OpenMVG: Open Multiple View Geometry

Pierre Moulon[1], Pascal Monasse[2], Romuald Perrot[3], and Renaud Marlet[2]

[1] Zillow Group pierrem@zillowgroup.com
[2] LIGM, UMR 8049, École des Ponts, UPE, Champs-sur-Marne, France
{pascal.monasse,renaud.marlet}@enpc.fr
[3] Université de Poitiers - Laboratoire XLIM, UMR CNRS 7252, Futuroscope, France
romuald.perrot@univ-poitiers.fr

Abstract. The OpenMVG C++ library provides a vast collection of multiple-view geometry tools and algorithms to spread the usage of computer vision and structure-from-motion techniques. Close to the state-of-the-art in its domain, it provides an easy access to common tools used in 3D reconstruction from images. Following the credo "Keep it simple, keep it maintainable" the library is designed as a modular collection of algorithms, libraries and binaries that can be used independently or as bricks to build larger systems. Thanks to its strict test driven development, the library is packaged with unit-test code samples that make the library easy to learn, modify and use. Since its first release in 2013 under the MPL2 license, OpenMVG has gathered an active community of users and contributors from many fields, spanning hobbyists, students, computer vision experts, and industry members.

Keywords: Reproducible research, Computer Vision, Multiple-view geometry, 3D reconstruction, Structure from Motion, C++, open source

1 Introduction

Computer vision is used extensively nowadays, even by our pocket devices thanks to our smartphones. Some of the computer visions tasks they perform include stitching images to create a planar mosaic and a spherical panorama, using image content-based search retrieval (bar codes, similar product search), and performing 3D reconstruction from photographs. Moreover, 3D content creation from images is more and more used: *e.g.*, digitizing our world for offline (surveying, cartography, VFX) or for online applications (gaming, AR/VR), digitizing dynamic elements for gaming (Kinect), and autonomous navigation of vehicles are all trendy topics.

Regarding the large scope of applications and the diverse needs of computer vision techniques relating to 3D reconstruction, it is clear that the community can have a major gain if a common framework can be used to communicate, make experiments, and build new prototypes. Often, high level and general purpose tools like Matlab or Intel IPP[4] are used, but they are not the best choice, since beside being costly they do not have all

[4] Intel Integrated Performance Primitives https://software.intel.com/en-us/intel-ipp/

the needed algorithms implemented. They include only a subset of the major Multiple-View-Geometry (MVG) algorithms and are not specialized for Structure from Motion (SfM). Other alternatives like OpenCV can be compelling, but again, only partial implementations exist. Since these alternatives want to cover a large scope of applications they do not focus on multiple view geometry and 3D reconstruction from images in an efficient way.

2 Photogrammetry software alternatives

Photogrammetry is the science of making measurements from photographs, especially for recovering the exact positions of surface points. The domain is mature; as witnessed on the internet[5], more than 80 software solutions (commercial, free or open source) are listed. 3D reconstruction from images knows a second breath nowadays, since the emergence of UAV is making a true revolution in land surveying, the acquisition of low altitude images being now a cheap and simple task.

We make here a distinction between multiple view geometry (MVG) and multiple view stereovision (MVS) software. The former is concerned with recovering camera locations and orientations from the data (images and camera intrinsics); it delivers also a sparse set of 3D points, built by triangulation from the feature points observed in the photographs. The latter deals with the dense 3D reconstruction; its output can be a dense point cloud, a faceted surface (mesh), or a set of planes, which can be visualized as a realistic 3D rendering of the scene. It relies on MVG to achieve that.

Commercial software. The solutions, integrating MVG and MVS in single products, are clustered around the markets they are addressing: UAV land surveying is addressed by the Pix4D products[6] and by DroneDeploy software[7], while the large scale close range photogrammetry market is mostly addressed by the Bentley ContextCapture[8] and CapturingReality[9] software.

Free software. Visual SfM (VSfM [13]) is a solution that is largely used. The main point that eases its usage is due to the fact the software is delivered with a graphical user interface (GUI) and that it uses multi-threading on CPU and GPU for high efficiency.

Open Source solutions. While some solutions deliver a software program (Bundler [1], ColMap [39], MicMac [40], PMVS [35], CMVS [34]), others deliver both a collection of libraries and softwares (MVE [37], OpenMVG, OpenSfM [41], OpenMVS [42], TheiaSfM [38])[10]. Combining OpenMVG with OpenMVS or MVE provides an end-to-end open-source photogrammetry pipeline.

[5] `https://en.wikipedia.org/wiki/Comparison_of_photogrammetry_software#Comparison`
[6] `https://pix4d.com/`
[7] `https://www.dronedeploy.com/`
[8] `https://www.bentley.com/en/products/brands/contextcapture`
[9] `https://www.capturingreality.com/`
[10] See `https://github.com/openMVG/awesome_3DReconstruction_list`

From a user point of view, commercial and freeware solutions are like black boxes that cannot be tuned or modified for the user needs, while open source solutions provide complete pipelines and interface to multiple view geometry algorithms that can be modified and customized.

Regarding the reproducible research side, open-source alternatives are interesting since they deliver a transparent implementation of some algorithms that anyone can test, use, check, and modify. While it is not easy to implement an algorithm in the right way, some software guidelines rules can help to provide transparency and fairness to the respective algorithm or paper implementation.

It is interesting to note that Bundler (more than 2000 citations) and VSfM (more than 200 citations) projects have helped spreading the usage of Structure from Motion into the computer vision community. Bundler was released as a PhD code dump under an open-source license. It caught a lot of attraction since it offers an easy to use command line software. Unfortunately, it did not receive any major evolution, cleanup or updates since its initial release. Although also initially developed during a PhD preparation [2], OpenMVG was designed from the start with the idea of providing a collection of tools, a test driven high quality library, a regular support and up to date features.

3 OpenMVG design

This section gives an overview of OpenMVG[11] functionality and design. OpenMVG goals are multiple, providing the computer vision community with: (i) an easy access to accurate implementation of multiple view geometry algorithms; (ii) an understandable source code library; (iii) a set of tools used to build complete applications such as SfM pipelines. OpenMVG includes functionalities for image loading and processing, feature detection and matching, multi-view geometry solvers and provides an easy access to linear algebra and optimization frameworks. It delivers a collection of modular core features arranged in small libraries (Table 1) that can be used independently or as building blocks in an entire pipeline in order to perform 3D reconstruction from images (SfM) or localize images into an existing 3D reconstruction.

OpenMVG is written in standard C++11 and uses the CMake build system bringing portable builds on x86, x86_64 and ARM targets. It relies on the Eigen [10] library to perform high performance linear algebra manipulations, the Ceres-solver [9] to solve large scale non-linear minimization such as bundle adjustment, and OSI-CLP [14] as a linear programming solver. Thanks to well documented and transparent interfaces, OpenMVG can be extended or interfaced with other software and even use custom data in a few easy steps.

OpenMVG goals. OpenMVG goals are twofold:

- an educational side: to provide easy to read and accurate implementation of state of the art "classic algorithms" that the community considers as "common knowledge".
- a knowledge diffusion side: to spread the usage of the computer vision techniques to the community by delivering easy to use code, libraries, samples, and pipelines.

[11] `https://github.com/openMVG/openMVG/`

OpenMVG philosophy. In order to complete its vision in the best way, OpenMVG follows as guidelines the credo **"Keep it simple, keep it maintainable"**. OpenMVG authors believe that it is more important for the reproducible research side to have a code that is easy to read and use than a code that is fast but difficult to edit due to cumbersome optimization.

Beside the readability criteria, algorithm effectiveness must be also demonstrated. This goal is achieved using Test Driven Development. The main motivations for using unit testing is that it helps:

- to assert that algorithm and code are working as expected;
- to perform non regression tests following code updates;
- to provide usage examples in real context;
- people to implement new things.

Thanks to its large collection of unit test, external users can integrate their new method, test if it works as expected, and use it later in a larger context with no new code requirements.

OpenMVG license. OpenMVG is licensed under the MPL2 (*Mozilla Public License 2*). The choice has been made to maximize its usage, even by industry partners, but force somehow contribution back to the existing library files. This license is similar to the well-known LGPL, but it has a file extent: a modification or a bug fix inside an existing file must be shared under the same license. However the license allows a larger work to be released under different terms and so enables the usage of OpenMVG powered code in a commercial application. As shown by the number of external contributions, the community is comfortable with this license (31 contributors, 100 Pull Requests, 500 issues handled).

4 OpenMVG functionalities

OpenMVG provides algorithms that perform tasks like image loading and processing, feature detection and matching, multi-view geometry solvers and an easy access to linear algebra and optimization frameworks. The different modules/libraries are listed in Table 1.

4.1 Generic photogrammetry data description

The OpenMVG processing pipeline is articulated around the `SfM_Data` container. It acts as a spine and allows to have a smooth communication between the tools during the whole process. This data container stores relations between images and their related data: *abstract views* (image metadata, IDs to the camera model and pose), *abstract camera models*, *camera poses*, *structure landmarks and image observations IDs*. Thanks to a generic I/O interface this container can be saved in binary (for compactness and fast reading/saving) or in JSON/XML (for easy transfer to third party projects). Thanks to this container an effective pipeline can be built for different purposes, like 3D reconstruction from images.

module name	usage
cameras	Abstract camera model
features	Abstract region description (point position, descriptors)
geometry	3D transformation (similarity, 3D pose)
matching	Abstract nearest neighbour interface
multiview	Multiple View Geometry solvers
robust_estimation	Robust estimation framework
stl	C++ STL extensions
tracks	Un-ordered feature tracking
exif	Exif data parsing
geodesy	Geodesy transformation
graph	Graph analysis tools
linearProgramming	Abstract linear programming interface
matching_image_collection	Abstract interface to match image collection
numeric	Linear algebra tools
sfm	reconstruction pipeline (SfM & Localization)
system	Benchmarking tools

Table 1: Set of OpenMVG modules

4.2 Image processing

OpenMVG provides a simple image handling module. The generic image class acts as a 2D template pixel container based on the Eigen matrix structure. It allows to have all Eigen optimizations available to perform efficient image processing operations. Built on top of this class, the user can have access to:

- Image I/O (png, jpeg, tiff);
- Image sampling (nearest, linear, cubic, spline) and warping;
- Primitive drawing (line, circle, ellipse);
- Color space conversion;
- Image filtering (gradient computation, linear convolution, non-Linear diffusion [22]).

4.3 Feature extraction and description

Detecting distinctive, repeatable image points and descriptors is a fundamental aspect of computer vision. This is a key step for object detection, image recognition and multi-view stereovision applications. OpenMVG allows to describe an image by a collection of regions. Since the region concept in OpenMVG is abstract freely chosen attributes can be embedded in the point description (*e.g.*, such as a point location, scale and orientation) and a binary descriptor of arbitrary length. The current implementation allows to detect and describe:

- Blob regions (Scale invariant points): SIFT [11] (based on VlFeat [20] and Sift Anatomy [12]), AKAZE [22].
- Corner regions: FAST keypoints [25].
- Affine invariant regions: Tree-Based Morse Regions (TBMR) [23], Maximally Stable Extremal Regions (MSER) [24].

4.4 Feature and image collection matching

OpenMVG provides an abstract nearest neighbor search framework that could be used with any vector dimension. The concrete implementations are: (i) BruteForce; (ii) ANN-kD trees [19]; (iii) Cascade hashing [21]. They can be used to compute nearest 3D points or to find corresponding points of a scene by matching features across a series of image pairs.

The image collection matching can be customized by: (i) choosing the appropriate nearest neighbor method; (ii) sending a custom pair list. Thanks to this customization the user can control the accuracy *vs.* time of the retrieval task or easily configure an exhaustive, a sliding window, a loop matching or even a custom matching (*i.e.*, selection of pair by similarity search based on vocabulary tree [33]).

Then the "photo-metric" putatives matches are filtered as geometric coherent matches using an interface to fit robustly multiple-view geometric models.

In order to better understand and visualize the relationship between the images and the computed data (features, matches), OpenMVG exports some SVG data, Fig. 1. Using the SVG format allows to preserve details when zooming thanks to its vectorial nature; it is really useful to see the pairwise matches, since the user can click on a match and see the matching features.

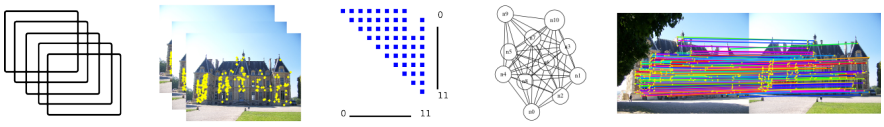

Fig. 1: OpenMVG SVG files exported during the image collection matching task (from Left to Right): image collection, computed features, adjacency matrix, visibility graph [36], pair matches.

4.5 Multiple View Geometry

On top of matching pairs, some multiple view geometric constraints can be checked. This can, for example, be employed to filter the set of matching feature points between images. OpenMVG provides various models and solvers, illustrated Fig.2:

- **Relative pose** from pairs of image-image matching points, such as homography (4-point algorithm [6] for transform of planar scene or scene viewed under pure rotation), fundamental matrix (7/8-point algorithm [6], in case of ignorance of camera internal parameters), essential matrix (5-point [8], in case of known camera internal parameters).
- **Absolute pose** from pairs of 3D-2D matching points by different algorithms, P3P (Perspective from 3 Points) [16], DLT (Direct Linear Transform) [6] (6 pairs), ePnP [15] (n pairs).

– **Similarity transformation** from 3D-3D matching space points, model with 7 degrees of freedom.
– **Triangulation** of 3D point from two view projections through linear method [6], non linear, and L_∞ distance [7].
– **Structure and Motion** with L_∞ norm [7].

(a) Homography (b) Fundamental (c) Relative Pose (d) Triangulation

Fig. 2: Multiple View Geometry Model estimation.

For each model, OpenMVG provides a simple and direct method to compute the resulting pose. For example, estimating the homography between two corresponding point sets `xLeft` and `xRight` can be performed in a few lines of code:

```
// Setup left, right corresponding points and solve for H
openMVG::Mat xLeft(2,4), xRight(2,4);
// Instantiation of homography solver
using H_Solver=openMVG::homography::kernel::FourPointSolver;
// Perform model solving
std::vector<openMVG::Mat3> Hs; //Multi. sol. for some solvers
openMVG::H_Solver::Solve(xLeft, xRight, &Hs);
```

Multiple View Geometry also deals with motion averaging. It consists in computing global motions from relative motions, that is, putting all viewpoints and orientations in a common coordinate system. OpenMVG implements rotation and translation averaging algorithms using various metrics:

- **Rotation averaging** with L_2 norm, non linear L_2 and L_1 [18].
- **Translation averaging** with L_2 norm [17], L_1, and L_∞ [5].

4.6 Robust estimation

Real world data is corrupted by noise and corresponding point pairs may contain outliers. Therefore it is mandatory to use a *robust* model estimation method. OpenMVG proposes various methods to perform robust estimation. Some are based on user-defined thresholds while the others estimate automatically the best model based on a statistical balance between the tight fitting of the data to the model and the number of inlier data. OpenMVG implements these methods:

- **Threshold priors** through MaxConsensus and RANSAC (RANdom SAmple Consensus) [26]

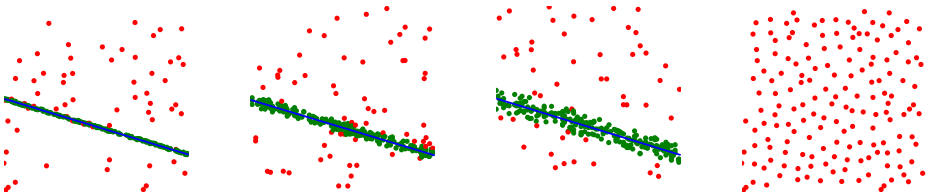

Fig. 3: An *a contrario*-RANSAC unit test example: Automatic threshold adaptivity for line estimation. On the right: no detected model is hallucinated in pure noise data.

- **Threshold free** with Leat Median of Squares (LMedS) and *a contrario*-RANSAC [27,28].

An example of robust line regression to 2D points is illustrated in Fig. 3. The robust estimation framework uses a kernel concept to keep genericity. The kernel is a template object that embeds the model solver and the error metrics (*i.e.*, a measure of the fitting error between the model and the data).

```
//--Robust a contrario fundamental matrix estimation example--
using KernelType=ACKernelAdaptor<
 fundamental::kernel::SevenPointSolver, // Solver
 fundamental::kernel::SymmetricEpipolarDistanceError, // Metrics
 UnnormalizerT, Mat3>;

// Build the Kernel object with corresponding points data
KernelType kernel(
 xLeft, leftImageWidth, leftImageHeight,
 xRight, rightImageWidth, rightImageHeight,
 true); // configure as point to line error model.

// Robust estimation
Mat3 F;
const size_t max_iter = 1024;
std::vector<size_t> vec_inliers;
openMVG::ACRANSAC(kernel, vec_inliers, max_iter, &F);
```

4.7 Camera models

OpenMVG provides an abstract camera interface that can be used seamlessly along the library with the following concrete implementations: pure pinhole [6], pinhole with 1 to 3 radial distortion coefficients [31], pinhole with 5 distortion coefficients (3 radial + 2 tangential) (aka. Brown-Conrady) [29,30], and fish-eye [32]. The abstract camera model allows easy computation of bearing vectors from 2D points, 3D point projection to camera and application or correction of lens distortion.

4.8 Structure from Motion

Using all previous modules, an incremental [3] and a global [5] 3D-reconstruction pipelines are implemented in OpenMVG. The first is more adapted for images with low cross-coverage, but it suffers from drift effects and low scalability due to its sequential nature. The second is fast for datasets with large image overlap and offers a good scalability. The two pipelines have been demonstrated to be very accurate compared to the other existing open solutions [1,13]. Ready to use Python scripts are delivered with the library in order to ease the usage of this tool-chain.

Bundle Adjustment All SfM pipelines rely on a generic bundle adjustment module that allows to perform non linear refinement of the SfM scene by minimizing the structure reprojection in the images (residual error). It consists of a non-linear minimization in a high-dimensional space. This module provides a fine grain control of which parameters (intrinsic (principal point, focal, distortion), extrinsic (rotation, translation), structure landmarks) will be held as constants or variable during the minimization. This fine grain control interface is done using bitwise operator that make the code compact and very expressive. An efficient multi-thread concrete implementation is provided through the Ceres-solver interface [9].

4.9 Localization

This module allows to find the camera pose and orientation of a collection of images in an existing reconstruction. Such a problem is common in virtual/augmented reality setup where one wants to localize the user in a known 3D world in order to display virtual elements at the right place, or when one wants to localize video frames in an existing map/asset for VFX issues (virtual camera system).

4.10 Geodesy

This module provides tools to use known 3D priors to fit the 3D reconstruction to a given user Spatial Reference System (SRS), such as ECEF, for geo-localization. Registration can be performed using Ground Control Points (GCP), and GPS data (pose center position prior) for (i) rigid transformation or (ii) non rigid constraints used in the bundle adjustment framework. Pose priors can also be useful in order to limit the number of pairs to match in a very large image collection in case of UAV/mobile mapping survey.

5 Reproducible research

The project tries to follow the best practice of open source software development. It uses some strict guidelines in order to deliver a high quality code that allows the community to be involved in any work in progress.

5.1 OpenMVG infrastructure

In order to build a project for a community it is necessary to maximize its accessibility and provide tools for feedback about the status of the library. To do so OpenMVG eco-system relies on free tools that allow to perform online version control system, continuous integration and documentation. Here is the list of the different tools used and their purpose:

- **Project management:** `https://github.com/openMVG/openMVG`
  – Github (version control system) for easy access and collaboration, issue tracking, milestones, fork, pull request, code review.
- **Documentation:**
  – reStructuredText for Github integration (visible as a formatted document and not as code), online doc generation & hosting, `http://openmvg.readthedocs.io/en/latest/`.
- **Continuous integration:**
  – Travis-CI for Unix (Linux, OsX).
  – AppVeyor for Windows (Visual Studio).
  – Docker for container based deployment.

5.2 Development principles

Updates rely on the simple rule that they must not break any existing code. Releases are pushed in the *master* branch with tagging; Each time a new release is planned, a new branch *develop* is started. Each *new feature development (X)* happens in a new branch. (i) A Github issue is created with a comprehensive step by step explanation that is required for completion of the feature; (ii) a branch *develop_X* is created from *develop*; (iii) each commit is linked to the Github issue; (iv) once validated, *develop_X* is merged to *develop*.

Github Pull Request (external contribution) are handled by a code review from the community (code style, check the code is easy to use, readable and understandable with comprehensive code comments and paper references), suggestion of an enhanced API or usage of existing functionalities, suggestion of unit test or samples if missing, suggestion to complete the documentation, continuous integration test and non regression, merge once tested and validated by the community to *develop* branch.

Creation of a new release follows these steps: (i) modify *develop* branch API internal version number; (ii) merge from *develop* to *master*; (iii) create a release tag; (iv) edit the Github release tag with a complete CHANGELOG; (v) advertise the new version and features to the community.

Thanks to this set of rules the quality of any modified line of code OpenMVG can be followed by the community and open to comments, tests and critics. People can join effort to develop a feature by using the fork mechanism and contribute actively.

5.3 Future development

The OpenMVG developers hope to continue improving its database of algorithms to follow the state of the art, extend the scope of its users, provide best in class “easy to

read and use" code, hoping to seduce some real time oriented users to add some SLAM algorithms.

Another aim could be to build an open format inspired by the modular `SfM_Data` OpenMVG scene description for 3D photogrammetry purpose, to seamlessly connect projects between existing and upcoming products.

Beside this project, some OpenMVG authors started a new project called "Awesome 3DReconstruction list" that collects the papers (tutorials, conference papers) and open-source resources related to 3D reconstruction from images (more than 120 references are collected).[12].

6 Community adoption

One difference to the other existing framework is that OpenMVG is trying to initiate a real exchange with its community. Some Github statistics give an idea of the community size:

Project name	Year of creation	Contributors	Watchers	Stars	Fork
bundler_sfm	2008	8	108	530	245
COLMAP	2016	5	14	82	34
MVE	2012	13	61	188	131
OpenMVG	2013	**31**	**156**	**802**	**392**
THEIASfM	2015	15	43	165	80

Despite it is hard to compare the statistics due to the differing year of creation of each project, note that OpenMVG has an active community (OpenMVG is neither the oldest nor the most recent project). Moreover, OpenMVG is used by professionals and laboratories for real application, for example:

Arc-Team[13] (a professional company operating in different branches of archaeology, from fieldwork to research, and specialized in the use and development of open source software and hardware for cultural heritage projects) uses OpenMVG softwares for 3D archaeological and architectural documentations in different logistical conditions: ordinary excavations, underwater contexts, remote sensing, underground environments, glacial archaeological researches and abroad missions, see Fig. 4.

Ebrafol[14] provides an independent alternative to judicial expertise and technical assistance in forensic dentistry and forensic anthropology). It uses OpenMVG and its connection to multiple view stereo tools to build 3D models of skull for 3D face reconstruction and to help injured animals by building accurate prostheses, see Fig. 5.

[12] `https://github.com/openMVG/awesome_3DReconstruction_list`
[13] `http://www.arc-team.com/`
[14] Brazilian Team of Forensic Anthropology and Legal Dentistry `http://ebrafol.org/`

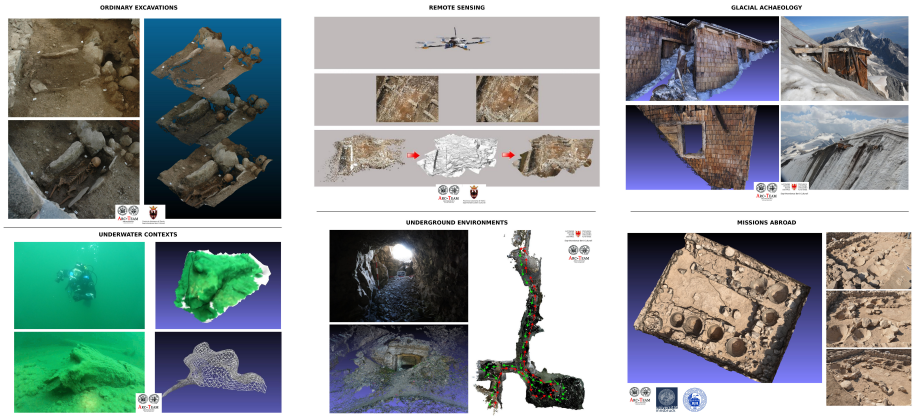

Fig. 4: Arc-Team reconstructions for archaeology.

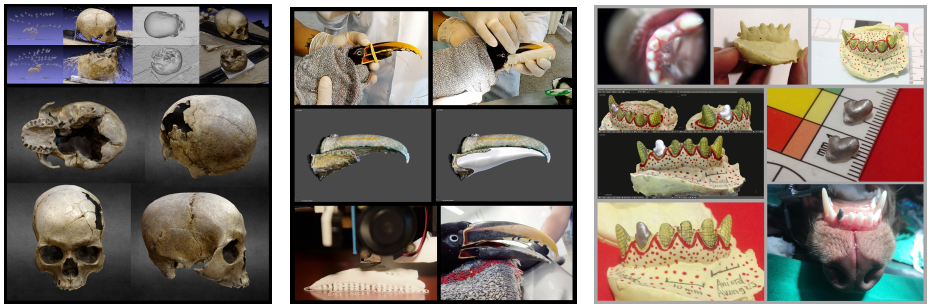

Fig. 5: EBRAFOL sample usage of OpenMVG reconstruction for skull reconstruction and accurate animal prothesis reconstruction and printing.

Digital Humanities Laboratory DHLAB[15] (an EPFL laboratory team that conducts research in historical and geographical information systems. The team is creating a web based service, through the development of a 3D historic GIS server, allowing to view, explore and compare SfM, LIDAR and historical handmade models). It uses OpenMVG to develop a reliable and powerful SfM pipeline in order to compute sparse and dense reconstructions of cities (taking advantage of existing aerial photography database and specific ground-based acquisitions), see Fig. 6.

The community also uses OpenMVG for non-professional work (Fig. 7).

7 Conclusion

We presented OpenMVG, a generic library for multiple view geometry aimed at providing the community with a reference tool. Its insistence on code quality and readability

[15] `http://dhlab.epfl.ch/`