Research on the Problem of 3D Reconstruction Based on Multi-view Images

Caoliwen Wang

June 2, 2024

Abstract

This paper conducts mathematical modeling research on the problem of 3D reconstruction in the field of computer vision, aiming to provide effective solutions and algorithm implementations through systematic analysis and modeling methods. We ultimately achieve the reconstruction of 3D models of surfaces based on multi-view images of objects. First, we describe in detail the background and requirements of the problem, clarifying the objectives and scope of the research on the 3D reconstruction problem. Next, through problem analysis, we gain a comprehensive understanding and breakdown of the problem, proposing several assumptions to simplify the complexity of the issue. In the model establishment and solving part, we complete the model establishment and solving in three steps, which are camera model establishment, feature point matching and camera calibration, and point cloud reconstruction and fusion. We use some statistical methods and optimization algorithms to construct and solve the model. We detail the model establishment process, solving steps, and result analysis, and demonstrate the application effect of the model through case analysis. In addition, this paper discusses the implementation and optimization of the code, introducing related programming techniques and acceleration methods at the implementation level to ensure the efficient operation of the model. Finally, we further discuss the algorithms for model solving and analyze the advantages and disadvantages of the model.

Key Words: 3D reconstruction, feature point matching, point cloud fusion

Contents

1 Problem Restatement

1.1 Problem Background

Structure from Motion (SFM) and Multi-View Stereo (MVS) are two key algorithms in the field of 3D reconstruction, playing important roles in recovering 3D information from 2D images. SFM reconstructs the 3D structure of a scene by analyzing images from different viewpoints without prior knowledge of the camera parameters, relying on correspondences between images to estimate the camera's motion trajectory and the scene's structure. MVS, on the other hand, reconstructs the 3D structure of a scene based on multiple views, usually assuming known camera parameters, and is used to reconstruct dense 3D models. The significance of these two algorithms lies in their ability to recover complex 3D information from simple 2D images, with wide applications in computer vision, remote sensing, virtual reality, and other fields.

In 3D reconstruction, the application of SFM and MVS is particularly important. SFM can be used for reconstructing large-scale scenes, such as cityscapes and archaeological sites, as well as for reconstructing dynamic scenes from drone aerial photography and mobile devices. MVS has important applications in cultural heritage preservation, game development, and film production. These applications not only provide more realistic and intuitive visual experiences but also offer crucial data support for urban planning, environmental monitoring, and architectural design. Furthermore, with technological advancements and growing application demands, SFM and MVS technologies are increasingly applied in fields such as digital city modeling and intelligent transportation systems. Therefore, as a key step in these technologies, camera calibration has also received corresponding attention and promotion in research and applications.

1.2 Problem Requirements

Based on the above background, we need to establish a mathematical model to address the following problems:

1. Establish a simple mathematical model of the camera;

2. Address the problem of camera calibration, that is, calculate the camera parameters from multiple 2D images. In this task, we primarily focus on the camera's extrinsic parameters, including the rotation angles and center coordinates;

3. Address the problem of 3D reconstruction, that is, reconstruct the geometry of a 3D object from multiple 2D images. The 3D object's geometry can typically be represented by point clouds, meshes, or signed distance fields. In this task, we mainly achieve the reconstruction of the 3D object's point cloud and mesh.

2 Problem Analysis

2.1 Analysis of Problem 1

Problem 1 requires us to establish a simple mathematical model of the camera.

In computer vision, the most commonly used camera model is the pinhole camera model, which simplifies the camera to a pinhole without thickness. Through this pinhole, light travels in a straight line and projects onto the imaging plane. This model mainly includes the camera's intrinsic parameters (such as focal length, principal point, etc.) and extrinsic parameters (the camera's position and orientation in the world coordinate system).

2.2 Analysis of Problem 2

Problem 2 requires us to establish a mathematical model for camera calibration.

The core of this problem is to use known multiple 2D images to determine the camera's extrinsic parameters, i.e., the camera's position and orientation in the world coordinate system. It is important to note that the data generally needs to be sequential, meaning adjacent photos typically have high similarity without significant differences. The common steps for calibration are as follows:

1. Detect feature points in the images;
2. Match feature points in sequentially adjacent images using various methods;
3. Combine the camera mathematical model and geometric relationships to establish and solve the equations.

2.3 Analysis of Problem 3

Problem 3 requires us to perform 3D reconstruction.

The core of this problem is to reconstruct the geometric features of a 3D object, such as point clouds and meshes, from known multiple 2D images. In fact, there are many works on reconstructing meshes from point clouds, with classical methods such as Poisson Reconstruction, which uses the Poisson equation to reconstruct a smooth surface from a point cloud; Alpha Shapes, which defines the points in the point cloud that form the surface of the object by adjusting parameters; Ball Pivoting, which rolls a ball over the point cloud to create triangular meshes based on the distances between points; and Delaunay Triangulation, which constructs Delaunay triangulation on the convex hull of the point cloud and then trims or refines the mesh based on the density of the point cloud or other attributes.

Therefore, the core of this problem is transformed into how to reconstruct a 3D point cloud from multiple 2D images. Firstly, to obtain a sparse point cloud, we use the classic SFM algorithm, considering the following steps:

1. Calculate the camera extrinsics for two adjacent images using the model from Problem 2;

2. In practice, the same camera can only have one set of extrinsics. However, we calculate the extrinsics for adjacent models based on the model from Problem 2, so we need to unify the extrinsics. We achieve this by gradually merging the sparse point clouds, considering the use of Bundle Adjustment (BA) during the merging process, and removing singular points with excessively large values in the calculations.

A sparse point cloud is far from meeting our reconstruction requirements, so we need to reconstruct a dense point cloud. Here, we adopt the classic MVS approach, utilizing matching propagation.[1]

3 Problem Assumptions

3.1 Assumptions for Model 1

For the simple camera model, we make the following assumptions:

1. Pinhole model assumption: Assume that the camera has no lens distortion, and the imaging process can be approximated by the pinhole model;

2. Small aperture assumption: Assume that light does not refract inside the camera, i.e., light does not deflect when passing through the small aperture;

3. Straight-line propagation assumption: Assume that light propagates in a straight line in a vacuum or homogeneous medium; otherwise, the propagation would be too complex to calculate;

4. Imaging plane assumption: Assume that the imaging plane (photosensitive element) is located on the focal plane of the pinhole, forming a clear image.

3.2 Assumptions for Model 2

For the camera calibration model, we make the following assumptions:

1. Constant intrinsic parameters assumption: Assume that the camera's intrinsic parameters (such as focal length and principal point) are constant across all images, usually requiring the data to be captured by the same camera;

2. Perspective projection assumption: Assume that the lines in the image are straight lines in 3D space, i.e., no perspective distortion is considered;

3. Camera motion assumption: Assume that the camera's motion is smooth without sudden large jumps when capturing different images, requiring the data to be continuous and orderly, i.e., adjacent photos change the viewing angle only slightly for the same object, which can be achieved by capturing a video and generating frames.

3.3 Assumptions for Model 3

For the 3D reconstruction model, we make the following assumptions:

1. Homography assumption: Assume that the camera motion between adjacent images can be described by a homography matrix;
2. Projective transformation assumption: Assume that points in the image follow projective transformation, i.e., straight lines in 3D space are still represented as straight lines in the image;
3. Parallel lines assumption: Assume that parallel lines at infinity converge to the same vanishing point in the image during reconstruction;
4. Light consistency assumption: Assume that the light rays observed from the same 3D point in different images are parallel;
5. 3D point consistency assumption: Assume that the same object or scene points observed in multi-view images are fixed and do not change over time, requiring reconstruction of stationary objects; otherwise, there will be significant noise.

4 Notation Explanation

Table 1: **Notation Explanation**

Notation	Explanation	Unit
X	Homogeneous coordinates of the target point in 3D	Millimeter (mm)
x	Homogeneous coordinates on the imaging plane	Millimeter (mm)
f	Focal length of the camera	Millimeter (mm)
$\boldsymbol{K}$	Camera intrinsic matrix	–
$[\boldsymbol{R}\|\boldsymbol{t}]$	Camera extrinsic matrix	–
$\boldsymbol{F}$	Fundamental matrix of matching	–
$\boldsymbol{E}$	Essential matrix of the camera	–

5 Model Establishment and Solution

5.1 Simple Camera Model

Definition 1 *In the camera coordinate system, a projection plane is fixed to project points in three-dimensional space onto the imaging plane (as shown in Figure 1). The distance from the camera to the imaging plane is called the focal length, denoted as f.*

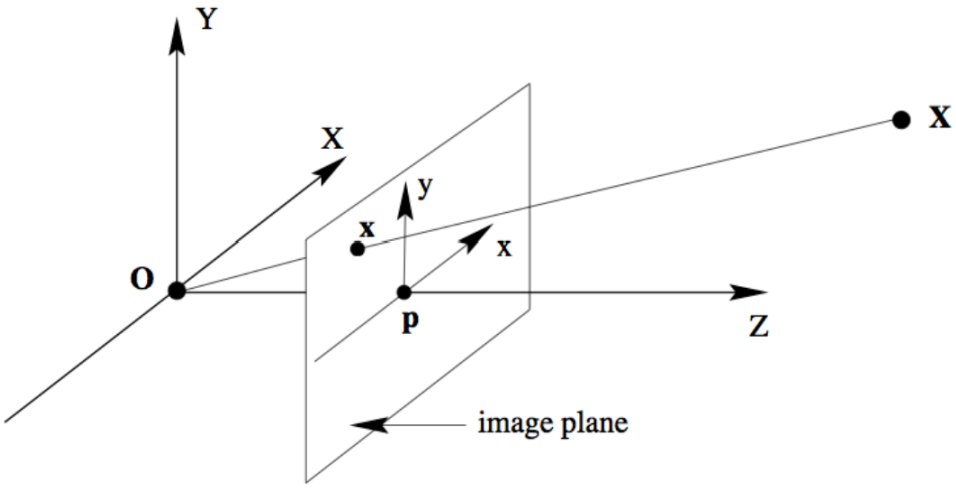


Figure 1: Imaging plane and focal length

When the camera is at the origin, we denote the homogeneous coordinates of the object point and the image point in the camera space as $\boldsymbol{X}$ and $\boldsymbol{x}$, respectively. Based on the assumption of light traveling in straight lines, the projection relationship can be obtained as:

$$\begin{bmatrix} x \\ y \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

To ensure the image has a consistent pixel scale, we also need to normalize the homogeneous coordinates on the imaging plane:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

In the derivation of equations (1) and (2), we assume the camera is at the origin, but this is not always the case. Denote $\boldsymbol{X}_{cam}$ as the coordinates in the camera coordinate system. We need to describe the motion of the camera relative to the world coordinate system through rotation and translation. In homogeneous coordinates, we have the following lemma:

Lemma 1 *Let $\boldsymbol{R}$ be the orthogonal matrix describing the camera orientation, and $\boldsymbol{t}$ be the translation describing the camera position. Then we have:*

$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Define the following notation:

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$[\boldsymbol{R}|\boldsymbol{t}] = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$$

These two matrices respectively describe the internal and external parameters of the camera. From Lemma 1 and equations (1) and (2), we obtain the imaging equation of the camera model:

$$\boldsymbol{x} = \boldsymbol{K}\,[\boldsymbol{R}|\boldsymbol{t}]\,\boldsymbol{X} \tag{3}$$

In the subsequent establishment and derivation of the model, we will assume that each camera has the same internal parameters (i.e., focal length).

5.2 Camera Calibration Model

5.2.1 Feature Point Detection and Matching

In the process of camera calibration, feature points refer to points in the image with unique, distinguishable, and stable attributes, which can be used to determine the correspondence between points in the image and the actual scene. Feature points are usually selected as points that are easy to recognize and stably appear in multiple images, such as corners, edge points, or blobs. Accurate detection of feature points is crucial for establishing a precise imaging model in camera calibration.

Here we use the commonly used SIFT and SURF for feature point detection and matching tasks. SIFT (Scale-Invariant Feature Transform) is a scale-invariant feature point detection algorithm that is invariant to image scaling and rotation. SIFT detects feature points by finding extreme points in different scale spaces and extracts the orientation information of the feature points. SURF (Speeded Up Robust Features) is an accelerated version of SIFT. It uses integral images to quickly compute feature points and is also invariant to image scaling and rotation.

For the specific matching, we use feature descriptors. A feature descriptor is a vector that contains image information around the feature point for the subsequent matching process. We first set the threshold for the difference between the feature descriptors of the two to be 0.7^2. If enough feature point matches cannot be found, the threshold is gradually relaxed. This relaxation process is output during code execution.

To ensure the accuracy of the matching, we use a bidirectional matching method, which requires that each feature point match only when two feature points match each other to avoid one-to-many matching problems.

Additionally, we should pay attention to improving the efficiency of matching, which is described in the special processing and optimization in the code.

5.2.2 Camera Calibration Based on Two Images

Definition 2 *For two adjacent images, for any pair of matching feature points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, a matrix $\boldsymbol{F}$ that satisfies*

$$\boldsymbol{x}_1^T \boldsymbol{F} \boldsymbol{x}_2 = 0$$

is called a fundamental matrix.

Solving the fundamental matrix essentially involves solving a linear system. We only need to find enough constraints, which means finding enough matching feature points. For this 9-dimensional homogeneous linear system, at most 8 pairs of feature points are needed to uniquely solve the fundamental matrix $\boldsymbol{F}$ up to a scale factor.

Definition 3 *For a given fundamental matrix $\boldsymbol{F}$ and a pair of matching feature points x_1 and x_2, denote:*

$$e = x_2^T \boldsymbol{F} x_1,$$

$$J = \frac{\delta(x_i'^T \boldsymbol{F} x_i)}{\delta x_i},$$

Define the corresponding first-order geometric error (Sampson Distance):

$$d(x_1, x_2) = \frac{e^T e}{J J^T}.$$

Then given a threshold τ, the criterion for determining an inlier is:

$$d(x_1, x_2) < \tau. \tag{4}$$

However, for two images, the number of matching feature points may far exceed 8. Therefore, we use the RANSAC algorithm to optimize the solution of $\boldsymbol{F}$. The algorithm follows these steps:

1. Randomly select 8 pairs of matching feature points;
2. Compute the fundamental matrix $\boldsymbol{F}$;
3. Calculate the number of inliers for this fundamental matrix using the criterion in equation (4);
4. Repeat steps 1-3, and select the fundamental matrix with the maximum number of inliers as the final estimate.

Definition 4 *For two given adjacent images and their corresponding fundamental matrix $\boldsymbol{F}$, the essential matrix of the two images is defined as*

$$\boldsymbol{E} = \boldsymbol{K_1^T F K_2}.$$

The following result holds:

Lemma 2 *Given the essential matrix of two adjacent images*

$$\boldsymbol{E} = \boldsymbol{U}\, diag(1, 1, 0)\boldsymbol{V}^T,$$

and the extrinsic matrix of the first camera

$$P_1 = \left[\boldsymbol{I} \,|\mathbf{0}\right],$$

the extrinsic matrix of the second camera can be one of the following four possibilities as shown in Figure 2:

$$P_2 = \left[\boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^{\boldsymbol{T}}\right| + \boldsymbol{u_3}]$$
$$P_2 = \left[\boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^{\boldsymbol{T}}\right| - \boldsymbol{u_3}]$$
$$P_2 = \left[\boldsymbol{U}\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{V}^{\boldsymbol{T}}\right| + \boldsymbol{u_3}]$$
$$P_2 = \left[\boldsymbol{U}\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{V}^{\boldsymbol{T}}\right| - \boldsymbol{u_3}].$$

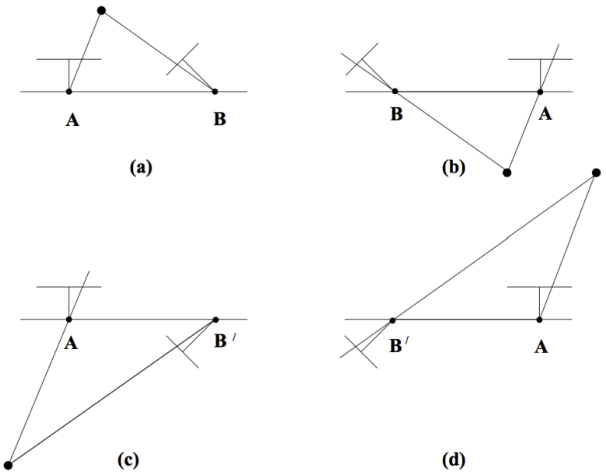


Figure 2: Possible configurations of the second camera in Lemma 2

Since we can always consider the case where the cameras differ by a rigid transformation, we can assume that the essential matrix of camera 1 is in the form of P_1 in Lemma 2. Consequently, the essential matrix of camera 2 can be directly obtained from the lemma.

We need to choose the optimal solution among these four possibilities. The criterion for selection is to ensure that as many points as possible lie in front of the cameras, which means the angle between the vector pointing from the camera center to the target point and the viewing direction is less than 90°. This method is called triangulation.

Through triangulation, we can obtain the extrinsic parameters of the second image's camera, thus completing the camera calibration model.

5.3 3D Reconstruction Model

5.3.1 Sparse Point Cloud Fusion

Starting from the two-camera calibration model, we solve the following system of equations based on the imaging formula (3):

$$\begin{cases} \boldsymbol{x_1} = \boldsymbol{K}\left[\boldsymbol{R_1}|\boldsymbol{t_1}\right]\boldsymbol{X}, \\ \boldsymbol{x_2} = \boldsymbol{K}\left[\boldsymbol{R_2}|\boldsymbol{t_2}\right]\boldsymbol{X}. \end{cases} \tag{5}$$

In this system of equations, we have already obtained $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, $[\boldsymbol{R_1}|\boldsymbol{t_1}]$, and $[\boldsymbol{R_2}|\boldsymbol{t_2}]$. Therefore, solving for the target points in 3D space is straightforward. Once each matching feature point is solved, 3D reconstruction based on the corresponding feature points of the two images is complete. Next, we will merge the point clouds reconstructed from multiple different images.

Based on Image1 and Image2, we can calculate the corresponding camera positions and orientations $[\boldsymbol{R}_1|\boldsymbol{t}_1]$ and $[\boldsymbol{R}_2|\boldsymbol{t}_2]$; similarly, based on Image2 and Image3, we can calculate the corresponding camera positions and orientations $[\boldsymbol{R}_2|\boldsymbol{t}_2]$ and $[\boldsymbol{R}_3|\boldsymbol{t}_3]$. However, the parameters for camera 2 obtained from the two sets of images might differ. In reality, there is only one unique set of camera parameters. To enhance the geometric consistency of our camera parameter estimates, we need to adjust and optimize the camera parameters.

Our optimization target is the extrinsic parameters of the cameras $[\boldsymbol{R}_1|\boldsymbol{t}_1]$. We must ensure that the reconstructed point cloud best fits the existing 2D image data, i.e., the corresponding point errors on the given images Image1, Image2, and Image3 are minimized. This leads to the following optimization problem:

$$min \sum_i \sum_j \left(\widetilde{\boldsymbol{x}}_i^j - \boldsymbol{K}\left[\boldsymbol{R_i}|\boldsymbol{t_i}\right]X^j\right)^2$$

where $\widetilde{\boldsymbol{x}}_i^j$ represents the 2D observation data of the jth point from the ith camera, and X^j represents the 3D coordinates of the jth point. Solving this problem will complete the optimization of the camera extrinsics and the 3D coordinates, a process known as Bundle Adjustment, which is formulated as:

$$[\boldsymbol{R}_i|\boldsymbol{t}_i], X^j = \sum_i \sum_j \left(\widetilde{\boldsymbol{x}}_i^j - \boldsymbol{K}\left[\boldsymbol{R_i}|\boldsymbol{t_i}\right]X^j\right)^2. \tag{6}$$

Equation (6) is a nonlinear optimization problem. We use TO to solve it.
Based on the optimized camera parameters, we can reconstruct the fused point cloud using the feature points from the images. For cases with more images, we first select two images to build the initial point cloud, and then iteratively add new points to the point cloud following the above process, eventually obtaining a point cloud fused with feature points from each image.

5.3.2 Dense Point Cloud Reconstruction

We have obtained a sparse point cloud from the given images. However, this point cloud is derived from feature point matching and recalculation, and its quantity is still insufficient for

reconstructing the object's geometric model. Therefore, we need to use methods to increase the number of points. Here, we use the Matching Propagation method to establish Multiple View Stereo (MVS).

Definition 5 *For a given pair of matched feature points, define:*

$$ZNCC(x_1, x_2) = \frac{\sum_i (I(x_1 + i) - \bar{I}(x_1))(I(x_2 + i) - \bar{I}(x_2))}{\sqrt{\sum_i (I(x_1 + i) - \bar{I}(x_1))^2 \sum_i (I(x_2 + i) - \bar{I}(x_2))}}$$

where the parameters are as shown in Figure 3.

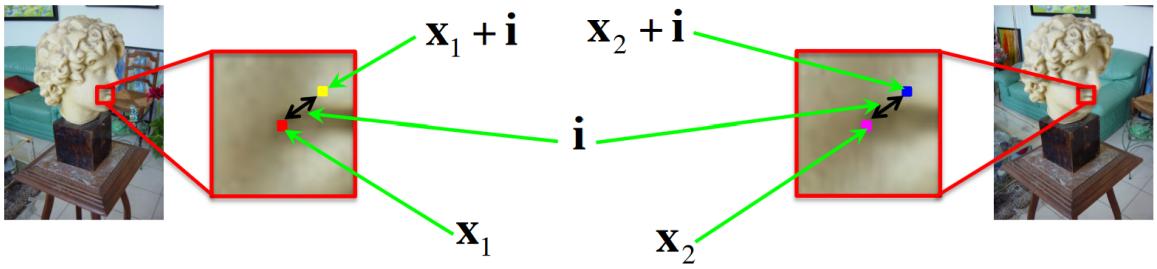


Figure 3: Definition of the ZNCC feature point matching function

The ZNCC function measures the matching degree of a pair of feature points. The better the ZNCC value, the better the matching degree, and thus we are more inclined to select new matching pairs from the vicinity of these feature points.

Based on this idea, we complete Matching Propagation with the following steps:

1. Maintain a priority queue Q;
2. Input all feature point pairs from two images into Q and use their ZNCC function values as the key values for the queue;
3. Pop the feature point pair with the best key value and search for potential matching pairs in the neighboring pixels of the feature point pair, compute their ZNCC values, and add them to Q;
4. Continuously iterate step 3 until the number of point clouds meets our requirements.

Thus, we have completed the 3D dense point cloud reconstruction of the object surface based on multiple images.

5.3.3 Mesh Reconstruction

The process of reconstructing a mesh from a point cloud is a crucial step in computer graphics, computer vision, and 3D modeling. A point cloud is a dataset composed of a large number of discrete points, which typically represent samples of an object's surface. A mesh, on the other hand, is a data structure composed of vertices, edges, and faces that provides geometric and topological information about the object. The process of converting point cloud data into a mesh representation facilitates visualization, simulation, and manufacturing applications. To achieve more suitable modeling geometry, we extend the dense point cloud model into a mesh model. Here are some commonly used methods:

1. Delaunay Triangulation. Delaunay triangulation is a classic method for reconstructing meshes from point clouds. The basic idea is to construct triangles between a set of points such that no point lies inside the circumcircle of any triangle. This method can be extended to three-dimensional space to generate a tetrahedral mesh. Its advantages include high computational efficiency and the ability to generate relatively uniform triangular meshes. Its drawbacks include sensitivity to noise and outliers.

2. Alpha Shapes. Alpha shapes are a generalization of Delaunay triangulation and can handle shape boundaries. The basic idea is to construct all triangles with circumcircle radii smaller than a given parameter α; by adjusting the parameter α, the level of detail in the reconstruction can be controlled. Its advantages include the ability to handle point clouds with complex boundaries. Its drawbacks include the need to select an appropriate parameter α.

3. Poisson Surface Reconstruction. Poisson Surface Reconstruction is an implicit function-based surface reconstruction method. The basic idea is to generate a continuous implicit function from point cloud data, where the zero level set of this function represents the reconstructed surface. By solving the Poisson equation, the reconstructed surface is generated. Its advantages include good robustness to noise and the ability to generate smooth surfaces. Its drawbacks include high computational complexity.

4. Ball Pivoting Algorithm (BPA). The Ball Pivoting Algorithm is a geometry-based surface reconstruction algorithm. The basic idea is to start from a point in the point cloud and roll a ball with a fixed radius on the surface of the point cloud to find adjacent triangles, repeating this process until all points are processed. Its advantages include the ability to generate high-quality triangular meshes. Its drawbacks include the need to choose an appropriate ball radius and requirements for point cloud density.

However, based on practical test results, when the data is not good, the reconstructed 3D dense point cloud often consists of several dispersed point cloud clusters in space. This means that directly using Delaunay triangulation and Marching Cubes methods to construct surfaces between different point cloud clusters can lead to severe distortion. The Poisson Surface Reconstruction method may stitch these point cloud clusters into a continuous surface, potentially causing point displacement; the Ball Pivoting method may fail to reconstruct a complete surface due to point cloud density issues. In comparison, the Alpha Shapes method is more suitable for such scenarios.

6 Special Handling and Optimization of Code

6.1 Interactive Interface

We have implemented an attractive interactive interface, as shown in Figure 4. The point cloud pairing method includes SIFT and SURF; the mesh reconstruction methods include Alpha Shapes, Poisson, and Ball Pivoting; the visualization options determine whether to visualize operations, including feature points and feature point matches; the image folder requires selecting the path to the folder with reconstruction images; the Meshlab path requires selecting the address of the Meshlab.exe file, allowing direct opening of the reconstructed point cloud; the .ply file represents the input for mesh reconstruction and requires a point cloud. Click "Point Cloud Pairing" or "Surface Reconstruction" to perform dense point cloud generation and mesh generation operations. Note that the two tasks are actually separate, with the latter only requiring the specification of the .ply path.

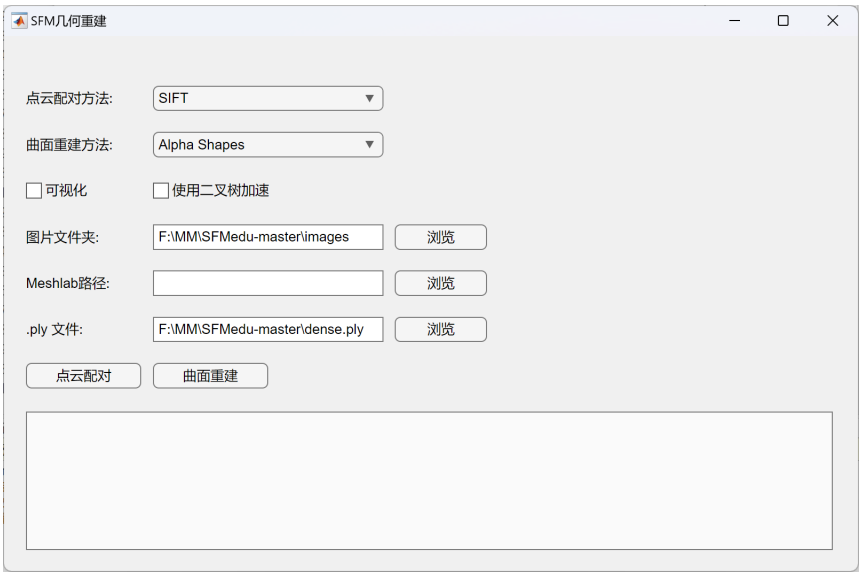


Figure 4: Interactive Interface

6.2 Acceleration of Feature Point Matching

In Section 5.2.1, we mentioned the need to accelerate the efficiency of feature point matching. The following solution has been adopted:

```
function [matchPointsID_i, matchPointsID_j] =
    matchSIFTdesImagesBidirectional(X_i, X_j, distRatio)

if ~exist('distRatio','var')
    distRatio = 0.6^2;
end

X_i = single(X_i);
X_j = single(X_j);
```

```
9
10 kdtree_j = vl_kdtreebuild(X_j);
11 kdtree_i = vl_kdtreebuild(X_i);
12
13 numi = size(X_i,2);
14 matchPointsID_j = zeros(1,numi);
15 for i = 1 : numi
16     [min_idx_j, min_val_j] = vl_kdtreequery(kdtree_j, X_j, X_i(:,i), '
       NumNeighbors', 2);
17     if (min_val_j(1) < distRatio * min_val_j(2))
18
19         [min_idx_i, min_val_i] = vl_kdtreequery(kdtree_i, X_i, X_j(:,
           min_idx_j(1)), 'NumNeighbors', 2);
20         if min_idx_i(1) == i&&  min_val_i(1) < distRatio * min_val_i(2)
21             matchPointsID_j(i) = min_idx_j(1);
22         end
23     end
24 end
25
26 valid = (matchPointsID_j~=0);
27
28 pointsID_i = 1:numi;
29 matchPointsID_i = pointsID_i(valid);
30 matchPointsID_j = matchPointsID_j(valid);
```

This code is used for bidirectional matching of SIFT feature descriptors between two images. It utilizes the k-d tree algorithm from the VLFeat library to accelerate nearest-neighbor searches among feature descriptors, thereby improving the efficiency and accuracy of feature point matching.

6.3 Special Handling to Avoid Iterative Termination Issues

During testing with the Teddy Bear dataset, we encountered two issues. The first issue is that during Newton's method iteration, the Hessian matrix may become nearly singular, leading to calculation errors. To address this issue, we added a regularization parameter to prevent the problem:

```
1 Y = [0;0;0];
2 eprev = inf;
3 lambda = 1e-5;
4 for n = 1:10
5   [e,J] = resid(Y,u,Q);
6   if 1-norm(e)/norm(eprev) < 1000*eps
7     break
8   end
9   eprev = e;
```

```
  JTJ = J' * J;
  dY = (JTJ + lambda * eye(size(JTJ))) \ (J' * e);
  Y = Y - dY;
end
```

Another issue encountered is that when computing the fundamental matrix using the RANSAC method for model parameter estimation, a solution might not be obtained, leading to program termination. To address this, we implemented a detection mechanism that removes the corresponding images from the dataset when such a situation arises. This effectively means that the problematic images are excluded from the initial dataset, thereby avoiding the issue. Due to the extensive code involved, it will not be displayed here.

6.4 Optimization Methods for Large-Scale Computations in Complex Loops

In the implementation of this project, feature point matching was the most time-consuming operation. Initially, we used the following code structure:

```
local_heap = [];
for yy0 = yMin0:yMax0
   for xx0 = xMin0:xMax0
      if match_im_i(xx0, yy0, 1) == -1
         xx = (xx0 + x1) - x0;
         yy = (yy0 + y1) - y0;
         for yy1 = max(yMin1, yy - 1):min(yMax1, yy + 2)
            for xx1 = max(xMin1, xx - 1):min(xMax1, xx + 2)
               if match_im_j(xx1, yy1, 1) == -1
                  AuxCost = sum(zncc_i(xx0, yy0, :) .* zncc_j(xx1, yy1,
                      :));
                  if 1 - AuxCost <= CostMax
                     local_heap(end+1, :) = [xx0, yy0, xx1, yy1,
                        AuxCost];
                  end
               end
            end
         end
      end
   end
end
```

It can be observed that each step of the loop is independent. However, due to the way elements are added, conventional parallel processing methods were not feasible. Additionally, practical evaluations revealed that the primary time-consuming task was the computation of 'AuxCost'. To address this, we leveraged MATLAB's efficiency with vectorized data computation and its unique array transformation capabilities to optimize the process. As a result, the time

required for feature point matching was reduced to less than half of the original duration. The specific adjustments made are as follows:

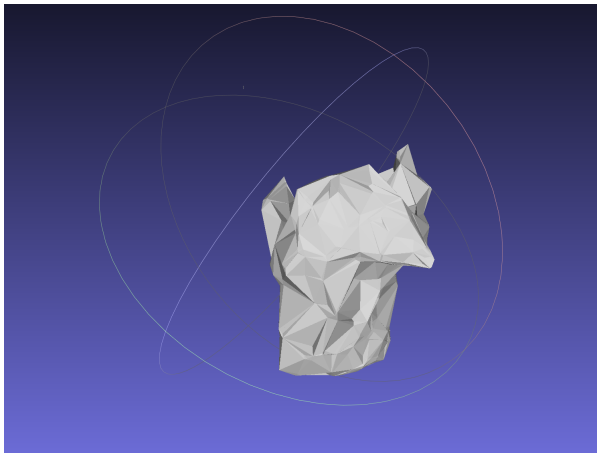
```
xx0_values = [];
yy0_values = [];
xx1_values = [];
yy1_values = [];

for yy0 = yMin0:yMax0
   for xx0 = xMin0:xMax0
      if match_im_i(xx0, yy0, 1) == -1
         xx = (xx0 + x1) - x0;
         yy = (yy0 + y1) - y0;
         for yy1 = max(yMin1, yy - 1):min(yMax1, yy + 2)
            for xx1 = max(xMin1, xx - 1):min(xMax1, xx + 2)
               if match_im_j(xx1, yy1, 1) == -1
                  xx0_values(end+1) = xx0;
                  yy0_values(end+1) = yy0;
                  xx1_values(end+1) = xx1;
                  yy1_values(end+1) = yy1;
               end
            end
         end
      end
   end
end
AuxCost = arrayfun(@(i) sum(zncc_i(xx0_values(i), yy0_values(i), :) .*
   zncc_j(xx1_values(i), yy1_values(i), :)), 1:length(xx0_values));
valid_cost = 1 - AuxCost <= CostMax;
local_heap = [xx0_values(valid_cost);yy0_values(valid_cost);xx1_values(
   valid_cost);yy1_values(valid_cost);AuxCost(valid_cost)]';
```

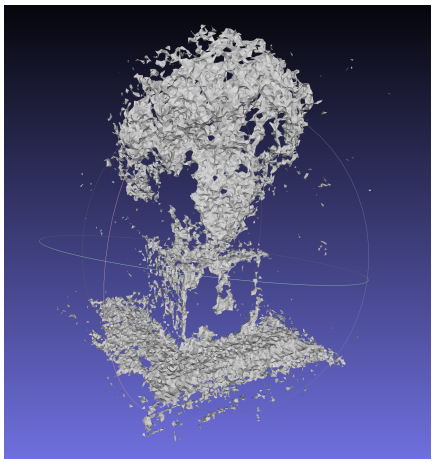
This way, the calculation of AuxCost is transformed into a vectorized computation through logical arrays.

6.5 Adaptive Parameters for Alpha Shapes

We previously mentioned the drawbacks of the Alpha Shapes method. To illustrate this more clearly, we show the results with different parameters on the David Simple dataset:



(a) Severe distortion with large parameters



(b) Smaller parameters are closer to the actual result

Figure 5: Impact of parameters on results

We aim to set parameters that yield results close to the right image, which requires choosing a suitable α. The value of α should be as large as possible, but still greater than the maximum distance between adjacent points within the same point cloud cluster and less than the minimum distance between different point cloud clusters. Achieving this directly is challenging, so we can only approximate an α that is as close as possible.
We estimate α using the following function:

```
function optimalAlpha = estimateOptimalAlpha(points,model)
    numPoints = size(points, 1);
    k = 6;
    switch model
        case 'randomPicking'
            testSize=4;
            subsetSize = min(2500, numPoints);
            temp=zeros(1,testSize);
            for i=1:testSize
            subsetIndices = randperm(numPoints, subsetSize);
            subsetPoints = points(subsetIndices, :);
            [~, D] = knnsearch(subsetPoints, subsetPoints, 'K', k);
            distances = D(:, 2:end);
            temp(i) = median(distances(:));
            end
            distance=median(temp);
            alphaFactor = 1.0;
        case 'allPicking'
            [~, D] = knnsearch(points, points, 'K', k);
            distances = D(:, 2:end);
            distance = median(distances(:));
            alphaFactor = 2.3;
    end
```

```
    optimalAlpha = alphaFactor*distance;
end
```

Initially, we considered allowing users to choose between two calculation methods. Later, we found that if all points are sampled to calculate distances, the optimal position for α is completely uncertain. Using the median directly requires repeatedly adjusting the scaling factor, which heavily depends on the tested point cloud data, leading us to eventually discard this approach.

Here, we provide a detailed explanation of the final random sampling method used. We consider randomly selecting several groups of points from the obtained point cloud data, calculating the distances between these points and their nearest neighbors, and using the average value as the basis to determine α. This method effectively avoids the mutual influence of different point cloud clusters, reduces computational complexity, and improves speed.

Despite these adjustments, the final reconstructed surface is not ideal; it only reflects the general shape of the surface, with sharp and protruding details, making the surface less smooth and even.

7 Model Results

We consider this problem to be a complex 3D reconstruction issue. Therefore, in addition to implementing the algorithm, we conducted a more in-depth study of 3D reconstruction, and the results are presented in this section.

7.1 Basic Model Results

We used MATLAB to reproduce the algorithms from Sections 5 and 6. The source code is available in the folder SFM_MVS.

7.1.1 SIFT and SURF Feature Point Selection and Matching

We first tested using the basic David_simple dataset, which includes only five images of the David sculpture.

The feature point selection results for SIFT and SURF are shown in Figure 6.

We can see that SURF selects significantly more feature points than SIFT. This discrepancy might be related to our implementation using the SIFT feature point selection from another library, while directly calling MATLAB's SURF function.

7.1.2 Sparse Point Cloud Reconstruction

We standardized the file structure for result presentation as shown in Figure 7.

(a) SIFT Feature Point Selection

(b) SURF Feature Point Selection

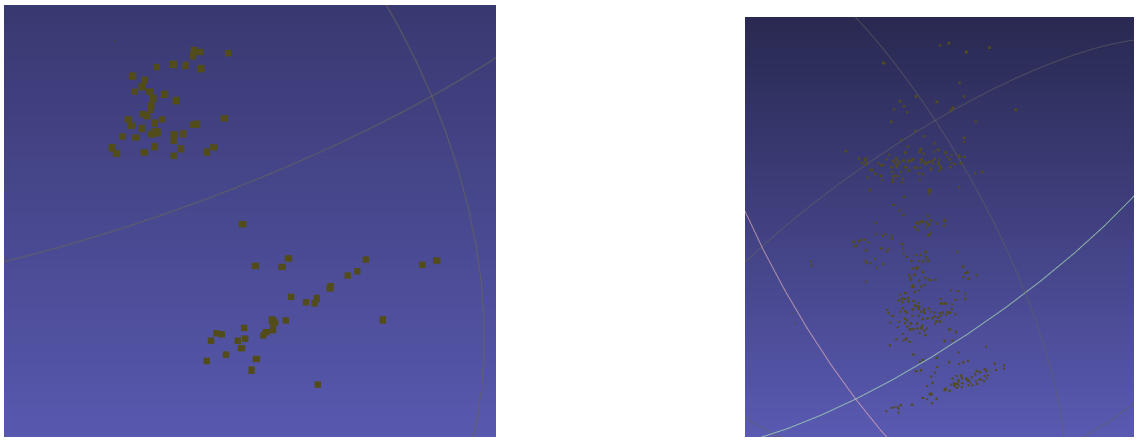
Figure 6: SIFT and SURF Feature Point Selection

Figure 7: Result Presentation Structure

After clicking on the folder named with the data, the first entry indicates the method used. In the SIFT and SURF architecture, alphaShapes and Poisson store the meshes, dense stores the dense point clouds, sparse stores the sparse point clouds, and pointcloud3d shows the visualization results of the dense point clouds.

The results of sparse point cloud reconstruction using SIFT and SURF are shown in Figure 8.

We can see that the SURF method results in more feature points, more matched feature points, and consequently more sparse point clouds. However, this is not necessarily beneficial, as selecting too many points from the environment can lead to higher noise in the final result.



(a) Sparse Point Cloud Reconstruction using SIFT

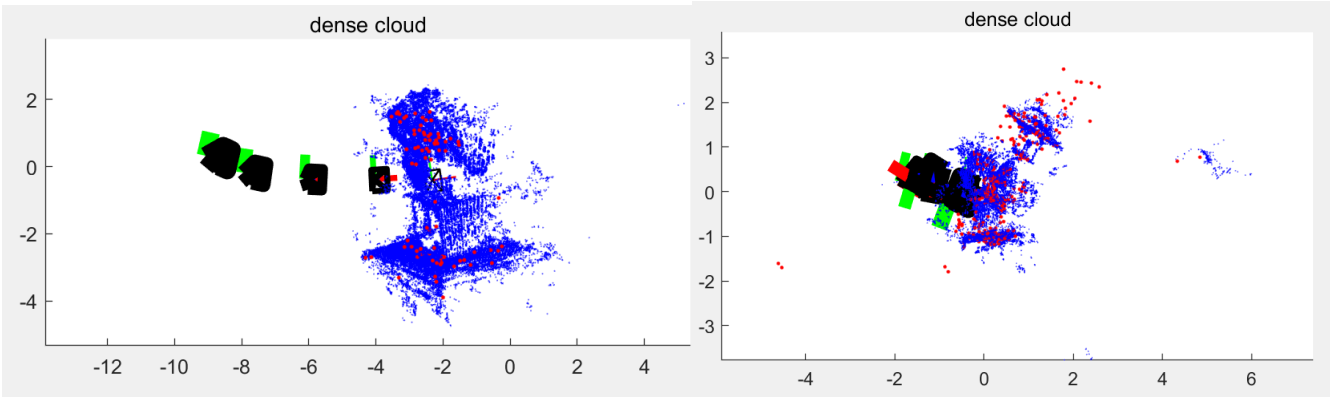
(b) Sparse Point Cloud Reconstruction using SURF

Figure 8: Sparse Point Cloud Reconstruction using SIFT and SURF

On the other hand, it is evident that sparse point clouds cannot fully represent the geometry of the object. It is difficult to extract geometric information about the object from sparse point clouds, which further emphasizes the necessity of dense point cloud reconstruction.

7.1.3 Dense Point Cloud Reconstruction

The results of dense point cloud reconstruction and camera visualization using SIFT and SURF are shown in Figure 9.



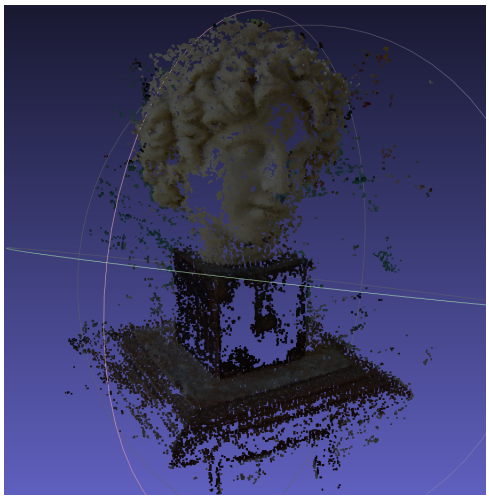
(a) Dense Point Cloud and Camera Visualization using SIFT

(b) Dense Point Cloud and Camera Visualization using SURF

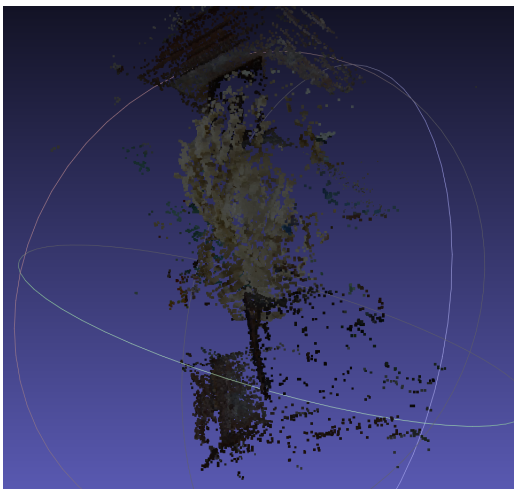
Figure 9: Dense Point Cloud and Camera Visualization using SIFT and SURF

Once again, we can see that the camera reconstruction results should ideally show a ”rotating around” effect around the object, consistent with our assumption in Section 3. It is also evident that the environment in SURF indeed negatively impacts the reconstruction, with the point clouds being noticeably less clear compared to the SIFT method.

The dense point cloud reconstruction results are shown in Figure 10.



(a) Dense Point Cloud Reconstruction using SIFT



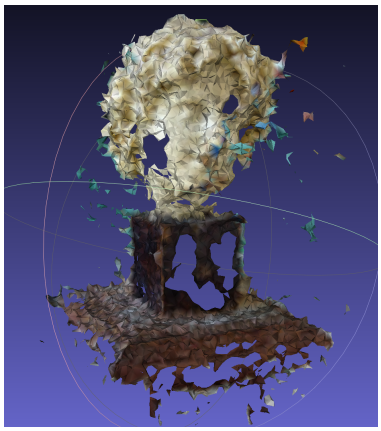
(b) Dense Point Cloud Reconstruction using SURF

Figure 10: Dense Point Cloud Reconstruction using SIFT and SURF

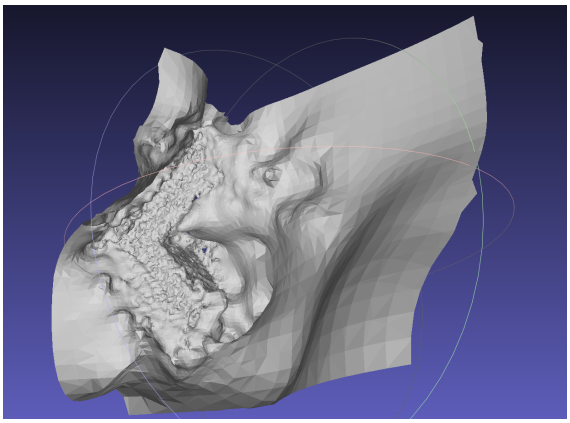
It can be observed that the reconstruction result using SIFT is good, whereas SURF shows many flip errors, possibly due to incorrect point matches.

7.1.4 Mesh Reconstruction

In Section 7.1.3, we compared the effects of SIFT and SURF feature matching on dense point clouds. The results with SIFT were better, so we present the mesh reconstruction results using SIFT in Figure 11.



(a) Alpha Shapes Mesh Reconstruction



(b) Poisson Mesh Reconstruction

Figure 11: Alpha Shapes and Poisson Mesh Reconstruction

We find that the result obtained using the Alpha Shapes method is quite good, while the result

using the Poisson surface reconstruction is poor, with little discernible geometry. This may be related to the high noise in the 3D reconstruction. The Poisson method uses implicit function representation to solve differential equations, and excessive noise may make it difficult to distinguish the main reconstruction target from the environment, leading to poor results.

7.1.5 Additional Tests and Results

We also tested different datasets, and the results are presented below. The SIFT method and Alpha Shapes mesh reconstruction were used.

Firstly, we tested the David dataset, which contains all the images of the David sculpture from the basic test, stored in the David folder.

Firstly, the results of the sparse point cloud reconstruction are shown in Figure 12.

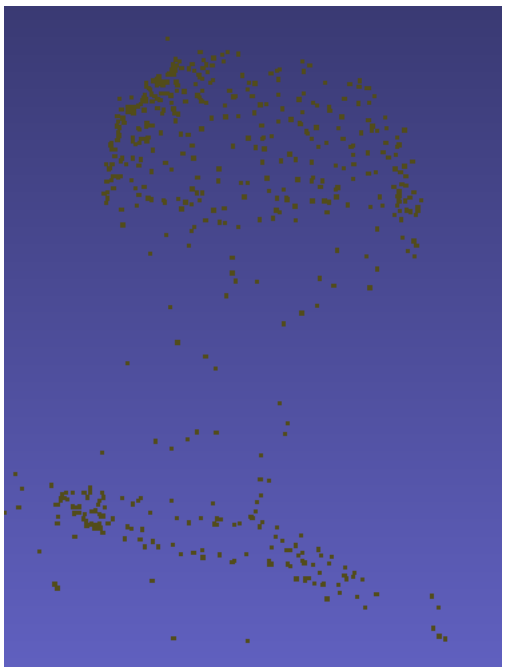


Figure 12: Sparse Point Cloud Reconstruction Results for David Dataset

It can be seen that sparse point clouds indeed cannot represent the overall geometry, but the general contour of the 3D object can be roughly perceived from the complete data.

Next, the results of dense point cloud and camera visualization are shown in Figure 13.

From the distribution of the point clouds, it meets expectations, although there is also considerable noise.

Next are the results of dense point cloud reconstruction, as shown in Figure 14.

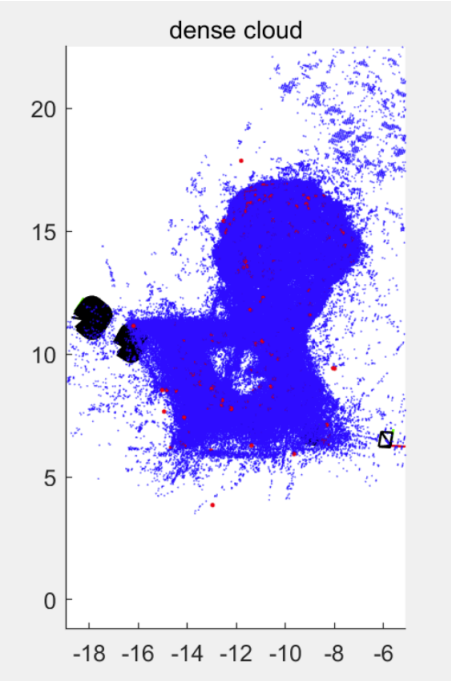
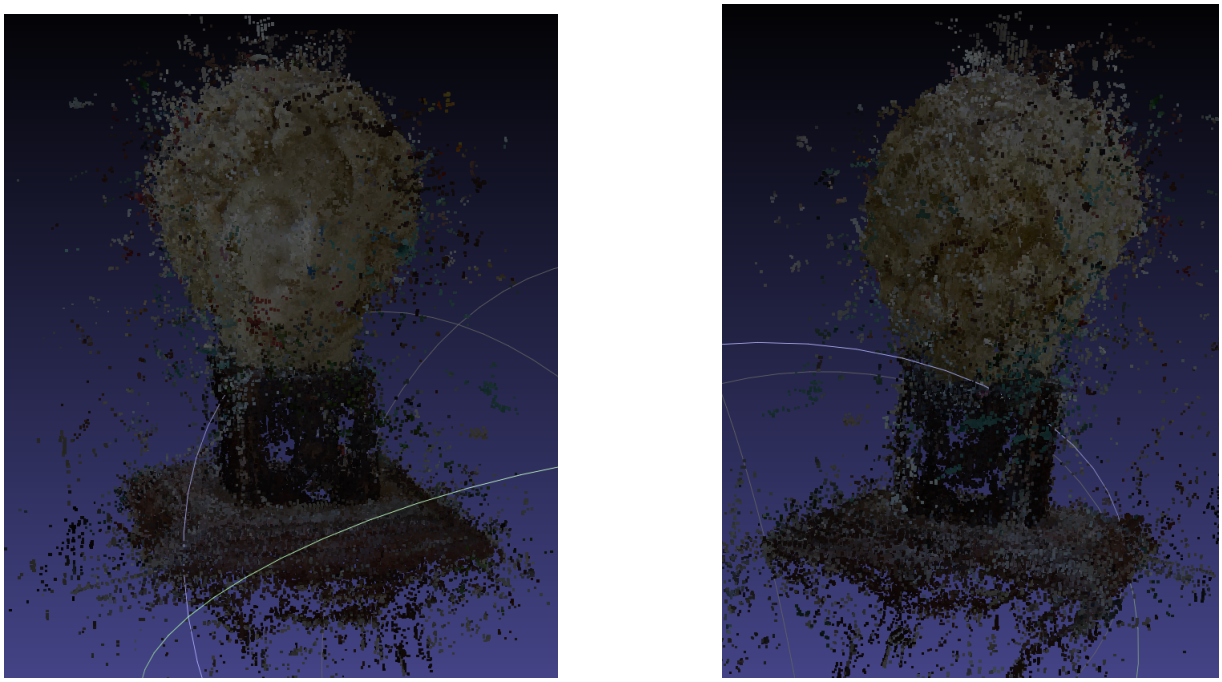


Figure 13: Dense Point Cloud and Camera Visualization Results for David Dataset



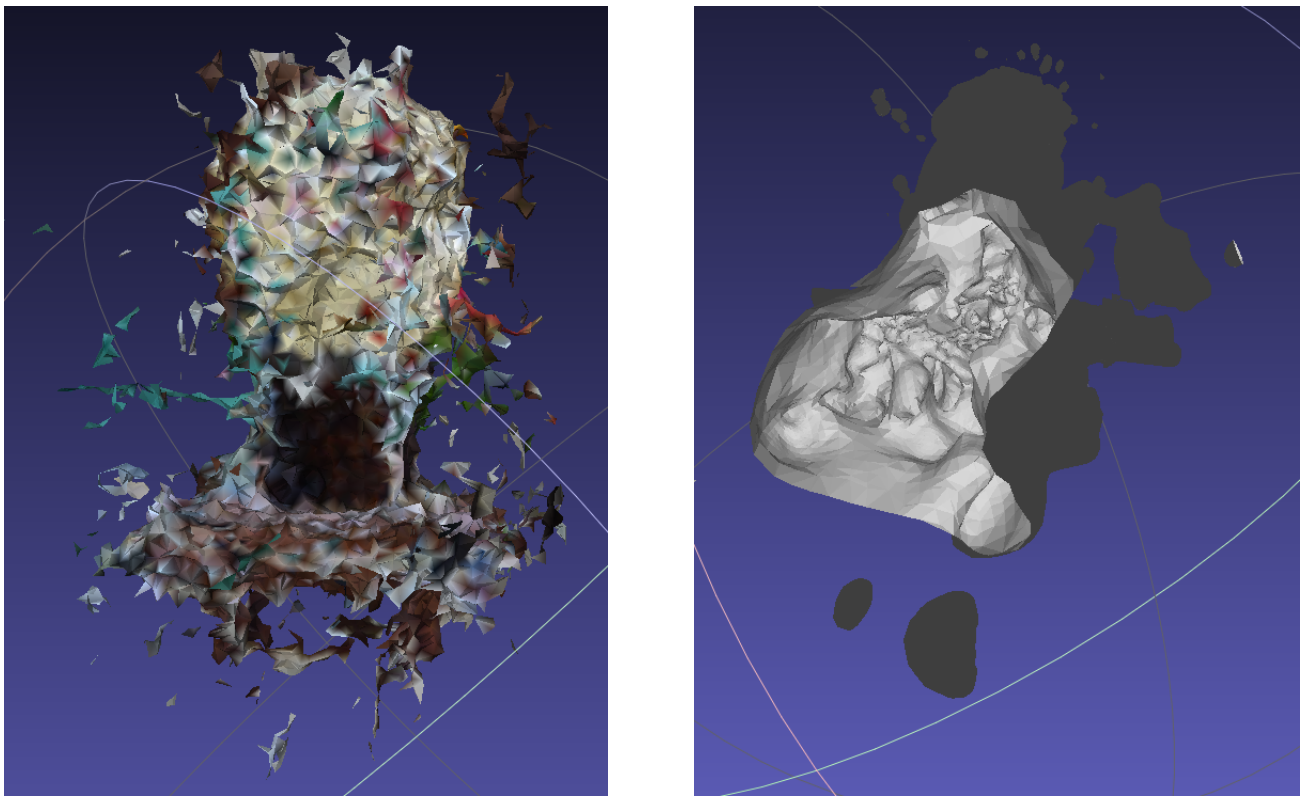
(a) Front View

(b) Side View

Figure 14: Dense Point Cloud Reconstruction Results for David Dataset

We find that the overall geometry of the reconstruction is complete, achieving a satisfactory result.

Finally, we present the mesh reconstruction results. Unfortunately, the noise had a significant impact on the mesh reconstruction, and the results were not as good as expected, as shown in Figure 15.



(a) Alpha Shapes Mesh Reconstruction

(b) Poisson Mesh Reconstruction

Figure 15: Alpha Shapes and Poisson Mesh Reconstruction for David Dataset

7.2 Colmap

During our 3D reconstruction tasks, we discovered Colmap. Colmap is an open-source, cross-platform photogrammetry software for sparse and dense 3D reconstruction. It can reconstruct the 3D structure of a scene from a set of images and supports various camera models and lens distortion corrections. Colmap consists of two main parts: a set of command-line programs for executing different stages of the reconstruction process and an optional graphical user interface for viewing and interactively editing the reconstruction results. This open-source library is continuously updated, and its sparse, dense, and mesh reconstruction methods are quite similar to those used in our tasks, so we also used this library for testing. Its algorithmic logic is similar to our approach, involving classic SFM and MVS.

We first tested with the David_simple dataset. The storage format of Colmap' s computation results is shown in Figure 16.

Here, "sparse" represents the sparse point cloud model, which is a specific storage format used

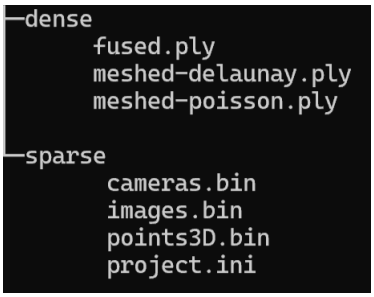


Figure 16: Colmap Result Structure

by Colmap. For details, please refer to the Colmap official documentation https://colmap.github.io/format.l
"Fused" indicates the dense point cloud reconstruction model, "meshed-delauney" represents Delaunay mesh reconstruction, and "poisson" represents Poisson mesh reconstruction.

The dense point cloud reconstruction results are shown in Figure 17.

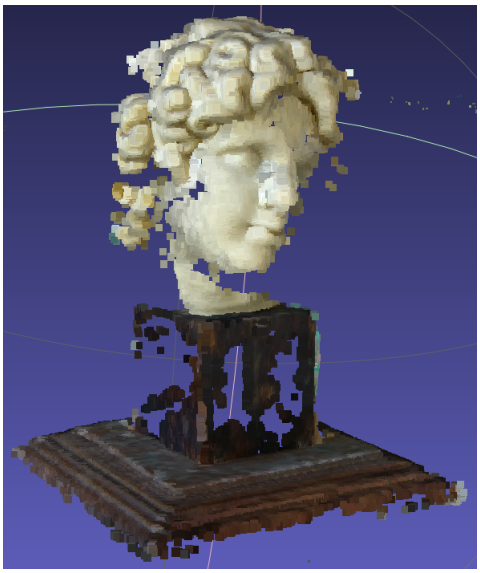


Figure 17: Colmap Basic Test Dense Point Cloud Reconstruction Results

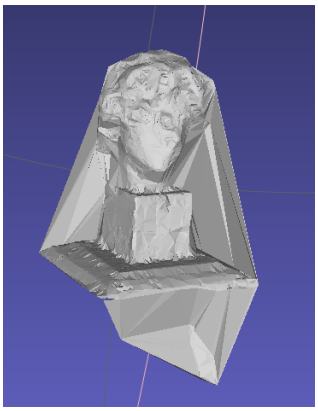
Compared to our results, the open-source Colmap results are also quite good. The results for the two types of mesh reconstruction are shown in Figure 18.

We can see that the Poisson mesh reconstruction method is quite realistic, while the Delaunay mesh reconstruction result is sensitive to noise and outliers, with environmental points causing excess surfaces in the reconstruction.

We also tested the remaining data and present the results as follows:

For the David dataset, our results are shown in Figure 19 and Figure 20.

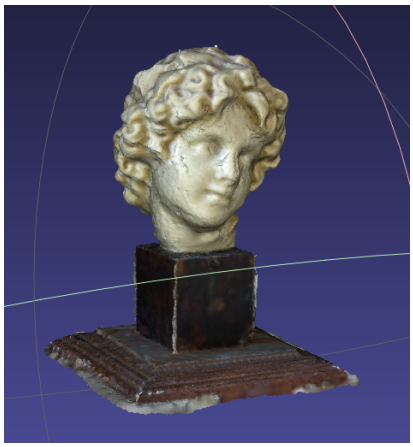
(a) Poisson Mesh Reconstruction



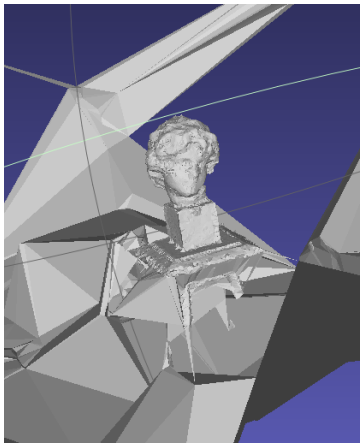
(b) Delaunay Mesh Reconstruction

Figure 18: Colmap Basic Data Mesh Reconstruction

Figure 19: Colmap Dense Point Cloud Reconstruction Results for David Dataset



(a) Poisson Mesh Reconstruction

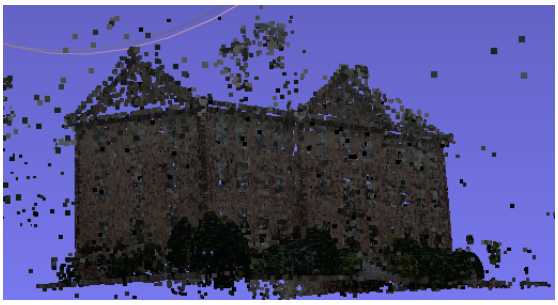


(b) Delaunay Mesh Reconstruction

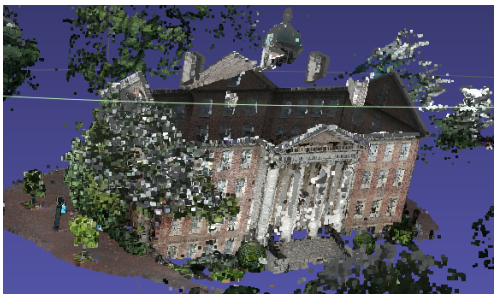
Figure 20: Colmap Mesh Reconstruction for David Dataset

From Figure 20b, it is evident that the Delaunay method is highly affected by the environment, with noise introducing a significant amount of extra mesh.

Next, we tested the Building dataset, which has a large number of images and a relatively slow processing time. Our results are shown in Figures 21 and 22. We also provide the sparse point cloud results here.

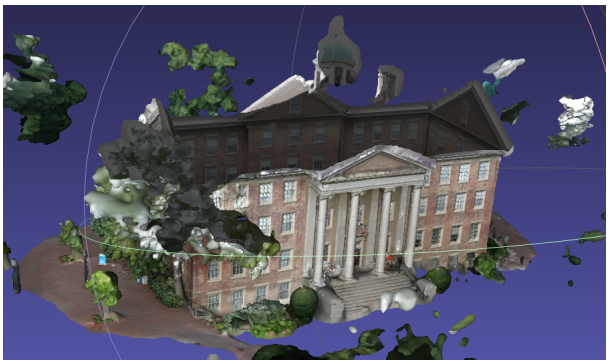


(a) Sparse Reconstruction

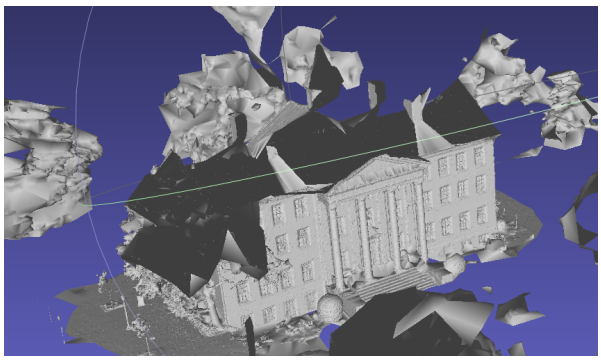


(b) Dense Reconstruction

Figure 21: Colmap Point Cloud Reconstruction for Building Dataset



(a) Poisson Mesh Reconstruction



(b) Delaunay Mesh Reconstruction

Figure 22: Colmap Mesh Reconstruction for Building Dataset

In fact, we also tested the MATLAB framework on this set of data, but the processing time was too long. It is evident that the Colmap framework has advantages, as it integrates many aspects of 3D reconstruction.

7.3 Context Capture

Context Capture is a 3D modeling and photogrammetry software developed by Bentley Systems. It allows users to create high-precision 3D models from a set of 2D images, which can come from drones, ground photography, or traditional aerial photography. Context Capture can handle various types of image data and uses computer vision and photogrammetry techniques to reconstruct the 3D geometry of scenes.

In fact, this is an industrial-grade 3D reconstruction software that reconstructs 3D geometry from multi-angle 2D images. We utilized drone aerial data from the East Campus of the University of Science and Technology for 3D mesh reconstruction. The results are shown in Figures 23 and 24. This is the modeling of the East Campus Hospital, and the reconstruction effect is very realistic.

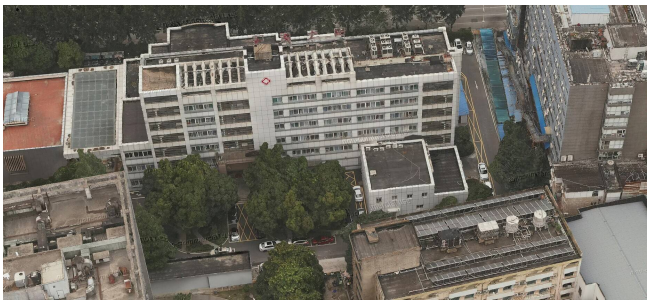


Figure 23: Context Capture Reconstruction Results for the University Campus (1)

Figure 24: Context Capture Reconstruction Results for the University Campus (2)

7.4 Neus

The methods discussed above utilize traditional computer vision techniques, mostly based on object geometry. Recently, with the emergence of NeRF and related work, there is a new rendering perspective on 3D reconstruction. Neus is one of the representative works in this area.

Neus uses a novel volumetric rendering approach to render images generated from implicit SDF and minimize the difference between the rendered images and input images. [2] In the

given examples, artifacts from Colmap or NeRF are largely avoided. The essence of the method is to train color and position functions using a multi-layer perceptron, render images using a camera, and optimize the results to minimize the discrepancy with known images, focusing on color error. Therefore, in addition to 2D images, Neus requires camera extrinsic parameters for the dataset.

According to our proposed model and algorithm, if multi-view images are known, camera extrinsic parameters can be obtained, and thus Neus can also achieve 3D reconstruction tasks for given images.

We tested the example data from Neus and obtained very good results. The file structure of the Neus method is shown in Figure 25.

Figure 25: Neus Result File Structure

In this structure, ‘mesh.ply‘ is the final generated mesh format, and ‘video‘ is the video rendering. For the Bear dataset, our test results are shown in Figure 26.

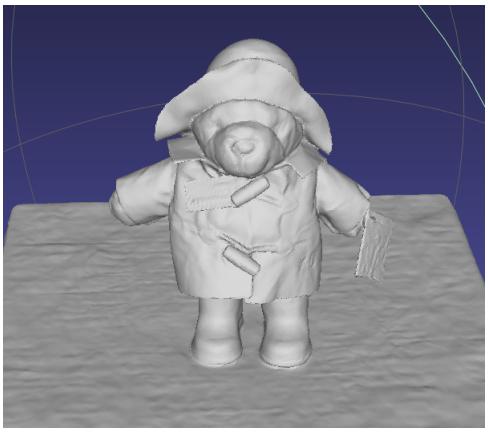


Figure 26: Neus Mesh Reconstruction for Bear Dataset

This process took about 10 hours on a server with a 4090 GPU, which demonstrates the substantial computational load of the Neus method. Indeed, this is due to the rendering. We found that the mesh reconstruction effect is very good.

We also tested Neus on the David dataset. However, it was first necessary to calculate the extrinsic parameters from the 2D images. We used Colmap and the LLFF library to achieve this. The final results are shown in Figure 27.

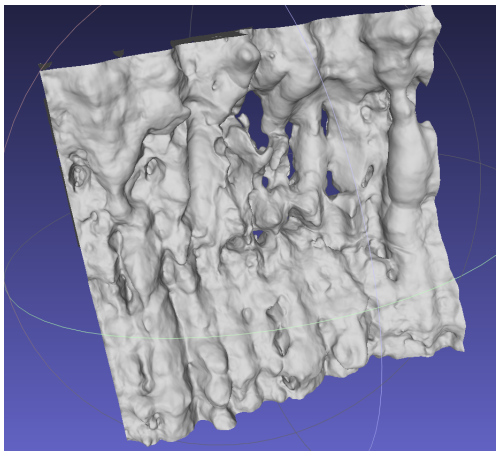


Figure 27: Neus Mesh Reconstruction for David Dataset

We found it difficult to identify any information related to David, and the video also made it hard to discern relevant information. We suspect that the errors in camera extrinsic parameters may have caused significant deviations in the trained model due to inaccuracies in image capture.

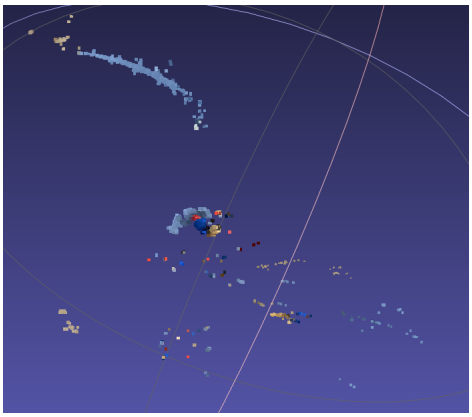
In addition, we tested the Clock data provided by Neus, and the test results are shown in Figure 28.

(a) Main View (b) Another Angle

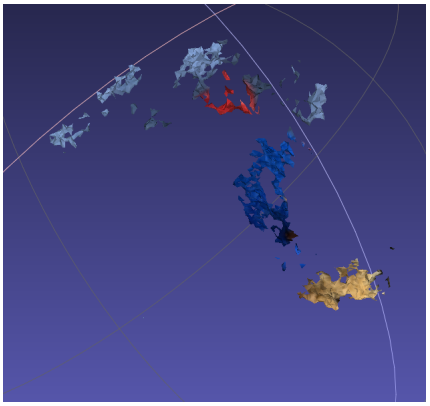
Figure 28: Neus Mesh Reconstruction for Clock Dataset

We were amazed by the results of this work for 3D reconstruction, as it achieved such well-constructed meshes.

On the other hand, we tested our model on the Bear dataset, and the results are shown in Figures 29.



(a) Dense Point Cloud



(b) Poisson Mesh Reconstruction

Figure 29: Testing Results of Our Model on the Bear Dataset

We found that our model only extracted part of the point cloud. This is because the dataset used by Neus has known camera parameters and does not require strict sequence constraints between images. However, our model requires that adjacent images do not change too much due to the need for matching feature points.

8 Further Discussion

8.1 Improvements to the RANSAC Algorithm

When using the RANSAC algorithm to estimate the fundamental matrix $\boldsymbol{F}$ between two images, we can also improve the 8-point method by using the 7-point or 5-point methods.

To solve the linear system, selecting 8 pairs of matching feature points uniquely determines the fundamental matrix. However, since these feature point pairs do not necessarily correspond exactly to the 3D coordinates, the estimated fundamental matrix may not match the true geometric information. If we do not retain some degrees of freedom in the solution, the fundamental matrix obtained from less well-matched feature points may not be ideal. To reduce the impact of noise on the fundamental matrix estimation, we can estimate the matrix using only 5 or 7 pairs of matching feature points each time, returning a set of fundamental matrices that satisfy the current constraints. We then select the one with the most inliers as the final estimate, which can yield better results and improve the robustness of the program.

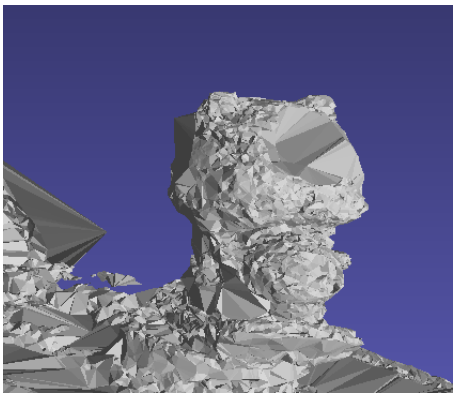
8.2 Data Preprocessing

We discovered a peculiar issue during data testing. For our own captured data, both our model and the integrated Colmap struggled to match feature points effectively. Therefore, we adjusted the data generation method from capturing still images to recording a video around the object and then extracting frames using code to ensure data continuity. The source code can be found in the ‘dataprocessing‘ folder, and requires the OpenCV library, which can be installed via pip.

We recorded a 360-degree video of a ”Bear” figurine, located in the ‘dataprocessing‘ folder, and the resulting images are in the ‘data‘ folder’s 12 subfolders. We tested this data with Colmap, and the results are shown in Figures 30 and 31.

Figure 30: Colmap Point Cloud Reconstruction for Bear Data

(a) Poisson Mesh Reconstruction



(b) Delaunay Mesh Reconstruction

Figure 31: Colmap Mesh Reconstruction for Bear Data

8.3 Further Acceleration Ideas

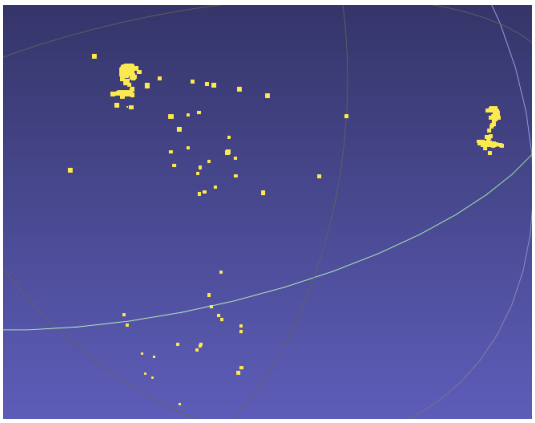
Reviewing the testing results, we found that the most time-consuming stages of the entire program are Merging Graphs and Dense Matching. It is noted that the latter does not cause data interference during computation and can be accelerated using parallel methods. For the Merging Graphs process, we conducted a detailed investigation and discovered that we were performing this process by traversing the dataset frame by frame. This approach works fine for smaller datasets, but as the dataset grows, the size of the Graph to be merged with the next frame becomes larger, and the merging time increases significantly, eventually causing the entire process to take several hours. We found this to be quite frustrating.

On Sunday, we suddenly thought of using binary trees to complete this process. According to the nature of binary trees, we can set the Graphs to be merged as sub-nodes of a certain node, recursively completing the previously frame-by-frame merging process. This reduces the number of merges significantly, and the total size of the objects to be merged is smaller than before, theoretically greatly reducing the time complexity. Moreover, by setting each frame in the dataset as a leaf node, we ensure that merging occurs between adjacent frames, which guarantees the reliability of the results.

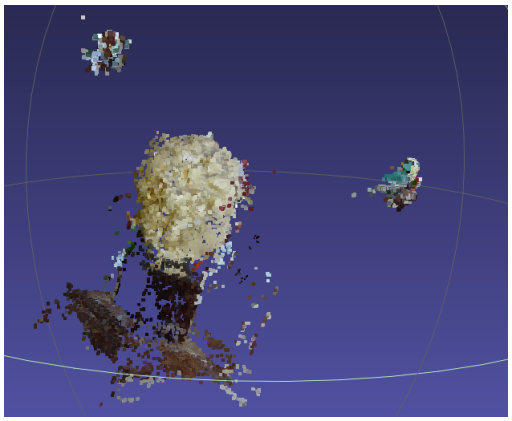
However, unfortunately, we encountered some issues during testing, primarily due to dimension mismatches in vector values. These problems were not encountered in the original processing because, due to time constraints, we modified the function internally based on data characteristics without thoroughly checking the function's internal and external logic, which may have impacted the results.

The results of simplifying the frame-by-frame merging process using the binary tree method on the David dataset are shown in Figures 32.

We noticed that the sparse and dense point clouds exhibited a "chunking" phenomenon, which was not desired. We suspect this might be due to the binary tree disrupting the sequence relationship of the image series, and the "chunking" effect occurring when integrating



(a) Sparse Point Cloud



(b) Dense Point Cloud

Figure 32: Results of Accelerating Frame-by-Frame Point Cloud Merging Using Binary Trees

information from two images with many points. This further underscores the importance of image sequence relationships in acceleration attempts.

Since this processing method is much faster than the original (for example, the complete David dataset was processed in only 10 minutes compared to several hours before), we might consider adjusting the image arrangement in the dataset to better support this merging method, which should yield good results.

9 Model Evaluation, Improvement, and Promotion

In this paper, we implemented a 3D reconstruction algorithm based on 2D images and compared it with existing results in the same direction. We have accelerated the model's running speed and enhanced its capability to better capture surface geometry. We reproduced classic camera calibration and multi-view reconstruction models, achieving good results on some datasets. Initially, our model heavily depended on data, but in Section 8, we proposed a method to generate good data. However, there are still areas for further optimization in the model:

1. Our model's camera modeling is overly idealized. In reality, the camera is a complex optical model, and we simplified it to a pinhole imaging model, neglecting possible distortions. This could affect the camera's intrinsic parameters and alter the matrix $\boldsymbol{K}$.

2. Our model uses optimization techniques frequently, particularly nonlinear optimization. There might be more stable numerical solutions for specific nonlinear optimization problems, which we have not explored.

3. Our model primarily uses library methods for mesh reconstruction, but the results have not been ideal, possibly due to excessive environmental noise. Noise reduction on the dense point cloud model before mesh reconstruction could be explored.

10 Acknowledgments

We would like to thank Professor Zhang Juyong for providing the aerial dataset of the USTC East Campus and the USTC3DV lab for the server. Some of the code operations involved are highly intensive and took a considerable amount of time on our own laptops.

References

[1] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[2] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021.