

Project acronym: LADIO Project number: 731970 Work package: Deliverable number and name: D4.1: Implement OpenCMPMVS	Title: Implement OpenCMPMVS Work Package: WP4 Version: 1 Date: August 31, 2017 Author: Tomas Pajdla
Type: <input type="checkbox"/> Report <input type="checkbox"/> Demonstrator, pilot, prototype <input type="checkbox"/> Website, patent filings, videos, etc. <input checked="" type="checkbox"/> Other	Co-Author(s): Fabien Castan Konstantin Pogorelov Yann Lanthony Benoit Maujean To: Albert Gauthier, Project Officer
Status: <input type="checkbox"/> Draft <input type="checkbox"/> To be reviewed <input type="checkbox"/> Proposal <input checked="" type="checkbox"/> Final / Released to EC	Confidentiality: <input checked="" type="checkbox"/> PU – Public <input type="checkbox"/> CO – Confidential <input type="checkbox"/> CL - Classified
Revision: Final	
Contents: Deliverable 4.1. Implement OpenCMPMVS	

Table of contents

Table of contents	2
Introduction	3
Multi-platform	4
OpenMVG to CMPMVS	4
Licensing	4
Code cleaning	5
Quality	5
Performances	6
IOs	7
Understanding algorithms and Verifications	7
Integration in Mikros Image pipeline	8
Testing and evaluation	9
Evaluation Datasets	9
Publicly available datasets	10
Production datasets	11
Software release	14
Future improvements	14
References	15

Introduction

CTU developed CMPMVS, a Multi-View Stereo (MVS) reconstruction pipeline, from 2010 to 2014 and released closed-source binaries named “version 0.6” in 2012. This deliverable was one of the most challenging deliverable of LADIO. The ambition was to provide a stable and multi-platform Multi-View Stereo pipeline that can be installed in production from the existing CMPMVS source code. The ambition was also to clean and document the code to make a public release under MPLv2 license. To give an idea of the amount of work, the original source code contained 219790 lines of code without any documentation. It has been cleaned, commented, optimized and extended. The new version released in this deliverable now contains 61768 lines of code.

The MVS pipeline consist in the estimation of a textured dense model from known cameras poses provided by the Structure-from-Motion pipeline. Here is an overview of the steps of the MVS pipeline:

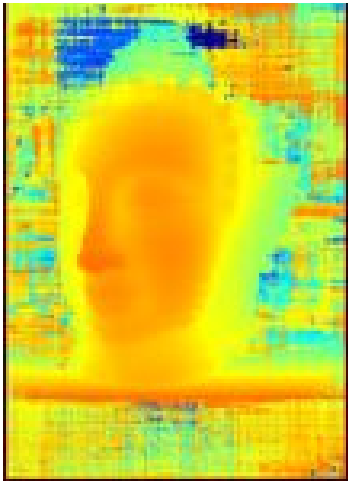

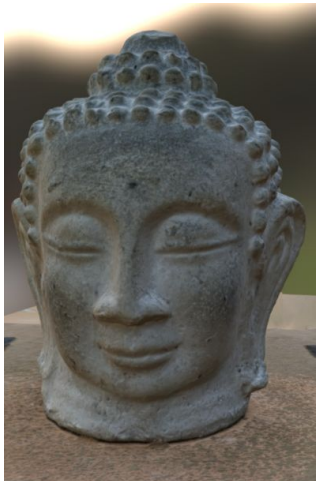
Depth Maps Generation	Depth Map Consistency Filtering	Meshing	Texturing
First estimation with a plane sweeping approach called Semi Global Matching (SGM) based on a similarity estimation in a global discretized volume. Then this result is refined per pixel to remove the depth discretization.	Filtering step based on the score of similarity and consistency across the results between depth maps.	Depth map fusion based on octree representation. Tetrahedralization of this dense point cloud. Full/Empty voting procedure on all the tetrahedra and a Graph Cut procedure to choose the final surface. And finally some filtering steps to clean the result.	Compute the automatic UV mapping (used to map all 3D triangles into 2D texture files). based on the visibility information associated to each vertex. Then, use color information from the input image into the final texture.
			

Table 1 : MVS pipeline

Multi-platform

The original source code had been fully developed on Windows. A large part of post-production companies, including Mikros Image, are using Linux. And having a multi-platform solution is also critical to make a successful open source project.

So first, we removed all the code relying on DirectX. We lost a few features regarding the simplification of the mesh, or some rendering generation of the results; but these features were currently not used.

We moved to a new cross-platform C++ build system, based on CMake and removed the original Microsoft Visual Studio project files.

Finally, we replaced all file system access based on Windows API with the more generic approach of `boost::filesystem`¹.

OpenMVG to CMPMVS

The standard CMPMVS input was a text file with camera poses and undistorted images. The first step of CMPMVS was to extract features and compute matches based on SiftGPU library. We have decided to remove the dependencies to these first steps to avoid the “research only” license and also to avoid unnecessary computation. Instead we directly export the 3D point cloud from openMVG into CMPMVS data structures.

Licensing

The research code created by CTU was based on many libraries and unfortunately some of them were licensed under GPL which is not compatible with MPLv2, the licensing model chosen by LADIO consortium.

So we analyzed all dependencies to remove all unnecessary libraries and replace the GPL libraries by other alternatives with compatible licensing. This has required a large rewrite in many parts of the software.

Dependencies		Solution
SiftGPU	Research only	Replaced by a direct exporter from openMVG
Maxflow	GPL	Replaced by boost boykov_kolmogorov_max_flow (Boost Software License)

¹ http://www.boost.org/doc/libs/1_64_0/libs/filesystem/doc/index.htm

CGAL	GPL	Replaced by Geogram ² (3-clauses BSD License)
CudaTemplates	GPL	Replaced by native CUDA code
octree	GPL	Removed
QSPLAT	ARR	Removed
colorBased	GPL	Removed
DataInterface	GPL	Removed
DirectX	Windows only	Removed

Table 2 : Library dependencies

Code cleaning

We used clang-format³ and cppcheck⁴ to clean the formatting of the source code. We even used clang-tidy⁵ to modernize it to C++11. There was no notion of constness in the declaration of variables or objects, so we fixed it manually on a large part.

We have spent a huge amount of time in testing parameters, testing different methods implemented for the same task before removing unused or useless methods.

We also replaced the custom implementation for parsing configuration files (INI) with the standard boost parser library.

Quality

When testing the software on many datasets, we discovered quality regressions on large datasets between the latest version from CTU and the public closed-source 0.6 release. The challenge was that those regressions were not visible on small datasets. So we analyzed one year of GIT history (the source code repository) to discover 3 problematic changes added after the 0.6 release. Two of them were thresholds modifications which decreased the quality on large datasets. And the third one was more ambiguous. It was a change in the depth map computation (with SGM plane sweeping) but one implementation can have better results on one dataset but not on another one. One implementation was using the best similarity values and the other one was using the average between all cameras used. We made a new version using the 2nd best similarity to get the best compromise on all datasets.

² <http://alice.loria.fr/index.php/software/4-library/75-geogram.html>

³ <https://clang.llvm.org/docs/ClangFormat.html>

⁴ <http://cppcheck.sourceforge.net>

⁵ <http://clang.llvm.org/extra/clang-tidy/>

At the end of the tetrahedralization, we also added a very basic filtering step to invert the full/empty status of the tetrahedra with 3 facets of the same status, which slightly reduces meshing artefacts.

Performances

First, we have performed a memory usage analysis. We have found unnecessary transfer operations between host and GPU memory in CUDA-optimized implementation of plane sweeping code. We have reorganized the memory usage to use in-place data access whenever it possible.

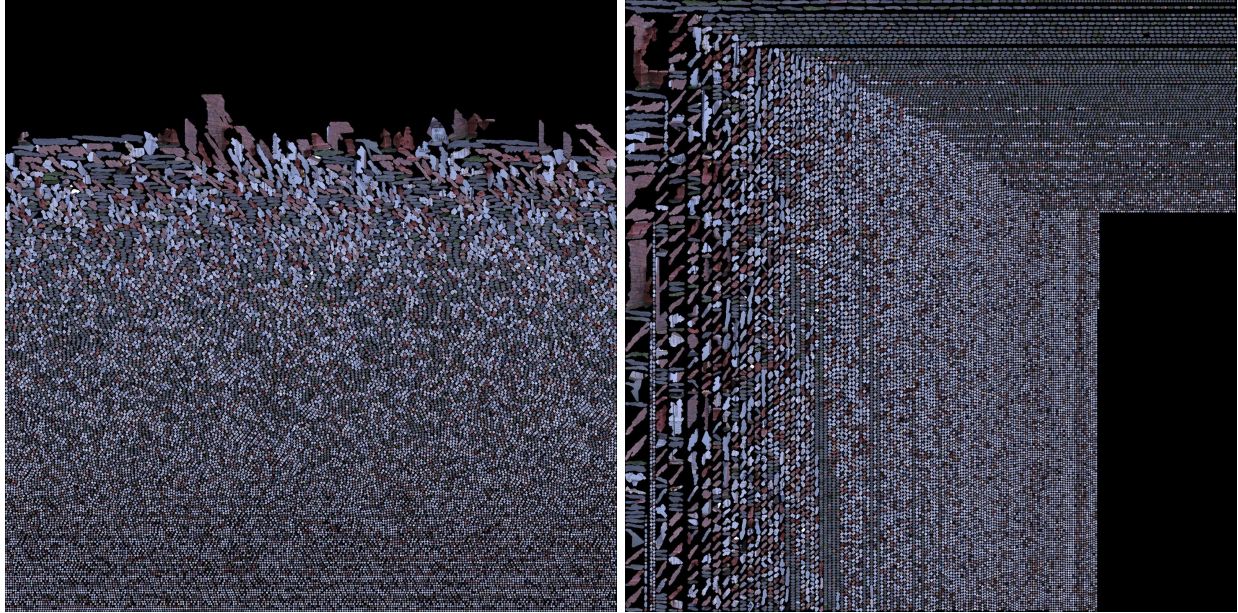
The last step of the pipeline is the texturing of the mesh. On small datasets, the computation time was cheap but it became surprisingly expensive on larger datasets.

When it was not possible to load all textures in memory, the software was loading/unloading files many times which was consuming a large part of the processing time. We have reorganized the code to load the input images once per texture file and directly contribute to all charts of this texture file.

We have also done manual loop unrolling on some critical functions to further optimize it.

One expensive part of the texturing was also the UVs creation to map triangles into the smallest number of texture files. The idea is to minimize the amount of unused space in texture files. But the amount of computation time required by the current implementation was not reasonable. So we have decided to slightly reduce the spatial optimization of charts grouping into texture files, in order to dramatically improve the performances and to make a compromise between the number of texture files generated and the amount of computation time. Instead of trying to minimize the amount of empty space by computing the exact envelope of each chart (expensive), we simply rely on their rectangular bounding boxes. We then sort them by area and dispatch them in a tree-based layout.

Additionally, when a texture atlas can't host the next chart, we fill the empty space using the smallest ones. This allows to keep a good ratio between computation time and the number of generated texture files.



*Figure 1 : Texture file sample with
Left: the previous, compute intensive, UV computation
Right: the new simplified UV computation*

We discovered that the most expensive step of the meshing process is the voting procedure inside the tetrahedralization. But a large part of this computation time was contention created by multithreading locking issues. By changing the global locking into atomic or local locking strategies, we obtained a significant improvement. On a medium size dataset, the absolute time for the meshing step went from 56 to 30 minutes (and from 1003 to 426 minutes in CPU time).

IOs

There was a lot of useless intermediate files stored on disk. We analyzed all the Inputs/Outputs of the CMPMVS pipeline and removed a large part of the unnecessary files, but we still have work planned to further improve file storage.

Understanding algorithms and Verifications

We analyzed a large part of the pipeline, found the corresponding papers, verified the differences between the implementation and the papers and fixed minor bugs.

We have finished to check and analyze the Depth Map Creation, the Tetrahedralization, the Graph Cut and the Texturing. We still have to analyze in more details the depth map refinement and the octree creation.

Integration in Mikros Image pipeline

To use the Multi-view Stereo reconstruction pipeline at Mikros Image, the main challenge was the integration and parallelization on render farm. We created new binaries which enable to compute the MVS pipeline step by step. These binaries have also an option to process a subset of the cameras which allows to parallelize the computation and refinements of depth maps on different machines in renderfarm. Mikros Image used the commercial render farm engine Tractor (from Pixar).

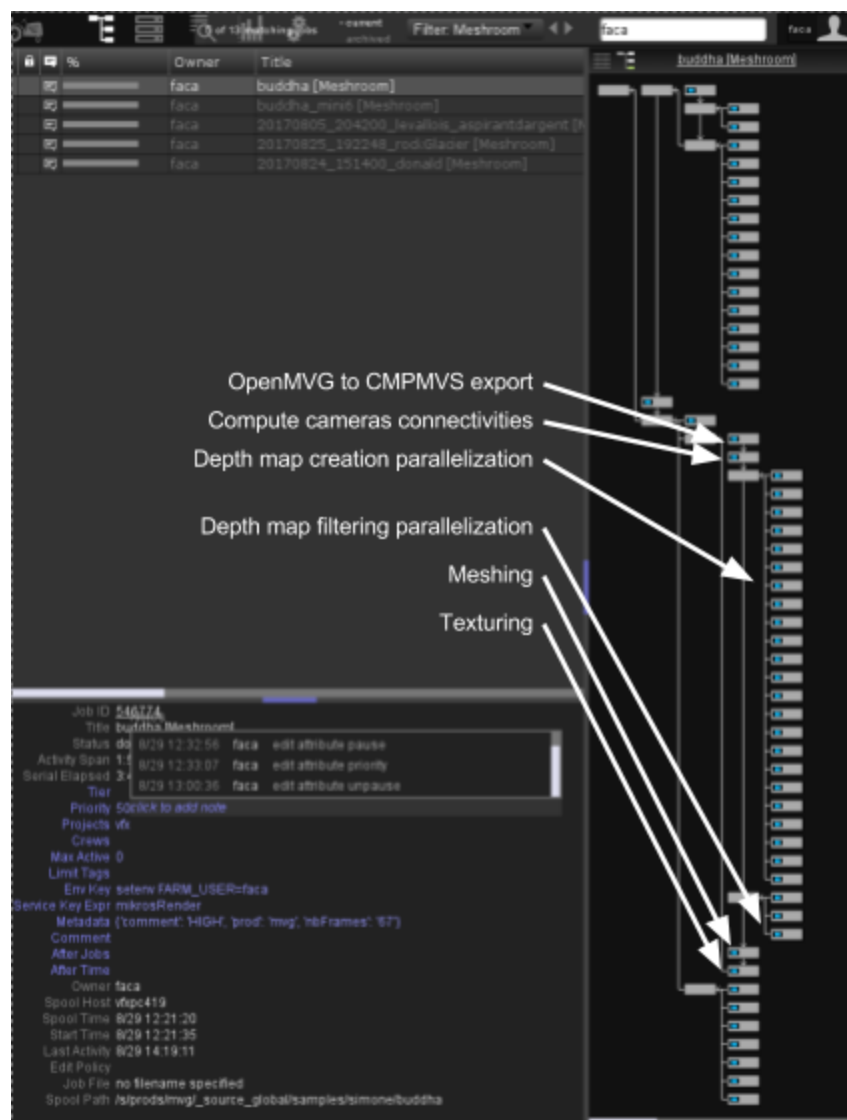


Figure 2 : MVS pipeline integrated into the Tractor render farm

The MVS pipeline has been integrated in production at Mikros Image and is already used on many productions.

Testing and evaluation

Evaluation Datasets

A new publication by Intel Labs (Tanks and Temples) appeared in August 2017 for benchmarking 3D reconstruction pipelines [Knapitsch2017]. The benchmark sequences were acquired outside the lab, in realistic conditions. Ground-truth data was captured using an industrial laser scanner. The benchmark includes both outdoor scenes and indoor environments. The original approach of this benchmark is to focus on evaluation of the results of the full SfM+MVS pipelines. We have started testing the new AliceVision pipeline on it, but not yet performed the final evaluation that can only be done by submission of the results to the platform. So we have to finish some identified improvements before making a public submission.

Here are some preliminary samples of our first tests :



Figure 3 : San Francisco Music Concourse dataset



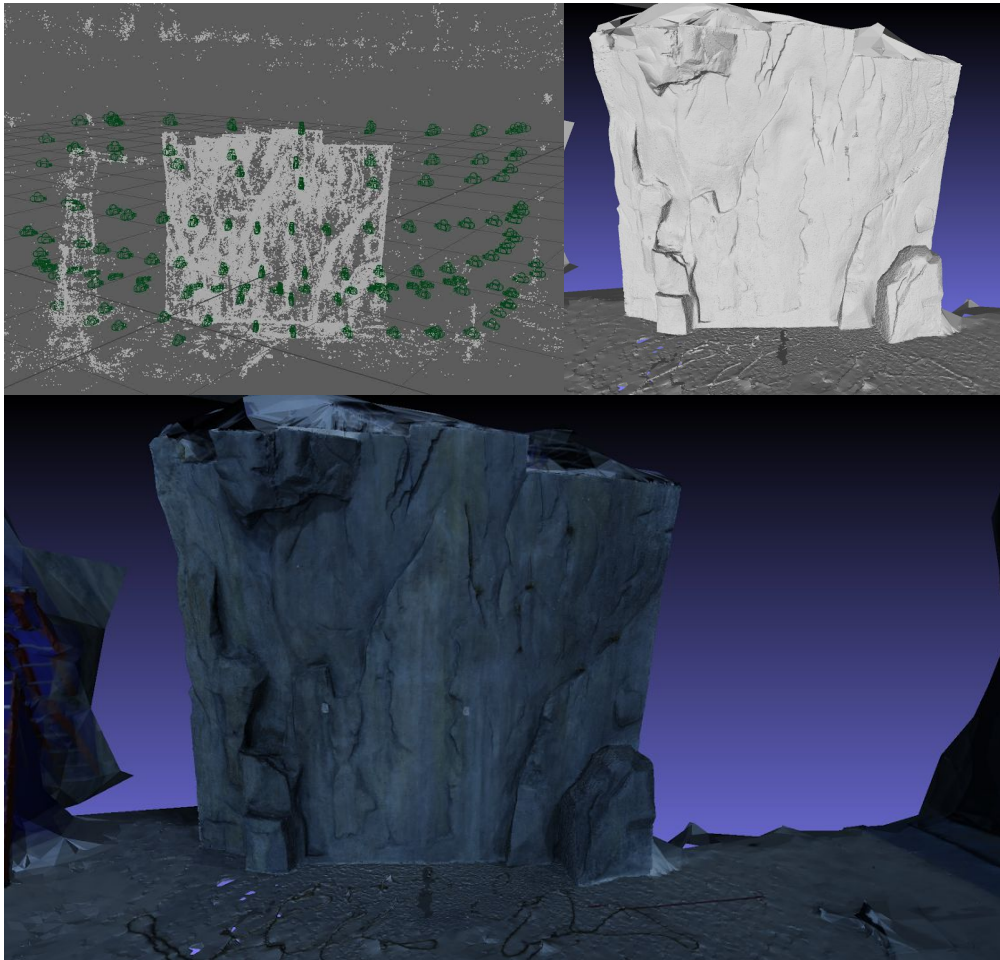
Figure 4 : Tank dataset

Publicly available datasets

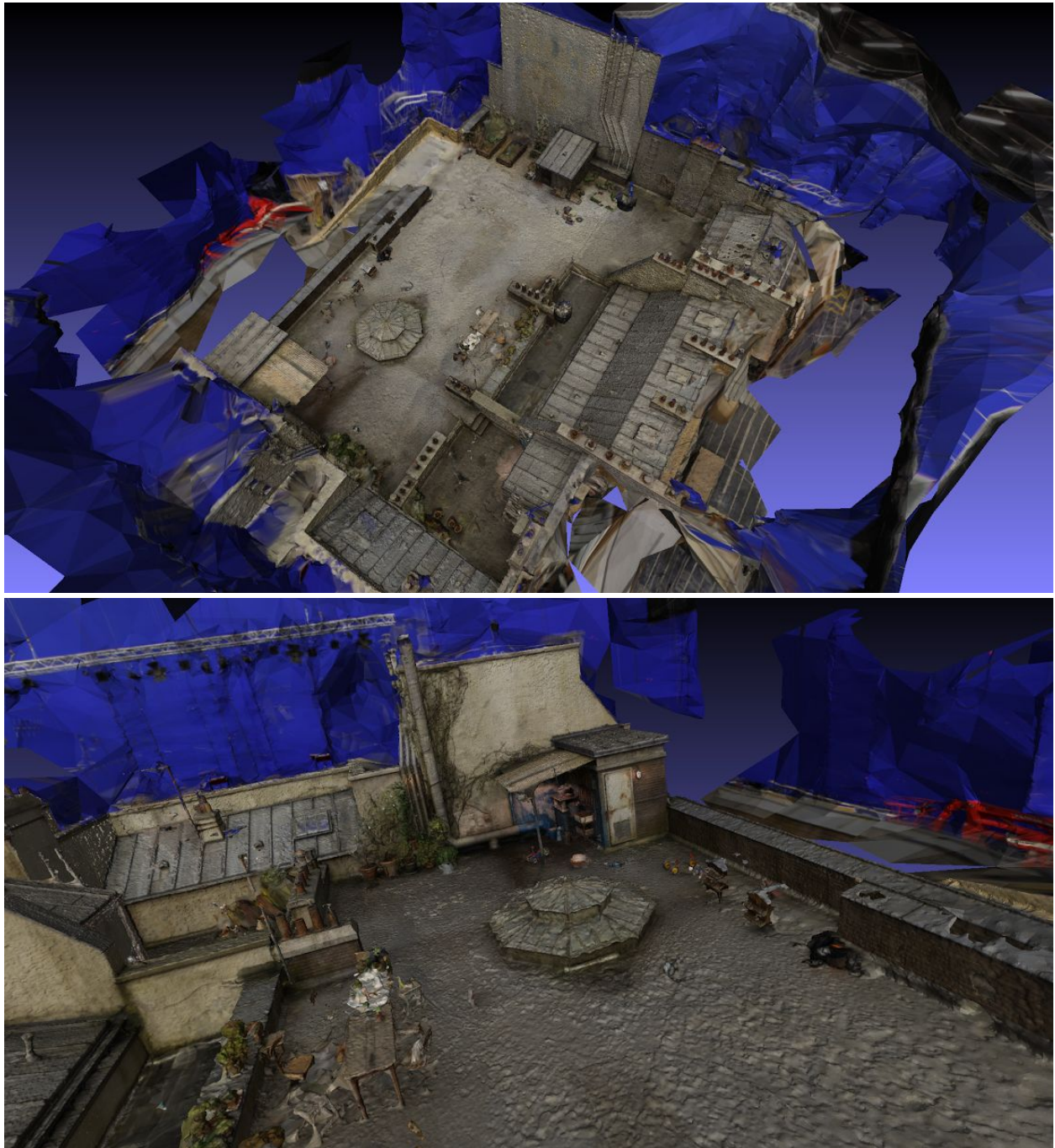
To demonstrate the capabilities of the AliceVision pipeline we created an account on the online 3D content platform “Sketchfab” and uploaded multiple datasets that can be visualized in 3D directly in a web browser.

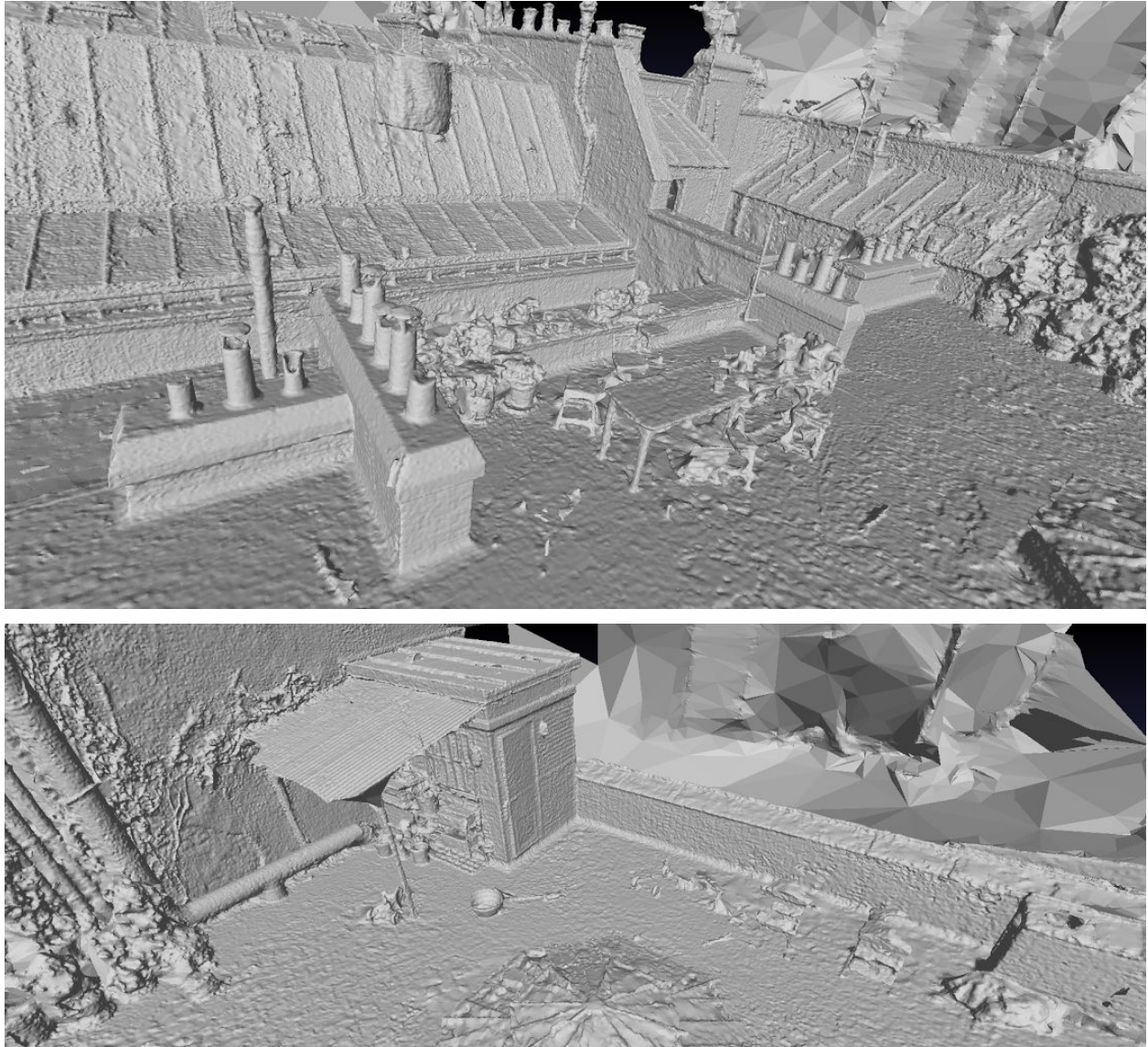
We have currently 11 models available: <https://sketchfab.com/AliceVision>

Production datasets

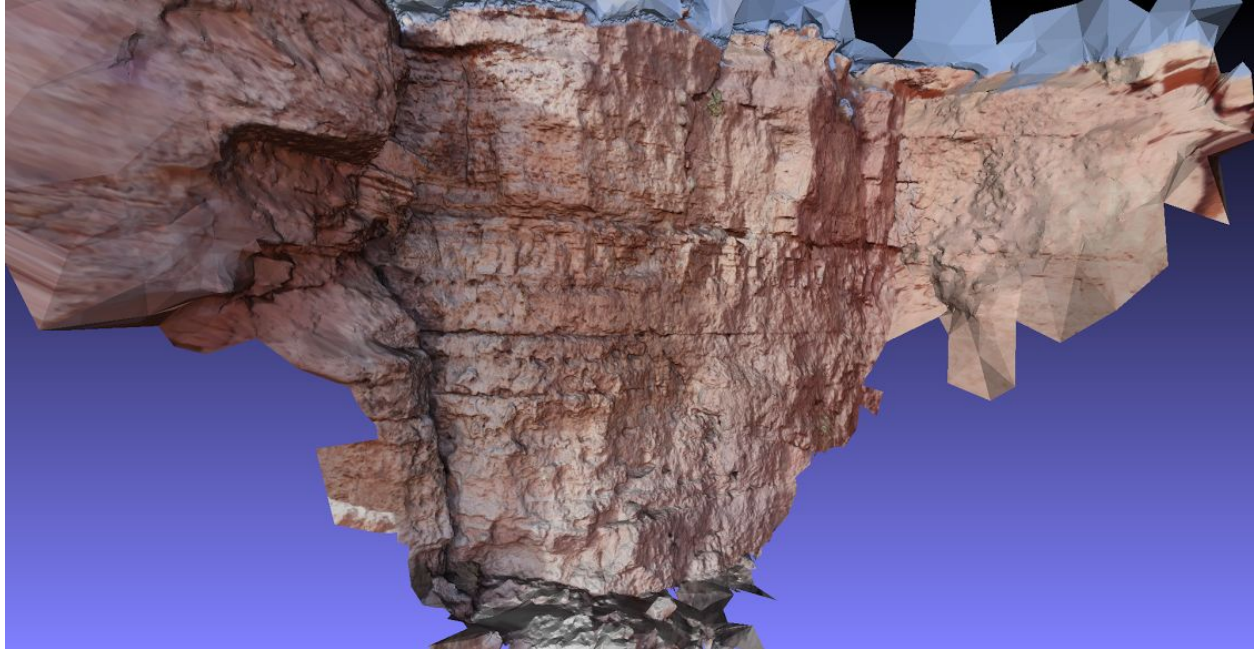


*Figure 5 : Artificial rocks for the commercial “Volkswagen Humans”
director Martin Kolina, production Big Production
(top left: point cloud and cameras from SfM, top right: mesh, bottom: textured mesh)*





*Figure 6 : 3D Reconstructions of the roof of a Parisian building for feature film “Santa & Cie”
(director Alain Chabat, production Légende Productions, Gaumont)*



*Figure 7 : Outdoor scene of rocks in France for feature film “Revenge”
(director Coralie Fargeat, production Monkey Pack Films)*

Software release

We now have a stable MVS pipeline that has been tested and used in real post-production use cases as planned for this deliverable. The corresponding source code is released in attachment to this deliverable and of course through the Cordis web pages.

We decided to postpone the release of the source code on GitHub, in order to merge our SfM and MVS pipelines into one repository and release it with the new version of Meshroom to maximize the marketing impact of this open source release.

Future improvements

After all the analysis of the Multi-view Stereo reconstruction pipeline, we have identified multiple steps that could be improved in Task 4.2.

In the depth map fusion, we use a uniform voxel representation. We can clearly see that adding new images at a different scale dramatically decrease the amount of details in the first area. So we plan to extend our internal representation to efficiently capture multiple levels of detail.

We have also identified that the complexity of the SGM (Semi Global Matching) code can be reduced by splitting the image into a grid to limit the amount of depth values to be estimated per pixel.

We have also identified that the depth map refinement step could generate too much noise. We

plan to improve it in 2 ways. First, we would like to estimate the error and confidence in the depth map result and use them in the depth map fusion to create multiple 3d point candidates with a weighting strategy. Secondly, we would like to add albedo/lighting constraints into the refinement step as we have started to prototype in [Melou2017a] and [Melou2017b].

Finally, we also plan to integrate standard mesh simplification to obtain smooth and low complexity meshes and better auto-UVs for texturing.

References

- [Knapitsch2017] Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction, Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun, 2017
<http://vladlen.info/publications/tanks-temples-benchmarking-large-scale-scene-reconstruction>
- [Melou2017a] Beyond Multi-view Stereo: Shading-Reflectance Decomposition, J M  lou, Y Qu  au, JD Durou, F Castan, D Cremers, 2017
- [Melou2017b] Dense Multi-view 3D-reconstruction Without Dense Correspondences, Y Qu  au, J M  lou, JD Durou, D Cremers, 2017