Software Engineering Report
3D Reconstruction with multiple photos

Shen JiYuan

***Abstract*—This project offer me an opportunity of having a better understanding of those advanced algorithm and also some famous library (which I didn't know before), and more importantly, learning how to take advantage of the huge potential utilization in them. This project is mainly about 3d reconstruction methods and improvement.**

***Index Terms*—Structure from motion ; 3D reconstruction ; Algorithm Optimization ;**

I. Introduction

In computer vision and computer graphics, 3D reconstruction is the process of capturing the shape and appearance of real objects, which can be accomplished either by active or passive methods. Our report mainly solve the question about how to recover 3D shape of an unknown object from information extracted from different views.

A. Application

The research of 3D reconstruction has always been a focus and difficulty. Using 3D reconstruction one can determine any objects 3D profile, as well as knowing the 3D coordinate of any point on the profile.The 3D reconstruction of objects is a generally scientific problem and core technology of a wide variety of fields, such as Computer Aided Geometric Design(CAGD), Computer Graphics, Computer Animation, Computer Vision, medical imaging, computational science, Virtual Reality, digital media, etc. For instance, the lesion information of the patients can be presented in 3D on the computer, which offers a new and accurate approach in diagnosis and thus has vital clinical value.

B. Main Problem

1) Image acquisition :

2D digital image acquisition is the information source of 3D reconstruction. Commonly used 3D reconstruction is based on two or more images, also it may only employ one single image sometimes. There are various types of methods for image acquisition that depends on the occasions and purposes of the specific application. Not only the requirements of the application must be meet, but also the visual disparity, illumination, performance of camera and the feature of scenario should be considered.

2) Camera calibration :

Camera calibration in Binocular Stereo Vision refers to the determination of the mapping relationship between the image points P1(u1,v1) and P2(u2,v2), and space coordinate P(xp, yp, zp) in the 3D scenario. Camera calibration is a basic and essential part in 3D reconstruction via Binocular Stereo Vision.

3) Feature extraction :

The aim of feature extraction is to gain the characteristics of the images, through which the stereo correspondence processes. As a result, the characteristics of the images closely link to the choice of matching methods. There is no such universally applicable theory for features extraction, leading to a great diversity of stereo correspondence in Binocular Stereo Vision research.

4) Stereo correspondence :

Stereo correspondence is to establish the correspondence between primitive factors in images, i.e. to match P1(u1,v1) and P2(u2,v2) from two images. Certain interference factors in the scenario should be noticed, e.g. illumination, noise, surface physical characteristic and etc.

5) Restoration :

According to precise correspondence, combined with camera location parameters, 3D geometric information can be recovered without difficulties. Due to the fact that accuracy of 3D reconstruction depends on the precision of correspondence, error of camera location parameters and so on, the previous procedures must be done carefully to achieve relatively accurate 3D reconstruction.

II. Preparation

In order to do the project, we first need to set up an environment, visual studio with openCV and PCL lib. Both of them can be downloaded for free.

A. Main steps

The task we are dealing with can be divided into several steps : extracting information from pictures, retrieving camera pose, and locating 3D points. Current 3D reconstruction techniques is useful in many applications, such as photo tourism, 3D map view and many other applications. And our project is aim at solving all the problems, and thus reconstructing 3D point cloud with sets of frames with unknown angles.

B. Feature Detection

Feature detection and matching techniques plays an important role in our application and as well as many computer vision areas. Luckily with the help of openCV, we can now easily use some good and mature algorithm as feature detector.

Some early development introduced finding feature points that are corner-like. Interestingness could also be evaluated as high change of intensity using various sliding window functions and this is the backbone of the work done by people like F?rstner and then Harris and Stephens . While eigenvalue based detectors like this, such as the Harris detector has rotation invariance, they are non-invariant to image scale.

Using of relevant algorithm goes as following:

```
//Using Canny in openCV
//void cvCanny(const CvArr* image,CvArr* edges,
    double threshold1,double threshold2,int
    aperture_size=3 );
//One thing you should pay attention to is that
//aperture_size=3/5/7
IplImage * src=NULL;
IplImage * dst=NULL;
dst = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U
    ,1);
cvCanny(src,dst,50,150,3);

//Using HoughLines in openCV
//void HoughLines( InputArray image,OutputArray
    lines,double rho,double theta,int threshold,
    double srn=0, double stn=0 )
Mat src;Mat dst;
cvtColor (src,dst,CV_BGRA2GRAY);
Mat contours;Canny (dst,contours,125,350);
vector<Vec2f> lines;
HoughLines (contours,lines,1,PI/180,80);

//Using Harris in openCV
//void cornerHarris(InputArray src,OutputArray
    dst,int blockSize,int ksize,double k,int
    borderType=BORDER_DEFAULT );
Mat dst, dst_norm;
dst = Mat::zeros( src.size(), CV_32FC1 );
int blockSize = 2;
int apertureSize = 3;
double k = 0.04;
cornerHarris( src_gray, dst, blockSize,
    apertureSize, k, BORDER_DEFAULT );
normalize( dst, dst_norm, 0, 255, NORM_MINMAX,
    CV_32FC1, Mat() );
convertScaleAbs( dst_norm, dst_norm_scaled );

//And Shi-Tomasi is the improvement of Harris
//void goodFeaturesToTrack(InputArray image,
    OutputArray corners,int maxCorners, double
    qualityLevel, double minDistance,InputArray
    mask=noArray(),int blockSize=3,bool
    useHarrisDetector=false,double k=0.04);
maxCorners = 1;
vector<Point2f> corners;
double qualityLevel = 0.01;
double minDistance = 10;
int blockSize = 3;
bool useHarrisDetector = false;
double k = 0.04;
goodFeaturesToTrack(src_gray,corners,maxCorners,
    qualityLevel,minDistance,Mat(),blockSize,
    useHarrisDetector,k);
```

More modern techniques such as Lowes Scale Invariant Feature Transform (SIFT) use sampling of scale space to detect points that are invariant to changes in both scale and rotation. This inspired the development of the Speeded Up Robust Feature (SURF) detector and has some gains in speed and robustness. SURF is implemented in OpenCV where it is widely used for various projects and is known to perform reasonably well. Since in this project, we choose to do some improvement on current SURF algorithm, so here I will simply omit SIFT code.

Using of SURF goes as following,with the help of openCV basic SURF is simple:

```
cv::Mat dst;
cv::Mat src = cv::imread(ImageName);
cv::cvtColor(src,dst,CV_BGR2GRAY);
std::vector<cv::KeyPoint> keypoints;
cv::SurfFeatureDetector surf(2500);
surf.detect(image,keypoints);
```

C. Structure from Motion

SFM refers to the process of estimating three-dimensional structures from two-dimensional image sequences which may be coupled with local motion signals. In order to perform reconstruction from correspondences points, there are math calculation involved, especially some matrix techniques. The math covers obtaining the Fundamental matrix from corresponding points, and extracting Camera matrices.

The location of points in multiple images can eventually lead to an estimation of its 3D location. Multiple locations and matches call for the need for fundamental geometry to describe relationships between images. The general taxonomy of this was proposed by Scharstein and Szeliski , and this involves the concept of epipolar geometry. The illustration below shows the concept of epipolar geometry where a plane could be bounded at the 3D location of the point and two other locations of where this point appears on two pictures. All possible configurations of this plane will include epipolar line segments that intersect at an epipole on each image. This means knowing where the points are on an epipolar line segment means its partner is on the corresponding epipolar line segment of the second pictures. Find 7 or more good matching points will allow the for estimation of the Fundamental matrix which would describe the necessary epipolar geometry

To explain this, I will use some photos:

Given a single image, the three-dimensional location of any visible object point must lie on the straight line that passes through the centre of projection and the image of the object point see the figure below. The determination of the intersection of two such lines generated from two independent images is called triangulation.

Clearly, the determination of the scene position of an object point through triangulation depends upon matching the image location of the object point in one image to the location of the same object point in the other image. The process of establishing such matches between points in a pair of images

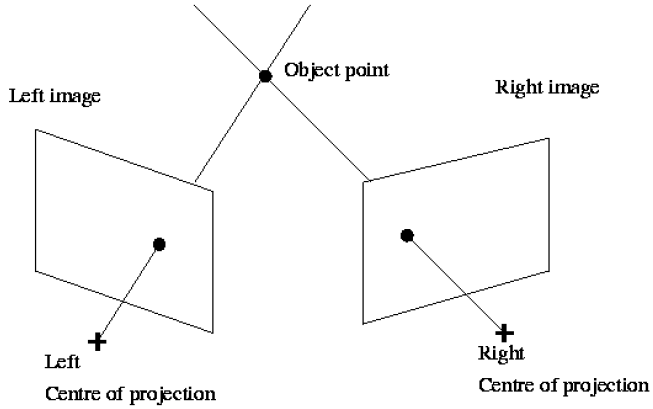

Fig. 1. The principle of triangulation in stereo imaging.

is called correspondence. It might seem that correspondence requires a search through the whole image, but the epipolar constraint reduces this search to a single line.

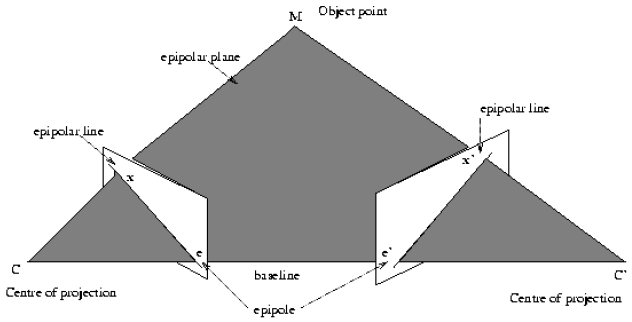

Fig. 2. The epipolar constraint

The epipole is the point of intersection of the line joining the optical centres, that is the baseline, with the image plane. Thus the epipole is the image, in one camera, of the optical centre of the other camera. And the epipolar line is the straight line of intersection of the epipolar plane with the image plane. It is the image in one camera of a ray through the optical centre and image point in the other camera. All epipolar lines intersect at the epipole.

Thus, a point x in one image generates a line in the other on which its corresponding point x' must lie. We see that the search for correspondences is thus reduced from a region to a line.

To calculate depth information from a pair of images we need to compute the epipolar geometry. In the calibrated environment we capture this geometric constraint in an algebraic representation known as the Essential matrix. In the uncalibrated environment, it is captured in the Fundamental matrix.

This has fewer degrees of freedom, as now there is only rotation and translation with a scale ambiguity, and this gives rise to fewer solutions when extracting camera matrices. Setting an initial origin point for the first camera matrix (P) allows the

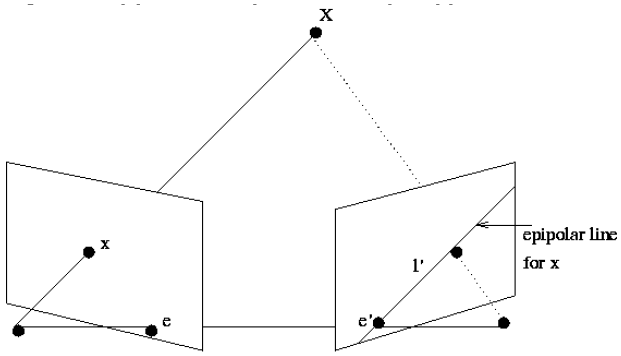

Fig. 3. The epipolar line along which the corresponding point for x must lie

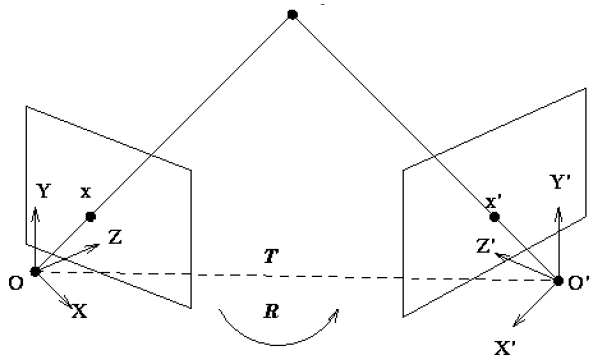

Fig. 4. The Euclidean relationship between the two view-centred coordinate systems

second camera matrix (P) to be computed once the Essential matrix is factored. The factored solution gives two possible solutions for rotation and a choice of sign for translation. These solutions could be interpreted geometrically: 1) Two cameras pointing inwards towards object; 2) Accidentally reversing the input for case 1; 3) Two cameras pointing at an object, but facing away from each other; 4) Accidently reversing the input for case 3.

Providing both Camera matrices and a list of calibrated 2D points from both images allow triangulation to take place to estimate 3D locations.Due to the ambiguity in the possible solutions, this may not always work. In the case of cameras that are faced away from each other, rays pointing outwards will never intersect or even come close to matching. However, picking the solution where the rays could point in the other direction, where virtually the rays are heading in the direction behind the image planes, will give a perfectly acceptable 3D location as scale is ambiguous either way when working with the Essential matrix.

D. Perspective-n-Point problem

The perspective-n-point problem (PnP) is one of the issues in the process of pose recovery. It could be interpreted as deduction of camera pose matrix, given a set of 3D point, and their corresponding 2D projections.

The Opencv build in function solvePnP implemented the algorithm to resolve K from provided matching 3D-2D pair by minimizing projection error.

E. Triangulation

In real world cases, it is rare for two rays to intersect in 3D space, as there is tiny rounding problem or other unsolvable issues. Previous researches provide many ways to optimize the intersect point with the two rays.

The method we applied was based on Richard and Hartleys Optimal Method of Triangulation algorithm :

1. Parameterize the pencil of epipolar lines in the first image by a parameter t. Thus an epipolar line in the first image may be written as (t).

2. Using the fundamental matrix F, compute the corresponding epipolar line (t) in the second image.

3. Express the distance function $d(u,(t))^2 + d(u,(t))^2$explicitly as a function of t.

4. Find the value of t that minimizes this function.

III. Implement

A. Improved Feature Matching

One of the most widely used feature detectors is the SIFT (Scale-invariant feature transform). It uses the maxima from a Difference-of-Gaussians (DOG) pyramid as features. The first step in SIFT is finding a dominant gradient direction. To make it rotation-invariant, the descriptor is rotated to fit this orientation. But by applying it to our picture, the result doessn't goes well. Then I have the idea of combining the matches from SIFT and CornerHarris, as our own feature detectors, and actually the result is really hehehe.

So here is another common feature detector, SURF (Speeded Up Robust Features). In SURF, the DOG is replaced with a Hessian matrix-based blob detector. Also, instead of evaluating the gradient histograms, SURF computes for the sums of gradient components and the sums of their absolute values.

However, since our pictures are taken in such a bad condition, The feature detectors used provide plenty of good matches, but also had a significant amount of noise. As these points are used to generate 3D coordinates, which in turn act as a reference for images further along the pipe, it was vital that the feature detector is as robust as possible. Enhancements to the SURF feature detector were made to filter out bad matches and this included process such as KNN ratio test, symmetry test, and RANSAC.

A SURF feature detector was first used to detect interesting points in both images and provided a list of Key Points for both images. Descriptors were then computed for either images or Key Points. For the ratio test, a BruteForceMatcher obtained two nearest neighbors through comparing the descriptors from both discriptor1 to discriptor2, and from descriptor2 to descriptor1. This would be used for the symmetry test later, but in the meantime, features that have a very distinct nearest neighbor, or where the first neighbor is much better than the second neighbor, are desired over with similar neighbors. The ratio test removes matches that are deemed bad, and the rest move through to the symmetry test. The two sets of DMatches obtained through the BruteForceMatcher that survived the ratio test are compared to see if a match from one image to another also has a pair in the other direction. Similarly, non-symmetrical matches are removed. The resulting matches are passed through a RANSAC test and these returns the Fundamental matrix. Outliers are filtered out.

Finally, we assume that if results of the matches of two images are used to be good references for further images, they must have plenty of matches. We set the threshold to be 30 matches. Otherwise, the user must provide better photos.

B. Structure from Motion

As discussed in the background, projective ambiguity can be avoided by converting a fundamental matrix into an essential matrix. This can be done using a camera calibration matrix K, generated from a provided frame. The matrix is constructed to represent reasonable scaling and assumes the focus is at the very center of the image.Given both images are assumed to be as a result of the same cameras, the conversion can be simplified to: $Mat_<double> E = K.t() * F * K;$

Decomposing E to obtain a P is simplified in OpenCV as there is a very useful SVD class that provides U and V matrices as described in the equation. Multiplying these with the appropriate orthogonal matrix and skew-symmetric matrix gives us two matrices for rotation, and a translational vector that can have either positive or negative signs. Constructing the P matrix gives four possible combinations, and it is difficult to tell which solution is valid in the real world unless triangulation is performed. As this operation is only performed once in the entire program run time, and involves only two frames, it was decided the user has to make the best judgment for what solution is required.

Initial 3D points generated and their links to certain Key Points are important for the additions of further images. The model here is checked to see valid before lookup entries are added to a table to be used in solving PnP problem.

C. Solving Perspective-n-Point problem

Also the Structure from motion implementation could recover camera matrix between different frames; however, the method is not applicable to other frames, which are not initial frames.As a result, only using SFM leads to the recovered point cloud shifted and rotated after each iteration, which in other words high projection error.

The reason behind is that structure from motion is a rough method to predict camera pose, mismatching features and rounding problem all could leads to errors. Even though RANSAC is applied to optimize the result, the pose estimated still could not be applied to add new frames without further calibration.

In order to predict camera matrix P that not violated to the existing point cloud, we use predicted cloud point to predict new camera pose. There is an openCV function called solvePnP (), which takes a set of 2D-3D matching points, Calibration matrix and Distortion matrix to output Camera Matrix.

Therefore, it is possible to predict camera matrix with existing cloud point. To find the 2D-3D matching pair, we implemented a Lookup Table which stores 2D coordinates of current frame and its triangulated 3D coordinates. We keep track of 2D-3D pair during each iteration of processing new frames. When new frames come in, feature mapping is performed with the previous frame. Keypoints are then searched up in the Lookup Table to get known 3D coordinates. Now we have a new 2D-3D pair of association.

By passing the 2D-3D pair of matching to solvePnP(), we have the new camera matrix that minimizes projection error. Triangulation then generates new 3D co-ordinates.

D. Triangulation

At first we choose the Linear-LS method described by Richard and Hartley as an option. However, the result is not so good. So we further extended our solution to Iterative-LS method by assigning weight in each iteration to generate more accurate result.

E. 3D Model Representation

The representation of point cloud is written in output file in ply format, it consist of the points' coordinates and RGB values.

Form as following:

float x float y float z uchar diffuse_red uchar diffuse_green uchar diffuse_blue

etc. -0.873216 0.076262 -4.77054 173 174 169

Then we use Meshlab to render our ply file.

F. Getting contour

In this part, at first I choose to using KNN search to find the normalization of each points, then by compare this value with each other, we will choose those points with

But later, I occasionally knew PCL, which is a big library to deal with point cloud. The library is a big one, so I will not put this part include the project. Since the implement is very easy, you can choose whether you want to test this part,if you do want, please contact me. I will tell you how to build the environment and run my project.

I will attach part of code below to describe how to use it.

```
// load point cloud data in
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud2(new
    pcl::PointCloud<pcl::PointXYZ>);
pcl::io::loadPCDFile ("test_pcd.pcd", *cloud2);

// compute normals;
 pcl::search::Search<pcl::PointXYZ>::Ptr tree(
     new pcl::search::KdTree<pcl::PointXYZ>());
 pcl::PointCloud<pcl::Normal>::Ptr normals(new
     pcl::PointCloud<pcl::Normal>);
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal
    > normal_est;
normal_est.setSearchMethod(tree);
normal_est.setInputCloud(cloud2);
normal_est.setKSearch(50);
normal_est.compute(*normals);

 //calculate boundary;
 pcl::PointCloud<pcl::Boundary> boundary;
 pcl::BoundaryEstimation<pcl::PointXYZ, pcl::
     Normal, pcl::Boundary> boundary_est;
 boundary_est.setInputCloud(cloud2);
 boundary_est.setInputNormals(normals);
 boundary_est.setRadiusSearch(0.2);
 boundary_est.setAngleThreshold(PI/4);
 boundary_est.setSearchMethod(pcl::search::
     KdTree<pcl::PointXYZ>::Ptr(new pcl::search
     ::KdTree<pcl::PointXYZ>));
 boundary_est.compute(boundary);

 //save boundary data
 pcl::io::savePCDFileASCII("bound.pcd",boundary
     );
```

IV. Result

As part of the report, I have to say, the result doesn't come out as good as I imagine at first. I don't know how to describe this feeling, but to be honest, I have to say, it's a easy job to finish with the help of SIFT + BUNDLE + PMVS , which can be found all online.

Bundler takes a set of images, image features, and image matches as input, and produces a 3D reconstruction of camera and (sparse) scene geometry as output. While Bundler producing sparse point clouds, for denser points, there is another beautiful software package called PMVS2 for running dense multi-view stereo. By running Bundler to get camera parameters, using the provided Bundle2PMVS program to convert the results into PMVS2 input, then running PMVS2, you can simply get a 3d point cloud of your object both precisely and efficiently.

Although at first my object is to repeat the whole process in my own way, and do some improvement, now it looks like I am just too young too naive. Cause my project can only take pictures which are taken from a close view, otherwise, as you can see, you may meet error when using PnP.

Better or worse, it is still my project, so let see the result.

TABLE I
A COMPARISON OF SIFT, PCA-SIFT AND SURF

method	Time	Scale	Rotation	Blur	Illumination	Affine
Sift	common	best	best	common	common	good
PCA-Sift	good	good	good	best	good	best
Surf	best	common	common	good	best	good

And this is my point cloud produced, if you are willing to use it as a Bundle input and thus doing further calculation, please contact me,I am willing to provide some tips when you install them in Linux.

Actually, since it's just the pure output of my project without any further help with current software, I have to say it's quite satisfying though not look as good as the output of PMVS.

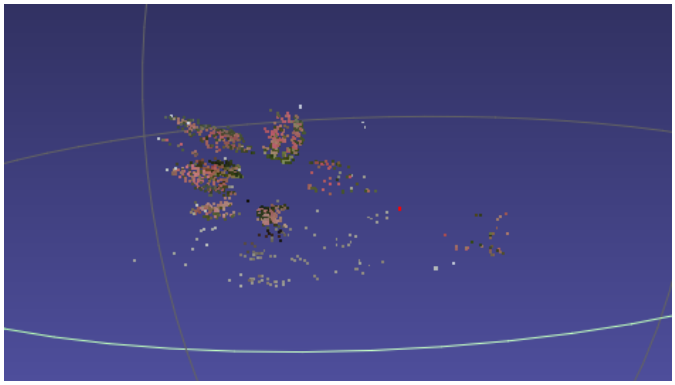

Fig. 5. The result of only using four images

Actually, after enhancing with some tools, you can get a really good point cloud, and thus a really good contour.

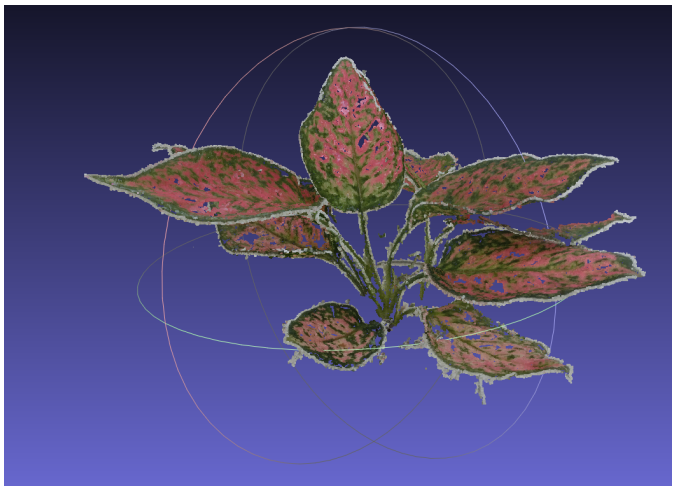

Fig. 6. A really Prefect result

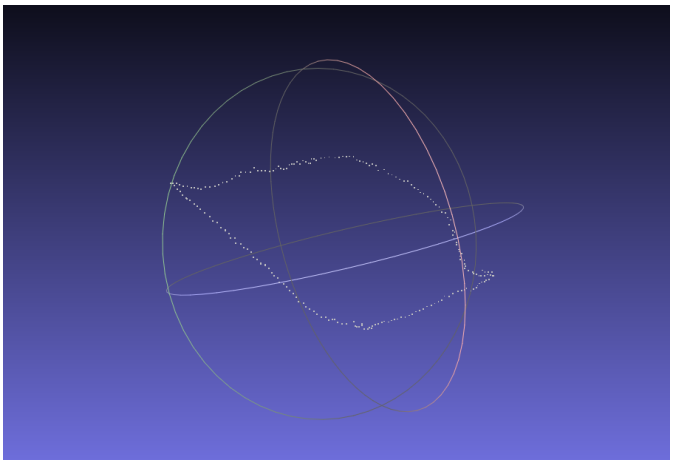

Fig. 7. Contour points info

V. Thanks

I really have to thank to my mentor for his help, it does mean a lot.