Point Cloud Processing

EE5532 Module 6 Assignment

Matthew Spencer

04/21/2021

Contents

Introduction

As point cloud processing plays a larger and larger role in today's autonomous industry and robotics research development, processing of LiDAR and other point-cloud based formats are becoming more and more prevalent. This module explores some basic 3D point cloud processing techniques using MATLAB tools and segmentation. Although the field for innovation is quite deep, this paper will only be starting to scratch the surface of what is capable when processing point cloud data. For example, using LiDAR data in conjunction with a predefined formatting syntax to connect the spatial relationship between pixels to create polygons is a popular way to create fully realized 3D mesh environments, such as with the surveyor centric point cloud XML format "LandXML" [1].

Part 1 creates a polynomial surface in MATLAB where random gaussian noise is then added to the surface. The Moore-Penrose pseudo-inverse method [2] is utilized on the noisy resulting data to apply a best-fit polynomial with the goal of creating a close estimate of the original constants. The test finishes once the polynomial can no longer resemble its original shape. Part 2 then focuses on importing a .LAS point cloud feature gathered from the open source "OpenTopography" website [3] of downtown Indianapolis, along with observing some rasterization solutions applied towards the Zagreb Cathedral data set located in Wurzburg, Germany. Results and discussion are presented along with the material, followed by some concluding thoughts at the end of the report.

Theory

The generated polynomial is expected to be able to retain its shape while under the influence of a low amount of gaussian noise. Due to the linear nature of the estimation process, estimation accuracy should start to deteriorate rather quickly once enough distortion from the noise is applied. 3D point cloud data in part 2 should be able to be relocalized to a local zero point using the minimum values within the data set while also applying a small scaling metric to line the X and Y coordinate data 1:1 with the grid.

Materials

- Windows 10 PC
    - 10400 i5 Intel Processor
    - 16 GB Ram
- MATLAB R2020a

Part 1

Estimation

A cubed polynomial [Eq. 1] was created in MATLAB and then displayed using the `mesh` command [Fig. 1]. Parameters a = 0.5, b = 0.2, and c = 1.3 were chosen as the constants for Eq. 1.

$$z = aX^3 + bY^3 + c$$

Equation 1: Cubed Polynomial

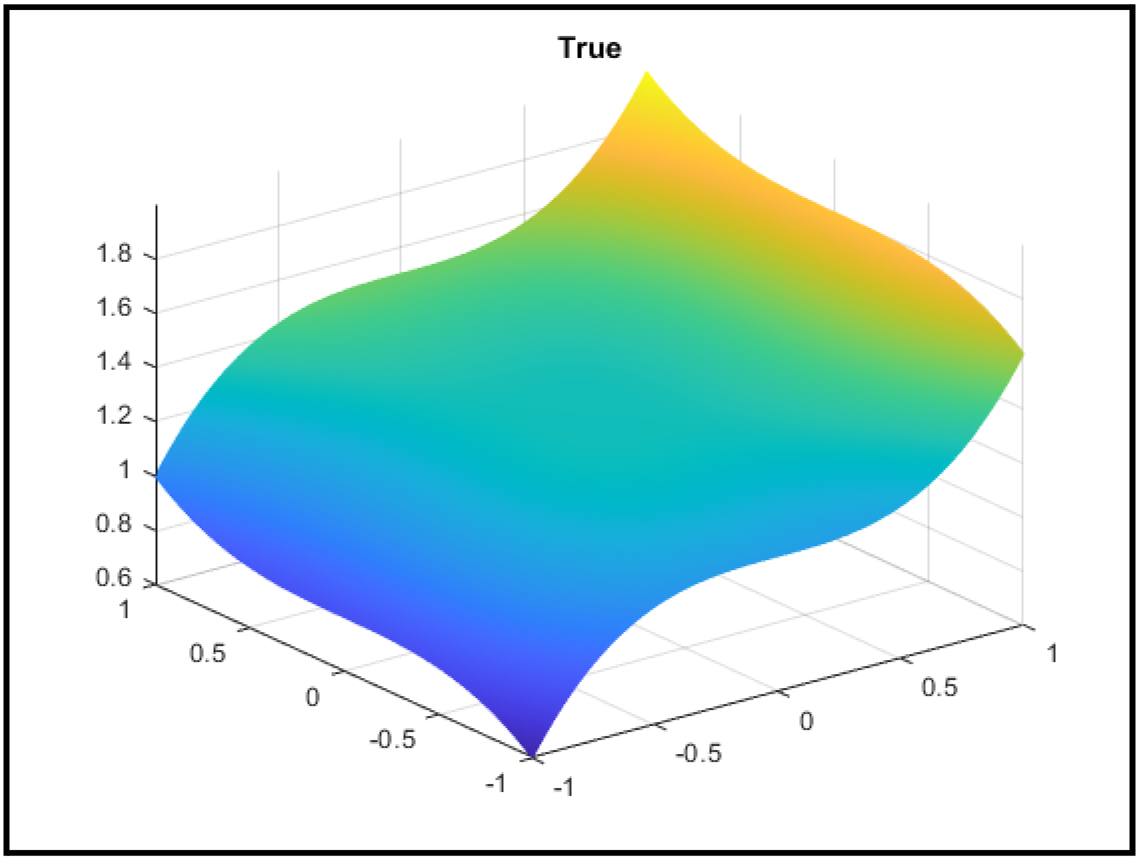

Fig. 1: MATLAB Truth Visualization of Eq. 1

Gaussian noise is then applied to the output seen in Fig. 1, where the script allows the user to input a value `sigma` which adjusts the amount of noise. The next four figures show the noise increasing by setting sigma to the following values: [0.2, 0.3, 0.4, 0.1].

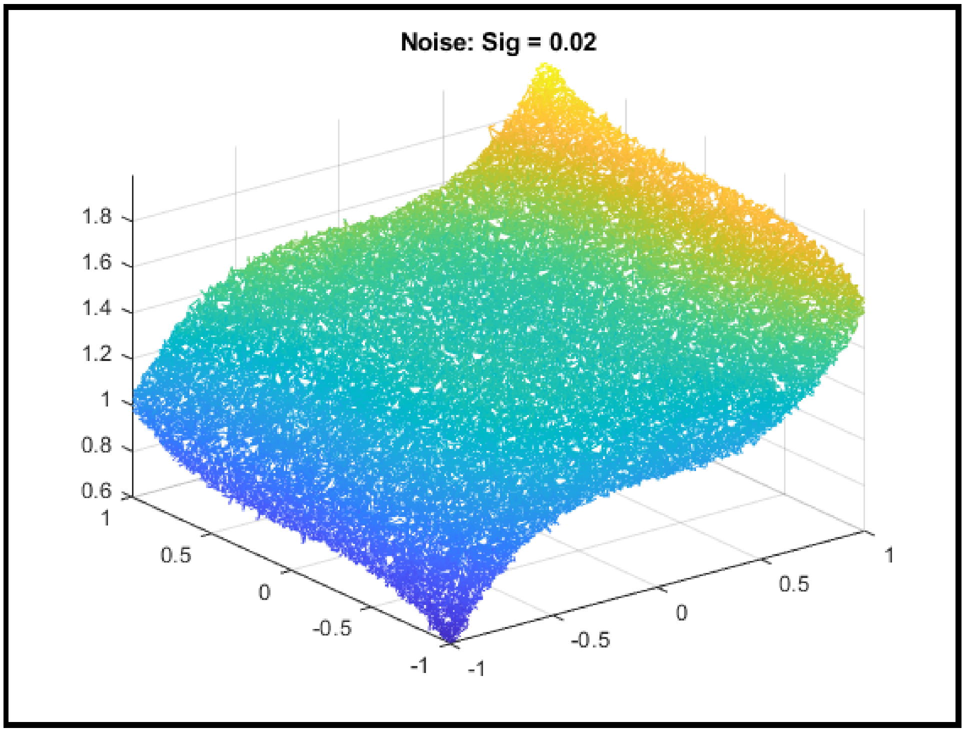

Fig. 2: Noisy Polynomial, sigma = 0.02

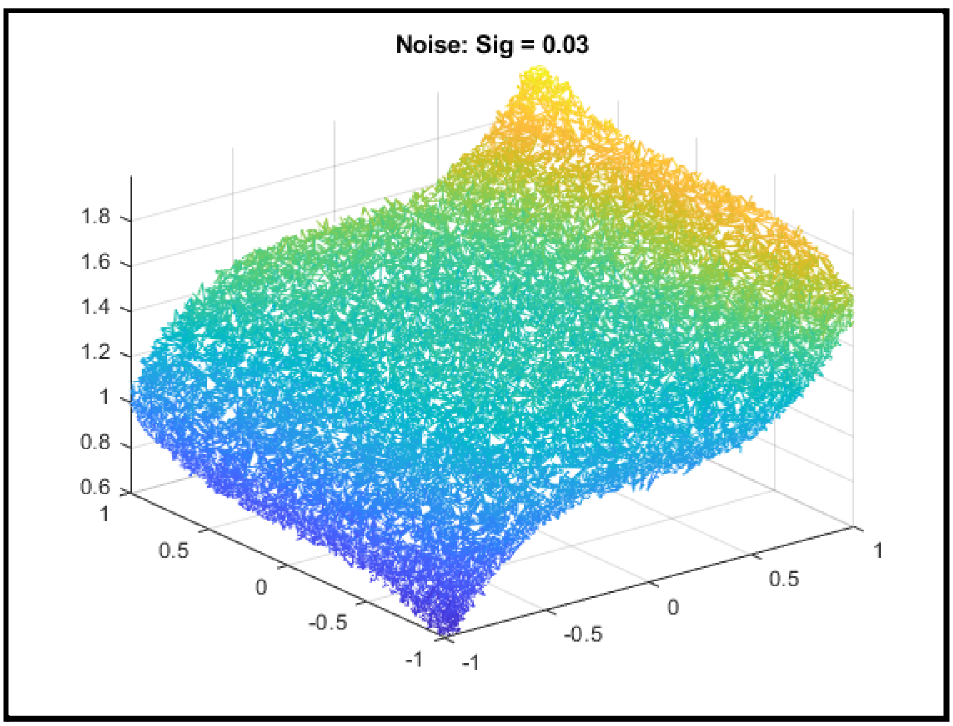

Fig. 3: Noisy Polynomial, sigma = 0.03

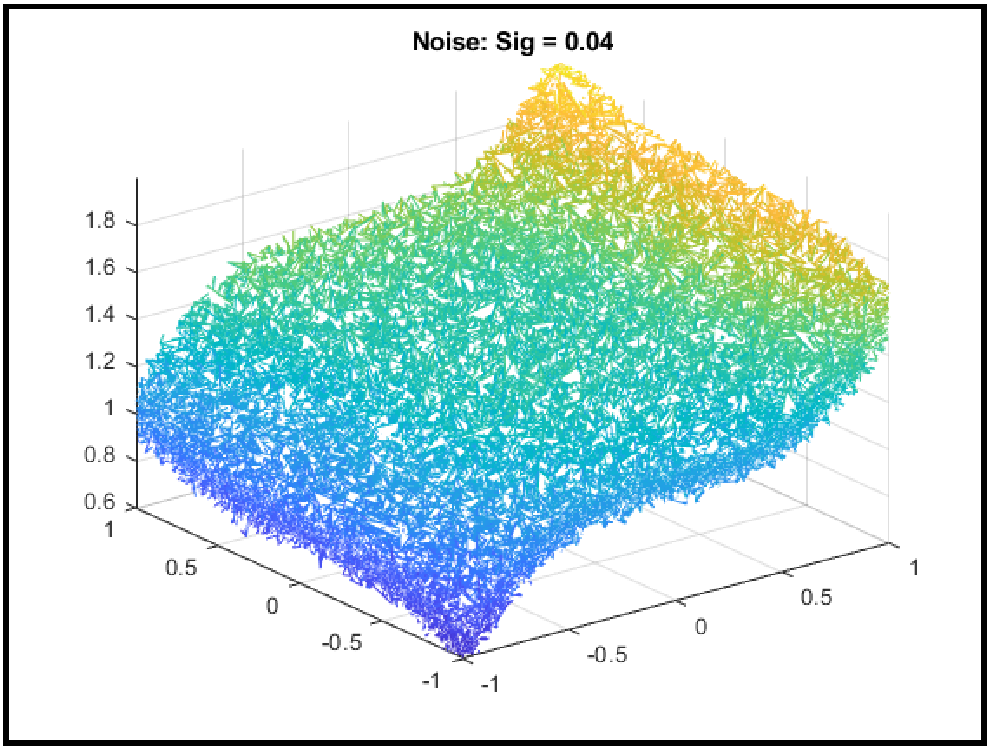

Fig. 4: Noisy Polynomial, sigma = 0.04

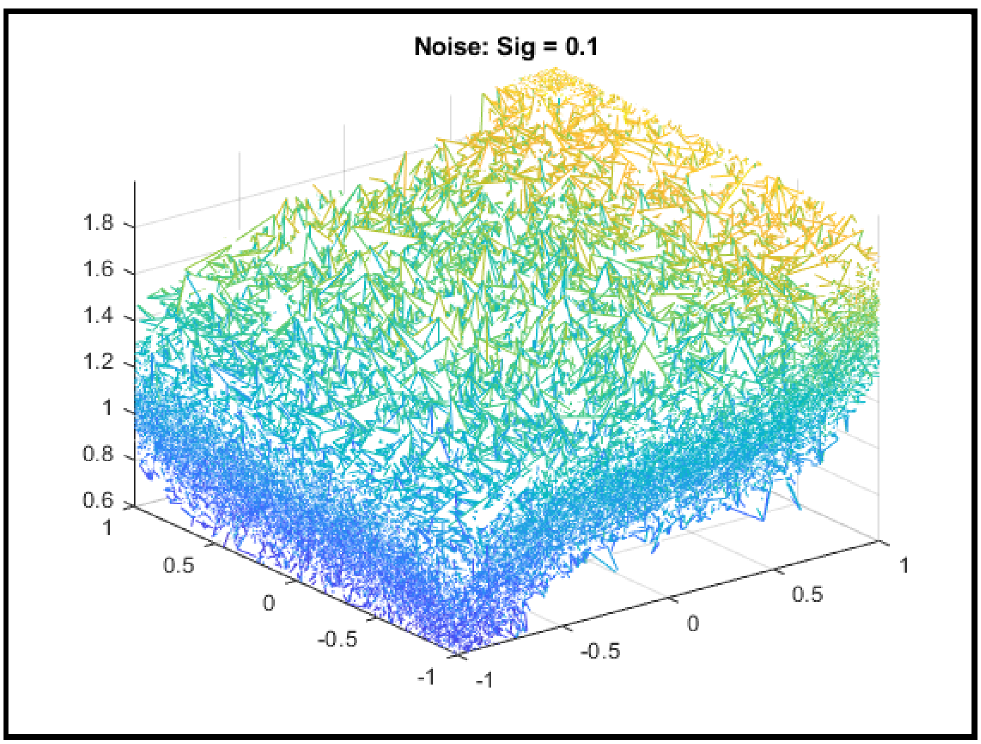

Fig. 5: Noisy Polynomial, sigma = 0.1

The Moore-Penrose pseudo inverse method [Eq. 3] was used to solve the linear algebra forward problem in Eq. 2 using the noisy data seen in the previous figures.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \cdots & \cdots & \cdots \\ x_n & y_n & 1 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \cdots \\ z_n \end{bmatrix}$$

Equation 2: Forward Linear Algebra [4]

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = (A^T * A)^{-1} * z$$

Equation 3: Moore-Penrose Pseudo Inverse

Using Eq. 2 and Eq. 3 on the polynomial we get the following estimated images for the previous noisy inputs.

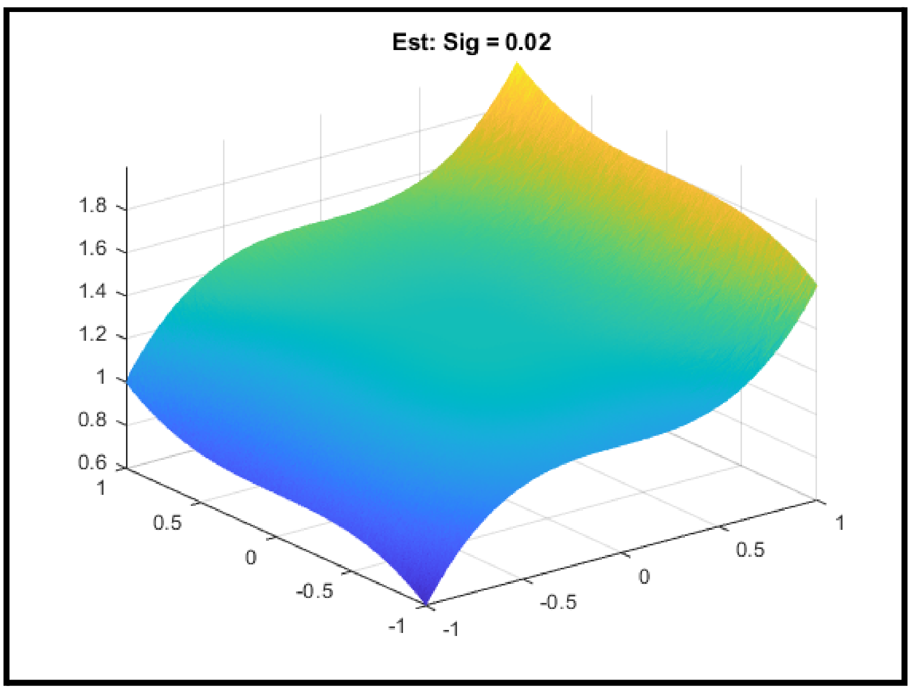

Fig. 6: Estimated Polynomial, sigma = 0.02

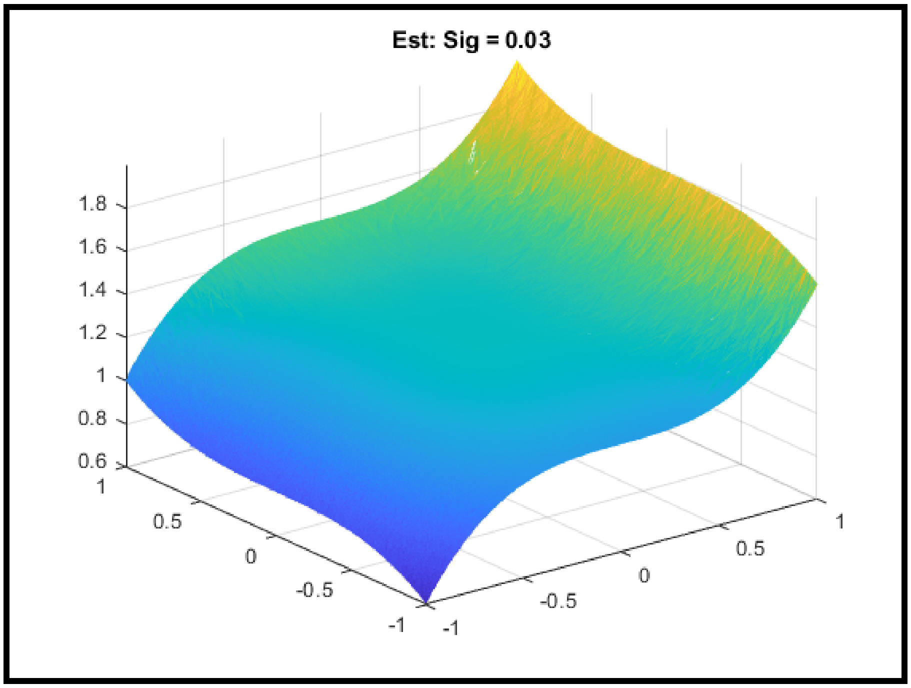

Fig. 7: Estimated Polynomial, sigma = 0.03

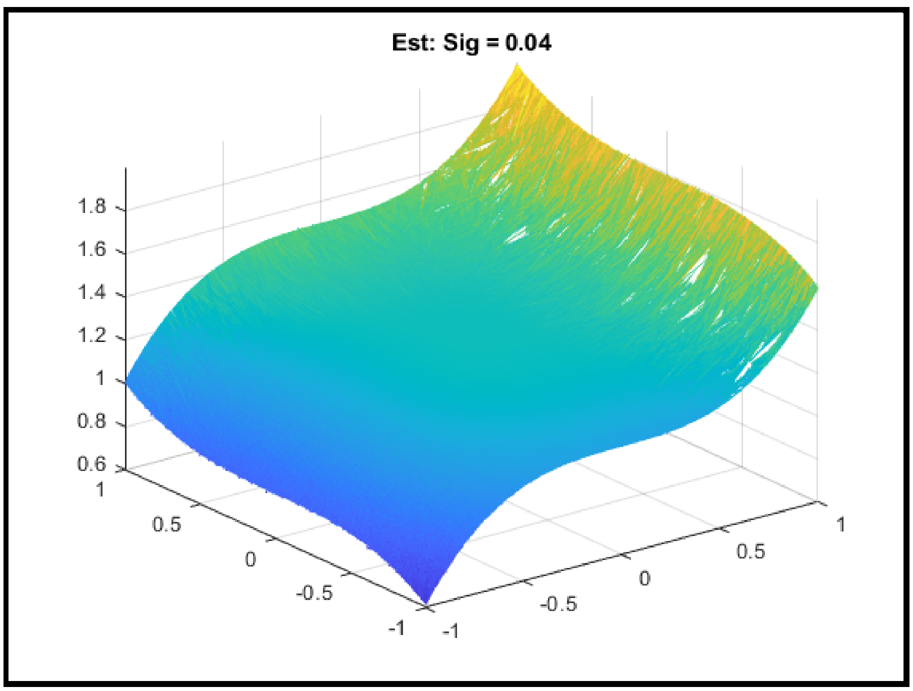

Fig. 8: Estimated Polynomial, sigma = 0.04

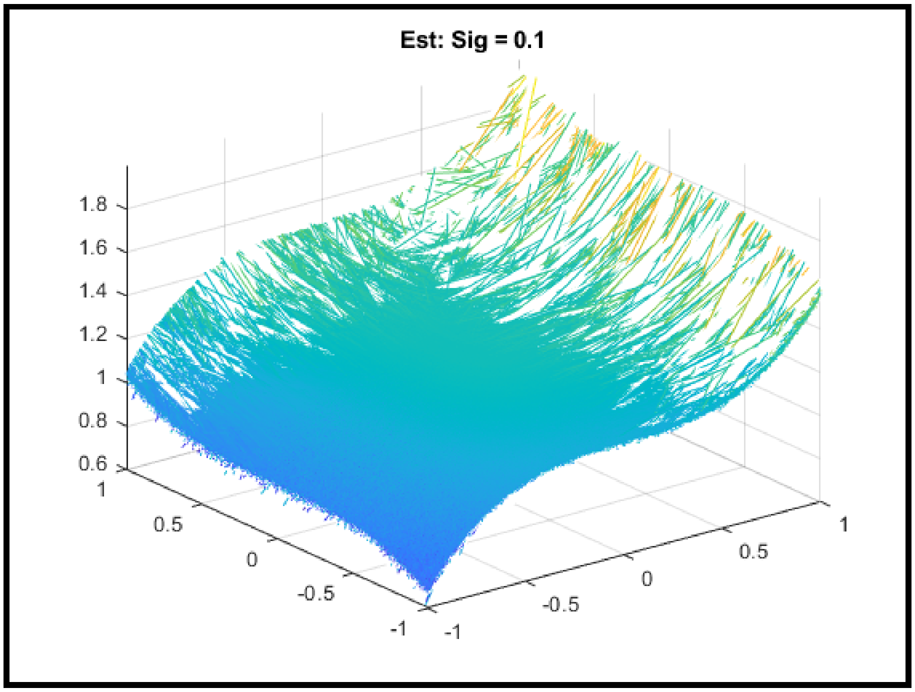

Fig. 9: Estimated Polynomial, sigma = 0.1

Table 1 displays the true constant values compared to the added gaussian noise for each test image.

Constants	True Values	Sigma			
		0.02	0.03	0.04	0.10
a	0.5	0.49674	0.49302	0.48771	0.42715
b	0.2	0.19847	0.19724	0.19484	0.17064
c	1.3	1.2999	1.3	1.3	1.3

Table 1: Estimated Values

Overall, the estimation handles quite well with a low amount of gaussian noise as expected shown by Tab. 1 and Fig. 6 and Fig. 7. A little bit of distortion is noticed on the upward slope of Fig. 8, but the estimation is still quite close. If further processing was done on the surface, the gaps in the mesh could potentially be repaired. The higher noise with a sigma value of 0.1 was shown to have a very negative impact on the estimation seen in Fig. 9, which is also reflected in the estimated a and b values shown in Tab. 1. What is surprising is the continued accuracy of the third estimated constant c.

Part 2

Localization

The first test done to an externally loaded point cloud map was to localize the coordinate system of the imported LiDAR map to a local (X, Y) zero point. The point cloud data being tested was gathered from the open-source point cloud data of a portion downtown Indianapolis, which was provided by the state of Indiana. This point cloud .LAS data file was loaded into MATLAB using the LiDAR imaging toolbox. The final result can be seen below in Fig. 10.

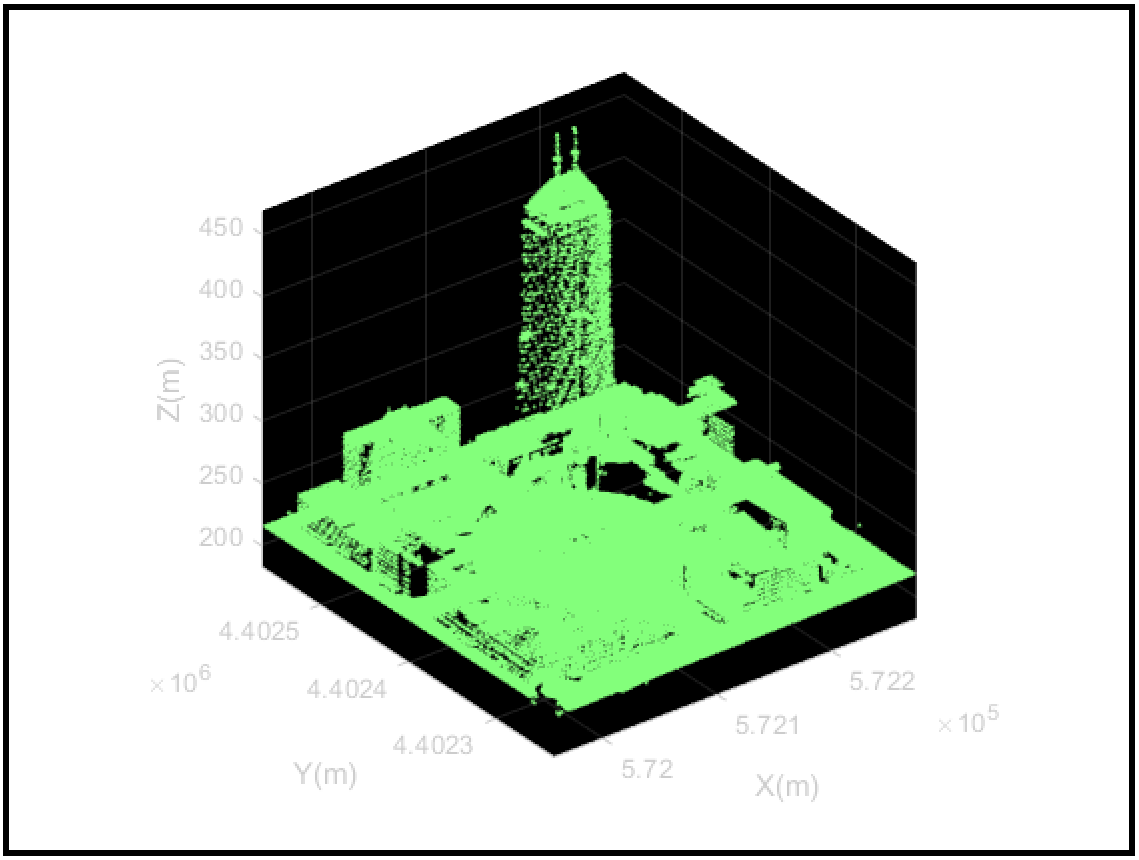

Fig. 10: Downtown Indianapolis .LAS file

In order to relocalize the point cloud data, the scaling was changed by a factor of 10 in order to create an integer grid instead of the decimated grid seen in Fig. 10. The minimum x, y, and z values of the dataset were then subtracted from each point and then re-initialized within a different matrix created by MATLAB. The final processed data was then rasterized using the `mesh` visualization tool instead of using the point cloud visualization tool [Fig. 11].

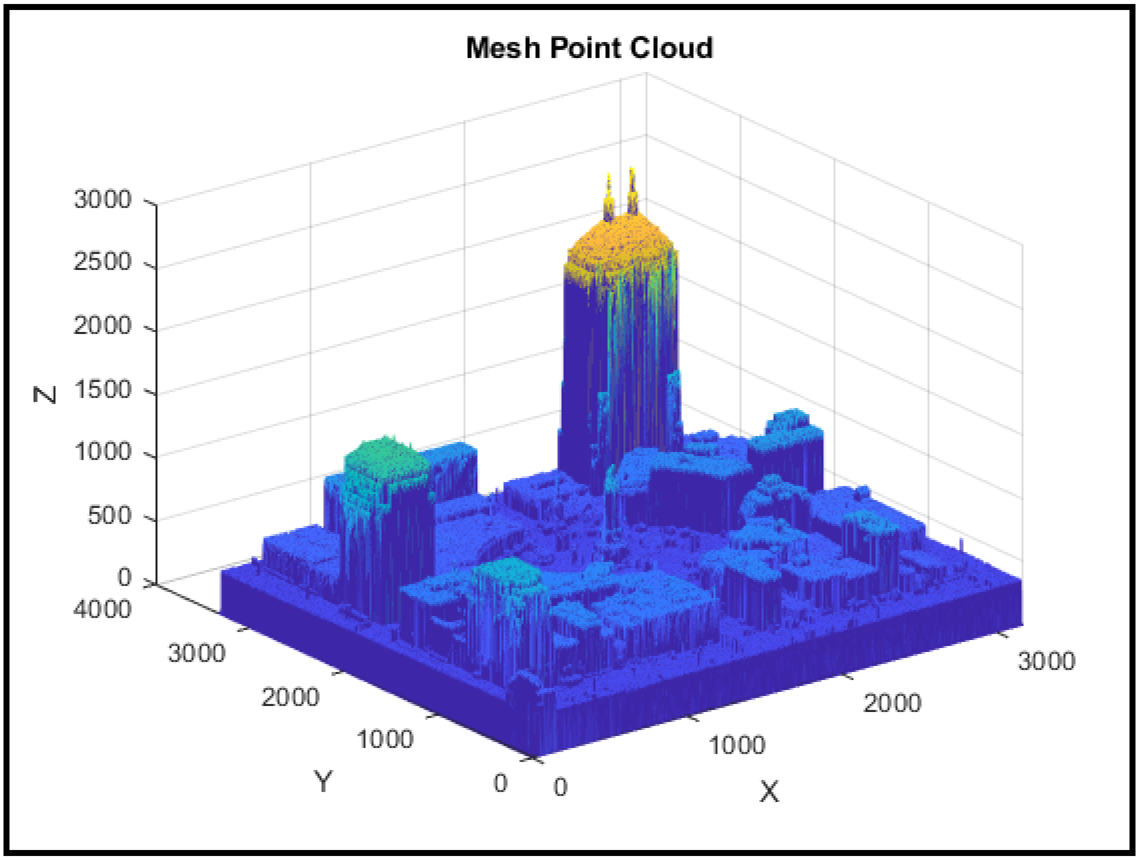

Fig. 11: Localized Point Cloud Mesh

Using the results from Fig. 11, a make-shift point cloud height-map was able to be reconstructed along with a grayscale point-cloud heightmap. Both can be seen below in Fig. 12 and Fig. 13.

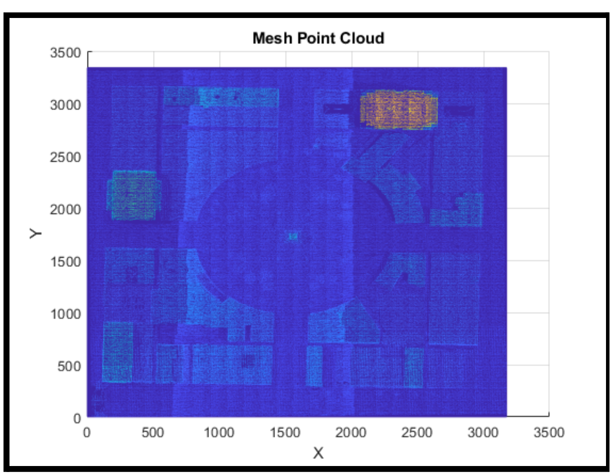

Fig. 12: Mesh Heightmap

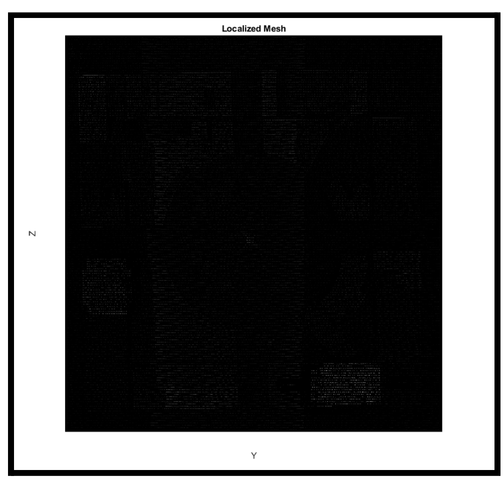

Fig. 13: Grayscale Heightmap

More Rasterization

This next section deals with a few rasterization techniques using the provided MATLAB code on Canvas [4] and the image processing toolbox. A different image was processed from the Zagreb Cathedral example data set than the one used in class. Fig. 14 displays the original LiDAR data.

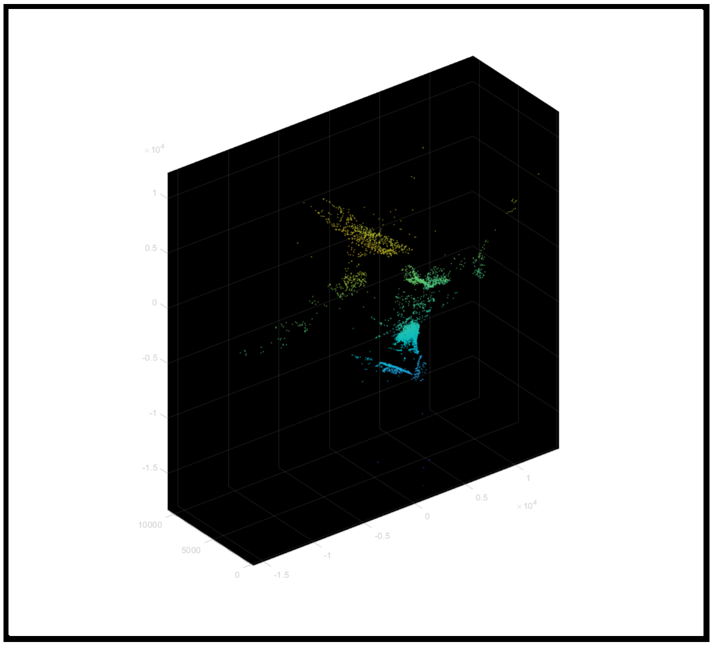

Fig. 14: Cathedral Point Cloud data scan005.3d

Fig. 15 and Fig. 16 display the rasterized range and reflectance data gathered from the original data set.

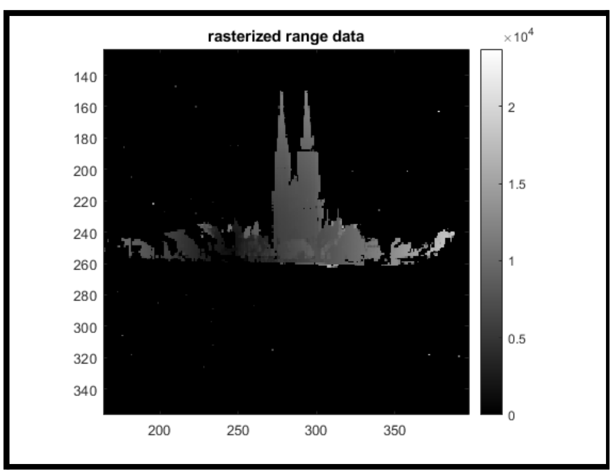

Fig. 15: Range Data

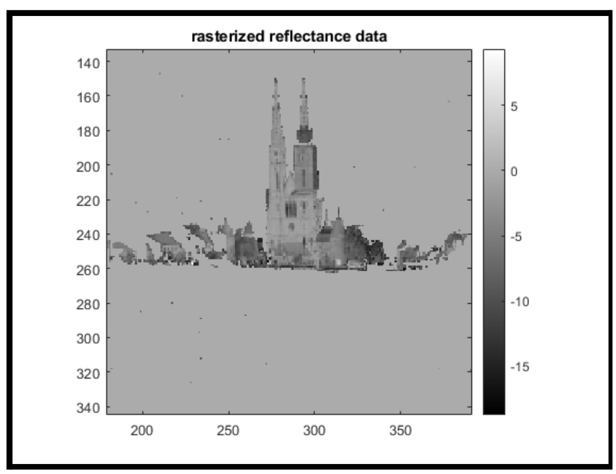

Fig. 16: Reflectance Data

Conclusions

Finding the estimation for a set of points in 3D space worked out surprisingly well for a polynomial shape. Even though the added gaussian noise did distort the shape quite a bit as seen in Fig. 4, the estimation still held together quite good overall. As mentioned prior, going back and doing a post-processing step may be something to look into for the future in order to fill in any missing data that resulted in the final estimation at higher levels of gaussian noise seen prominently in Fig. 9.

Point cloud data taken from downtown Indianapolis was successfully localized and zeroed to an integer coordinate grid for processing and analyzation. Using the quick `mesh` visualization function may be useful in a scenario where a user or company would need a better visual understanding of the physical structures and landmarks of a given area. Other rasterization techniques to find range data and reflectance data within a 3D data set were successful as well, where the reflectance data was able to generate an overlay image of the rasterized data displayed in Fig. 16.

Future research and learning is still on the horizon for students and researchers as it relates to processing 3D point cloud data, as the tools and hardware for it become more optimized and cheaper seemingly with each passing week. By utilizing a suite of powerful software processing tools and applications such as MATLAB, the field of image data processing and its implementations have never been more accessible.

References

1. "LandXML", [Online] Available: http://landxml.org/, [Accessed: Apr. 21, 2021]
2. "Moore-Penrose inverse", *Wikipedia*, Mar. 30, 2021, [Online] Available: https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse, [Accessed: Apr. 21, 2021]
3. "OpenTopography", *High-Resolution Topography Data and Tools*, 2021, [Online] Available: https://opentopography.org/, [Accessed: Apr. 21, 2021]
4. M. Roggemann, "EE5532 Module 6 Lesson 3", *Michigan Technological University*, 2021, [Online] Available: https://mtu.instructure.com/courses/1347065/pages/module-6-lesson-3?module_item_id=17461965#, [Accessed: Apr. 21, 2021]

Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pgrm   : Module 6 Part 1
% Author : Dr. M. Roggemann
% Edited : Matthew Spencer
% Class  : EE5367
% Date   : 04/19/2021
% Desc   : Creates a x^3 polyonomial and adds gaussian noise. The noisy
%          poly is then ran through a least squares estimator to retrieve
%          an approximation based on the noisy data.
% Input  : Sigma = The amount of gaussian noise added to the poly
% Output : Mesh plots of the true, noisy, and est poly.
% Notes  : None.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Module_6_Pt1(sigma)
%------------------------------------------------------------ Initialization
    % Clear all images and cmd window
    close all;
    clc;

    % Init grid
    len  = 500;
    xmin = -1;
    xmax = 1;
    ymin = -1;
    ymax = 1;

    dx = (xmax - xmin)/(len-1);
    dy = (ymax - ymin)/(len-1);

    xaxis = xmin:dx:xmax;
    yaxis = ymin:dy:ymax;

    [X,Y] = meshgrid(xaxis,yaxis);

    % Init constants for x^3 poly
    a = 0.5;
    b = 0.2;
    c = 1.3;

    % Calculate Z axis values for 3D poly
    Z = a*(X.^3) + b*(Y.^3) + c;
    zmax = max(max(Z));
    zmin = min(min(Z));

    % Disp true poly
    figure
    mesh(X,Y,Z)
    title('True')
    axis([xmin, xmax, ymin, ymax, zmin, zmax])

```

Appendix 1: Module_6_pt1.m Part 1

```
%---------------------------------------------------------------- Calculations
    %make noisy data
    Z = Z + sigma*randn(len,len);
    X = X + sigma*randn(len,len);
    Y = Y + sigma*randn(len,len);

    figure
    mesh(X,Y,Z)
    title(['Noise: Sig = ', num2str(sigma)])
    axis([xmin, xmax, ymin, ymax, zmin, zmax])

    % Calculate the estimate
    Xvec = reshape(X,[],1);
    Yvec = reshape(Y,[],1);
    Zvec = reshape(Z,[],1);

    A11 = sum(Xvec.^6);
    A12 = sum( (Xvec.^3) .* (Yvec.^3) );
    A13 = sum(Xvec.^3);
    A21 = A12;
    A22 = sum(Yvec.^6);
    A23 = sum(Yvec.^3);
    A31 = A13;
    A32 = A23;
    A33 = length(Xvec);

    Dvec1 = sum((Xvec.^3) .* (Zvec));
    Dvec2 = sum((Yvec.^3) .* Zvec);
    Dvec3 = sum(Zvec);

    A = [A11 A12 A13; A21 A22 A23; A31 A32 A33];

    D= [Dvec1; Dvec2; Dvec3];

    estP = pinv(A)*D;

    Zest = estP(1)*(X.^3) + estP(2)*(Y.^3) + estP(3);

    % Disp the estimate
    figure
    mesh(X,Y,Zest)
    title(['Est: Sig = ', num2str(sigma)])
    axis([xmin, xmax, ymin, ymax, zmin, zmax])
```

Appendix 2: Module_6_pt1.m Part 2

```
%-------------------------------------------------------------------- Output
    % True values compared to estimate values
    ['input value of a = ', num2str(a)]
    ['estimated value of a = ', num2str(estP(1))]
    ['input value of b = ', num2str(b)]
    ['estimated value of b = ', num2str(estP(2))]
    ['input value of c = ', num2str(c)]
    ['estimated value of c = ', num2str(estP(3))]

% EOF ======================================================================
```

Appendix 3: Module_6_pt1.m Part 3

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pgrm   : Module 6 Part 2
% Author : Dr. M. Roggemann & Matthew Spencer
% Class  : EE5367
% Date   : 04/19/2021
% Desc   : Localizes a topological point cloud data set from a .LAS file by
%          subtracting the minimum x,y,z value from each corresponding
%          point and then initializing that data in a new matrix.
%          Rasterization taken from class provided code is then used on a
%          different portion of the Zagreb Cathedral to view the range and
%          reflectance data.
% Input  : None.
% Output : Localized downtown Indianapolis point cloud and rasterized range
%          and reflectance data of the Zagreb Cathedral.
% Notes  : None.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Module_6_Pt2()
%------------------------------------------------------------ Initialization
    % Clear all images and cmd window
    close all;
    clc;
    format longG

    % Load image
    lasReader = lasFileReader('city.laz');
    ptCloud = readPointCloud(lasReader);

    % Relocate point cloud data to 0,0
    xmin = ptCloud.XLimits(1) * 10;
    xmax = int64(ptCloud.XLimits(2) * 10 - xmin)+1;
    ymin = ptCloud.YLimits(1) * 10;
    ymax = int64(ptCloud.YLimits(2) * 10 - ymin)+1;
    zmin = ptCloud.ZLimits(1) * 10;
    zmax = int64(ptCloud.ZLimits(2) * 10 - zmin)+1;
    zavg = 0;
    len  = ptCloud.Count;
    newCloud = zeros(ymax,xmax);
    X = zeros(len,1);
    Y = zeros(len,1);
    Z = zeros(len,1);

    % Create zeroed point cloud for processing
    for c = 1:len
        pt = ptCloud.Location(c,:);
        y  = int64((pt(1) * 10) - xmin)+1;
        x  = int64((pt(2) * 10) - ymin)+1;
        z  = int64((pt(3) * 10) - zmin)+1;
        X(c) = pt(2);
        Y(c) = pt(1);
        Z(c) = pt(3);
        zavg = zavg + z;
        newCloud(x,y) = z;
    end

```

Appendix 4: Module_6_pt2.m Part 1

```
% Show point cloud
figure
pcshow(ptCloud)
xlabel('X(m)')
ylabel('Y(m)')
zlabel('Z(m)')
title('Point Cloud')

% Show mesh cloud
figure
mesh(newCloud)
xlabel('X')
ylabel('Y')
zlabel('Z')
title('Mesh Point Cloud')

% Show
jumpImg = mat2gray(newCloud);
jumpImg = imresize(jumpImg, 0.8);
figure
imshow(jumpImg)
xlabel('Y')
ylabel('Z')
title('Heightmap')

% ---------------------------------------------------------------- Cathedral
fid = fopen('zagreb_cathedral/scan005.3d');
formatspec = '%f %f %f %f';

sizeA = [4 Inf];
A = fscanf(fid,formatspec,sizeA);

ptcloudX = A(1,:);
ptcloudY = A(2,:);
ptcloudZ = A(3,:);
refl = A(4,:);
N = length(ptcloudX);

stepl = 1000;
scatter3(ptcloudX(1:stepl:N),ptcloudY(1:stepl:N),ptcloudZ(1:stepl:N),'.')

fid = fopen('zagreb_cathedral/scan005.3d');

formatspec = '%f %f %f %f';

%rows 1 through 3 contain x,y,z data, row 4 contains reflectance data
sizeA = [4 Inf];
A1 = fscanf(fid,formatspec,sizeA);
```

Appendix 5: Module_6_pt2.m Part 2

```
    ptcloudX1 = A1(1,:);
    ptcloudY1 = A1(2,:);
    ptcloudZ1 = A1(3,:);
    refl1 = A1(4,:);
    N1 = length(ptcloudX1);

    figure;
    scatter3(ptcloudX1(1:step1:N1),ptcloudY1(1:step1:N1),ptcloudZ1(1:step1:N1),'.')
    hold
    scatter3(ptcloudX(1:step1:N),ptcloudY(1:step1:N),ptcloudZ(1:step1:N),'.','r')

    %concatenate, make on big point cloud
    Xnew(1:N) = ptcloudX;
    Xnew((N+1):(N+N1)) = ptcloudX1;
    Ynew(1:N) = ptcloudY;
    Ynew((N+1):(N+N1)) = ptcloudY1;
    Znew(1:N) = ptcloudZ;
    Znew((N+1):(N+N1)) = ptcloudZ1;
    reflnew(1:N) = refl;
    reflnew((N+1):(N+N1)) = refl1;

    interval = 2000;
    data3d(:,1) = Xnew(1:interval:(N+N1))';
    data3d(:,2) = Ynew(1:interval:(N+N1))';
    data3d(:,3) = Znew(1:interval:(N+N1))';

    figure
    pcshow(data3d)

    %experiment with "rasterizing" data
    imglen = 500;
    cen = imglen/2 + 1;
    maxX = max(Xnew);
    minX = min(Xnew);
    maxY = max(Ynew);
    minY = min(Ynew);

    meanX1 = (maxX + minX)/2;
    meanY1 = (maxY + minY)/2;

    %shift data to center it in X and Y
    Xnew = Xnew - meanX1;
    Ynew = Ynew - meanY1;

    dx = (maxX - minX)/(imglen - 1);
    dy = (maxY - minY)/(imglen - 1);
```

Appendix 6: Module_6_pt2.m Part 3

```
    reflectance_data = zeros(imglen,imglen);
    range_data = zeros(imglen,imglen);
    counter = ones(imglen,imglen);

    for m = 1:length(Xnew)
        xbin = round(Xnew(m) / dx);
        ybin = round(Ynew(m) / dy);

        c = xbin + cen;
        r = cen - ybin;

        if r == 0
            r = 1;
        end
        if r > imglen
            r = imglen;
        end
        if c == 0
            c = 1;
        end
        if c > imglen
            c = imglen;
        end

        range = sqrt(Xnew(m)^2 + Ynew(m)^2 + Znew(m)^2);

        range_data(r,c) = range_data(r,c) + range;

        reflectance_data(r,c) = reflectance_data(r,c) + reflnew(m);

        counter(r,c) = counter(r,c) + 1;

    end

    rast_range_data = range_data ./(counter+1);
    rast_refl_data = reflectance_data ./ (counter+1);

    figure
    imagesc(rast_range_data)
    axis('image')
    colormap(gray(256))
    colorbar('EastOutside')
    title('rasterized range data')
    figure
    imagesc(rast_refl_data)
    axis('image')
    colormap(gray(256))
    colorbar('EastOutside')
    title('rasterized reflectance data')


% EOF ========================================================================
```

Appendix 7: Module_6_pt2.m Part 4