Carleton University

Comp 4905 - Honours project

3-D modelling using stereo cameras

Taronish Dastur
101028771

Supervised by: Professor Mark Lanthier, Department of Computer Science

Fall 2019- Winter 2020

Abstract

This project attempts to form a 3-dimensional model of images captured by a stereo camera and integrate it with an existing project titled Cube-shaped Robots with Unsupervised Model Construction. The pair of images captured by the camera are used to form a depth map from which a 3-dimensional model is generated. After which, the 3-D points are transformed and modified to fit the Cube-shaped robots project. Finally, this model is imported to the Cube-shaped robots project to enhance its capabilities. The project was done in Python 3 primarily utilizing the OpenCV and Open3D libraries. The Cube-shaped robots project was written in Processing 2.2.1.

As a result of this project it is possible to generate a 3-dimensional model of any object or scene captured by a stereo camera. The integration with the Cube-shaped robots project allows for many possible uses to create structures where the Cube-shaped robots can take the shape of an object captured by a camera.

Acknowledgments

I would like to thank my supervisor Professor Mark Lanthier for his help and support throughout the development process. I would also like to acknowledge the work of Sehwan Lee in the Cube-shaped Robots project which served as an inspiration to do this project.

Table of Contents

List of Figures

1.0 Introduction

This project aims to create a 3-dimensional model of any view captured by a stereo camera by using a depth map. This model would be integrated with another project titled Cube-shaped Robots with Unsupervised Model Construction by Sehwan Lee which simulates robotic cubes capable of forming shapes.

The project was divided into three parts based on goals to achieve. The first part involved capturing the images using a stereo camera and generating a depth map. From this depth map a 3D point representation (point cloud) was obtained. The second part dealt with modifying the point cloud in order to remove noise and downsize the 3-dimensional points to be read by the Cube-shaped robot project files. The third and final part of the project was to read the point cloud and process its corresponding Cube shapes within the Cube-shaped Robots project.

The following sections provide an introduction into the various concepts utilized within the project as well as a description of the Cube-shaped robot project. It also describes the application of these concepts and libraries and functions used to achieve the final output.

2.0 Previous work and motivation

The implementation of the Cube-shaped robot project opens doors to a lot of possible applications in the field of robotics. The project is a simulation of cube shaped robots which communicate with each other to form a given 3-dimensional shape. Cubes start from the floor level and climb over each other to take a 3-D shape given to them. The cubes on the floor level act as beacons to the incoming cubes. Once a cube is in range of a beacon it is given a position to fill. Once a floor level is complete cubes can then climb over each other to form the second level of the model.

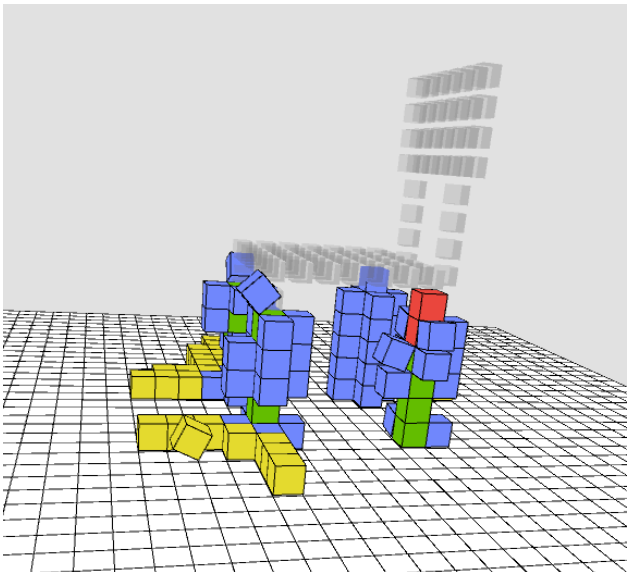


Figure 1: Example of Cube-shaped robot project

Given the right hardware this simulation can be translated into large scale projects such as an architectural tool to help build structures, provide temporary support in the building process etc. It can also be used on a smaller scale to replicate a given model such as a chair or a room. This small-scale application of cube robots was the motivation behind this project. By taking an image of an object or room, a model can be generated and given as an input shape to the cubes. The cubes would then be able to replicate the given objects to be used as a construction tool, toy, mapping tool etc. This would open the doors to many future additions to the project and allow for multiple applications of the simulation.

3.0 Process and concepts

The following sections show the processes involved in the 3-dimensional model generation and all concepts related to it.

3.1 3D model generation

This section talks about the processes involved in the first part of the project i.e. 3-D model generation. This includes capturing the images, finding a depth map and 3-dimensional representation of the images.

3.1.1 Camera calibration

Camera calibration is used to find values about the camera's internal and external properties. Internal parameters of a camera are values which define the internal geometric and optical characteristics of a camera. This includes the image center of the camera screen, focal length, size of the pixels etc. External parameters define the camera's geometrical orientation in space (the world coordinate system). External parameters are defined by the rotation and transformation vertices.

The OpenCV library's calibration process is used for calibrating the camera. The calibration functions use Zhang's calibration method. Zhang's calibration method uses a planar lattice (such as a chessboard) to locate lattices. A chessboard makes a good planar lattice template as the black and white surface as well as straight lines make it easier to detect the corners and intersections. Images of this template are captured by the camera at various positions and orientations. The internal and external camera parameters are determined by direct linear transformation method through points on the template and its corresponding image points [Y. M. Wang, Y. Li and J. B. Zheng, 2010].

Since calibration processes are time consuming and the camera is kept constant, the outputs of the calibration process is saved in a separate file which can be accessed throughout the project without the need to repeat it.

For a stereo setup the calibration images were captured by both, the left and right cameras separately at their fixed positions from each other. Even if the camera's used are the same it is good to calibrate them separately to account for minor individual changes in the camera parameters. After the calibration values are obtained, they can be checked by reprojection of the 3-D images points into the 2-D image points using the intrinsic and extrinsic camera parameters obtained through calibration. The differences in the points obtained by the projection formula are compared to the points from the original image. If this difference is minimal, we can conclude that the calibration process was successful, and values obtained are accurate.

3.1.2 Preparing images for depth mapping

Distortion is caused by the lens not being perfectly straight. This causes the light entering the lens to bend before falling on the light sensor. So, the images captured by a camera with such a lens are slightly different from the original scene being captured. Any camera is prone to distortion regardless of how expensive or professional it is.

There are two types of distortion, radial and tangential. Radial distortion causes straight lines to appear curved. Tangential distortion is caused by the lens not being parallel to the image, this causes certain areas of the image to look closer than the actual object. There are two types of radial distortions, barrel and pincushion. Barrel distortion produces an image which gives a "fisheye" effect, i.e. A larger field of view towards the corners. Pincushion distortion on the other hand gives the effect of a zoom lens where the objects to the center appear closer together [Nasim Mansurov, 2013].

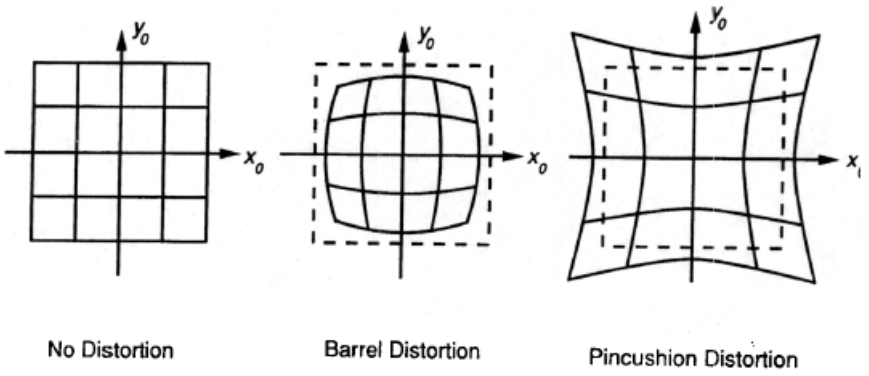


Figure 2: Two types of radial distortion

The intrinsic and extrinsic camera parameters obtained from the calibration process are used to remove distortion in the image. The camera matrix (K) is obtained from the internal parameters and distortion coefficient (dist) is obtained from the external parameters. The OpenCV library provides a function to undistort which uses the following formula.

By knowing the intrinsic camera matrix (K) and the distortion coefficient (dist) from the calibration process, we can remove these distortions. The OpenCV function undistort, uses the K and dist values to return an image modified to remove the distortions.
x'=x+x—(k1r2+k2r4+k3r6+...)+[p1(r2+2x—2)+2p2x—y—](1+p3r2+...) [Pierre Drap and Julien Lefèvre, 2016]

$$y' = y + \underbrace{\bar{y}\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \dots\right)}_{\text{Radial Distorsion}} + \underbrace{\left[p_2\left(r^2 + 2\bar{y}^2\right) + 2p_1 \bar{x}\bar{y}\right]\left(1 + p_3 r^2 + \dots\right)}_{\text{Tangential Distorsion}}$$

Figure 3: Formulas for distortion in a camera lens

3.1.3 Depth map

A depth map is a representation of the distance of each point in an image from the camera. There are many tools available to find the depth map of an image captured using technology such as LIDAR, infrared, cameras etc. This project uses a stereo camera setup to take pictures and the pair of images obtained are used to find a depth map. A stereo camera involves two cameras of the same type placed on the same axis in order to capture two images of the same scene. The differences of the location of the same point between the two images is then translated into distance. This process is called Stereo Rectification.

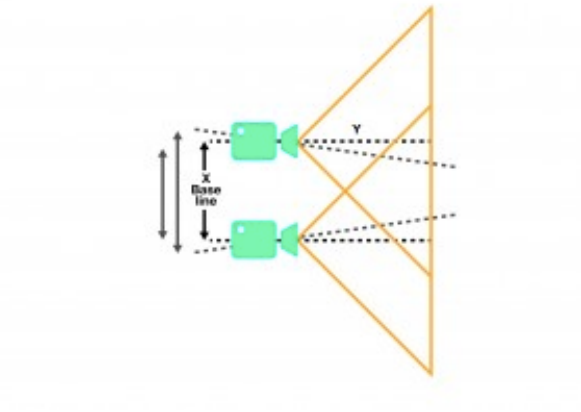


Figure 4: Stereo camera setup

Stereo cameras are designed in the same way as the human eye. Each eye (or camera) receives a slightly different view of the scene they are looking at, known as disparity. This can be seen when you look at an object with one eye and then the second eye. The scene shifts slightly left or right depending on the eye. Objects closer to the eye seem to shift more than objects further

away from the eye. The same principle is used to find the distance of different points in an image from the camera. The greater the horizontal transformation of a point between the left and right images captured by a stereo camera, the closer the object is.

As seen in figure 4, each camera has a field of view which it can capture. Object out of this field of view cannot be used in the rectification process. Only the parts which overlap between the two cameras are used in the mapping process. Sometimes objects are seen by one camera are hidden by other objects or light in front of it in the second camera. This is known as occlusions. Stereo camera processes cannot account for occlusions and this has to be taken care of while capturing the images. The distance between the two cameras is known as baseline. Modifying the baseline changes joint field of view between the two cameras. When the cameras are kept further apart, i.e. the baseline is large, the regions of overlap are less but the depth mapping is more accurate because corresponding points between the two cameras move by a larger distance. Contrarily when the cameras are kept closer together the overlap is larger, but the depth mapping is less accurate. Building a good stereo camera requires choosing an optimal baseline to ensure best depth mapping while getting the largest overlaps in field of view.

Block matching algorithms are used to find depth using similarities in the two images. Each pixel in the left image is bound by a small window. Then, each window in the right image on the same horizontal axis is taken and a sum of square differences is calculated between the of the window of the left and right image. The window with the least difference is considered to be the matching window between the right and left camera. We can then calculate the shift between the two windows and find the disparity between the two images which can then be used to find depth.

3.1.4 3D mapping

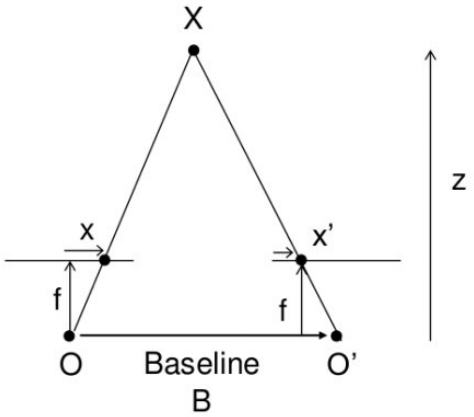


Figure 5: Mathematical representation of a stereo camera setup

Disparity is mathematically defined as $x - x' = BfZ$, where B is the baseline, f is the focal length of the camera and Z is the distance of the camera from the object point. The disparity value is obtained from the depth map. By rearranging the formula, we can find the value of Z

such that Z = fB/d. The values of the X and Y positions can be obtained using the formula X = xZ/f and Y = yZ/f. These X and Y values are obtained from the Pinhole camera model as shown in figure 6. Here, The X,Y,Z coordinates are the 3-dimensional world coordinates and x, y are the 2-dimensional image coordinates formed on the camera's sensor.

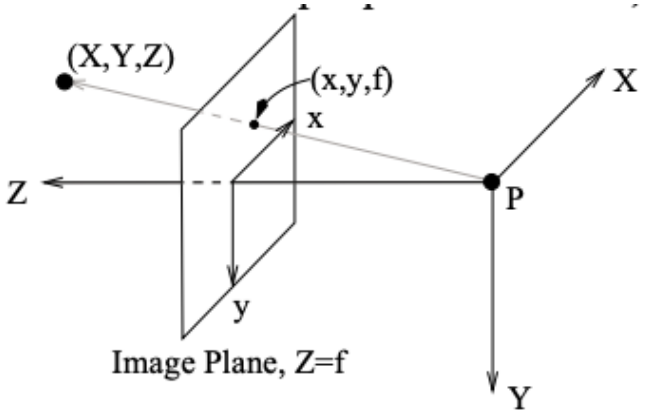


Figure 6: Pinhole camera model

3.2 Preparing the point cloud

Since this project involves integration with another project known as Cube-shaped Robot, the second part of this project is a gateway between my project and the Cube-shaped robot project. It modifies and shapes the 3-dimensional points, so they are ready to be read and translated into cube format. The points are in the form of a point cloud file which contains the x, y, z coordinates of the point as well as its r, g, b colour values.

The 3-dimensional point cloud obtained from the depth map of stereo images has millions of points, so it is first downsized by reducing it by a certain factor depending on the image. For most images the floor or walls are also translated into the point cloud. These points are found and removed as they are not required. The three axes are brought to the 0 which brings the point cloud to the ground level and centers the model for the cubes to form the shapes in part 3. Noise is removed by an outlier removal process which checks points in a neighbourhood and removed points which don't lie in any neighbourhood. Finally, the points are modified to remove gaps between the points and any points which are not connected to anything.

3.3 Integration with the Cube-shaped Robot project

For the final part the 3-dimensional point cloud is imported into the Cube-shaped Robot project. The existing code was modified to be able to read a point cloud file and store the x, y and z locations. Each point corresponds to a cube location and the cubes at the floor level act as starting points. The code was also modified to handle the larger shapes by releasing new cubes at a time interval so that the incoming cubes do not crowd the floor space.

4.0 Methodology

The following section describes in detail how the concepts described in the introduction were implemented for the project as well as the libraries used to achieve it. The project was done using Python 3. The Cube-shaped robot project was coded in a java-based programming language best suitable for 3-dimensional image modelling called Processing. Primarily two libraries were used; OpenCV and Open3D. OpenCV provides functions for image handling and computer vision. Open3D is an open source python-based library for 3-dimensional data.

4.1 Obtaining images

The images were obtained using a stereo camera. Since the stereo camera requires capturing two images of the same scene from two different positions at first, I attempted to try and use only one camera positioned to imitate the left and right fields of view. For this I created a tripod setup such that the camera could be translated horizontally while keeping the center of their fields of view parallel (fig 7). To calibrate the camera, an image of a chessboard was placed on the wall and the tripod was moved to capture it from various angles while ensuring the chessboard was completely visible from both the left and the right camera positions. After obtaining more than 10 image pairs, the calibration values are obtained for the left and the right camera. After calibration was complete, the setup was used to capture pictures of an object.

Figure 7: Tripod setup for capturing stereo images

However, a perfect stereo setup is difficult to obtain due to the calculations of distances and angles required between the two cameras. It is even more difficult to obtain by using one

camera placed in two positions. While this worked to obtain depth maps, the depth maps obtained from this technique were not full enough to be useful. There were many holes in the map, and this would not lead to a good cube representation of the object. So, as an alternative to this I used pairs of stereo images obtained from an online dataset at http://vision.middlebury.edu/stereo/data/scenes2006/.

Figure 8: Pair of stereo images

4.2 Calibration

The OpenCV library provides a function called findChessboardCorners which uses the image of a chessboard to calibrate the camera. As long as you know the size of the chessboard and the image captured shows the chessboard clearly, the function finds points of the corners of the chessboard by breaking apart the squares of the chessboard and putting them back together. [Arturo de la Escalera * and Jose María Armingol]. The findChessboardCorners function returns points where the chessboard corners are found. These are used as image points for the calibration.

Multiple images of the chessboard in various orientations and positions is required to generate as many object and image points as possible. Image points are the 2-D representations of the image and object points are their corresponding 3D representations in space. This is used in the OpenCV function calibrateCamera, which returns the projection error, Camera Matrix, distortion coefficients, rotation vector and translation vector. The projection error has to be as close to 0 as possible to get efficient calibration. Distortion coefficients are vectors which provide information about the distortion which is used to remove radial and tangential distortions. The rotation and translation vectors are vectors of rotation and translation vectors estimated for each view to the camera. The camera matrix or the Intrinsic matrix contains the internal information about the camera. It is a 3x3 matrix (fig 9) which contains information

about the focal length/width of a pixel within the camera (fx), focal length/height of the pixel (fy) and x and y positions of the optical center of the camera (cx, cy).

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 9: 3x3 Camera matrix

4.3 Depth maps

To find the depth map a block-based algorithm is used. The OpenCV object type StereoSGBM has a set of values which are set in order to find a depth map between two stereo images. These values control how block window size, smoothness, noise etc of the depth map. Since these values need to be changed depending on the image, I developed a GUI to control these values and see how they effect the depth map. The values for everything except the window size and minimum number of disparities is set based on previous calculations. The two values are controlled by a trackbar to find the best possible depth map for every image pair.

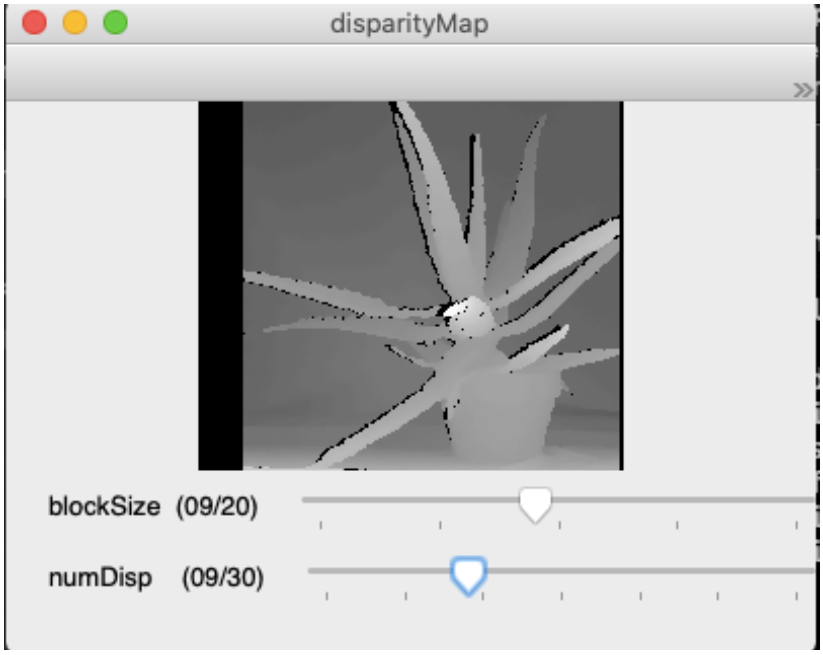


Figure 10: GUI for controlling depth map parameters

After the depth map is obtained from the pair of images the distance of each point from the camera is calculated. The x, y, z coordinates of each point as well as the r, g, b value of each pixel is stored in point cloud file. This file can be read by a 3-D model viewer such as Meshlab.

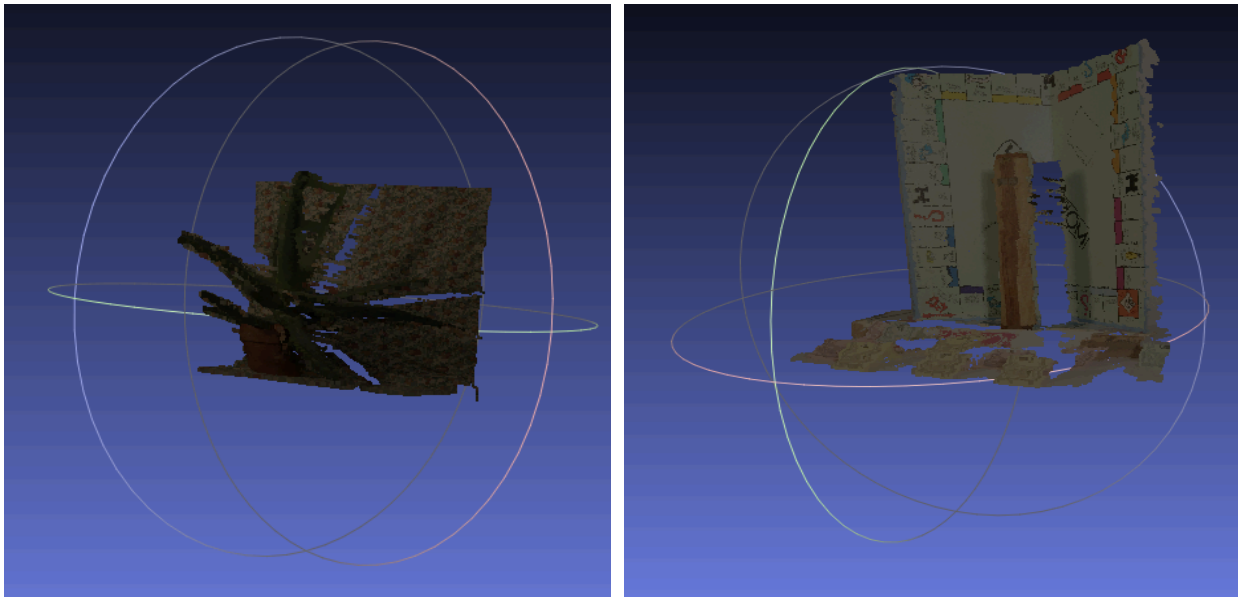


Figure 11: 3D model representation of point clouds in Meshlab

4.4 Modifying the point cloud

The point cloud file obtained is very large in size and cannot be read by the Cube-shaped robot project due to restrictions of the programming language in which it is written. So, in order to successfully merge the point cloud with the Cube project I first started by reducing the size of the point cloud. This was done by reading the x, y, z coordinates of the point cloud and dividing it by a large factor depending on the individual point cloud. This downsized the point cloud without losing its shape. After this I attempted to remove extra points such as the walls and floor which is not required for the cube integration. For this I found a region of maximum points in each axis. This was done by finding the maximum occurring x, y and z value. Then, if any point fell within each of the three values, it was removed from the list of points. For example, if we consider the y axis, the points below the maximum y value are removed in order to remove the floor.

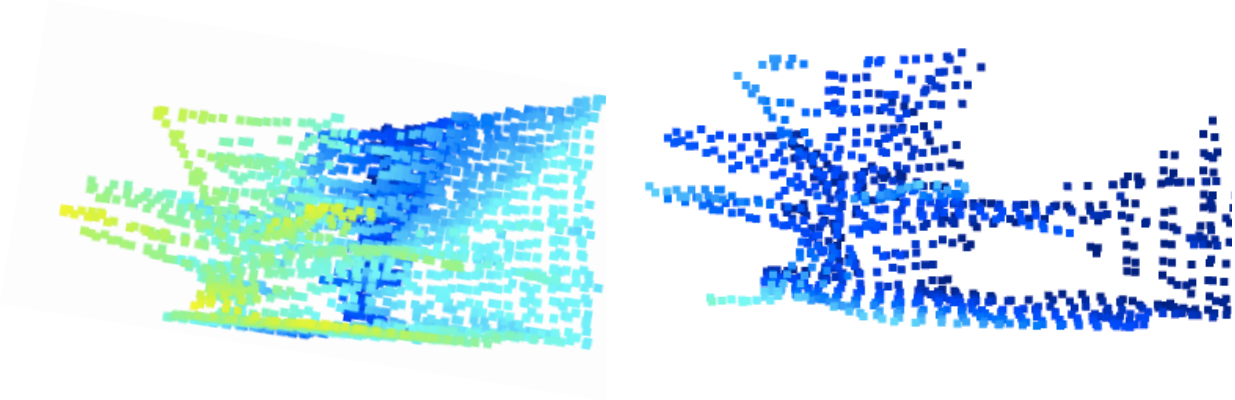


Figure 12: Before and after removing excess points of the wall and floor

The point cloud file in its unmodified form also has a lot of noise. These points are by products of the depth map and do not contribute to the overall object that is to be captured. To remove these points, I have used an outlier removal function provided by the Open3D library. The statistical_outlier_removal function goes through every point and its neighbours and removes points which lie out of the range of points in the neighbouring areas. The function can be modified to check a smaller or larger radius thus making the outlier removal process stronger or weaker.

Points on each axis are also reduced to its smallest value. This is done by finding the minimum value of each axis and subtracting every point in the list of points by that value. This process brings the model to the center at ground level. So, every axis starts from 0. This is an important step to make sure that when it is imported into the cube project the model isn't floating and aligned to the center of the floor cubes.

The downsizing process leaves behind gaps in the model. This makes the cube shaped model unrealistic as cubes are simply floating in the air. To remove any possible floating point, two methods were used. First for every point I found 10 nearest neighbours. After this I added blocks to connect the point with each of its 10 neighbours. For example, if a point lies at (1, 2, 3) and one of its neighbours is at (2, 6, 5), then points are added at (2, 3, 3), (2, 4, 3), (2, 5, 3), (2, 6, 3), (2, 6, 4). As shown in Figure 13, the yellow points are added to create a pathway between the two points. In order to add points, the counter starts from the first point and then adds points starting from the x axis value of the first point to the x axis value of the second point, and similarly continues adding from the y and z axis as well.

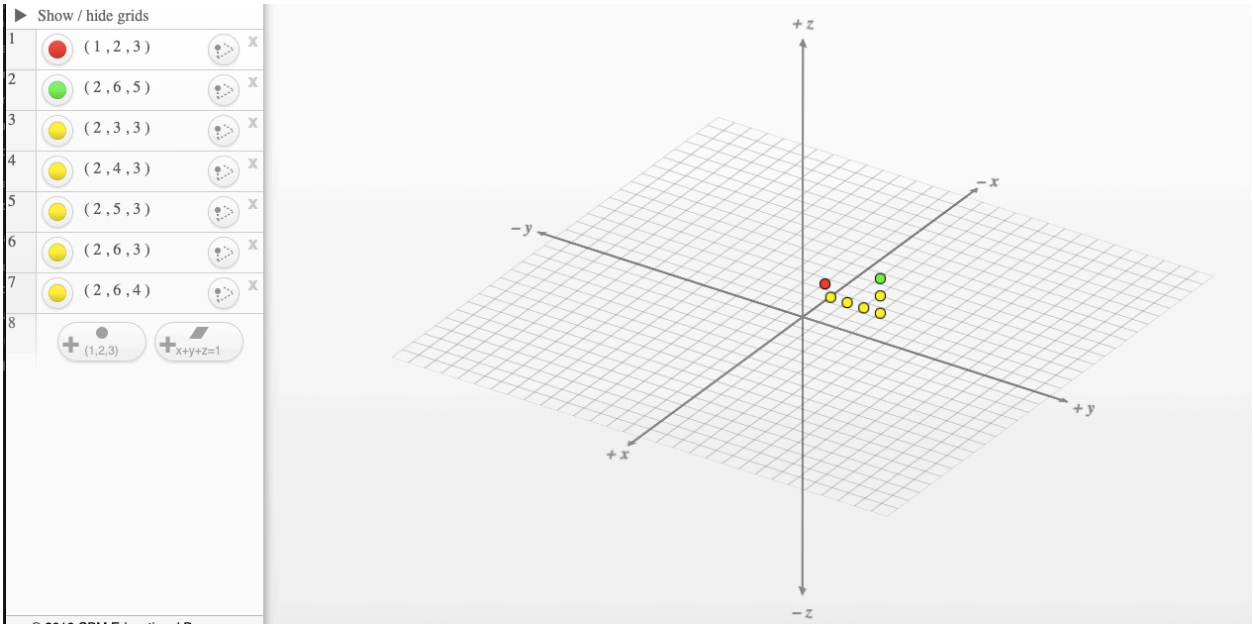


Figure 13: Pathway created between two points by adding points

A second technique used to remove the floating cubes is to add points below any possible floating points. However, before adding the points to the ground two things are checked. First it is checked that there is no point below the point we are checking and second that the point is not directly next to another point in either four directions to its side. If neither of these conditions are met, then points are added from that point to the ground. This heuristic ensures that points which are floating as a result of the model are preserved, such as the handle of a char or leaves of a plant.

4.5 Merging with Cube-shaped Robot

After the point cloud is reduced and modified so that only the important points remain, it can be used by the Cube-shaped robots project. The code has been modified to read a point cloud and store the x, y, z coordinates. I have also added the ability to further downsize the point cloud if required. This is done because the Processing tool used for the project has a limitation that does not allow any line of code to run for more than 5ms. Due to this the file sizes have to be reduced and point clouds made as small as possible so that it is readable while retaining its shape. As the shapes given as input are much larger than the predefined shapes in the project, I have also added the capabilities of releasing new cubes in an interval of 10 seconds. This gives the cubes which have found a location time to move up and make space for the new incoming cubes. Finally, the cubes of the model have been enlarged and borders are added to show a well-defined shape.

5.0 Challenges

The entirety of the development process for this project was met with its fair share of challenges. The following section describes the challenges faced during the project and steps taken to overcome them.

The first challenge came in being able to capture stereo images using a single camera. As mentioned previously stereo camera setups are difficult to place in the right position so that the two cameras are perfectly aligned with each other and at the correct distance. This problem was further enhanced as I tried to achieve a stereo camera setup using only one camera. While this is achievable, it is difficult. I achieved this by creating a setup which used a box cut-out on a tripod to keep the cameras aligned on the same plane. This allowed for a primitive stereo setup. Camera calibration was done using images of the chessboard taken by placing the camera at both the right and left position. After calibrating the cameras, I captured pictures of an object by placing the camera at both the positions. I also fixed the images to remove lens distortion. The two images were used to find a depth map. However, because the stereo setup was not perfectly calculated while I could get a depth map from the two images, the map wasn't defined enough to be useful. It contained holes in the mapping. I then tried to filter the depth map using a window based median filter to patch the holes. But because the depth map was sparse, the filtering did not provide a usable 3-D point cloud. In order to overcome this, I used stereo images from an online repository [http://vision.middlebury.edu/stereo/data/scenes2006/]. The repository provides a pair of images captured by a stereo camera and the focal length of the cameras as well as the baseline distance between them. The depth map obtained from these images was complete and did not have holes in it.

A second challenge I faced was the limitation with the programming language in which the cube-shaped robot project was written. While it is a good language for 3-dimensional image manipulation, it is not good for handling large data. A feature within the drawing tool of the language has a pre-set timeout function which does not allow any bit of code such as loops, list creations, file reading etc to take more than 5 seconds. For this I added the part two of my project which dealt with modifying the point cloud files to fit the requirements of the Processing language.

Another challenged faced was when the model was translated to the cube version, there were many gaps between the cubes. This is not a realistic model as cubes cannot float in the air without support. To fix this I added points between the nearest neighbours of every point and added points below any potential floating cubes. This process is described in detail in section 4.4 of the report.

6.0 Results and Future Goals

All aspects of the project were met according to expectations. However, there were certain limitations of hardware which prohibited the use of a mobile phone to act as a stereo camera. However, given two stereo images, the project is capable of transforming the 2-D images into 3-D models by using their depth maps and then use that model with the Cube-Shaped Robot project, thus adding onto the capabilities of the original project. The following examples show the outputs of the various stages in development for multiple shapes.

Figure 14 A: Stereo Images of a Model 1

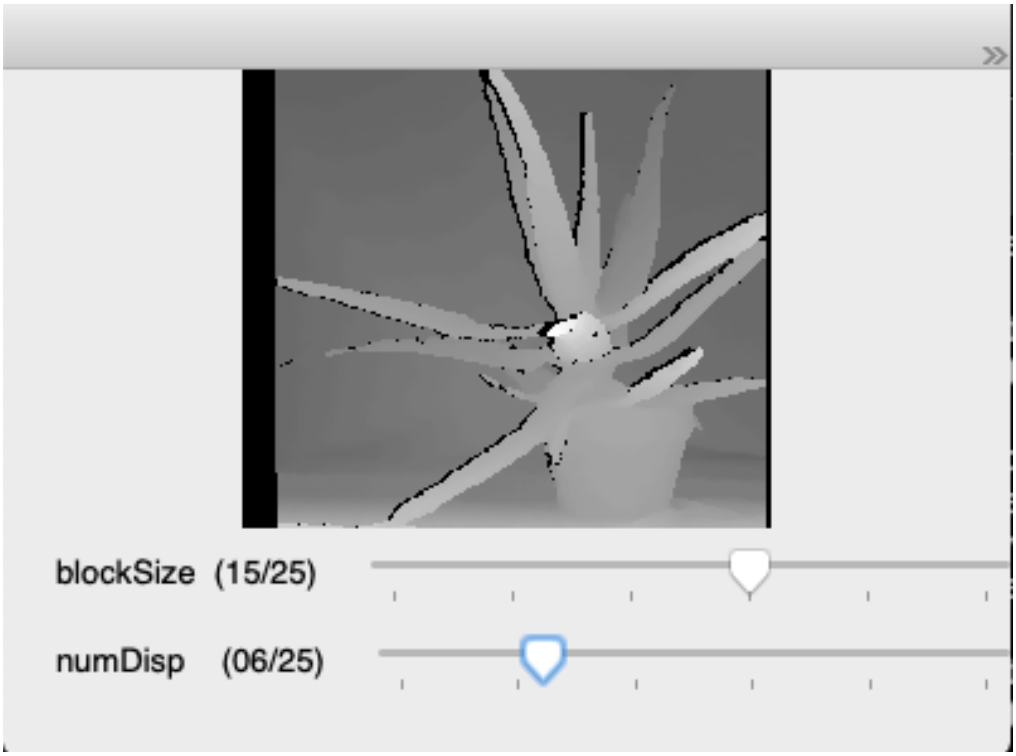
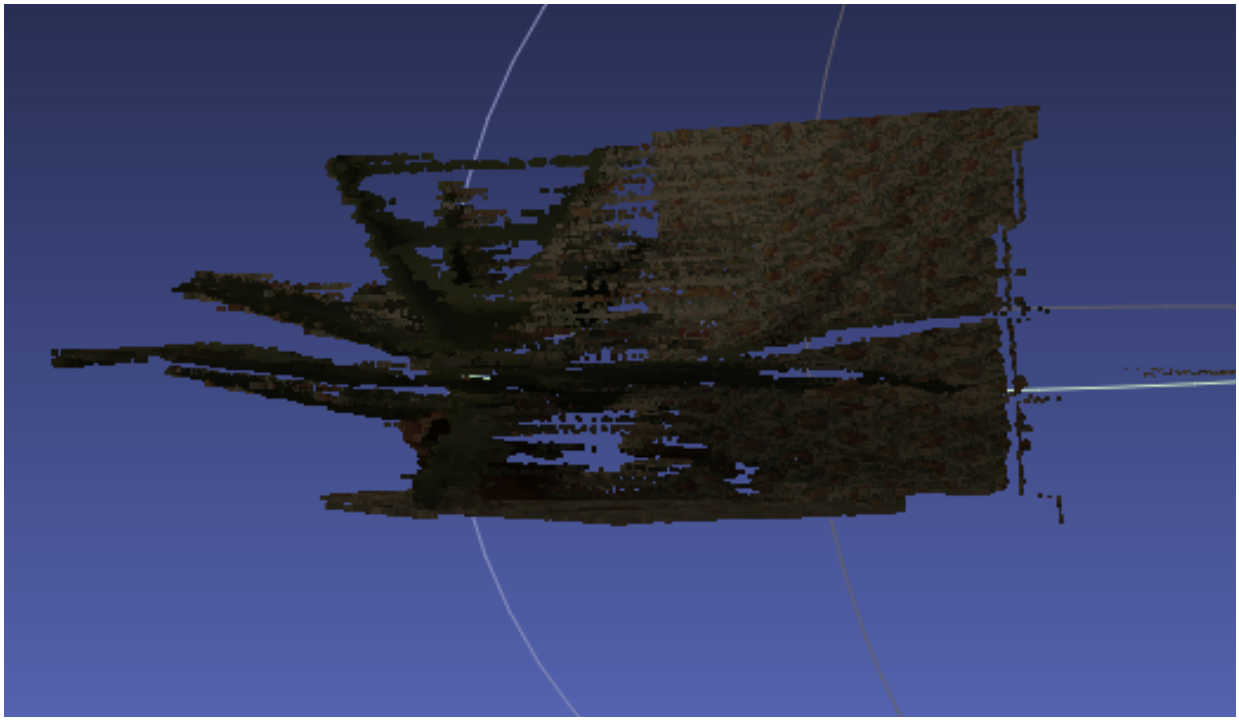


Figure 14 B: Depth map of Model 1



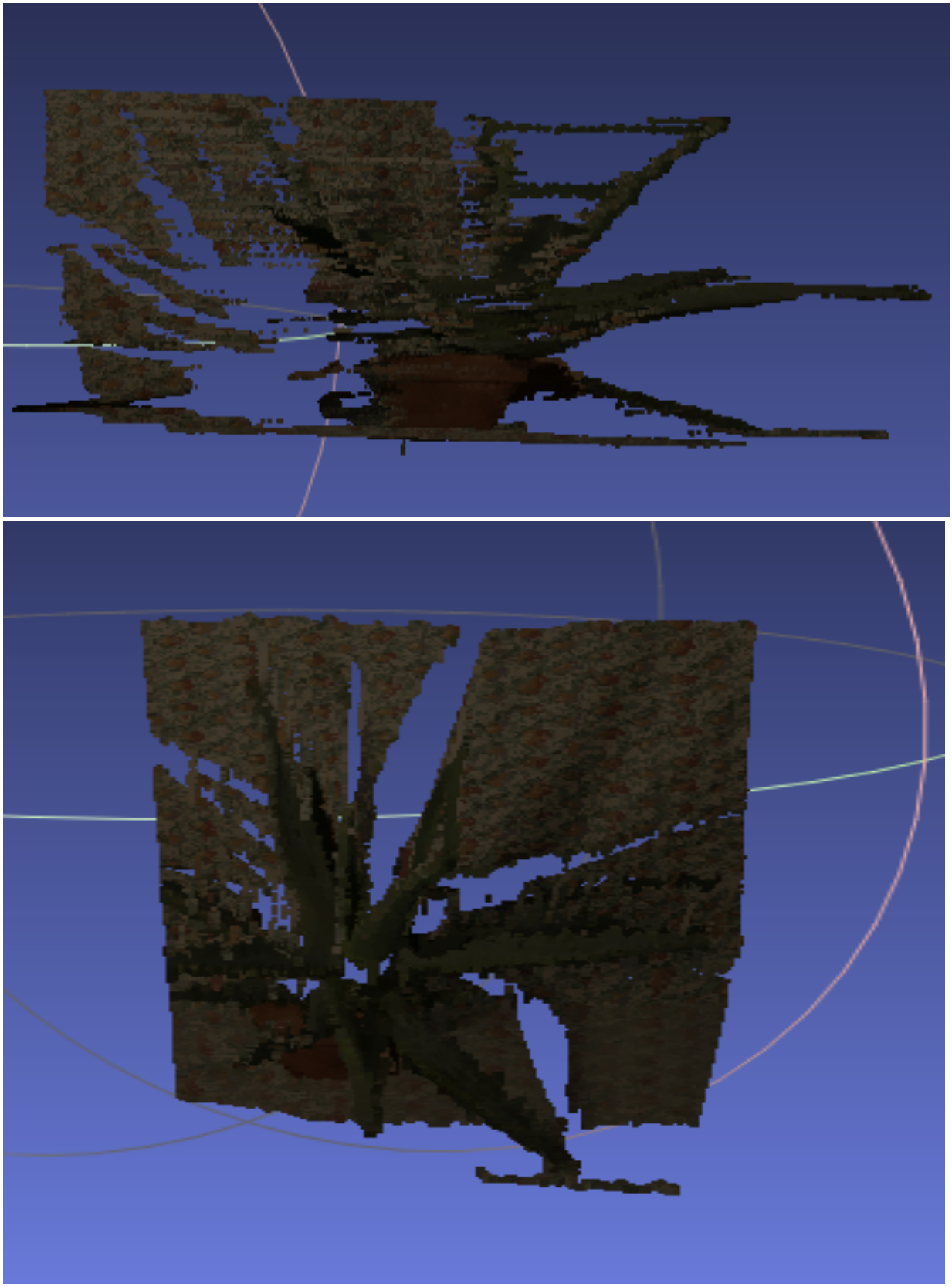


Figure 14 C: Various angles of 3-Dimensional model for Model 1

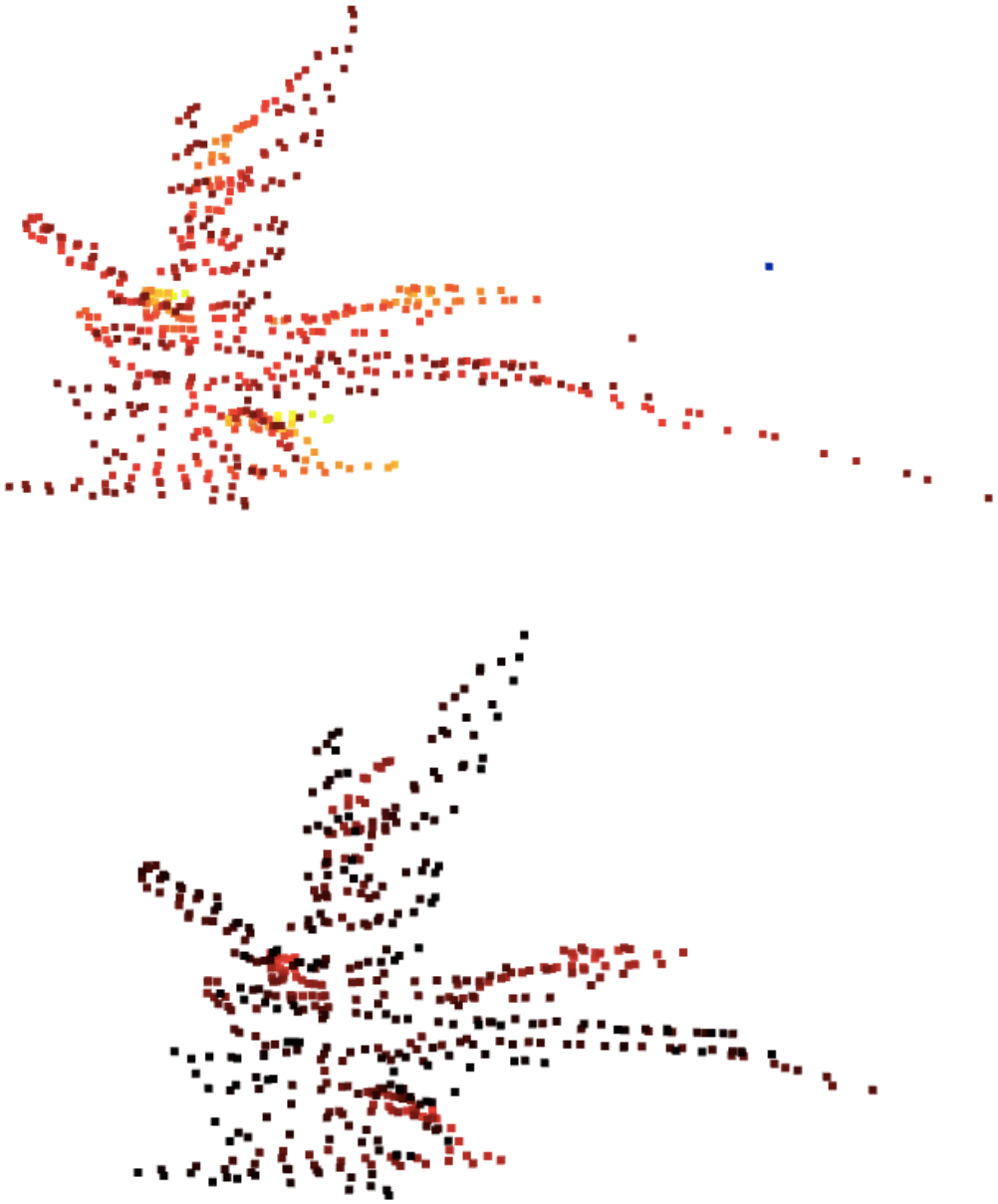
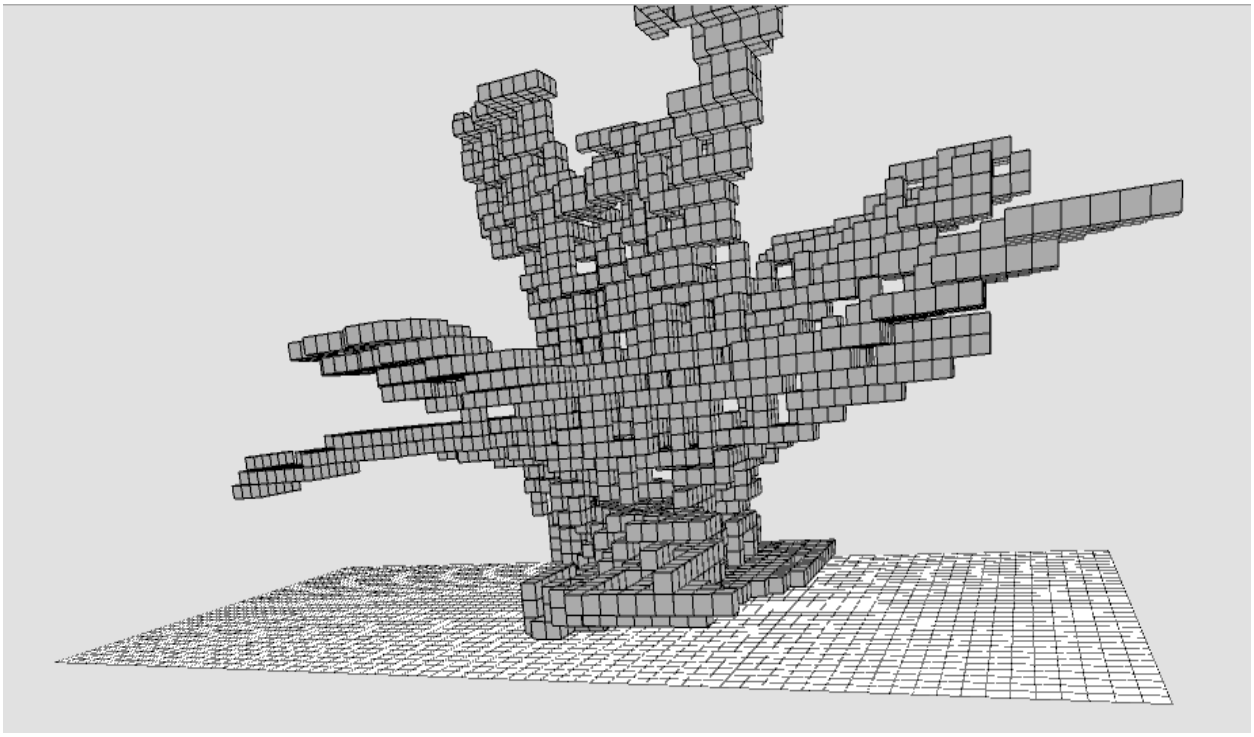
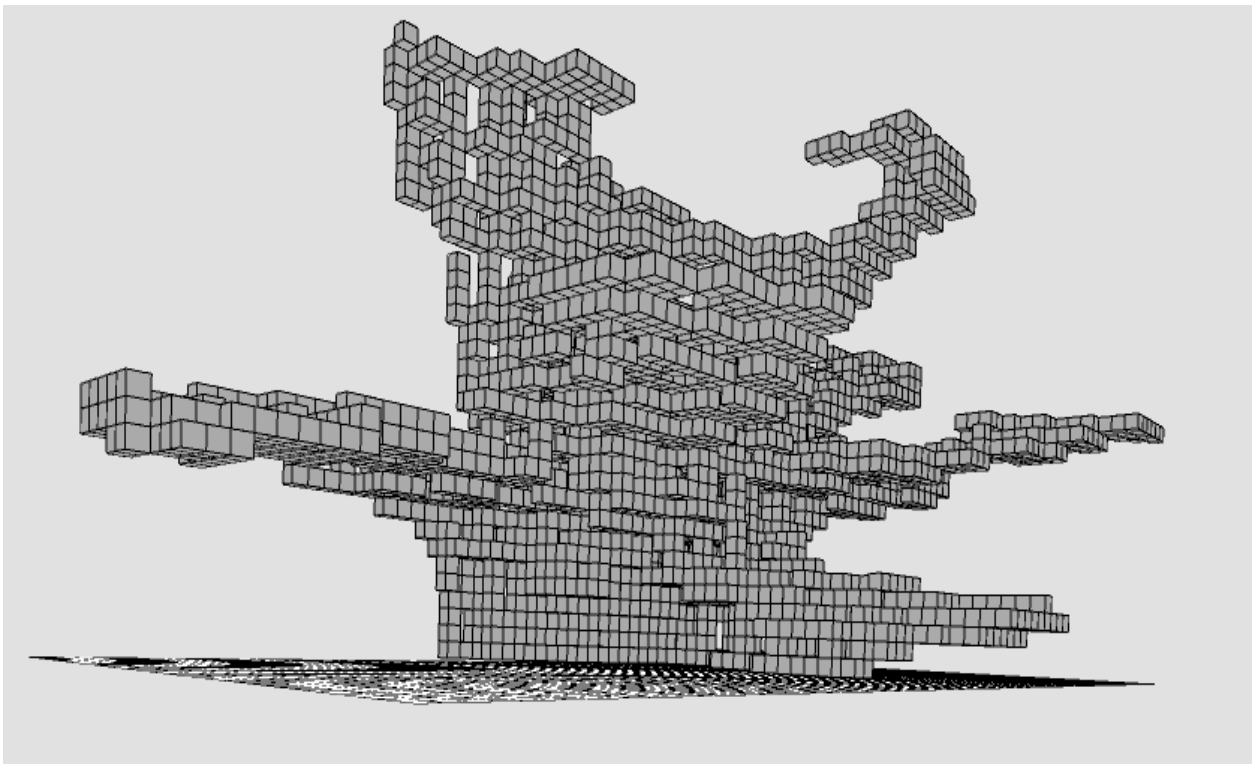
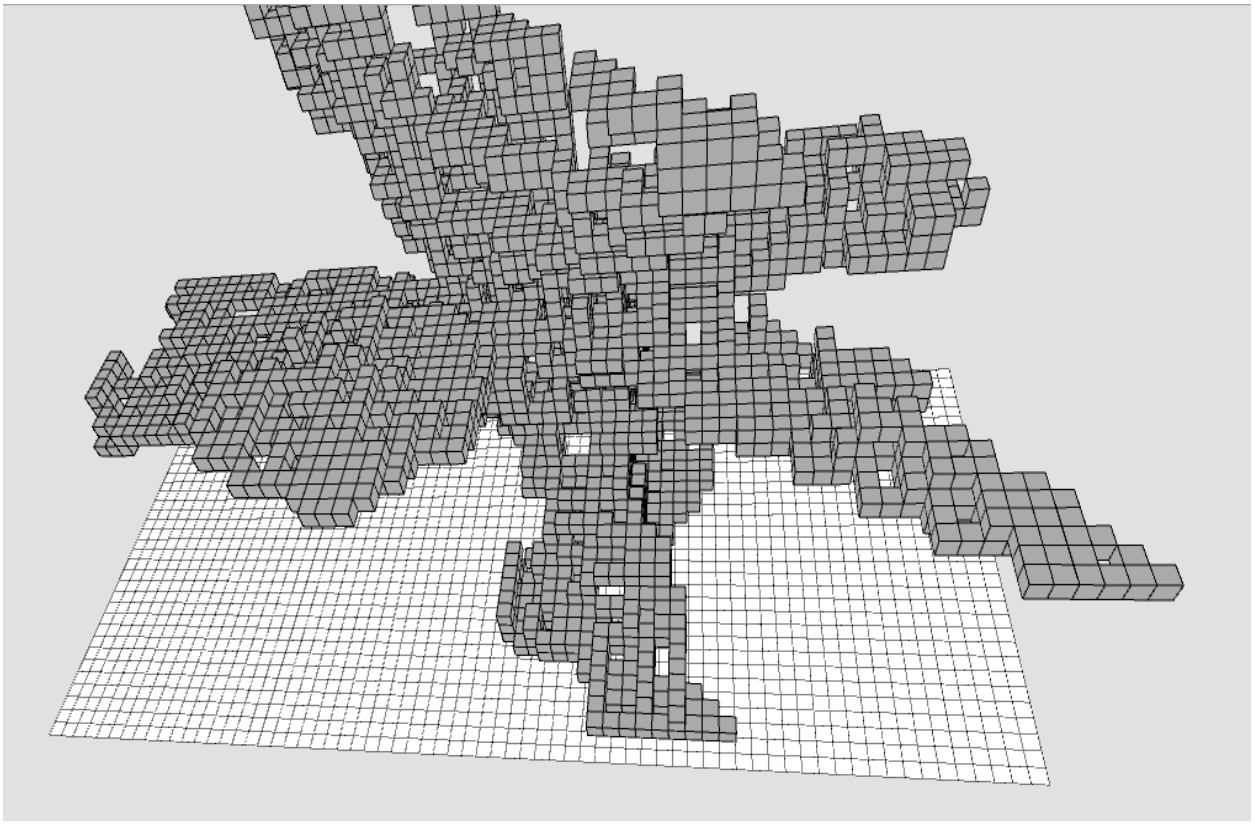
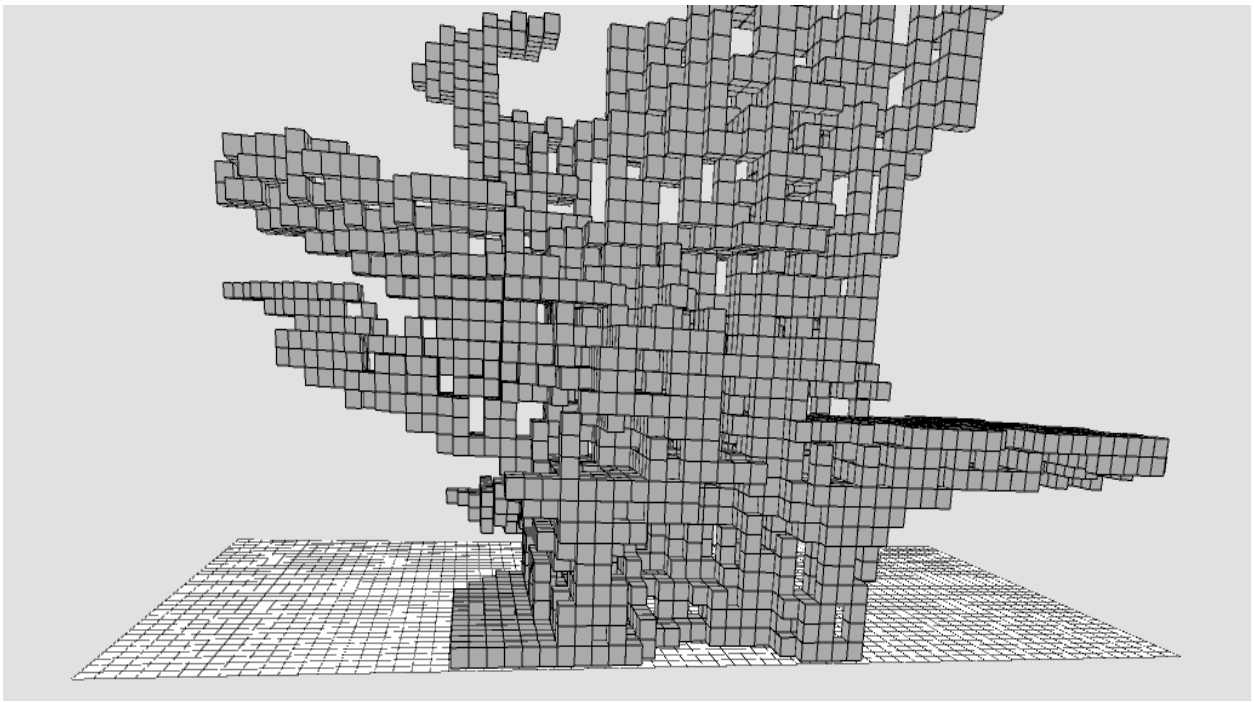


Figure 14 D: Outlier removal process for Model 1









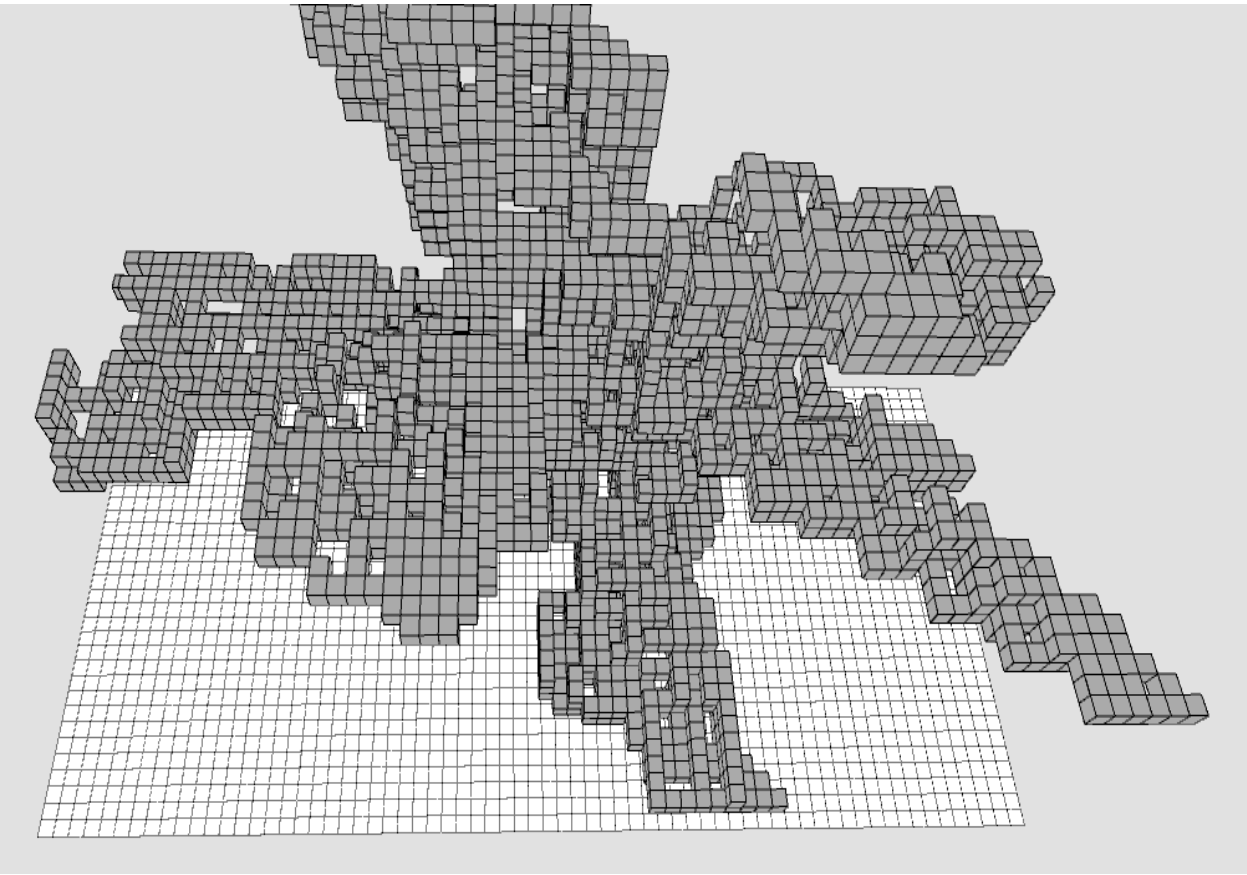


Figure 14 E: Various angles of Cube representation for Model 1

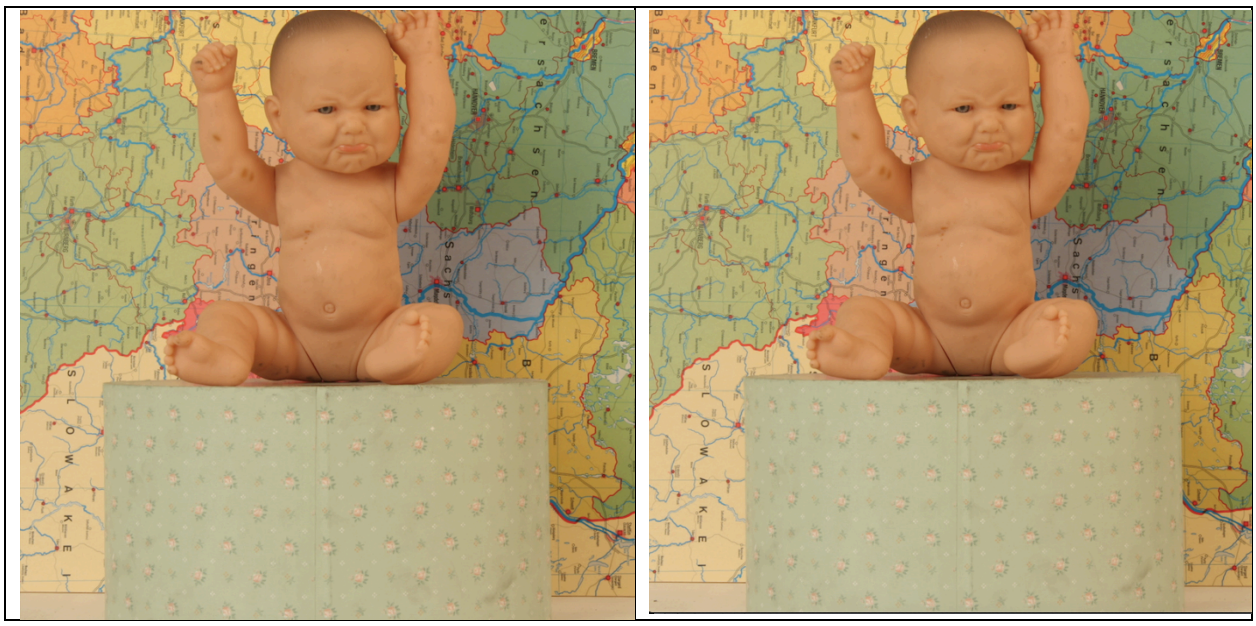


Figure 15 A: Stereo Images of Model 2

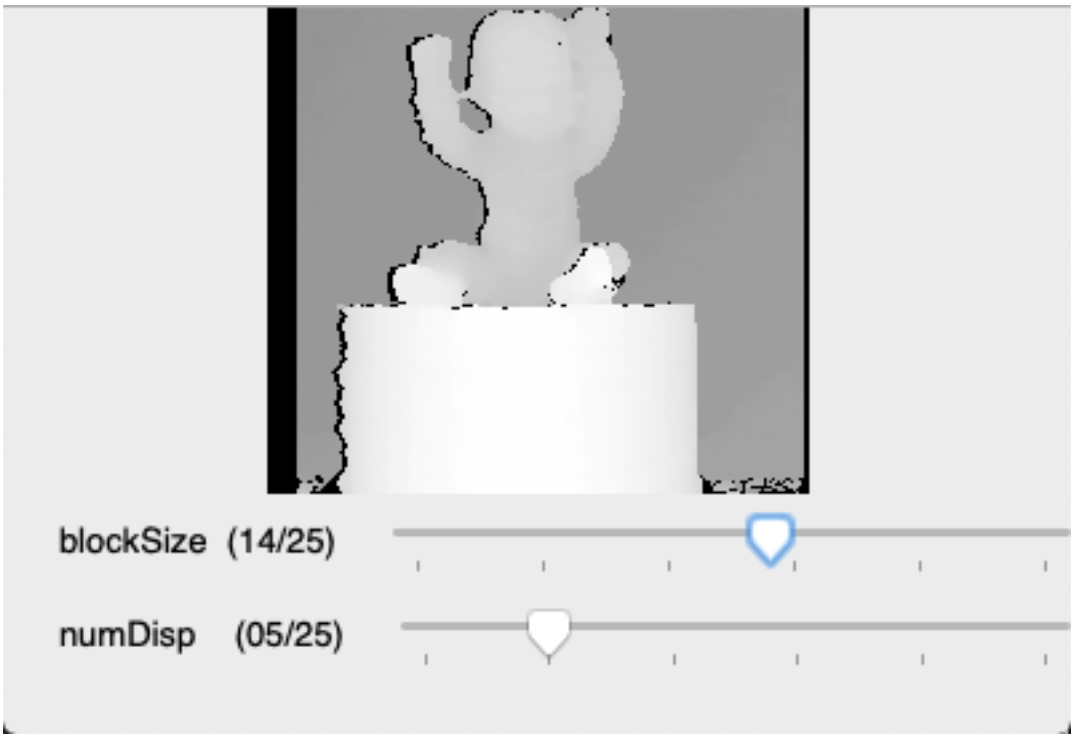
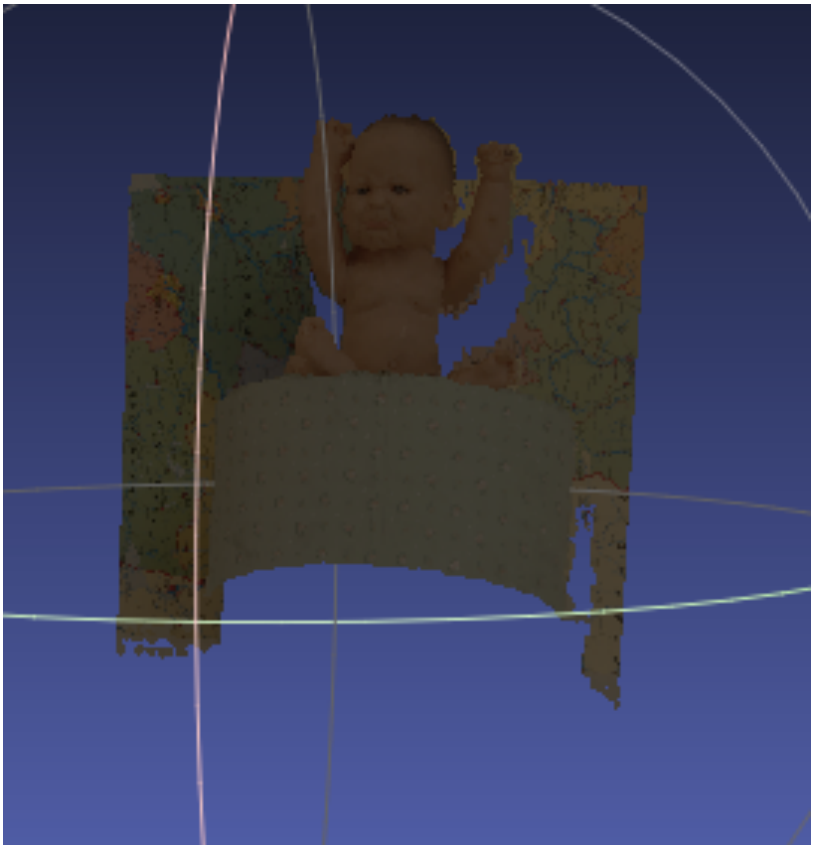
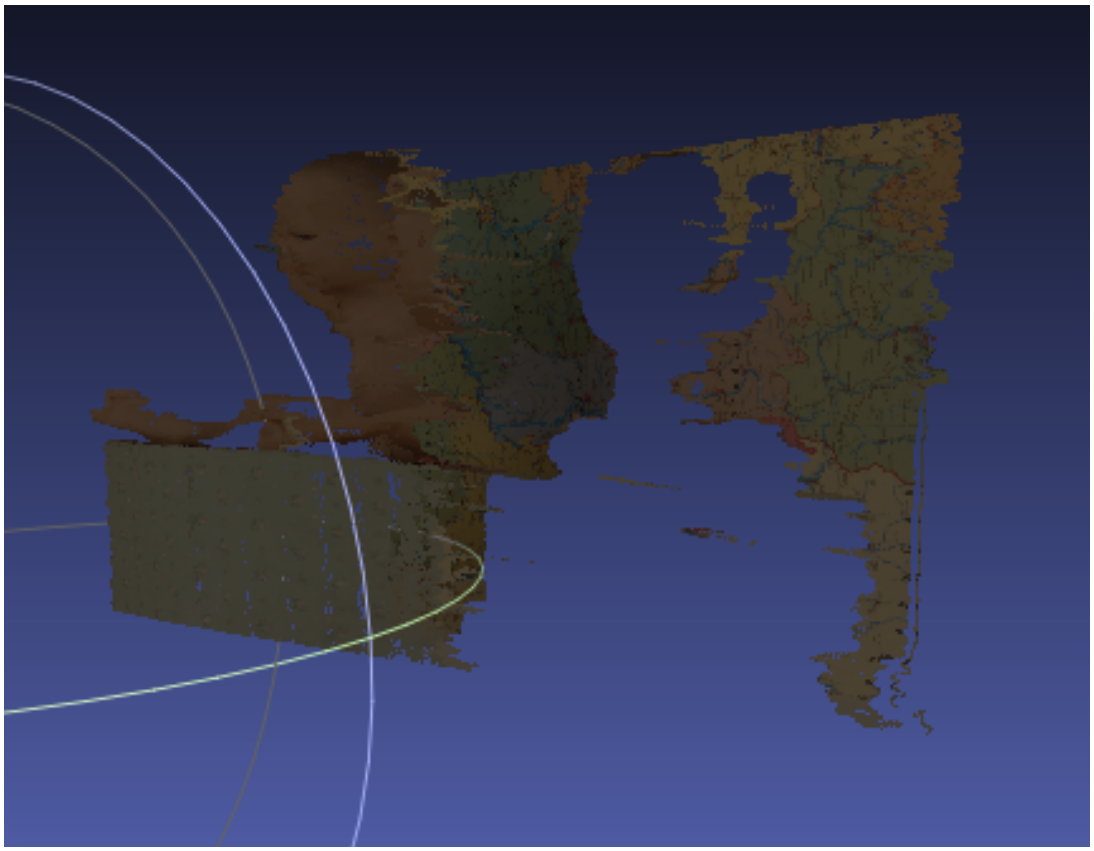
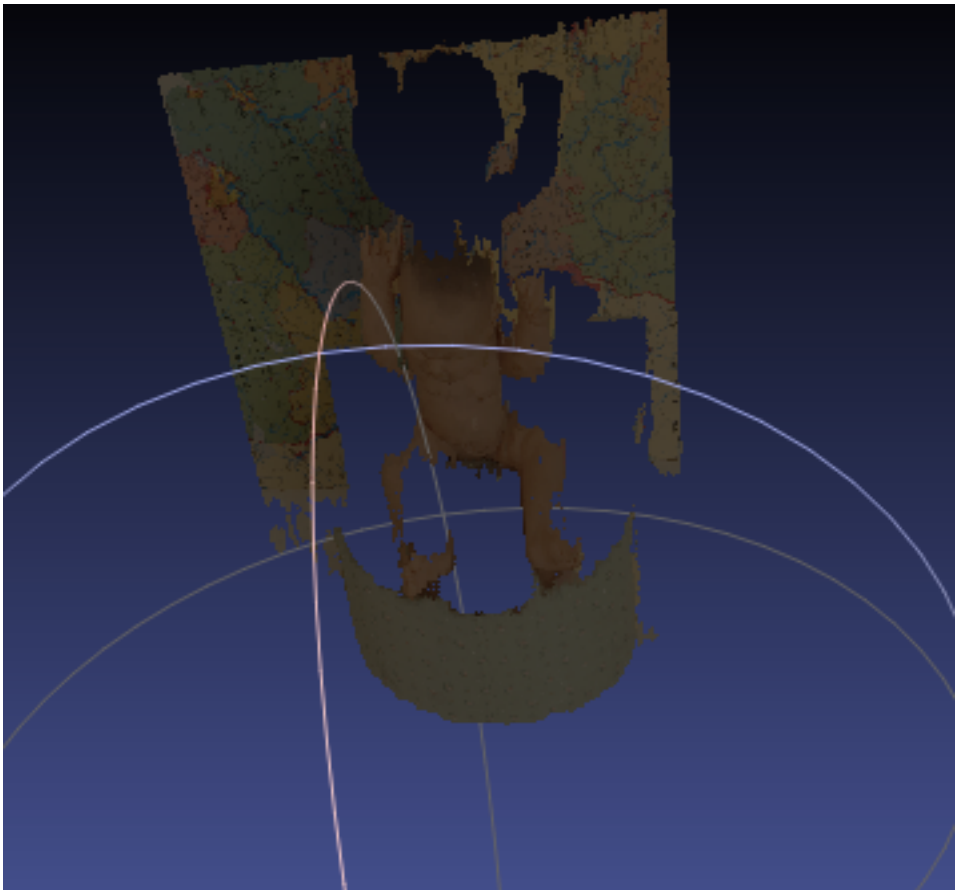


Figure 15 B: Depth map of Model 2







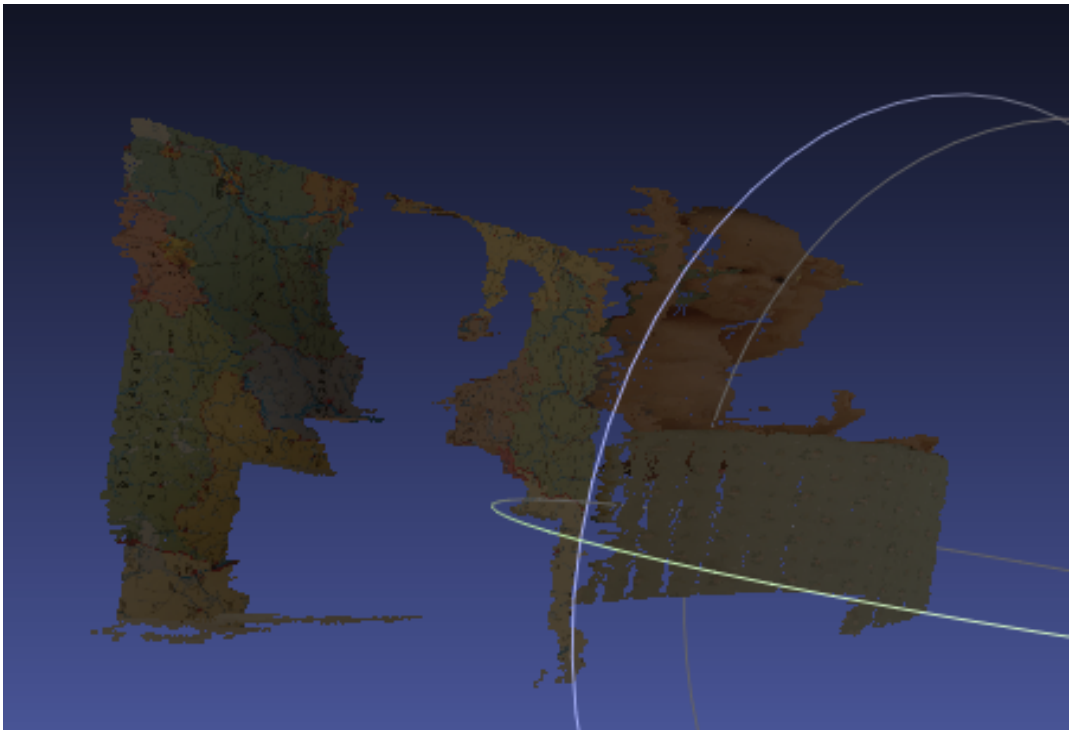


Figure 15 C: Various angles of 3-Dimensional model for Model 2

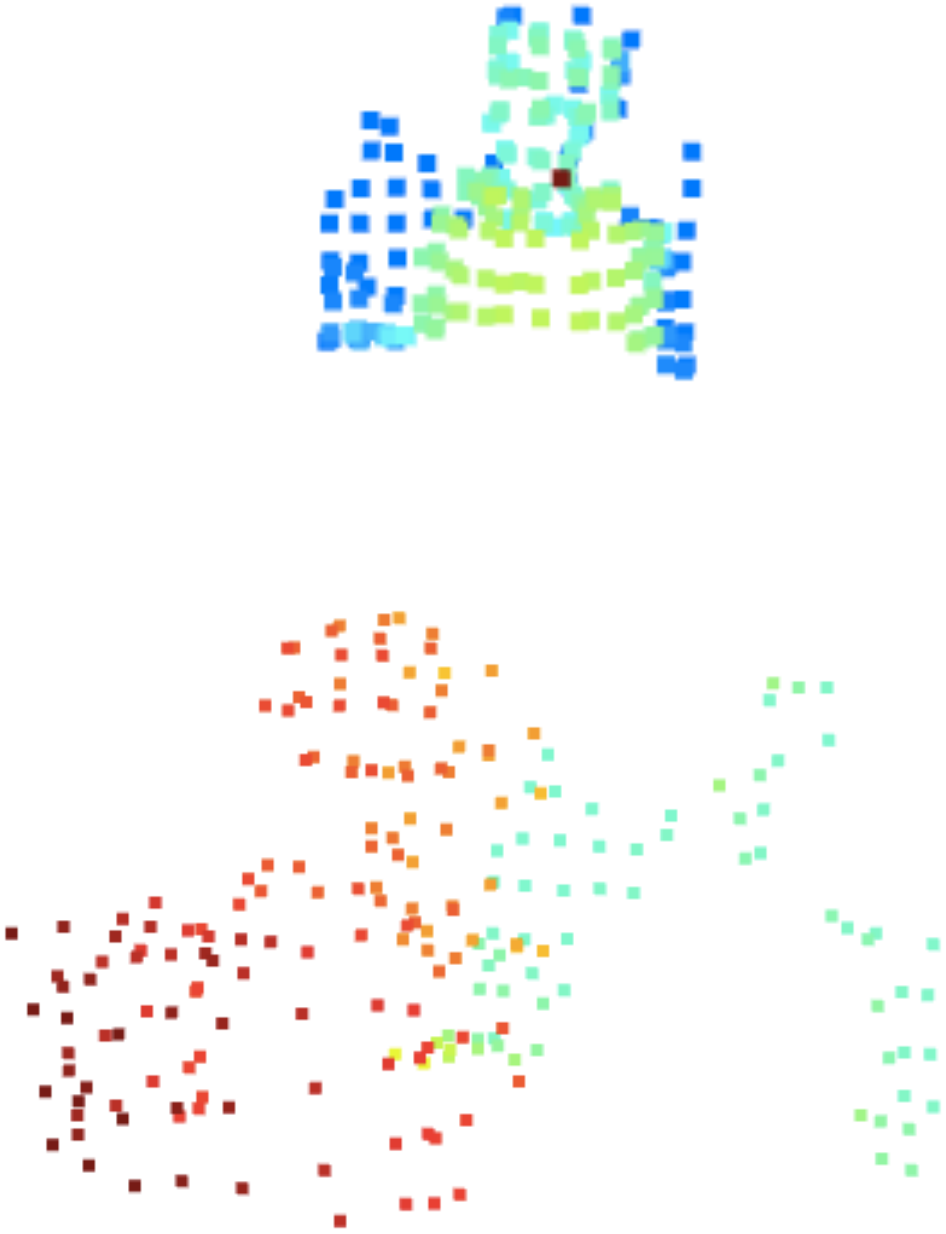
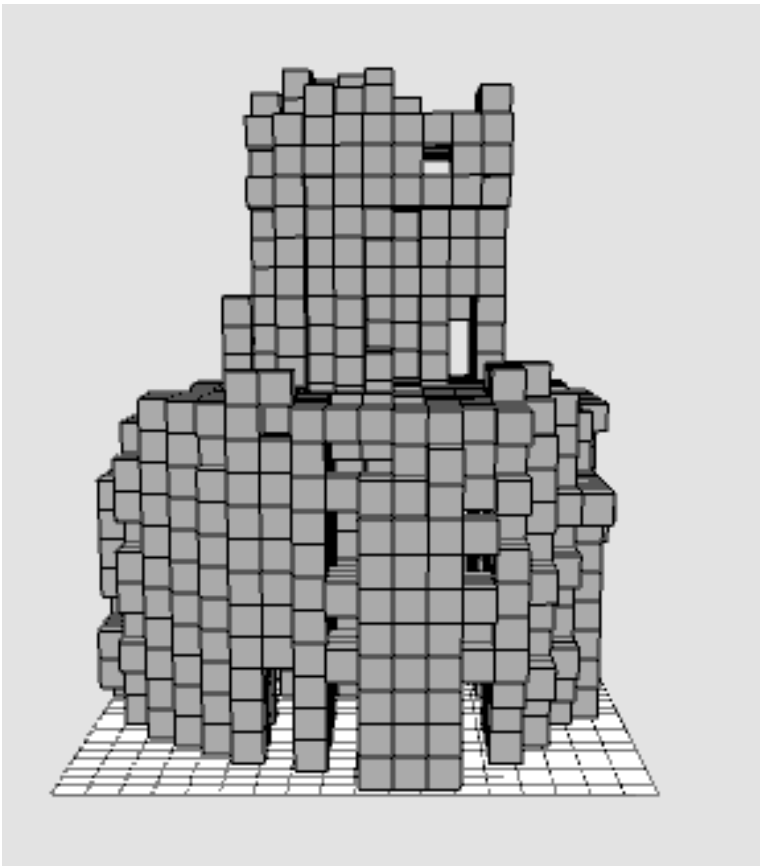
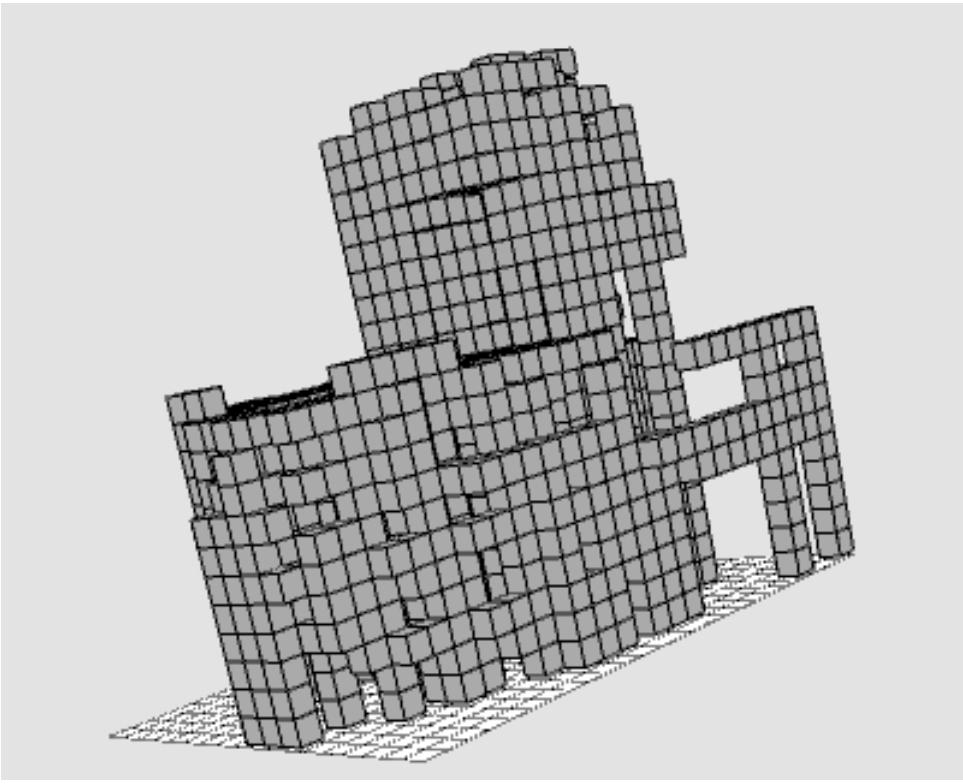
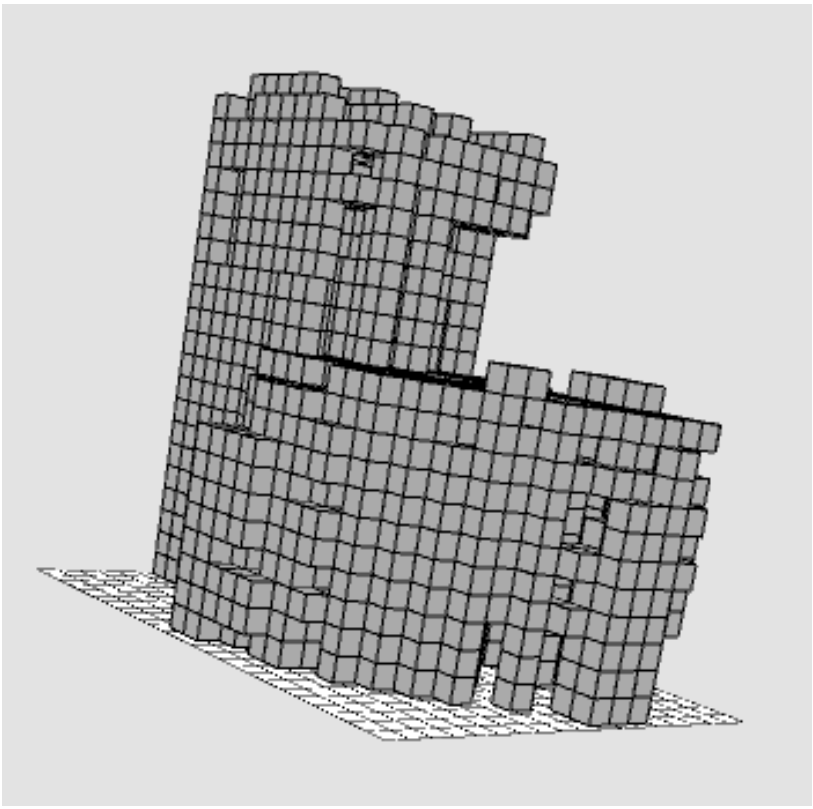
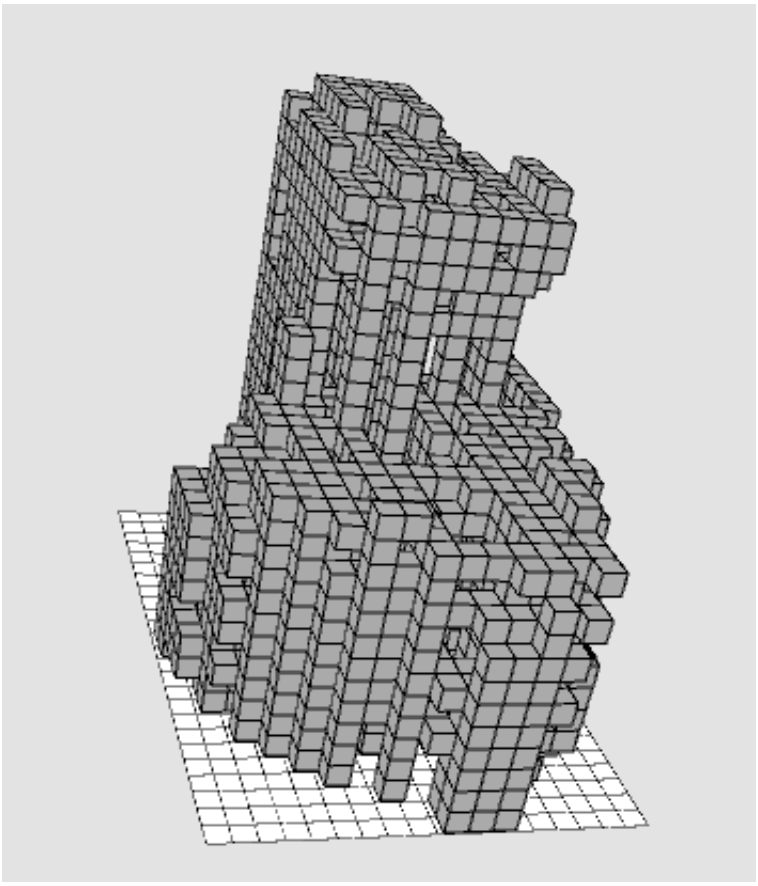


Figure 15 D: Outlier removal process for Model 2









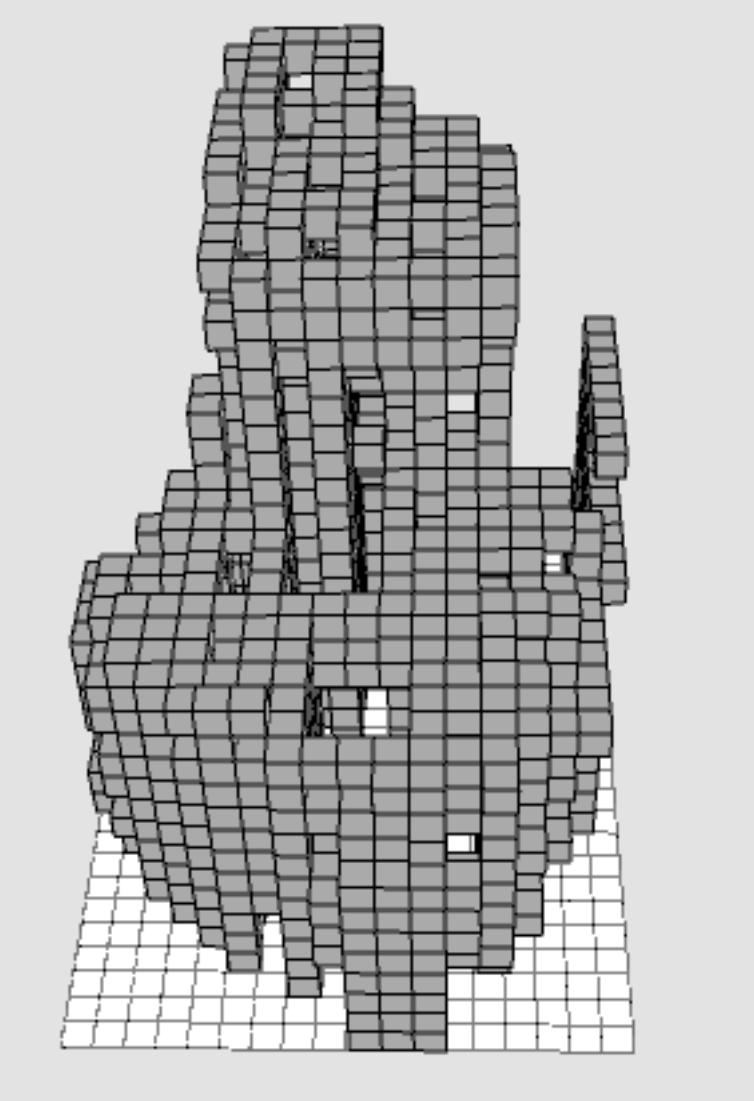


Figure 15 E: Various angles of Cube representation for Model 2

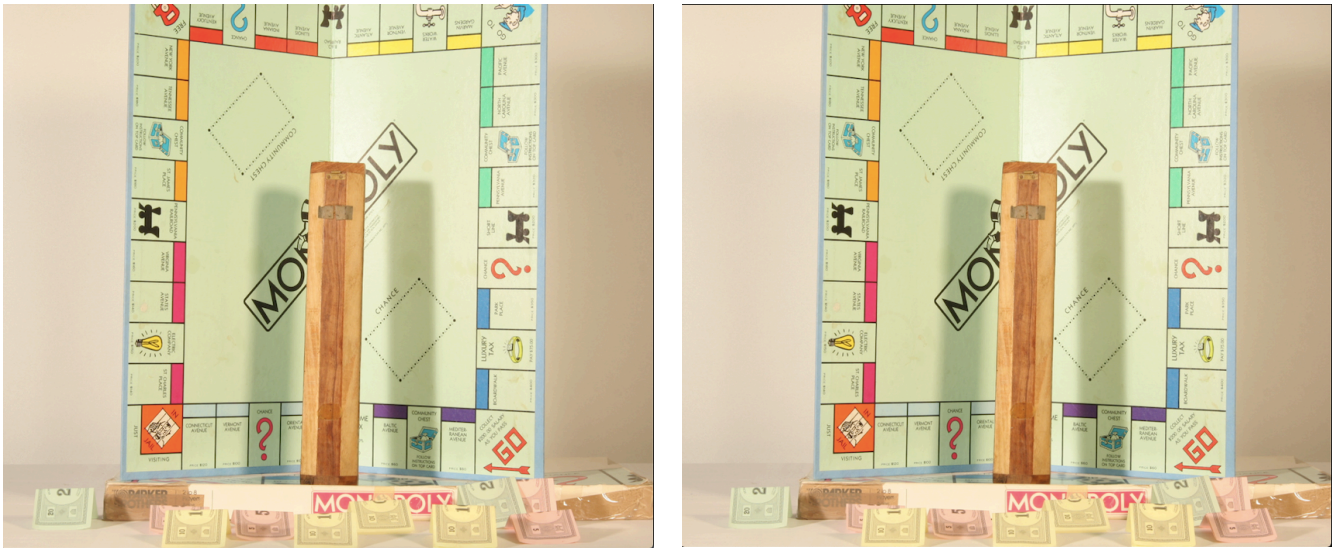


Figure 16 A: Stereo Images of Model 3

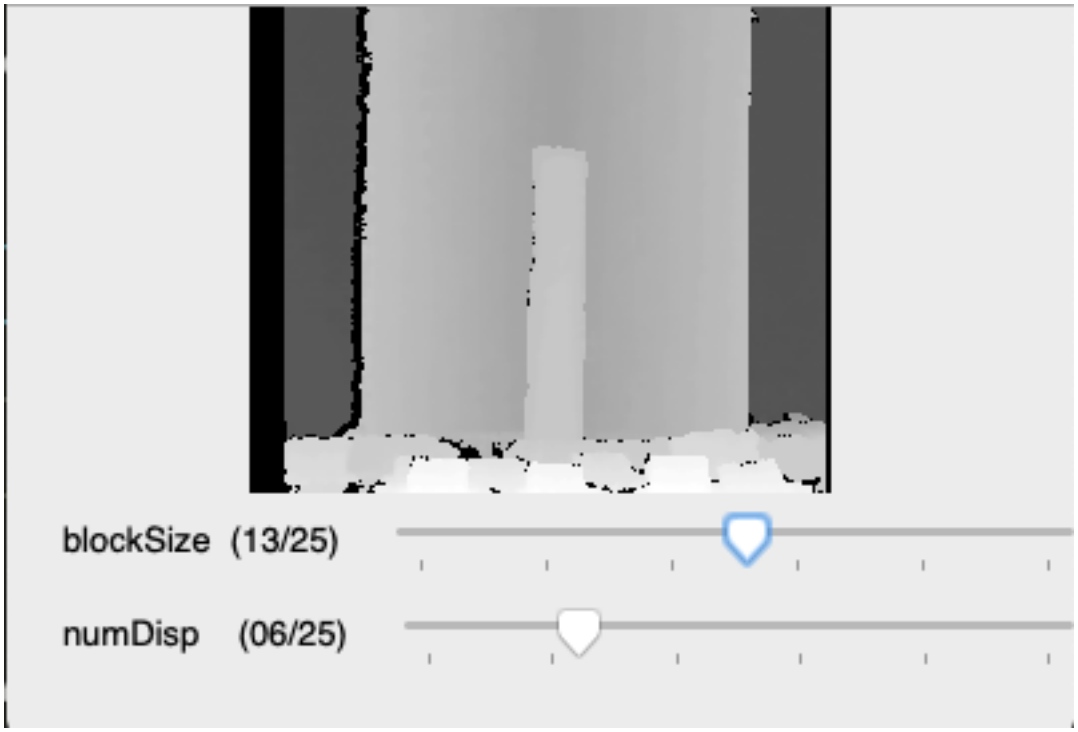
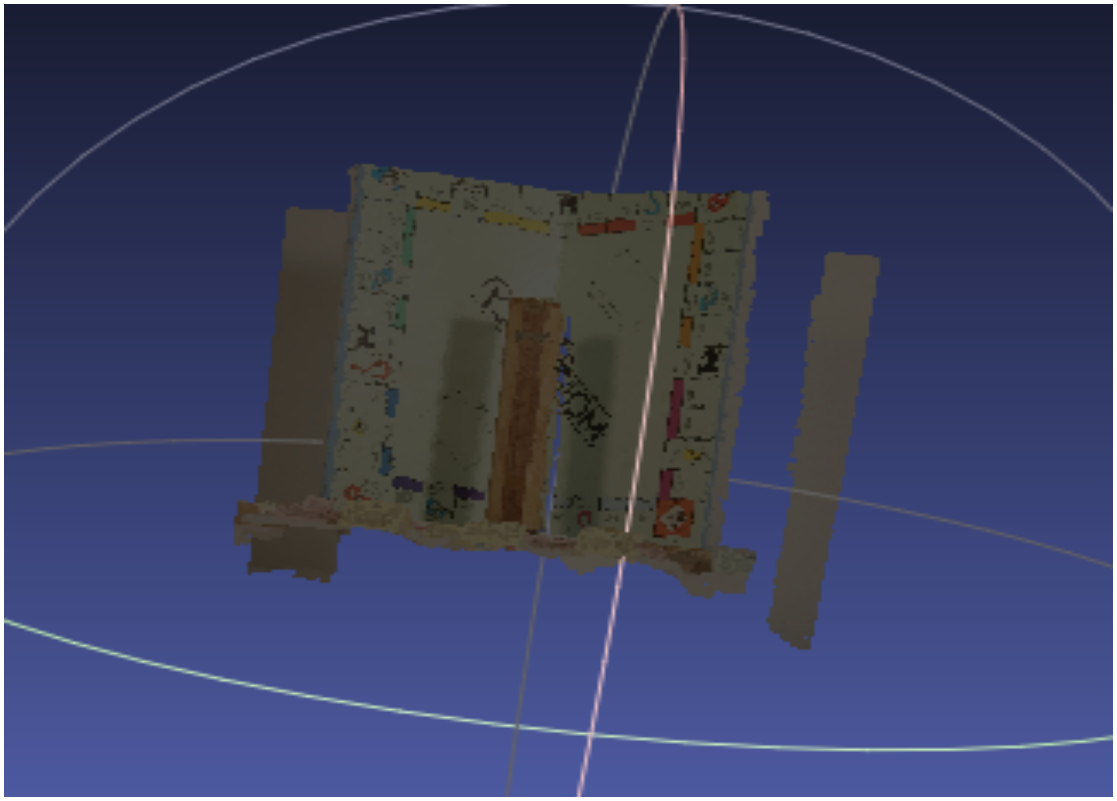
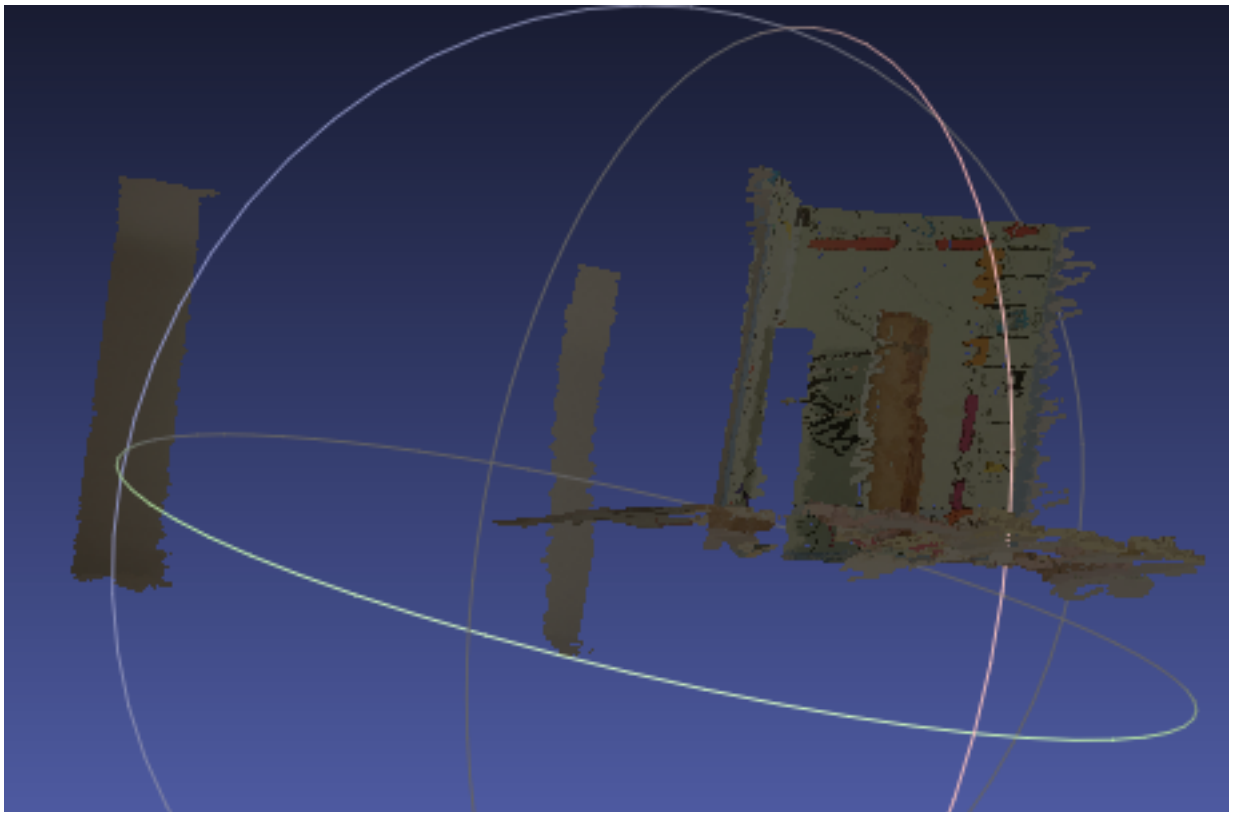


Figure 16 B: Depth map of Model 3





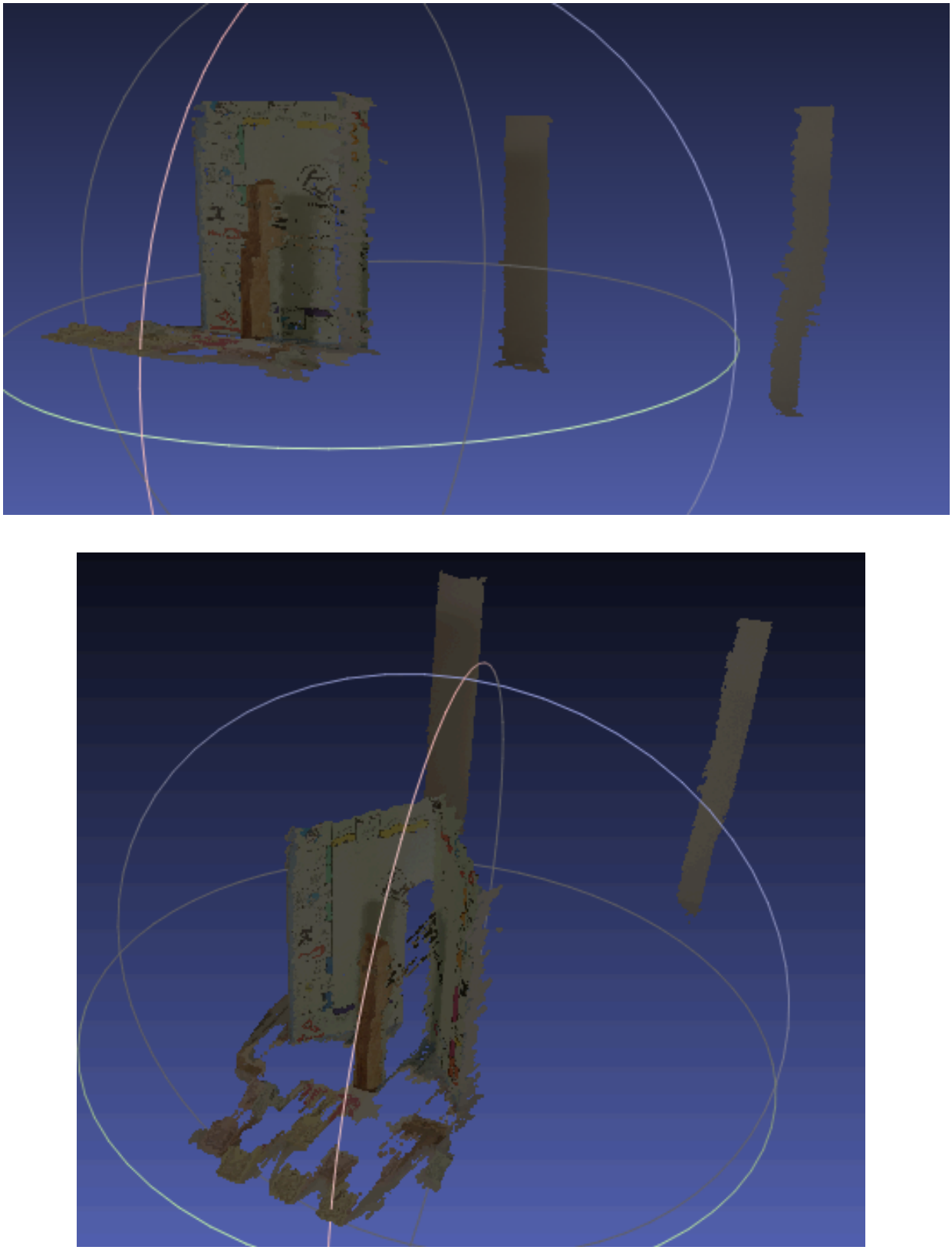


Figure 16 C: Various angles of 3-Dimensional model for Model 3

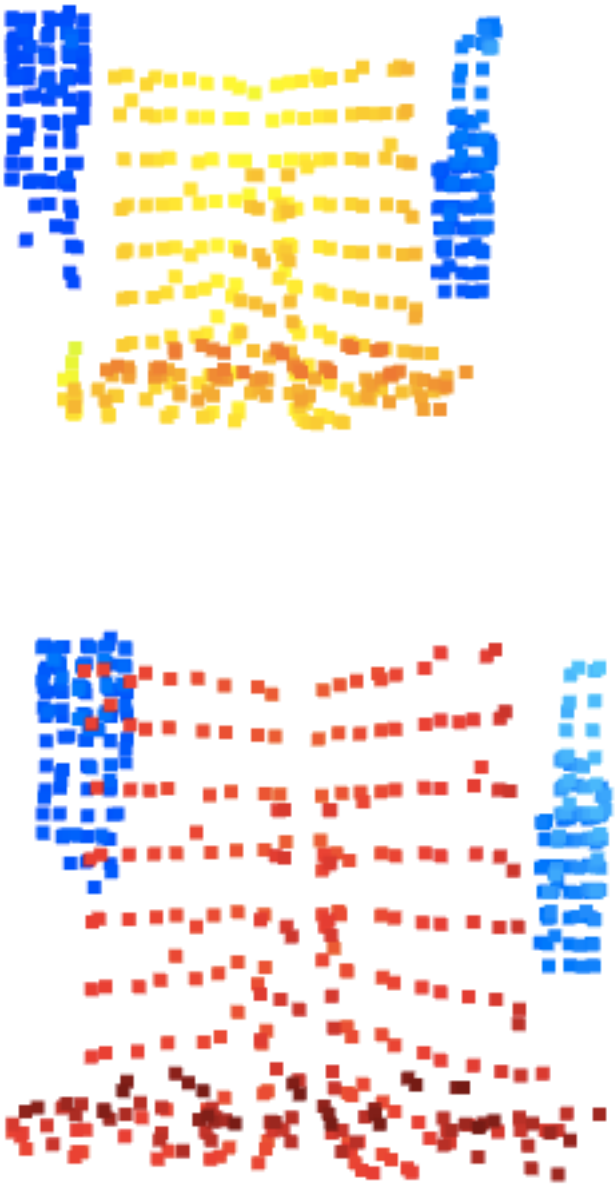
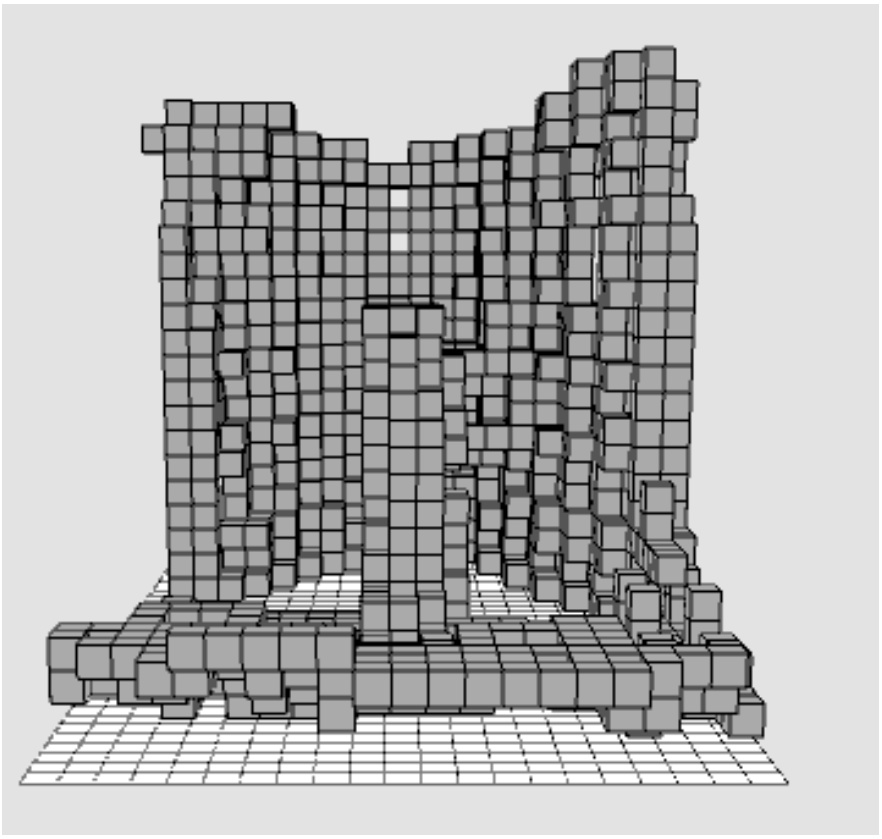
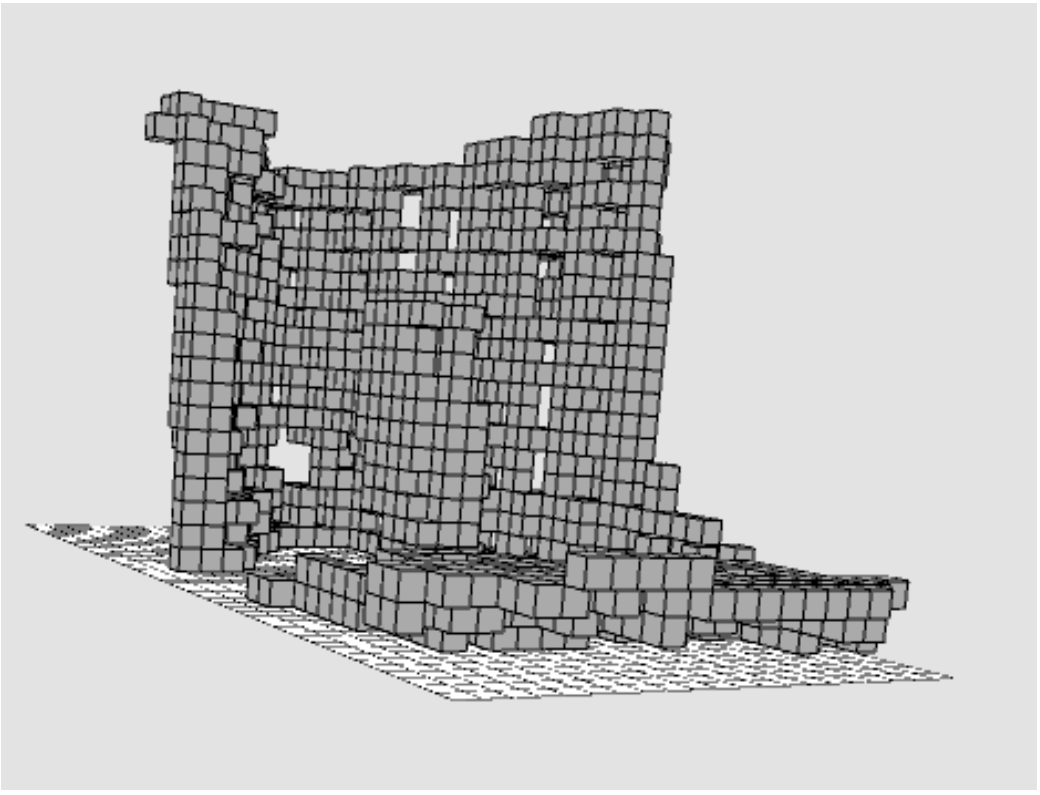
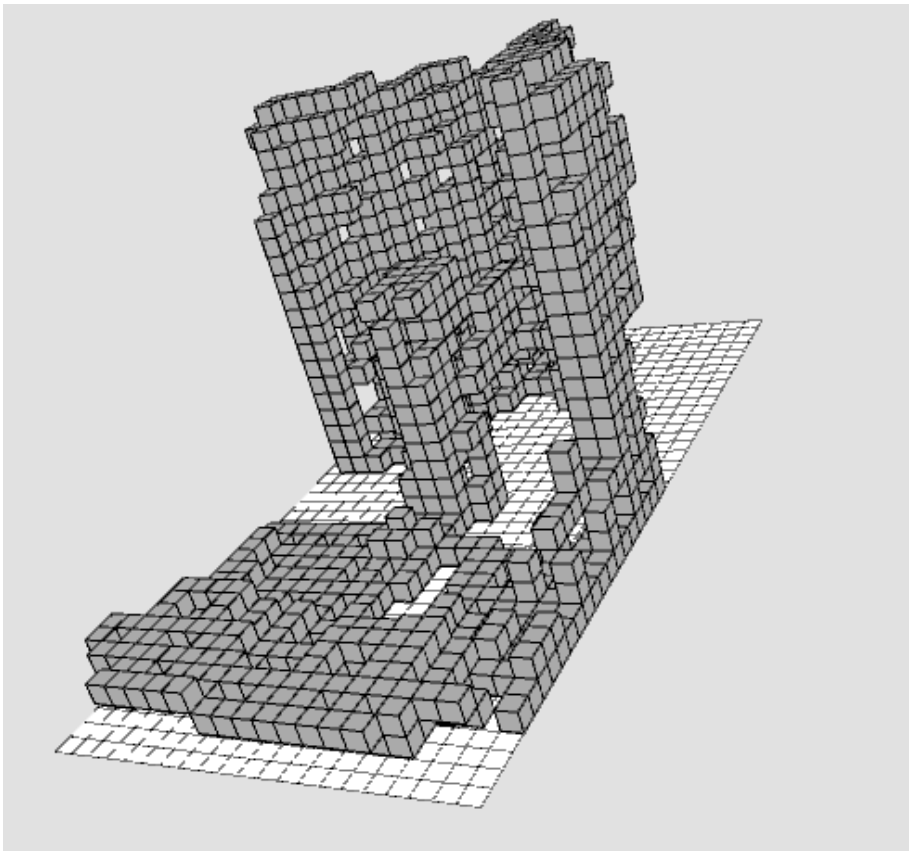
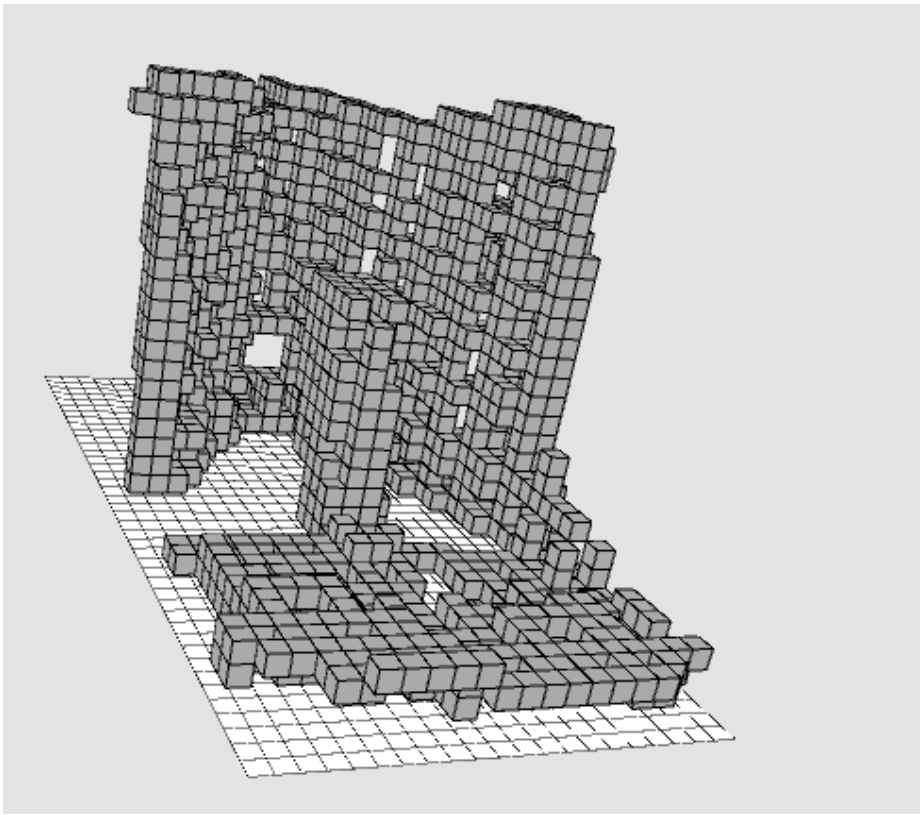


Figure 16 D: Outlier removal process for Model 3









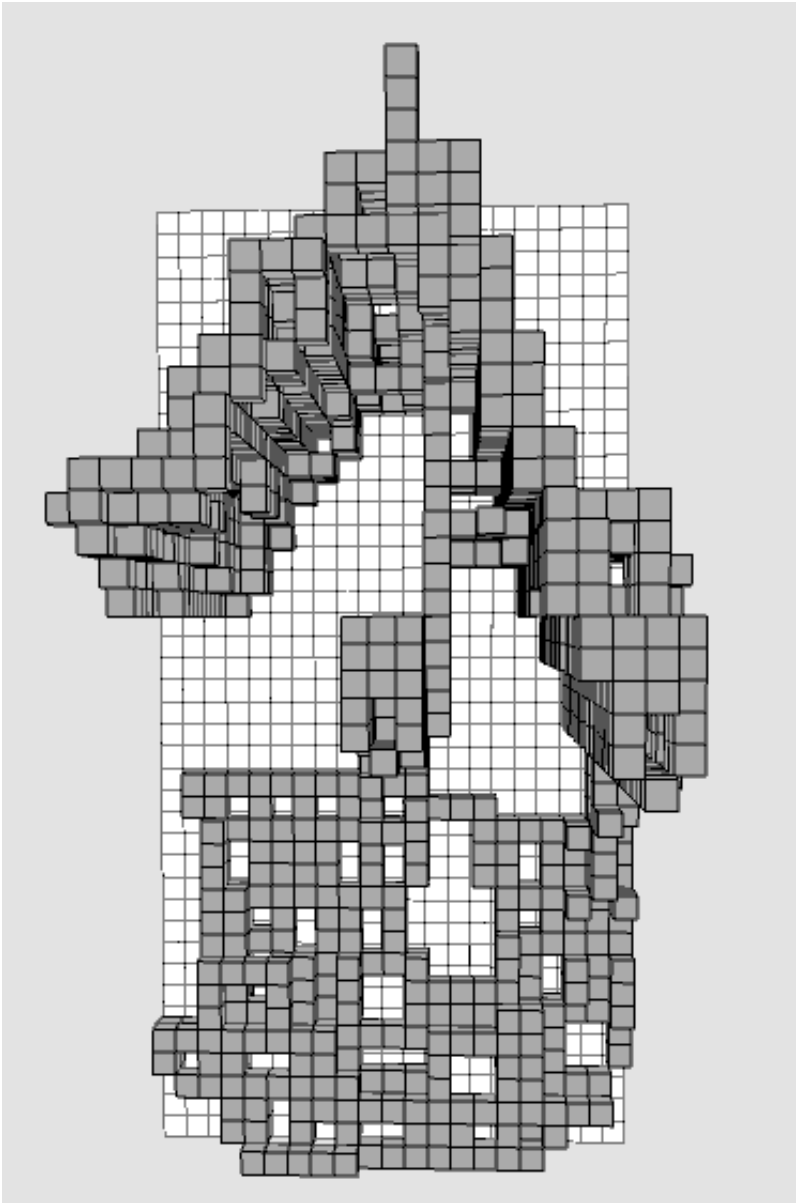


Figure 16 E: Various angles of Cube representation for Model 3

There are many ways in which this project as well as the Cube-Shaped Robot project can be enhanced. As a future goal for this project I would like to add the ability to capture the object from multiple angles and merge them to create an even more detailed and accurate 3-dimensional model. I would also like to fix the hardware flaws to set up my own stereo image system to capture the images. In terms of the Cube-Shaped Robot project, changes can be made in terms of how the cubes are released and how they climb to increase the efficiency with which the cubes build the model and to ensure completion every time.

Artificial intelligence concepts can also be used to build the structures. As a future project, principles of physics and architecture can be used while building the structures to account for realistic assumptions such as gravity. For example, while building a chair, the legs would need

support structures to stay upright until the seat is added. Another way AI can be used with this project is to teach a model how to recognize various features and shapes such as round corners, triangles, rectangles etc. within the point cloud. This can be utilized to compare features in the original point cloud to features in the model to provide a statistical analysis of the accuracy of the derived cube-shaped model.

References

De la Escalera, Arturo, and Jose María Armingol (2010), Automatic chessboard detection for intrinsic and extrinsic camera parameter calibration, pp 2027-2044.

Y. M. Wang ; Y. Li ; J. B. Zheng (2010), A camera calibration technique based on OpenCV

Nasim Mansurov (2013) What is Lens Distortion?

Pierre Drap and Julien Lefèvre (2016) An Exact Formula for Calculating Inverse Radial Lens Distortions

Kenji Hata and Silvio Savarese, CS231A Course Notes 3: Epipolar Geometry

Libraries used:
Opencv : https://opencv.org/
Open3d : http://www.open3d.org/
Stereo image dataset: vision.middlebury.edu/stereo/