Algorithm for creation of digital twin from UAV

Leonardo Pelonero[1], Fabio Vitello[1]
INAF - Osservatorio Astrofisico di Catania, Via Santa Sofia 78 - 95125 - Catania

The most recent innovations in the field of 3D modeling and virtual reality have led to the development of new tools and technologies to manage and interpret this particularly complex data. Notably, in the field of topographic and environmental surveys, the use of tools for the generation and visualization of 3D models is becoming increasingly popular. The aim is to maintain the same quality guaranteed by traditional and established measurement techniques, but with shorter timeframes. This significantly reduces the cost of a traditional site survey and lightens the workload of field specialists.

Agisoft Metashape, in combination with UAV (Unmanned Aerial Vehicle) systems, emerges as a suitable solution for this case study. Metashape is an advanced and versatile photogrammetry software for creating 3D objects, or "Digital Twins", which can be used in Geographic Information System (GIS) applications. The application allows, starting from raw aerial images (both in controlled and uncontrolled conditions), to process: spherical panoramas, orthophotos, 3D point clouds, digital surface models, and digital elevation models (DEM). One of the positive aspects of using Metashape is that most of the processes are fully automated.

A Digital Twin is a virtual representation of a real object or system, which can be used to monitor, analyze, and predict its real counterpart's behavior. These models, created with Metashape, represent a significant step forward in the field of data analysis in various areas of use: environmental monitoring, urban planning, land management, construction, and infrastructure. They allow us to reflect the planimetry, its current condition, and its future behavior; providing new insights in various fields, from maintenance to the detection and management of possible anomalies caused by floods and weather events.

Image Preprocessing

In order to use Metashape, it is necessary to conduct a surveying mission using a UAV drone. It is essential to capture a series of overlapping images of the area of interest, ensuring that there is sufficient overlap (usually 70-80%) between the images to facilitate their future alignment. Depending on the number of images collected and the quality of the camera used during the surveying, this will impact the data processing times and the quality of the final result.

In order to automate the process and avoid the use of the GUI interaction, Metashape provides a set of APIs that allow interaction with the software. Thus, a series of Python scripts have been implemented to process the entire workflow of Metashape, ranging from the import of raw images to the exportation and saving of all the results obtained.

The instruction to run the script on Linux environments in headless mode from the command line is as follows:

```
./metashape.sh -r <main.py> <image_folder> [output_folder]
```

If the script is run on a system without a graphical interface, it may be necessary to use the additional argument '-platform offscreen':

```
./metashape.sh -r <main.py> <image_folder> [output_folder] -platform offscreen
```

The first step of the project manages the input parameters specified at the invocation script. In particular, it is necessary to specify the folder where the photos are located to be processed and the destination directory to save the obtained results.

```python
# manage input parameters
try:
    if len(sys.argv) < 2:
        print("Usage: general_workflow.py <image_folder> [output_folder]")
        raise Exception("Invalid script arguments")
    image_folder = sys.argv[1]

    # check image_folder exist
    if not os.path.isdir(image_folder):
        raise FileNotFoundError(f"{image_folder} path does not exist")

    # only image_folder
    if len(sys.argv) == 2:
        # Save on /storage
        username = getpass.getuser()
        output_folder_name = os.path.basename(image_folder) + '_' +
datetime.datetime.now().strftime("%d%m_%H%M")
        output_folder = os.path.join('/storage/Metashape_Hammon/Modelli', output_folder_name)
        os.makedirs(output_folder, exist_ok=True)

    # image e output
    if len(sys.argv) == 3:
        output_folder = sys.argv[2]
        # if output_folder do not exist, create it
        if not os.path.exists(output_folder):
            os.makedirs(output_folder)
```

```
    if len(sys.argv) >= 4:
        raise Exception("Too much input arguments")

except Exception as e:
    print(f"Error: {str(e)}")
```

The script, so implemented, handles exceptions for non-existent specified paths, and saves the results in a folder with the name of the project, also specifying the date and time when the process is started.

To have feedback on the progress timing and to keep track of which different process is underway, the ProgressPrinter class provides a series of functions that, when invoked periodically during the execution of long or intensive operations, show an estimate of the remaining time. Constant control of the process's progress and adjustments is made possible by this.

```
class ProgressPrinter:
        def __init__( self, name ):
                self.name = name  # process name
        def __call__( self, percent ):
                print("{} progress: {:.2f}%".format(self.name, percent), end="\r", flush=True)
```

The procedure, from the alignment of the photographic material to the generation of the 3D model, is included in four main phases that are explained below. These steps handle most of the data processing needs, with most operations being executed automatically based on the user's parameters.

Photo Alignment

To ensure proper processing, the script only filters the type of photos in the format of our interest, avoiding any possible extra data generated by the photo acquisition phase.

The presence of poor quality, grainy or blurred photos can negatively affect the results of camera alignment as well as the processing of the final texture. For this reason, all photos are filtered based on their image quality. Images with a quality lower than 0.5 are excluded from the process. The image quality value is calculated based on the sharpness level of the image.

```
def find_files(folder, types):
    return [entry.path for entry in os.scandir(folder) if (entry.is_file() and
os.path.splitext(entry.name)[1].lower() in types)]

# check presence of image in image_folder
photos = find_files(image_folder, [".jpg", ".jpeg", ".tif", ".tiff"])

doc = Metashape.Document()
doc.save(output_folder + '/project.psx')
chunk = doc.addChunk()
chunk.addPhotos(photos)
doc.save()
```

```
# estimate image quality
for camera in chunk.cameras:
    chunk.analyzePhotos(camera)
    if float(camera.photo.meta['Image/Quality']) < 0.5:
         camera.enabled = False
doc.save()
print(str(len(chunk.cameras)) + " images loaded")
```

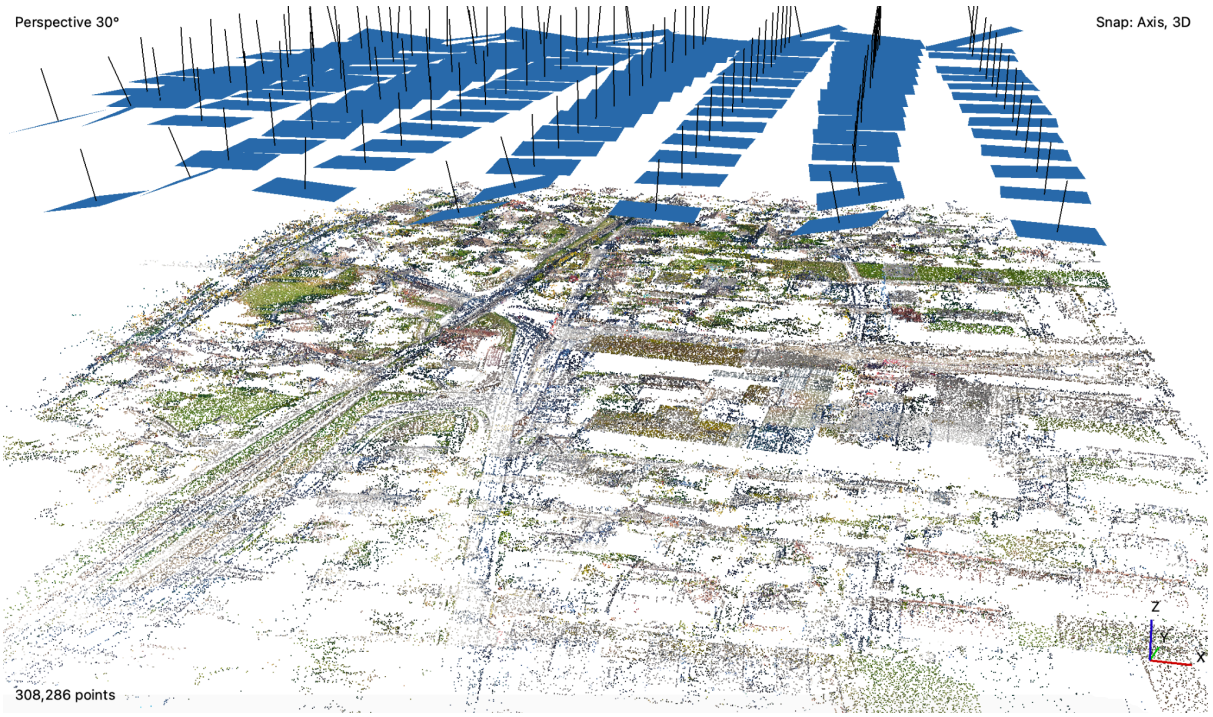
Once the project is created and the photos are imported in Metashape, they will need to be aligned. In this phase, the position of the camera and the orientation of each photo are determined, and the model is built with a sparse points cloud. The position and orientation of the camera are calibrated by internal and external orientation parameters, which are based on the focal length of the camera, coordinates and position of the main point of the frame, and lens distortion coefficients.

```
progress_printer = ProgressPrinter("matchPhotos")
chunk.matchPhotos(downscale = 1,
                  keypoint_limit = 40000,
                  tiepoint_limit = 10000,
                  # generic_preselection = True,
                  reference_preselection = True,
                  progress=progress_printer) # Progress callback
doc.save()

chunk.alignCameras()
doc.save()
```

We can mention some of the settings parameters for image alignment:

- downscale: set the accuracy level. The lower the level, the more precise the estimate (0 - Highest, 1 - High, 2 - Medium, 4 - Low, 8 - Lowest) the more time-consuming it is.
- generic preselection: favors the alignment process when the set of photos is very high;
- reference preselection: it adapts better in the case where only a few reference points are detected, for example, during surveys of wooded areas or cultivated fields;
- keypoint limit: upper limit of feature points on every image;
- tie point limit: upper limit of matching points for every image. It is recommended to set the value as 10,000. A limit that is too high or low could compromise areas in the generation of the point cloud.

As a result, a sparse point cloud and a set of camera positions are generated. The sparse point cloud represents the alignment results of the photos and will not be used directly. On the contrary, the set of camera positions is necessary for subsequent processing of the 3D model in Metashape.



Dense cloud points

Metashape gives the possibility to create and view the model in the form of a dense point cloud. It is generated using: the initial sparse point cloud, accurate camera alignments and point interpolation. This process fills the spaces between the sparse points, resulting in a high-resolution, detailed, and complete 3D representation of the captured scene.

The dense cloud can be produced by specifying the source data: depth maps or tiled model. By specifying this, it will then be possible to specify the quality of the point cloud; higher settings always imply a longer processing time (1 - Ultra high, 2 - High, 4 - Medium, 8 - Low, 16 - Lowest). In this case, the depth maps are calculated on each image. Enabling reuse depth in the chunk allows it to be utilized in the creation of the point cloud.

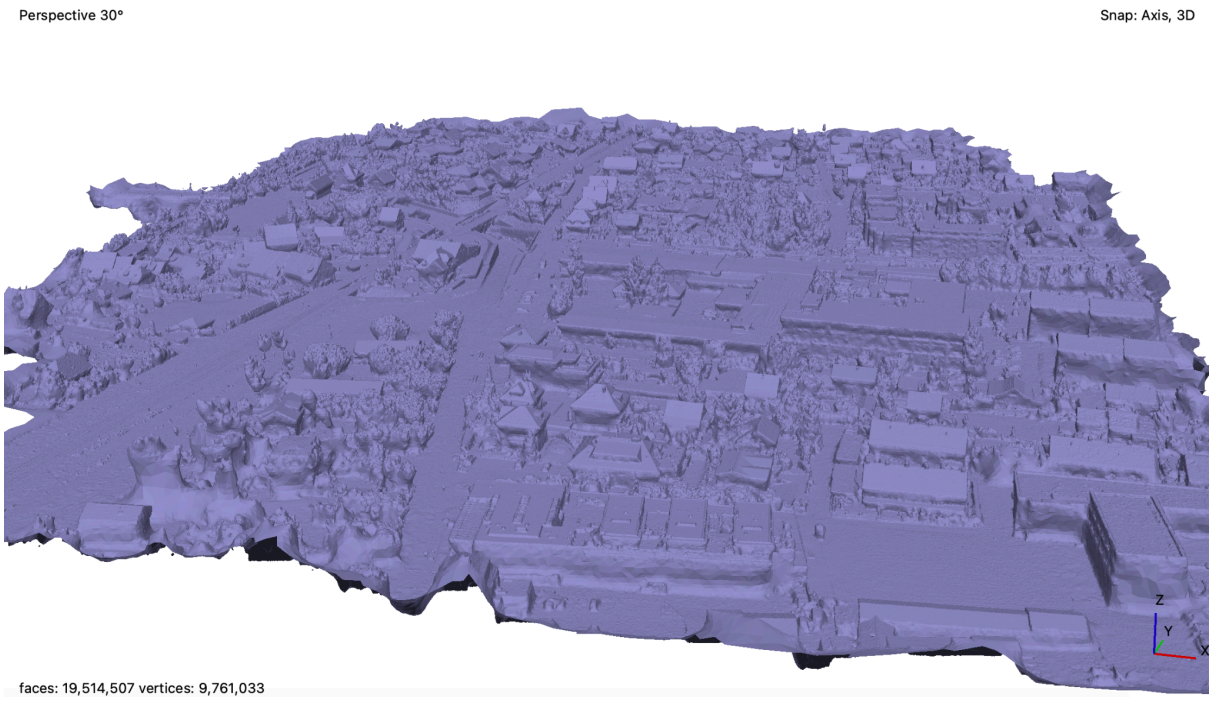
```
progress_printer = ProgressPrinter("buildDeptMaps")
chunk.buildDepthMaps(
                downscale = 2,
                filter_mode = Metashape.MildFiltering,
                reuse_depth = True,
                progress = progress_printer)
doc.save()

progress_printer = ProgressPrinter("buildPointCloud")
chunk.buildPointCloud(
                point_colors = True,
                point_confidence = True,
                keep_depth = True,
                progress = progress_printer)
doc.save()
```

For the creation of the point cloud, the most important parameters are point_colors, point_confidence, and keep_depth. The point_colors parameter includes the points color information. It can be set to False to reduce the process time. The point_confidence parameter filters the dense cloud point, which can help remove any remaining outliers. The keep_depth parameter maintains the depth maps, useful for subsequent calculation of DEM.

Build 3D Model (mesh)

This phase involves the creation of a 3D model. The depth maps created prior to this step are designated as source data, which guarantees higher quality results compared to models with sparse cloud source data. If the area of interest to be processed is very large, it is possible to subtask the creation of the 3D model into blocks, specifying the size of the blocks in meters and the reference coordinate system (blocks_size, blocks_crs).

With enabled interpolation mode, the software generates a geometric model without empty holes based on refined level interpolations. The problem is that this process risks generating large additional geometric areas. By disabling the interpolation, it leads to accurate reconstruction results since only areas corresponding to point cloud points are reconstructed.



The specified surface type (HeightField) is optimized for modeling on planar surfaces such as terrains or basereliefs. This parameter is excellent for processing aerial photographs as it requires less memory and allows the processing of larger data sets. This setting can be changed to Arbitrary as needed to allow the processing of closed objects like buildings, at the cost of higher memory consumption.

```
progress_printer = ProgressPrinter("buildModel")
chunk.buildModel(source_data = Metashape.DepthMapsData,
                 surface_type = Metashape.HeightField,
                 face_count = Metashape.HighFaceCount,
                 interpolation = Metashape.DisabledInterpolation,
                 # split_in_blocks = True,
                 vertex_color = True,
                 progress = progress_printer)
doc.save()

progress_printer = ProgressPrinter("buildUV")
chunk.buildUV(page_count = 2,
              texture_size = 4096,
              progress = progress_printer)

progress_printer = ProgressPrinter("buildTexture")
chunk.buildTexture(blending_mode = Metashape.MosaicBlending,
                   texture_size = 4096,
                   fill_holes = True,
                   ghosting_filter = True,
                   progress = progress_printer)
doc.save()
```

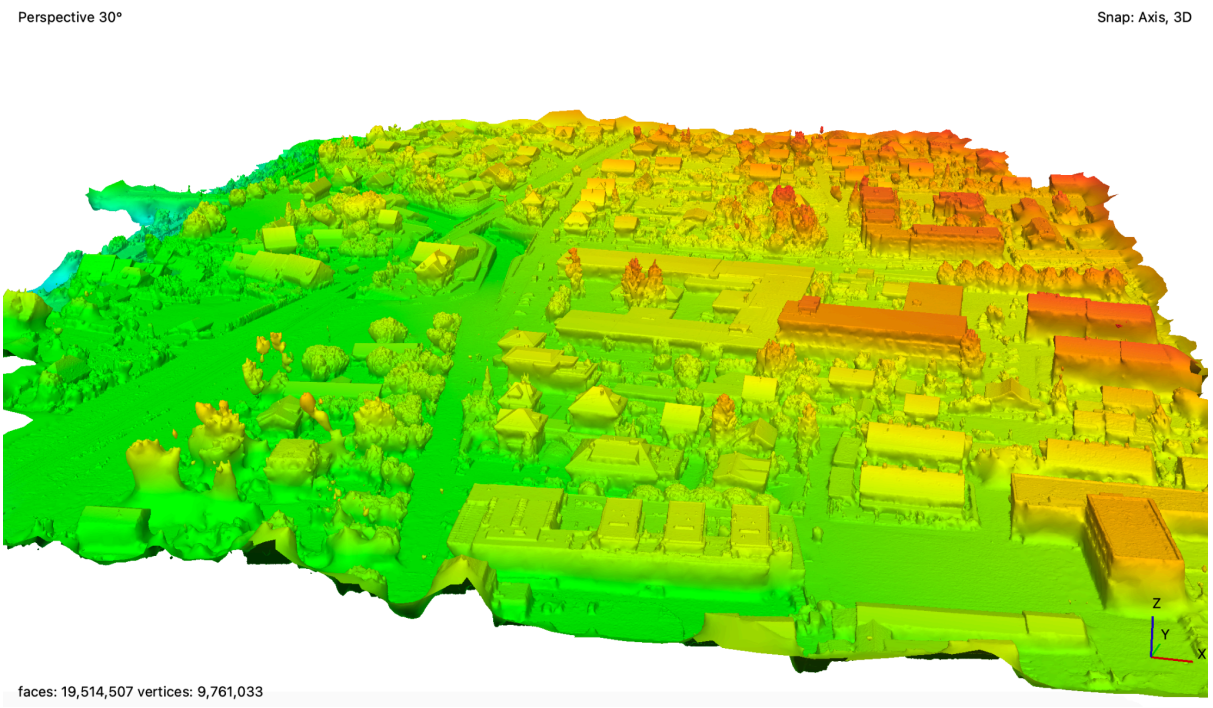
Once the 3D model is created, it will be possible to cover it by applying the texture. In the texture building process, it is possible to specify the size of the pixel structure.

Building DEM and Orthomosaic

In conclusion, the reconstructed geometry can be structured and used for the generation of the Orthomosaic model and the Digital Elevation Model (DEM).

DEM represents a set of measurements that record the elevation of the earth's surface and also contain information about the spatial relationships between these measurements. In Metashape, the DEM can be rasterized from the point cloud, depth maps, or the 3D model. The most accurate results are obtained by setting the dense point cloud as source data.

It is possible to generate both the Digital Terrain Model (DTM), which represents the earth's surface without objects on top like buildings or plants, and the Digital Surface Model (DSM), which is the surface with all elements on top.




```
progress_printer = ProgressPrinter("buildDem")
chunk.buildDem(source_data = Metashape.PointCloudData,
               interpolation = Metashape.EnabledInterpolation,
               progress = progress_printer)
doc.save()

progress_printer = ProgressPrinter("buildOrthomosaic")
chunk.buildOrthomosaic(surface_data = Metashape.ElevationData, progress = progress_printer)
doc.save()
```

In aerial photographic survey data processing, exporting high quality orthomosaic models is a common task. The process consists of combining the images (orthophotos) projected onto the surface based on the preferred x, y, or z axis.

Saving and exporting the results

The different processing stages for the reconstruction of a 3D model require a lot of time. The Metashape software allows you to save the current project in the *.psx format, which saves the references with the results of each carried out process.

Each step in the implemented code indeed corresponds to a project save. The export of the individual results obtained in their formats is also shown below.

```
# export results
chunk.exportReport(output_folder + '/report.pdf')

if chunk.model:
    print("--Exporting Model")
    chunk.exportModel(output_folder + '/model.obj')
```

```
if chunk.point_cloud:
    print("--Exporting Point Cloud")
    chunk.exportPointCloud(output_folder + '/point_cloud.las', source_data =
Metashape.PointCloudData)

if chunk.elevation:
    print("--Exporting DEM")
    chunk.exportRaster(output_folder + '/dem.tif', source_data = Metashape.ElevationData)

if chunk.orthomosaic:
    print("--Exporting Orthomosaic")
    chunk.exportRaster(output_folder + '/orthomosaic.tif', source_data = Metashape.OrthomosaicData,
split_in_blocks=True)
```

Credits and References

The demo image processing and 3D model creation were performed with the software Agisoft Metashape Professional (https://www.agisoft.com).

The photos used for the project were sourced from Agisoft Metashape sample data and available Wingtra dataset (https://wingtra.com).

Regarding the script mentioned, the source code can be found in the respective repository (https://github.com/VisIVOLab/UAV-digital-twin).