Daniel Silva

CAP6419

Final Project – Single View Modeling

In this assignment, we were tasked with 3D models using single view metrology techniques.

Approach

The first step in reconstruction is to find the vanishing points along each axis in the image. To do so, I first mark *n* parallel lines along each of the x, y and x, axis'. The best fit of the intersection of the lines is the vanishing point. To find the best fit, I leveraged an approach described by Bob Collins that enables one to find the intersection between n parallel lines. The approach begins by building the "second moment" matrix using the lines as such:

```
         [  a_i*a_i   a_i*b_i     a_i*c_i ]
M = sum  [  a_i*b_i   b_i*b*i     b_i*c_i ]
         [  a_i*c_i   b_i*c_i     c_i*c_i ]
```

for i to N

The eigenvector of matrix M that is associated with the smallest eigenvalue is then the best fit vanishing point.

The next step was to select reference points and calculate scale factors to avoid affine distortions in the model. To do so, I select the origin of the scene's coordinate frame as an initial reference point, as well as an additional reference point on each of the X, Y, and Z axis'. The user must also specify either the true distance of these points from the origin or estimate the distance using unit lengths. With these 4 reference points I can then calculate the scale factor for each axis. Taking the x- axis, for example, the scale factor is calculated using the following equation:

$$a * distance_x * [V_x - ref_x] = [ref_x - O]$$

$$where\ a\ is\ the\ scale\ factor\ for\ x\ axis$$

$$V_x\ is\ the\ vanishing\ point\ along\ x\ axis$$

$$ref_x\ is\ the\ selected\ reference\ point\ on\ x\ axis$$

$$O\ is\ the\ coordinate\ frame\ origin\ in\ the\ image$$

$$distance_x\ is\ the\ true\ distance\ between\ the\ origin\ and\ reference\ point$$

Similarly, the scale factors for the y and z axis are calculated by:

$$b * distance_y * [V_y - ref_y] = [ref_y - O]$$

$$c * distance_z * [V_z - ref_z] = [ref_z - O]$$

With the vanishing points and scale factors computed, we then build the 3 x 4 projection matrix *P* that maps between image and world coordinates.

$$P = [aV_x \;\; bV_y \;\; cV_z \;\; O]$$

The projection matrix can then be used to build the homography matrices for each of the XY, YZ, and XZ planes. For example, H_{xy} is the 1st, 2nd, and 4th columns of *P*, H_{yz} is the 2, 3, and 4th, H_{xz} is the 1st, 3rd, and 4th.

```
% % Build projection matrix
P = [(x_vp*x_scale) (y_vp*y_scale) (z_vp*z_scale) origin];

% Get homography matrices from projection matrix
H_xy = P(:, [1 2 4]);
H_yz = P(:, [2 3 4]);
H_xz = P(:, [1 3 4]);
```

The last step before computing 3D world coordinates of the scene was to extract texture maps from the various planes in the image. To do so, I first performed a perspective transformation of the image using the above homography matrices to remove distortion from each of the planes. Then, I simply manually cropped the textures from the transformed images.

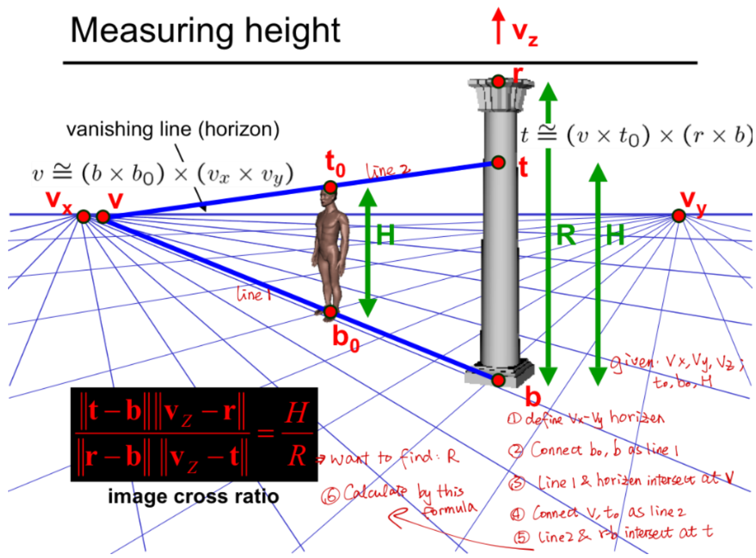
Finally, we compute the 3D world coordinates of selected image points using between plane and on plane measurements. To. For the first step, I begin by marking a base point $b = [x\ y\ 0]$ on the XY plane, along with a corresponding interest point $r = [x\ y\ R]$. Our first goal is to compute the height *R* of point *r*. To do so, the previously selected origin *O* and Z axis reference point ref_z are used to form the cross ratio shown below with the interest point.

$$\frac{\|t-b\|\|V_z-r\|}{\|r-b\|\|V_z-t\|} = \frac{H}{R}$$

$$note\ that\ H\ is\ the\ height\ of\ ref_z$$

Note that we are still missing point *t* in this equation, which is a point along the line between *b* and *r*. This point is obtained by first calculating the line between point *b and* b_0, the origin. We then find this line's intersection with the line at infinity, which we call point *v*. Next, we calculate the line between point *v* and our reference point ref_z. Finally, the intersection between this line and the line between points *b* and *r* gives us the point *t*.

With these four colinear points (*b, t, r,* ref_z*)* we can now solve the above cross ratio for *R,* the height of our interest point. The below figure does an excellent job of visualizing this cross-ratio computation.



With the true height of the interest point found, the next step was to calculate the true X-Y position of the coordinate. I accomplished this using on plane measurements and the homology H_z. Using the planar homology H_z between the two parallel planes we can map true X-Y coordinates to image plane u-v coordinates. From the paper, H_z is built as follows:

$$H_z = [aV_x \;\; bV_y \;\; aRV_z + O]$$

Thus, being that we have the u-v image coordinates of points *b* and *r,* we compute their world X-Y position through H_z^{-1}.

Once this initial 3D world coordinate at height *R* is computed, my program then prompts the user to select more image points at the same height. With height being given, for each of these points we then simply need to compute the X-Y coordinates using H_z from above. This enables one to quickly define a surface of points. Once the user is done with this Z level, they can select a new base and interest point, compute world coordinates at the new Z level, and continue in this fashion until all desired world coordinates are found.

The final step of creating the 3D model was to write the computed world coordinates to a PLY model, while mapping points to a corresponding texture map. I encountered the most challenges during this step, the biggest of which was including texture in the PLY models. From what I learned; one can only specify the RGB values for each coordinate in a PLY file, and you cannot use an entire texture map as the mesh or 'skin' of the model. Thus, to create high quality PLY models with texture I would have to manually select an extremely large number of points until the texture could be discerned from the scatter plot of colored coordinates when viewing the PLY model. Perhaps I am missing something, or did not do enough research, but I was not able to build or display my PLY models with a mesh. Therefore, my program solely displays the PLY models as texture-less point clouds.

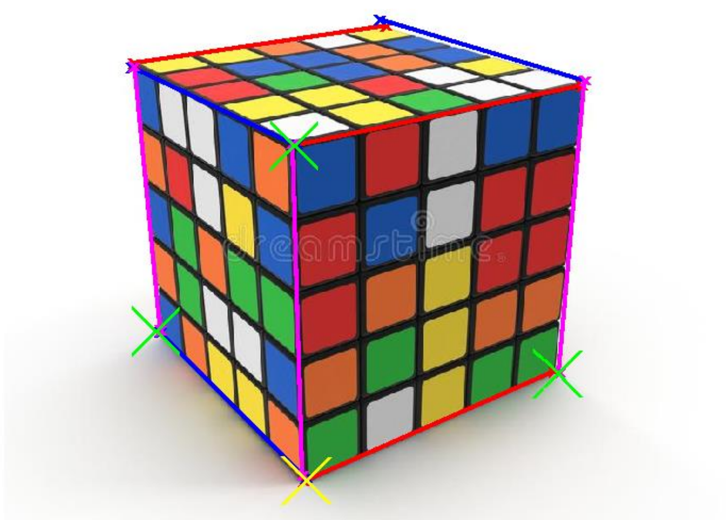
With more time I would have loved to dig further into the PLY format to fix these issues I had with texturing. Another approach could have also been to interpolate RGB pixels from the textures when creating the PLY file, enabling me to build a denser point cloud without a large amount of manual point selection. I also would have loved to work on multi-image reconstruction, however, I found this to be quite difficult and did not continue for sake of time.

Results

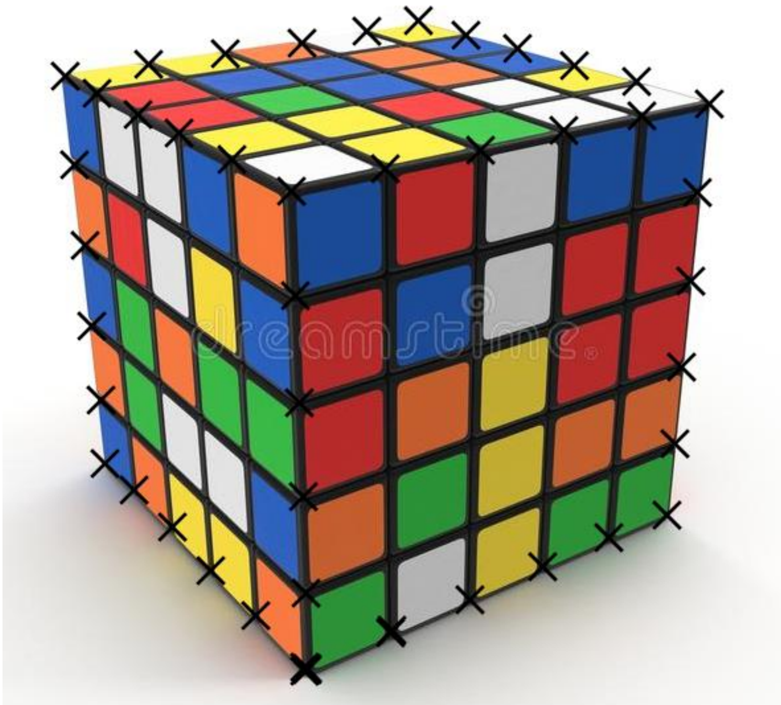
To demonstrate the performance of my implementation I began with a simple model of a Rubix cube, which provided easy unit lengths.

The below figure visualizes the vanishing line and reference point annotations.

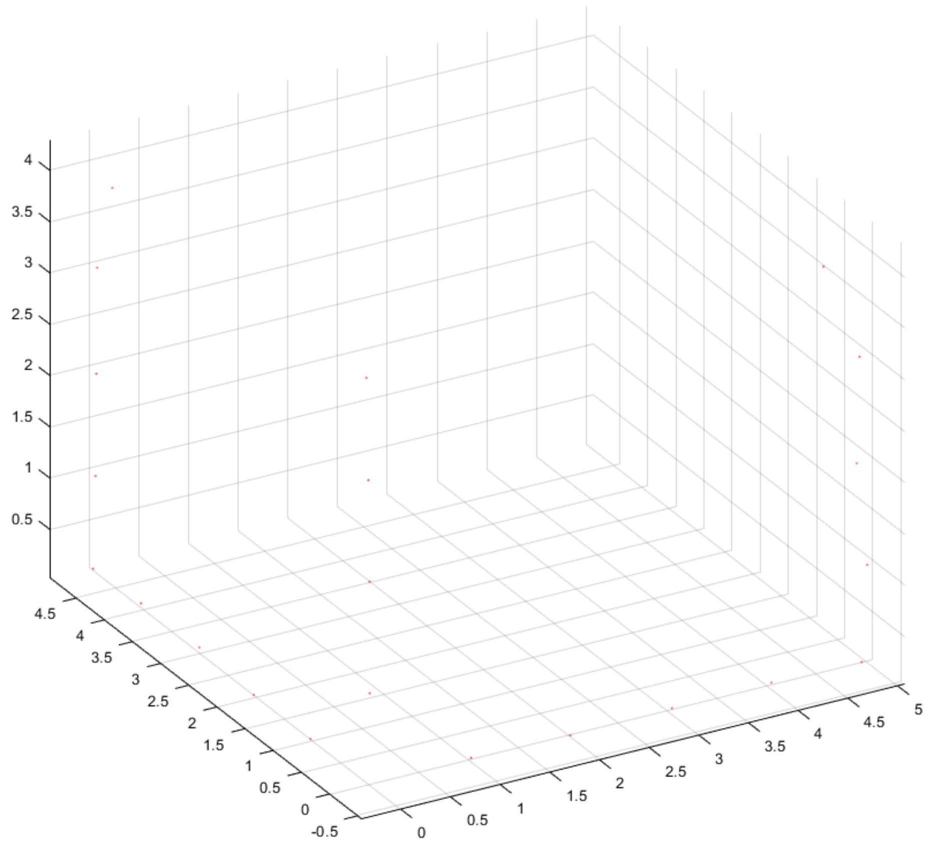
Origin (Yellow) Ref points (Green) X lines (Red) Y lines (Blue) Z lines (Pink)



The below figure visualizes the selected interest points.

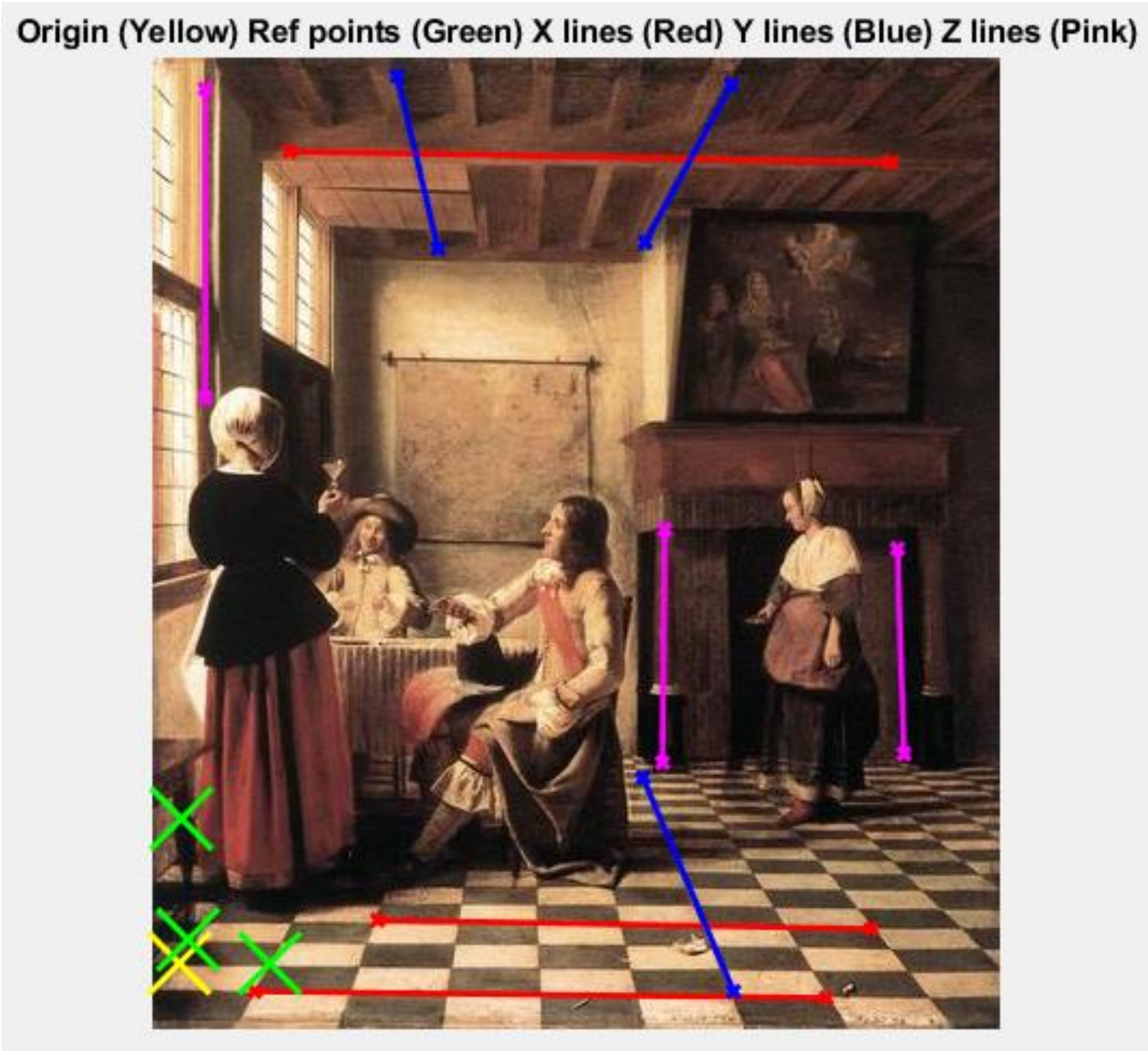


Below is the non-textured PLY model. Note that each of the selected points from above are plotted in red. Zooming in may make viewing the coordinates easier.

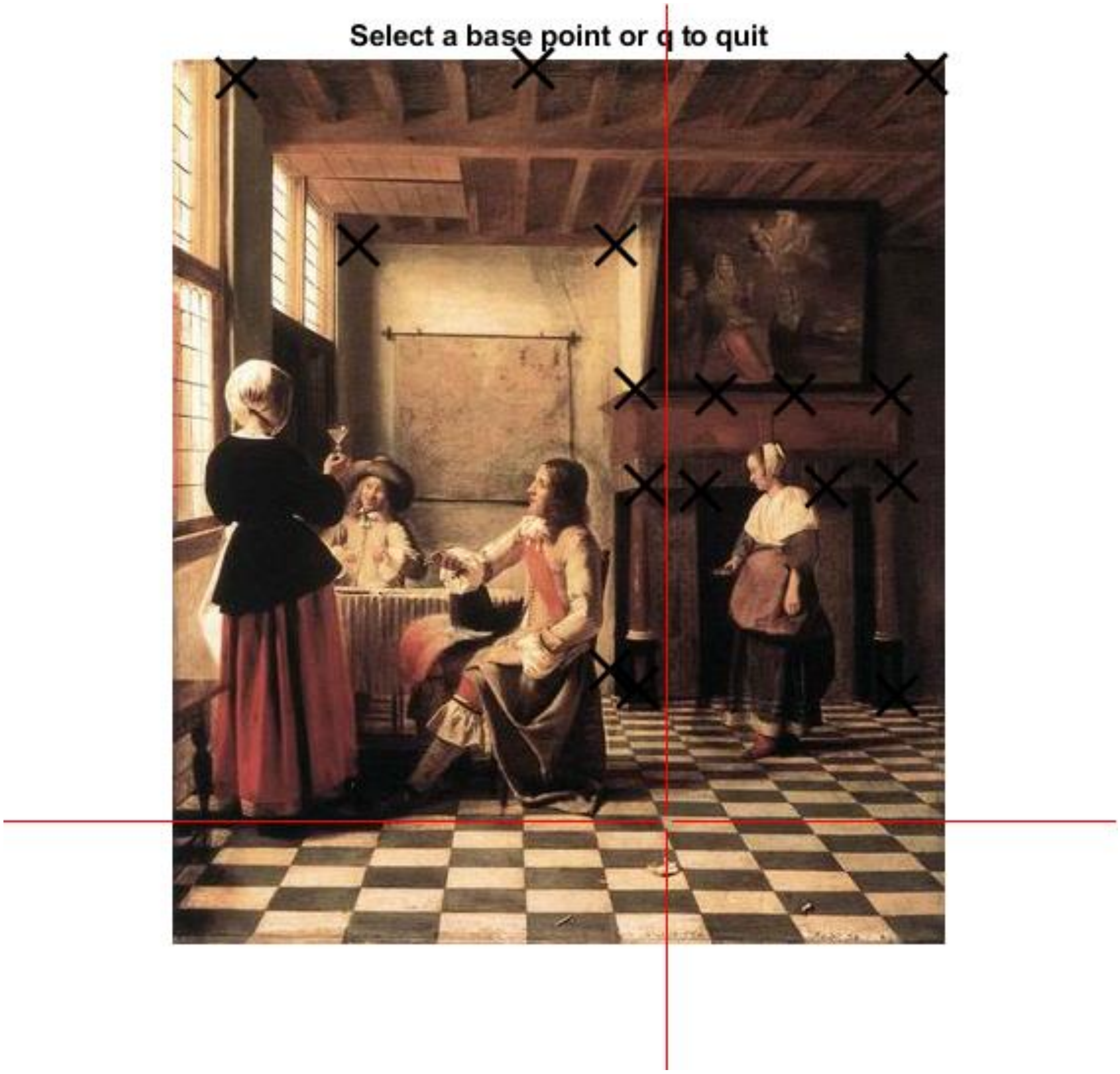


Painting example

Annotated Painting:

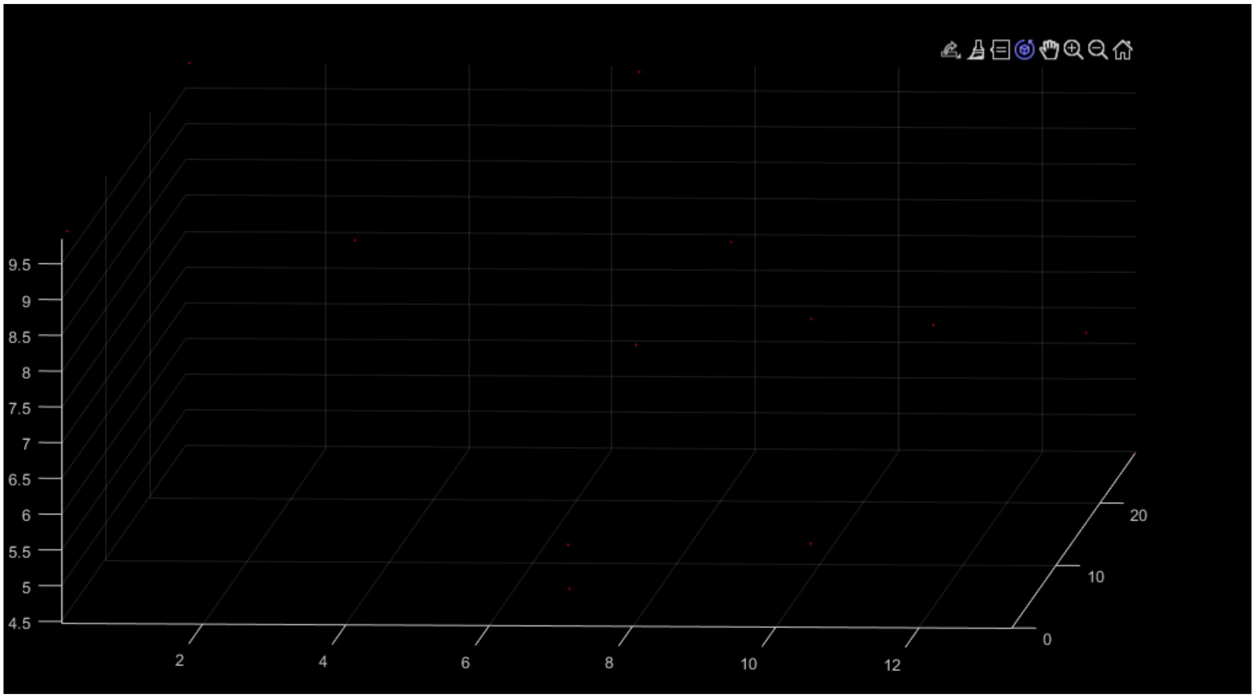


Selected points:

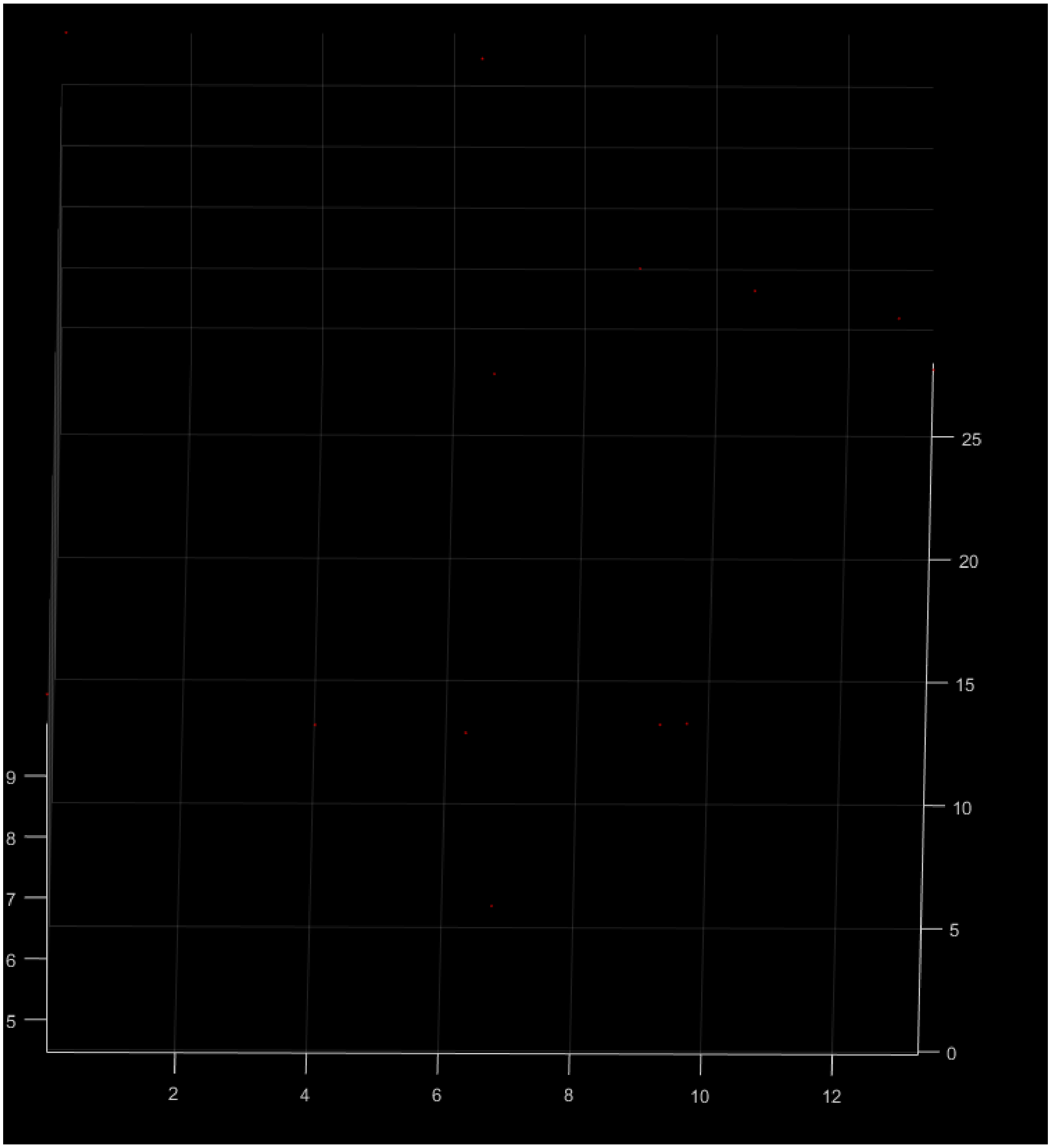


PLY:

I noticed that this model is significantly skewed, most likely due to inaccurate unit lengths and scale factors. It is hard to tell due to the lack of textures, but the points on the roof can be identified, as well as the lines above the fireplace.



This a view from overhead


9
8
7
6
5
2
4
6
8
10
12
0
5
10
15
20
25