3D Harris Corner Detector

ASRAF ALI Abdul Salam Rasmi
LUKAMBA Deogratias
MACAULAY Sadiq

Masters in Computer Vision
University of Burgundy
January 6, 2019

Supervisor: **Prof. FOUGEROLLE Yohan**
VIBOT, IUT Le Creusot, UMR6306 CNRS/uB

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Nowadays, computer vision techniques are applied in various applications such as medical image analysis, 3D reconstruction, and those applications typically depend on features recognition [1]. The usage of 3D data is rapidly increasing due to its application in the automation industry. Due to the excessive amount of 3D data and availability of low-cost capture devices, extracting specific features from 3D data has become growing research field. By extracting such features it will be easy to process only selected features for specific applications (like object registration, mesh segmentation and simplification), instead of storing and processing whole 3D objects. For this reason, it is necessary to select distinctive points on the 3D model to keep the efficiency of the process applied to them [2].

Unlike 2D images finding interest points on the 3D model is a challenging task because of several reasons like interest point definition, protrusion of outstanding local structures, the topology of 3D meshes, etc [2]. The method proposed in this paper is a robust technique to find the interest points on 3D models using Harris Operator. Our work is to study and modify the proposed source code; also to render the 3D model (showing interest points) using OpenGL.

1.2 Objectives

The objective for undertaking this project is stated below.

- To understand and Implement a Research Article.

- To develop our programming skills in C++ (development of internal data structure, graphical user interface, 3D rendering, etc.)
- To develop transversal skills (project management, team working, report writing, presentation, etc.)

Chapter 2

Background

2.1 Overview

This project is about implementing an interest point detector on 3D meshes using Harris 3D operator. A 3D mesh is defined as a structural build of a 3D Model consisting of polygonal or triangular faces supported by an object file format (.off). That file is used to represent the geometry of a model by specifying the triangle of the model's faces, number of vertices and edges.

The implementation of the source code for Harris interest point detection proposed in [2] which form the basis of our project inputs a .off file meaning "object File Format" that has certain parameters in it to give an output of the interest points. The .off file consists of arrays of values that represent a set of vertices, edges, and faces of the 3D image it represents. The implementation of the interest point algorithm sterns from the computation of these sets of vertices represented in the .off file such that the features of the 3D images are expressed from the relation between each vertex v and its immediate neighboring vertices, v_k. We can use Harris operator to display the interest point by computing the local neighbors of each vertex. The process can be classified as follows,

- Compute **Local Neighborhoods** around a vertex.
- Find a local canonical system.
- Fit **Quadratic Surface** to the step of points.
- Compute derivatives on the fitted surface.

- Calculate **Harris Response** for each vertex by using calculated derivatives.
- Select the **Interest Point**
- **Display** the Interest Points on the 3D Model (**Rendering**)

2.2 Compute a Local Neighborhood

The paper presented three methods to compute neighborhoods: Spatial, Adaptive and Ring Neighborhood. Since the performance of Ring Neighborhood is faster in terms of computation and interesting when small neighbors are considered , we propose to use it. The idea is to select each vertex and for each vertex we search the immediate neighbors around the vertex called the first ring neighbors. The process follows by repeating the process to find the immediate neighbors of the neighbors of the previous ring.

The second step is to compute the canonical local system to display data in 2-D or 3-D for the best fitting plane, for instance, we use PCA which is the simplest technique for dimensionality reduction. It performs an orthogonal linear projection on the principal axis called the Principal components (eigenvector of the covariance matrix). Also, we need to perform the normalization of that fitting plane. For that, we select the lowest eigenvalue which corresponds to the eigenvector taken as the norm.

2.3 Fit the Quadratic Surface

We fit a quadratic surface on the eigenvector that presents obviously the lowest eigenvalue which means to the normalized neighborhood. This quadratic surface is computed by using the least square method to get the paraboloid of the six terms as shown in the function below:

$$z = f(x,y) = \frac{p_1}{2}x^2 + p_2 xy + \frac{p_3}{2}y^2 + p_4 x + p_5 y + p_6 \tag{2.1}$$

A quadratic surface is fitted to the transformed surface patch, and its derivatives are calculated to construct a 2×2 matrix E, elements of which involve the integration of Gaussian weighted derivatives in two directions. Hence, the following gives us the relationship between

derivatives and least square method:

$$A = \frac{1}{\sqrt{2\pi\sigma}} \int_{R^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} . f_x(x,y)^2 dxdy \tag{2.2}$$

$$B = \frac{1}{\sqrt{2\pi\sigma}} \int_{R^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} . f_y(x,y)^2 dxdy \tag{2.3}$$

$$C = \frac{1}{\sqrt{2\pi\sigma}} \int_{R^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} . f_x(x,y) f_y(x,y) dxdy \tag{2.4}$$

$$A = p_4^2 + 2p_1^2 + 2p_2^2 \tag{2.5}$$

$$B = p_5^2 + p_2^2 + p_3^2 \tag{2.6}$$

$$C = p_4 p_5 + 2p_1 p_2 + 2p_2 p_3 \tag{2.7}$$

$$E = \begin{pmatrix} A & C \\ C & B \end{pmatrix} \tag{2.8}$$

We limit the equation to second degree having six terms since it will provide us a good noise reduced performance and compactness. Furthermore, $f(x,y)$ is invariant to rotation and translation in x and y directions since the coefficients are adapted in good manner with different coordinate systems. Then we use the matrix to calculate the Harris response by considering a fixed fraction of points that has the largest response to be selected as interest points.

Chapter 3

Software Requirements

3.1 QT Creator

QT is a cross-platform software development tool which can run on various software and hardware configuration. It can run with GUI for various applications in embedded, computer and mobile platforms. Comparing to other Integrated Development Environment (IDE) QT has full features of C++, supports Object oriented Programming, has a simplified GUI editor and above all it is compatible with multiple the operating systems[3].

3.2 Standard Template Library

The C++ STL (Standard Template Library) is a powerful set of C++ template classes that provides general-purpose classes and functions with templates. STL implements many popular and commonly used algorithms and data structures like vectors, lists, queues and stacks [4]. The core of the C++ Standard Template Library has three well-structured components,

- **Containers** are used to manage collections of objects of a certain kind. There are several different types of containers like deque(double-ended queue), list, vector, map etc.
- **Algorithms** act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.
- **Iterators** Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

3.3 OpenGL

OpenGL (Open Graphics Library) is a standard application program interface (API) for defining 2-D and 3-D graphic images. OpenGL has an advantage that an application can be created the same effects in any operating system using any OpenGL-adhering graphics adapter. OpenGL specifies a set of "commands" or immediately executed functions. Each command directs a drawing action or causes special effects. A list of these commands can be created for repetitive effects. OpenGL is independent of the windowing characteristics of each operating system, but provides special "glue" routines for each operating system that enable OpenGL to work in that system's windowing environment. OpenGL comes with a large number of built-in capabilities requestable through the API. These include hidden surface removal, alpha blending (transparency), anti-aliasing , texture mapping, pixel operations, viewing and modeling transformations, and atmospheric effects (fog, smoke, and haze)[5].

3.4 GLUT - OpenGL Utility Toolkit

GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms. GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is *simple, easy, and small* [6].

3.5 LaTeX

LaTeX is basically a typing system that includes features for the preparation of Professional documents like Scientific Articles, Presentations and Presentations and more other important documentation. It is a high quality typesetting system that encourages the authors to be more specific on the contents rather than the appearance of the documents [7]. This report was written in LaTeX which gave us many advantages for instance writing the mathematical equations in LaTeX is much easier than writing it in other tools like Microsoft Office Word.

3.6 Git

Git is basically a control system useful in managing and sharing collaborative works, especially in the field of programming. Git comes with an advantage of tracking the history of entire changes made to the codes which makes it more secured. It is very optimized as it allows to commit changes, create branches, merge codes, and compares the codes with the previous section. It supports a variety of nonlinear development work flows, henceforth it is more flexible [8].

GitHub is a web based Git repository hosting service. In this project GitHub was used to store all the data in a cloud which can be easily accessed by any developer. GitHub holds all data in Git format from where it can be extracted with Git tools. This made it an effective means of communication for this project and provides the possibility of collaborative work.

Chapter 4

Project Management

4.1 Task Segregation

As explained in the methodology, the 3D Harris Corner Detector has various steps in computation of interest points but not all the process can be separately programmed and integrated later. Due to this reason we divided the task into three sections,

- Load a mesh and Compute Harris Response.
- Rendering a 3D Model and Display the vertices as points.
- Create a User Interface and show Interest Points on the 3D Model.

To start we divided the task among ourselves as mentioned above but then we decided to concentrate on the first two tasks before we go for User Interface since loading the mesh and computing response was a challenging task.

4.2 Weekly Meeting

We managed to share the tasks equally so that all the batch members are either learning new concepts or teaching new concepts. We had periodic weekly meetings in order to make sure every one are currently updated with what we learned new that week. The below shown is the summary of weekly meetings held.

Week	Date	Task Completed	By Whom
1	13/10/2018	Project Planning	Deo, Sadiq, Abdul
2	27/10/2018	Configuration and installations of necessary libraries on Ubuntu operating system to run the source code	Sadiq, Abdul
3	3/11/2018	Understanding the Implementation of source code Implementation of source code in QT Environment	Deo Sadiq
4	10/11/2018	Mid Term Report	Abdul, Deo, Sadiq
5	17/11/2018	Loaded a Mesh successfully into a Class Vector and displayed the mesh in Console	Abdul, Deo
6	1/12/2018	Rendered a triangular mesh using structures and dynamic arrays implementation with OpenGL	Sadiq
7	8/12/2018	Rewrote the Rendering Algorithm using Class and Standard Template Library (from the previous code written by Sadiq)	Deo, Abdul
8	15/12/2018	**EXAMINATION WEEK**	-
9	22/12/2018	Developed the Users Interface to display the 3D model with users interactions (using glut)	Sadiq
10	28/12/2018	Computed the neighborhoods of the vertices.	Abdul, Deo
11	4/1/2019	Finalized the Codes and Report	Abdul, Deo, Sadiq

Chapter 5

Implementation

5.1 Brief

The project has been developed and implemented on the Qt5 C++ IDE first on Linux Operating System. Thanks to the development of the QT IDE to be a cross-platform among different operating system. The task required for the implementation of this project includes the following,

- Configuration of the Qt5 with third party libraries.
- Loading the mesh.
- Rendering of the mesh.
- Computing the Neighborhood of the vertices.

The whole processes for the implementation involve work around the data set of a $.off$ as an input file which contains a number of vertex points, faces, and edges that represents a 3D model. We have computed the data of this file at different stages to obtain the resultant output as we expect. It is also very important to note that the successful implementation of the project was dependent on the use of some necessary C++ libraries including the OpenGL and GLUT together with some default libraries on the Qt5 IDE.

5.2 Configuring Qt5

Basically, we need the OpenGL libraries for rendering the mesh and a GLUT library that provides an interface with the OpenGL library to display the mesh. The direction on the process of installation of both libraries is provided in the web link [9], [10].

A *.pro* executable file is created by default in every new project folder on the qt. The process of adding the third-party libraries to the Qt5 takes a simple command line in the project file which includes adding the path to the directory of the already installed libraries for the linker as follows.

```
LIBS += -L/usr/include/ -lglut -lGLU -lGL
```

5.3 Loading a Mesh

At this point we need to read the group of data in the input file (a *.off* file format) for our computation. It is very important to first confirm the file format before considering reading through the data. The first line of the *.off* file always contains a label for the file written *OFF* this must first be checked. The second line of the file contains a summary of the file including the total number of Vertices, Faces, and Edges these values provide are read and store in private attributes of the class **Mesh**. Following are set of float values that represent the vertex points up to the total number of vertices, each line of the vertex point has values separated with blank spaces corresponding to the x, y, z coordinates of the vertex the data of each vertex is read line after line and stored in a vector class (Mesh) with the default constructor that takes in there float inputs values. The same approach was employed to read the face data which is the next set of data after the vertex which consist of the first value on each line that indicates the type of face data (3 == triangular faces and 4 == quadratic face) but for our computation, we have considered that the faces are triangular faces. for a triangular face, only the first three column values are read from the file up to the number of faces, each of the row values represents the index values of the vertices. Storing the data in vector class containers provide the easy and efficient reading of the data in their respective containers for consequent computation.

5.4 Computing the Neighborhood

The Algorithm for computing Neighborhood starts with finding the adjacent faces of each vertices. As explained earlier, the *.off* file has both vertices and faces. As we loaded the Mesh, we just need to access the containers directly. Starting from vertex $'0'$, we check through all the vertices and for the adjacent vertices of vertex $'0'$ and we store it in a vector (sorted and duplicates are removed). The process is repeated for $'N'$ vertices. After getting

adjacent vertices, now our target is to find the neighbor vertices of each vertex repeating the iterations equal to ring size. Finally we will get $'n'$ neighborhood points of each vertex.

5.5 Rendering

The OpenGL is required to draw our 3D model to be displayed in the GUI. for proper rendering of the model, it was necessary that we have to compute the normals of the mesh which was calculated as follows,

- Compute the center of gravity of the mesh and subtracting it from each vertex

$$C = \sum_{i}^{N} \frac{V_i}{N}$$

  This allows us to be able to render the model at the center.

- The data obtained from the computation of the center of gravity and biggest absolute coordinate of the vertices the were used to scale the vertex data to obtain a model of the same size for every mesh at the center of the mesh.

- The normal at each triangle are computed and then averaged at each vertex

$$N_v = \frac{\sum_{i}^{N} N_i}{||\sum_{i}^{N} N_i||}$$

  and computing the cross product between the edges of the triangle:

$$N_i = \frac{cross(e_0, e_1)}{||cross(e_0, e_1)||}$$

- Now we are able to pass all the necessary computations to OpenGL functions for the drawing of the mesh in the *DrawMesh()* and also to highlight the vertices required "Interest points" on the 3D model.

5.6 Display 3D Model

The GLUT libraries provide a users' interface with the OpenGL library. The interface we have created includes different computations that give interactive properties to the user by initializing the color and texture properties to the mesh by using specific OpenGL functions. The implementation gives room for basic user interactions such as zooming the model (with key $'i'$ and $'o'$), translating the model(arrow keys), and rotation (key $'j'$ and $'l'$) among other function keys for animations, enable glPolygonmode, and light.

Chapter 6

Results and Discussion

Through the development of all the required tasks for the completion of the project, as it has been already discussed in the previous sections, we have been able to arrive at the point of efficiently implementing the load and render 3D mesh concepts in C++ Programming with some required libraries, as well develop an interactive graphic user Interface for displaying 3D models. A description of the processes involved is described below.

6.1 Reading the Data file

The Mesh class we have created provides the implementations that read through the required input file and stores the necessary data into a *vector<class>* container in the form of a matrix which allows easy access to the data for other computations. The class Definition is as follows.

```
class Mesh
{
private:

    int nb_vertices;
    int nb_faces;
    int nb_edges;
    float x,y,z;
    float v_max;
}
```

```
3 0 1 16 0.729412 0.556863 0.898039
3 0 16 15 0.729412 0.556863 0.898039
3 1 2 17 0.729412 0.556863 0.898039
3 1 17 16 0.729412 0.556863 0.898039
3 2 3 18 0.729412 0.556863 0.898039
3 2 18 17 0.729412 0.556863 0.898039
3 3 4 19 0.729412 0.556863 0.898039
3 3 19 18 0.729412 0.556863 0.898039
3 4 5 20 0.729412 0.556863 0.898039
3 4 20 19 0.729412 0.556863 0.898039
3 5 6 21 0.729412 0.556863 0.898039
3 5 21 20 0.729412 0.556863 0.898039
3 6 7 22 0.729412 0.556863 0.898039
3 6 22 21 0.729412 0.556863 0.898039
3 7 8 23 0.729412 0.556863 0.898039
3 7 23 22 0.729412 0.556863 0.898039
3 8 9 24 0.729412 0.556863 0.898039
3 8 24 23 0.729412 0.556863 0.898039
3 9 10 25 0.729412 0.556863 0.898039
3 9 25 24 0.729412 0.556863 0.898039
3 10 11 26 0.729412 0.556863 0.898039
3 10 26 25 0.729412 0.556863 0.898039
3 11 12 27 0.729412 0.556863 0.898039
3 11 27 26 0.729412 0.556863 0.898039
3 12 13 28 0.729412 0.556863 0.898039
3 12 28 27 0.729412 0.556863 0.898039
3 13 14 29 0.729412 0.556863 0.898039
3 13 29 28 0.729412 0.556863 0.898039
3 15 16 31 0.729412 0.556863 0.898039
3 15 31 30 0.729412 0.556863 0.898039
```

(a) Face Data of $seashell.off$

```
Number of vertices is: 915
Number of faces is: 1680
Number of edges is: 5040

0, 1, 16
0, 16, 15
1, 2, 17
1, 17, 16
2, 3, 18
2, 18, 17
3, 4, 19
3, 19, 18
4, 5, 20
4, 20, 19
5, 6, 21
5, 21, 20
6, 7, 22
6, 22, 21
7, 8, 23
7, 23, 22
8, 9, 24
8, 24, 23
```

(b) Read Data (only second column to fourth column required)

Figure 6.1: Reading Face Data

```
OFF
915 1680 5040
-0.440800 -0.010667 0.018667
-0.280800 0.021333 0.018667
-0.144800 0.069333 0.018667
-0.048800 0.109333 0.018667
0.063200 0.157333 0.018667
0.135200 0.197333 0.018667
0.247200 0.253333 0.018667
0.327200 0.261333 0.018667
0.391200 0.245333 0.018667
0.439200 0.213333 0.018667
0.447200 0.157333 0.018667
0.439200 0.101333 0.018667
0.415200 0.061333 0.018667
0.343200 0.013333 0.018667
0.247200 -0.018667 0.018667
```

(a) Vertex Data of $seashell.off$

```
Number of vertices is: 915
Number of faces is: 1680
Number of edges is: 5040

-0.4408, -0.010667, 0.018667
-0.2808, 0.021333, 0.018667
-0.1448, 0.069333, 0.018667
-0.0488, 0.109333, 0.018667
0.0632, 0.157333, 0.018667
0.1352, 0.197333, 0.018667
0.2472, 0.253333, 0.018667
0.3272, 0.261333, 0.018667
0.3912, 0.245333, 0.018667
0.4392, 0.213333, 0.018667
0.4472, 0.157333, 0.018667
0.4392, 0.101333, 0.018667
0.4152, 0.061333, 0.018667
0.3432, 0.013333, 0.018667
0.2472, -0.018667, 0.018667
-0.440907, -0.01116, 0.016228
-0.28304, 0.018868, 0.006471
-0.148853, 0.06391, -0.008164
```

(b) Read Data

Figure 6.2: Reading Vertices (Points) Data

6.2 Drawing the Mesh

After these data were successfully loaded for use, follows the computation for rendering and displaying the mesh, which produced the following output results.

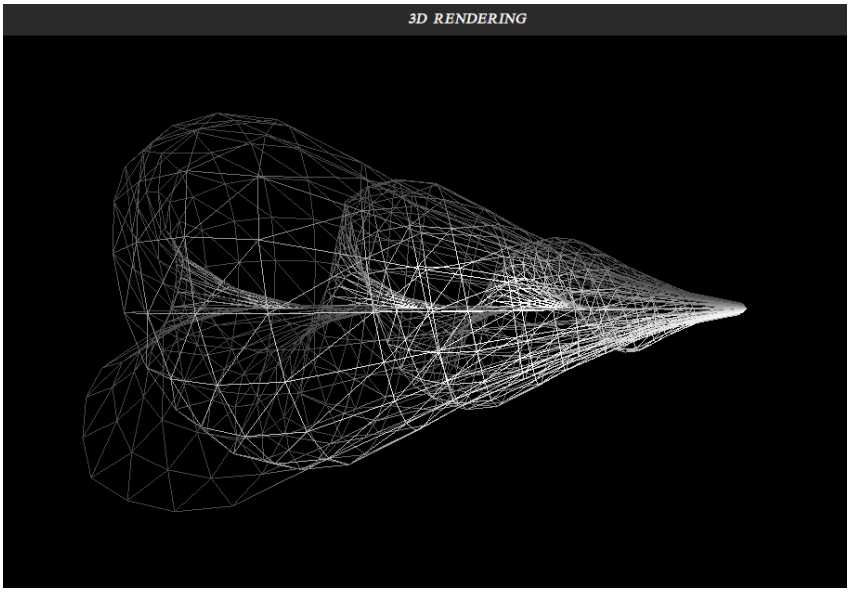

Figure 6.3: Rendering a 3D Model using OpenGL

6.3 Interest Point Detection

Consequently, as the part of the project, it is required to display the Interest points of the 3D model, an algorithm was implemented to highlight all vertices of the 3D model. After completing the implementation of the 3D interest point detection, the vertices can now be selected to show only the interest points of the 3D model, however, due to time restrictions, we couldn't fully implement the algorithm for Harris corners detection on the 3D model the current output of our model remains as follows.

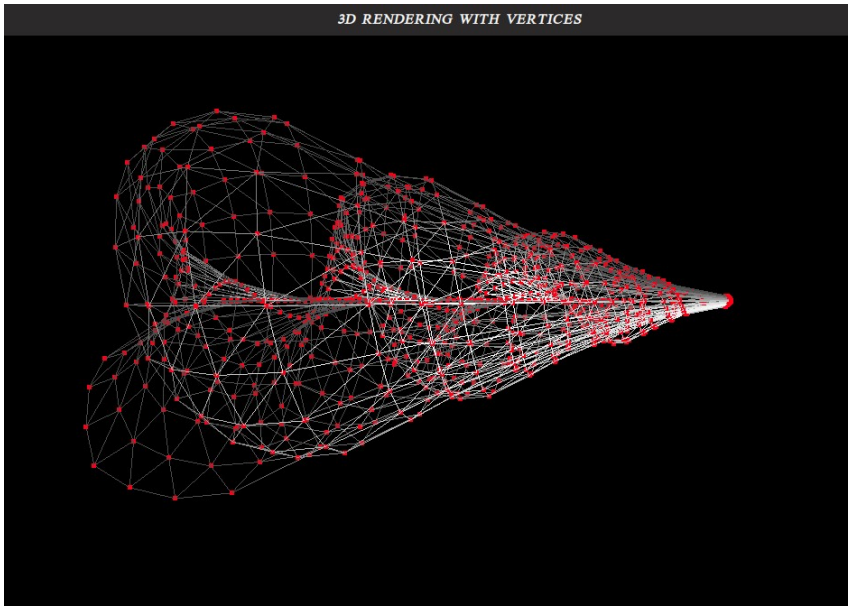

Figure 6.4: 3D Rendering with Vertices

Chapter 7

Conclusion

7.1 Conclusion

The major tasks of this project is loading a mesh and rendering a 3D model. We used STL implementation for both loading a mesh and rendering. Due to time restrictions and work load we were not able to do the calculation part of Harris Response. We are aiming to do the rest during holidays for better understanding on how to deal with meshes which forms the basics of 3D reconstruction.

7.2 What we have learned...?

The objectives of doing this project was clearly stated at the beginning and we have achieved most of them. The learning outcomes by doing this project is stated below.

- As beginners in C++ Programming language, all of us developed our programming skills, specifically in **Object Oriented Programming** and **Graphical User Interface**.
- We understood the schema behind installation and configuration of **Third party Libraries** and setting-up a **Programming Environment** (Qt).
- We learned about **Meshes** which is a good start to learn about 3D reconstruction and its application.
- We learned about how to understand **Research articles and journals** and how to undergo a **Research Implementation**.

- We developed our skills in **management of tasks**, **team work** and **sharing knowledge** effectively.
- By now we are good in using **LaTeX Editor** for Report writing and other documentations.

Bibliography

[1] I. Agnaou and B. Bouikhalene, "New approach for 3d object forms detection using a new algorithm of susan descriptor," *Int. Arab J. Inf. Technol.*, vol. 14, pp. 614–623, 2017.

[2] I. Sipiran and B. Bustos, "A Robust 3D Interest Points Detector Based on Harris Operator," in *Eurographics Workshop on 3D Object Retrieval* (M. Daoudi and T. Schreck, eds.), The Eurographics Association, 2010.

[3] "C++ gui programming with qt 4 > a brief history of qt : Safari books online." `https://my.safaribooksonline.com/0131872494/pref04`. Accessed: 2018-12-09.

[4] "C++ stl tutorial." `https://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm`. Accessed: 2018-12-09.

[5] "Opengl coding resources." `https://www.opengl.org/resources/`. Accessed: 2018-12-09.

[6] "Glut - the opengl utility toolkit." `https://www.opengl.org/resources/libraries/glut/`. Accessed: 2018-12-09.

[7] "Latex - a document preparation." `https://www.latex-project.org/`. Accessed: 2018-12-02.

[8] "What is git: become a pro at git with this guide | atlassian git tutorial." `https://www.atlassian.com/git/tutorials/what-is-git`. Accessed: 2018-12-09.

[9] "Opengl wiki." `https://www.khronos.org/opengl/wiki/`. Accessed: 2019-01-02.

[10] "The freeglut project :: Installation instructions." `http://freeglut.sourceforge.net/docs/install.php`. Accessed: 2019-01-02.