ENVIRONMENT MODELLING FOR ROBOT VISION USING KINECT

Supervisor: Professor Dr. Md. Haider Ali

Ahnaf Tabrez Alam	**12101090**
Bashir Ahmed Rakib	**12101084**
Ekramul Hoque	**12101096**

Department of Computer Science and Engineering

School of Engineering and Computer Science

BRAC University

Submitted on: 22nd December 2015

DECLARATION

We, hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole or in part, has been previously submitted for any degree.

Signature of Supervisor

Professor Dr. Md. Haider Ali

Signature of Author

Ahnaf Tabrez Alam

Bashir Ahmed Rakib

Ekramul Hoque

ABSTRACT

This research represents an integrated approach of reconstructing three dimensional environments for robotic navigation. It mainly focuses on three dimensional surface reconstructions of the input data using Kinect, a depth sensor. With an increase in the application areas making use of point clouds, there is a growing demand to reconstruct a continuous surface representation that provides an authentic representation of the unorganized point sets and render the surface for visualization.

The main goal of this research is the study of various surface reconstruction algorithms and the creation of a three dimensional model of an object and/or an entire three dimensional environment from a set of point clouds. It starts by scanning an environment or an object using Kinect and store the point cloud generated using OpenGL and Microsoft Visual Studio. Then it focused on creating a mesh out of the stored point cloud in MATLAB, using a computational geometric approach called Delaunay Triangulation. Finally, combining surfaces and applying surface reconstruction method the three dimensional model is obtained.

ACKNOWLEDGEMENT

This thesis was suggested by Professor Dr. Md. Haider Ali, Chairperson, Department of Computer Science and Engineering, BRAC University. This is the work of Ahnaf Tabrez Alam, Bashir Ahmed Rakib and Ekramul Hoque, students of BRAC University, studying Computer Science and Engineering from the year 2012. The document has been prepared as an effort to compile the knowledge obtained by us during these four years of education and produce a final thesis which innovatively addresses one of the issues of research in computer vision.

We would like to express our gratitude to Almighty Allah (SWT) who gave us the opportunity, determination, strength and intelligence to complete our thesis.

We would like to thank our supervisor, Professor Dr. Md. Haider Ali sincerely for his consistent supervision, guidance and unflinching encouragement in accomplishing our work.

We also want to acknowledge Ms. Dilruba Showkat, Lecturer, Department of Computer Science and Engineering, BRAC University, Dr.M.Tanseer Ali, Assistant Professor, Department of Electrical and Electronics Engineering, AIUB and our peer Khaled Mohammad Ali who have consistently shown their interests in our work and assisted us in preparing the report.

Last but not least, our gratitude towards BRAC University Robotics Club for providing the Kinect sensor for our research.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

In the field of computer graphics surface reconstruction is a challenging problem with a wide range of application like medical imagery [1], unmanned aerial vehicle [2], video games and other graphic applications. Surface reconstruction from raw geometric data has received an increasing attention due to the ever range of geometric sensors which tends to be very expensive.

Since Microsoft released the Kinect camera, which has a depth sensor in addition to the RGB-sensor, a quite cheap hardware is available that is able to extract 3D data of its surroundings. Nowadays various applications like unmanned aerial vehicle, human recognition, ground mobile robot has implemented this Kinect sensor.

Surface reconstruction using Kinect sensor can be tricky and a lot of researches has taken place to address this particular issue. Among these researches, Kinect Fusion developed by Microsoft in 2011, a popular technique that creates three dimensional reconstructions in real-time.

On their research, depth data streamed from a Kinect sensor are fused into a single global implicit surface model of the observed scene in real time. The current sensor pose is obtained simultaneously by tracking the live depth frame relative to the global model using a course to fine iterative closest point algorithm. The system also works in complete darkness mitigating any issues concerning low light conditions and RGB-D based systems. They have been successful to map medium sized room with volumes of $< 8m^3$ [3].

However the technique is executed almost exclusively on the graphics card allowing both tracking and mapping to perform at the frame-rate of the Kinect.

Such expensive requirement calls for a light weight model where our system emerges. Our system architecture uses a 64-bit Windows operating system, a Kinect to stream depth maps, OpenGL and MATLAB to process and render the final three dimensional reconstructions.

1.1 Motivation

Emergence of three dimensional printers has revolutionized the field of computer graphics and design modeling. Being able to replicate a real object as accurately as possible is a very complex task. The scanning techniques for data acquisition and hardware requirement for post processing is still very expensive. Thus the challenge to simplify such complex task has been the core motivation for us.

Amongst the various input devices which provide depth information by scanning an object, Kinect sensor has gained rapid popularity around the world for its versatile features satisfying the purpose of 3D scan and texture mapping. Robotics club of BRAC University first introduced this magnificent device to us during their exhibition of a gesture detecting robot a few years ago. Further study on this gadget made us interested to use it in our research.

1.2 Thesis Outline

Chapter 1 is the formal introduction of the thesis. We have discussed our motivation and approach towards our proposed topic in this chapter.

Chapter 2 is the background study that covers all the important basics needed for this research along with their formal definitions and representations. In this chapter we have described the concepts on Scanning Techniques, Kinect Sensor, Graphics Rendering Basics, Triangulation Basic, Delaunay Triangulation and Surface Reconstruction. Moreover, we have also introduced the tools we have used.

Chapter 3 focuses on the proposed work. Firstly, we have described our whole workflow to achieve the results following our proposal. Then we have discussed the Data Acquisition Technique followed by Data Calibration and Processing. Later, we have elaborated our approach of acquiring three dimensional models of both Object and Environment. Additionally we have attached all the pseudo codes of our algorithms with each part in this chapter.

Chapter 4 is the result analysis part where we have explained our acquired results following our approach towards proposed work. We have attached visual of both the input and generated output in this chapter.

In chapter 5, we have made a comparative analysis of our results based on some performance metrics.

In chapter 6, we have talked about the limitations of our project as well as mentioned about some approaches to overcome those limitations and works that can be derived from this research.

2. BACKGROUND STUDY

2.1 Scanning Techniques

3D reconstruction is a technique of collecting 3D data via an input device and processing it to a virtual 3D model. It is a widely used technique in visual computing, since modern applications like games or robotic vision tend to be more photo realistic leading to high costs in content creation [4].

2.1.1 Methods

According to the equipment used, traditional methods for acquiring dense point cloud of object for 3D reconstructions can be categorized into two main streams i.e. the multi view stereo route and the 3D laser scanner route. In recent years depth camera has become one of the popular 3D measurement equipment due to its small size, portability and ability to perform real-time data acquisition.

a) Multi-view stereo: The process includes taking two or more images of the same object from different angles with a certain degree of overlap, finding corresponding points within the overlapping region to estimate the relative position of the cameras and reconstructing the 3D coordinate of the object [5].
b) Laser scanner: The structure includes a laser source that can beam laser, a sensor that measures the distance to the surface, control unit and so on. These type of scanners measures the distance by calculating the time taken for a pulse of light to make a roundtrip i.e. a laser is used to emit a pulse of light and the amount of time before the reflected light is seen by a detector is timed. However the speed of light makes the requirement very strict thus leading to expensive hardware. Nevertheless, accuracy is very high close to sub-micron level for these scanners [5].

c) Depth camera: Like any other RGB camera, video capturing process is similar with a depth camera; however it also takes down the distance between the object and camera on the basis of time of flight [5].

2.2 Kinect

The Kinect camera was first introduced by Microsoft in cooperation with Prime Sense in June 2009. It has been implemented in various applications like unmanned aerial vehicle [2] , human recognition, 3D model reconstruction [3] , ground mobile robot navigation [6] and medical applications [1].

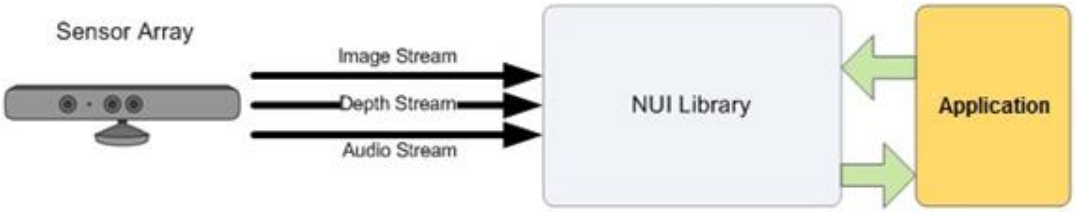


Figure 1: Kinect Workflow [7]

2.2.1 Hardware Features

a) It has a standard CMOS color sensor, to retrieve an RGB picture
b) To gather depth information it includes an infrared laser which shoots IR rays through the whole scene and a CMOS sensor records them. The distance to camera is measured by the size and the position of the recorded IR dots [4].
c) In addition the camera has a built in 3D microphone to get audio information.

Table 1: Hardware Specification

Resolutions	640x480 pixels of raw distance
Sampled Resolutions	1600x1200 pixels
Depth image rate(frequency)	30Hz
Measurement accuracy	4mm
Capture range	0.8m-3.5m
Price	$200

Retrieving all the data on the chip, the system produces a depth map which is aligned with the RGB picture and they are combined to a single RGBD texture that is sent over the USB port [4]. In addition Microsoft has provided SDK for developers to create open source projects and enhance individual study.

2.2.2 Kinect SDK

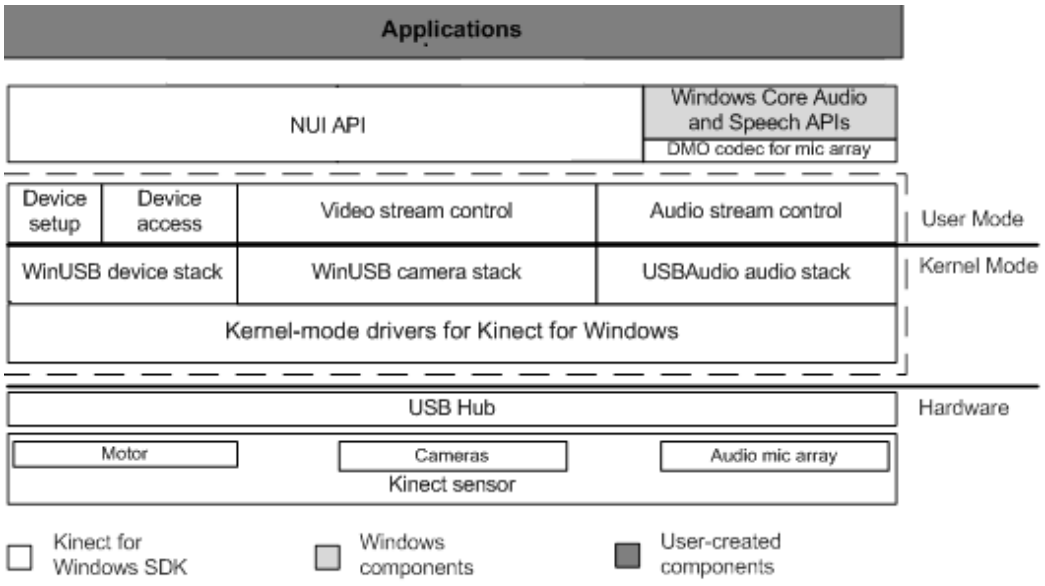


Figure 2: Kinect SDK [7]

Kinect SDK provides native API which allows access to features like depth streaming, color streaming, gesture recognition, facial tracking etc. The Natural User Interface (NUI) is the core of the Kinect for Windows API which manages audio stream, color image data and depth image data. Detailed description of this API will be discussed through implementation later in the research [7].

2.3 3D Graphics and Rendering Pipeline

In computer graphics, rendering pipeline is the process of producing image on the display from the world description. Pipeline can greatly improve the throughput due to its massive parallelism [8].

3D Graphics Rendering Pipeline accepts description of 3D objects in terms of vertices of primitives shapes (triangle, point, line and quad) and produces the color value for the pixels on the display [8].

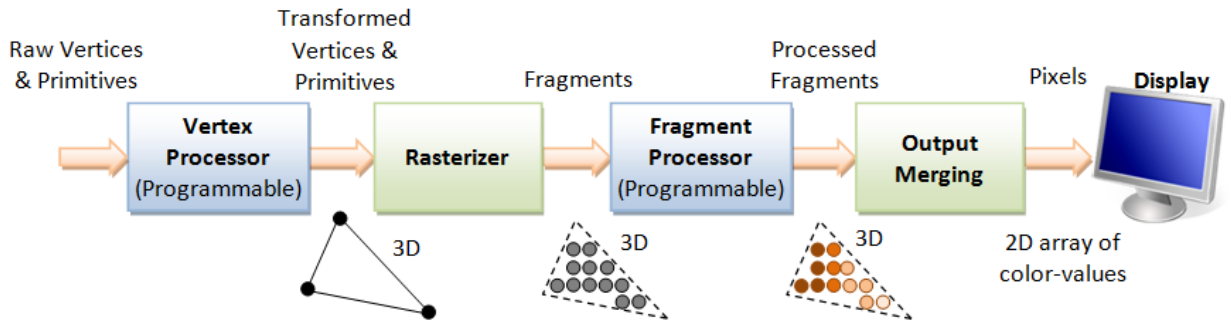


Figure 3: 3D Graphics and Rendering Pipeline [9]

Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

2.3.1 Key stages

a) Vertex Processing: Process and Transform individual vertices.

b) Rasterization: Convert each connected vertices into set of fragments. A fragment represents as a pixel in 3D space, which is aligned with the pixel grid, consisting position, color, normal and texture.

c) Process Individual fragments.

d) Convert all the fragments into 2D color pixel for display.

2.4 Image Registration

Image registration is the procedure of transforming different sets of data into a single coordinate system. These sets of data may vary in their source, time, viewpoints etc. Image registration algorithms try to recover the transformation parameters that express a mapping of one image onto another, where both of them are from the same scene [10]. One of the main uses of registration is to find the differences in the underlying scene.

To get a full view of the desired 3D scene, rotation and stitch operations may require over the data sets. The Rotation is a geometric transformation of data about one of the axes of a Coordinate system. The following three basic rotation matrices rotate vectors by an angle θ about the x, y, or z axis, in three dimensions, using the right hand rule — which codifies their alternating signs.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \ldots\ldots\ldots (1)$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \ldots\ldots\ldots (2)$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \ldots\ldots\ldots (3)$$

Now, Stitching is the process of combining multiple adjacent images together. While constructing a large scene for an environment, we need to stitch the point cloud data. Today's digital maps and satellite photos are output of Image stitching algorithms. The panorama view of digital camera is also a result of stitching.

2.5 Surface Triangulation

Surface Triangulation is the process, in which a surface is subdivided into multiple pieces of triangles usually considered that each side of a triangle (except from the outer edges) is entirely shared by another triangle [11]. This is an essential process to visualize a complex shaped surface in a more simpler way. When considering a surface as a collection of triangles then it is easier to formulate a mathematical process to define, determine and manipulate the characteristics of that surface.

2.5.1 Triangulation over a Set of Points

Triangulation of a set of points refers having a set P of n number of points in the plane and a set E of straight line segments that joins two points from P and no line segments intersects each other except in the joining points [12]. Different triangulation algorithms can be applied on a provided set of Point Cloud Data to achieve a triangulated surface which further can be used to generate concrete 3D surface through different polygonization technique.

2.5.2 Triangle Mesh

In computer graphics, a triangle mesh means a collection of vertices, edges and faces in that resides together in forms of multiple triangles to define a 3D object. Tough Triangulation and Triangle meshes are completely different; they are interconnected because to form a triangle mesh, at first, we have to go through the process of triangulation.

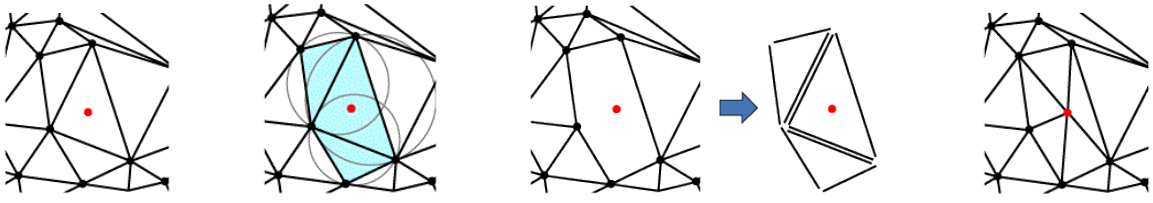
2.5.3 Triangulation Techniques

There are several triangulation techniques in practice. For this paper, we will be focusing on triangulation over a point set which resides on a 2D plane and then we will formulate a procedure to obtain a 3D triangle mesh from our triangulated points.

One approach of triangulation would be iterating through all of the points and connecting each adjacent point as an edge of a triangle [13]. However, selecting these adjacent points are critical and we might end up with a very much distorted shaped triangulated surface from the desired if adjacent points are not chosen efficiently.

2.6 Delaunay Triangulation

In computational geometry and mathematics, one of the efficient and widely approached triangulation techniques is the Delaunay Triangulation. We had a set of points P and assuming all the points are on the same plane. The Delaunay triangulation for the set P would be DT (P) such that there is no point in P which is inside the circumcircle of any triangle in DT (P). Things should be noted that, for a set of points on the same line there exists no Delaunay triangulation. On the other hand, there can be more that one Delaunay triangulation for four or more points which are on the same circle.



Red dot = new vertex.

Vertex falls inside the circumscribed circle Of the two indicated triangle

Adjacent edges of those two triangles removed.

Form new triangles with the remaining edges.

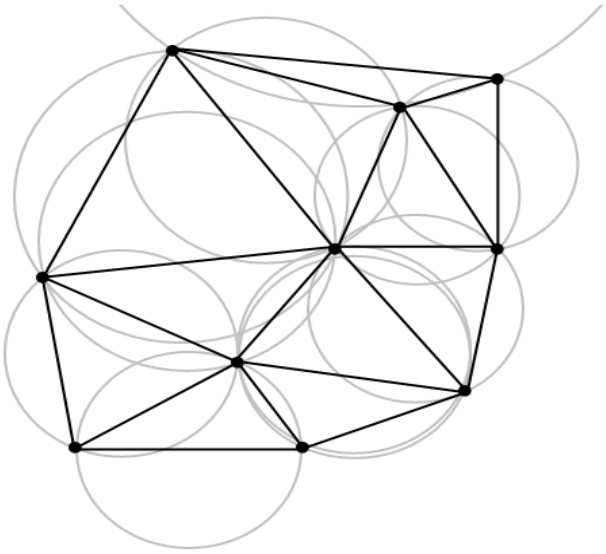


Figure 4: Delaunay Triangulated Points [22]

In the figure we have projected a set of Delaunay triangulated points where no points are inside the circumcircle of any other three points.

2.6.1 Process of Achieving Delaunay Triangulated Points

There are several ways of getting Delaunay triangulated points from a given set of points on a plane. The basic algorithm to approach is

1. Construct a super triangle which contains the convex hull of the entire point set.
2. Take the first point to be triangulated.
3. Connect the point from the three vertices of the super triangle.
4. Then draw a circumcircle of the triangle.
5. Determine another point inside the triangle from the point set.
6. Connect the point from the vertices of the triangle inside which the point is contained.
7. Iterate from 4 to 6 until all the points are iterated.
8. When the iteration is complete remove the vertices of the super triangle that was created at first.

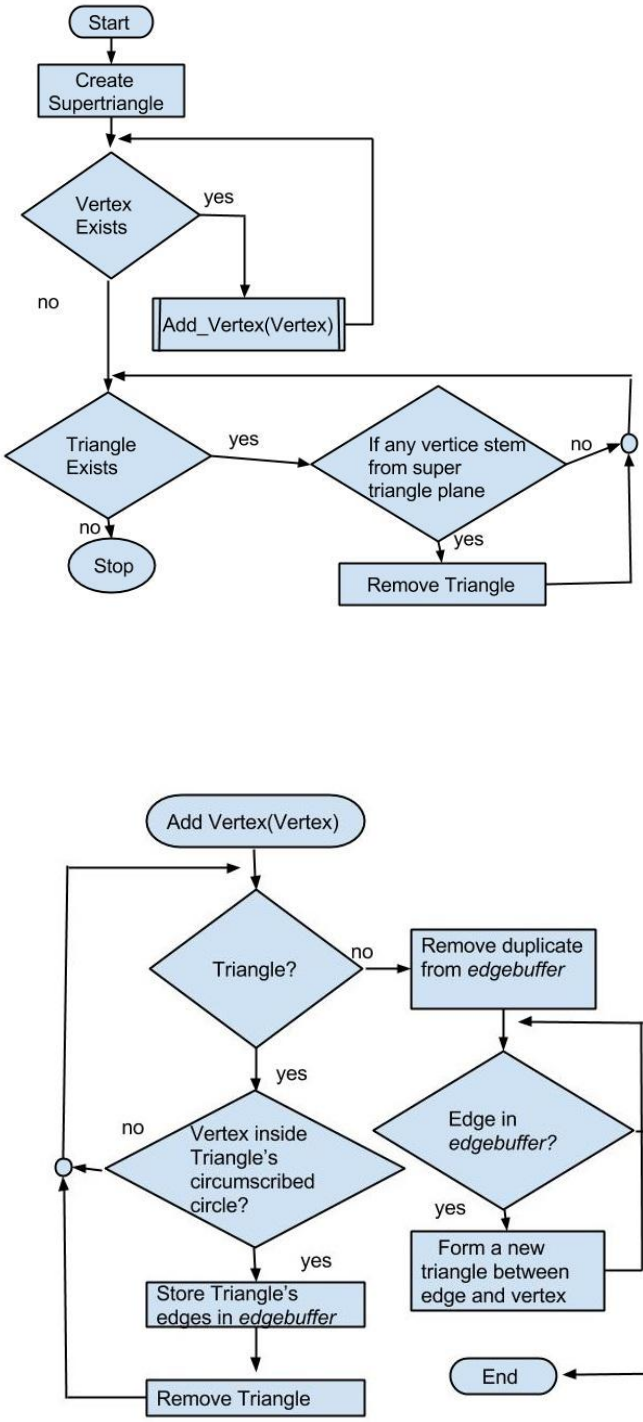


Figure 5: Flow Chart of Delaunay Triangulation

2.6.2 Normal

Surface Normal, in a three dimensional space, refers to a vector which is perpendicular to a surface of a specific point located in the boundary of a shape. Each face in a mesh has a perpendicular unit normal vector. The Face Normal points away from the front side of the face [14]. On the other hand, a Vertex Normal at a vertex of a polygonal shape is a directional vector associated with that vertex, intended as a replacement to the true geometric normal of the surface. It is usually computed as the normalized average of the surface normals of the faces that contain that vertex.

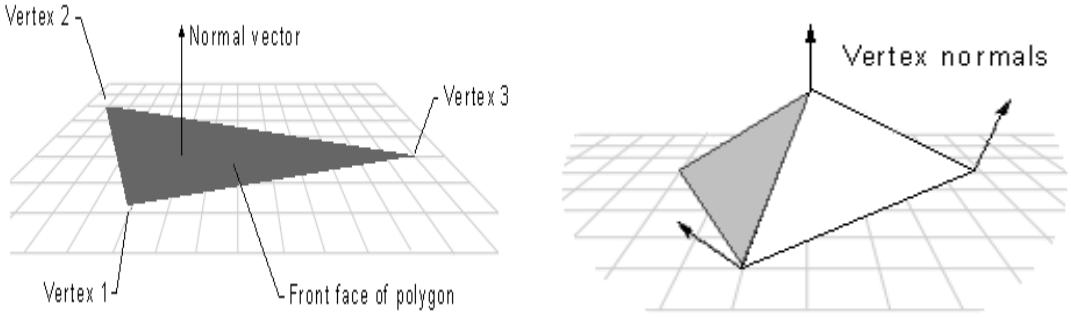


Figure 6: Face and Vertex Normal [14]

The above algorithm leaves us with a Delaunay triangulated form of our given point set where the angles of every triangles are maximized.

2.7 Surface Reconstruction

In context of computer graphics, Surface reconstruction, is the process by which a fairly accurate outer shape of an object can be generated from partial information such as the known peripheral points of the same. It leads to the visual implementation of an object, by defining points in a 3 dimensional array of X, Y and Z axis. However, different input devices (i.e. laser scanner, photogrammetric image measurements, geometric sensors, motion sensors etc.) provide an unorganized point cloud which makes the reconstruction very difficult. Moreover, partial data, unavoidable presence of noise and unable to define the topology of original surface makes the process immense to generate an accurate shape [15].

According to Navreet, the main steps of surface reconstruction are:

- Eradication of noise and reducing computation time by down sampling the point set
- Defining the peripheral topology of the object
- Production of polygonal surface model
- Refining the model by essential editing [16]

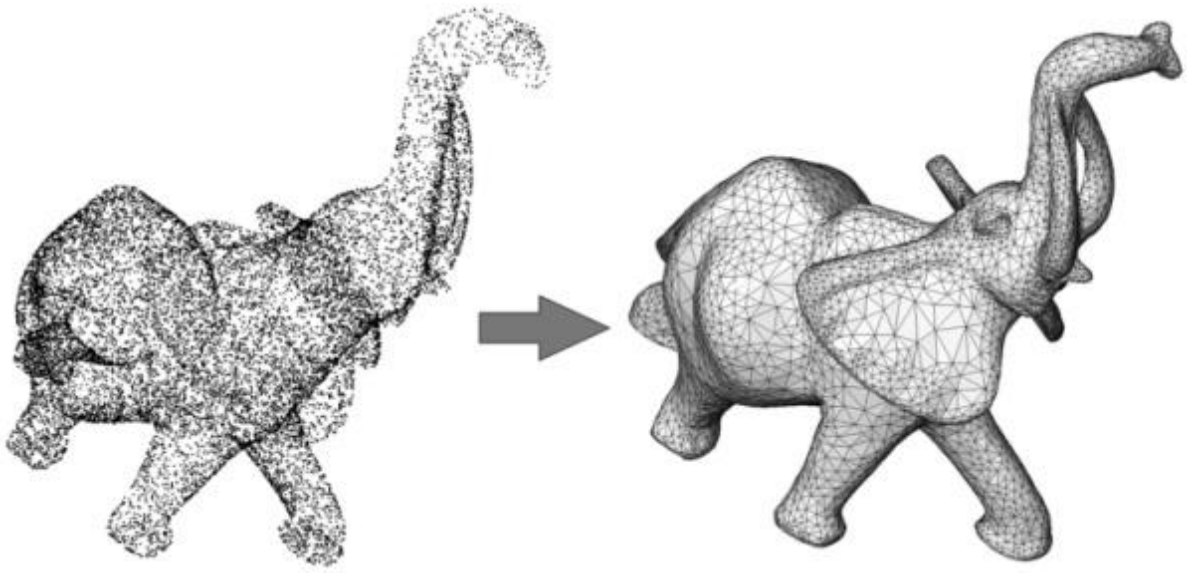


Figure 7: Surface Reconstruction [17]

2.7.1 Reconstruction Methods

Navreet also stated three major surface reconstruction methods [16]. A brief description of them is given below.

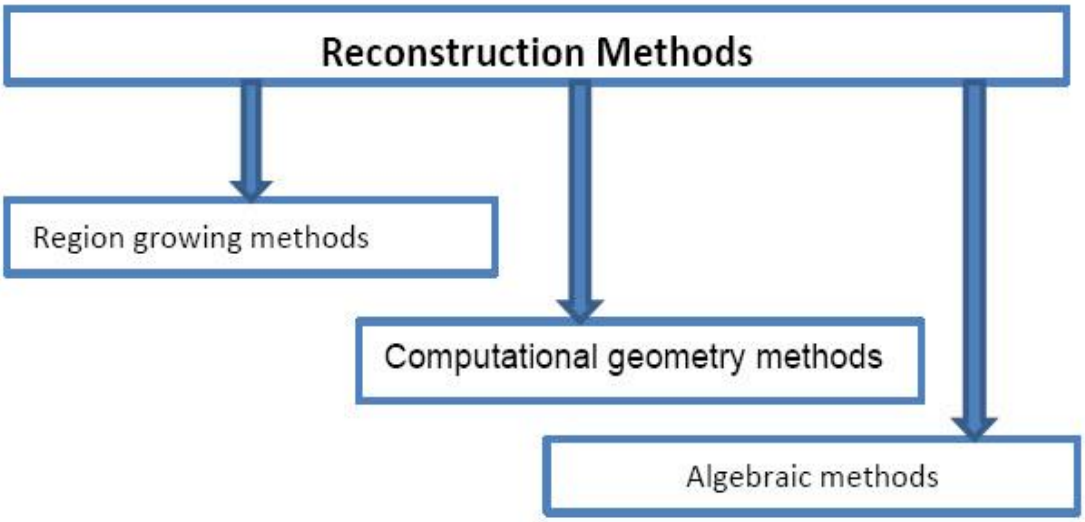


Figure 8: Reconstruction Methods [16]

Region Growing Methods spread information and gradually form the surface of the object. Bernardini's Ball-Pivoting algorithm is an example to this method.

Computational geometry methods rely on the appliance (i.e. Delaunay triangulation). They interpolate the original points. Therefore, noise can heavily affect these methods efficiency. Alpha Shapes and the Crust algorithm fall in this category.

Algebraic methods focus on developing a suitable function for all the points. Their prime goal is to make the function efficient to avoid noisy output. The signed distance based and implicit functions are example to this group.

2.7.2 Shading

Shading refers to portraying the depth sensitivity in a 3D model by altering levels of darkness. It is the process of varying the color of an object, according to its angle to lights and its remoteness from lights to construct a photorealistic output model. It is performed during the rendering process.

Flat shading is a lighting approach that shades each polygon of an object according to the angle between the direction of the light source and the polygon's surface normal, their respective colors and the intensity of the light source. It is usually used for fast rendering. In contrast, in Smooth Shading the color differs from pixel to pixel. It predicts that the surfaces are curved. Interpolation techniques is used calculate the values of pixels between the vertices of the polygons. Gouraud and Phong's shading are part of smooth shading [18].

2.8 Tools

The key tools applicable for Surface reconstruction can be listed as follows:

2.8.1 OpenGL

Renowned software interfaces for graphics hardware, in short an API which contains over hundreds of functions to create interactive three dimensional graphics application. It is extensively used in the fields of CAD, virtual reality, scientific visualization, information visualization, flight simulation and video games.

OpenGL is OS independent, which do not include facilities like windowing tasks or obtaining user input. One must build up their model using a set of geometric primitives defined by this library. It supports three classes of geometric primitives: points, line segments and closed polygons [19].

2.8.1.1 OpenGL rendering Architecture [9]:

- Display List: All data, geometry (vertex) and pixel data can be stored in a display list.
- Vertex Operation: Each vertex and normal coordinates are transformed from object coordinates to eye coordinates. If lighting is enabled, color of the vertex can be calculated.
- Primitive Assembly: Projection matrix transforms the primitives and clips the volume. Then the 3D scene is mapped to window space coordinates by perspective division.
- Pixel Transfer Operation: The data are either stored in texture memory or rasterizer directly to fragments.
- Texture Memory: Texture Images are loaded into texture memory to be applied onto geometric objects.
- Rasterization: Conversion of both geometric and pixel data into fragment.
- Fragment Operation: Convert fragment to pixels onto frame buffer.

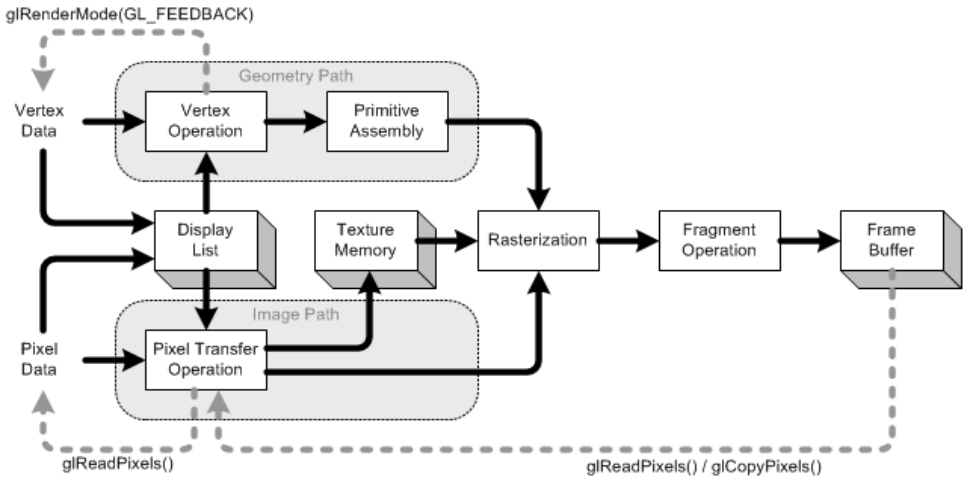


Figure 9: OpenGL Rendering Architecture [9]

2.8.1.2 Application program [19]:

i. GL: core graphics capability. Specify graphics primitives, attributes, geo-metric transformations and many other.
ii. GLU: utilities on top of GL. Functions useful for drawing and transforming objects.
iii. GLUT: input and windowing functions.

2.8.1.3 Key features:

i. Model View Matrix and Projection Matrix.
ii. Viewport transform.
iii. Z-buffer and Hidden-Surface Removal for output merging.
iv. Provides point sources, spotlight and ambient light.
v. Texture Filtering.

Our system uses the data array which holds a copy of the image we get from the Kinect during the scanning process which will be discussed more in the implementation section. The input stream is then displayed by setting OpenGL texture and camera viewpoint functions.

2.8.2 MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization and programming environment. Furthermore, it has sophisticated data structures, contains built-in editing and debugging tools and support object oriented programming. It also has easy to use graphics commands that make the visualizations of results immediately available. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization and several other fields of applied science and engineering. These factors make MATLAB and excellent tool for teaching and research [20].

In 2015 MATLAB has improvised its Computer Vision toolbox which provides algorithms and functions for 3D reconstruction and 3D point cloud processing [21]. Some of the features used in this research include:

1. Scaling and down sampling of point cloud data.
2. Transformation, rotation and registration of point cloud data.
3. Read, Write and Store Point Clouds.

Moreover, MATLAB provides an optimized function for Delaunay Triangulation over point cloud data. Given these factors and other computational performances, MATLAB has been chosen for simulating our results.

3. PROPOSED WORK

In this paper we are proposing to design a complete system that will take a real scene and/or object as input and generate its 3D model that can be rendered on screen and also be rotated, zoomed in and out. Additionally, for the environment part we have recorded the RGB data (Red Green Blue) along with the depth information. Also, we will be trying to map the RGB data on our generated surface so that our generated surface seems identical to the input environment.

In order to achieve our desired system, we have divided our working process into several sub categories-

a) Data Acquisition,
b) Data Calibration,
c) Data Processing and
d) 3D Model.

After data calibration we have divided our research work into two parts-

a) Object Modeling and
b) Environment Modeling.

Both object model and environment model consists of data processing and 3D model.

In the upcoming few sections we will see the pseudo codes of the basic algorithms that we need to formulate the proposed system following by a little elaboration. After that, the result analysis section is divided into two sub divisions that consists object modeling and environment modeling. The results of the object modeling segment have been formatted categorizing into three phases- a) input, b) data processing and c) output similarly the results of the environment modeling are formatted into two parts a) input and b) process and c) output.

3.1 Workflow

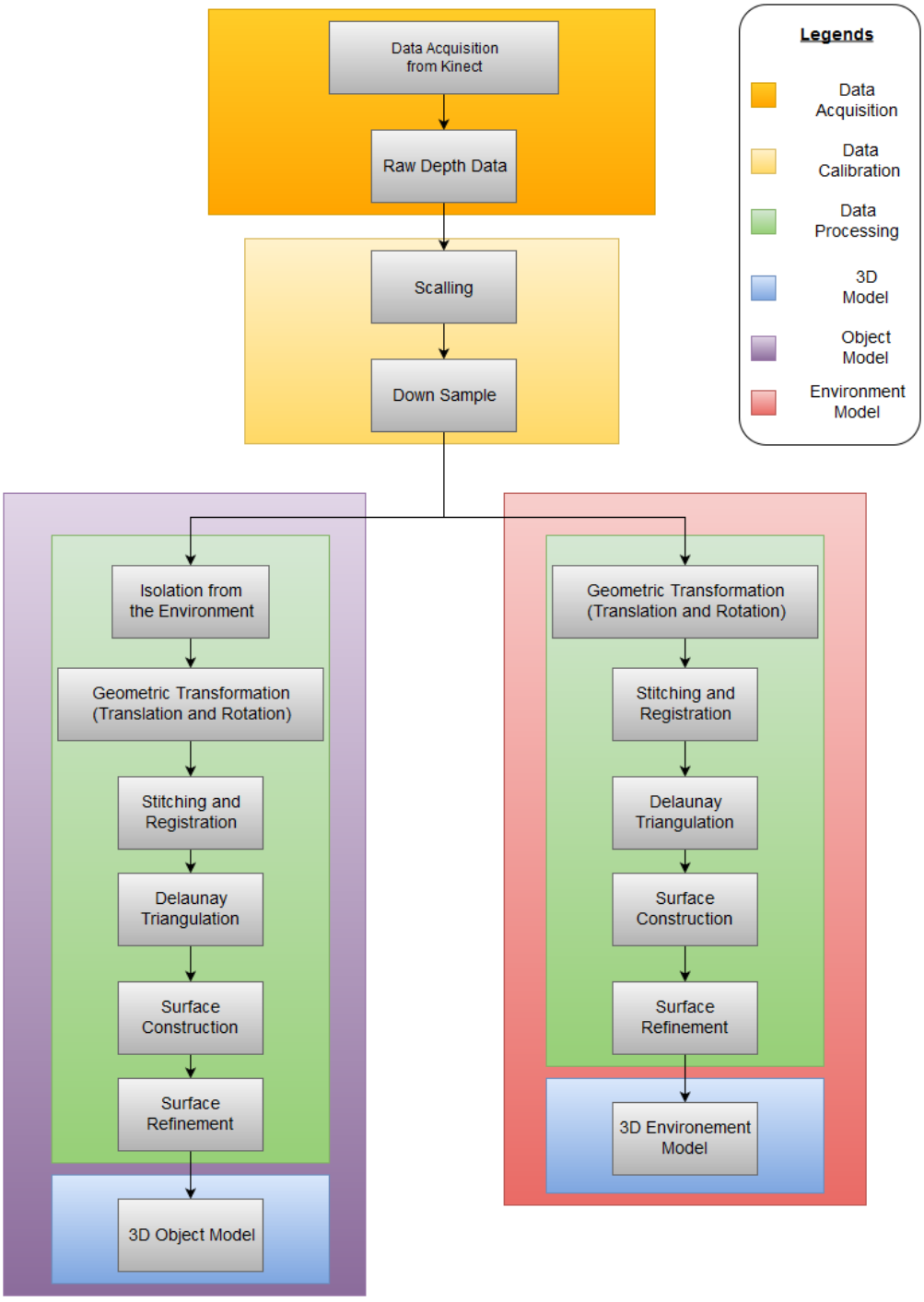


Figure 10: Work Flow

3.2 Data Acquisition

3.2.1 Environment Setup

Kinect SDK provides API which recognizes a gesture as user input. These APIs are compatible for C++ development and are required to include in the project to call those functions. We have used Visual Studio 2012 to integrate OpenGL and C code.

APIs used in the project:

a) NuiApi: Core features like audio stream, color and depth image stream.
b) NuiImageCamera: defines the NUI image and camera services.
c) NuiSensor: Refers to the Kinect plugged in to the environment.

3.2.1.1 Kinect Coordinate System

The Kinect uses a Cartesian coordinate system centered at the RGB camera lens. The positive Y axis points up, the positive Z axis points where the Kinect is pointing and positive X axis points to the left.

3.2.1.2 Alignment

The Kinect uses its depth camera and RGB camera to map a coordinate into space. This mapping follows a certain registration process so that the color and depth data do not overlay upon each other giving poor depth map quality. The Kinect SDK provides a function which can identify a pixel in the RGB image with a particular point in the depth image. In particular, we have stored the column and row (i.e. x and y coordinate) of the color pixel in order of each depth pixel.

3.2.2 Initialize Kinect

Pseudo code

Check Kinect ()

Output: sensor S;

```
1. numSensors= get sensor count from NUI Sensor Api.
2.     if numSensors is equals to zero
3.         then return false
4.     end if
5.     S= uses color and depth stream from NUI Sensor.
6.     return S
```

The process contains two parts: First we find an attached Kinect Sensor, and then we initialize and prepare the Kinect to read data from it. Prior to this function we have declared a constant/handle as an identifier of the Kinect Camera.

3.2.3 Getting Depth frame from the Kinect

Pseudo code

Get Depth Data (S, F)

Input: depth sensor S, frame F

Output: depth data array D

```
1.  get S from frame  F
2.  lock the frame
3.  Initialize D with empty
4.  for each point p ϵ F do
5.      X = get row data from the frame f for p
6.      Y  = get column data from the frame f for p
7.      Z = get depth data from the frame f for p
8.      map (X, Y, Z) to color
9.      add (X, Y, Z) to D
10.     Write D in buffer
11. end loop
12. release frame
13. return D
```

In this function, first we acquire frame from the sensor and lock it so that it doesn't get corrupted while we are reading it. Then we retrieve the depth pixel data (row, column and depth in the depth image) and map it on to the row and column of the pixel in the color image. The 3D coordinates of that pixel is retrieved as well and written into the buffer. Finally we release the frame so that the Kinect can use it again.

3.2.4 Getting RGB frame from the Kinect

Pseudo code

Get RGB Data (S, F)

Input: rgb sensor S, frame F

Output: rgb data array C

```
1.  get S from frame  F
2.  lock the frame
3.  Initialize C with empty
4.          for each point p ϵ F do
5.          R = get red data from the frame f for p
6.          G = get green data from the frame f for p
7.          B = get blue data from the frame f for p
8.          add (R, G,B ) to C
9.          Write C in buffer
10. end loop
11. release frame
12. return C
```

In this section, we have determined RGB color for each depth pixel. The color image frame is in BGRA format, one byte per channel, laid out row by row. We converted each 4-byte BGRA value into RGB float value. The rest of the execution is similar to the depth retrieving function regarding frame lock and release.

3.3 Data Calibration

3.3.1 Scaling

Pseudo code

Scale (P, M)

Input: raw point cloud P, max of magnifying scale M

Output: scaled point cloud P_1

Initialize X = x axis of P, Y = y axis of P, Z = z axis of P

1. **for** i = 0 to length of X **do**
2. X_{MIN} = **min** of X
3. X[i] = X[i] – X_{MIN}
4. X_{MAX} = **max** of X
5. X[i] = X[i] / X_{MAX}
6. X[i] = X[i] * M
7. Y_{MIN} = **min** of Y
8. Y[i] = Y[i] – Y_{MIN}
9. Y_{MAX} = **max** of Y
10. Y[i] = Y[i] / Y_{MAX}
11. Y[i] = Y[i] * M
12. Z_{MIN} = **min** of Z
13. Z[i] = Z[i] – Z_{MIN}
14. Z_{MAX} = **max** of Z
15. Z[i] = Z[i] / Z_{MAX}
16. Z[i] = Z[i] * M
17. **end loop**
18. P_1 = make point cloud (X, Y, Z)
19. **return** P_1

The function takes a sample point cloud with a given parameter M. For each point, the x, y, z coordinates are taken. The minimum value of each coordinates is retrieved and subtracted from all the points in that axis. Then maximum value is taken which also divides all the points in that axis. Finally the coordinate is multiplied with the given parameter M to return a new set of points.

3.3.2 Down Sample

Pseudo code

Downsample (P, G_S)

Input: initial point cloud P, size of grid axis G_S

Output: down sampled point cloud P_1

```
1.  create grid with axis size G_S
2.  divide the P into grids along XY plane
3.   G = list of all the grids
4.  for each grid g ϵ G do
5.          A = list of all the points in grid g
6.          A_Z-AVG = average Z axis coordinate value of A
7.          for each point a ϵ A do
8.                  A_Z = Z axis coordinate value of a
9.                  if A_Z = A_Z-AVG
10.                         then keep a in A
11.                         else drop a from A
12.                 end if
13.         end loop
14. end loop
15. P_1 = make point cloud by joining the grids of G
16. return P_1
```

The function receives a sample point cloud data with a grid size. The sample is then divided into dimensions of the given grid size. For each grid, the average of Z-axis coordinate is calculated and compared with all the other points inside that grid. Points which do not match with the average calculated are dropped while the others are kept into a new matrix and returned.

The function is used to improve the triangulation performance as it removes duplicate variables and redundant computation.

3.4 Data Processing

3.4.1 Isolation from the Environment

Pseudo code

Isolation from Environment (P, X_{MAX}, X_{MIN}, Y_{MAX}, Y_{MIN}, Z_{MAX}, Z_{MIN})

Input: point cloud P, object range in x, y and z axis (X_{MAX}, X_{MIN}, Y_{MAX}, Y_{MIN}, Z_{MAX} and Z_{MIN})

Output: isolated object point cloud P

```
1.  for each point p ϵ P do
2.          P_X = X axis coordinate value of p
3.          P_Y = Y axis coordinate value of p
4.          P_Z = Z axis coordinate value of p
5.          if P_X < X_MIN or P_X > X_MAX
6.                  then drop p from P
7.                  else if P_Y < Y_MIN or P_Y > Y_MAX
8.                          then drop p from P
9.                          else if P_Z < Z_MIN or P_Z > Z_MAX
10.                                 then drop p from P
11.                                 else keep p in P
12.                         end if
13.                 end if
14.         end if
15. end loop
16. return P
```

The function takes a sample point cloud with ranges of X, Y, Z coordinates. The sample is passed through a loop where points which fall between these ranges are kept while others are discarded.

The function is used mostly to isolate an object for triangulation. From an actual scene the background is subtracted which allows the object to be triangulated and push into further processes.

3.4.2 Geometric Translation

Pseudo code

Geometric Translation (P, T_X, T_Y, T_Z, R_{AXIS}, R_Θ)

Input: point cloud P, translation parameter over x, y and z axis (T_X, T_Y, T_Z), rotation axis R_{AXIS} and angle R_Θ

Output: transformed point cloud P_1

1. **for** each point p ϵ P **do**
2. P_X = X axis coordinate value of p
3. $P_X = P_X + T_X$
4. P_Y = Y axis coordinate value of p
5. $P_Y = P_Y + T_Y$
6. P_Z = Z axis coordinate value of p
7. $P_Z = P_Z + T_Z$
8. **end loop**
9. A = affine rotation over R_{AXIS} with angle R_Θ
10. P_1 = point cloud rotation (P, A)
11. **return** P_1

The function takes a sample point cloud with translation parameter, rotation axis and rotation angle. For each points P, its coordinate is translated according to T_X, T_Y, and T_Z. Then new point set is rotated along the given axis, R_{AXIS} at an angle R_Θ.

The function is implemented at stages where we have merged all the point clouds taken from different angles. Furthermore this function aids into the orientation of the coordinates when we apply registration and stitching.

3.4.3 Stitching

Pseudo code

Stitch (P, P_{REF})

Input: a list of all the point clouds P, a reference point cloud P_{REF}

Output: a singled merged point cloud P_1

1. place P_{REF} in P_1
2. **for** each point cloud p ϵ P **do**
3. determine the position of p in reference to P_{REF}
4. P_{TEMP} = place p in its new position
5. P_1 = merge P_1 and P_{TEMP}
6. **end loop**
7. **return** P_1

The function basically takes an array of point clouds and merges them in a large scaled single coordinate system. A reference point cloud is used to determine the position of the each variable scene, and it merges the point clouds accordingly.

The function is used create a complete view using the multiple amount of point cloud data retrieved from several scene of the same object or environment with various point of view.

3.4.4 Delaunay Triangulation

Pseudo code

DelaunayTriangulate (P)

Input: raw point cloud set P: $P_{1.....n} \epsilon P$

Output: Triangulated Point Cloud Set dt: $dt_{1,2...m} \epsilon dt$

```
1.  Bag triangleBag, edgeBag
2.  Create Super Triangle, t1
3.  triagleBag.push(t1)
4.  for i = 1 to n do
5.      for j = 1 to size of triangleBag do
6.          Circle c = draw circum circle( tj )
7.          if coordinate(Pi ) ϵ area
8.                  then    edgeBag.push(edges(tj))
9.                          triangleBag.remove(tj)
10.         end if
11.     end loop
12.     remove duplicate edge (edgeBag)
13.     for k = 1 to size of edgeBag do
14.         triangle = formTriangle(edgeBag.get(k), Pi )
15.         dt.push(triangle)
16.     end loop
17. end loop
18. for l = 1 to size of dt do
19.     if dtl contains vertex from initially created Super Triangle
20.         then    remove dtl
21.     end if
22. end loop
23. return dt
```

The function takes a set of points P as input that resides in a 2D space. Before calling this function we need to choose a base plane from our 3D point cloud data where we will like all the points to be triangulated. We have to calculate the convex hull of that point set to be delivered as input in the function for creating the initial super triangle. The super triangle basically fits the whole convex hull of the input point set. After through the iteration over the point set according to the algorithm we get a set of triangulated points as output.

3.5 Object 3D Model

Pseudo code

Object 3D Model (P)

Input: a list of all the point clouds P

Output: reconstructed 3D model M

1. **for** each point cloud p ϵ P **do**
2. scale the point cloud P
3. down sample P
4. reduce the noise of P with threshold value
5. isolate the object from environment
6. **end loop**
7. define a reference point cloud P_{REF} from P
8. **for** each point cloud p ϵ P **do**
9. apply geometric transformation on P based on P_{REF}
10. **end loop**
11. P_1 = stitch all the point clouds P
12. apply Delaunay triangulation on P_1
13. M = surface reconstruction over P_1
14. display M
15. **return** M

The function takes all the point clouds related to the target object as input. Initially, it calibrates the point cloud data and reduces the noise with threshold value. It then isolates the object from the environment in each point cloud. After defining a reference, it transforms all the other point clouds accordingly. Stitching is performed to achieve a single coordinate system for all the scenes. Finally, Delaunay triangulation and surface reconstruction is performed over the merged point cloud. The output is displayed in 3D model.

3.6 Environment 3D Model

Pseudo code

Environment 3D Model (P)

Input: a list of all the point clouds P

Output: reconstructed 3D model M

1. **for** each point cloud p ϵ P **do**
2. scale the point cloud P
3. down sample P
4. reduce the noise of P with threshold value
5. **end loop**
6. define a reference point cloud P_{REF} from P
7. **for** each point cloud p ϵ P **do**
8. apply geometric transformation on P based on P_{REF}
9. **end loop**
10. P_1 = stitch all the point clouds P
11. apply Delaunay triangulation on P_1
12. M = surface reconstruction over P_1
13. display M
14. **return** M

Environment modeling is almost identical to object modeling algorithm. However, it does not require any isolation of the object as it deals with the whole environment. The function takes all the point clouds of the environment. At first, it calibrates the point cloud data. It follows the general process of geometric transformation and stitching. After Delaunay triangulation and surface reconstruction is performed, the output is displayed in a 3D model.

4. RESULT AND ANALYSIS

In this part we will discuss the result of the project.

4.1 Object 3D Model

4.1.1 Sample: Bucket

4.1.1.1 Input:

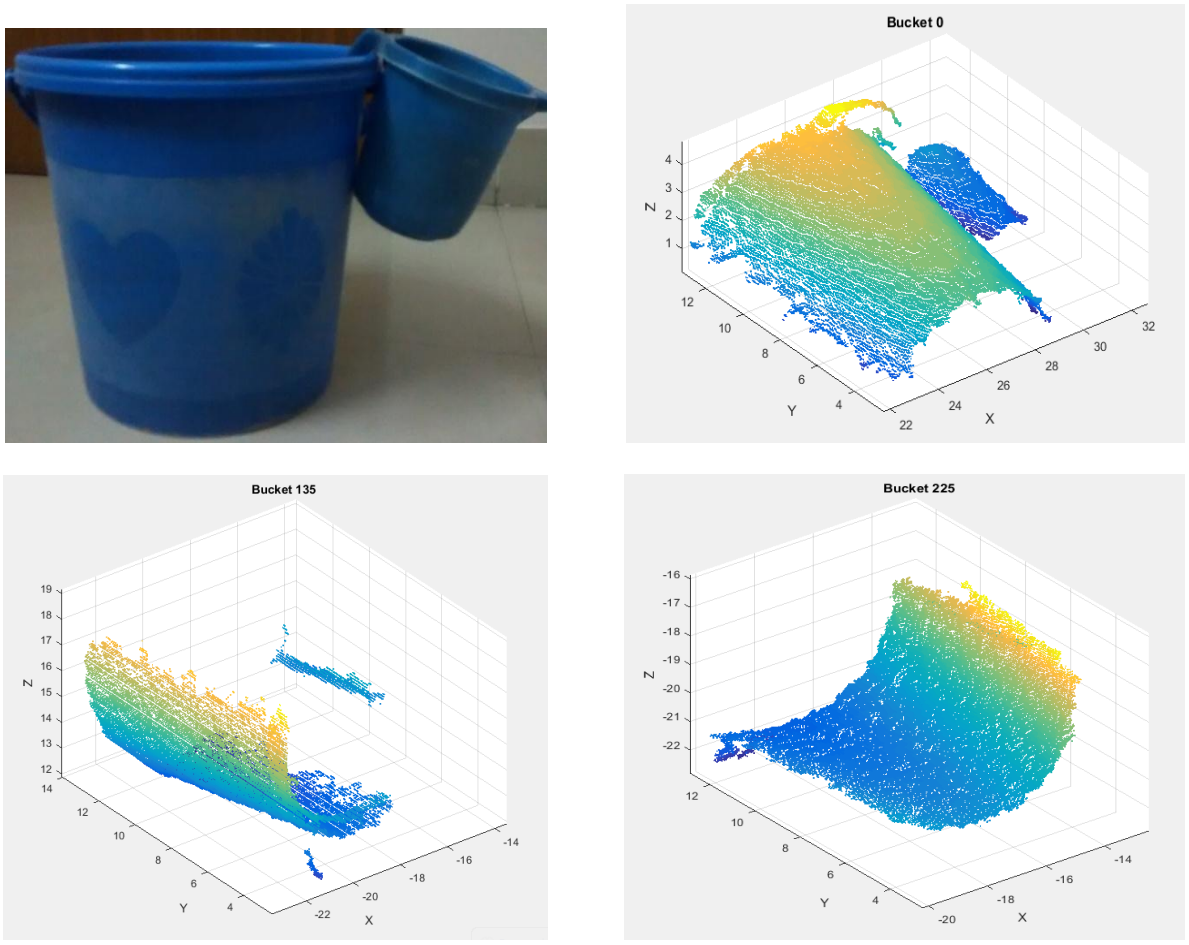


Figure 11: Real Image and Raw point cloud sample from 0 degree, 135 degree and 225 degree angle

In figure 11, the upper left image represents the real photo of the bucket. We have taken depth data of the same bucket from 3 equilateral angles. With a defined reference angle among them, we have performed geometric rotation over y axis on the other two point cloud in 135 and 225 degree respectively. Rest of images shows their individual positions.

4.1.1.2 Process:

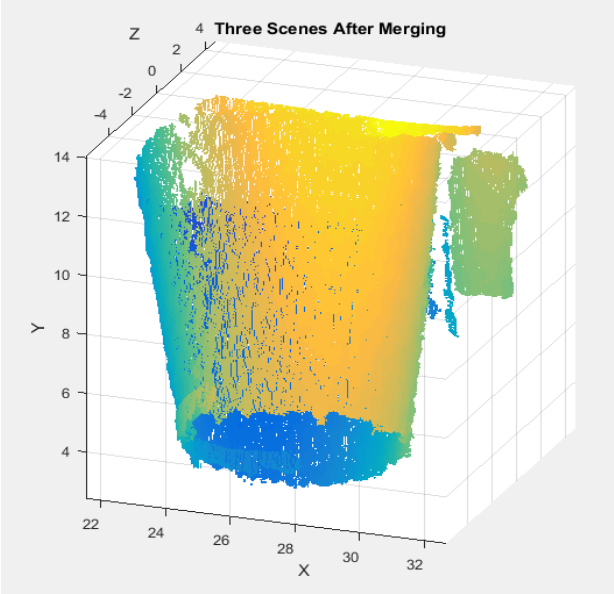


Figure 12: Sample after Stitching and Registration

We have stitched and registered the three point clouds all together. In figure 12, the image displays the dimensions of the object in a single point cloud.

4.1.1.3 Output:

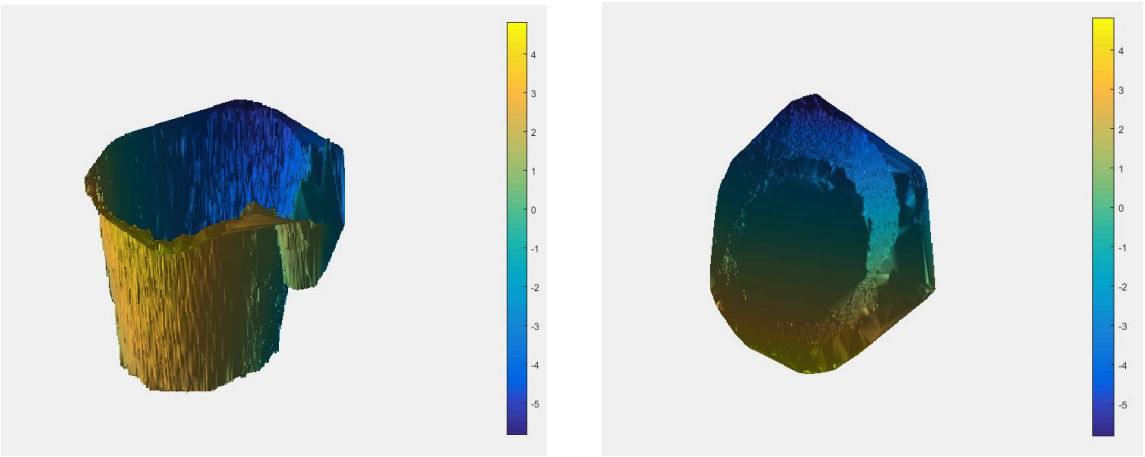


Figure 13: Sample after Surface Reconstruction

From the acquired point cloud, we have performed triangulation and surface reconstruction to gain the 3D model of the object. In figure 13, we have the side and top view of the surface reconstructed object model respectively.

4.1.2 Sample: Chair

4.1.2.1 Input

Figure 14: Real Image and Raw point cloud sample

In figure 14, the left image represents the real photo of a chair. We have taken a single scan for the depth data from the front angle. Right image is the point cloud view of the object.

4.1.2.2 Process

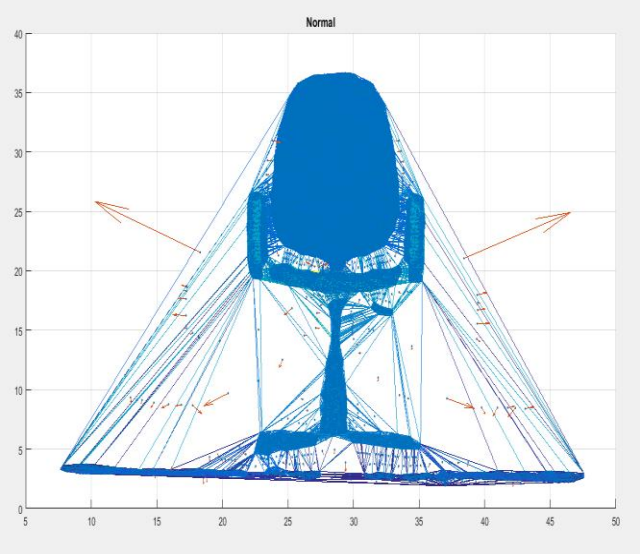


Figure 15: Surface Normal

We have isolated the object from its environment. Afterwards, we have performed the triangulation over the point cloud and determined the surface normal of the each triangle. Based on the normal, we have removed the triangles which are not in the front face, to get a refined set of triangles. In figure 15, the triangulated object along with the surface normal vectors of each triangle is displayed.

4.1.2.3 Output

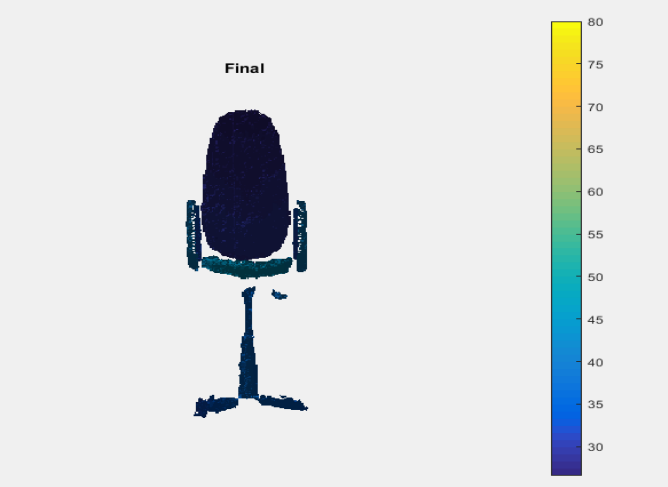


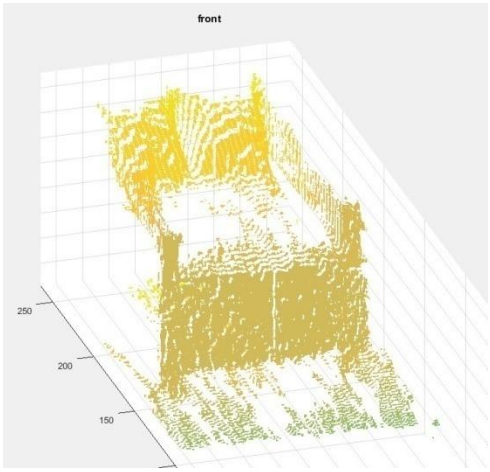
Figure 16: Sample after Surface Reconstruction

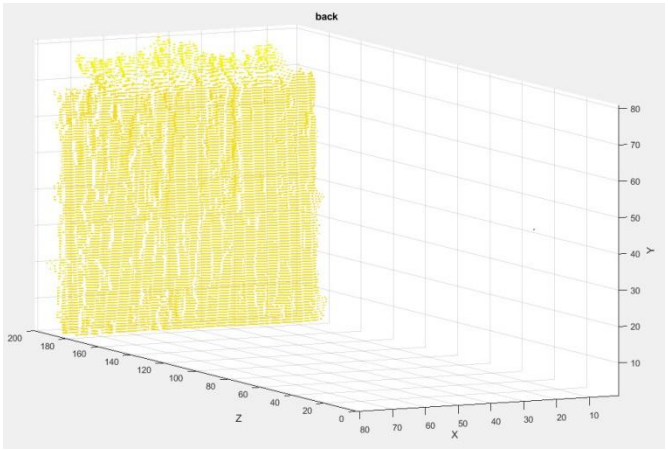
After acquiring an optimal set of triangles of the object, we have performed surface reconstruction over the triangulated object. The filled surface with phong lighting and interpolate shading

4.1.3 Sample: Sofa

4.1.3.1 Input

Figure 17: Real Image





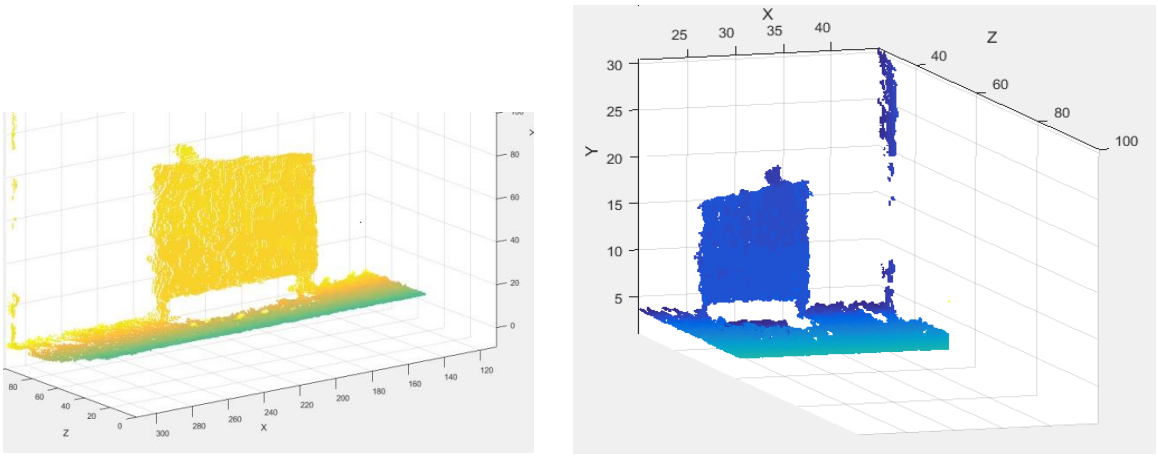


Figure 18: Raw point cloud sample

Figure 17 is the real image of a sofa set. We have taken four scans of the object to get its depth data. Figure 18 represents the front, back, left and right side of the object respectively. Defining front side as the reference, we have performed geometric rotation over the back, left and right side of the object.

4.1.3.2 Process

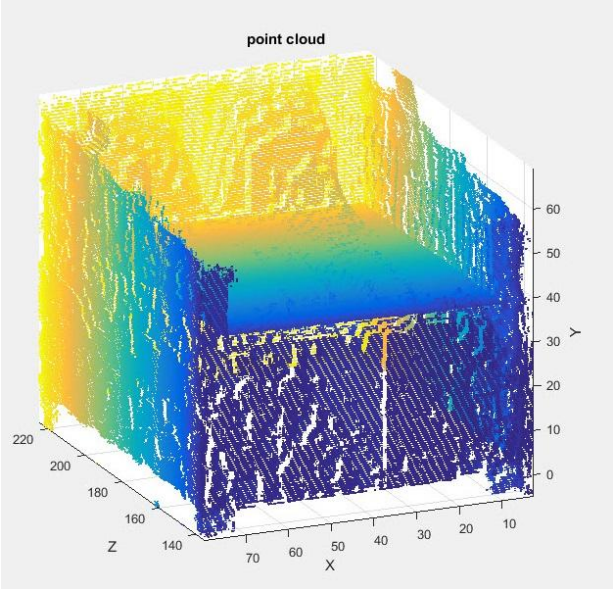


Figure 19: Sample after Stitching and Registration

We have stitched and registered the four point clouds and merged them in a single coordinate system. Figure 19 represents the registered point cloud of the object.

4.1.3.3 Output

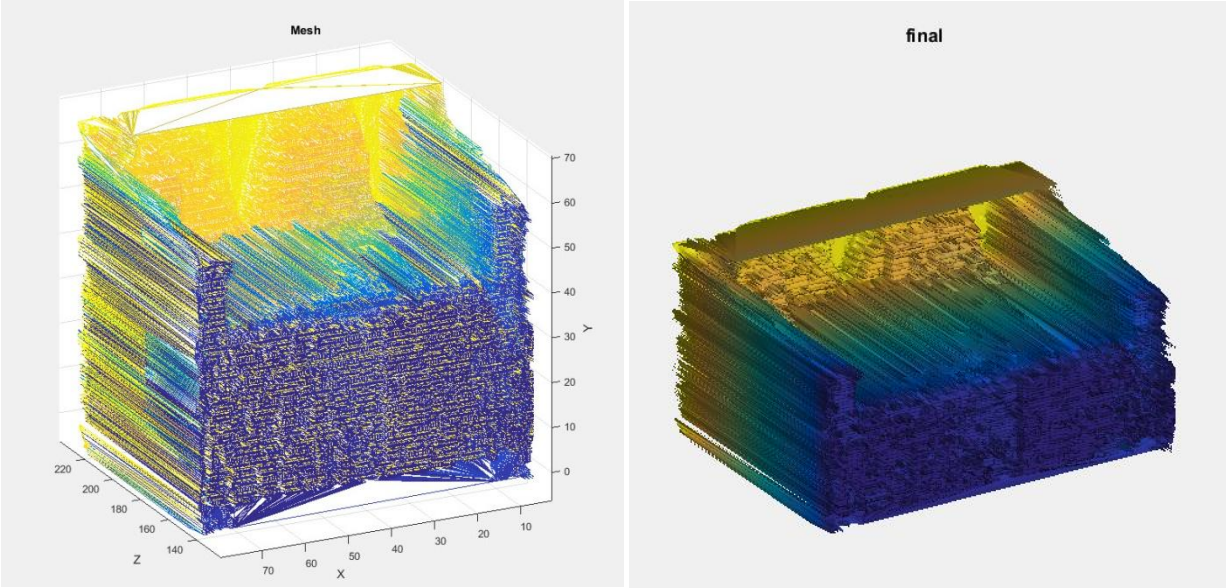


Figure 20: Sample after Surface Reconstruction

We have performed the triangulation over the merged point cloud. Later on, the output has been passed through surface reconstruction method to achieve the 3D model view of the object. Figure 20 represents the sample model in mesh and after surface reconstruction.

Our proposed object 3D modeling process has provided results which satisfies our objectives for generating a fairly accurate structure of any object, given enough depth information. However, the inseparable noise affiliated with the input data slightly reduces the success rate of our algorithm. Then again, proper application of the noise reduction techniques may increase the performance of the algorithm and result in highly efficient 3D object models.

4.2 Environment 3D Model

4.2.1 Sample: Lab Room

4.2.1.1 Input

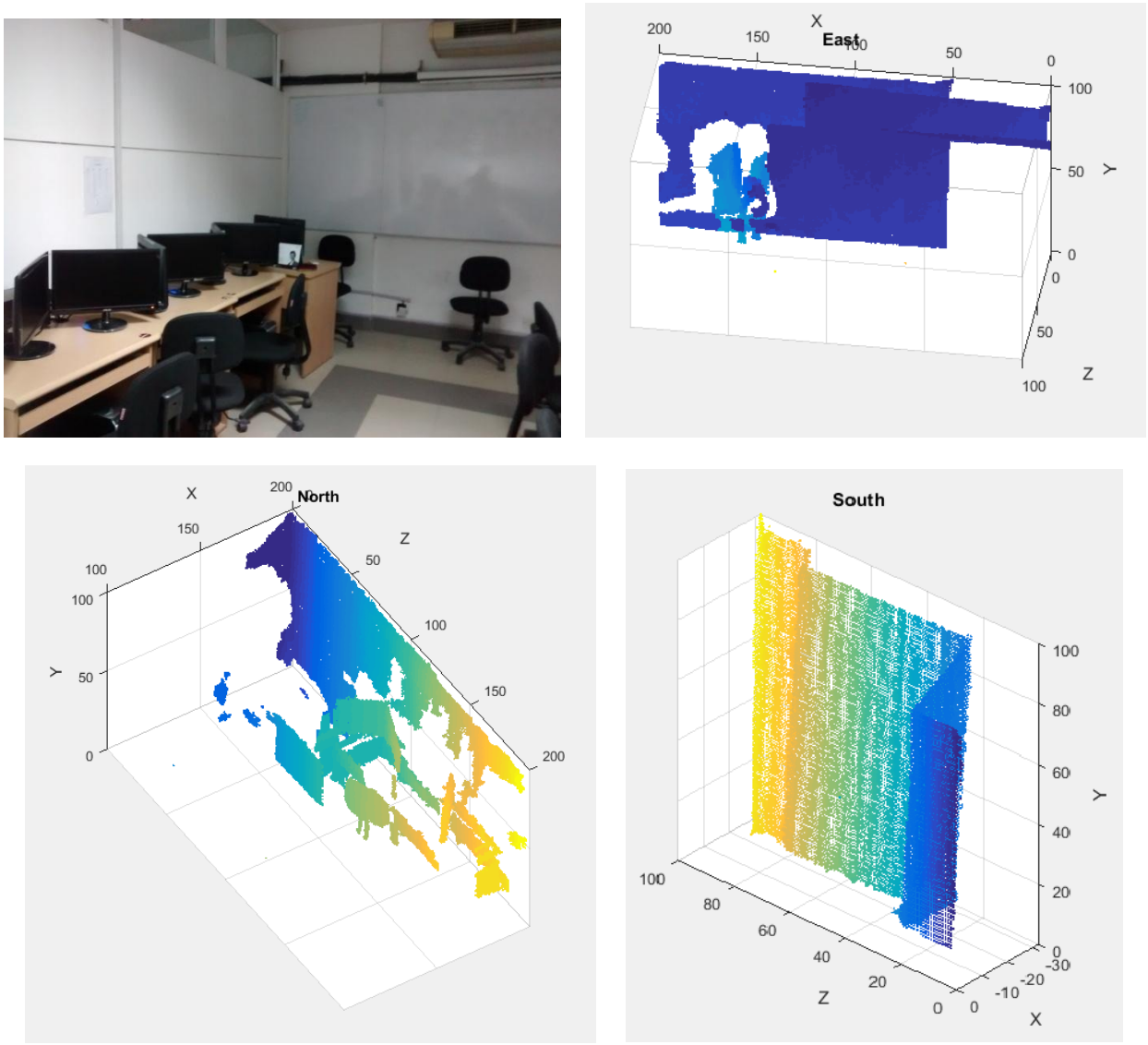


Figure 21: Raw Environment Input

In figure 21, we have the original image and three sections of the graphics lab room of our university in point cloud view. Considering the east side as reference, we have performed geometric transformation over north and south view of the room.

4.2.1.2 Output

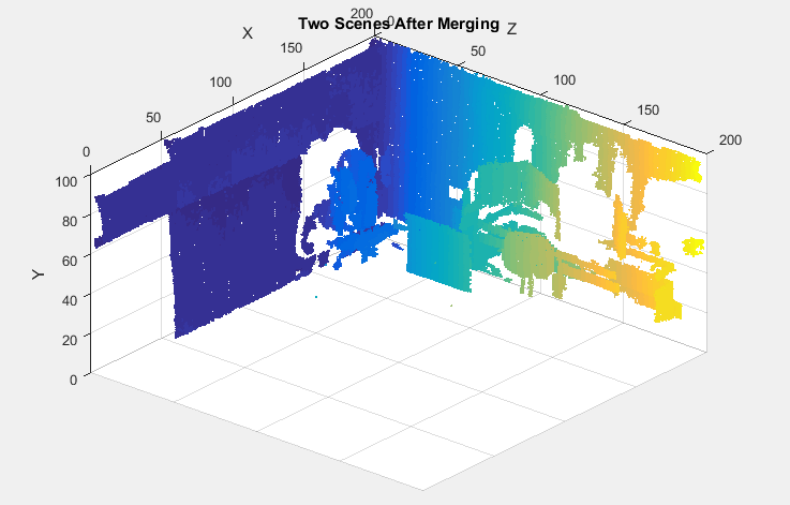


Figure 22: Two Merged Environment

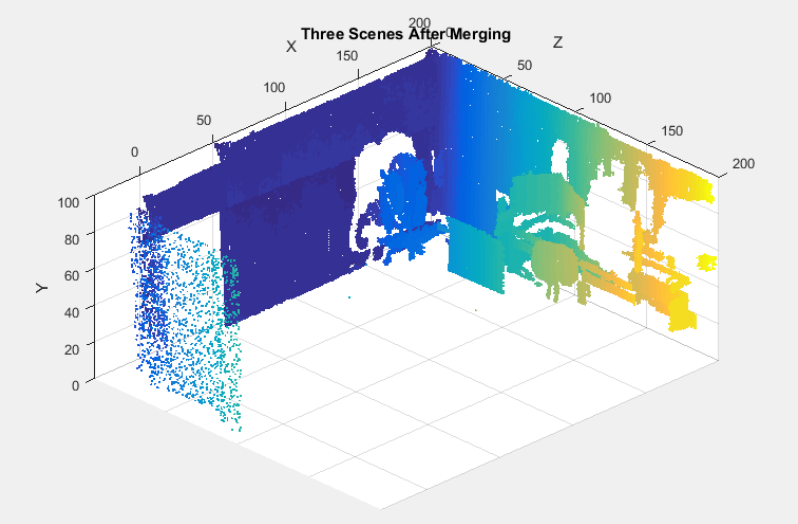


Figure 23: Three Merged Environment

In figure 22 and 23, we can observe the merged scene of the previously mentioned scenes. We can identify the physical shapes of the chairs, tables and the computer monitors on the point cloud view. In this experiment, we have successfully implemented the stitching and registration technique over the point clouds of the environment. Figure 22 and 23 represents the merged output of two and three scenes respectively.

4.2.2 Sample: Room 1

4.2.2.1 Input

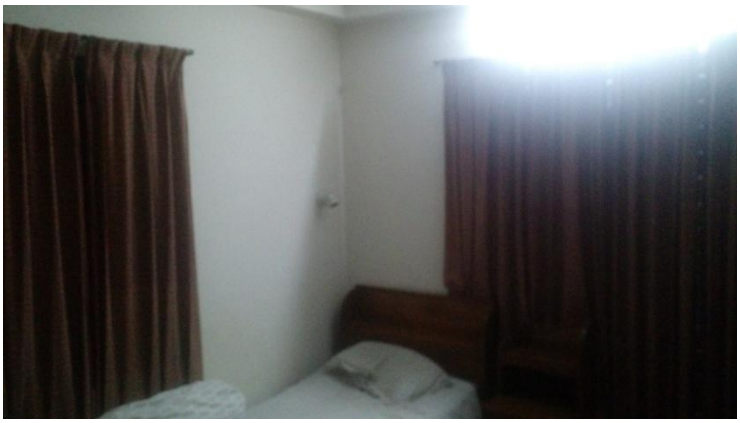


Figure 24: Original Environment

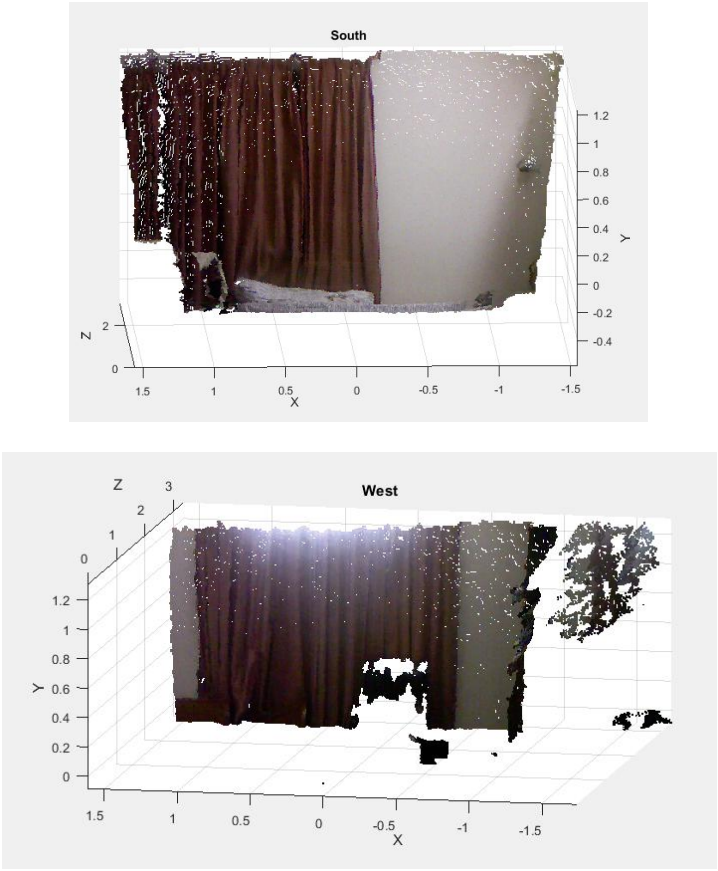


Figure 25: Point Cloud Scenes

We have taken two sides of a Room 1 for our experiment. Figure 24 and 25 represent the actual image of the room and the individual point cloud views of the taken sides.

4.2.2.2 Process

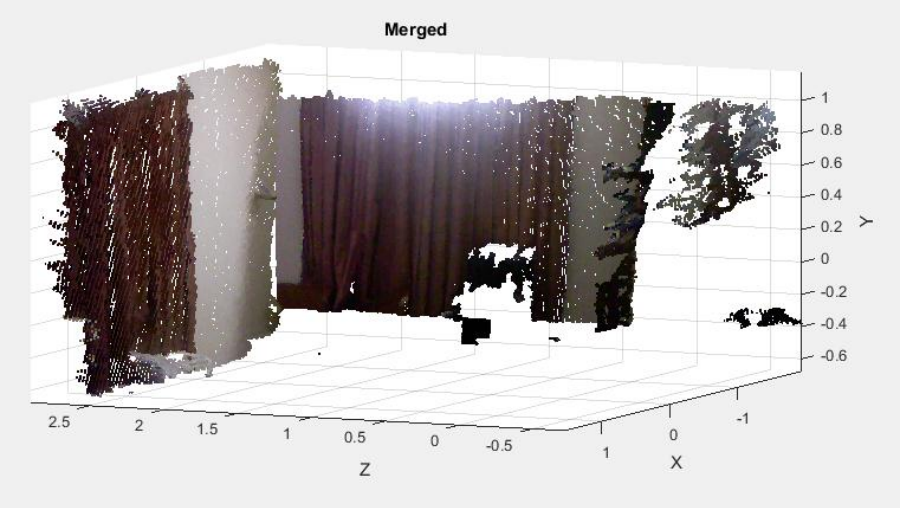


Figure 26: Merged Environment

In figure 26, we have projected the point cloud merge of the two different sides of the room in a single coordinate system. The merged environment holds the original texture information as it has been acquired.

4.2.2.3 Output

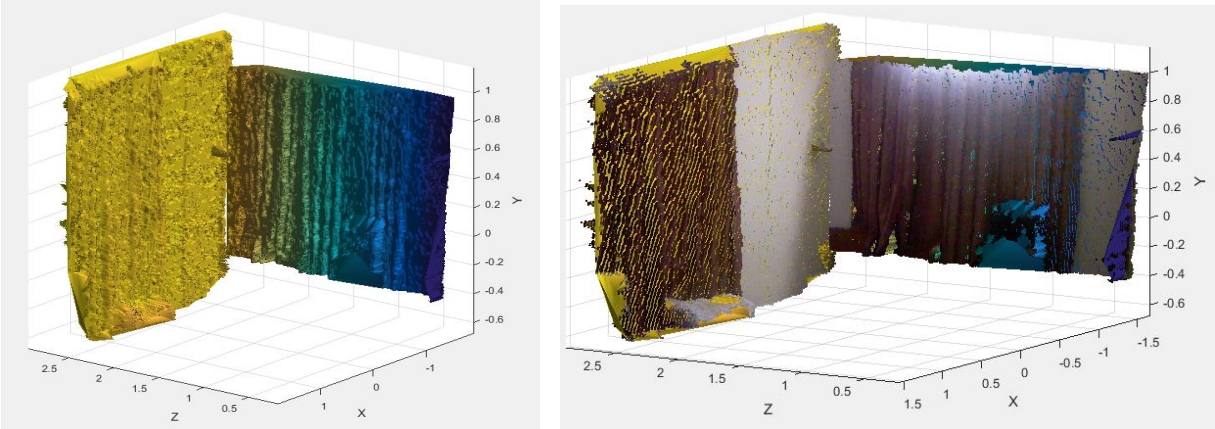


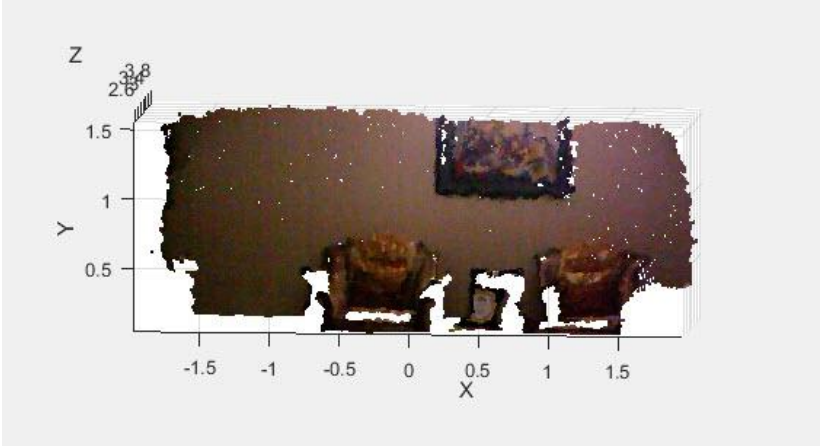
Figure 27: Reconstructed Surface Model without and with Color Texture

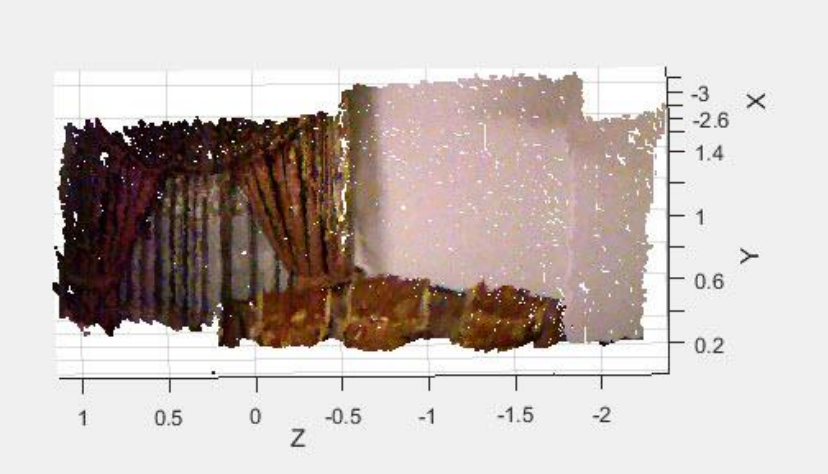
In this experiment, we have been able to reconstruct the surface of the environment after triangulating the merged point cloud. We projected the 3D model of the targeted scene without and with color as they are presented in figure 27.

4.2.3 Sample: Room 2

4.2.3.1 Input

Figure 28: Original Environment





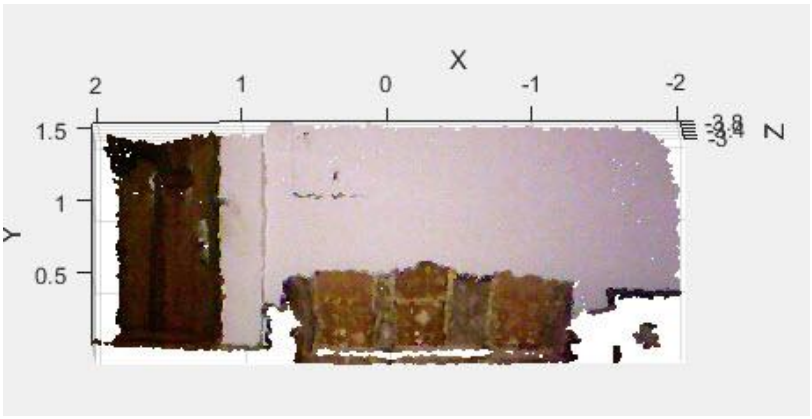


Figure 29: Point Cloud Scenes

We have performed another experiment in a Room 2 with three scenes. Figure 28 is the panorama view of the room and figure 29 represents individual point cloud views of west, south and east side.

4.2.3.2 Process

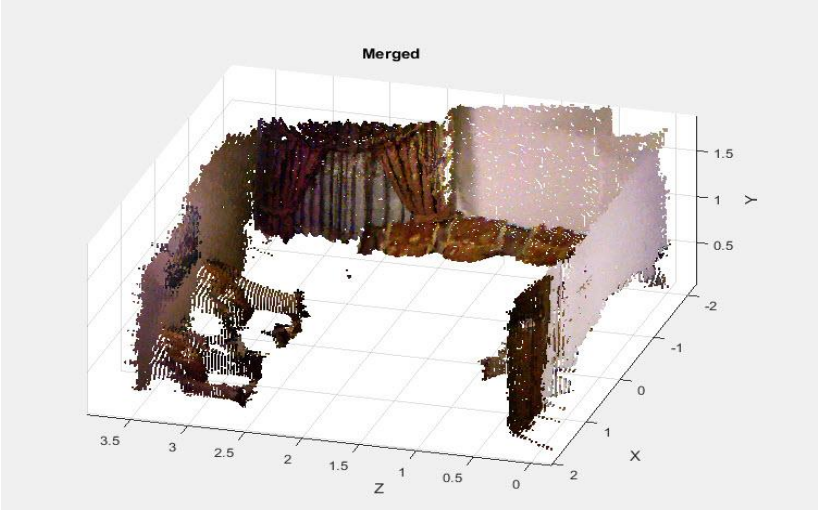


Figure 30: Merged Environment

We have performed the stitch and registration operation over our targeted point clouds. We have merged them together and projected in a single one as displayed in figure 30.

4.2.3.3 Output

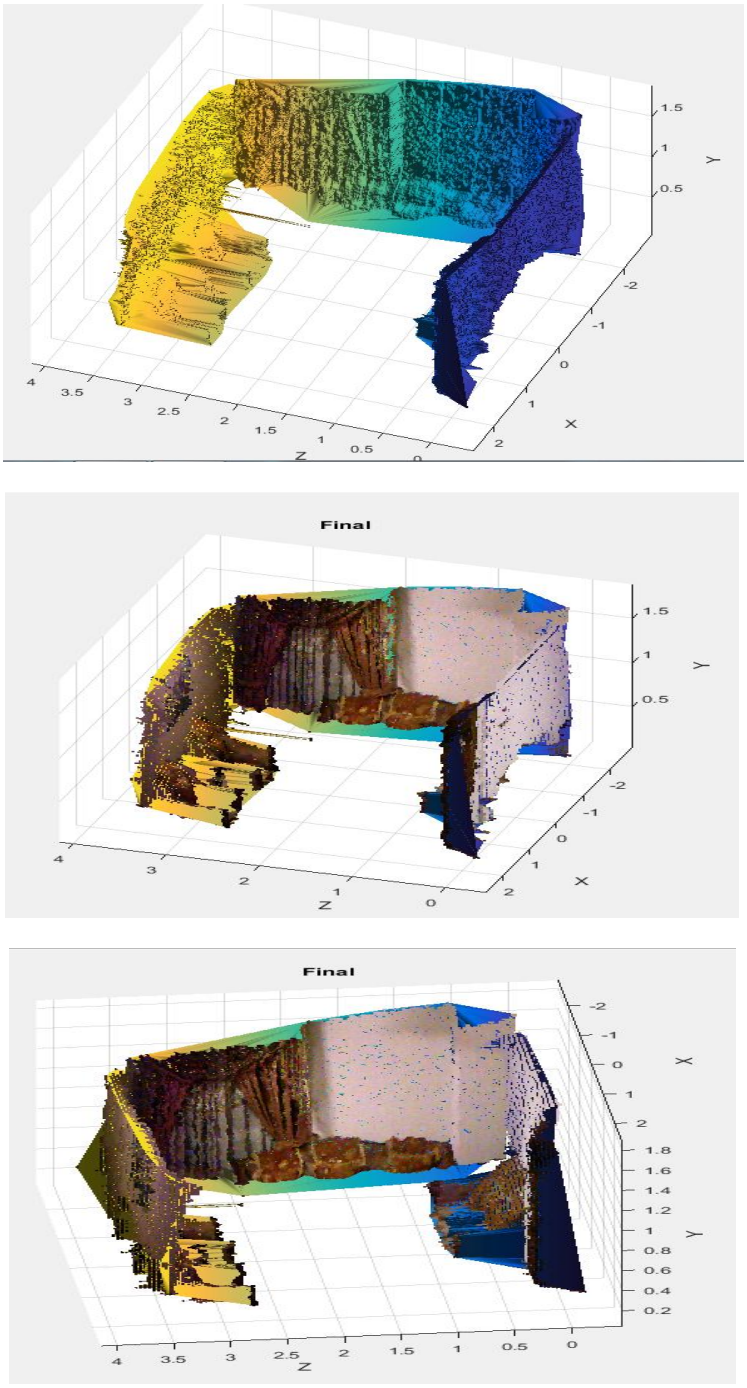


Figure 31: Reconstructed Surface Model without and with Color Texture

In figure 31, we have displayed the output results of our experiment as we triangulated and reconstructed the surface of the environment. The images represent the model without and with original color texture.

5. DATA ANALYSIS

We have performed our testing in three different objects with different number of scans. First, we have used a chair which is taken by only a single depth data scan. Then we have performed the test on a bucket with three scans from different angles. Finally, a sofa has been scanned from four side of it to conduct our experiment.

Table 2: Performance Analysis of Object Model

Object	No of Scan	Dimension (L x W x H) cm^3	No of Raw Points	No of Sampled Point	No of Triangle	Runtime (sec)
Chair	1	48.77 x 48.77 x 91.44	282058	49955	99876	9.964
Bucket	3	36.58 x 30.48 x 36.58	560316	110555	36387	4.711
Sofa	4	76.34 x 76.34 x 64.26	717077	145888	126918	8.229

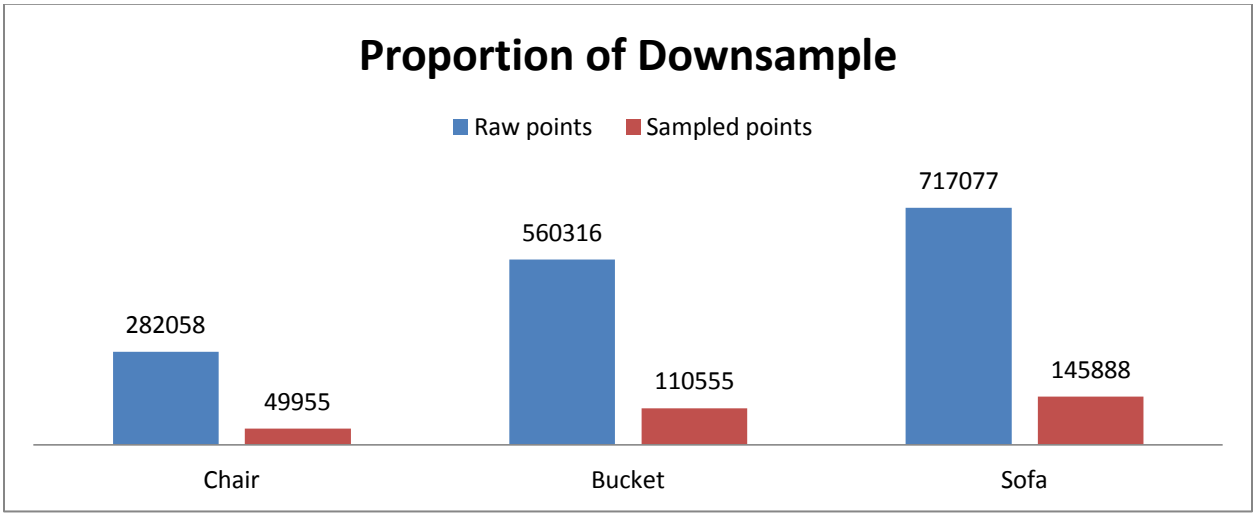


Figure 32: Proportion of Down Sample Runtime of Object Model

At first we have presented a comparison of these three objects based on their actual number of points during raw scanning and number of sampled points after performing down sample process. For chair, the number of points is reduced count goes from 282058 to 49955. The down sample process successfully compact the bucket information with only 11055 points where the number of raw points was 560316. Again, for the sofa the number of points has been down

sampled from 717077 to 145888. For all three objects, the sampled number of points is 18%-20% of the original scan points.

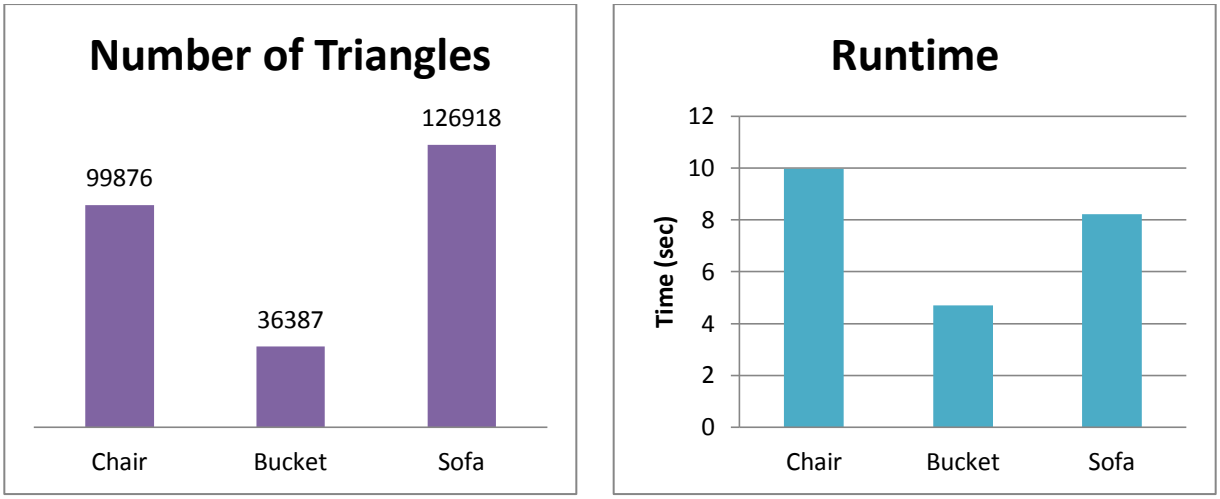


Figure 33: Number of Triangle and Runtime of Object Model

In figure 27, we have displayed the number of triangles generated from the object and the runtime of execution. For chair, it created 99876 triangles and took approximately 10 seconds. The bucket stated the minimum for both triangles (36387) and runtime (4.7 seconds). Finally, 126918 triangles were needed to model the sofa over 8 seconds.

Again, we have experimented in two different rooms with different number of scans and analyzed the performance of our algorithm. For Room 1 we have chosen two sections of the environment. Afterwards, we have taken three sections for Room 2 and performed the same modeling technique. We analyzed the down sample, triangle and the required runtime.

Table 3: Performance Analysis of Environment Model

Environment	No of Scan	Dimension (L x W x H) m^3	No of Raw Points	No of Sampled Point	No of Triangle	Runtime (sec)
Room 1	2	1.4 x 2.2 x 1.7	300458	280479	436419	20.82
Room 2	3	2.8 x 3.0 x 1.4	640341	439266	878407	29.81

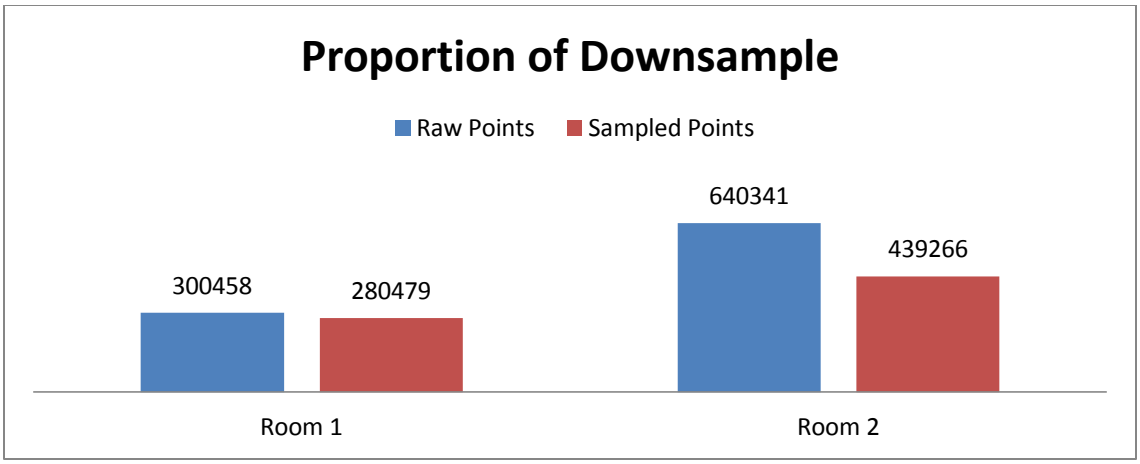


Figure 34: Proportion of Down Sample of Environment Model

As environment requires dense depth data for better modeling, we have chosen to keep the original information as much as possible. Yet, to reduce over head and duplicate points, we down sampled the data a bit. Room 1 initially gave 300458 points in its scanned data which have been compacted in 280479, reducing around 7%. On the other hand, we managed to down sample the number of points by 31.5%, from 640341 to 439266.

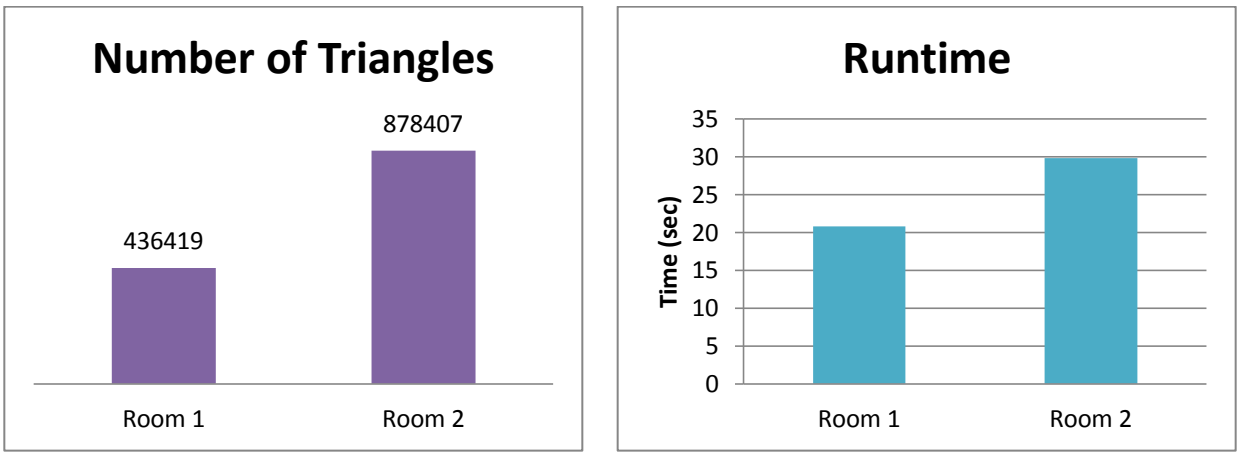


Figure 35: Number of Triangle and Runtime of Environment Model

Room 1, which is measured around 3.08 square meters, has taken 436419 triangles to construct the model in approximately 21 seconds. Compare to that, Room 2, covering 8.4 square meters, has generated 878407 triangles to produce the output result and taken near about 30 seconds.

6. CONCLUSION AND FUTURE PROSPECTS

Here in this paper we have proposed a complete system that will scan the environment through Kinect optical scanning sensor and reconstruct the surface from the scanned data with proper geometric property. However, we have not been able to implement some features very efficiently due to limited amount of time i.e. generating the 3D surface in real time, developing parallel algorithm etc. In this area there is a scope for improvement in the proposed system.

6.1 Triangulation over three dimensional space

We have used two dimensional triangulation processes for triangulating points in this research. We can improve this algorithm to iterate directly over three dimensional spaces and considering a circumscribed sphere for checking the Delaunay validity of triangles. This approach can make our triangulation process more robust and the noise amount of the generated mesh will be much lesser.

In our proposed structure, we have triangulated all the points of the depth image. However if we could consider the flat zone of the surface as combination of minimum triangles and took much more amount of triangles in the edges, we could increase our throughput and reduce the computation time. The product of reconstructed surface would have been much better outlook as well.

6.2 Data Acquisition Technique

The technique used to acquire point cloud data in our research has caused a lot of noise and errors. A circular trajectory of the sensor is needed and applies loop closure to perform a global optimization. Moreover, we can provide a dense surface prediction by utilizing raycast method against live depth map aligned.

In the environment modeling part, we have manually selected the reference point cloud for applying geometric translation over the rest. There may be a scope for automating this whole process so that we do not need to manually specify any reference point cloud.

6.3 Parallel Processing

Currently, research on simultaneous localization and mapping (SLAM) has focused more on real time tracking and reconstruction. Parallelizing the process is the essential ideal for real-time reconstruction. Taking advantage of high GPU processing hardware we can easily parallelize the algorithm used in our system such as registration and reconstruction part.

In recent times a camera named RealSense R200 introduced by Intel enables the user to generate a three dimensional model in real time using libraries created in their SDK. This feature relates to the purpose our research but the camera itself is new in this research field and a lot of ground has to be covered like system integration, understanding the functions to extract point cloud, etc before its actual implementation.

With the arrival of Microsoft Kinect sensor and its extensive available resources will open up many new possibilities for augmented reality, human –computer-interaction and other field. In this research, we have presented a workflow to reconstruct 3D object and environment. There are three steps in our system:

I. Data acquisition from Kinect Sensor and pre-process.
II. Registration of the point-cloud.
III. Integration of the depth data using geometric method, Delaunay Triangulation following 3D model generation.

Our hope is to scale the system further, reconstructing larger scenes with more memory efficient representations and also we hope our system will open many new topics for research both in terms of underlying technology, as well as the interactive possibilities it enables.

REFERENCES

[1] Bethany Jean Juhnke, "Evaluating the Microsoft Kinect compared to the mouse as an effective interaction device for medical imaging manipulations," 2013.

[2] Regius Asiimwe and Amir Anvar, "Automation of the Maritime UAV Command,Control, Navigation Operations, Simulated in Real-Time Using Kinect Sensor: A Feasibility Study," *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, vol. 6, no. 12, 2011.

[3] Richard A. Newcombe et al., "KinectFusion: Real-Time Dense Surface Mapping and Tracking," , 2011.

[4] Klemens Jahrmann, "3D Reconstruction with the Kinect-Camera," , February 2013.

[5] Song Tiangang, Lyu Zhou, Ding Xingyang, and Wan Yi, "3D Surface Reconstruction Based on Kinect Sensor," *International Journal of Computer Theory and Engineering*, vol. 5, no. 3, June 2013.

[6] Ayrton Oliver, Burkhard C Wunsche, Steven Kang, and Bruce MacDonald, "Using the Kinect as a Navigation Sensor for Mobile Robotics," November 2012.

[7] Kinect for Windows Architecture. [Online]. https://msdn.microsoft.com/en-us/library/jj131023.aspx

[8] 3D Graphics with OpenGL. [Online]. https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

[9] OpenGL Rendering Pipeline. [Online]. http://www.songho.ca/opengl/gl_pipeline.html

[10] Morgan McGuire and Harold S. Stone, "Techniques for multi-resolution image registration in the presence of occlusions," , 1997.

[11] Triangulation. [Online]. http://mathworld.wolfram.com/Triangulation.html

[12] Victor Alvarez and Raimund Seidel, "A Simple Aggregative Algorithm for Counting Triangulations of Planar Point Sets and Related Problems," in *SoCG 13 Proceedings of the twenty-ninth annual symposium on Computational geometry*, New York, 2013, pp. 1-8.

[13] Leif P. Kobbelt and Mario Botsch, "An Interactive Approach to Point Cloud Triangulation," 2000.

[14] Microsoft MSDN. [Online]. https://msdn.microsoft.com/en-us/library/windows/desktop/bb173380%28v=vs.85%29.aspx

[15] DanFeng Lu, HongKai Zhao, Ming Jiang, ShuLin Zhou, and Tie Zhou, "A Surface Reconstruction Method for Highly Noisy Point Clouds," in *Variational, Geometric, and Level Set Methods in Computer Vision*. Beijing, China: Springer Berlin Heidelberg, 2005, pp. 283-294.

[16] Navpreet Kaur Pawar, "Surface Reconstruction from Point Clouds," , 2013, p. 10.

[17] Pierre Alliez, Laurent Saboret, and Gaël Guennebaud. (2015, October) CGAL. [Online]. http://doc.cgal.org/latest/Surface_reconstruction_points_3/

[18] Sheilly Padda, Sonali Gupta, Apoorva Arora, and Priya Sharma, "Different Shading Algorithms for Image Processing," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 5, November 2014.

[19] David Sheiner, Graham Sellers, John Kessenich, and Bill Licea-Kane, *OpenGL Programming Guide*, 8th ed.

[20] David Houcque. (2005, August) Introduction to Matlab for Engineering Students. Document. [Online]. https://www.mathworks.com/matlabcentral/linkexchange/links/2089-introduction-to-matlab-for-engineering-students

[21] 3-D Point Cloud Processing. [Online]. http://www.mathworks.com/help/vision/3-d-point-cloud-processing.html

[22] Piyush Kumar. (2006, November) Triangle++. [Online]. http://compgeom.com/~piyush/scripts/triangle/