Morton Integrals [Legrand et al. 2015], to fully exploit fine grain parallelism on the GPU. An adaptive partition of the space is created on the

GPU, and a single representative point is kept for each cluster, preserving salient features and detailed structures. With this method, it is also possible to account for per-sample attributes, such as colors and normals. Real time simplification performance is achieved on today's hardware, adapting 3D content to a target platform at the speed of content generation and demand.

### 2.2.2 Impact for the Project

With real time performances and a feature-preserving adaptive behavior, this prototype can be easily used as a pre-processing step in the other tasks of the project, instantly making very large datasets manageable for the different parts of the pipeline, without losing important features and detailed structures.

Since both point clouds and meshes are handled, and points attributes are preserved, this prototype allows the fast simplification of geometry data in many different scenarios: as fast as it is captured for example, or as it is generated by another data-processing tool of the project.

### 2.2.3 Usage

This prototype is a command line tool, that requires Windows and a CUDA capable GPU. It takes as input a 3D mesh or point cloud with normals, in PLY format. It can be executed as follow :

```
fsimp -i inputFile.ply [options]
```

Where the options can be :

- `-i [--input]` ARG : required, input file, .ply or .off
- `-o [--output]` ARG : output file, .ply
- `--morton-precision` ARG : Morton code precision (size of the implicit discretization grid). default value : 256
- `-e [--max-error]` ARG : error threshold. default value : 0.000001
- `-c [--keep-color]` : preserve color information
- `-n [--keep-normals]`: preserve normals
- `-p [--points-only]`: point cloud simplification
- `-m [--color-metric]` ARG : use color metric with specified error threshold
- `-h [--help]`: displays usage information

## 2.3 FSSR (FLOATING SCALE SURFACE RECONSTRUCTION)

### 2.3.1 Description of Prototype

The tool Floating Scale Surface Reconstruction from [Fuhrmann et al. 2014b] computes a surface mesh from a multi-scale input point cloud. The input point cloud must consist of surface samples

with positions, normals and scale values. Optionally, attached colors and confidences can be additionally provided to improve reconstruction quality.

FSSR calculates a global implicit surface function from the input sample set to represent the complete scene. This global function is obtained by intelligent merging of radial basis function for the input samples in a multi-scale way. A Gaussian basis and a compactly supported polynomial weighting function are used for each sample to locally represent them. The implicit scene representation is then sampled by means of a sparse Octree with finer sampling at closely captured areas and a coarser evaluation grid for scene parts which were captured at larger distance. In this way FSSR provides multi-scale surface meshes with high detail at points of interest which were captured closely and less triangles for unimportant scene side parts.


**Figure 2. Multi-scale reconstruction of the Elisabeth memorial in color (left), with shading (middle) and false coloring indicating scale (right).**

### 2.3.2 Impact for the Project

FSSR has a large impact for the project since it nicely fits to the Harvest4D idea that the users and devices define the capture process and not the algorithms. This is achieved by FSSR as it virtually does not require user-defined parameters. Thus it is very easy to use. Users do not need to take care about scale issues and can simply provide input data with strongly varying scale. Furthermore, it supports surface reconstruction as long as the required positions, surface normals and scale values are available. This is why it can be employed for various uses cases, such as scene reconstruction from Multi-View Stereo input or aligned laser range scans. It can also handle large scenes thanks to its compactly supported basis functions and avoidance of global operations.

### 2.3.3 Usage

The FSSR command line tool takes as arguments one or more PLY input point cloud names and lastly the name of the output PLY file in which it stores the extracted surface mesh. Here is a simple example:

```
fssrecon.exe inPointSet0.ply inPointSet1.ply outSurface.ply
```

Additionally, low-confidence vertices, small isolated clutter components and degenerated triangles should be removed by the tool mesh clean. Tool input arguments are the names of the input PLY file to be cleaned and the name of the file in which the cleaned result is saved, e.g.:

```
meshclean.exe outSurface.ply cleanOutSurface.ply
```

Both tools describe their expected input when called without any argument. Moreover, a more detailed user guide can be found here:

https://github.com/simonfuhrmann/mve/wiki/FSSR-Users-Guide

## 2.4 CHANGE DETECTION

### 2.4.1 Description of Prototype

The prototype allows the detection of the geometric differences between two point clouds or two meshes of the same environment acquired in different times [Palma et al. 2016]. In the specific, it computes a change value that can be used to segment each model in two subsets (Figure 3): the static part, composed by the shared geometry between the two times; the dynamic part, composed by the geometry that is changed in the time due to the object movements, object that disappears, and occlusions.

Starting from two 3D model, the prototype removes the outliers using a probabilistic operator. Then, it detects the actual change using the implicit surface defined by model geometry under a multi-scale implicit surface reconstruction that, compared to the classical proximity measure, offers a more robust change/no-change characterization near the temporal intersection of the scans and in the areas exhibiting different sampling density and direction. The resulting classification is enhanced with a spatial reasoning step to solve critical geometric configurations that are common in man-made environments, like occlusions and small rigid transformations with coherent temporal overlapping, by checking the local coherency of the segmentation.

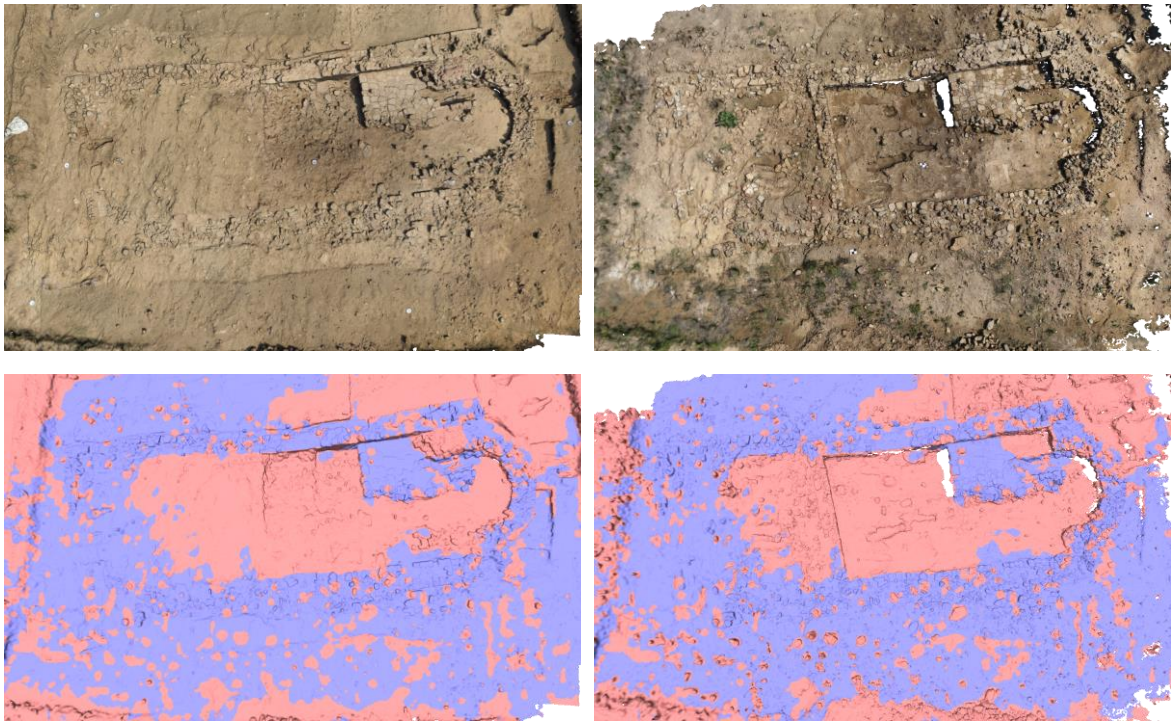

Figure 3: Change detection results on a two captures of an archeological excavation. The 3D models are generated using images acquired by a drone. (Top) Rendering of the two 3D model with a vertex color. (Bottom) Color mapping of change detection results (red = change, blue = no-change).

### 2.4.2 Impact for the Project

The prototype meets the objective of the Work Package 6.1 for the automatic detection of temporal changes in the input geometry. The output of the prototype is very useful for the other task of the project that can take advantage from an accurate segmentation between the static and the dynamic part of the input clouds. For example, the dense 3D reconstruction can take advantage of the segmentation in order to create a more complete model where to put together all the static regions and the change areas from only one time. Another example is the efficient visualization of change evolution of a scene where the segmentation can help in the design of a tool with the purpose to maximize the perception of the most significant changing regions and to minimize the effects of the negligible one on the user's attention.

### 2.4.3 Usage

The prototype is a command line tool that takes in input two 3D models and returns the processing at different steps of the algorithm. The format used to read and save the 3D models is the PLY format described in the Deliverable D 9.1. The prototype assumes that the input clouds have the a per-point normal and saves the output with two new per-vertex attributes: "quality" and "radius". The "quality" attribute stores a change probability in the range [0.0, 1.0]. The

"radius" attribute stores the point radius that depends on the density of the cloud around the point.

In the specific, the tool takes four arguments:

```
detectChange.exe time0.ply time1.ply save0.ply save1.ply
```

where the first two arguments are the filenames of the input 3D models and the last two arguments are the filenames where to save the final processing. The tool takes different command line options:

- `-p`, to stops the algorithm after the preprocessing of the input point clouds. The preprocessing involves the point radius computation and the outlier removal;
- `-o`, to set the outlier threshold (default 0.5);
- `-g`, to stop the algorithm after the local temporal comparison of multi scale reconstructed implicit surface;
- `-c`, to stop the algorithm after the check of the consistency of the change quality value using the coherency of the point normal;
- `-a`, to compute the multi scale implicit surface in an adaptive way using an octree instead of a uniform grid. In this way the prototype can adapt the level of details of the multi scale analysis to the local density of the input data.

An additional tool `staticDynamicSep` allows the storage of the output in different file configurations. Given the point clouds *A* and *B* and the relative change regions, *C(A)* and *C(B)*, and no-change regions *NC(A)* and *NC(B)*, there are three different options:

- `staticDynamicSep output_t0.ply output_t1.ply nc_t0.ply nc_t1.ply c_t0.ply c_t1.ply` saves 4 independent 3D model with the no-change regions NC*(A)* and *NC(B)* (`nc_t0.ply` and `nc_t1.ply`) and the change regions *C(A)* and *C(B)* (`c_t0.ply` and `c_t1.ply`)
- `staticDynamicSep output_t0.ply output_t1.ply nc.ply c_t0.ply c_t1.ply` saves a model with the union of the no-change regions *NC(A)+NC(B)* (`nc.ply`) and two models with the change regions *C(A) and C(B)* (`c_t0.ply` and `c_t1.ply`)
- `staticDynamicSep output_t0.ply output_t1.ply t0.ply t1.ply` saves a model for each time that is as complete as possible (*NC(A)+NC(B)+C(A)* = `t0.ply` and *NC(A)+NC(B)+C(B)* = `t1.ply`)

## 2.5 POINT CLOUD COMPRESSION

### 2.5.1 Description of Prototype

This prototype allows the fast multi-modal compression of point clouds, i.e. both geometry and color information are stored in a compressed form in the resulting compressed point cloud. As already described in Deliverable 4.31, our method [Golla & Klein, 2015] is based on an initial decomposition of the volumetric data into local regions that are used as compression chunks. This is followed by a per-region data processing where the further compression of the point clouds belonging to a voxel is based on a decomposition into point clusters that can be represented by patches, parameterized over planar 2D domains. In order to represent finescale details, height and occupancy maps on these domains are generated. The height maps account for offsets of the original geometry from the domains, while the occupancy maps account for holes in the geometry, like e.g. windows in a building façade.

The final compression is based on compressing height and color maps with the JPEG or the lossy JPEG 2000 standard. JPEG is faster, while JPEG 2000 delivers smaller file sizes at comparable error rates. Occupancy maps are compressed via the lossless JBIG2 algorithm. Furthermore, the patch positions, orientations and sizes are compressed with the Lempel-Ziv-Markov chain algorithm (LZMA).

### 2.5.2 Impact for the Project

The compression of the point clouds represents a key component to store the huge amounts of acquired data in a compact format. Such a compact representation is not only relevant for storing huge point clouds but it is also important for a rapid data exchange as less data has to be transmitted and a decompression performed by the user allows to unpack the compressed data into a point cloud. Our data-compression technique allows to obtain a compact representation of a point cloud containing both geometry and reflectance information.

### 2.5.3 Usage

The prototype is a command line tool that can be used as follows:

1. `compresscloud –input input.ply –output outputfolder`
   compresses input point cloud and outputs compressed format to outputfolder.
2. `decompresscloud –input inputfolder –output reconstructed.ply`
   decompresses the data stored in the input folder and outputs to reconstructed point cloud.

## 2.6 COLOR DAG

### 2.6.1 Description of Prototype

This prototype showcases our voxel scene compression (see Figure 4), as described in [Dado et al. 2016].

The program is developed in C++ (using OpenGL and GLSL), with GPU support. It loads various file formats (ply, obj, pvm).

Figure 4. The prototype is able to compress voxel scenes to a great extent, enabling very high-resolution results.

### 2.6.2 Impact for the Project

This prototype provides a tool for voxelizing triangle meshes as well as directly loading voxel scenes and subsequently compressing them using an SVO-based compression scheme. To this end, we decouple geometry and attributes and compress them separately, for which the subsequent re-integration was a challenge. With this technique, we can decrease the memory footprint by an order of magnitude compared to a standard SVO.

### 2.6.3 Usage

Execute Research/Research.exe. The scene to load can be specified in Research/properties.txt under `obj_to_dag`, and the levels of the hierarchy in Research/shaders/shader_properties.txt under `shader_max_level`. As it can take quite long to generate the compressed scene, we provide two pre-processed examples; the arena and citadel scene, both at 17 levels. The current settings load the arena scene, while for the citadel scene you need to change the following in Research/properties.txt:

```
octree_type upclab2048
obj_to_dag ../Research/data/citadel.obj
dag_file ../Research/citadel
```

Navigation is possible via WASD. Hold `SHIFT` to move faster and `CTRL` to move slower There are many further options and controls which are discussed in detail in Readme.txt.

## 2.7 SLF RECONSTRUCTION

### 2.7.1 Description of Prototype

The prototype allows the computation of the Surface Light Field starting from a set of photos and a dense triangular mesh [Palma et al. 2013]. The input of the prototype is a set of photo aligned with the Multi-View Environment prototype and a dense triangular mesh obtained with the FSSR prototype (developed in the Task WP5). In the specific the prototype computes the Surface Light Field as a combination of two components (Figure 5): the diffuse color, using statistical analysis on the input color projected on the mesh; the residual component, that is the positive residual from the diffuse color, computed as linear combination of Hemispherical Harmonics functions. The coefficients of the Hemispherical Harmonics functions are obtained with a robust formulation of a weighted least square problem that avoids the creation of artifacts in the final rendering.

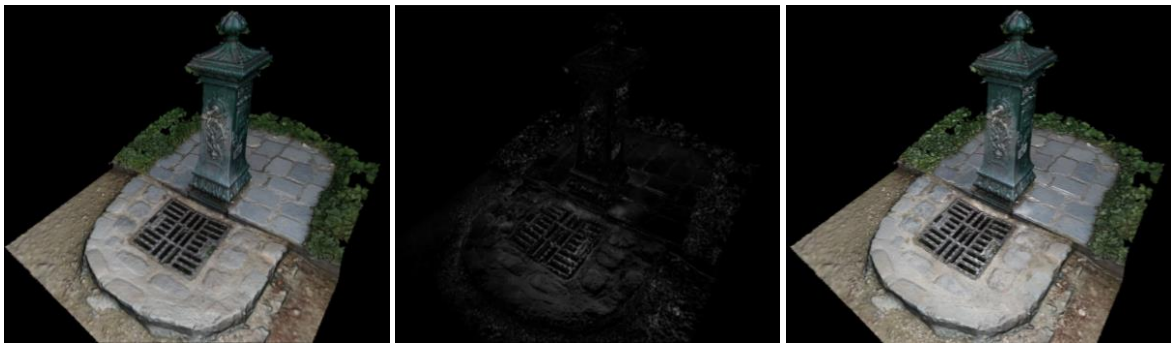

**Figure 5. Rendering of the Surface Light Field of a drinking fountain. (Left) Diffuse color. (Center) Residual appearance effects modelled using 16 Hemispherical Harmonics. (Right) Final rendering by combing the diffuse color and the residual appearance effects.**

### 2.7.2 Impact for the Project

The prototype meets the objective of the Task 7.2 for the estimation of the reflectance of an object in the case of acquisition in under constrained scenarios, for example a photographic acquisition of an object inside an uncontrolled lighting environment (outdoor, inside a museum, etc.).

The output of the prototype can be very useful to improve the real-time rendering of the model as it allows a more photorealistic visualization.

### 2.7.3 Usage

The prototype is a GUI tool that takes in input a dense triangular mesh and a set of aligned photos with the relative camera parameters and allows the computation of the Surface Light Filed for each vertex of the model using the algorithm described in [Palma et al. 2013]. The GUI guides the user in the different computation steps.

The format of the input data is the Multi-View Environment project format (for a complete overview see https://github.com/simonfuhrmann/mve/wiki/MVE-File-Format) to load the input photo with the relative camera and the Harvest4D PLY format (defined in the Deliverable 9.1) for the input 3D mesh.

In the specific, after the loading of the input data, the user must save all the color data projected on each vertex by each camera with the relative quality value using the command "Save Color Data". Then the user can compute two different approximations of the reflectance behavior of the objects: the only diffuse color that is saved in the vertex color of a new 3D model (command "Compute Diffuse Color"); the complete Surface Light Field that is saved in a binary file (command "Compute SLF"). For the computation of the Surface Light Field, the user can select the maximum order of Hemispherical Harmonics functions to use. The binary file of Surface Light Field contains the diffuse color and the coefficients of the Hemispherical Harmonics of each vertex. The tool allows the real-time visualization of the computed Surface Light Field.

# 3 VISUALIZATION PROTOTYPES

This section present the prototypes developed or partially developed in the Work Package 8 for the visualization of the output data. Figure 6 shows the association with the data that these visualization prototypes can take in input allowing the user interaction.

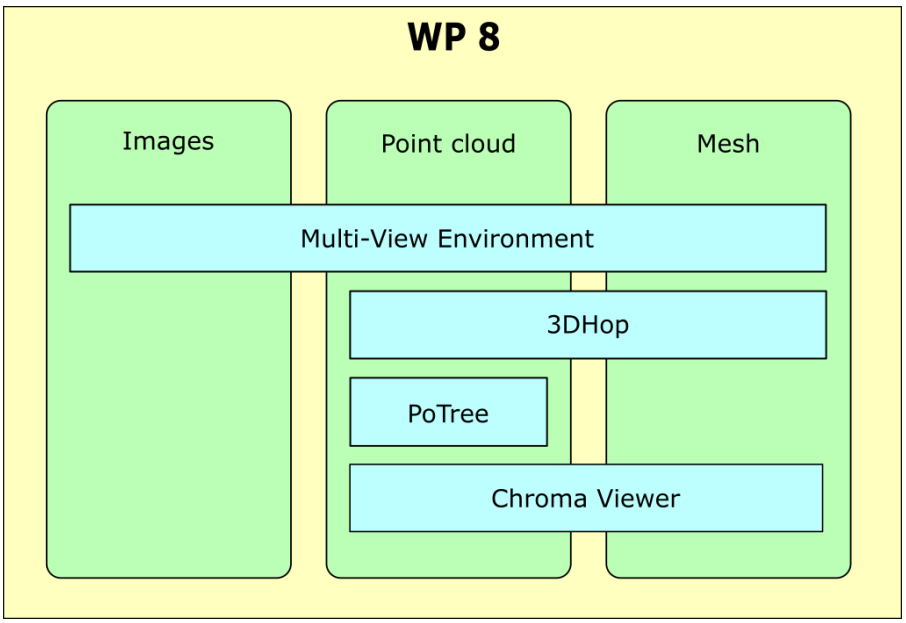


**Figure 6. Association of the visualization components available in the projects with the type of input data that are able to visualize.**

## 3.1 MULTI-VIEW ENVIRONMENT

### 3.1.1 Description of Prototype

MVE [Fuhrmann et al. 2014a] provides a graphical user interface called UMVE to inspect MVE scene directories with their images, SfM data, dense scene point clouds or surface meshes. See Section 2.1 for the data processing flow of MVE.

### 3.1.2 Impact for the Project

UMVE is often used during the project since it allows quick and easy management and especially validation of produced reconstructions.

### 3.1.3 Usage

To use UMVE, simply start the executable `umve.exe` and open an existing or create a new scene directory via the respective GUI elements. Alternatively, the tool can be called with a scene directory or mesh as command line argument to directly open it, e.g.:

```
umve.exe StatueScene/scenePointCloudL2.ply
```

UMVE provides a list of all registered cameras visualized by their image icons on its left window side. Using this list and the tab V*iew inspect*, all view dependent data can be selected and shown, such as original and undistorted images or depth maps. The tab *Scene inspect* provides access to the 3D visualization of the current scene where camera distributions, point clouds and meshes are rendered. It also allows opening and addition of meshes or selection of individual cameras or points to investigate their relationship with other scene parts. Furthermore, the right hand side provides some basic GUI elements to control rendering or change the scene by, for example, triangulation of a single depth map of a selected view.

For a more detailed user guide, please visit:

https://github.com/simonfuhrmann/mve/wiki/MVE-Users-Guide

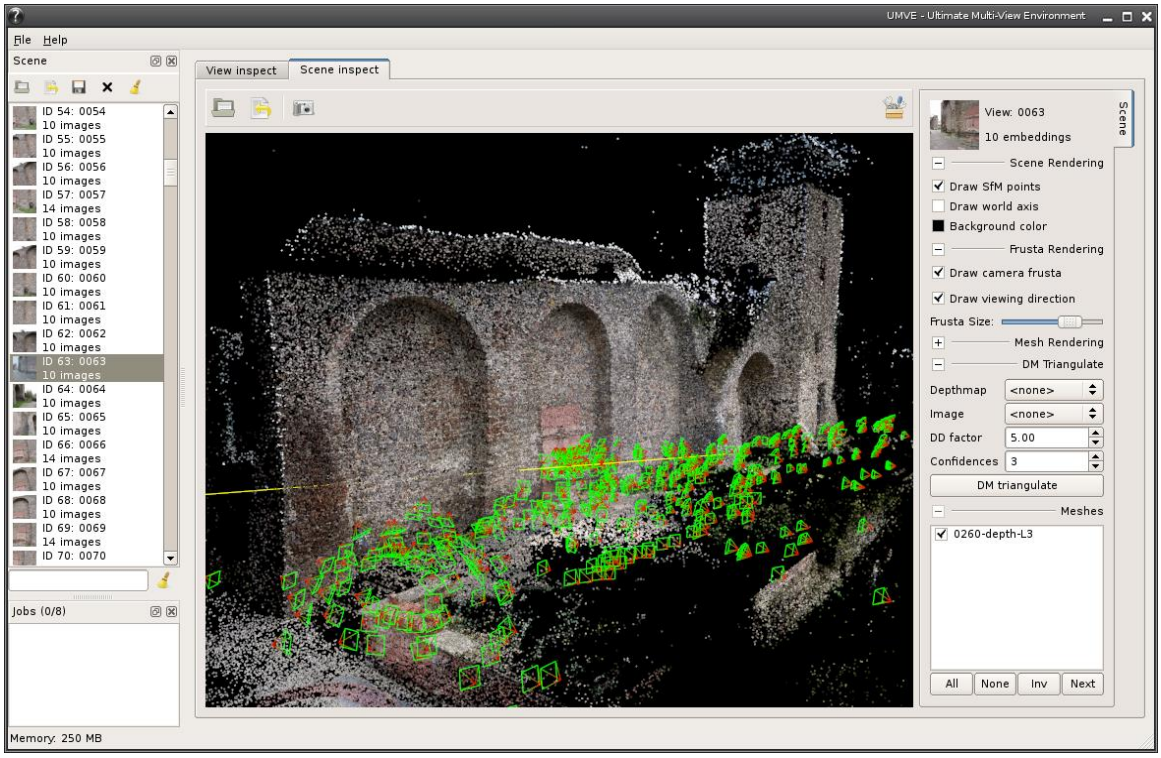


**Figure 7. The Multi-View Environment (MVE) GUI shows a sparse point cloud of a city wall in Darmstadt as well as the distribution of cameras which were registered for the scene.**

## 3.2 3DHOP

### 3.2.1 Description of Prototype

3DHOP (3D Heritage Online Presenter) is a collection of tools and templates for the creation of multimedia interactive Web presentations of digital 3D models. It allows easy visualization of 3D models directly inside HTML pages and the streaming of multiresolution 3D meshes and point clouds over HTTP, supporting the exploration of very large models on commodity computers and standard internet connections. 3DHOP is a cross-browser application. It is supported and successfully tested on the current stable version of Google Chrome, Mozilla Firefox, Internet Explorer and Opera. It requires a preprocessing of the input model to create a multi-resolution compressed data representation that is more suitable for the web streaming of very high-resolution 3D model [Ponchio et al. 2015]. One of the proposed template allows the space-time navigation of the temporal 3D data. It use the output of the Change Detection prototypes to allow the user to see how the 3D scene evolves in the time.

### 3.2.2 Impact for the Project

The prototype meets the objective of the Work Package 8.3 for the Web rendering of huge 3D model (triangular meshes and point cloud) with per-vertex color. The proposed GUI allow the free and smooth navigation of 3D model immediately using a multiresolution compressed format to load on the fly only the need data for the rendering. Furthermore, the proposed template for the space-time navigation (Figure 8) meets the objective of the Work Package 8.2 for the space-time data visualization allowing to the user to see in an effective way how the 3D scene evolves in the time.


Figure 8. 3DHOP user interface for the space-time navigation of an archeological excavation.

### 3.2.3 Usage

The prototype is composed by a set of tool to create the multi-resolution model using the NEXUS format (http://vcg.isti.cnr.it/nexus/) for the input model and by a JavaScript configurable library to create the Web viewer. The NEXUS multiresolution format is computed with the following command:

```
nxsbuild.exe inputModel.ply -o outputModel.nxs
```

where the first argument is the filename of the input 3D model and the option "-o" contains the filename where to save the final processing. The compression of the NEXUS format is obtained with the following command:

```
nxsedit.exe outputModel.nxs -z -o compressedOutput.nxs
```

For the space-time navigation template, we need to preprocess the models to store the per-vertex change value computed with the Change Detection prototype in the alpha value of the per-vertex color. The tool is the following:

```
changeToAlpha.exe input.ply output.ply
```

where the file `output.ply` can be used as input for the tool `nxsbuild.exe`. The web viewer uses a Trackball navigation paradigm:

- `Drag + press left mouse button`, to rotate;
- `wheel`, to zoom;
- `Ctrl+ drag + press left mouse button`, to pan;
- `Drag + press wheel`, to pan;
- `Alt+ press wheel`, z-panning;
- `Double click left mouse button`, re-centering of the trackball.

For the temporal navigation there is a simple slider to go from one time to the next in a smooth way.

The final NEXUS file can be load on the web server and used to create the Web page. For more info about the configuration and the layout of the web page where to load the model see the "How to do" page at the following link (http://3dhop.net/howto.php).

## 3.3 POTREE

### 3.3.1 Description of Prototype

Similar to 3DHop, Potree is a remote visualization tool that runs purely in a web browser. Unlike 3DHop, it focuses purely on point clouds (meshes are not supported). Potree uses WebGL and runs on all recent browsers supporting WebGL. Potree is focused on rendering very large point clouds. For this, it uses an internal hierarchical octree representation which is computed in a preprocessing step. It also offers different visualization modes, including screen-aligned splats, geometric splats, and Gaussian splats, to explore different quality/performance tradeoffs.

### 3.3.2 Impact for the Project

Potree contributes to Task 8.3, Mobile and Web Rendering. With the ever increasing sizes of data sets, it becomes infeasible to distribute full data sets of a 3D scene to all users. Therefore, technologies to distribute only the relevant parts of a data set on demand to users over the web are becoming crucial.

### 3.3.3 Usage (Demo)

Potree can be tested on the Harvest4D homepage under the link “3D Scenes”:

https://harvest4d.org/?page_id=860

At the time of writing, it is recommended to use Google Chome as web browser for its faster performance. Note that Potree runs perfectly well on Laptops due to its adaptive rendering technology. However, for Laptops with NVIDIA Optimus technology, it is recommended to start Chrome with the option “Run on high-performance NVIDIA processor” (available in the context menu of the application icon) to ensure that rendering is done by a dedicated NVIDIA card instead of slower, integrated GPUs.

The demo of Potree shows the Arene de Lutece data sets containing 45M points. See Figure 9 for an example screenshot of this scene with the user interface elements.


Figure 9. Potree user interface rendering the demo data set.

**Navigation:**

By default, Potree follows a “walk” navigation paradigm, i.e., the a,s,d,w keys allows moving forward, backward, and sideways, while dragging with the left mouse button rotates the camera.

A menu on the right-hand side shows a number of options:

- Points(m): number of points (in Million) to be displayed (allows adapting to rendering speed of computer)

- Pointsize: size of a “reference” splat in pixels. For fixed point size, this corresponds to the actual splat size.
- blendDepth: rendering parameter to determine the depth range in which overlapping splats are considered to belong to the same surface and are therefore blended
- SizeType: selection between a fixed point size (as given above) or a point size that is related to the view distance, thus allowing rendering surfaces without holes
- Quality: different rendering modes: screen-aligned squares or circles just rasterize a primitive. “splats” renders high-quality Gaussian splats that are blended in a given depth range, as in [Botsch et al. 2005]. “Interpolation” renders screen-aligned paraboloids that produces surfaces in a similar way as a Voronoi tessellation [Schuetz et al. 2015]
- Stats: enables display of rendering statistics like frame rate, visible nodes and visible points
- BoundingBox: shows the bounding boxes of the octree data structure (see Figure 10).

On the lower right side, there are further icons to enable operations like measurement, clipping, or switching to fly or trackball navigation modes.

### 3.3.4 Usage (own models)

In order to test Potree with other models, the current version of Potree can be downloaded from

http://www.potree.org/

The PotreeConverter is an offline tool to convert input point clouds into the Potree hierarchical file format.

Potree itself is a client-side WebGL application but needs to be deployed on a web server, along with the converted point cloud data, to be accessed inside web browsers.

Documentation for the two components can be found at the mentioned URL. Figure 11 shows a point cloud of Barcelona with the Sagrada Familia in the background, rendered in Potree.

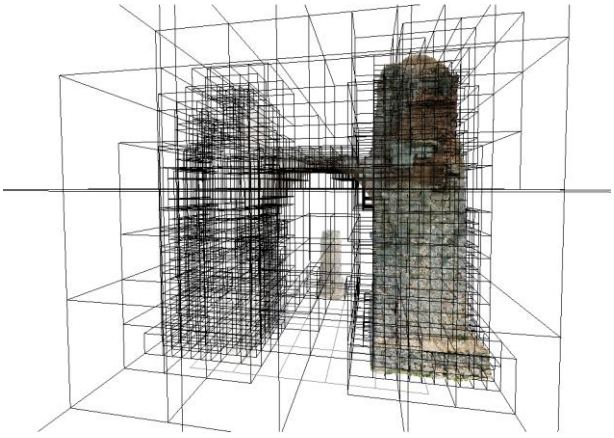

Figure 10. Visualization of the Potree octree data structure.

Figure 11. Barcelona point cloud with Sagrada Familia.

## 3.4 CHROMA VIEWER

### 3.4.1 Description of Prototype

This prototype showcases the effect of improved depth perception based on the chromostereopsis phenomenon (see Figure 12), as described in [Schemali and Eisemann 2014].

The program is developed in C++ (using OpenGL, GLSL, and wxWidget), with GPU support. It loads various file formats.

Please notice, to make the software usable right away, we provided a simple city model, but need to point out that it is *not* part of the deliverable. Instead it was purchased by us and we cannot warrant any rights regarding this model.

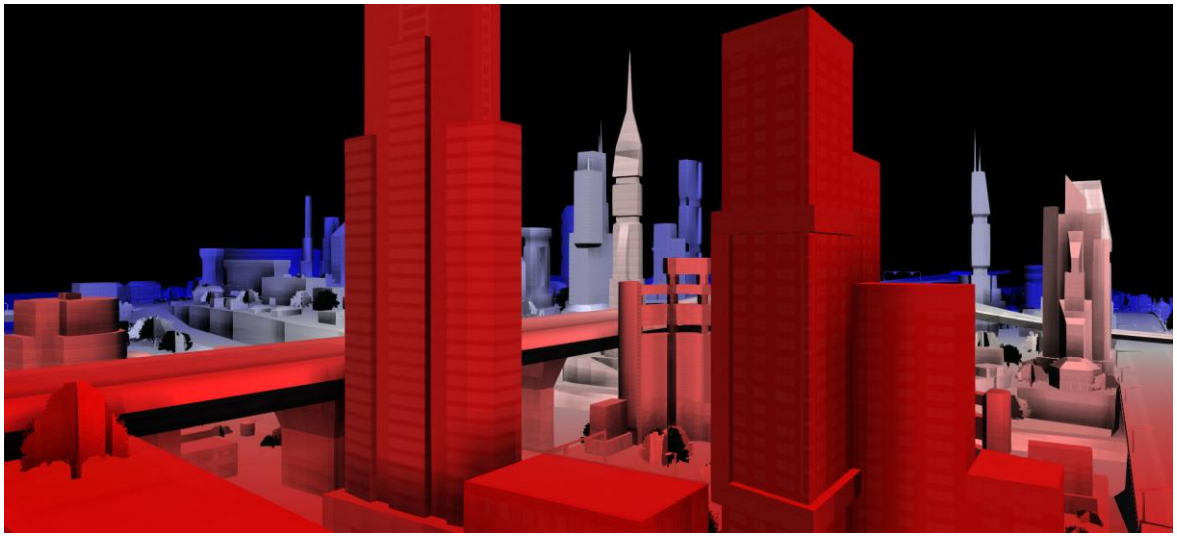

Figure 12. The prototype provides improved depth perception when wearing the ChromaDepth glasses.

### 3.4.2 Impact for the Project

This prototype provides a tool for viewing 3D data on a regular, trichromatic 2D screen with an improved perception of the third dimension (the z-axis) compared to regular projection methods. All this requires are the ChromoDepth glasses, which are easy to use and cheap to obtain. Using our tool, the user achieves a better understanding of 3D data, which is crucial when visualizing complex space-time data.

### 3.4.3 Usage

Execute bin/Win32/Research.exe, then proceed to the directory Content/jingtang/ and select and load Chromo.xml.

Navigation is possible via the cursor keys and page up/down or WASDQE. To see the improved depth perception effect, ChromaDepth glasses are required.

# 4 REFERENCES


- [Palma et al. 2013]
  Gianpaolo Palma, Nicola Desogus, Paolo Cignoni and Roberto Scopigno
  *Surface Light Field from Video Acquired in Uncontrolled Settings*
  In: International Conference DigitalHeritage, 2013

- [Fuhrmann et al. 2014a]
  Simon Fuhrmann, Fabian Langguth and Michael Goesele
  *MVE – A Multi-View Reconstruction Environment*
  In: Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage, 2014.

- [Fuhrmann et al. 2014b]
  Simon Fuhrmann and Michael Goesele
  *Floating Scale Surface Reconstruction*
  In: ACM Transactions on Graphics (Proceedings of SIGGRAPH), 2014.

- [Schemali et al. 2014]
  Leila Schemali and Elmar Eisemann
  *ChromoStereoscopic Rendering for Trichromatic Displays*
  In: International Symposium on Non-Photorealistic Animation and Rendering, 2014.

- [Legrand et al. 2015]
  Hélène Legrand and Tamy Boubekeur
  *Morton Integrals for High Speed Geometry Simplification*
  In: High Performance Graphics, 2015

- [Golla et al. 2015]
  Tim Golla and Reinhard Klein
  *Real-time Point Cloud Compression*
  In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015

- [Ponchio et al. 2015]
  Federico Ponchio and Matteo Dellepiane
  *Fast decompression for web-based view-dependent 3D rendering*
  In: International Conference on 3D Web Technology, 2015

- [Schuetz et al. 2015]
  Markus Schuetz and Michael Wimmer
  *High-Quality Point Based Rendering Using Fast Single Pass Interpolation*
  In: International Conference DigitalHeritage, 2015

- [Palma et al. 2016]
  Gianpaolo Palma, Paolo Cignoni, Tamy Boubekeur and Roberto Scopigno
  *Detection of Geometric Temporal Changes in Point Clouds*
  In: Computer Graphics Forum (In press), 2016

- [Dado et al. 2016]
  Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery and Elmar Eisemann
  *Geometry and Material Compression in High-resolution Voxelized Scenes*
  In: Eurographics, 2016