II. FROM APERO ORIENTATIONS TO OTHER SOFTWARE

I MATHEMATICAL FORMULATION OF THE PROBLEM

a) Beginning of the second part

In the second part of my internship a new function called Apero2NVM was implemented. This function takes the output of MicMac, both in terms of tie points and orientations, and writes them into a .nvm (N-View Matches) file. This file is accepted as input by four of the five selected free and full-reconstruction software, i.e. VisualSfM, MVE, SURE, MeshRecon. CMPMVS requires a different input. However a result from the Apero2PMVS function (already implemented in the Micmac suite) can be used for this purpose. RunSFM and OSM-Bundler cannot take this file .nvm as input of the dense matching step, because they run the entire process in one single step. But the core of these solutions is close to the one implemented in VisualSfM.

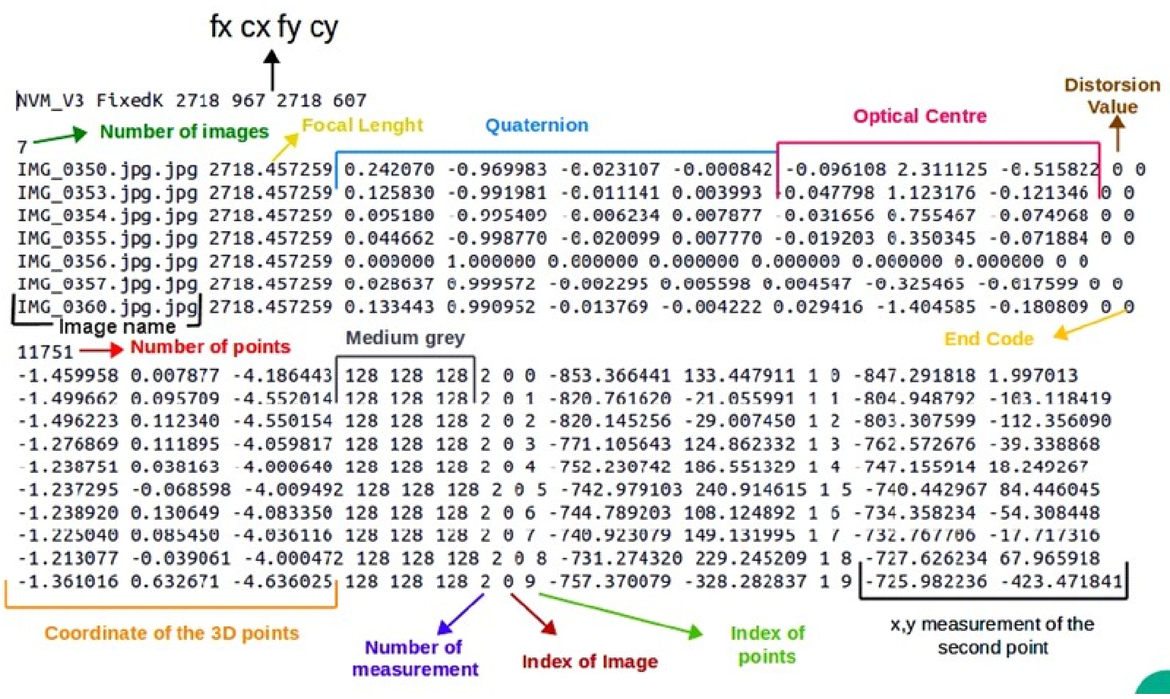
b) Content of the NVM File

The .nvm file consists of two main groups of data.

- The first group contains a list of information about image locations and corresponding camera orientations. For each image, the following data are stored: <File name>, <focal length>, <quaternion WXYZ>, <camera center>, <radial distortion>.

- The second group contains a list of information about tie points and their projection to the object space (sparse point cloud). For each 3D point, the following data are stored: <XYZ> (i.e. 3D coordinates in object space), <RGB>,<number of measurements> (i.e. number of images where that point was detected), <Image index>,<Point Index>, <xy> (i.e. 2D coordinates in image space).

Additionally, as showed in figure 14, the number of images and the number of 3D points are written at the top of their corresponding groups of data.



```
NVM_V3 FixedK 2718 967 2718 607
7
IMG_0350.jpg.jpg 2718.457259 0.242070 -0.969983 -0.023107 -0.000842 -0.096108 2.311125 -0.515822 0 0
IMG_0353.jpg.jpg 2718.457259 0.125830 -0.991981 -0.011141 0.003993 -0.047798 1.123176 -0.121346 0 0
IMG_0354.jpg.jpg 2718.457259 0.095180 -0.995409 -0.006234 0.007877 -0.031656 0.755467 -0.074968 0 0
IMG_0355.jpg.jpg 2718.457259 0.044662 -0.998770 -0.020099 0.007770 -0.019203 0.350345 -0.071884 0 0
IMG_0356.jpg.jpg 2718.457259 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0 0
IMG_0357.jpg.jpg 2718.457259 0.028637 0.999572 -0.002295 0.005598 0.004547 -0.325465 -0.017599 0 0
IMG_0360.jpg.jpg 2718.457259 0.133443 0.990952 -0.013769 -0.004222 0.029416 -1.404585 -0.180809 0 0
11751
-1.459958 0.007877 -4.186443 128 128 128 2 0 0 -853.366441 133.447911 1 0 -847.291818 1.997013
-1.499662 0.095709 -4.552014 128 128 128 2 0 1 -820.761620 -21.055991 1 1 -804.948792 -103.118419
-1.496223 0.112340 -4.550154 128 128 128 2 0 2 -820.145256 -29.007450 1 2 -803.307599 -112.356090
-1.276869 0.111895 -4.059817 128 128 128 2 0 3 -771.105643 124.862332 1 3 -762.572676 -39.338868
-1.238751 0.038163 -4.000640 128 128 128 2 0 4 -752.230742 186.551329 1 4 -747.155914 18.249267
-1.237295 -0.068598 -4.009492 128 128 128 2 0 5 -742.979103 240.914615 1 5 -740.442967 84.446045
-1.238920 0.130649 -4.083350 128 128 128 2 0 6 -744.789203 108.124892 1 6 -734.358234 -54.308448
-1.225040 0.085450 -4.036116 128 128 128 2 0 7 -740.923079 149.131995 1 7 -732.767706 -17.717316
-1.213077 -0.039061 -4.000472 128 128 128 2 0 8 -731.274320 229.245209 1 8 -727.626234 67.965918
-1.361016 0.632671 -4.636025 128 128 128 2 0 9 -757.370079 -328.282837 1 9 -725.982236 -423.471841
```

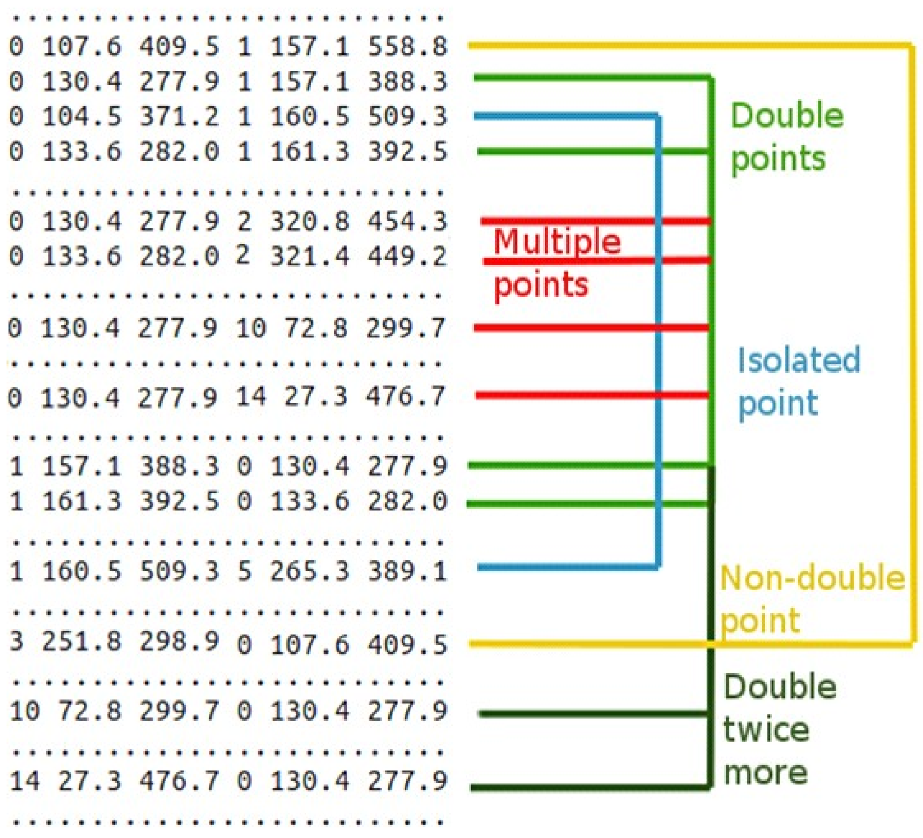
Figure 14 : Extract of the top of an .nvm file

C) <u>Analysis of the list of tie-points extracted by micmac</u>

Micmac extracts the tie points among the images by running the command-line tool Tapioca. As output, a folder is created for every image that includes information about the 2D measurements of points extracted in that image. In particular, inside each folder, a file is stored for each overlapping image. Each line of this file includes: the XY coordinates of the tie point in the first image and the XY coordinates of the tie point in the second image. For example, the file *IMG_02.txt* of the folder *Pastis_IMG_01* contains the coordinates of tie points extracted between the images IMG_01 and IMG_02.

In my function, Apero2NVM I first merged all this measurements inside one single txt file. Additionally, I have added an index for each image. So index 0 means that the measurement comes from the first image, index 5 means it comes from the 6th image, index n comes from the (n+1)th image.

Next, the measurements were grouped into several classes. **Double points** are those points whose coordinates are repeated in the merged list. This happens because every measurement is often written twice: for example, in the file *IMG_02.txt* of the folder *Pastis_IMG_01* we will find the same information stored in the file *IMG_02.txt* of the folder *Pastis_IMG_02*. When a point is detected in more than two images, it is called a **multiple point**. When a measurement (in first position) is found at another line (in second position), and the first measurement of this second line is different, it is called a **non-double point**. At the contrary, when the second measurement of one line is found in first position of another line, but the second measurement associated to it comes from another image, it is called **isolated point**. Finally when a couple of measurements appear only once, it is called a **for-gotten point**. Figure 15 shows an example of the different classes of points.



```
..........................
0 107.6 409.5 1 157.1 558.8
0 130.4 277.9 1 157.1 388.3
0 104.5 371.2 1 160.5 509.3
0 133.6 282.0 1 161.3 392.5
..........................
0 130.4 277.9 2 320.8 454.3
0 133.6 282.0 2 321.4 449.2
..........................
0 130.4 277.9 10 72.8 299.7
..........................
0 130.4 277.9 14 27.3 476.7
..........................
1 157.1 388.3 0 130.4 277.9
1 161.3 392.5 0 133.6 282.0
..........................
1 160.5 509.3 5 265.3 389.1
..........................
3 251.8 298.9 0 107.6 409.5
..........................
10 72.8 299.7 0 130.4 277.9
..........................
14 27.3 476.7 0 130.4 277.9
..........................
```

Figure 15 : Extract and schema of the merge list of tie points

Here in the original report follow a mathematic analysis of the system of projection and the distortion, but it’s not useful here, the images are undistorted, and the 3D systems of reconstruction are alike…

II THE APERO2NVM FUNCTION

Here in the original report follows an introduction of Micmac organization and about the creation of a new function inside Micmac (TestLib), it was also cut for simplification.

a) Arguments of the function.

```
mm3d Apero2NVM -help
*****************************
* Help for Elise Arg main *
*****************************
Mandatory unnamed args :
* string :: {Images Pattern}
* string :: {Orientation name}
Named args :
* [Name=Nom] string :: {NVM file name}
* [Name=Out] string :: {Output folder (end with /)}
* [Name=ExpTxt] bool :: {Point in txt format ? (Def=false)}
* [Name=ExpApeCloud] bool :: {Exporte Ply? (Def=false)}
* [Name=ExpTiePt] bool :: {Export list of Tie Points uncorrected of the distortion ?(Def=false)}
* [Name=KpImCen] bool :: {Dont add a little rotation for pass from Image Centre to PP ?(To be right fix
LibPP=0 in tapas before)(Def=false)}
```

Basically the function Apero2NVM works with two Mandatory arguments that are *Images Pattern* and the *Orientation folder*. Arguments corresponding to the *name of the file* and the *output folder* are automatically set, but can be modified by the user. Four other extra possibilities can be chosen. The user can export the sparse point cloud with *ExpApeCloud*, as well as the final list of original tie-points with *ExpTiePt*. Furthermore, it is possible to work with tie points extracted in txt format with the *ExpTxt* option. Finally, KpImCen allows the user to keep a rotation matrix of orientation not corrected, but still have the Image Centre as a Principal Point. In this case it will not give the same geometrical result as Micmac would do with the same orientation. Three additional files are also exported by the command, i.e.a 3DPtsList.txt, DroitesList.txt and TiePtsUndistortedList.txt. The first file is the list of the tie-point coordinate x,y,z. The second file includes a list of Image center and direction-vector for each measurement, so typically 6 numbers for one measurement. The last file is a list of tie points measurements corrected of the distortion. In these files and all the data are on the same line for each 3D point. Each measurement is defined by an image index and a couple (x, y) of image coordinates.

Apero2NVM is divided in sub-functions that execute a precise task, and after they are called in the Apero2NVM_main part of the file, located at the end, which is exactly the same than a main function in C++ programming.

b) Division in sub-function

I will develop here, one by one the function I have written, and describe the executed algorithm. In the chronological order the functions principles are:

FindMatchFileAndIndex
CopyAndMergeMatchFile
CorrectTiePoint
GlobalCorrectionTiePoint
Triangulation
UndistortIm
TransfoOri_andWriteFile
And the main:
Apero2NVM_main

At the beginning of the execution we have a folder which contains the images, and an Ori folder and a Homol folder, which respectively contains orientations and lists of tie-points. The Homol folder includes other folders called PastisImageName, which contains list of measurements for each image and the other images which have an overlap with this image. So logically the number of these PastisImageName folders is equal to the number of images. Below a description of each sub-function is detailed.

FindMatchFileAndIndex

vector<string> FindMatchFileAndIndex(string aNameDir, string aPattern, int ExpTxt, vector<vector<string> > &aVectImSift, vector<vector<int> > &aMatrixIndex)

The aim of this function is to find:
- The PastisImageName folders;
- The list of tie point measurements associated to them. First, it finds the name of each file and stored it in the address of aVectImSift
- The index associated to every file name. They are stored at the address of aMatrixIndex that has the same size than aVectImSift . The index starts with 0 for the first image and + 1 is added for every other image until the end of the pattern list.

Summary of the algorithm:
- Find the name of the images in aPattern;

- Add Homol/Pastis before this name;
- Go in this folder directory;
- For each directory, find the file with the list of measurement;
- Compare each file to the image name and give the associated index;
- Put Index and name in matrix;
- And return every directory in a vector.

CopyAndMergeMatchFile

vector<vector<double> > CopyAndMergeMatchFile(string aNameDir, string aPattern, string DirOut, int ExpTxt)

The aim of this function is:
To read the measurement files, aggregate them, and copy them in a matrix. Initially options for create and written a single file was done but put in comment.

Summary of the algorithm:
- Create the folder of output;
- Get the vector of directory and the matrix, calling the previous function;
- Indicate that the addresses of the two matrixes are the current declared matrix;
- Go in each directory, for each of them read the entire file;
- For each file push back the value preceded by the index, at the following place in a vector;
- Push back every vector of each directory in a single matrix;
- Close all;
- Finally return the new matrix.

CorrectTiePoint

vector<vector<double> > CorrectTiePoint(string aNameDir, string aPattern, string DirOut, int ExpTxt)

The aim of this function is:
To correct the list of aggregated tie-point measurements with respect to the presence of:
- double points
- multiple points
- non-double points
Every time a list was made for verification, but for lightness it was put in comment.

Summary of the algorithm:

- Get the matrix of measurement by calling the previous function;

For the double points:

- For every image, except the last one, take the measurements of every point, i.e. its (x ,y) coordinates;
- Go in all the next images;
- Take the measurements of every point;
- See if it corresponds to the same point and in this case push back 2 vectors of index measurement the index is equal to n° line -1.
- Once done, course the matrix in inverse order and erase when the id is written in the first vector of index;

For the multiple points:

- For every image, take the first measurement of every point;
- Go on the following points on the same image;
- Take also their first measurement;
- See if the first measurement corresponds to the same point, in this case push back 2 vectors of index measurement and go to the next point in the matrix;
- Once done, course the matrix and erase when the id is written in the first vector of index, but before add the second measurement of the first point at the line corresponding at the id of the second vector of index, so at the second point.

For the non-double points:

- For every image, except the last one, take the first measurement of every point;
- Go in all next images;
- Take the second measurement of every point;
- See if the first measurement of the first point corresponds to the second measurement of the second point, in this case it's the same point, push back the first measurement of the second point at the end of the first point, push back 2 vectors of index measurement;
- Once done, course the matrix in inverse order and erase when the id is written in the second vector of index;
- Eventually return the modified matrix.

GlobalCorrectionTiePoint

vector<vector<double> > GlobalCorrectionTiePoint(string aNameDir, string aPattern, string DirOut, string aOri, int ExpTxt, bool ExpTieP)

The aim of this function is:

To delete the redundancy in the entire matrix of points, and write all the measurements associated to each point on a single line.

Summary of the algorithm:
- For every image, except the last one, take the measurements of every point;
- Go in all next images;
- Take the measurements of every point;
- See for every second measurement if it is equal to the first measurement. In this case it's the same point; push back 2 vectors of index measurement;
- Then course the two vector of index measurement;
-Take in the Matrix of measurements all the image indices of the measurement corresponding to the id of the first vector;
-See if they are the same than the image index of the measurement corresponding to the id of the second vector;
- If not, add the corresponding measurement at the end of the measurement of the second point;
- Once all the indices of the first point are checked in this way, delete these points;
- If the option ExpTiePt is true, export the final matrix which is not corrected from the distortion;
- Eventually, return this matrix of final measurement. All the redundancy has been removed.

Triangulation

void Triangulation(string aNameDir, string aPattern, string aOri, string DirOut, int ExpTxt, bool ExpTieP, vector<vector<double> > &aFullTabTiePoint)

The aim of this function is:
To project into the 3D space the rays coming from the tie-point measurements and to find a 3D point. The List of Tie-point will also be corrected of the distortion.

Summary of the algorithm:
- Get the orientation of all cameras;
- Get the center of all cameras;

For the 3D projection
- For every point get the good camera for each measurement, via the associated image index;

-Create a direction-vector for each tie point, from the center of the selected camera to the tie point;
- Correct the distortion for each tie-point;
- Check now if the undistorted tie-point measurements are out of the box of the images. If still more than two measurements remain inside, write the point in the address of new matrix and do the same for the center associated with the direction-vector;
- Clear the old Matrix;
For the 3D intersection
- For every point, take every possible couple of camera centers and direction-vectors;
- Call the function *LeastSquareSolv* for every couple and get the 3D point;
- For each group of 3D solution associated to the same tie point, keep the barycenter;
- Write the list of matrix 3DPts, Droites (center + vector) and TiePtsCorrected in a new txt file.

LeastSquareSolv is a secondary function that takes as input two camera centers and their associated two vectors director and gives as output the solution that minimizes the residuals with an adjustment in a least square sense.

The intersection of the straight lines give a point how can be write as: 3DPt = centre + coeff*direction-vector. So here we have two coefficients and a 2x2 matrix are easy to invert and transpose.

For the final solution, since we take the barycenter of each group of 3D solutions coming from every possible couple, we give the same weight to all measurements. Moreover the Sigma0² of my tests was between 10^{-6} and 10^{-8} after a single iteration; and for a distance between 10^{0} and 10^{1} measurement units. So, considering that I do the mean of all these solutions, I will obtain approximately 10^{-4} of precision for a 3D projection and intersection. Thus for rays which pass of around 1000 pixels of the image centre, a simple Tales equation gives a precision of feature positioning of 1/10 pixel.

UndistortIm

void UndistortIM(string aNameDir, string aPattern, string aOri, string DirOut, bool KeepImC)

The aim of this function is:
To undistort all the images and to save them in a .jpg format in the output folder (if only the option KeepImc is selected)

Summary of the algorithm:
-Verify if keepImC is selected, if not the undistortion will be done in the latter in next function;

- Find the name of every image;
- Compose the DRUNK command for undistortion;
- Call DRUNK, that is a function embedded in the MicMac suite for removing image distortion, knowing internal orientations;
- Convert these images from .tif to .jpg.

TransfoOri_andWriteFile

void TransfORI_andWFile(string aNameDir, string aPattern, string aOri, string DirOut, string aNAME, vector<vector<double> > aFullTabTiePoint, bool ExpCloud, bool KeepImC)

The aim of this function is:
To take the orientation of every image, convert these orientations into the expected format, take the image measurement and 3D points, and write them all in the .nvm file.

Summary of the algorithm:
- Read the orientation of each camera;
- Get the Focal Length;
- Get the Principal Point;
- Get the Image Size;
- Make a vector of each previous variable
- Call the function *CorrectRotation* that writes a rotation matrix for handle the difference between the PP and the image center;
- Call the function *CorrectDecentre* and the function of undistortion and compute the dataset of image corrected from these two aspects.
- Call the function *MakeQuaternion* and push back the solution in a vector;
- Find the center of each camera in the final system of reconstruction;
- If ExpApeCloud is true begin to write the point Cloud;
- Write the header of the .nvm file;
- For each image write the orientation;
- For each point, write its corrected coordinates, add medium grey for color, add the number of measurements, and for each measurement, specify the id of image and the id of measurement;
- If ExpApeCloud is true write 3D point and grey color in the point Cloud;
- Finally add footer to the end of the .nvm file and close it;

MakeQuaternion is a secondary function that takes in input the camera orientations. It will use the matrix of rotation of the camera and return a vector of four coefficients.

The normalized sum of this coefficient must be equal to 1. They are calculated from the matrix of rotation in a way that:

If R = [Rot(1,1)+Rot(1,2)+Rot(1,3); Rot(2,1)+Rot(2,2)+Rot(2,3); Rot(3,1)+Rot(3,2)+Rot(3,3)]

The quaternion Q = a + b i + c j + d k

i,j,k are the vector of the orthogonal base, with :

$a=0.5\ (\ 1+Rot(1,1)+Rot(2,2)+Rot(3,3)\ )^{0.5}$

$b=0.25\ (Rot(3,2)-Rot(2,3))/a$

$c=0.25\ (Rot(1,3)-Rot(3,1))/a$

$d=0.25\ (Rot(2,1)-Rot(1,2))/a$

Rounding problems can also happen when the “a” value is close to 0 or other value near 1 or 0, so I also write a correction for it.

CorrectRotation : this function writes a rotation matrix based on the angle between the optical center and the difference between the principal point and the image center. If the options KeepImC is true, the rotation matrix is the identity matrix. By default this option is false.

CorrectDecentre : this function make a new image using a plan to plan projection. The goal of this function is to pass from the plan of the photograph which has a normal vector situated from the optical center to the Principal Point, to a plan which has a normal vector from the same optical center, but to the image center.

More concretely, if the coefficients of the plan of the undistorted image are a=b=0, c= -d=f the focal length, the coefficients of the centered image are $a'= \Delta ppx$, $b'= \Delta ppy$, $c'=f$, $d'=-f^2- \Delta ppx^2- \Delta ppy^2$, the ratio of the distance from the optical center to the two plans is simply: $K=-d'/(a'x+b'y+c'f)$, x and y are the coordinate on the picture subtracted from the center. Eventually the plan is rotated and translated in order to register the image properly.

Apero2NVM_main

*int Apero2NVM_main(int argc,char ** argv)*

The main function is like a C++ main function, i.e. it calls all the functions written inside itself after compilation when we launch the execution of the program. This function works with a ElInitArgMain who displays help if we ask for, otherwise it keeps the argument given by users and associates them to the variables used by the other sub-function.

III TESTS AND FEEDBACK

a) Sparse cloud with AperiCloud and Dense MicMac Cloud

Before writing a function for 3D projection and intersection, I tried to use the AperiCloud function embedded into Micmac. I tested different options and tried to understand how it works. Actually, the AperiCloud computes one single 3D point for each couple of image measurements and if some other measurements give similar 3D solutions (i.e. they are inside a sphere of known ray), it merges them into a single point. But still lots of identical features give different points that will not be merged in the final AperiClould. Thus the AperiCloud point cloud features “grapes” of points that actually correspond to the same point. Furthermore, the correspondence between image measurements and 3D projected points is not easily available.

This is why I wrote the function called “*Triangulation”* for projecting the filtered tie points into a 3D sparse point cloud. A medium grey color (R=128; G=128; B=128) was selected for the color attribute: actually, the sparse cloud in the .nvm file is used only for initialization and color is not important. So, the sparse cloud produced by running the “*Triangulation”* function is spatially close to the AperiCloud output. But for every 3D point we know exactly the image measurements it comes from. The number of points is approximately 3 times smaller than the AperiCloud cloud; this is because all the redundancy were removed before projecting the tie points in the 3D space.

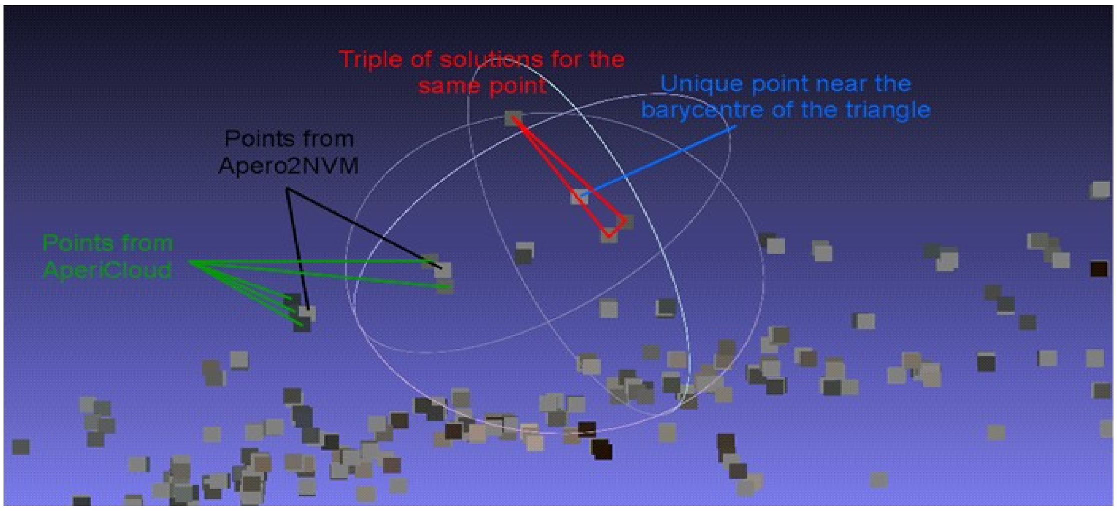
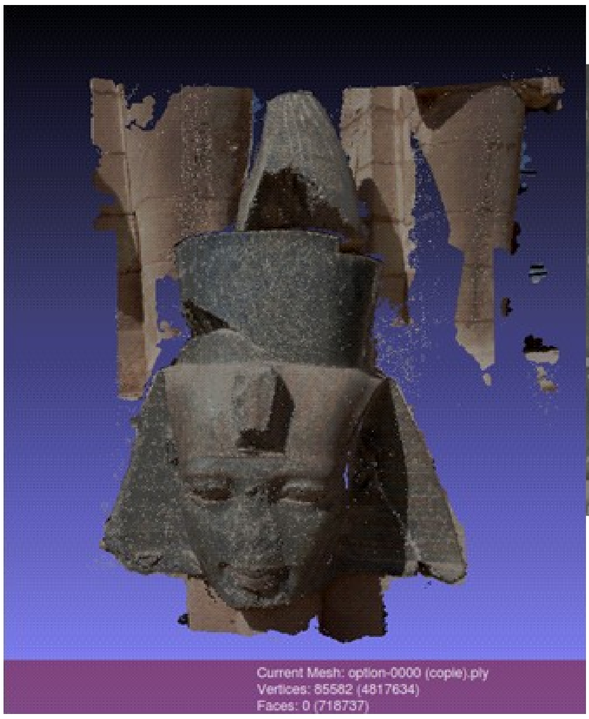

Triple of solutions for the same point
Unique point near the barycentre of the triangle
Points from Apero2NVM
Points from AperiCloud

Figure 19 : Comparison of the two sparse clouds

No filtering has been applied, but the number of outlier obtained is very low. Comparing the sparse cloud produced against the MicMac final dense cloud (compute at half the original resolution) it appears that the sparse cloud falls geometrically alike the final dense points cloud as shows the figure 20, just below visualized on MeshLab software.



Current Mesh: option-0000 (copie).ply
Vertices: 85582 (4817634)
Faces: 0 (718737)

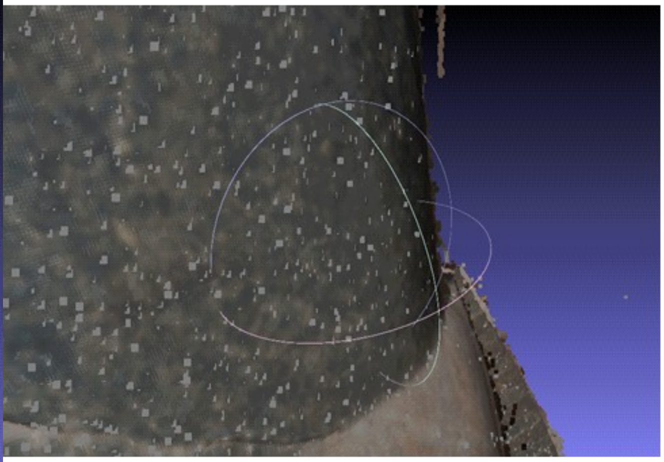


Figure 20 : Comparison of Apero2NVM Sparse cloud and Micmac final cloud

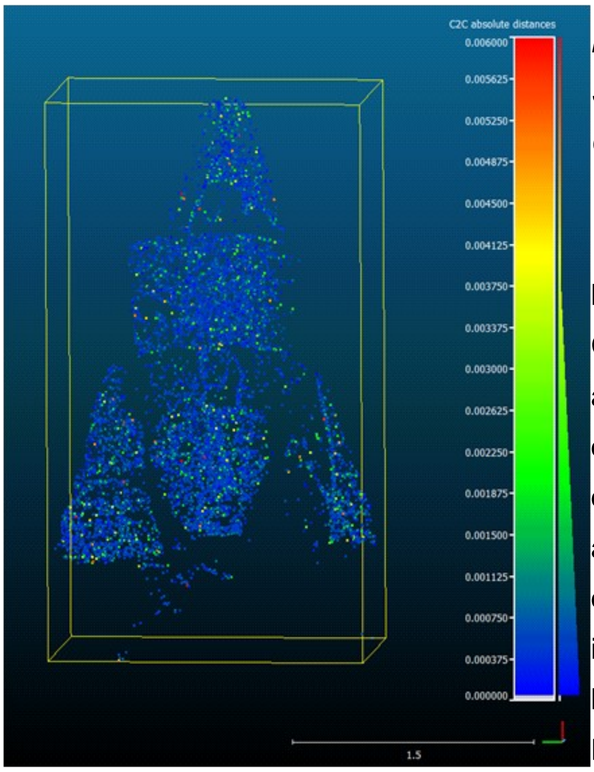


Figure 21 : Comparison of Apero2NVM Sparse cloud and Micmac AperiCloud with Cloud Compare

This figure shows a comparison which has been computed on the software Cloud Compare. A distance cloud to cloud has been applied with no modification of the coordinate of the clouds. On the right a scale the difference which are undertaken between 0 and 0.006 spatial unit the value of a standard deviation. However the mean of the difference is smaller, 8.9 10^{-4} Spatial Unit, and all the blue points on the figure are below this value. Here a Spatial Unit is approximately 0.5 mm.

Different comparison among other photogrammetric solutions was made here but cut for simplification.

b) Other software against MicMac

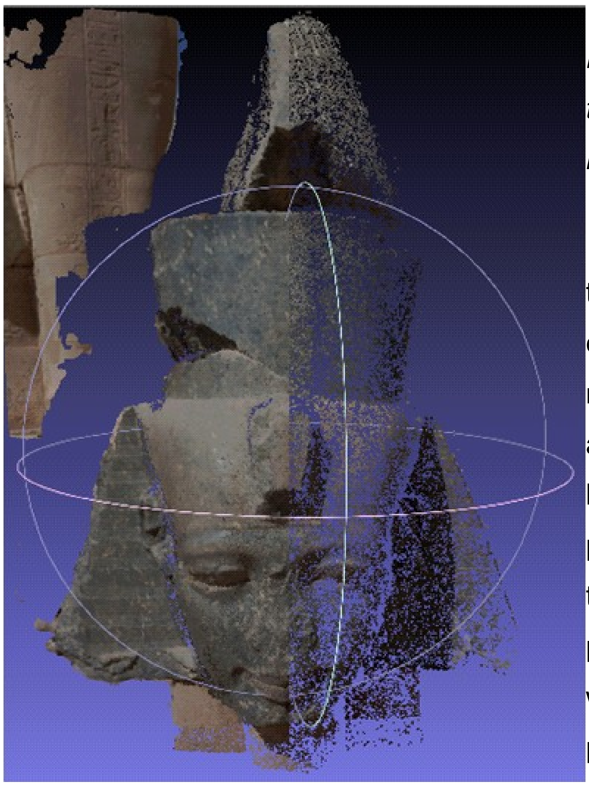


Figure 28 : Half of Micmac cloud on the left, on the right half of SURE cloud after correction, merged and filtered

We can see on the figure 28 on the left that after correction the central part of the two reconstructions correspond perfectly together. The main difference is the density of the cloud. Actually the two solutions producing the highest number of points are Micmac and MVE. SURE also produces a very dense result in less time than these two solutions if we add every depth maps produced together. Finally, the cloud extracted with CMPMVS features the lowest number of points.

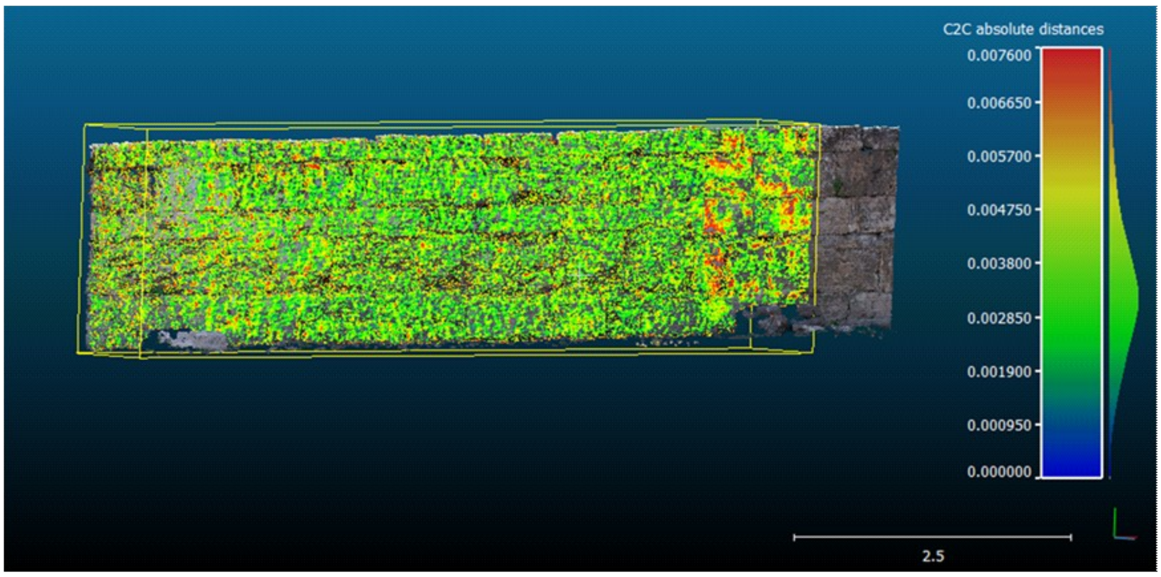


Figure 29 : MicMac and VisualSFM Dense cloud comparison

Here is a metrical comparison among VisualSfM and MicMac C3DC, done 8 times smaller than the original images, the mean of differences is 3.8 mm, the Standard Deviation is 3.1 mm for a Ground Sample Distance of 1.1 cm. So for this example the similarity of the result is very thin, a third of a GSD.