Université de Bourgogne

Software Engineering

Project Report

Implementation of Robust Harris Interest Point Detection on 3D Meshes

Authors:
Tewodros W. AREGA
Vamshi KODIPAKA
Hardiksinh PARMAR

Supervisor:
Prof. Yohan FOUGEROLLE

January 6, 2019

Contents

1 Introduction

1.1 Overview

The paper that we have implemented is titled, "Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes". The paper was published by Ivan Sipiran and Benjamin Bustos in 2011. In the paper, they used an effective and efficient extension of the Harris operator for 3D meshes. [1]

In this project, we implemented the paper using C++ programming language in QT IDE. The project is divided into four modules: Harris value calculation, interest point detection, 3D rendering and graphical user interface. Finally, we tested the system for 3D meshes by changing different parameters like ring size and constant fraction.

1.2 Objective

The main objective of the project is to implement the research paper. The specific objectives of the project are to determine the interest points of the 3D mesh, to render the 3D mesh using OpenGL and develop the graphical user interface.

2 Methodology

2.1 Tools Used

We have used the following tools to implement the project. The programming language is C++, the IDE (Integrated Development Environment) is QT creator and the operating system is Windows 10. To render the 3D mesh, we used OpenGL version 2.0. We also used Eigen Library[6] to handle the matrix operations and linear algebra.

2.2 Modules

We have divided the project into four modules. Here are the modules of the project:

2.2.1 Internal data structure of the 3D mesh

A 3D mesh is represented as a set of vertices and a set of faces.

A **vertex** in 3D is represented by its 3 coordinate positions. Here is the data structure of vertex:

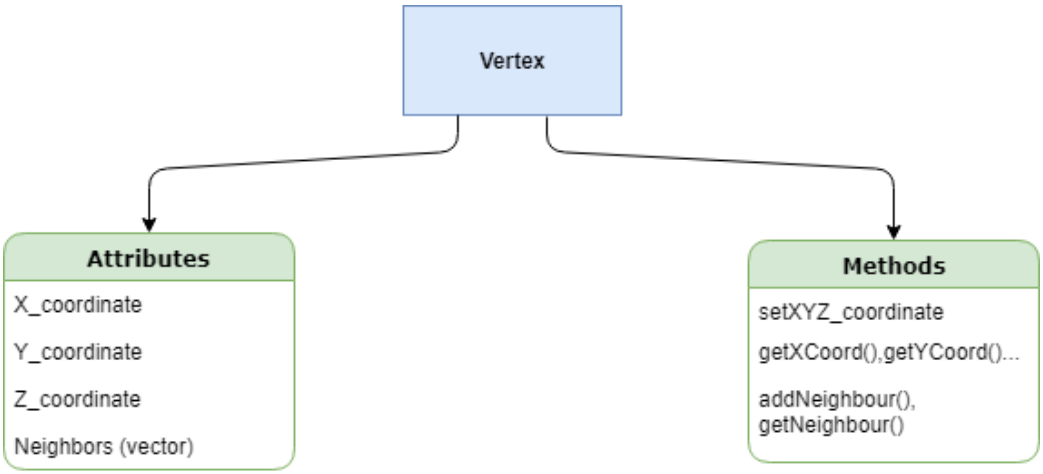


Figure 1: Vertex data structure

As you can see from **figure 1**, the vertex class has attributes like *xcoordinate*, *ycoordinate* and *zcoordinate* which are type double whereas *neighbour*

attribute is vector of integers that stores index of the neighbours of the vertex.

In methods of vertex class, we have *setxyz* function which sets the values of x,y and z coordinate. The methods *getXCoord, getYCoord, getZCoord* are used to retrieve the values of x,y and z coordinates respectively. Whereas *addNeighbour* and *getNeighbour* functions add new neighbour and return the set of all neighbours.

A face is a flat or curved surface on a 3D mesh which is made up of 3 vertices. Here is the data structure of face:

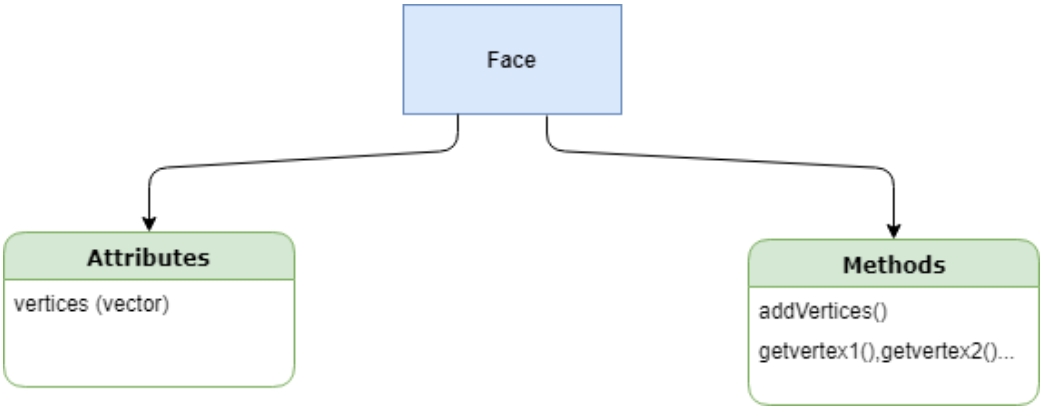


Figure 2: Face data structure

As you can see from*figure 2*, the face class has an attribute *vertices* which is a vector of integers which stores the index of the three vertices of the face. It has also methods *addVertex, getvertex1, getvertex2,getvertex3* which pushes vertex to the *vertices* vector and returns the index of the vertex respectively.

The mesh that is used for this project is a triangle mesh. Triangle mesh is a type of polygon mesh that comprises a set of triangles (typically in three dimensions) that are connected by their common edges or corners.[9]

The mesh class has attributes which are **vector of vertices** and **vector of faces**.

2.2.2 Determining the Harris Operator Value

The main processes needed to determine the Harris value are summarized in the diagram below:

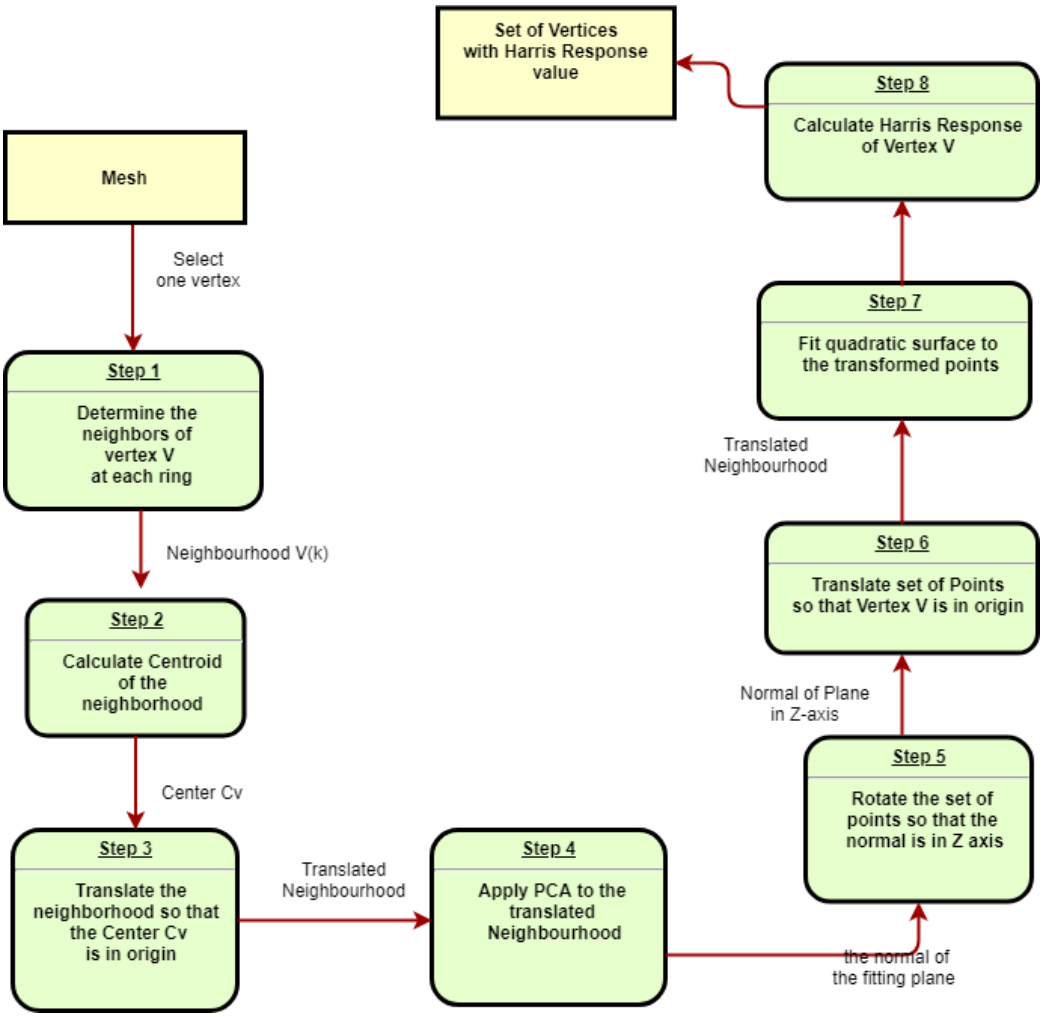


Figure 3: Summary of Harris value calculation

To determine the Harris value of each vertex, the first step is calculating the neighbours of each vertex. The neighbours can be direct or neighbours at different ring size. Here is one typical example:

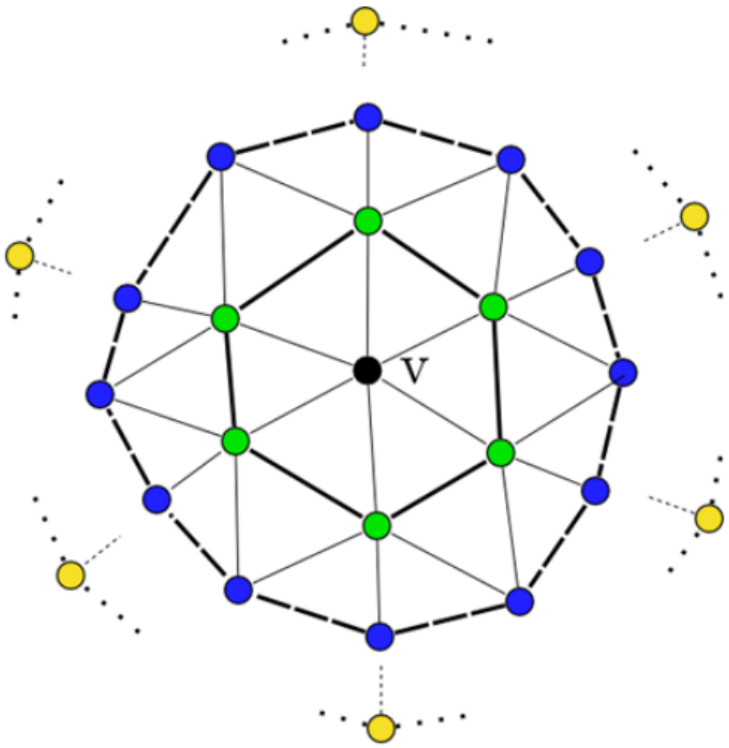


Figure 4: A vertex V and its neighbours - Image taken from [1]

To calculate the neighbours of vertex, we used two algorithms. The first one determines the direct neighbours or the neighbours at ring one(green vertices). The second one determines the neighbours at rings greater than one.

Algorithm 1: Get Direct Neighbours of a Vertex

Require: i index of the vertex, faces vector of faces

Ensure: Set of the neighbours

```
N ← ∅
Begin:
        for j ← 0  to |faces|
            if  faces[j] has index i, then
                 insert the remaining indices to set N
            end if
        end for
       return N
End
```

Figure 5: Algorithm to Get Direct Neighbours

Algorithm 2: Get Neighbours at ring N

Require: i index of the vertex, faces vector of faces, M set of direct Neighbours,

N ring number

Ensure: Set of the neighbours at ring N

```
SetN(0) ← i
SetN(1) ← M(i)
Begin:
        for j ← 2  to N
            SetN(j) ←  M(SetN(j-1)) + SetN(j-1) + SetN(j-2)
            SetN(j-2) ← SetN(j-1)
            SetN(j-1) ← SetN(j)
         end for
       return SetN(j-1)
End
```

Figure 6: Algorithm to Get Neighbours at Ring N

The first algorithm basically checks for the index of the vertex in each face and if it is found, the remaining vertices are added to the set. Since we used set STL(Standard Template Library), it doesn't store duplicate indices. Finally the algorithm returns the set of direct neighbours.

For the second algorithm[3], it has an additional inputs like ring number and set of direct neighbours of the vertices. To get the neighbours at ring N, we first get the direct neighbours of each vertices at ring N-1. Then it is subtracted by all the sets of neighbours in previous two rings.

The next step is calculation of centeriod of the neighbourhood. The coordinates of the centroid are found by averaging the x-, y- and z coordinates of the vertices of the neighbourhood. Then we translated the points so that the centroid is in origin of the coordinate system.

When Applying PCA(Principal Component Analysis), which is used to rotate the normal of the surface in z-direction, to the translated neighbourhood, we used Eigen Library to handle the matrices and calculate the eigenvalues and eigenvectors. We choose Eigen Library over the other libraries, because it uses template library which is easy to use and very fast.

One of the main processes in determining the interest points is the calculation of Harris Response of the vertex. To calculate that, first we fit a quadratic surface to the transformed points.

$$z = f(x, y) = \frac{p_1}{2}x^2 + p_2xy + \frac{p_3}{2}y^2 + p_4x + p_5y + p_6$$

$$A = p_4^2 + 2p_1^2 + 2p_2^2,$$
$$B = p_5^2 + 2p_2^2 + 2p_3^2,$$
$$C = p_4p_5 + 2p_1p_2 + 2p_2p_3.$$
$$E = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$$
$$h(x, y) = \det(E) - k\big(\mathrm{tr}(E)\big)^2 \quad \text{(Harris Operator Value)}$$

Figure 7: Equations of used to fit Quadratic Surface - Image taken from [1]

We used the linear least square method to fit the surface. From the Eigen Library, we used *"colPivHouseholderQr().solve()"* method to solve the linear system. After that we calculated the Harris operator value using the formula specified in the paper.

2.2.3 Selecting Interest Points

We have summarized the interest point selection process in the following figure.

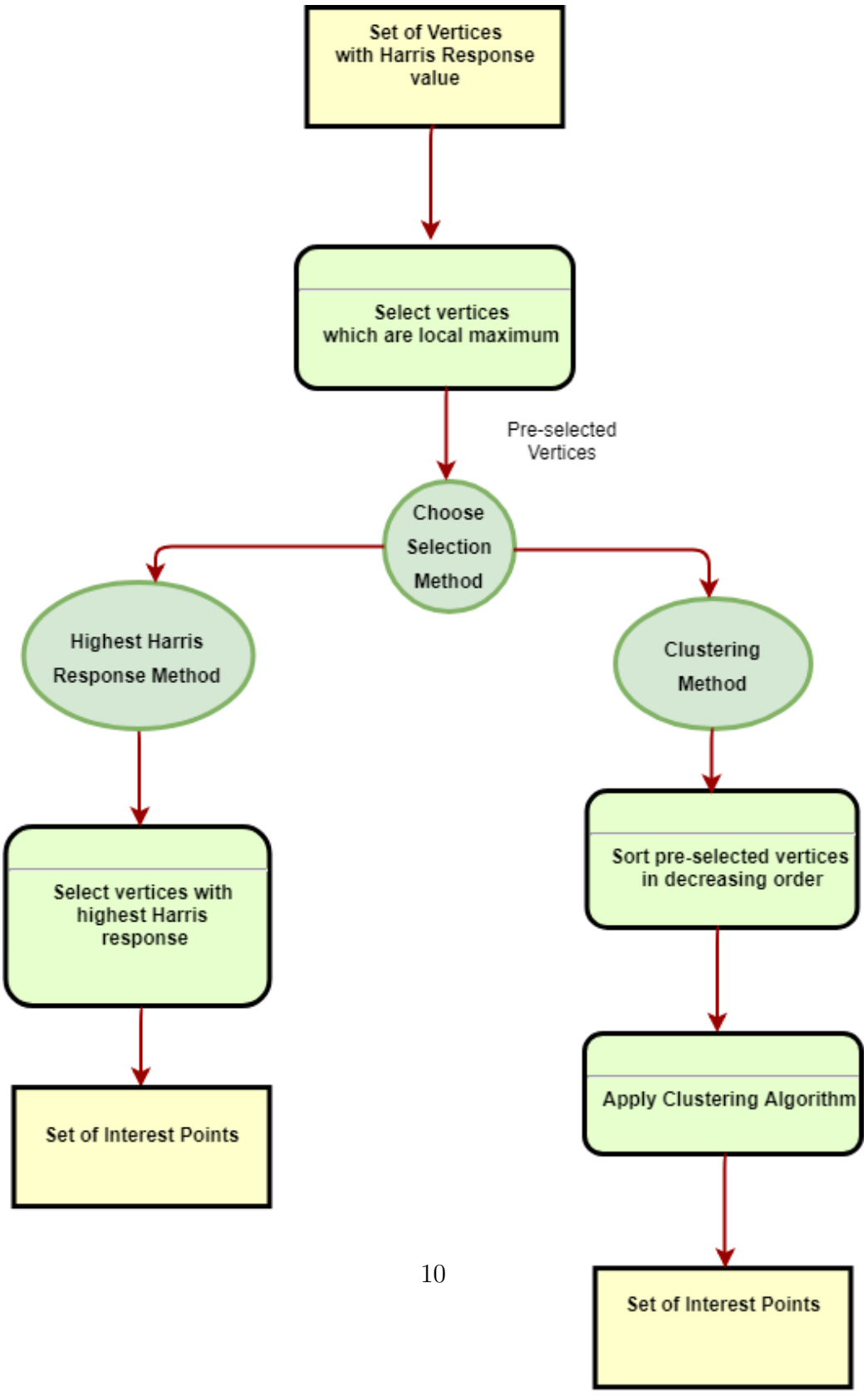


Figure 8: Summary of Interest Point Selection

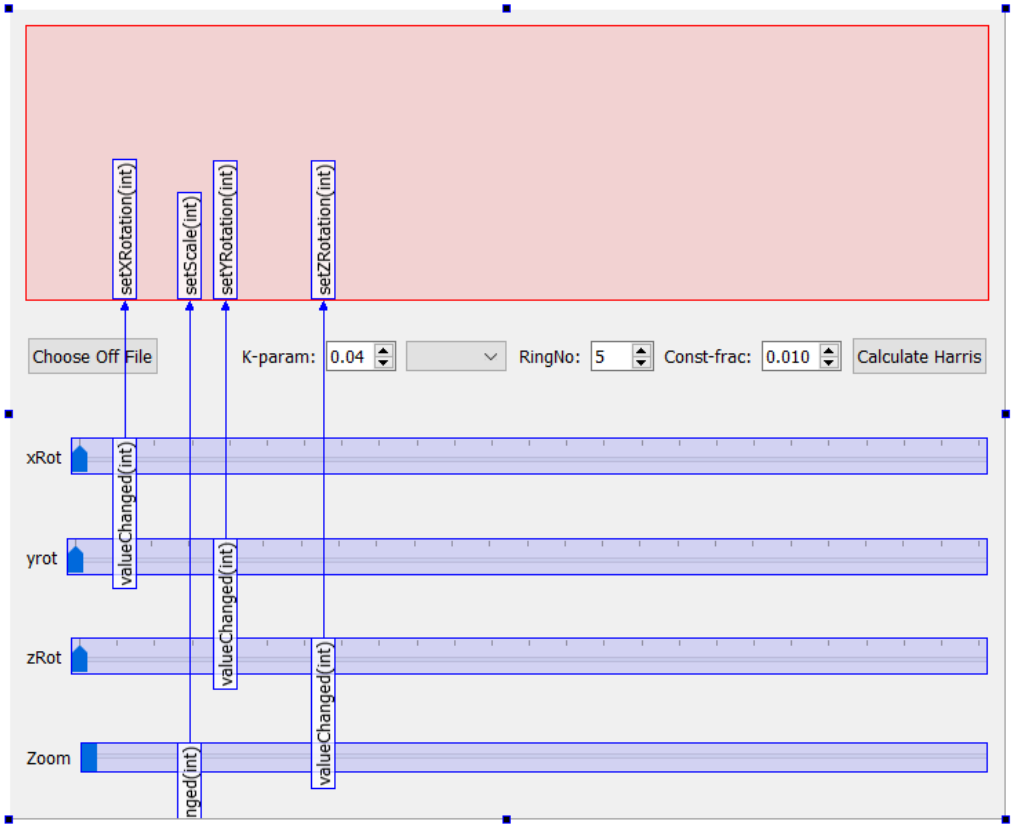
2.2.4 Graphical User Interface and 3D Rendering

In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another. Signals and slots are used for communication between objects.[4]

A signal is emitted when a particular event occurs. A slot is a function that is called in response to a particular signal.Signals and slots are loosely coupled: A class which emits a signal neither knows nor cares which slots receive the signal. Qtś signals and slots mechanism ensures that if you connect a signal to a slot, the slot will be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type.[4]

Qt provides the QGLWidget class to enable OpenGL graphics to be rendered within a standard application user interface. By sub-classing this class, and providing re-implementations of event handler functions, 3D scenes can be displayed on widgets that can be placed in layouts, connected to other objects using signals and slots, and manipulated like any other widget.[4]

In our GUI, we have six signals and six slots as you can see from the figure below. Some of them are predefined and some of them are user-defined signals and slots. While implementing GUI and 3D rendering, we used H. Kongś Opengl tutorial[2] as a starter.



Sender	Signal	Receiver	Slot
zoomSlider	valueChanged(int)	myGLWidget	setScale(int)
rotZSlider	valueChanged(int)	myGLWidget	setZRotation(int)
rotYSlider	valueChanged(int)	myGLWidget	setYRotation(int)
rotXSlider	valueChanged(int)	myGLWidget	setXRotation(int)

Figure 9: Signals and Slots

In zooming, when the slider moves, it emits a signal called *valueChanged(int)*. This signal is one of the many predefined signals, which stores the position of the slider. In response to the signal, a slot (*setScale()*) is called. Then this function updates the scale value and redraws the shape of the 3D mesh on the *glwidget* using the widgetś *updateGL()* method.

In rotation, it is activated by two events. The first one is mouse press and move and the second one is slider movement. When a mouse is pressed and moves inside *glwidget*, the *mouseMoveEvent()* is called and it sets the angle of rotation as well as it emits a signal called *xRotationChanged()* to update the position of x,y and z sliders. In response to the signal, a predefined slot (setValue()) is called and it changes the position of the slider. The second event is the slider movement which is similar to zooming. When the slider moves, it sends a predefined signal called *valueChanged()*. This method has the value of the slider. Then a slot function(*setXRotation()*) is called in repsonse to the signal. This functions sets the rotation angle and updates the *glwidget* using the widget's *updateGL()* method.

For the buttons, we used the predifened signal called *clicked()*. When the button is clicked it emits the signal and then the slot *(onButtonClicked())* is called and executes user-defined instructions.

In rendering part, we added the three basic rendering functions to the glwidget class. The first method is *initializeGL()* which intializes OpenGL to render the 3D mesh by defining colors and materials, enabling and disabling certain rendering flags, and etc.We also used it to change material color and to enable automatic normalization of normals while zooming the object. Whereas *resizeGL()*, is used to resize the viewport. It basically ensures that the OpenGL implementation renders the 3D mesh onto a viewport that matches the size of the widget, using the correct transformation from 3D coordinates to 2D viewport coordinates.[5] The function *paintGL()* performs painting operations using OpenGL calls.

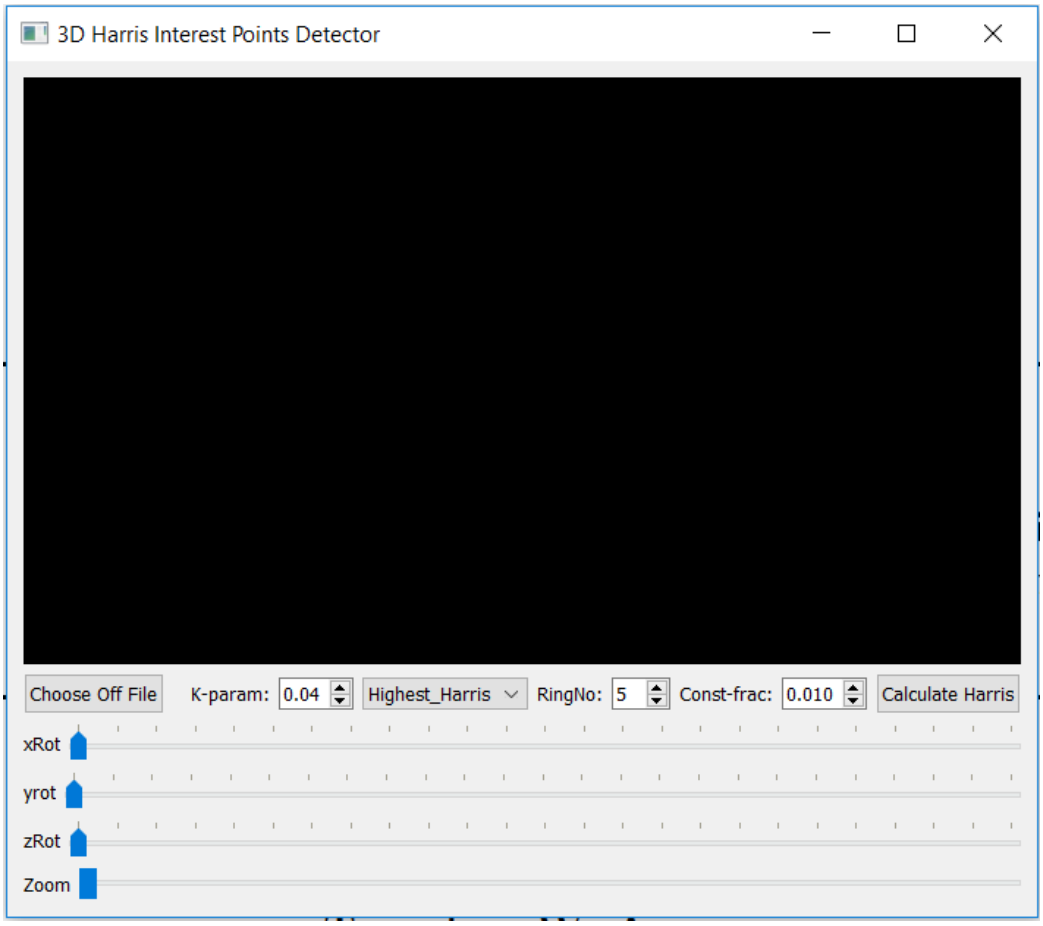


Figure 10: Graphical User Interface

The graphical user interface of our system looks like the figure above. The first button which is "Choose Off" button, is used to choose off file from the computer. The next four spinboxes and comboboxes are used to vary the value of the parameters which are K-parameter, Interest Point selection, Ring Size and Constant-Fraction. Finally the "Calculate Harris" button is used to calculate and display the 3D shape selected. The four sliders below them are used to zoom in and out and rotate the shape in x, y and z directions.

3 Results and Discussion

3.1 Results

After implementing the system, we tested the interest points of two shapes[7, 8] by changing the parameters. The first shape is centaur and the second one is human. The parameters that we have changed are the number of rings, the method of selection for interest points, constant fraction and K parameter. For the number of rings, we varied the number starting from 3 up-to 10. In the method of selection for interest points, we used both highest Harris value and clustering. For constant fraction, we varied the values starting from 0.001 to 0.01. For the K parameter, we varied the values in the range of 0.04 and 0.1.

Here are the results:

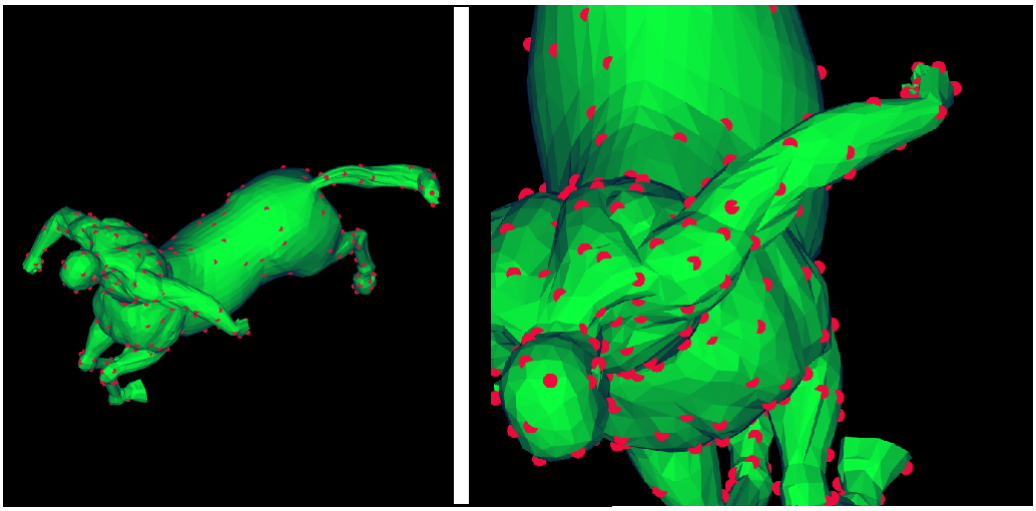


Figure 11: Clustering Method(left) and Zoomed Version(right)

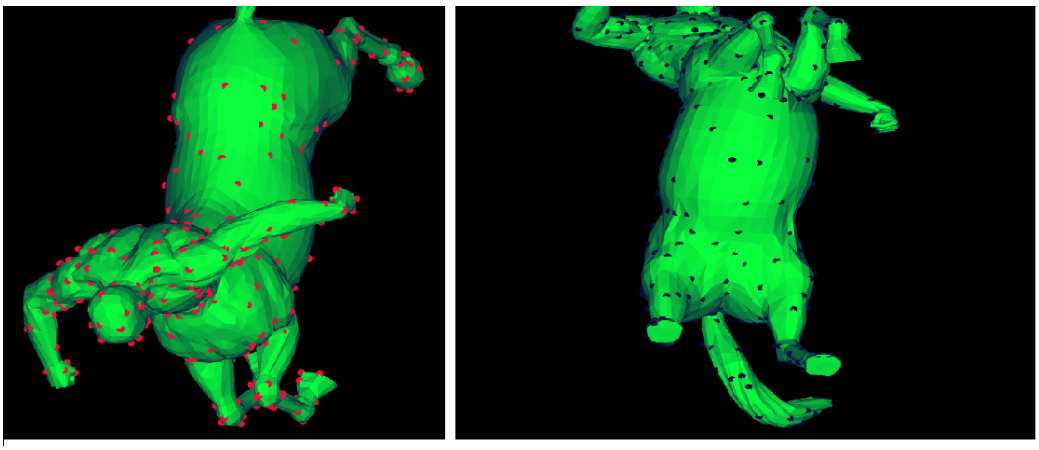


Figure 12: Clustering Method from different Views

In figure 11 and 12, there are shapes of Centaur with different views and scaling. The value of the parameters are K-parameter:0.04, Selection:Clustering, Ring Size:5, Constant-fraction:0.01.

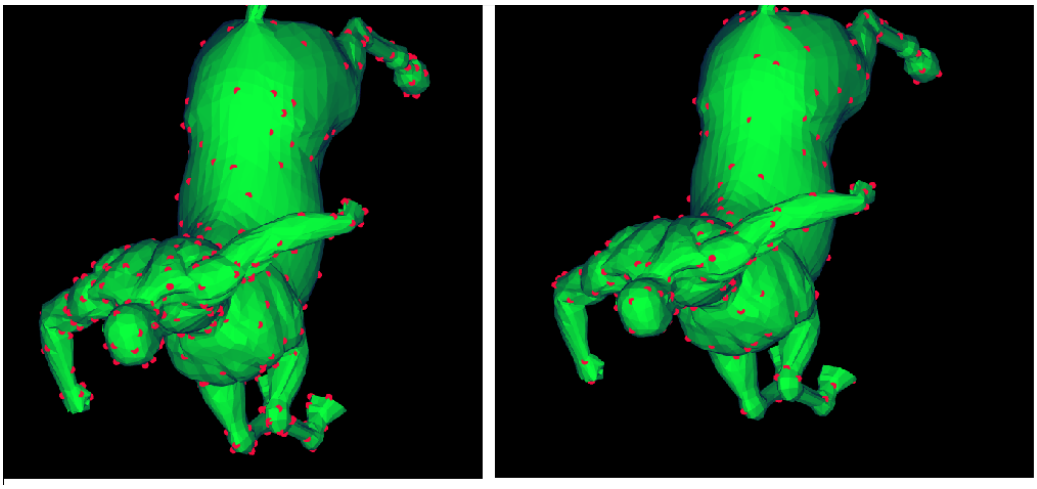


Figure 13: Clustering Method ring size 5(left) and 10(right)

In figure 13, on the left there is Centaur shape with the following parameter values: K-parameter:0.04, Selection:Clustering, **Ring Size:5**, Constant-fraction:0.01. Whereas the shape on the right side has K-parameter:0.04, Selection:Clustering, **Ring Size:10**, Constant-fraction:0.01. Both of them

have the same parameter values except the **ring size**.

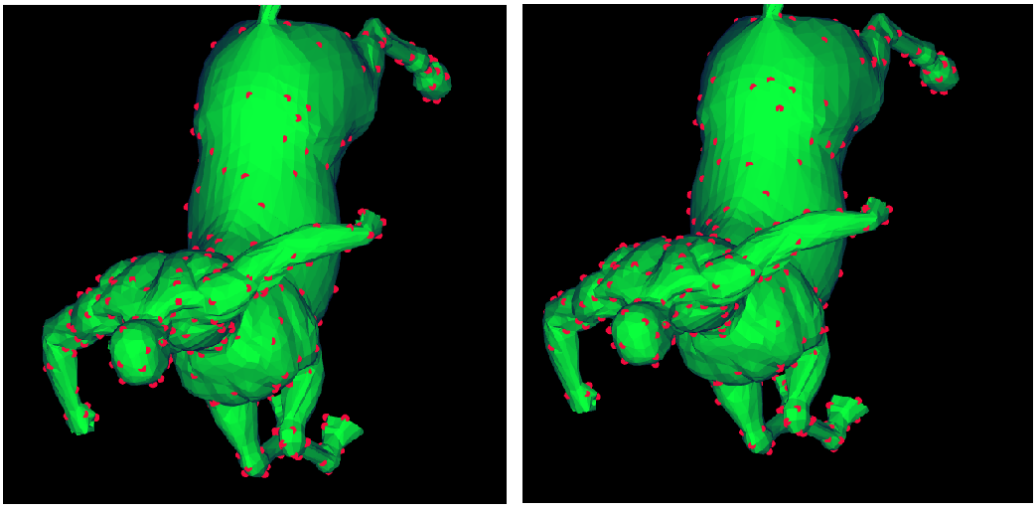


Figure 14: Clustering Method k parameter 0.04(left) and 0.1(right)

In figure 14, on the left there is Centaur shape with the following parameter values: **K-parameter:0.04**, Selection:Clustering, Ring Size:5 Constant-fraction:0.01. Whereas the shape on the right side has **K-parameter:0.1**, Selection:Clustering, Ring Size:5, Constant-fraction:0.01. Both of them have the same parameter values except the **K-parameter**.

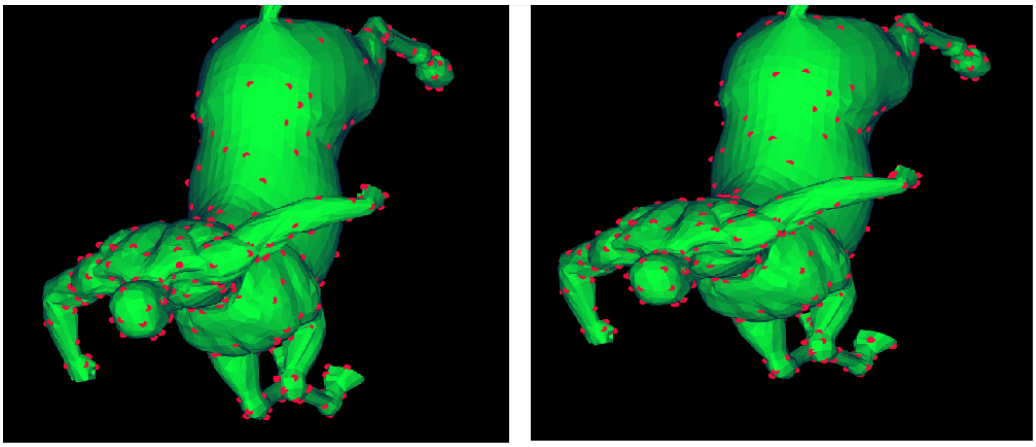


Figure 15: Clustering Method constant fraction 0.01(left) and 0.001(right)

In figure 15, on the left there is Centaur shape with the following parameter values: K-parameter:0.04, Selection:Clustering, Ring Size:5, **Constant-fraction:0.01**. Whereas the shape on the right side has K-parameter:0.04, Selection:Clustering, Ring Size:5, **Constant-fraction:0.001**. Both of them have the same parameter values except the **constant-fraction**.

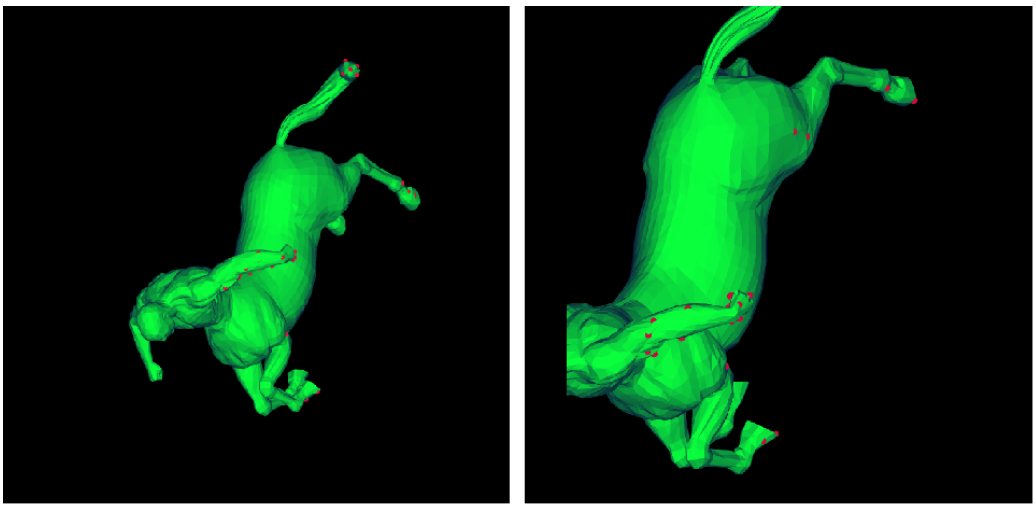


Figure 16: Highest Harris Method ring size 5(left) and 10(right)

In figure 16, on the left there is Centaur shape with the following parameter values: K-parameter:0.04, Selection:Highest Harris Response, **Ring Size:5**, Constant-fraction:0.01. Whereas the shape on the right side has K-parameter:0.04, Selection:Highest Harris Response, **Ring Size:10**, Constant-fraction:0.01. Both of them have the same parameter values except the **ring size**.

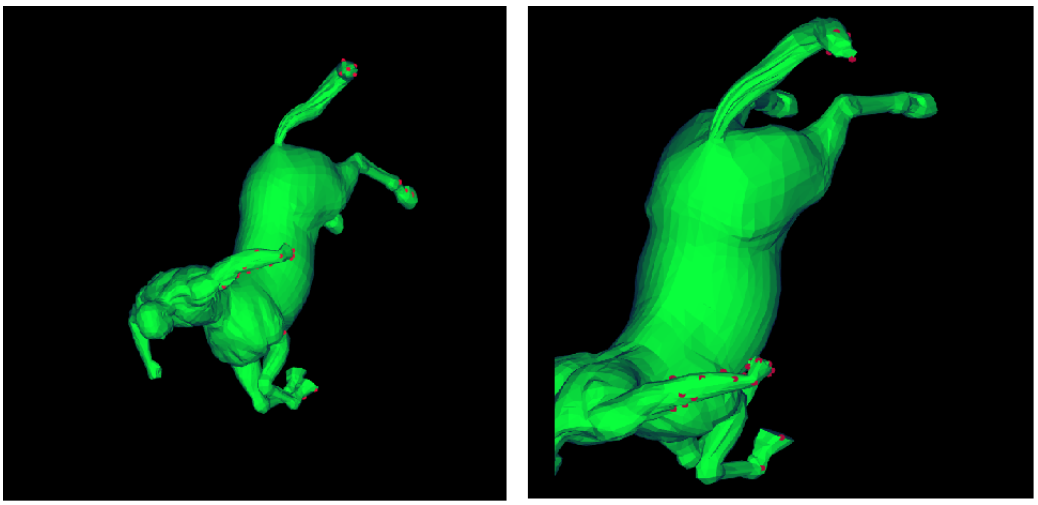


Figure 17: Highest Harris Method k parameter 0.04(left) and 0.1(right)

In figure 17, on the left there is Centaur shape with the following parameter values: **K-parameter:0.04**, Selection:Highest Harris Response, Ring Size:5 Constant-fraction:0.01. Whereas the shape on the right side has **K-parameter:0.1**, Selection:Highest Harris Response, Ring Size:5 Constant-fraction:0.01. Both of them have the same parameter values except the **k parameter**.

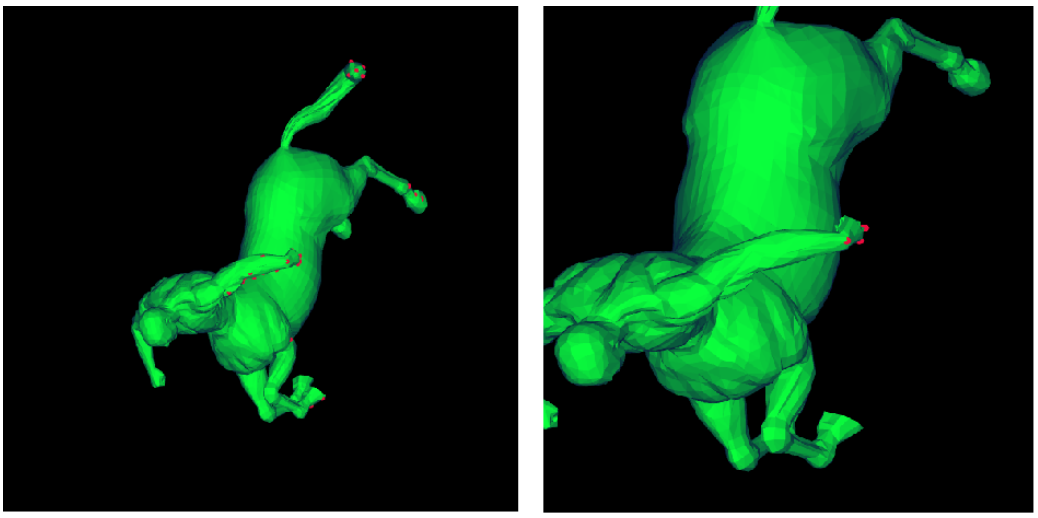


Figure 18: Highest Harris Method constant fraction 0.01(left) and 0.001(right)

In figure 18, on the left there is Centaur shape with the following parameter values: K-parameter:0.04, Selection:Highest Harris Response, Ring Size:5 **Constant-fraction:0.01**. Whereas the shape on the right side has K-parameter:0.04, Selection:Highest Harris Response, Ring Size:5, **Constant-fraction:0.001**. Both of them have the same parameter values except the **constant-fraction**.

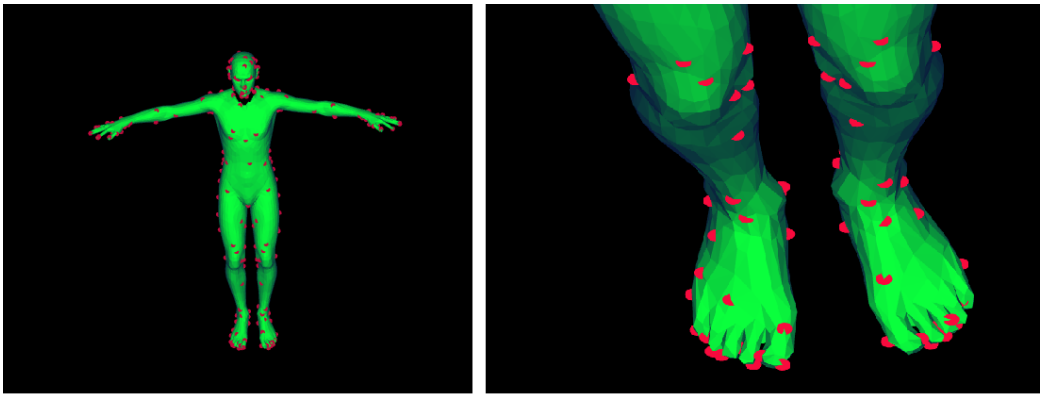


Figure 19: Human Shape Clustering Method(left) and zoomed version(right)

In figure 19, there are shapes of Human with different views and scaling.

The value of the parameters are K-parameter:0.04, Selection:Clustering, Ring Size:5, Constant-fraction:0.01.

3.2 Discussions

From the results that we have got, we have observed that increasing the ring size in both selection methods (Clustering and Highest Harris) resulted an increase in the number of interest points. When we compare clustering selection method and highest Harris method with the same parameters, the first one has a lot more interest points than the latter. When we change the k-parameter, we have observed that the one with lower k-parameter value has more interest points than the other one. By changing the constant fraction parameter, we have learned that as you increase the value of constant fraction and keep the other parameter values the same, the number of interest points increases.

References

[1] Ivan Sipiran and Benjamin Bustos, A robust 3D interest points detector based on Harris operator, Proc. Eurographics Workshop on 3D Object Retrieval (3DOR'10),
https://users.dcc.uchile.cl/ isipiran/harris3D.html

[2] K Hong, QT5 TUTORIAL OPENGL WITH QGLWIDGET - 2018, https://www.bogotobogo.com/Qt/Qt5_OpenGL_QGLWidget.php(Accessed: 22 Nov 2018)

[3] Prof. Yohan FOUGEROLLE, 2018, Lecture: Question and Answer Session on Harris 3D project, lecture notes, Software Engineering, Universite de Bourgogne, delivered on 22 Oct 2018

[4] (Citation styles only), Qt Documentation Archives, Signals & Slots, http://doc.qt.io/archives/qt-4.8/signalsandslots.html (Accessed: 14 Dec 2018)

[5] (Citation styles only), Hello GL Example https://doc.qt.io/archives/4.3/opengl-hellogl.html (Accessed: 1 Jan 2019)

[6] (Citation styles only), Eigen Library http://eigen.tuxfamily.org/index.php?title=Main_Page (Accessed: 10 Nov 2018)

[7] (Citation styles only), McGill 3D Shape Benchmark http://www.cim.mcgill.ca/ shape/benchMark/ (Accessed: 22 Oct 2018)

[8] (Citation styles only), TOSCA Dataset http://tosca.cs.technion.ac.il/book/resources_data.html (Accessed: 22 Oct 2018)

[9] (Citation styles only), Triangle mesh https://en.wikipedia.org/wiki/Triangle_mesh (Accessed: 28 Nov 2018)