Media Engineering and Technology Faculty
German University in Cairo

Displaying of point clouds as meshes

Bachelor Thesis

Author: Mohamed Adel Fahim
Supervisors: Dr. Rimon Elias
Eng. Mohamed Karam

Submission Date: 25 May, 2019

Media Engineering and Technology Faculty
German University in Cairo

Displaying of point clouds as meshes

Bachelor Thesis

Author:	Mohamed Adel Fahim
Supervisors:	Dr. Rimon Elias
	Eng. Mohamed Karam
Submission Date:	25 May, 2019

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Mohamed Adel Fahim
25 May, 2019

Acknowledgments

I would like to offer my sincerest thanks to my supervisor Dr. Rimon Elias, I want to thank him for always being there whenever I had any problem or question regarding the work on my thesis. I also want to thank him for his support, patience and motivation that he has given me through out this semester.

I would also like to thank my family and my close friends for their constant support and motivation. I would not have made it this far without them.

Abstract

Point cloud data plays an important role in a variety of applications including geospatial applications, virtual reality (VR) applications, 3D gaming and medical imagining. Point cloud data does not offer the order of which the points are to be drawn.

There are specialized software that deal with the reconstruction and drawing of point cloud data. However they are aimed at specific file formats namely PLY and PCD.

In this thesis we will introduce a way to deal with point cloud data which was provided in a text file format, draw the result in different ways, compare between them and finally give a few ideas for further improvement of the reconstruction process.

Contents

Chapter 1

Introduction

1.1 Motivation

There are many ways to triangulate and draw point cloud data, but all of them only work when the file is in PCD or PLY format, which prevents point cloud data created purely as a number of points without the added characteristics of PCD/PLY files from being utilized.

1.2 Aim of project

The Aim of this bachelor project is to use triangulation to reconstruct point Cloud Data given in text file format and explore different ways of drawing the resulting shape and compare between them.

1.3 Thesis Overview

The thesis consists of five chapters including this one which are :

Chapter 2 presents the background knowledge required in the project.

Chapter 3 shows the approach adopted in this project and the implementation of the project.

Chapter 4 compares between the different outputs of the project.

Chapter 5 gives a conclusion and talks about the future work on this project.

Chapter 2

Background

2.1 Definitions

2.1.1 Point Cloud

A point cloud is a collection of data points defined by a given coordinates system, a point cloud usually defines a shape whether real or created in 2D or 3D coordinate systems, they are different than other methods to represent shapes as a point cloud usually does not have directions or a clear indication of the shape they are supposed to represent, thus point cloud data is usually in need of an extra step to draw the shape it is made from.

Point cloud data is generally but not solely created from 3D scanners, and are used to create 3D meshes and other models used in 3D modeling for various fields such as 3D printing, 3D gaming, medical imaging and various virtual reality (VR) applications [1].

2.1.2 Voronoi Diagram

Voronoi Diagrams (also known as Voronoi Partitions) are simply the partioning of a plane into convex polygons where each polygon is created based on a point known as the generating point, where all the points in that polygon are closer to that particular points that other generating point.

The applications of Voronoi Diagrams are widespread among many fields ranging from science and technology to visual art,Figure 2.2 shows an example of Voronoi Diagrams which illustrates the generating points of the convex polygons [2].

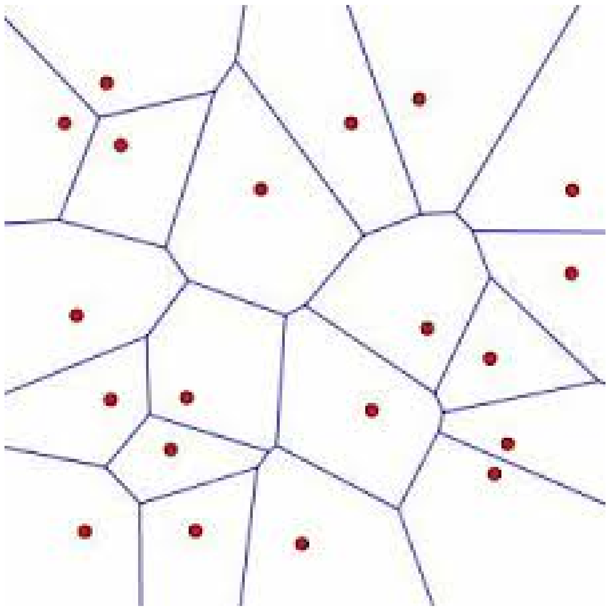


Figure 2.1: Example of a Voronoi Diagram [3].

2.1.3 Convex Hull

The convex hull is a ubiquitous structure in computational geometry ,even though it is a useful tool in its own right, it is also helpful in constructing other structures like Voronoi Diagrams [4], and can be applied in Euclidean spaces (both 2D and 3D).

The convex Hull (also known as convex envelope) of points is the minimal convex set wrapping our polygon. Assuming our set of points is named P, the convex hull is formally defined as intersection of all convex sets containing P, or as the set of all convex combinations of points in P, Figure 2.2 shows an example of convex hull applied to a set of points.

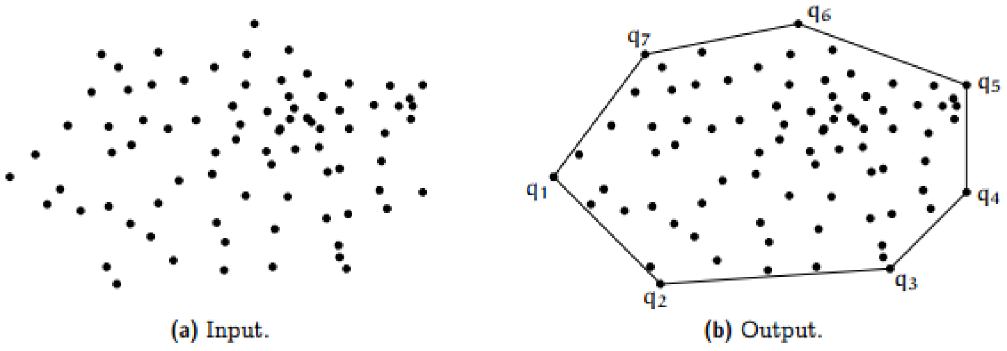


Figure 2.2: Example of input and output of a convex hull algorithm [5].

2.1.4 Triangulation

Triangulation is a common approach for reconstruction of geometric figures for a given set of points, and is simply the division of a surface or plane polygon into a set of triangles.

There are numerous different algorithms for triangulation but most are not suitable for 3D reconstruction and are more suited for 2D reconstruction which is simpler [6].

2.1.5 Delaunay Triangulation

Delaunay triangulation is a triangulation algorithm created by Boris Delaunay in 1934 and is named after him. Deluany triangulation subdivides the plane into triangles, such that for a set of discrete points P no point in the set of points is inside the circumcircle (a circle touching all the vertices of a triangle), while maximizing the minimum angle of all triangles in the triangulation.

Delaunay triangulation has a strong relationship with Voronoi Diagram such that for a discrete set of points P the Delaunay triangulation corresponds to the dual graph of the Voronoi Diagram for P, such that circumcenters of Deluany triangulation are the vertices of the Voronoi Diagram.

For Delaunay triangulation in 2D, the Voronoi vertices are connected via edges, this is derived from the adjacency-relationships of the Delaunay triangles, where if two triangles share an edge their circumcenters are connected with an edge in the Voronoi Diagram [7], Figure 2.3 shows an example of this relationship.

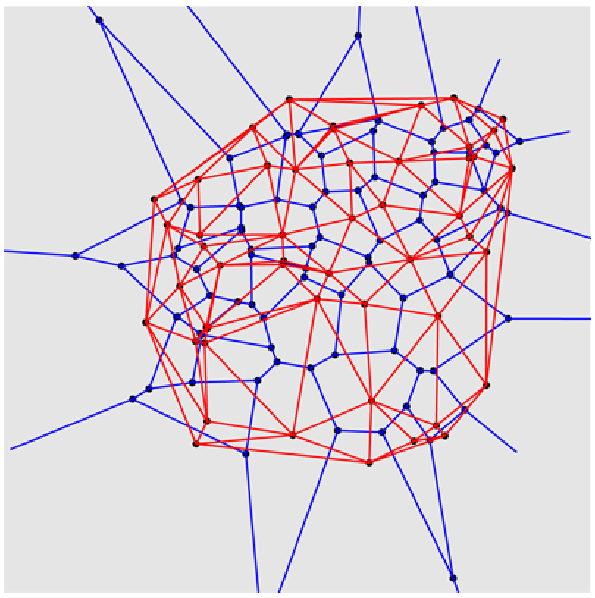


Figure 2.3: The blue in this picture is the Voronoi Diagram and the red is the dual Delaunay triangulation [8].

2.2 Technologies

2.2.1 C++

C++ is a general-purpose programming language that was created based on C and is basically an enhanced version of C with classes, C++ is a intermediate-level general-purpose middle-level programming language, sophisticated, efficient and provides facilities for low-level memory manipulation [9].

2.2.2 C#

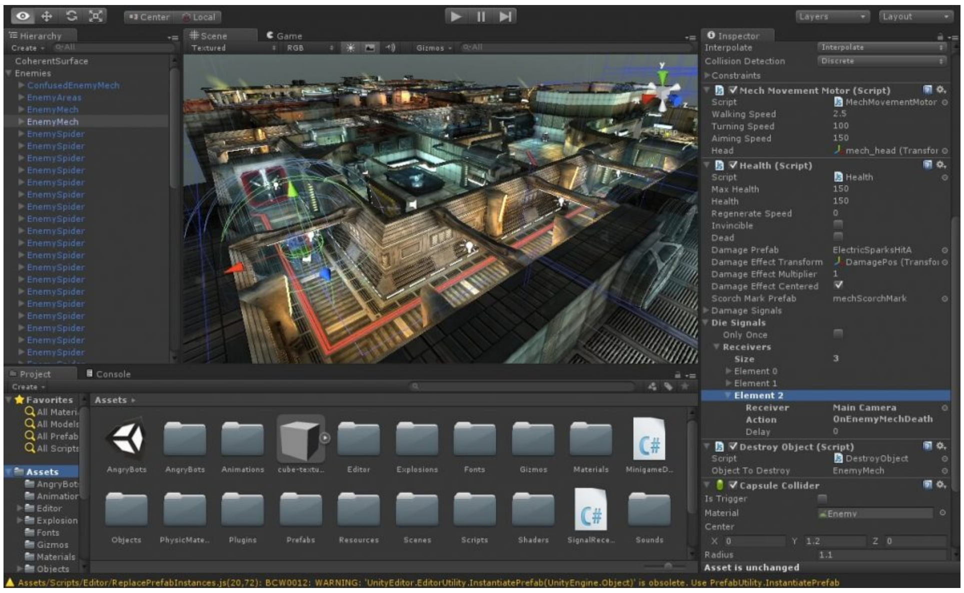
C# is a general-purpose, multi-paradigm programming language and is a strongly typed language that was developed by Microsoft and is used in essentially all of their products.

It is more recently, Windows 8 and windows 10 applications. It is also a part of .NET so it is used alongside languages like ASP in web development and apps.

2.2.3 Unity

Unity is a cross-platform game engine developed by Unity Technologies, and is the world's leading real-time engine. It is used to create half of the world's games.

Unity is mainly used to create both 2D and 3D games [10], but is also a powerful tool for rendering graphics. Unity mainly uses the C# language. Figure 2.4 shows unity's graphical prowess.



```
Interpolate                  Interpolate
Collision Detection          Discrete
Constraints
Mech Movement Motor (Script)
  Script                     MechMovementMotor
  Walking Speed              2.5
  Turning Speed              100
  Aiming Speed               150
  Head                       mech_head (Transfor
Health (Script)
  Script                     Health
  Max Health                 150
  Health                     150
  Regenerate Speed           0
  Invincible
  Dead
  Damage Prefab              ElectricSparksHitA
  Damage Effect Transform    DamagePos (Transfo
  Damage Effect Multiplier   1
  Damage Effect Centered     ✓
  Scorch Mark Prefab         mechScorchMark
  Damage Signals
  Die Signals
    Only Once
    Receivers
      Size                   3
      Element 0
      Element 1
      Element 2
        Receiver             Main Camera
        Action               OnEnemyMechDeath
        Delay                0
Destroy Object (Script)
  Script                     DestroyObject
  Object To Destroy          EnemyMech
Capsule Collider
Is Trigger
Material                     Enemy
Center
  X 0            Y 1.2            Z 0
Radius                       1.1
Asset is unchanged
```

```
Assets/Scripts/Editor/ReplacePrefabInstances.js(20,72): BCW0012: WARNING: 'UnityEditor.EditorUtility.InstantiatePrefab(UnityEngine.Object)' is obsolete. Use PrefabUtility.InstantiatePrefab
```

Figure 2.4: Example of Unity's rendering of a scene from a game [11].

2.2.4 OpenCV

OpenCV (Open Source Computer Vision Library) has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV is designed for computational efficiency and with a strong focus on real-time applications, Written in optimized C/C++.

The library can take advantage of multi-core processing. Enabled with OpenCV, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform, and is known to be the fastest library in rendering images[12], however it can not draw 3D objects, thus the need to rely on other external libraries. Figure 2.5 is an example of OpenCV applying Deluanay triangulation on a video sequence.

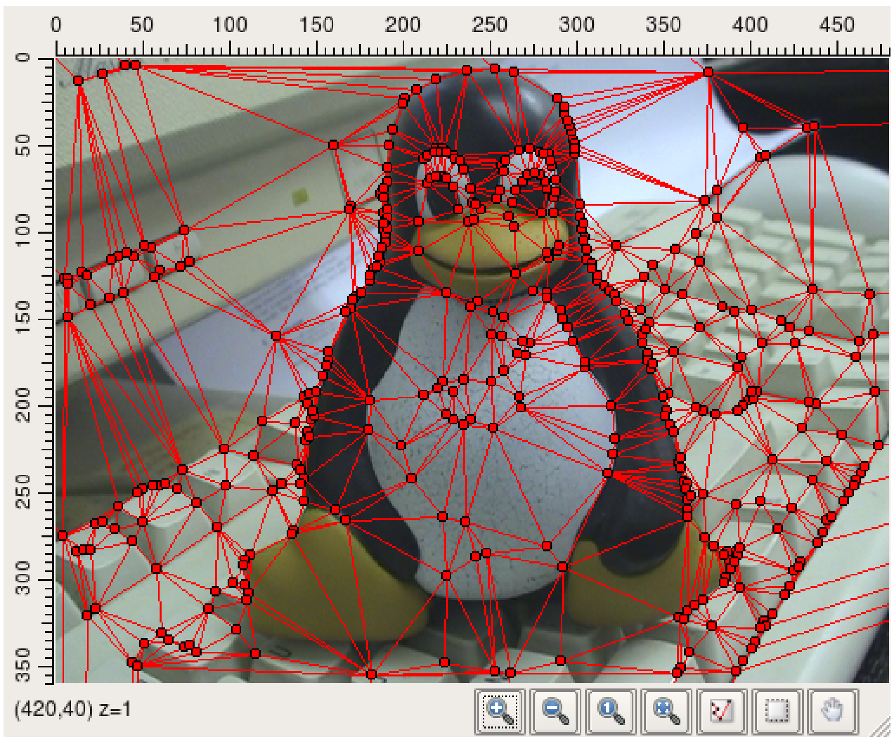


Figure 2.5: Example of Deluanay triangulation on a frame from a video sequence.

2.2.5 OpenGL

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU) to achieve hardware-accelerated rendering [13].

OpenGL is used to make games or graphics in 2D and 3D, and is used with C++ which makes it perfect for geometric mathematical operations of large number of points that usually precede the rendering of the graphics, Figure 2.6 shows an example of what OpenGL can create.

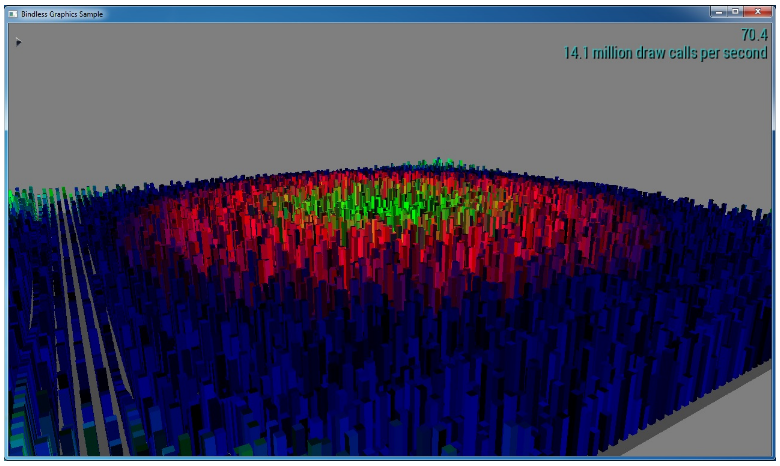


Figure 2.6: Example of OpenGL drawing a large number of shapes [14].

2.2.6 CGAL

CGAL is an industrial level software project that provides access to efficient and reliable geometric algorithms in the form of a C++ library.

CGAL is used in various areas needing geometric computation, such as geographic information systems, computer aided design, molecular biology, medical imaging, computer graphics, and robotics.

The library offers data structures and algorithms like triangulations, Voronoi Diagrams, Boolean operations on polygons and polyhedra point set processing, arrangements of curves, surface and volume mesh generation, geometry processing, alpha shapes, convex hull algorithms, shape reconstruction, AABB and KD trees.

CGAL also provides both 2D and 3D triangulation and has two different types of 3D triangulation namely normal triangulation and Deluany reiangulation and can write he resulting output to be used later to redraw the triangulation without having to do the triangulation again [15].

2.2.7 Vcpkg

Vcpkg is a command line package manager developed by Microsoft to manage C and C++ libraries. Vcpkg greatly simplifies the acquisition and installation of third-party libraries on Windows,Linux and Mac, and supports both open-source and proprietary libraries[16]. On windows Vcpkg uses power shell for it's command line interface.

2.2.8 Visual Studio 2017

Visual Studio 2017 is an integrated development platform created by Microsoft for Android, iOS, Windows, web, and cloud. It provides a good interface to manage projects, and is compatible with a wide range of programming languages and libraries [17].

Chapter 3

Implementation

First of all, this chapter will talk about the approach adopted in this project and why it was chosen, then talk about the GUI and give a brief walk through on how to interact with it and finally illustrate briefly the steps of the implementation for both OpenGL and Unity.

3.1 Approach

In this project, there were three factors that decided the approach that had to be taken in consideration. First of all the huge number of points that has to be dealt with (almost two million points which is the number of points to represent a human brain in the given sample).

Secondly the way the input was given which was in a text format containing five space separated numbers which represented x, y, z, gray scale and a number denoting which points should be hidden to take off the front layer of the skull to show the brain (unique to the sample being worked on) .

Finally the data was provided in a text file format which prevented the usage of Unity extensions targeting point cloud data and point cloud specialized libraries like PCL, and the huge number of points meant that the library used to draw the shape had to be efficient and fast to allow acceptable camera movement and rendering time.

Thus the idea behind of this approach is to find a library to that does the computation of triangulation as efficiently as possible, and find the best means to draw the resulting points efficiently, furthermore output this data in a form (text file in this case) that can be used to later to skip the steps of have to triangulate the data again on every use and thus save time when redrawing the data.

Therefore the library CGAL was chosen for applying the triangulation, as for drawing the output there were three possible software/libraries to draw the results namely OpenGL, OpenCV and Unity, which we will talk about their comparison later in Chapter 4.

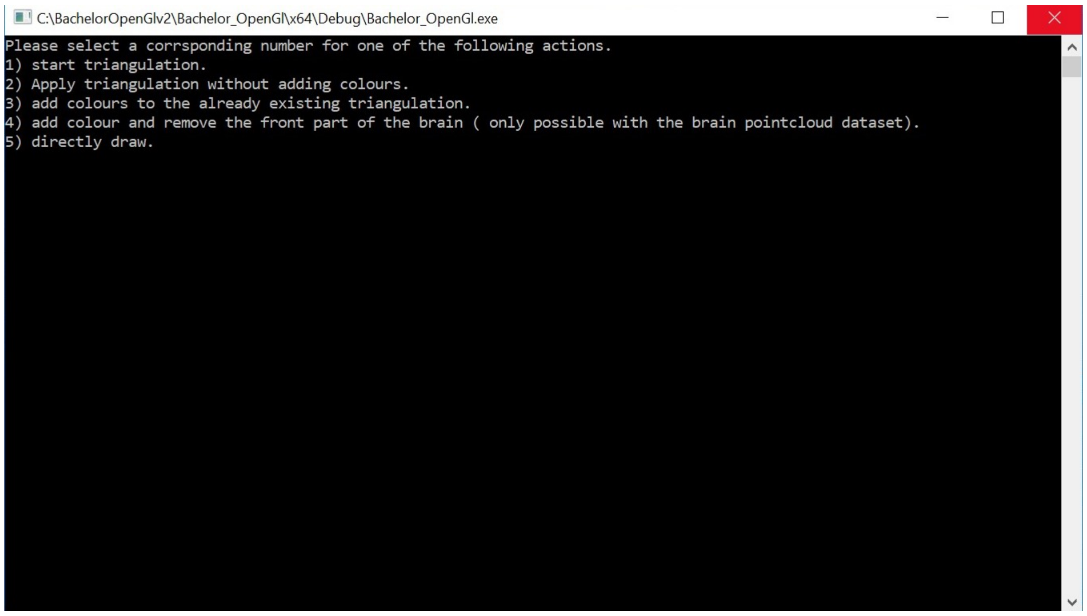
3.2 Vcpkg, CGAL and Visual studio 2017

Vcpkg was chosen over other package managers (like mingw and cygwin) for its compatibility with Visual Studio 2017 without the need of specifically linking vcpkg to Visual Studio 2017 as both are developed by Microsoft.

Furthermore Vcpkg was chosen because CGAL is generally a hard library to install and relies on many other packages to function (87 libraries in total) and has slightly more complicated steps to fully use compared to other libraries as they involve using Cmake, Vcpkg simplifies the steps to make CGAL work to a single line in power shell.

However, this project uses an old version of vcpkg (commit with the number 63265da) because newer commits have turned off Cmake which is a crucial part of making CGAL work.

3.3 GUI


C:\BachelorOpenGlv2\Bachelor_OpenGl\x64\Debug\Bachelor_OpenGl.exe

```
Please select a corrsponding number for one of the following actions.
1) start triangulation.
2) Apply triangulation without adding colours.
3) add colours to the already existing triangulation.
4) add colour and remove the front part of the brain ( only possible with the brain pointcloud dataset).
5) directly draw.
```

Figure 3.1: the starting options in the program.

As shown in Figure 3.1, on starting application you are given five choices to allow you to start triangulation from the start or skip some of the steps if they have been already done before. This allows you to draw already triangulated files or add colours to the triangulated files if it is not included. depending on which choice is made a set of options are presented for the user to choose from as shown in Figure 3.2.

Then the user is finally presented with a list of information detailing elapsed time and number of points that also act as a way to check the progress as the code is running as shown in Figure 3.3. The files that this project can work on must be in text format and have space separated X,Y,Z points with the gray scale and extra value denoting the points to hide.

However, with minor adjustments to the part of the code that reads the files the program can be made to work for any text file with space separated X, Y, Z and any number of extra data or no data at all.

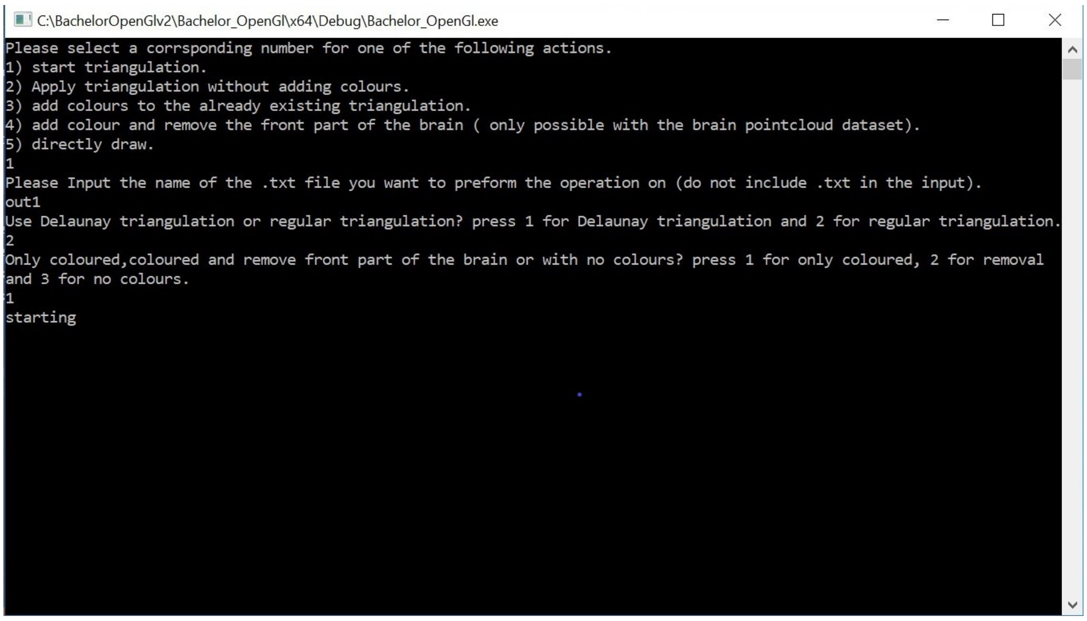
```
short xx, yy, zz, vv, ii;
ifstream myfile;
myfile.open(inputName + ".txt");
while (myfile >> xx >> yy >> zz >> vv >> ii)
{
if (deluany)
shape.insert(Point(xx, yy, zz));
else
points.push_front(Point(xx, yy, zz));

count++;
}
```

for example the code above (found in all classes) can be changed to the code below to work on .txt file containing x,y and z points only

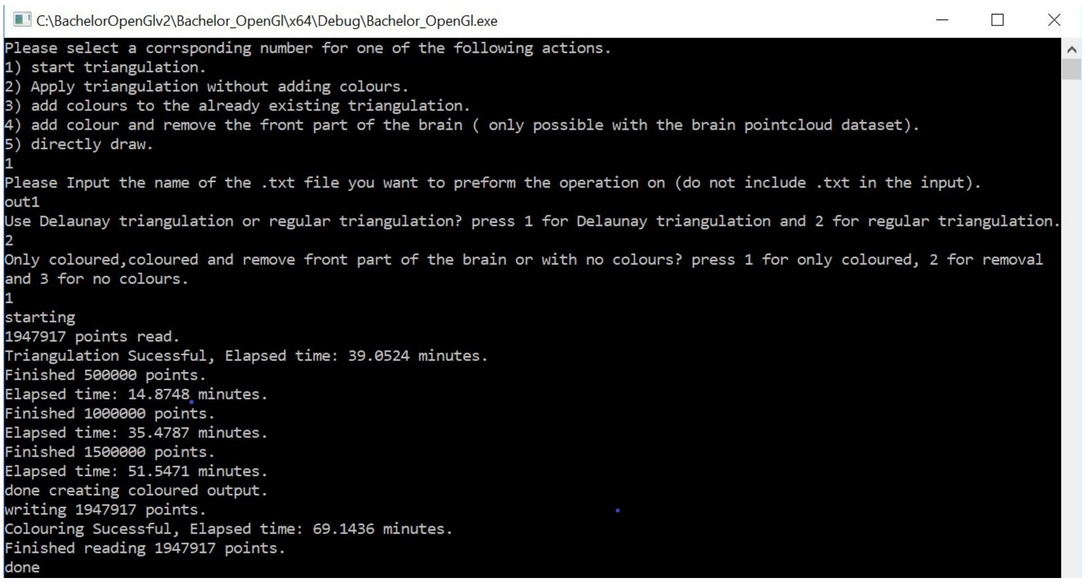
```
short xx, yy, zz;
ifstream myfile;
myfile.open(inputName + ".txt");
while (myfile >> xx >> yy >> zz )
{
if (deluany)
shape.insert(Point(xx, yy, zz));
else
points.push_front(Point(xx, yy, zz));

count++;
// cout << "done with point no." << count << endl;
}
```



```
C:\BachelorOpenGlv2\Bachelor_OpenGl\x64\Debug\Bachelor_OpenGl.exe
Please select a corrsponding number for one of the following actions.
1) start triangulation.
2) Apply triangulation without adding colours.
3) add colours to the already existing triangulation.
4) add colour and remove the front part of the brain ( only possible with the brain pointcloud dataset).
5) directly draw.
1
Please Input the name of the .txt file you want to preform the operation on (do not include .txt in the input).
out1
Use Delaunay triangulation or regular triangulation? press 1 for Delaunay triangulation and 2 for regular triangulation.
2
Only coloured,coloured and remove front part of the brain or with no colours? press 1 for only coloured, 2 for removal
and 3 for no colours.
1
starting
```

Figure 3.2: Example of the sub options available.



```
C:\BachelorOpenGlv2\Bachelor_OpenGl\x64\Debug\Bachelor_OpenGl.exe
Please select a corrsponding number for one of the following actions.
1) start triangulation.
2) Apply triangulation without adding colours.
3) add colours to the already existing triangulation.
4) add colour and remove the front part of the brain ( only possible with the brain pointcloud dataset).
5) directly draw.
1
Please Input the name of the .txt file you want to preform the operation on (do not include .txt in the input).
out1
Use Delaunay triangulation or regular triangulation? press 1 for Delaunay triangulation and 2 for regular triangulation.
2
Only coloured,coloured and remove front part of the brain or with no colours? press 1 for only coloured, 2 for removal
and 3 for no colours.
1
starting
1947917 points read.
Triangulation Sucessful, Elapsed time: 39.0524 minutes.
Finished 500000 points.
Elapsed time: 14.8748 minutes.
Finished 1000000 points.
Elapsed time: 35.4787 minutes.
Finished 1500000 points.
Elapsed time: 51.5471 minutes.
done creating coloured output.
writing 1947917 points.
Colouring Sucessful, Elapsed time: 69.1436 minutes.
Finished reading 1947917 points.
done
```

Figure 3.3: Example of a finished run in the program.

3.4 Classes and methods (C++ and OpenGL)

3.4.1 `Triangulation_CGAL`

The class `triangulation_CGAL` in this class is the method responsible for the triangulation of the point cloud data. This class uses the Computational Geometry Algorithms Library CGAL which has two types of 3D triangulation which are Delaunay triangulation and a regular triangulation and depending on the need of the user either of them can be chosen for triangulating the output.

Both triangulation methods do their own convex hull and output the files with extra information containing the vertex base, number of points, dimension of the triangulation and other info that CGAL uses when accessing the triangulation data later, both types of triangulation support insertion and removal of points, Delaunay triangulation in CGAL is implemented as a data structure where points are inserted one by one and the Voronoi Diagram is recreated and triangulation is also calculated again using the new Voronoi Diagram which makes it quite slower than normal triangulation.

Normal triangulation, on the other hand, is implemented as incremental regular triangulation, also known as weighted Delaunay triangulations and is used as a method that takes a list as an input making it vastly faster than the Delaunay triangulation implementation. It also runs on points even if they have no weight and in this case the implementation of the algorithm is exactly the same as the Delaunay triangulation,

However, due to the implementation of the Delaunay triangulation as a data structure not as a method, the output differs than when using normal triangulation even though they use the same algorithm. This is because Delaunay triangulation data structure is applied on point by point thus the distance between the points is changed every single time a new point is inserted (different Voronoi diagrams because of different distances). Unlike normal triangulation which in the case of no weighted points uses the Delaunay triangulation but only once on the whole data as it is a method used on a list, Figure 3.4 shows the derivation diagram of the 3D triangulation classes implemented in CGAL.

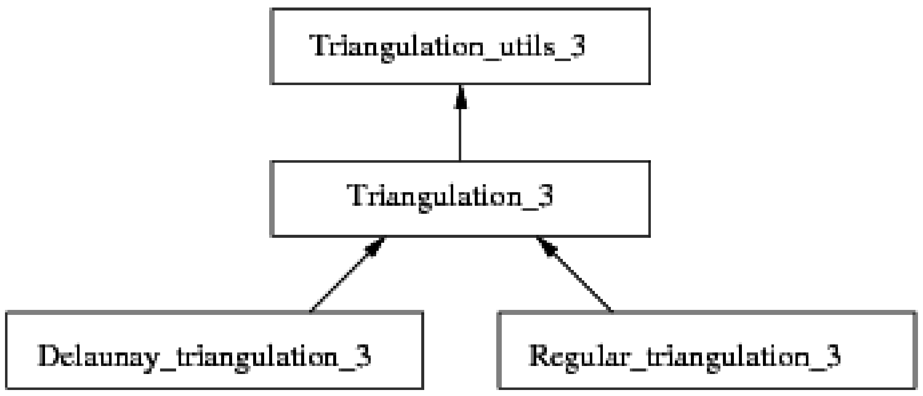


Figure 3.4: 3D triangulation derivation diagram [18]

3.4.2 Manage_TraingulationOutput

The first task of the class `Manage_TraingulationOutput` is to read the data from the output file created by the triangulation and discard all unnecessary data and output the file in an appropriate format depending on the options chosen when running the program.

First of all, if the user selected a colourless triangulation then the file will output the data to a new file with the format of space separated X, Y and Z.

Secondly, if the user selects to colour the triangulation without removing the front part the method then loads the original file and compare it to the extracted data and match the gray scale value to the new data and output it in to a file the a space separated X, Y, Z and gray scale format.

Finally, if the user decides to remove the front part of the brain (which is an option added for the data set used in this bachelor and it will use the original input to match the values denoting which points to remove, and the colour of each group of points and output that to a file in the format of space separated X, Y, Z, gray scale and value format denoting which points to remove in this example.

The data above is stored in a vector, because it can have its size dynamically assigned unlike arrays in C++, however vectors have two downsides which are taking a huge amount of time to insert elements due to how it is implemented and accessing the data because of poor debugger choices.

The long time taken to insert element is because the vector implemented as an array that starts with a fixed size and this size is increased every time the limit is used by creating a new array with a bigger size, this is addressed by reserving the length for the vector array before inserting any data into it, the size is collected through the triangulated output file by reading the second line in it which always contains the number of points as shown by the code below.

```
myfile.open(newFile + ".txt");
myfile.ignore(10000, '\n');

string str;
std::getline(myfile, str);
length = std::stoi(str);

dataX.reserve(length);
dataY.reserve(length);
dataZ.reserve(length);
dataV.reserve(length);
dataI.reserve(length);

dataXTri.reserve(length);
dataYTri.reserve(length);
dataZTri.reserve(length);
dataVTri.reserve(length);
dataITri.reserve(length);
```

The other issue occurs because the debugger when accessing an element from a vector must first check that no change happened to the vector, in this case the size is fixed and no changes happen to the vector array.

```
short *dataXPTri = &dataXTri[0];
short *dataYPTri = &dataYTri[0];
short *dataZPTri = &dataZTri[0];

auto start = std::chrono::high_resolution_clock::now();
if (coloured) {
short *dataXP = &dataX[0];
short *dataYP = &dataY[0];
short *dataZP = &dataZ[0];
short *dataVP = &dataV[0];
short *dataIP = &dataI[0];
```

To bypass this issue pointers that point to the start element of the vector are created as shown by the code snippet above, then the other elements are accessed by adding to that pointer a value for the index in a for loop. This emulates how the vector is accessed but without the debugger interfering, the code also includes some checks to show progress as the processes can take a long time, this is shown in the code snippet below.

```
for (int i = 0; i < length; i++) {
if (i == 1000000 || i == 1500000 || i == 500000) {

std::cout << "Finished " << i << " points." << endl;

auto finish1 = chrono::high_resolution_clock::now();
chrono::duration<double> endTime = finish1 - start;

std::cout << "Elapsed time: " << endTime.count() / 60 << " minutes.\n";

}

for (int j = 0; j < length; j++) {

if (*(dataXPTri + i) == *(dataXP + j) && *(dataYPTri + i) == *(dataYP + j)
&& *(dataZPTri + i) == *(dataZP + j)) {
dataVTri.push_back(*(dataVP + j));
if (remove)
dataITri.push_back(*(dataIP + j));
     break;
}

}
}
}
```

3.4.3 Draw_Triangulation_OpenGL

The class `Draw_Triangulation_OpenGL` is responsible for managing the GUI and giving the correct flags to the other classes mentioned above depending on the choice of the user in the GUI that is first presented to him when he runs the program.

After giving the correct flags to the other classes, this class takes the output result from the `Manage_TraingulationOutput` class and draws the triangulation through OpenGL, with any of the different options available depending on the desire of the user, it also implements a camera method which you can control through the keyboard, Figure 3.5 is a UML class of the OpenGL implementation.

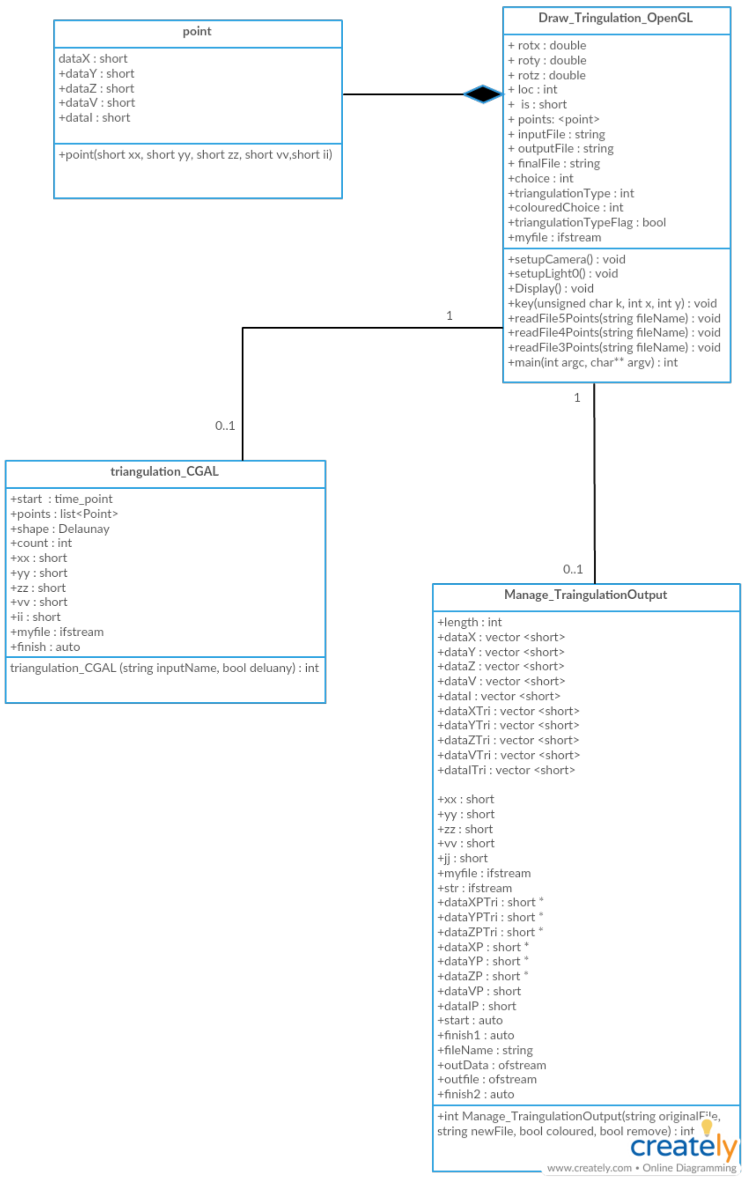


Figure 3.5: UML diagram [3].

3.5 C# scripts (Unity)

There are two ways to draw points in Unity, via meshes or via OpenGL. This section will talk about and explain both scripts then talk about how the camera movement is implemented.

3.5.1 DrawTriangulation_OpenGL

This script is attatched to the main camera and is responsible for drawing the results produced from `Manage_TraingulationOutput` method in C++ (CGAL does not support C# so triangulation has to be done in C++) using OpenGL .

First of all, OpenGL in Unity is quite different than using pure OpenGL in C++. Draw methods of OpenGL in Unity are executed immediately even before the camera is rendered, which means that the camera will clear the screen making the OpenGL drawing not visible. The only way to combat that is to draw in `OnPostRender` which is called only after the camera is rendered, however OnPostRender is only called if it is attached to the camera.

Furthermore `GL.Begin(GL.TRIANGLES)` (equivalent to `glBegin(GL_TRIANGLES)` in C++), unlike in pure OpenGL, does not use the vectors given to the `draw` function as coordinates but use them as percentages from the attached object. And since the `OnPostRender` will only work when attached to the camera, that means that the shape will always be rendered in the place the camera is at showing as a percentage of the full screen. Making it impossible to view the whole shape or even rotate it and go inside it since any movement from the camera will move the drawn shape, this is shown in the official documentations of Unity on the `GL.TRIANGLES` method[19].

Moreover, in most cases since the points are created with a coordinate system in mind and the fact that to use OpenGL in Unity, it has to be attached to the camera, then most if not all of the shape will be drawn outside of the camera view and will never be visible. Thus an extra step needed to be taken when loading the data, which is finding the maximum point in the X,Y,Z coordinates and dividing the points by them before drawing to ensure that all points stay within the screen, this is shown by the code below.

```
GL.PushMatrix();
     GL.Begin(GL.TRIANGLES);

       for(int i=0; i <arrayOfX.Count-2 ;i+=3){
     mat.SetPass(0);
     GL.LoadOrtho();
     GL.Vertex3(((float) arrayOfX[i])/maxX, ((float) arrayOfY[i])/maxY,
     ((float) arrayOfZ[i])/maxZ);
     GL.Vertex3(((float) arrayOfX[i+1])/maxX, ((float) arrayOfY[i+2])/maxY,
     ((float) arrayOfZ[i+1])/maxZ);
     GL.Vertex3(((float) arrayOfX[i+2])/maxX, ((float) arrayOfY[i+2])/maxY,
     ((float) arrayOfZ[i+2])/maxZ);
      }

  GL.PopMatrix();
```

3.5.2 DrawTriangulation_Mesh

This C# script is attached to the main camera and is responsible for drawing the results produced from `Manage_TraingulationOutput method` in C++ using meshes.

A mesh in Unity can be created by specifying the vertices and the order in which the points of the triangles are to be looked at, however meshes in Unity have two limitations; first of all, the given vertices have to be a power of 3, otherwise an error is given unlike OpenGL where the last vertex in such case is ignored, and a mesh can only hold 16 bit worth of points (65535 points).

Furthermore, a single game object can only hold one mesh. Thus, an array of vertices, an array of directions and an array for colors are initialized with the size of 65535(16 bit) then a mathematical equation is applied when the program is run to approximate the number of vertices to the nearest number that is a power of 3 and then using that number to get the number of times the number of meshes needed to be created to carter all the points, below is the code showing the formula used for the approximation.

```
int maxSize = size;
if(! (size%3 == 0)){
int quotient = size /3;
size = 3*(quotient+1);
}
Debug.Log(size);
int maxVectorSize = 65535;
int numberOfMeshes = System.Convert.ToInt32(System.Math.Ceiling(
(( double ) size) / (( double ) maxVectorSize))) ;
```

Then the number of meshes are used to create an equivalent number of game objects and are then inserted into an array of game objects to later attach the meshes to.

After that a loop is created that reads the file line by line and inserts the coordinates and colors (if colors exist in the input file) into their respective arrays and a simple counter starting from the number zero and incrementing on every insertion until they are full (65535 passes). Then the an object is chosen in order from the previously created objects and then the mesh is attached to it. All the meshes are created at the same point so that the final shape is correctly displayed. This shown by the snippet of code blow.

```
vertices [meshCount]  = new Vector3 ((float) System.Convert.ToDouble(entries[0]),
(float) System.Convert.ToDouble(entries[1]),
(float) System.Convert.ToDouble(entries[2]));
triangles [meshCount] = index;
colors [meshCount] = new Color ((float) System.Convert.ToDouble(entries[3])/255,
(float) System.Convert.ToDouble(entries[3])/255,
        (float) System.Convert.ToDouble(entries[3])/255, 0);

        index ++;
        meshCount ++;

           }

       }
   if(meshCount == maxVectorSize){
   listOfObjects[currentMesh].AddComponent<MeshFilter>();
   listOfObjects[currentMesh].AddComponent<MeshRenderer>();
   listOfObjects[currentMesh].GetComponent<MeshRenderer>().material = mat;
   listOfObjects[currentMesh].GetComponent<MeshFilter>().mesh.vertices = vertices;
   listOfObjects[currentMesh].GetComponent<MeshFilter>().mesh.triangles = triangles;

   if(entries.Length == 4 || entries.Length == 5)
       listOfObjects[currentMesh].GetComponent<MeshFilter>().mesh.colors = colors ;

   currentMesh++;
   meshCount =0;
   index = 0;
```

Moreover, conditions are included in the code that check the number of numbers per line and accordingly process the data, where if the length is three it draws without adding colors, if it is four then it draws and adds colors. Finally if it is five it takes the fifth value and removes all points having that value less than 17 (used to remove the front part of the brain data) which is specifically coded for the data set given in this project and is used as an example. To add colors to the mesh, a shader that is compatible with mesh colors needs to be used. In this case the Particles/Standard Surface Unity built in shader is used but any other Particle shader could work.

3.5.3 Camera Movment

This C# script is attached to the main camera and is responsible for the camera movements, using the w a s d keys move the camera up,left,down and right respectively. Zooming in and out is done via the mouse wheel, while rotating the camera is done by simply moving the mouse. This is done by getting the mouse x and y axis and using those as a value for a rotation in each update. The code below shows how this is done.

```
y = Input.GetAxis("Mouse X");
x = Input.GetAxis("Mouse Y");
rotateValue = new Vector3(x, y * -1, 0);
transform.eulerAngles = transform.eulerAngles - rotateValue*3;
```

Chapter 4

Comparison

All comparisons made in this chapter were run on one PC, and using the same triangulation output (normal triangulation colored).

Furthermore, this chapter will compare the results between CGAL's Delaunay triangulation and normal triangulation in both OpenGL and Unity, and then compare drawing the results on both Unity and OpenGL. Finally, this chapter will talk about OpenCV and why it is not a feasible option for drawing the results.

4.1 Specifications

- Processor Speed : 2.50GHZ
- Processor Type : Processor Intel(R) Core(TM) i7-6500U CPU
- RAM: 8GB

4.2 CGAL's Delaunay triangulation and normal triangulation

Figures 4.1-4.5 show both Delaunay and normal triangulation drawn on both OpenGL and Unity. As shown by the pictures, Delaunay triangulation shows better results in both cases. This result is caused by the way the triangulations are implemented in CGAL.

First of all, both triangulations have the same algorithm when dealing with non-weighted points. However, the difference between them is that Delauany triangulation is implemented as a data structure, this means that the points are inserted point by point, therefore triangulation is applied on them each time, calculating the Voronoi diagram of the points each time a point is inserted. This significantly slows the triangulation process but as Figures 4.1-4.5 show, it also increases the quality of the output.

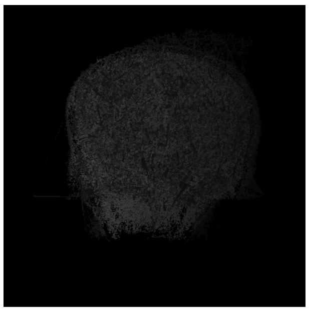


Figure 4.1: Delaunay triangulation

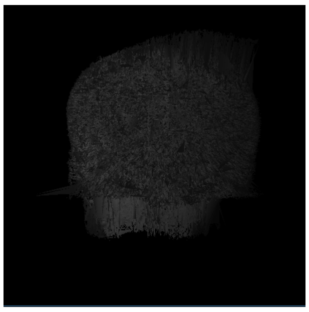


Figure 4.2: Normal triangulation

OpenGL

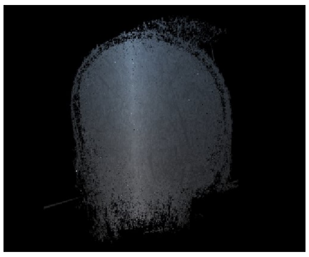


Figure 4.3: Delaunay triangulation

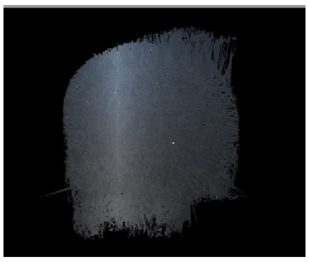


Figure 4.4: Normal triangulation

Unity

Unlike Delauany triangulation, normal triangulation is a method called on a list of points. While this is significantly faster, it forsakes the act of calculating the Voronoi diagram and the change of distance between points that happens in the Delauany triangulation, and as shown by Figures 4.1-4.5 this triangulation method results in uneven results and an output that is not as accurate as Deluanay triangulation.

Furthermore, Figures 4.5-4.8, where part of the skeleton from around the brain is removed, follow the same trend with Delaunay triangulation showing higher qualities results over normal triangulation.

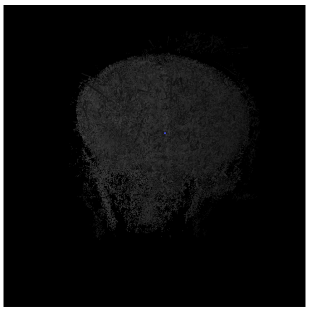


Figure 4.5: Delaunay triangulation

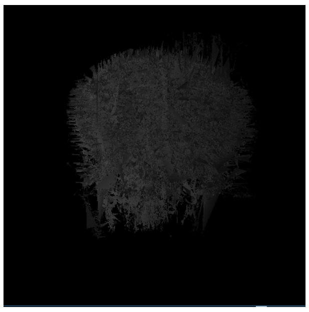


Figure 4.6: Normal triangulation

OpenGL front part removed

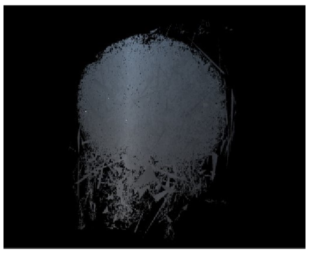


Figure 4.7: Delaunay triangulation

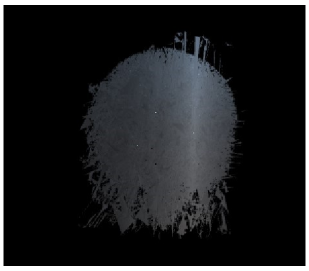


Figure 4.8: Normal triangulation

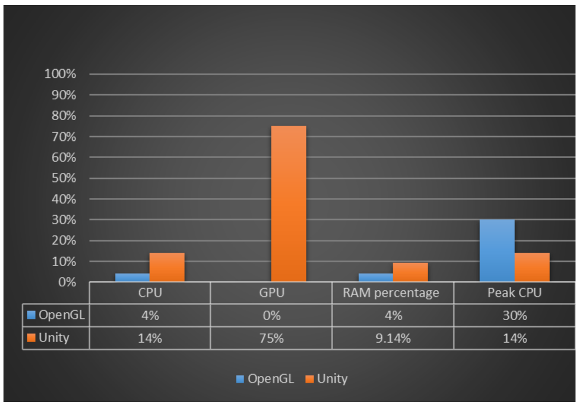
Unity front part removed

4.3 OpenGL and Unity

Both OpenGL and Unity have proven to be capable of drawing the large number points and both have the same quality with slight differences due to the difference in color and difference in lighting, as unit has more sophisticated lighting than OpenGL, as shown in figures 4.1-4.8.

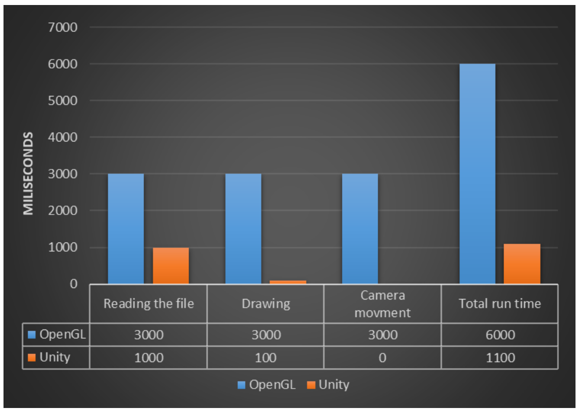
As shown in figure 4.9, OpenGL has a total run time of approximately 6000 milliseconds where they are split evenly where 3000ms is needed to read the file and created the necessary arrays and another 3000ms are needed to draw the necessary triangles from the created arrays. Furthermore, any movement with the camera requires the exact same time as drawing would take (3000ms) as OpenGL has to move every single triangle in the direction of movement since OpenGL does not have a camera.

While, Unity, has a total run time of 1100 milliseconds where reading the data from the file and doing the necessary math takes approximately 1000 milliseconds and creating the meshes takes approximately 100 milliseconds. Due to the nature of how the code is written where the meshes are created while reading the file the script was run without the creation of the meshes first, then the script was run again with the creation of the meshes and the difference between the different runs was recorded to divide the total run time into the time needed to read the data and the time needed to draw the meshes.



	CPU	GPU	RAM percentage	Peak CPU
🟦 OpenGL	4%	0%	4%	30%
🟧 Unity	14%	75%	9.14%	14%

Figure 4.9: Total run time



	Reading the file	Drawing	Camera movment	Total run time
🟦 OpenGL	3000	3000	3000	6000
🟧 Unity	1000	100	0	1100

Figure 4.10: Resources used

OpenGL

However, as shown in figure 4.10 drawing with OpenGL takes little resources, specifically taking an average of 0 % CPU increasing to an average of 30 % when moving the camera with 0 % GPU , however Visual studio itself needs an average of 4 % GPU and 400 MB RAM.

Furthermore, since Unity only needs to draw the shape once and is able to keep it drawn without refreshing unlike OpenGL movement with the camera is instantaneous since Unity implements a real camera and does not have to move all the meshes on movement. But Unity drains significant resources mainly on the GPU, Unity takes an average of 14 % CPU, 731 MB RAM and 75 % GPU which is higher than OpenGL in all aspects.

4.4 OpenCV

Finally, unlike OpenGL and Unity, OpenCV has proven to be inadequate for drawing 3D shapes as OpenCV will need to rely on external libraries to allow it to draw 3D shapes, and does not provide any means for helping the triangulation of the files to compenstate.

Chapter 5

Conclusion And FutureWork

5.1 Conclusion

In conclusion the aim of this thesis is to draw the resulting shape of a point cloud with a huge number of points (provided in a text file) using triangulation and compare between different methods of drawing the resulting shape.

Three methods were explored in this thesis of which two were found adequate namely using meshes in Unity and using OpenGL and one was found unusable namely OpenCV.

According to the results in Chapter 4, OpenGL was proven to be superior when the users want to draw the shape without moving the camera around as it uses far less resources (mainly in GPU usage). While Unity has proven to be superior when the user can afford the increased resources needed for Unity to run since it has zero delay when moving the camera compared to a three second delay in OpenGL.

5.2 Future Work

For future work, different approaches to triangulation can be experimented with, such as turning the text file into a PCD file and using the PCL library or trying different manual implementation of the Delaunay triangulation.

Appendix

Appendix A

Lists

List of Figures

Bibliography

[1] Margaret Rouse. Point cloud. https://whatis.techtarget.com/definition/point-cloud.

[2] Eric W Weisstein. Voronoi diagram. http://mathworld.wolfram.com/VoronoiDiagram.html.

[3] David Austin. http://www.ams.org/publicoutreach/feature-column/fcarc-voronoi.

[4] Brilliant.org. https://brilliant.org/wiki/convex-hull/, April 3rd 2019.

[5] Harshit Sikchi. https://medium.com/@harshitsikchi/convex-hulls-explained-baab662c4e94.

[6] Masood Varshosaz, H Helali, and Davood Shojaei. The methods of triangulation. January 2005.

[7] https://en.wikipedia.org/wiki/Delaunay_triangulation.

[8] Mikael Vejdemo Johansson. http://cs.smith.edu/~jorourke/DCG/.

[9] https://www.programiz.com/cpp-programming.

[10] https://unity.com/.

[11] David Reddick. https://vrvisiongroup.com/exciting-things-are-coming-to-unity-development-for-vr-ar-and-ai-enthusiasts/, 24th Of October 2018.

[12] https://opencv.org/.

[13] https://en.wikipedia.org/wiki/OpenGL.

[14] https://docs.nvidia.com/gameworks/content/gameworkslibrary/graphicssamples/opengl_samples/gl-samples.htm.

[15] https://www.cgal.org/.

[16] https://github.com/Microsoft/vcpkg.

[17] https://docs.unity3d.com/ScriptReference/GL.TRIANGLES.html.

[18] https://doc.cgal.org/latest/Triangulation_3/index.html.

[19] https://visualstudio.microsoft.com/.