Synthesis Project: Digitizing Real-World Scenes from Images

Final Report

Takis Arapakis Natasja van Heerden
Guillermo Rodriguez-Mon Barrera
Qu Wang and Xin Wang

Synthesis Project: Digitizing Real-World Scenes from Images

Final Report

by

Takis Arapakis Natasja van Heerden
Guillermo Rodriguez-Mon Barrera
Qu Wang and Xin Wang

TU Delft

Preface

The geomatics synthesis project is a first year project that is carried out by teams of five students and combines the geomatics background with practical work. The existence of an involved external stakeholder may define the aims of the research and provide the appropriate equipment, but all the problems have to be answered by the team. The final project will be presented at the 2018 Geomatics day which subject is innovative application of Geomatics.

This report is written by Digitizing real-world scenes from Images team and presents the process of creating a handy software that helps in geomatics applications. The software will extract points clouds and 3D models of various scenes from a sequence of overlapping images. Softwares that enable easy and fast acquisition can be very useful tools in geomatics engineers' hands. In the following pages we will examine the sequence of algorithms required for that transformation, as well as the relative coding environment. The main challenges the team had to overcome was the familiarization with the new classes and libraries and the development of a solid cooperation that will enable the integration of the individual parts to the final project.

In this process we were guided by our teachers Liangliang Nan and Ravi Peters. We would like to thank them for their help and guidance during this project.

Takis Arapakis Natasja van Heerden
Guillermo Rodriguez-Mon Barrera
Qu Wang and Xin Wang
Delft, June 2018

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

GUI	Graphical User Interface
MVS	Multi-View Stereo
PRM	Project Risk Management
SfM	Structure from Motion
SIFT	Scale-invariant Feature Transform
UI	User Interface

Executive Summary

3D computer models are starting to play a more and more important role in our society. Real-world situations are often too complex to explain in a 2D map and also the interest in virtual reality, serious gaming and other technologies that can be based on 3D computer models, is growing.

The people of the Floriade project in Almere are well aware of this trend. The Floriade project is the development of the Floriade world expo that will be held in Almere in 2022. Their vision is growing green cities. Many innovative sustainable techniques and solutions will be displayed during the expo.

In order to improve the experience for visitors of the Floriade expo and reach people that are not able to visit the expo, they want to create a lightweight 3D computer model of the project, that everyone can use to create applications about the Floriade expo. This way applications can be created that are as innovative as the project itself. For example mobile applications for information and routing, but also for virtual reality and other innovative technologies.

To be able to capture the real-world scene and make it into a 3D computer model, a software is needed. Our goal was to create an open source software with an intuitive user interface, that takes 2D images of a building object and creates a 3D solid computer model. To do this we had to familiarize ourselves with the C++ programming language and with the algorithms and libraries used for point cloud and mesh generation. We also compared our results with existing similar software and tested our software for different cases.

The tools that we used to develop the software are the C++ programming language and Qt Creator and Microsoft Visual Studio for software design. The images of the building objects are taken either with hand-held cameras or with an Unmanned Aircraft System (UAS, drone) by us or the Floriade team in Almere.

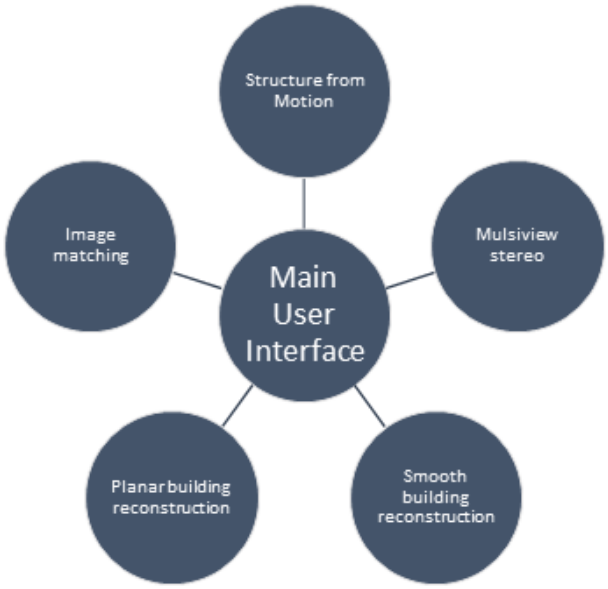


Figure 1: Software concept

The basic concept of the software design is that of a modular process, where each part of the software is developed separately. Each module will be connected and integrated in the user interface, which is the skeleton of the software (figure 1). The inputs and outputs of each module are predefined standard formats. Intermediate results can be saved and viewed in the program. This way the reconstruction doesn't always have to start from the beginning.

The user interface should provide an efficient and smart way to allow interaction between the user and the program. The interface should be well structured, simple, intuitive, keep users informed and prevent errors. The program has a main window that holds the functionality and information of the program and it has a viewer window where the 3D pointcloud and mesh can be viewed.

For creating the modules of the software we've used different algorithms and libraries. To create the modules we used the following algorithms:

- **Image matching:** Scale-Invariant Feature Transform (SIFT)
  Images that are taken with either a handheld camera or with a UAS (drone) will be matched via this algorithm. The result is matching key points between every two images.
- **Sparse pointcloud creation:** Structure from Motion (SfM)
  With the output of the image matching a 3D sparse point cloud can be created. Pose estimates for registered images and a set of 3D points which represent the scene structure are generated as outputs.
- **Dense pointcloud creation:** Multi-View Stereo (MVS)
  From the sparse point cloud, a dense point cloud is created. The output is in ply format and has list of oriented vertices with properties x,y,z, the normals nx,ny,nz and the colours in red, green, blue.
- **Smooth surface reconstruction:** Poisson Reconstruction
  With this algorithm a smooth triangulated mesh is created from a dense pointcloud. This reconstruction method is especially helpful when it comes to curved building objects.
- **Piece-wise planar surface reconstruction:** PolyFit
  This algorithm creates a lightweight planar surface model from a dense pointcloud. This algorithm works well for many buildings, since buildings often exist of straight facades and roofs.

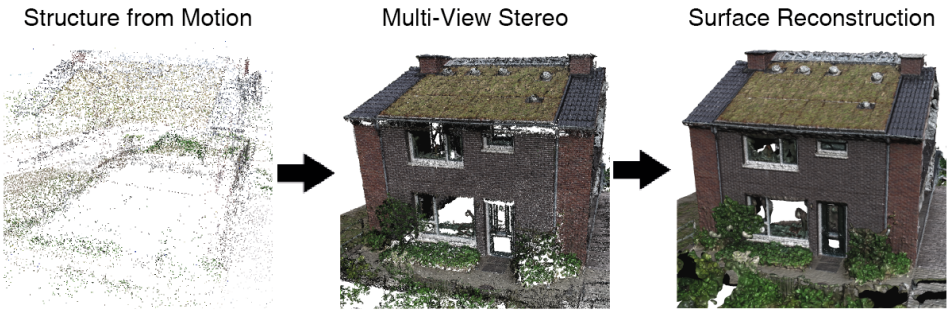


Figure 2: Sparse pointcloud, dense pointcloud and mesh (poisson)

When testing our software for different cases, we find out that it works rather well for materials like brick and wood. Materials without any distinctive texture like glass or trespa get ignored almost completely by the reconstruction software.

We compared our software to other similar software to be able to assess our work. When comparing our software with the commercial software Pix4D, we find out it gives some promising results. One of the cases we compared is the Almere watch tower. The results of our software actually look better, because it does not have the wrong projection of the tower (figue 3). This is due to the lack of horizontal overlap in the datasets of images taking by the UAS

(drone). Our software dropped the images with not enough overlap, so they did not influence the reconstruction.

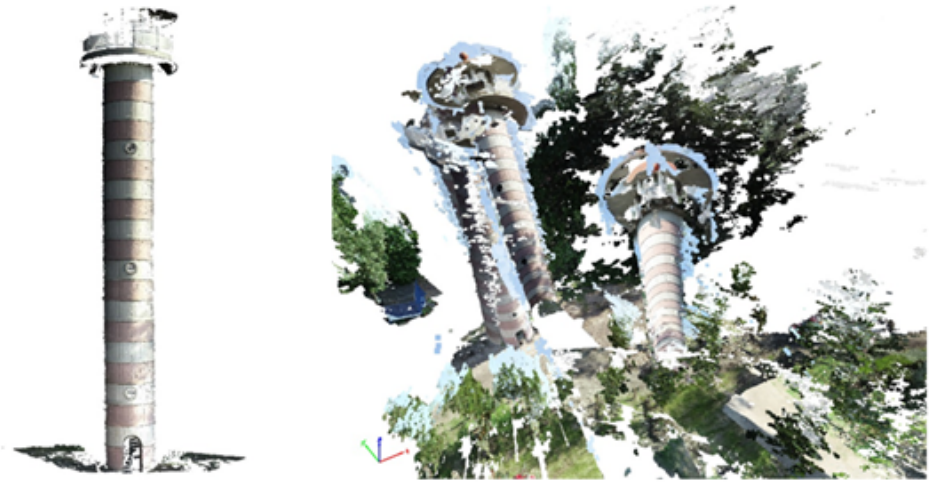


Figure 3: Comparing the resulting pointcloud of our software (left) to commercial software (right)

The prêt-à-loger in the green village in Delft (seen in figure 5.3), is an excellent case for testing and comparing the software, because it consists of many different materials. When looking at the results of our software and Pix4D from a distance, the results look similar. However when zooming in the Pix4D pointcloud is much denser and the mesh more detailed. The Pix4D results looks very realistic, even up close, but is also really expensive (figure 4).

Figure 4: Comparing the resulting mesh of our software (left) to commercial software (right)

Overall we can say our software does a good job. The results are similar or a bit less, but on the contrary of the very expensive commercial software, ours is for free. Also in the case of the image dataset with lack of horizontal overlap, our software actually gave a more useful result.

With our software the people of the Floriade project will be able to capture their scene on different moments in time and create a 3D time-lapse of their project. This can then again be used for even more innovation.

1

Introduction

This report will provide an overview of ”Digitizing Real-world Scenes from Images” team project which is part of the Geomatics Synthesis project 2018. The aim of the synthesis project is to test students capabilities in a real world assignment. Students are demanded to use their knowledge from previous courses and work together in teams, simulating a working environment. The report's objective is to present a methodology in order to create point clouds and valid 3D models by a sequence of images that depict a scene or an object from different viewpoints. Photogrammetry is a suitable technique for quickly and efficiently extracting 3D point cloud data which can then be used to create the models. It has many advantages over 3D scanning techniques for the modeler such as speed and cost. The algorithms used to extract the data are reasonably mature and robust. However care should be taken when capturing images to achieve good results and experience is needed (James et al., 2015). The team's scope was to incorporate the required algorithms into the fully functioning software,NARUX3D produced by all team members, and evaluate the resulting models in terms of detail also compare them to other existing software.

The following courses which belong to the first year geomatics curriculum provided aid in various ways. **Python programming** equipped us not only with python knowledge, but also with further programming understanding. The software was developed in C/C++ programming language, so experience on object oriented programming was important.**Sensing Technologies** in order to understand the photogrammetry algorithms. Otherwise the debugging and definition of inputs - outputs at each step would be impossible. The algorithms use certain photogrammetric constraints and thresholds so **GeoDatasets and Quality** course could provide scientific implications when managing to optimize our results. In the **3D modelling course** we developed algorithms to get meshes and 3D watertight solids from point clouds, like the second step of our software. Finally, the **Spatial Decision Support Systems** course familiarized us with a User interface since we used QT designer for QGIS in order to create a plug-in. Also we used Github to share and update the code. Github is a platform that coordinates distributed work among team members, and it is a basic component of software creation. All the aforementioned courses cover several aspects of the development of a geomatics software. However C/C++ and the working environments were completely new to most of us, so the technical part exceeded by far the expected working time and effort. Extensive use of the software and adequacy drone use which would give a better estimation of its capabilities hasn't been done properly by the time this report is written.

1.1. Requirements

The utility of 3D models lies on two main categories, real world applications and virtual applications, which can be either implementation of digital or real models. Real world application require accurate 3D models to simulate phenomena like floods, fires, or estimate wind flow

and temperature. As for visualization the examples are numerous. Architects require 3D models to gain a better aspect of space that will assist the design; nowadays they use virtual reality software for even better experience. Also the use of 3D models in games and theaters is the component that gives such a realistic results. 3D printing has been a slow revolution. There are endless possibilities for 3D printing, but these are only accessible if the product can be modeled on a computer first. [1]

A point cloud is a set of 3D samples of the surface of an object. The fast development of scanning techniques enables us to obtain dense enough point clouds to provide much more detailed information about the exact shape, size and location of spatial objects, compared to other data formats like raster or vector. Thus it is more suitable for constructing 3D building models with details. Traditional ways of generating point clouds is by using laser scanners and multi-beam echo sounding, methods that require expensive and not-flexible equipment. Photogrammetry however requires only an existing camera or mobile phone and the software. An average laptop can process an adequate number of images to create a model satisfactory for many applications on site, and additionally colour can be adjusted to points and surfaces. Furthermore a camera equipped with a GPS can produced georeferenced 3D models, which is essential for geomatics applications. However as it isn't a raw capture of data as 3D scanning is, but a result of a process it cannot reach the same level of validity. The limits of the software will be discussed in a later chapter.

1.2. Project's objectives

Point cloud extraction is a fundamental process in modern Geomatics. We will try to describe a sequence of steps that will make the transformation from 2D images possible. These steps (algorithms), along with all the necessary libraries, have been incorporated in our software. All the source code used is open source, available on Github or can be downloaded from other websites and used without restriction. The five main algorithms - modules of the software are the feature detection and matching (SIFT), the structure from motion, the dense reconstruction(Multiview Stereo) and surface reconstruction. Each method will be presented separately, together with relevant information about its accuracy and requirements. Before that, the techniques and challenges of software creation will be thoroughly discussed by our point of view. In order to develop the software the team worked on two different creating environments, Microsoft Visual Studio and QT creator and all the coding and libraries used are in C++ programming language. The workload was divided between members of team 3D Floriade, so except from a quick overview, each one of us had to expertise in a module and incorporate the libraries into the working environment. In this way everybody familiarized with c++ and the basic components of software creation. *The two objectives of this report is firstly to present our software, through images of user interface, point cloud quality and comparison with other existing softwares. The second part is to describe the methodology, evaluate the suitability of the used methods compared to others, describe the limitations of the method and refer to the domains that could benefit from this technology.* The end product is an open source software available on github[2] followed by the relative documentation and user guide.

1.3. Related work

A large increasing number of stand alone commercial software that perform photogrammetric processing of digital images and generates 3D spatial data already exist. They are usually accompanied by a high yearly subscription fee. Some of them are Acute3D, Pix4D, Autodesk Memento, Agisoft photoscan, Visual SFM. A list of existing softwares can be found on wikipedia[3]. A characteristic of such softwares, is the 3D viewer and viewing options it offers. Usually it is layer based, which enables changing between point cloud and mesh. Some soft-

[1] https://www.academyclass.com/blog/practical-uses-of-3d-modelling-in-daily-life/
[2] https://github.com/Natasja1992/3d_floriade/
[3] wikipedia.https://en.wikipedia.org/wiki/Comparison_of_photogrammetry_software

wares like pix4D Mapper adjust their algorithms to different cases, for example cases where photos are nadir or oblique flight, for map reconstruction, or terrestrial for model reconstruction.
There are also different algorithms than the ones we used for image matching and generating the dense point cloud, rather than SIFT and Multi View Stereopsis. For the key point detection, We used a feature detection algorithm since we need rotation invariance along with scale invariance. That is important in order to process photos taken by hand held camera or images found online. Another feature detection descriptors are SURF, FAST, BRIEF, ORB (Bay et al., 2006)(Rublee et al., 2011). Apart from multiview stereo algorithms more ways exist to matching images and expanding the initial point cloud like Semi GLobal Matching algorithms (Hirschmuller, 2005) (Bethmann and Luhmann, 2015).

1.4. Report Structure

The report is constructed out of the following parts. The next chapter will provide an overview of requirements and constraints. These will then be analyzed. In chapter three the concept of software development will be presented and in chapter four the five modules will be further explained and our methods will be reported. In the last chapter our software will be assessed. The report ends with conclusions and future work recommendations.

2

Requirement Analysis and Constraints

Business and Stakeholder requirements define the business need in business terminology that all involved parties can understand but the devil lies in the details. Solution providers need to know what the software must do, what kind of data it requires as an input, and which are the quality standards of the software to meet the business needs. The list of the requirements can be divided into three critical ones, functional requirements, non-functional requirements and killer requirements.

2.1. Functional Requirements

The deliverable of the project is a software, consisted of a number of algorithms. We must ensure the robustness of the software and that its computations fulfill our demands. More specifically the software should meet the following requests:

- Efficient data management
- Create dense point cloud
- Reconstruct 3D model from objects with smooth surfaces
- Reconstruct 3D model from objects with piece-wise planar surfaces
- Export result of each step

2.2. Non-Functional Requirements

The non-functional requirements include:

- Satisfaction of the stakeholder's goals as set from the beginning or rearranged during the project's execution.
- Suitability of input data for testing the software. Every software executes a number of algorithms which need an input and provide an output. Matching images and expanding the key points is a complex procedure that requires input images that meet certain criteria. Evidently the images must be of the same scene or object and have a sufficient overlap. The algorithms are capable of excluding distorted images, outliers and moving objects that interfere in some images, but in general a good quality of images will give a better result. It is important that members of the team will expertise in taking suitable images. Finally, the input data should be suitable for generating point clouds. For example, scenes with repetitive patterns, buildings with uniform material and glass walls should be avoided.
- Use of a drone to capture every surface of the buildings, including the roof. Apart from this capability, a drone can be programmed to follow a specific orbit and take pictures at

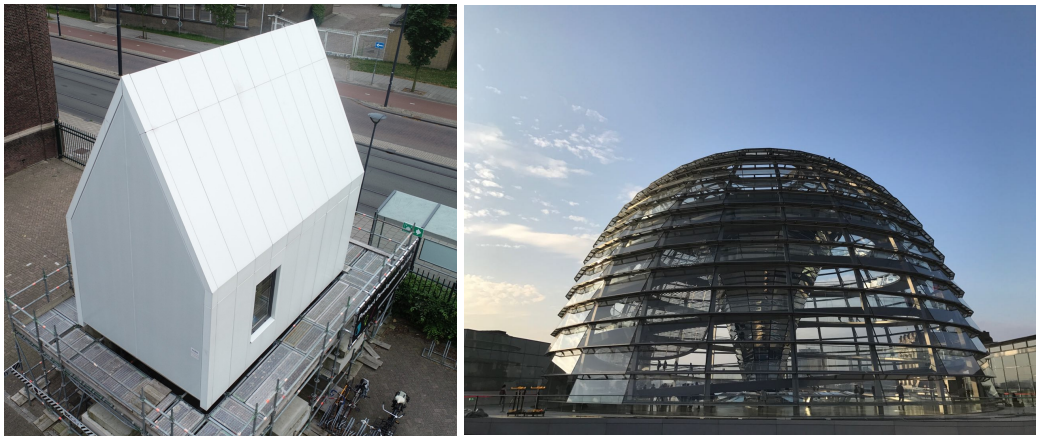


Figure 2.1: Bad examples for image matching

a specific timestamp. Once the camera's parameters are known we can compare them with the software's output for further evaluation. If GPS information is available, it is also possible to use it to speedup or improve image matching.

2.3. MoSCoW Rules

A successful project should be well organized from the beginning in order to better manage human resources and time. The MoSCow method is a prioritization technique used in management, business analysis, project management and software development to reach a common understanding with the stakeholders on the importance they place on the delivery of each requirements. It classifies the tasks of the project to those that have, should, could and won't be performed. The MoSCow table for our project is illustrated below (Table **??**).

Must	Should
• Suitable input data/object • Easy use interface • Can export intermediate and final result (point clouds, 3d models) • Can preview the intermediate and final result • Image matching • Implementation SfM algorithm • Implementation MVS algorithm • Object with smooth surface reconstruction	• Can rotate point cloud or models in the preview panel • Object with piecewise planar surface reconstruction
Could	**Will not**
• User could define the parameters of the algorithms • Can delete or replace images • One database to store images • 3D printed model • 3D Model editing	• The software will not go into LOD 3 • Support multi-object 3D reconstruction at one time

Table 2.1: MoSCoW method

2.4. Boundary Conditions

The project took place from April 4th to June 20th. The final presentation is during the Geomatics day, at June 22th. During this span intermediate targets were set and the working activities were mainly carried out at the faculty of Architecture on TU Delft campus. The project was conducted with the help of the project coordinator and coaches of TU Delft and each week a scheduled appointment was arranged between the team and the supervisors of the team.

The purpose of this project was to provide an easy to use open source software to reconstruct 3D models from images. The project partner is the department of 3D Floriade of the municipality of Almere and the software's purpose is to help them create virtual and physical 3D models of local structures. The aim of this project is to make students aware of the technology behind software that are extensively used in geomatics applications. It goes further from geomatics field, into real programming and managing of existing libraries. The team had to learn a new programming language (C++), not from the beginning, like creating simple scripts, but by exploring a structured complicated system.

2.5. Project Risk Assessment

Project risk is the total uncertain events that may have positive or negative effects on achieving project objectives. It can be either threats (additional constraints, new conditions or requirements) which might turn into obstacles as the development proceeds, or opportunities (more partners involved, new theories or methods found, or additional tools provided) that can contribute to the whole working process. There are two categories of project risks. Known risks which have been recognized and unknown risks that can not be managed (Duncan, 1996).

To ensure that the development of the project is gradient, the known risks and the uncertainties need to be managed. Project risk management will magnify the impact of positive events and restrain the influence of negatives. Like Figure 2.2 shows, the process of PRM consist of the following steps: Risk Identification, Risk Analysis and Risk Response Planning (Duncan, 1996).

Figure 2.2: PRM process

2.5.1. Risk Identification

In the process of risk identification, uncertain events which may jeopardize the whole project are documented together with their characteristics. For our project, the following risks have been recognized:

- Selection of objects for 3D reconstruction and corresponding data (images) collection;
- Design of data structures (input, output and exchange) for different software modules;
- Integration of algorithm libraries written in different versions into one software platform;
- Tasks finished on time (learning curve of C++/QT Creator/Microsoft Visual Studio/-Software Development, heavy computation tasks);
- Validation and modification of generated 3D models for successful 3D printing;
- Communication between sub-groups of the team and with other stakeholders;

2.5.2. Risk Analysis

In this part we mainly perform qualitative risk analysis. To start with, risks or uncertain events mentioned above are broken down into different categories: internal vs. external; technical vs. organizational. Then, possibility of occurrence and corresponding impact of these risks are evaluated. Based on these two factors, risks are prioritized into different levels (Table 2.2).

	Technical	Organizational
Internal	• Design of data structure across system; • Algorithms libraries integration; • 3D models validation and modification;	• Communication between sub-groups; • Time management on different tasks;
External	• Object selection and data(image) collection;	• Involvement of Floriade team;

Table 2.2: Risks break-down

Internal risks are uncertain factors that our team might meet, while working on the project. Concerning the technical part, a leading issue we dealt with, was the designing of data structures for different modules of the software platform, such as image input, image matching, generating sparse point-cloud, building dense point-cloud and surface modeling. This had to be done from every member of the team, in order to keep the software developer aware of the right input and output formats and paths, as the main modules' libraries use unique structures. For interoperability across software platform, these data structures had to be designed in parallel with the integration of libraries. A major technical risk was the integration of algorithm libraries; doing that on C++ was a completely new procedure and the risk of exceeding the time deadlines was high. Also, some of the libraries were in different programming languages and maybe outdated so we had to search for the right ones to integrate them all together. The validation and possible modification of generated 3D models also had to be planned correctly since we had to schedule the printings. As far as internal organizational risks concern, the team members worked on different modules individually but there was a point where the work had to be merged. This part was tricky, and we had to ensure that all the formats and structure were the same. Due to the aforementioned risks, time management was essential. We had to organize our work in an efficient way, so that tasks related were resolved one after another based on their priority.

External risks are uncertain events in which our client or partner, team Floriade is involved. Our initial plan of data collection or image capturing was relying on their support. Aerial photos of buildings in Floriade area are captured by a drone operated by a pilot with flying license from Floriade team. However, there are no buildings on the site right now and there is the possibility that buildings they select are not suitable for 3D reconstruction (with little outstanding features to be recognized) or photos taken by the drone are not adequate (not enough overlapping) existed.

Table 2.3 shows chances of occurrence as well as potential impact of above mentioned risks (Scale ranging from 1 to 5. 5 as the most). Based on these two indexes, above mentioned risks (internal vs. external, technical vs. organizational) were evaluated and prioritized.

2.5.3. Risk Response

Risk response is the process of developing alternative plans to control potential risks. Risks we analyzed before are addressed in following three strategies: Avoid, Mitigate and Accept. Risk avoidance eliminates risks by changing project plan or objective; Risk Mitigation means reducing possibility and impact of risks to be within acceptable threshold limits; Risk acceptance implies acknowledging risks but not acting on them unless they actually happen (Duncan, 1996).

	Occurrence Possibility	Potential Impact	Priority
Design of Data Structure	5	4	5
Algorithm Libraries Integration	5	5	5
3D Models Validation and Correction	4	4	4
Communication between Sub-groups	3	4	3
Time Management on Tasks	4	4	4
Objects Selection and Data Collection	3	4	3
Involvement of Floriade Team	3	4	4

Table 2.3: Results of Risk Analysis

Table 2.4 shows the different methods we take for uncertainties in our project based on their priorities.

Risk Avoidance	• Adjusting project plan to provide more time for process of 3D model validation and modification; • Using normally used open-source libraries and data structures;
Risk Mitigation	• Implementing familiar algorithms, not unknown ones for 3D reconstruction to reduce potential problems; • Holding group-meetings regularly to share working process of sub-groups on different software modules;
Risk Acceptance	• Taking photos of buildings using hand-held cameras by ourselves as a 'backup' plan for data collection/image capturing process;

Table 2.4: Risk Response for our project

3

Concepts

This chapter will be about the concepts for our project. First we will discuss the concept of our software design and implementation itself. After that we will explain how we are going to test our software.

3.1. Software Concept

The result software will be a program based on C++ language code, developed in the QT Creator and Microsoft Visual Studio environment. It will be an inter-operable computer software, supported by different Operating Systems.

3.1.1. Software Basics

The development of the software is based on the target users of the end software. The software will need to present all the functionalities required in the digitizing of real-world scenes from images process and adapted to users which are not acquainted with the different theoretical concepts used in the whole process.

The software development is a modular process where each part of the project will be developed separately. Each module will be defined and, in a final step, all the modules will be integrated into a main User Interface, which will be the skeleton of the software (Figure 3.1).

Developing a Software in a modular process will improve the agility of the creation of the program, reducing the dependencies of each developer with the others. However, a continuously interactions between developers is required, in terms of guaranty the final integrity of the software.

Each module will have a determinate input source and a determinate output result. The input will be provided by the Main module in a standard format, which will be readable by each module. Inside the module, the input will be processed and, using different predefined tools based on theoretical concepts, the resultant output will be obtained (Figure 3.2).

Something relevant to mention is that the input and output formats will be predefined standard formats, which will be described in the Main Module. This fact will allow to maintain the integrity of the software. Each module will be allowed to change the format of the different sets inside their domain. However, when the set is submitted to another module, the format employed will be the standard format predefined in the main module.

Finally, the software will allow the user to preview different intermediate results. Those results will be presented in different viewers adapted for the type of data which will be presented. Intermediate results will be able to save in external files to be loaded in future projects too. Other advantage of the Modular process is that the process does not need to be started

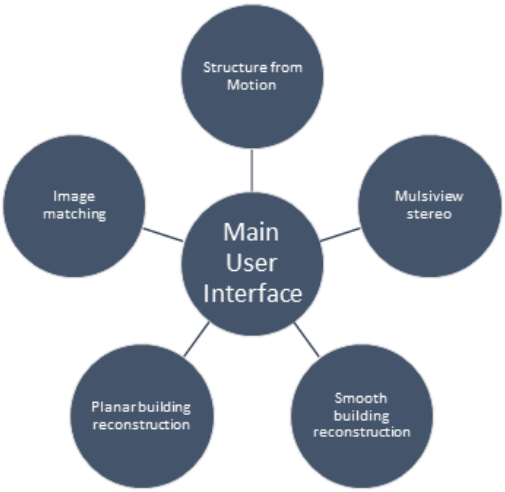


Figure 3.1: Modular integration

from the beginning. It will be possible to load an external file with the information of previous steps of the process to continue from an intermediate point (Figure 3.3)

Each module will have access to different repositories with pre-implemented functions which will be used on the digitizing of real-world scenes from images process. Those libraries will be adapted and included in the final software (Figure 3.4).

3.1.2. Modules

The software is composed by a set of modules, interrelated by a main module. The user will be able to interact with each module using a graphic interface, developed to supply all the requirements present in the process.

Main Module

The main module will be the manager of the software. It will link the inputs and the outputs of the different steps of the whole process. The Main module will connect the request of the user, provided by the User Interface, to the module which stores the response output. The different features related with this are:

- Project management: This feature will the core of the process. It will store the main settings of the project, as each input and output elements presented in the reconstruction.
- Image management: Control of the set of Image which will be the main output of the software.
- File management: The user will be able to load external files to start the process in an intermediate point, using a previous intermediate input.
- Process management: The task of this feature is to dispose each step of the reconstruction process. The main module will bring the required input for each module and it will store the resultant output.
- Render management: Different rendering options as Screenshot tools.
- Viewer management: Viewers to be allowed to see models of the different outputs of the modules.

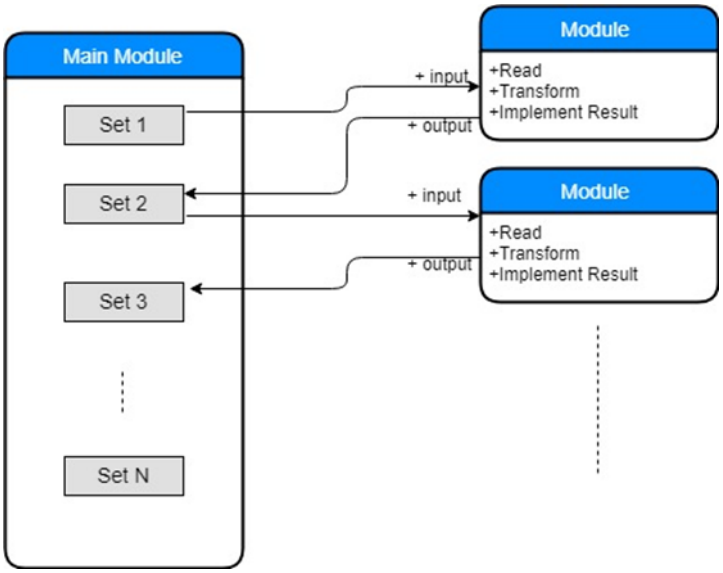


Figure 3.2: Modular process

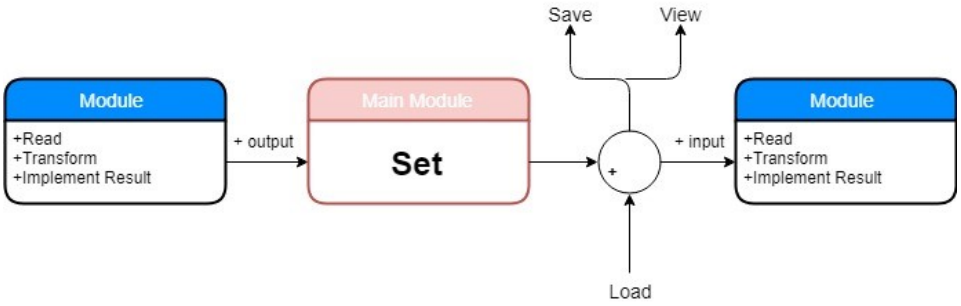


Figure 3.3: Interaction

- Support management: Provides support tools, as the selection of the main settings.

The main module will be in charge of store the different elements of the whole process, in a standard format which allows to the other modules to use them without drawbacks. The Table 3.1 shows the different formats:

Viewer Module

The Viewer will allow to the user to see, in a graphical way, the evolution of the process. The viewer module is composed by two different elements:

- 3D Viewer: 3D model viewer based on OpenGL and with the implementation of the capabilities provided by the libQGLViewer library, a C++ library based on Qt that eases the creation of OpenGL 3D viewers.

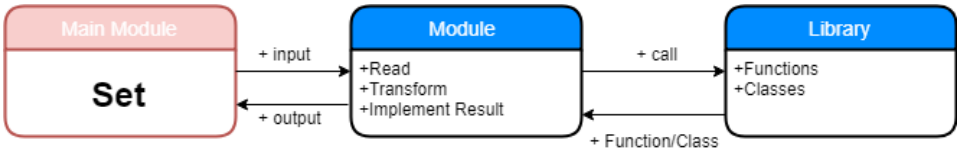


Figure 3.4: Library process

Image	Path of each element
Project	Settings file element with all the different settings
KeyPoints	File element with the pixels of the KeyPoints
PointCloud	Table element with the 3D coordinates
Features	Text element with features of Image
3D Model	3D element with 3D objects

Table 3.1: Main formats

- Logger: Logger is used to present indications of the process, data about input and outputs, features of image, etc.

Image Matching Module

Features of the different images will be provided in this module. Those features will be obtained from the set of images selected by the users (input element). The result will be the features of the different images with their KeyPoints.

Structure from Motion (SfM) Module

This module 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. SfM produces sparse point clouds (Measure of set of points as data file which is rendered. There is a need to carry out surface reconstruction).

Multiview Stereo Module

This module takes a set of images and camera parameters, then reconstructs 3D structure of an object or a scene visible in the images. Only rigid structure is reconstructed (the software automatically ignores non-rigid objects such as pedestrians in front of a building). The software outputs a set of oriented points instead of a polygonal model.

Poisson Surface Reconstruction Module

The Poisson surface reconstruction creates surfaces (the mesh on all of the surfaces is complete) from oriented point sets.

PolyFit Module

PolyFit Module takes as input a point cloud (possibly noisy, incomplete, and with outliers) of a piecewise planar object and outputs a lightweight polygonal surface model.

3.2. Software Testing

Once the software is working we will start testing it on different aspects. We will test how it handles different kind of buildings or objects, the stability of the program and check performance.

In order to check how our program is working we will test it for different cases. We will create datasets with images of different building and can also test the program for different kind of objects to see how it handles those. We can find buildings or objects with different kind of materials like brick or wood. We can also try a building with more glass, although we already know this is probably not going to work very well. Also different shapes can be tried. Both buildings with lots of straight surfaces and also buildings with round or other

more divergent shapes. We will check and report on the handling and results of the different cases, and if possible improve the program.

We will also test the stability of the program itself. We will do this by providing incorrect input and make sure that exceptions are handled correctly. Another test is checking what happens when an error occurs in one of the modules. How does the program handle this and are intermediate results saved? Also the GUI itself will be tested. We will check how stable the program is by clicking and dragging a lot and also see what happens to the GUI while running.

Last we will check the computation times of the program. It will be interesting to see how the program performs and also some possible problems can come to light this way. For the most part the actual optimization of the program will probably be outside our scope, since we don't have enough time and knowledge on hardware, performance and other subjects to do this, so this will be future work.

4

Engineering Design

In this chapter we will discuss the technical detail of our software. We will discuss the user interface and also the different modules. After that we will analyze our process and evaluate risks.

4.1. User Interface

The user interface (UI) has been developed taking into consideration all the services that the software should present. The user interface should provide an efficient and smart way to allow interaction between the user and the program.

4.1.1. User Interface Design

The user interface has been created based on different principles (an example of the interface is presented in the figure 4.1):

- Structure principle: Clear, useful, consistent, well-related items.
- Simplicity principle: Simple, clear and well-related to longer procedures.
- Visibility principle: All needs for a given task visible. Not-overwhelm.
- Feedback principle: Keep users informed about the state of the interface.
- Tolerance principle: Prevent input/output errors.
- Reuse principle: Reuse internal/external components, maintaining consistency with porpoise.

4.1.2. User Interface implementation

The user interface has been developed in two different windows. The different windows will allow the user to use the software without any other external tool.

Main Window

The main window will provide not only the possibility of access to the different modules inside the reconstruction process, but also different capabilities that the user can use to manage the whole project.

The main window will include a text viewer to present indications of the process, data about input and outputs, features of image, etc. Finally, it will include an image viewer. The appendix A presents a predefined example of the interface.

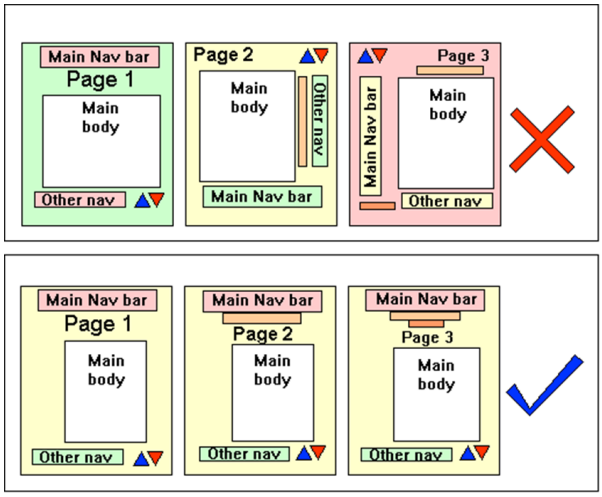


Figure 4.1: Examples of UI

Viewer Window

The viewer window will provide a 3D viewer, based on the API OpenGL language and OpenQGL library. This viewer will be used to present the different PointCloud created on the process, as the final 3D Model. The figure 4.2 present and example of the viewer.

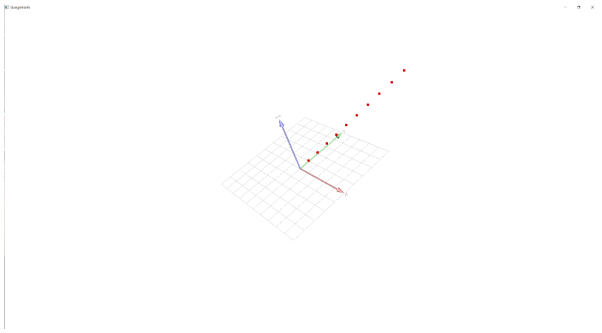


Figure 4.2: Example of the Viewer window

4.2. Components

4.2.1. Image Matching

In order to match two images, we first need to detect features, also called key points, and then match key point pairs. The essential question of local image feature descriptor is robustness. There are plenty of algorithm for image matching, for example SIFT(Lowe, 2004), SURF(Bay et al., 2006), BRISK (Leutenegger et al., 2011), ORB (Rublee et al., 2011), FREAK(Alahi et al., 2012). Sufficient research (Suo et al., 2012, Karami et al.) had proofed that both SIFT and SURF have excellent performance, but their running speed are quite slow (table 4.1) . Adopting to exist popular open source third party libraries, our project implemented both OpenCV SURF library[1] and SiftGPU[2]. In practice, both of them provide reliable matching result, while SiftGPU has a more faster running speed, therefore in this project we are going to implement SIFT algorithm and SiftGPU library.

In this project we are going to implement Scale-Invariant Feature Transform (SIFT) algorithm developed by Lowe (2004). SIFT is a method to detected local features, and this

[1] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

[2] https://github.com/pitzer/SiftGPU

	SIFT	SURF	BRISK	ORB	FREAK
Average running speed (s)	13.8936	4.17246	1.10988	0.35294	0.844758
Rotating image successful match number	697	1038	9	125	56
Resolution variation successful match number	1699	1774	67	416	307
Illumination variation successful match number	1559	1828	170	318	276
Scale variation successful match number	697	1038	9	125	56
Orientation variation successful match number	589	1894	62	306	528

Table 4.1: Different algorithms performance when matching same two (Suo et al., 2012)

algorithm extracts interest points or keypoints and their scale and orientation descriptors to get features, then runs image interest points matching. This algorithm has already implemented by plenty of studies and software, also been included by many open source libraries, and got reliable result. The whole process of Image matching contains two parts: implementation SIFT to detect keypoints and get descriptors, matching keypoints between every two images, which process can be divided into 5 steps:

1. Scale-space extrema detection: First step is to build a scale space for each image. The purpose of a scale space is to simulate the vary scale feature of image data, find invariant keypoints in a continuous changing scale to make sure that the images are invariant in affine distortion and orientation. It is implemented efficiently by using a difference-of-Gaussian function.

In the Gaussian Image Pyramids, the scale space, $L(x, y, \sigma)$ is produced from the convolution of a variable-scale Gaussian, $G(x, y, \sigma)$, with an input image, $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y);$$

where * is the convolution operation in x and y. The Difference of Gaussian is the difference between two neighbor Gaussian images in one Octave, $D(x, y, \sigma)$, this function provides a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$,

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y);$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G;$$

And to detect the extrema, the DOG image pixel value must be maxima or minima among its 26 neighbours pixels (Figure 4.3).

2. Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability. The extrema points getting from step 1 contains low contrast feature points and unstable edge response points which should be filter out. The extrema point is a three-dimensional vector $X = (x, y, \sigma)$, using three-dimensional Taylor expansion, set $X0$ as the discrete difference center, $D(x, y, \sigma)$turns into:

$$D(X) = D(X_0) + \frac{\partial D^T}{\partial X}(X - X_0) + \frac{1}{2}(X - X_0)^T \frac{\partial^2 D}{\partial X^2}(X - X_0);$$

Setting $\widehat{X} = X - X_0$, taking the derivative of $D(X)$, setting this function with respect to x and setting it to zero will get:

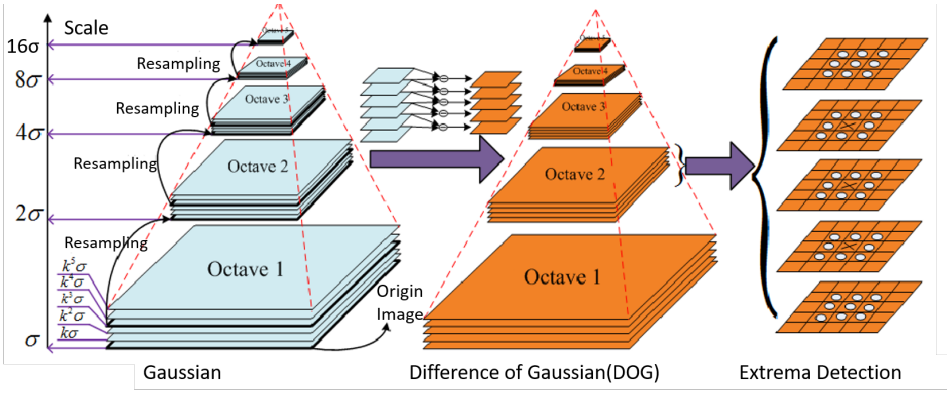


Figure 4.3: Scale-space extrema detection (https://blog.csdn.net/lhanchao/article/details/52345845)

$$(\hat{X}) = -\frac{\partial^2 D^{-1}}{\partial X^2}\frac{\partial D}{\partial X};$$

Taking $(\hat{X})$ into $D(X)$ will get:

$$D(\hat{X}) = D + \frac{1}{2}\frac{\partial D^T}{\partial X}\hat{X};$$

If the offset value $\hat{X}$ is larger than 0.5 in any dimension, then it means that the extremum lies closer to a different sample point. In this case, the sample point is changed and the interpolation performed instead about that point. The final offset $\hat{X}$ is added to the location of its sample point to get the interpolated estimate for the location of the extremum. If in a certain interpolation times, the offset value is still larger than 0.5, this point should be discarded. Also, if $|D(\hat{X})|$ is less than 0.03, it means that this point has low contrast, and should also be discarded.

According to Lowe, SIFT implements a threshold to eliminate edge extrema points. After this approach the extrema points remaining now are strong interest points/keypoints.

3. Orientation assignment: To implementing feature point rotation invariance, one or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

In a certain radius of every keypoints, all the pixels will be involved to calculate gradient and orientation. The radius r is 3×1.5σ, σ is the scale in this octave. The gradient $m(x,y)$, and orientation $\theta(x;y)$ can be calculated as:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \arctan\left(\frac{(L(x,y+1) - L(x,y-1))}{(L(x+1,y) - L(x-1,y))}\right)$$

Also, the samples don't have the same weight, a Gaussian-weighted circular window with $\sigma m = 1.5\sigma$ is used here, then add samples to the histogram. The highest value of direction will be the keypoint's main orientation, $|\vec{v}|$, other values larger than 80%$|\vec{v}|$ will be assistant orientation.

4. Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination. First rotate the coordinate axis as the same direction of keypoint's orientation; recalculated the coordinates, as shown in figure 4.5.

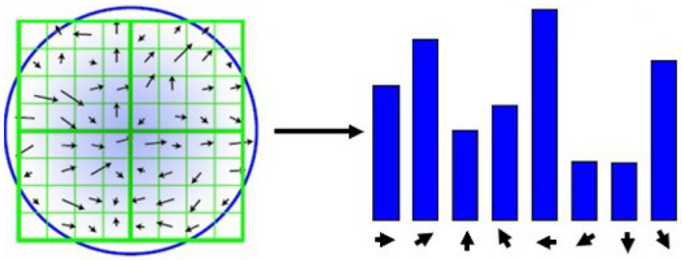


Figure 4.4: Orientation Assignment (https://blog.csdn.net/zddblog/article/details/7521424)

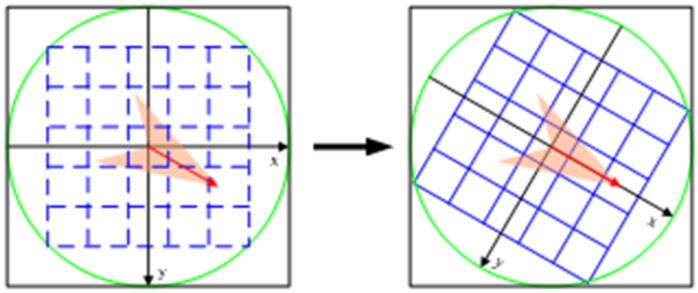


Figure 4.5: Rotate the Coordinate Axis (https://blog.csdn.net/zddblog/article/details/7521424)

Create 16 x 16 window of the keypoint, divided into 16 sub-blocks of 4x4 size. Similar to step 3 established 8 bin orientation histogram. Linking all sub-blocks histogram will get a 128 dimension vector. To eliminate the effects of light, the vector should be normalized. In this process, each keypoint will has a 4×4×8= 128 dimension descriptor.

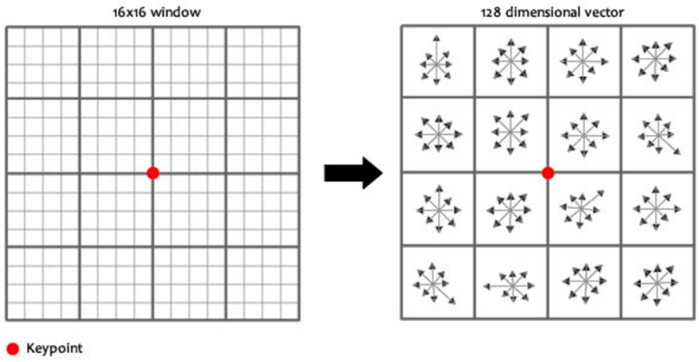


Figure 4.6: Dimension Descriptor (https://blog.csdn.net/abcjennifer/article/details/7639681)

5. Keypoints matching: Each original image has certain scales and keypoints with 128 dimension descriptors. Between every two images, match every pairs of descriptors in all scale. In practice , our program uses Euclidean distance to match keypoints. For a keypoint in one image, find the closest two keypoints in another image, the distance is d1 and d2, and d1 < d2. If d1/d2 is smaller than a certain ratio, this pair of krypoints match is accepted. The ratio between 0.4 to 0.6 will perform best. A lower ratio will get higher accuracy matching, while the number of matching pairs will fewer, vice versa.

4.2.2. Structure from Motion

Structure from Motion (SfM) is the process of reconstructing 3D geometries from a sequence of 2D images taken from different viewpoints on same objects. There is a variety of SfM strategies, including incremental, hierarchical and global approaches [14]. In our project,

we implemented the incremental one, which is the most popular strategy for reconstruction from unordered photos.

Figure 4.7 shows the pipeline of incremental SfM.

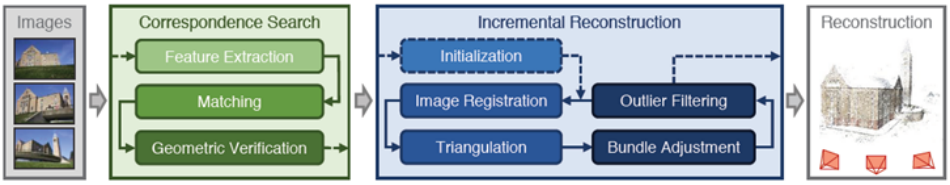


Figure 4.7: Pipeline Structure from Motion (SfM) [14]

In our project, the first part of SfM process Correspondence Search has been realized in module Image Matching. The output of this module is 'a set of geometrically verified image pairs and their associated inlier correspondences' [14]. A description of their geometric relation could also be added.

The main part of SfM is Incremental Reconstruction. Outputs from Image Matching are used as inputs. Pose estimates for registered images and a set of 3D points which represent the scene structure are generated as outputs. Incremental reconstruction starts with Initialization. Two views or a pair of overlapping images are chosen as start point; Later, in the step Image Registration, other images are registered to the existing model by using feature correspondences to triangulate points in already registered images; During the step of triangulation, scene points are triangulated and added to point set from newly added images which cover new scene parts from different viewpoint; Last and also the most important step is Bundle Adjustment (BA). BA is 'the joint non-linear refinement of camera parameters and point parameters that minimizes the reprojection error' [14]. It prevents SfM process from drifting to a non-recoverable state.

The output of SfM module of our project is a sparse point cloud which would be further refined by Multi View Stereo. Figure 4.8 shows one testing result of SfM (Sparse Reconstruction).

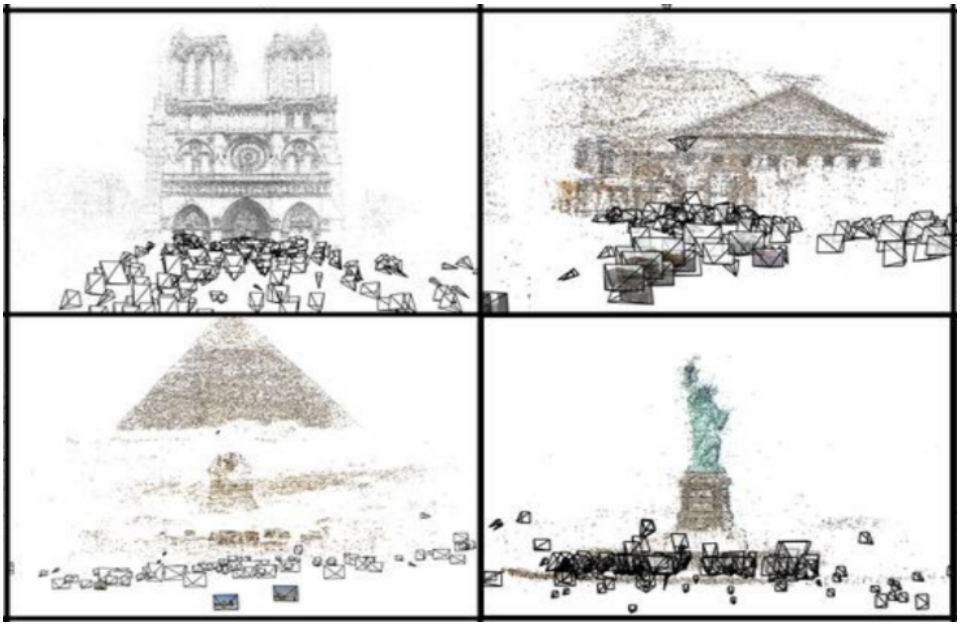


Figure 4.8: Testing Results of Sparse Reconstruction (SfM)

4.2.3. Multi-View Stereo

The method used for extracting a dense point cloud is a multi view stereopsis algorithm developed by Furukawa and Ponce (2010). It is a patch-based multi-view stereo method. The algorithm takes the features, and a set of images and camera parameters calculated on the two earlier stages as inputs and returns a dense set of small rectangular patches, which cover the surfaces visible in the images. A patch is a local tangent plane approximation of a surface. It is characterized by its center, normal vector and reference image. The centers of this dense set of patches will constitute the point cloud. The output file will be a text PLY file featuring the points in 3D space, their normal vector and the colour which is extracted by the reference image.

Multiview stereo algorithms have the advantage that suit to challenging goals. They can cope efficiently with occluded scenes and scenes where moving obstacles appear in different pictures, which is common while gathering such data. This approach doesn't require any assumptions of the topology of objects or scenes and does not need any initialization like a visual hull or bounding box. It can efficiently reconstruct the surfaces visible by the camera. However it may face problems where surface texture is poor and the overlapping of of images isn't sufficient.[5]
The MVS algorithm was chosen instead of a semi global matching approach like SURE or Photoscan because they deal better with images from close distances. The global methods often yield better performance than local methods, but with a more complex algorithm. Semi Global Matching methods are proven to deal well with low texture uniform surface and edges but are not applicable for close range because too much information is lost (Yan et al., 2016).

This algorithm is implemented as a match, expand and filter procedure. The matching yields a sparse set of patches on the image regions that features have been detected. In our case we have obtained the sparse point cloud from the previous step, structure from motion, so we implemented only the expansion part. The aim of the expansion is to fill every image cell, neighbor to the patch (in this algorithm consist of 2*2 pixels) with at least one patch, by identifying a set of nearest cells, and then spreading the initial matches. The photometric discrepancy function is used to check the photometric consistency of each picture compared to the reference picture. If the photometric discrepancy score is below a threshold, the path is initialized. In the end visibility constraints are used to filter incorrect matches. In this algorithm the expansion and filtering steps are iterated three times, which helps the process of complicated surfaces and rejects outliers more efficiently.The result of the expansion from the sparse point cloud can be seen in figure4.9

Figure 4.9: Final patches after expansion

The necessary functions to construct the algorithm are provided in the pmoulon library "CMVS-PMVS" which is accessible on github. The computation time of this algorithm is high, especially when dealing with a large number of high resolution pictures. But the time sacrifice is worthy because despite a dense and accurate point cloud the normals of the points are calculated. The use of GPU for speed up is highly recommended as it limits the computation time to around one third. Evaluation of the method on various datasets can be find at the URL[3]

4.2.4. Smooth Building Reconstruction

In this section will be discussed how we are going to construct smooth surfaces from the created dense pointcloud. This process is especially helpful for non-planar buildings and objects. In order to reconstruct the surfaces we are using the poisson surface reconstruction method.

The paper of Kazhdan et al. (2006) explains how a surface can be constructed by approaching it as a spatial Poisson problem. In this approach a watertight, triangulated approximation of the the surface is reconstructed from pointclouds with oriented vertices. These are vertices where the position, normals and possibly other properties are known. This reconstruction is done by computing the indicator function of the model as accurately as possible and extracting the isosurface.

The indicator function can be computed by deriving a relationship between the gradient of the smoothed indicator function and an integral of the surface normal field. The surface integral is approximated by summing the oriented points. Last the Poisson problem is solved.

The advantage of this approach is that it considers all the points at once instead of segments of the data like other methods. This way very smooth and detailed areas can be created from noisy data. This can be helpful in our case.

In our software we will implement the approach described in the paper [9]. Their source code can be found on GitHub[4]. Here we can find several executables that can be called with certain flags. We will create meshes with color so the input dense pointcloud should not only have the position- and the normal- values of the points but also the color information. We will provide this input in a ply format. And by using the color flag also process this color into the mesh. The output will be a triangle mesh in ply format.

4.2.5. Piece-Wise Planar Building Reconstruction

Another approach for creating a 3D mesh from the dense pointcloud is via planar reconstruction. This method lends itself very well for planar buildings. Since buildings often consist of a few straight planes like walls and roofs, this approach can be very useful. It can give a clean and clear output, with only a few polygons, which can be very efficient in large models with a lot of buildings. The method we will use is the polyfit method as described in the paper of Nan and Wonka (2017). Their source code can be found on github[5].

[3] http://vision.middlebury.edu/mview/eval/
[4] https://github.com/mkazhdan/PoissonRecon
[5] https://github.com/LiangliangNan/PolyFit

5

Software Testing

After the successful release of the software the final part was to input images from real buildings and scenes and test its behavior in our software. In the following subsections an analysis of the output pointclouds and meshes will be discussed through matches, figures, and comparisons.

5.1. Testing for Different Building Objects

We tested our software for different cases. The cases all have different characteristics and materials. We chose to reconstruct a Watchtower based in Almere (figure 5.1, the photos of which were captured by a drone, offering of the Floriade department. The second building was a wooden house located on the TU Campus (figure 5.2. The images are terrestrial and were captured by two different mobile phones. For the third and fourth tests we managed to take aerial drone photographs of another house at the TU Campus(Home with a skin, figure 5.3) and a church located in Delft (figure 5.4. For home with a skin the result was a solid shape point cloud which 3D model was created with both surface reconstruction methods, Poisson reconstruction and PolyFit. The following figures show the different cases that we handled. They show one of the input pictures, the dense pointcloud and the created mesh via poisson reconstruction, and in the case of the 'home with a skin' also the PolyFit mesh.

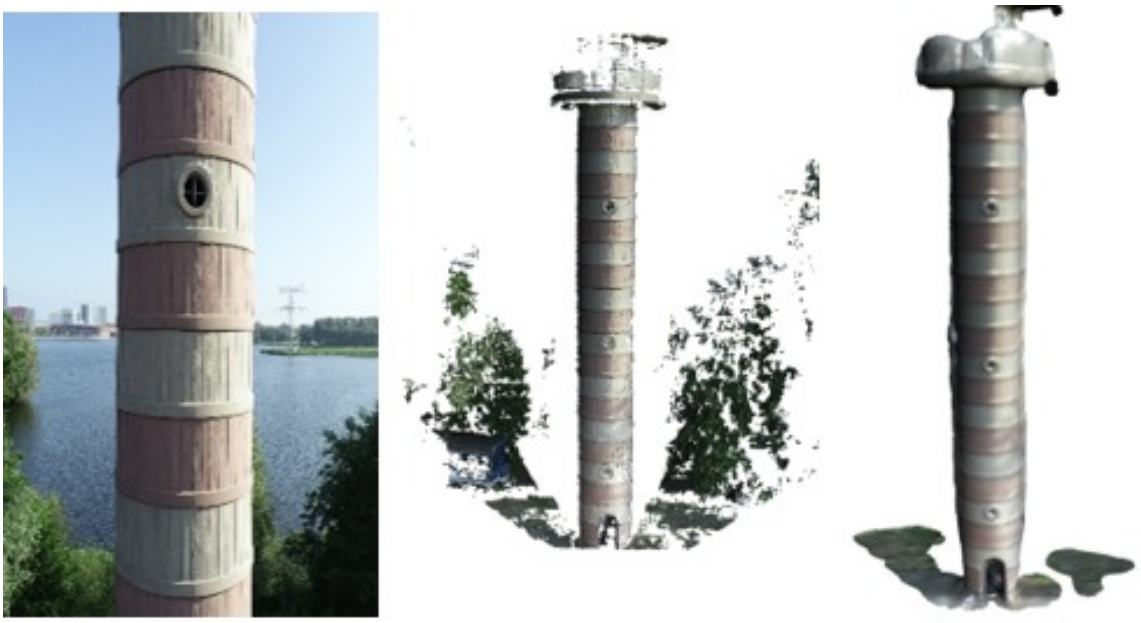


Figure 5.1: The Almere watchtower

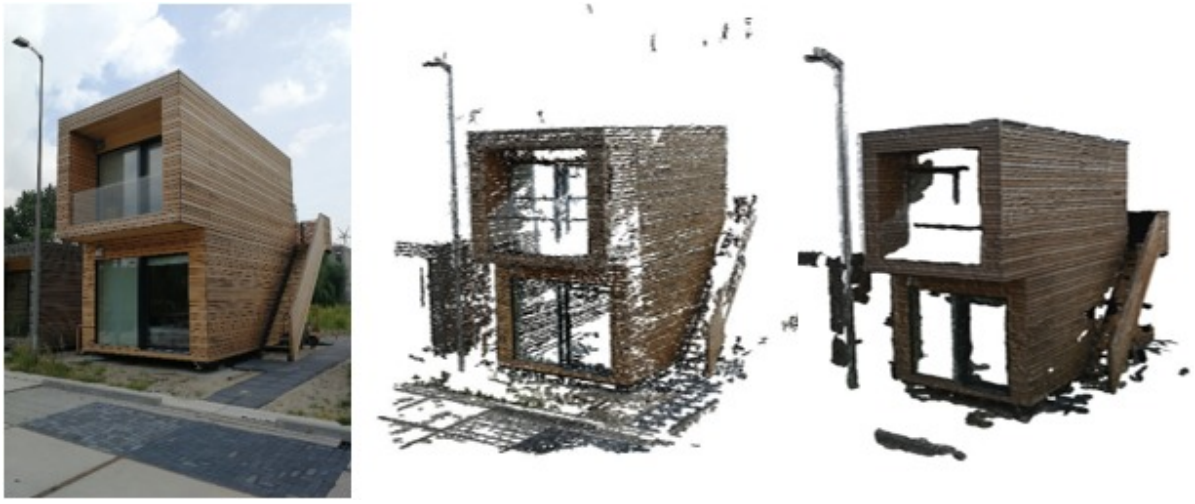


Figure 5.2: Wooden house in Green Village Delft

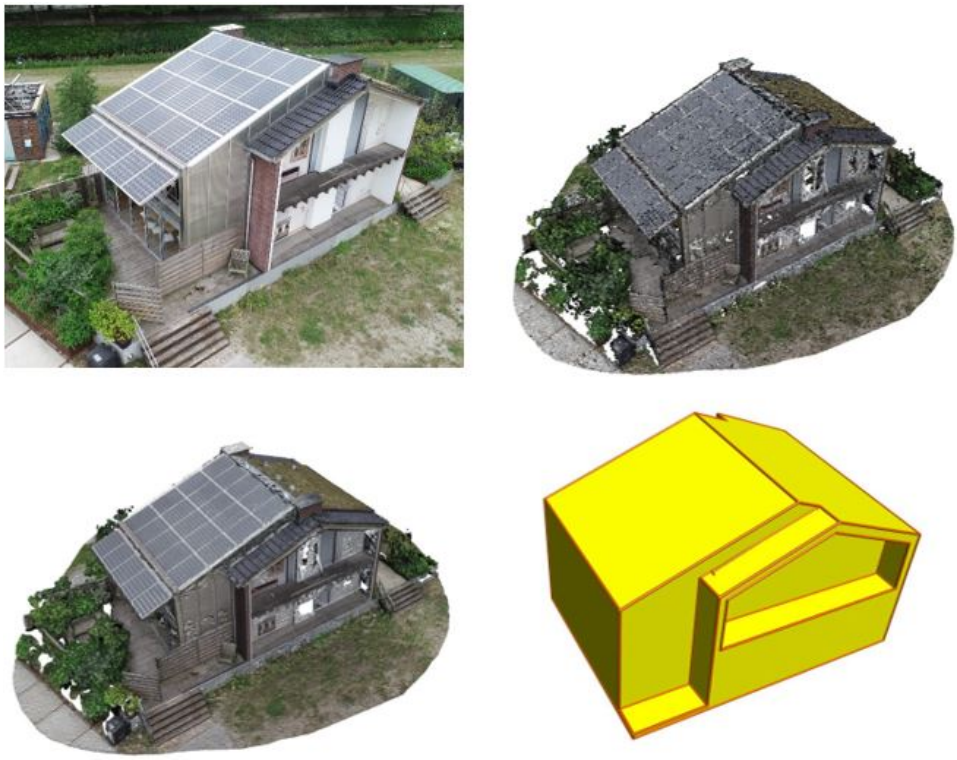


Figure 5.3: Home with a skin in Green Village Delft

Table 5.1 contains the name of the structure, the camera used, the number of input images and the number of images that were finally used after photogrammetric consistency filterings, the viewpoint angle of the camera and finally the total number of points, after cutting to the extent of the structure.

When looking at the results, we can see that our software works well for brick and wood. However for glass surfaces and other uniform surfaces, it does not work.

Figure 5.4: Church in Delft

Object	Camera	number of pictures	angle of sight	number of points (after clipping to extent)
Watchtower	Drone	99 - 27	235	892,068
Church	Drone	400 - 400	180	4,287,413
Wooden House	Mobile Phone	59 - 45	270	1,679,687
Home with a skin	Drone	99 - 99	360	552,249

Table 5.1: Reconstruction parameters

5.2. Comparing with Similar Software

Comparing with commercial software which has similar function would be a good way to assess our own software, NARUX3D. The one we chose is Pix4Dmapper, a well-known drone mapping and photogrammetry software platform. One of its many features is reconstructing 3D scenes from 2D images. The input of Pix4Dmapper is a set of images taken by drone or any other cameras. The output results could be 3D point cloud, digital surface and terrain model, orthomosaic photos, volume calculation result, contour lines and even 3D textured models[1]. Pix4Dmapper also provides cloud service for storage, computing, analysis and sharing (figure 5.5).

[1] https://pix4d.com/product/pix4dmapper-photogrammetry-software/

Figure 5.5: Pix4Dmapper workflow (`https://pix4d.com/product/pix4dmapper-photogrammetry-software/`).

Basically, comparing to our software, Pix4Dmapper has more functions and great cloud service. But it comes with a high price: A yearly subscription of Pix4Dmapper costs 2600 EUR (monthly 260 EUR). For small project group or individual researcher who just want to get 3D point cloud out of 2D images, is it worth it? To answer this question, we use images of three different buildings as input and compare the output results (3D point cloud and meshes) from our software and Pix4Dmapper.

Case 1: Tower from Almere:
The first case we tested is a tower from Almere. With the help of 3D Floriade team in geo-department of Almere, a series of aerial photos are taken by the drone. The tower is built with concrete, quite isolated (no surrounding plants or other construction) and has only a few decoration . Figure 5.6 shows the images of tower in Almere.

Figure 5.6: Tower in Almere. From 3D Floriade team in Almere.

These aerial images are used as input for our software as well as for Pix4Dmapper. Output results, both point cloud and mesh are compared.
Figure 5.7 shows the difference in point cloud/mesh output results from our software and Pix4Dmapper.

Figure 5.7: Comparison results with Pix4Dmapper. (Upper left: Point cloud generated by our software; Upper right: Point cloud generated by Pix4Dmapper; Lower left: Mesh generated by our software; Lower right: Mesh generated by Pix4Dmapper).

Both point cloud and mesh output from our software are better. They preserve the original form of tower with structure and texture details. Output results from Pix4Dmapper in this case are not good: three clusters of point cloud are built separately with tearing parts.
The reason that Pix4Dmapper is not doing so well as expected may be caused by the way data was collected: Images were taken while the drone, which was quite close to the tower, was ascending from the ground. In this way, each image only covers part of the tower in vertical direction. Later, drone changed its orientation, flew to the other side and took some photos. And before it landed, it flew to another side and took photos again.
Figure 5.8 shows flying path of the drone and images distribution on the map. In this way, photos that were taken near the same location overlapped vertically with each other but has no horizontal overlap at all. And photos taken in different locations have no overlaps in both directions. In this way, Pix4Dmapper will generate three separate towers instead of one. Our software, however, will drop those images that have not enough matches and generate a better result.

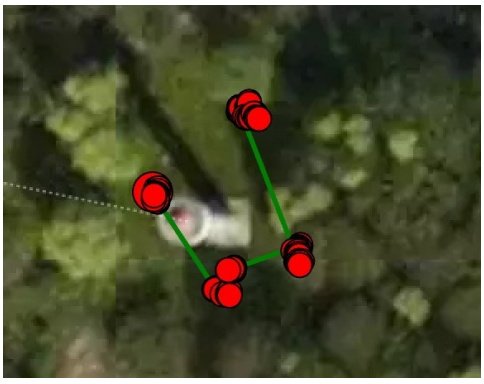


Figure 5.8: Flying path around tower in Almere.

Case 2: PRÊT-À-LOGER in Green Village of TU Delft:
The second case we tested is the prêt-à-loger, a sustainable building with 'second-skin' (a smart greenhouse structure with energy saving measures) located in green village of TU Delft[2]. It consists of multiple types of material: wood, brick, glass, steel and plant cover. It is a good case to test if our software as well as Pix4Dmapper can handle facades with different material or texture. Figure 5.9 provides overview of prêt-à-loger.

Figure 5.9: PRÊT-À-LOGER in Green Village, TU Delft.

In this case, a drone was used to capture photos of prêt-à-loger. The building itself was set as point of interest (POI) and drone was flying around it, taking photo every two seconds. During the flight, 99 images were captured, and these aerial photos are used as input for our software as well as Pix4Dmapper. Figure 5.10 shows the differences in output point cloud and mesh from our software and Pix4Dmapper. In general, point cloud generated by Pix4Dmapper is much denser than that from our software. Also, the mesh result from Pix4Dmapper is cleaner and looks so realistic. Mesh result from our passion reconstruction is not that good but very close.
This case shows that, our software could handle buildings with complex facade material and textures. As long as input images have good overlaps, it could generate similar or slightly worse results, both point cloud and mesh comparing to other popular commercial software, like Pix4Dmapper.

Figure 5.10: Comparison results with Pix4Dmapper. (Upper left: Point cloud generated by our software; Upper right: Point cloud generated by Pix4Dmapper; Lower left: Mesh generated by our software; Lower right: Mesh generated by Pix4Dmapper).

[2] https://www.thegreenvillage.org/projects

Case 3: 'White House' near BK, TU Delft:
The third case we used to test NARUX3D and Pix4Dmapper is a small white house built upon scaffolding near BK (Bouwkunde, building for Department of Architecture) on campus of TU Delft. Facades and roof of this small hut are made of precast slabs which are totally white without any texture, together with steel structure and glass. It is a quite challenging case for our software, as we use SIFT algorithm for image matching and it will not work so well on surfaces without any distinctive features. In this scenario. It would be nice to see how far NARUX3D goes and if highly competent Pix4Dmapper could produce much better result. Like Case 2, images of the building are captured by a drone orbiting surround it with same interval (every two seconds). Figure 5.11 shows one of the aerial photos of the 'White House'.

Figure 5.11: 'White House' near BK, TU Delft.

Figure 5.12 shows the difference in output point cloud generated by NARUX3D and Pix4Dmpper. It is clearly to see, Pix4Dmapper is the winner: Point cloud generated by NARUX3D doesn't cover any white surface at all. It can only preserve the 'skeleton' of the building as it may only recognize corners and seams between slabs. Pix4Dmapper, on the other hand, reconstructed all surfaces in white successfully and result point cloud reached a high level of detail.

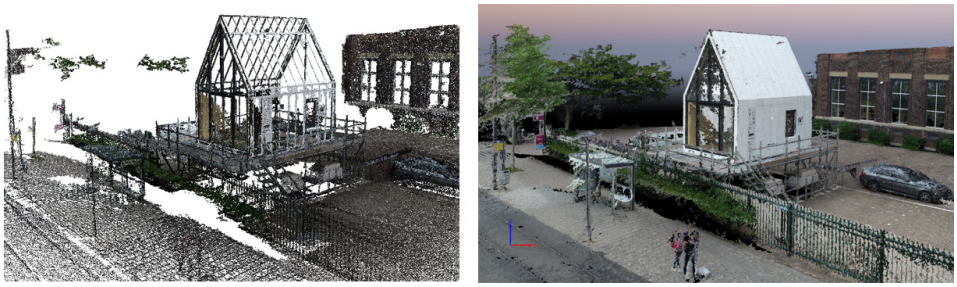


Figure 5.12: Comparison result. (Left side: Point cloud generated by NARUX3D; Right side: Point Cloud generated by Pix4Dmapper).

This failure case of NARUX3D reveals its disadvantage: As SIFT algorithm for image matching cannot deal with surfaces without distinctive feature, if target is in unified color or built with one material, its surface cannot be reconstructed. On the contrary, Pix4Dmapper is quite capable of dealing these cases. As Pix4Dmapper is a commercial software, it is not quite clear which algorithm(s) they use. For future improvement, we may try implementing semi-global matching and other algorithms to make our software more adaptive to difference situations.

5.3. Software Limitations

Our NARUX3D provides a good alternative to expensive commercial software, however it has several limitations:

1. As we mentioned in section 5.3 case 3, our software is not able to generate point cloud of surface without distinctive features.
2. NARUX3D can only reconstruct one building or object at one time, while some commercial software such as pix4D is able to reconstruct the whole scene. Because of that, when we tried to reconstruct Almere tower at first time, it failed. It is because that in NARUX3D Poisson reconstruction will take every point as one object, and we need to delete other redundant point cloud manually, leaving only point cloud of the tower (Figure 5.13). Only in this way could we get a good result.
3. In PolyFit, the input point cloud must be enclosed. More precisely, point cloud of the building could have small holes here and there but missing wall or roof cannot be accepted. For example(Figure 5.14) if a user use a cell phone to take image of a building but without getting any photos of the roof, NARUX3D can only generate point cloud of walls. In PolyFit as the roof of the building is missing, point cloud is regarded as 'unclosed' and cannot be used for piecewise planar reconstruction.

Figure 5.13: Almere tower point cloud (red: deleted points).

Figure 5.14: Example building captured by cellphone.

6

Conclusions and Future Work

The objective of this project was to develop an software NARUX3D to reconstruction 3D models from images. Considering the limited of time and resource, to achieve this purpose the project team organized all functions of NARUX3D into MoSCoW rules, and designed the interface of different model. At the same time, all team members went to Almere and Delft to collect data.

In the software implementation phase, NARUX3D generates point clouds from images first ,then reconstruct 3D models from the point cloud. In the first step, NARUX3D chose 3 algorithms to transform images into point clouds: SIFT, SfM, MVS, and use open source libraries to implantation: SiftGPU [1], Bundler [2] and PMVS [3]. In the second step, there were two options to generate 3D models: smooth surface building reconstruction or piecewise planar building reconstruction.

Software development demands perfect coordination of the different modules, by dividing modules and work separately we were able to break the code into parts, thus speeding up the process. We encountered a lot of issues with input-output handling (definition, same formats) To solving the issue we define the API of different models, and using version control systems such as GitHub to share and merge the code should be the main means of member communication. By working with GitHub branches, all members have access to the progress of each branch and the risk of changing accidentally a part of the code is minimized. In the end, the branches were merged into the master branch and the code was compiled and tested. Additionally, an adequate Understanding of the algorithms is needed, in order to navigate into the tree of scripts and libraries to find all the key components and create a functional code.

The final phase is software test and result evaluation. The project team took images from several different kinds of objects and NARUX3D was able to reconstruct a large variety of objects (i.e., building with different shapes and material) even while using handheld cameras and mobile phones. In some cases like the Almere Watchtower, NARUX3D performed better than commercial software when generation point cloud, as it filtered a large number of images that would yield inaccurate results. The robustness of the software is also good; a minimum RAM of 8 GB is needed in order to process large datasets. Concerning the accuracy and robustness, we would like to remind again that the 3D models are the output of a constructing technique that is of a very lower cost in terms of hardware requirements and the knowledge background of the team was limited.

Still there were some limits: NARUX3D was not able to generate point cloud of surface without distinctive features; can only reconstruct one building or object at one time; to implement PolyFit successfully, the input point cloud must be enclosed.

For the future work, if it is possible to use a Laser scanner to scan the building, and we will compare the laser scanner output with the point cloud and 3D model generated by NARUX3D,

[1] https://github.com/pitzer/SiftGPU/

[2] http://www.cs.cornell.edu/~snavely/bundler/

[3] https://github.com/pmoulon/CMVS-PMVS

test the accuracy of the algorithms. In addition, add some functions to the software, such as allowing users to define the parameters of the algorithms since the parameters depending on the characteristics of the input images and the desired output. A longer project like a thesis preparation could allow a reasonable selection of algorithms, and extensive field testing with different cameras or effort to reconstruct monuments from online pictures. Also, NARUX3D efficiency on business environment is something to be tested in the future.

A

User Interface

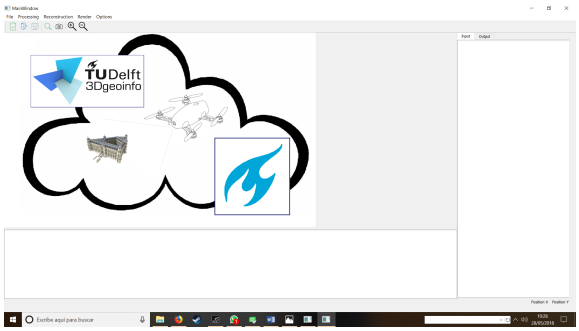


Figure A.1: Main Window

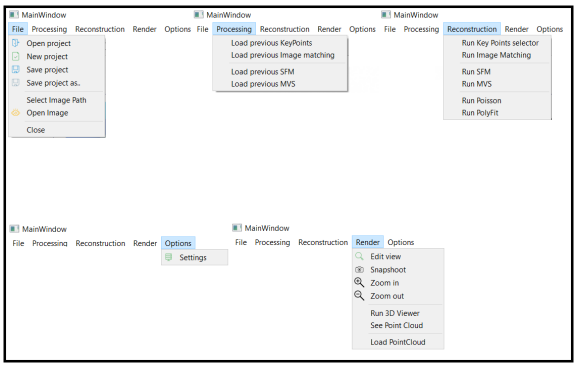


Figure A.2: Main Window Submenus

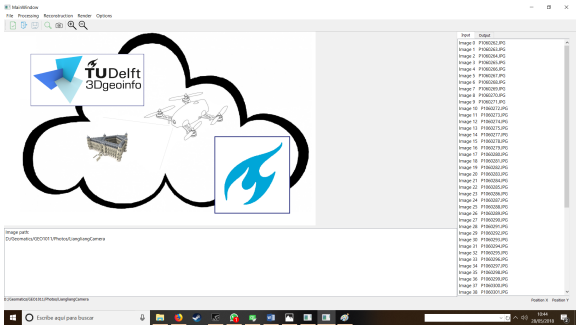


Figure A.3: Image Path selection

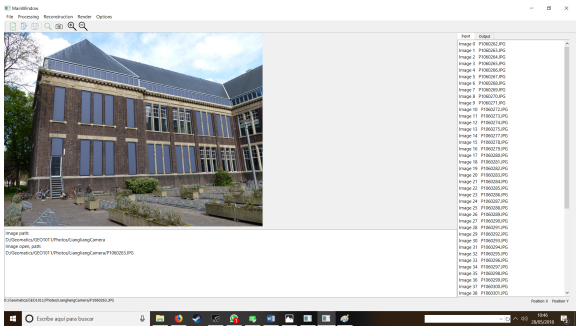


Figure A.4: Image Viewer

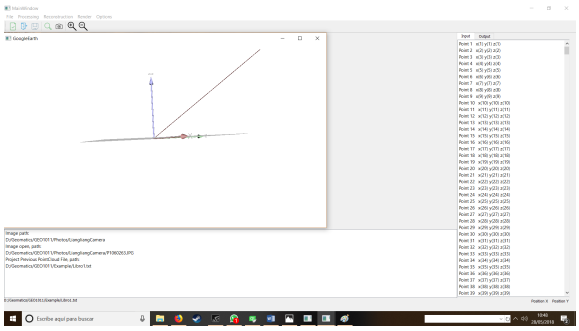


Figure A.5: PointCloud load

B

Media outreach

The purpose of this appendix is to inform society on the outcome of the Geomatics Synthesis Project “Digitizing Real-world Scenes from Images”.
The potential user of NARUX3D is someone who wants to reconstruct a real word 3D building from a set of overlap images which might be captured by a USA(drone) or a hand hold cameras or even a smartphone. To accomplish this purpose, a lot of open source third party libraries were included, for example siftGPU, Bundler, CMVS-PMVS, PoissonRecon.
After users generation point cloud from this software, they have two options to reconstruct 3D model: using Poisson reconstruction for smooth surface model, or save the point cloud as an independent file and use another open source software PolyFit[1] to build piecewise planar model. Poisson reconstruction is the best choice for smooth surface, and the input point cloud can be a part of buildings, also the result is more granular. And PolyFit performance better on piecewise planar building, and can only work with a point cloud which can generate solid model, in other words, a part of building is not enough.
For now we publish NARUX3D in github[2]. Since it is an open source application, anyone is able to access it by keyword search and use it without any charge. Our software gives potential users an alternative of expensive commercial software for example to reconstruct 3D buildings.

[1] https://github.com/LiangliangNan/PolyFit
[2] https://github.com/Natasja1992/3d_floriade

C

Poster

Digitizing Real-World Scenes from Images

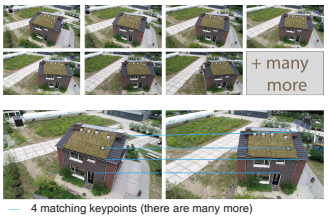
Takis Arapakis Natasja van Heerden Guillermo Rodriguez-Mon Barrera Qu Wang Xin Wang

Project description

3D computer models are a great way to visualize the real world and more and more applications are created to use these models in for example mobile apps, virtual reality and other applications. The Floriade project in Almere wants to be able to create a lightweight 3D model of their project, that different developers can use to realise all kinds of innotiative applications. To create these 3D models we have developed an open source software called **NARUX3D**, that creates 3D models from 2D images. With NARUX3D the people from Floriade will be able to capture the developement of their project in 3D on different moments in time.

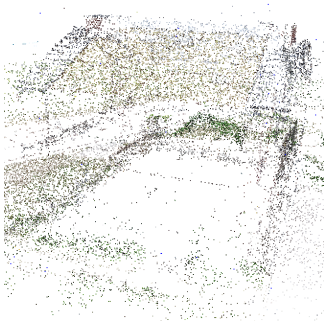
- Using images from handheld cameras
- Using images from UAS (drones)
- Creating point clouds in the computer
- Creating efficient 3D computer models
- The C/C++ programming language was used for creating the software
- An intuitive graphical user interface

Image Matching



Images that are taken with either a handheld camera or with a UAS (drone) will be matched via the Scale-Invariant Feature Transform (SIFT) algorithm. The result is matching key points between every two images.

Structure from Motion



With the output of the image matching a 3D sparse point cloud can be created. To do this, the Structure from Motion (SfM) method is used.

Multi-View Stereo

From the sparse point cloud, a dense point cloud is created. To do this a multi-view stereopsis (MVS)algorithm is used.

Surface Reconstruction

From the dense point cloud, surfaces are reconstructed. For straight building objects the polyfit method is used to fit planes to the point cloud. For more curved building objects the Poisson reconstruction method is used to create smooth surfaces.

Geomatics Synthesis Project

TU Delft Delft University of Technology

Bibliography

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. Freak: Fast retina keypoint. *Computer vision and pattern recognition (CVPR), 2012 IEEE conference*, 2012.

[2] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science*, 3951, 2006.

[3] F. Bethmann and T. Luhmann. Semi-global matching in object space. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2: pp. 807–814, 2015.

[4] W. R. Duncan. A guide to the project management body of knowledge. 1996.

[5] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 2010.

[6] H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference*, 2:pp. 807–814, 2005.

[7] D. W. James, J. Eckermann, F. Belblidia, and J. Sienz. Point cloud data from photogrammetry techniques to generate 3d geometry. *Proceedings of the 23rd UK Conference of the Association for Computational Mechanics in Engineering*, 2015.

[8] E. Karami, S. Prasad, and M. Shehata. Image matching using sift, surf, brief and orb: Performance comparison for distorted images. *arXiv preprint*, 2017.

[9] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. *SGP '06 Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.

[10] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. *Computer Vision (ICCV) 2011 IEEE International Conference on*, 2011.

[11] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004.

[12] L. Nan and P. Wonka. Polyfit: Polygonal surface reconstruction from point clouds. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. *Computer Vision (ICCV), 2011 IEEE international conference*, 2011.

[14] J. L. Schonberger and J. M. Frahm. Structure-from-motion revisited. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[15] C. Suo, D. Yang, and Y. Liu. comparing sift,surf,brisk orb and freak in some different perspectives. *Beijing Surveying and Mapping*, 2012.

[16] L. Yan, L. Fei, C. Chen, Z. Ye, and R. Zhu. A multi-view dense image matching method for high-resolution aerial imagery based on a graph network. *Remote Sensing*, 2016.