Ultrasound Imaging --- From 1D to 3D

Group name: ZYyyds

Group members: 赵衍智，程子镔，姚立方

Division of labor:

1D~2D, 2D~3D (stack-by-stack) 赵衍智

2D~3D(forward mapping) 程子镔

2D~3D(inverse mapping) 姚立方

Typesetting of the report 赵衍智

Typesetting of the PPT 程子镔

Step 1: From radiofrequency ultrasound signals to an image

Objective: using data of signals to construct a 2D image

Method:

1. Loading RF data, find the spacing interval in x- and z-direction respectively
2. Denoising
3. Adapting signal peaks
4. Envelope
5. Resize
6. Intensity mapping
7. Image processing

a) Loading RF data, find the spacing interval in x- and z-direction respectively

Process

(i) Spacing interval in x-direction equals to the distance between elements of the transducer, which is

$$space_{\_x} = 0.78125 \times 10^{-3} m.$$

(ii) According to the formular

$$\begin{aligned} &Pixel\ sizez \\ &= (sound\ of\ speed \times sampling\ interval)/2 \\ &= sound\ of\ speed/(2 \times sampling\ frequency) \end{aligned}$$

the space interval in z-direction is

$$space_{\_z} = \frac{1500}{2\times(20\times10^{6})} m.$$

b) Denoising

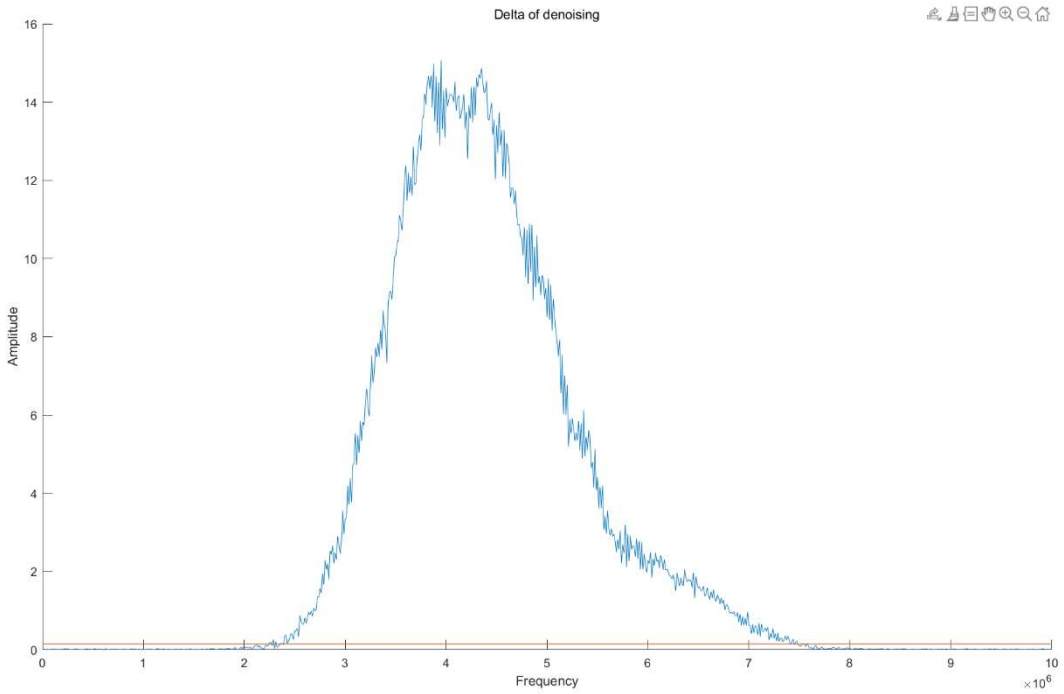
Functions applied: ***abs(), fft(), butter(), filter()***

Process:

Using the Butterworth bandpass filter to denoise the signals. For the 192 signals, we denoise these signals one by one:

(i) Traverse 192 signals from the beginning to the end. For any signal among 192 signals, apply *'fft'* on it to get the relation of frequency(let the frequency to be positive) and amplitude. There are 801 amplitude points after *'fft'*, which are stored in matrix *'P1'*. (*'P1'* is changing when processing)

(ii) For any signal among 192 signals, find the maximum amplitude, labeled *'max_magnitude'*, define $delta = \frac{\max\_magnitude}{100}$ as the threshold of the amplitude corresponding to this signal. For any two signals, *'delta'* are not the same.



For example, the blue curve is the relation of frequency and amplitude corresponding to a signal, the orange straight line is the *'delta'* of this signal.

(iii) For any signal among 192 signals, traverse all the 801 amplitude points from the beginning to the end (the frequency increases as traversing). If $amplitude > delta$, stop the traverse, and define *'low_cut'* as the frequency corresponding to this amplitude point. Then traverse all the 801 amplitude points from the end to the beginning (the frequency decreases as traversing). If $amplitude > delta$, stop the traverse, and define *'high_cut'* as the frequency corresponding to this amplitude point.

(iv) Use *'low_cut'* and *'high_cut'* as the cutoff frequencies of the Butterworth bandpass filter. For each signal among 192 signals, *'low_cut'* and *'high_cut'* are unique, so for any two signals, the Butterworth bandpass filter are not the same.

(v) Store the 192 signals after denoising into matrix *'rf_file_1'*.

Figure:

c) Adapting signal peaks

Functions applied: *max(), min(), mean(), normalize()*

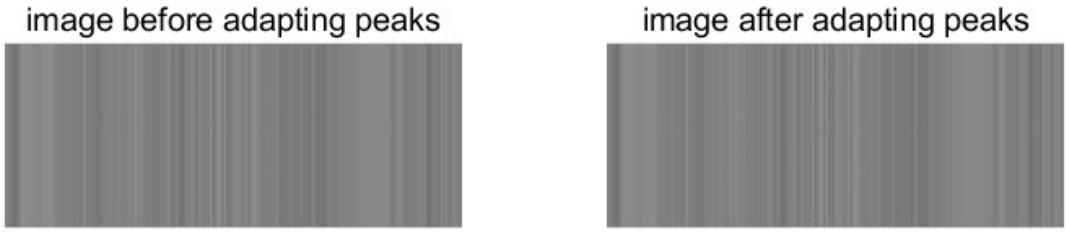
Process:

(i) Traverse the 192 signals in matrix 'rf_file_1', for any signal among 192 signals, there are 1600 data points. Store the maximum data point and the minimum data point into matrix *'max_matrix'* and *'min_matrix'* respectively.

(ii) Define *'threshold1'* to be the average of elements in *'max_matrix'*; *'threshold2'* to be the average of elements in *'min_matrix'*.

(iii) Applying the function *'normalize()'* to normalize abnormal signals. Traverse 192 signals, if the data points of the signal are all between *'threshold1'* and *'threshold2'*, then we do not change the signal. If there are data points that are beyond *'threshold1'* or *'threshold2'*, then we use *'normalize()'* to adapt the signal into the range [*threshold1,threshold2*]. Then store the signals after adaping into the matrix *'rf_file_2'*.

Figure:

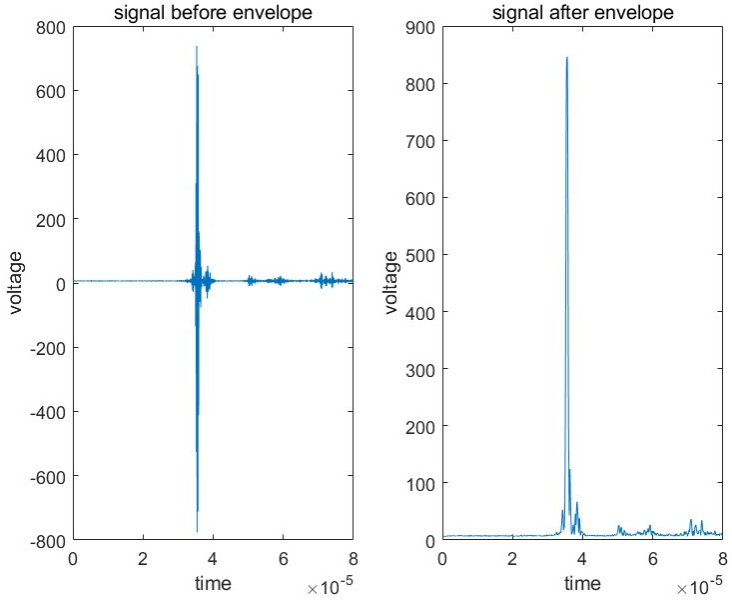


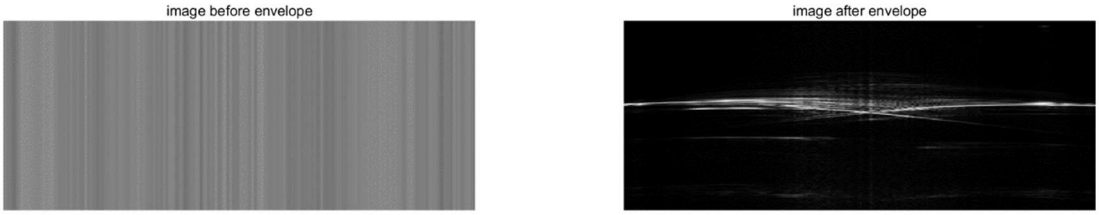
The process 'Adapting signal peaks' does not change the image, because when we imshow the image, we need to transform the matrix 'rf_file_2' into a 8-bit grey image matrix. Function *'normalize()'* change all the data points into range [*threshold1,threshold2*] in the meanwhile, and the scale is isometric. So when doing the transformation, the 8-bit grey image matrices are the same for the two circumstances, has nothing to do with adapting peaks. But the process 'Adapting signal peaks' has a great contribution to the latter process 'Envelope' because it has normalize the abnormal signals which have very high peaks, so that it can increase the contrast ratio of the image.

d) Envelope

Functions applied: *envelope()*

Process: use function 'envelope' to find the envelope of each signal. Then store the envelopes of 192 signals into the matrix 'envelope_matrix'.





e) Resize

Functions applied: ***floor(), imresize()***

Process:

(i) For the spatial resolution for the image is $0.25mm \times 0.25mm$, then the column numbers of the 8-bit grey image matrix is:

$$x_{\_width} = [\frac{192 \times space\_x}{0.25 \times 10^{-3}}]$$

(ii) The row numbers of the 8-bit grey image matrix is:

$$z_{\_xidth} = [\frac{1600 \times space\_z}{0.25 \times 10^{-3}}]$$

(iii) Use function *'imresize()'* to resize the matrix *'emvelope_matrix'*.

f) Intensity mapping

Functions applied: ***min(), max(), uint8()***

Process:

(i) Traverse 192 signals in matrix *'envelope_matrix'*. Normalize all the data points in *'envelope_matrix'* into the range of [0,1] and store the data points into matrix *'image_matrix'*.

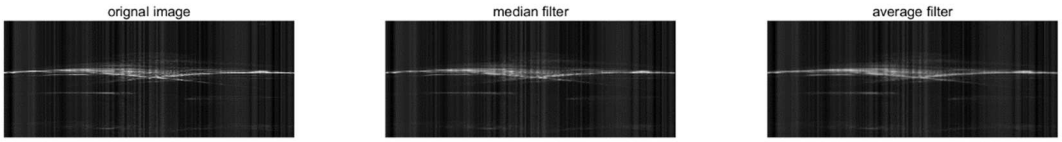
(ii) Use function 'uint8' to transform the *'image_matrix'* into a 8-bit grey image matrix.

g) Image processing

Functions applied: ***filter2(), medfilt2(), fspecial(), imfilter()***

Process:

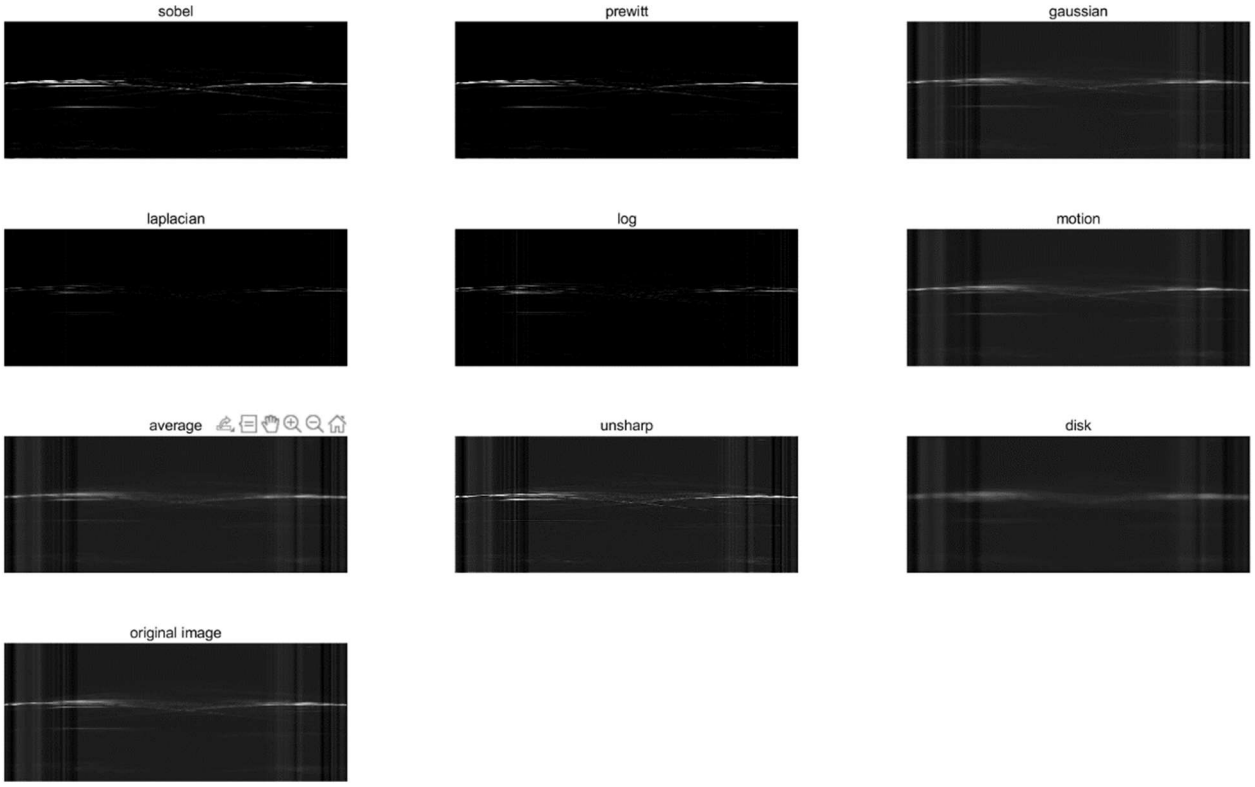
(i) Use a 3×3 median filter and an 3×3 average filter to smooth the image. Then compare the results.



The result of median filter is better. So in the latter process, we use median filter.

(ii) Try different operators to sharpen the image. Using function *'imfilter()'* and operators *'sobel', 'prewitt', 'gaussian', 'laplacian', 'log', 'motion', 'average', 'unsharp', 'disk'* to sharpen the image and compare the results.

Final results:



Discussion:

From the results we find that

(i) median filter is better at smoothing the image, compared with average filter.

(ii) Operators *'sobel'* and *'prewitt'* are better at sharpening the image.

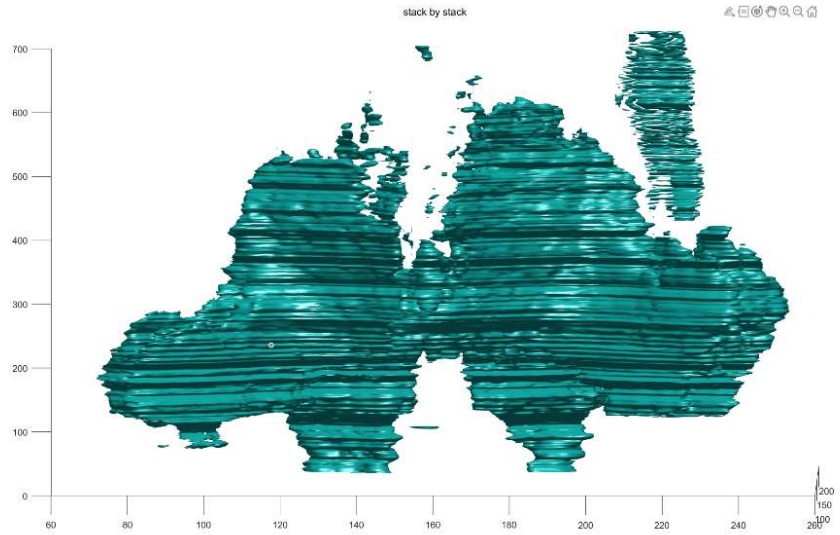
Step 2: 3D image reconstruction by coordinate transform

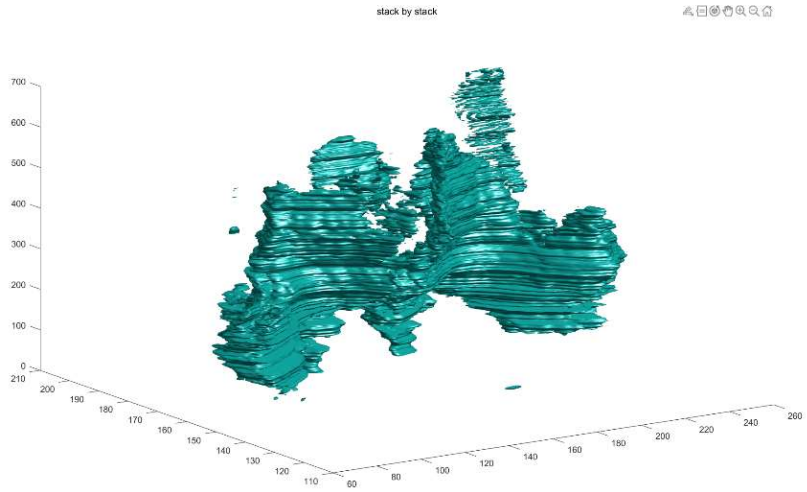
Section1: Reconstruct a 3D volume by putting the 2D FrameData stack by stack and display it

Objective: put the 2D frame data stack by stack to reconstruct a 3D volume.
Method: put the 2D frame data stack by stack to form a 3D matrix.

Functions applied: *isosurface()*
Process:
Put the 2D-matrix in *'framedata'* into a 3D-matrix and show the 3D-matrix using the function *'isosurface()'*

Result:





Discussion: There is no gap in the volume. But the edge of the volume is not smooth. And the precision is quite low. It has ignored the differences of the distance and angle of different frame plains.

Section2: Forward/Inverse Mapping

Objective:

Though the stack-by-stack method is quite easy to achieve, the disadvantage is apparent, in explanation, which is ignoring the differences of the distance and angle of different frame plain. As shown in figures above, the result is very coarse and distorted. In order to solve this problem, it is necessary to use the data of *GpsData.mat* to determine the map relationship of the pixels and voxels more precisely.

Since the coordinates of the four vertexes of each frame is given, then that of each pixel can be calculated by dividing the frame plain in equal parts. A new problem is that the voxel is at the nodes(a node is a point that all coordinates are interger) of a cube grid, while the pixel might not be. The first method is *Forward Mapping Method*, which means for each pixel, we find the nearest voxel and give the value to it; The other method is called *Inverse Mapping Method*, which means for each voxel, we find the nearest frame plain, and then find the nearest pixel.

Subsection1: Forward Mapping Method

Method:

In details, the forward mapping can be summarized in steps:

(i) Rebuild a regular value;

(ii) Find the nearest voxel of each pixel;

(iii) Assign the value of pixel to the voxel;

(iv) Display the 3D matrix by matlab.

To rebuild the regular value, the transformation matrix between gps frame and the world frame should be found first. Let (u, v, w) to be the coordinates in GPS transmitter space, while (x, y, z) to be that in world space. The affine transformation can be expressed then as:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{12} & a_{13} & a_{14} \\ a_{31} & a_{12} & a_{13} & a_{14} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix}$$

If there are four such transformations, the relationship is then

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{12} & a_{13} & a_{14} \\ a_{31} & a_{12} & a_{13} & a_{14} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \\ v_1 & v_2 & v_3 & v_4 \\ w_1 & w_2 & w_3 & w_4 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Thus, after getting the information of the calibration points, Matlab can help to get the transformation matrix:

$$T = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{12} & a_{13} & a_{14} \\ a_{31} & a_{12} & a_{13} & a_{14} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \\ v_1 & v_2 & v_3 & v_4 \\ w_1 & w_2 & w_3 & w_4 \\ 1 & 1 & 1 & 1 \end{pmatrix}^{-1}$$

Then we can do this transform for every GPS data, as shown in fig(1)

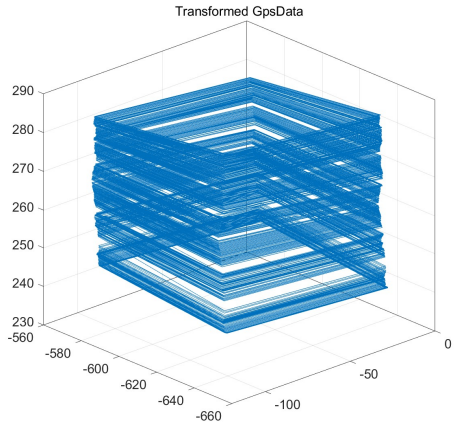


Fig 1: The transformed GPS data in the world space

After that, we can use function *min* and *max* to find the minimum and maximum of the coordinates of the vertexes. Consider a redundancy of one resolution, we can build the volume by them. As shown in fig(2), the red lines represent the bound.

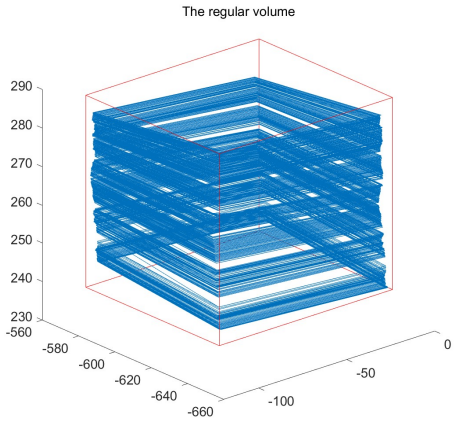
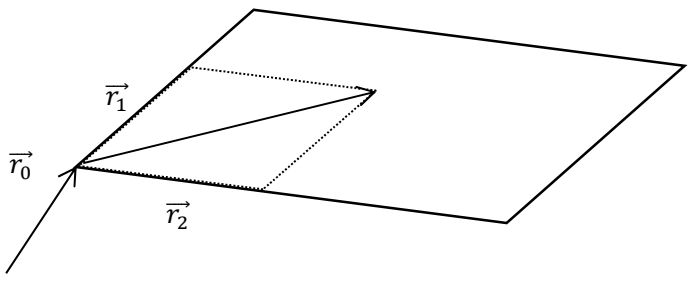


Fig 2: The regular volume

It is known that every point of a plain can be expressed by two linearly independent vectors, which is chosen as the two sides of each frame. In this way, the orientation of the plain makes no difference. As shown below, the position of the pixel is then

$$\overrightarrow{r_{ij}} = \frac{i}{240}\overrightarrow{r_1} + \frac{j}{320}\overrightarrow{r_2} + \overrightarrow{r_0}$$



Finally, to find the nearest voxel, we can change the origin to one of the vertexes of the regular volume, and then zoom the coordinates by $\frac{1}{resolution/mm}$, which make all the voxels at the nodes. Hence, we can use *round* function which return the nearest integer of each component to get the nearest voxel, since for whatever $\alpha = x, y, z$, $(\alpha_{pixel} - \alpha_{nodes})^2 \geq (\alpha_{pixel} - round(\alpha_{pixel}))^2$, the total distance will be smallest as well.

There appears a new problem, which is that different pixels may map to the same voxel. If we ignore this, just assign the last pixel to the voxel, the result is like fig(3)

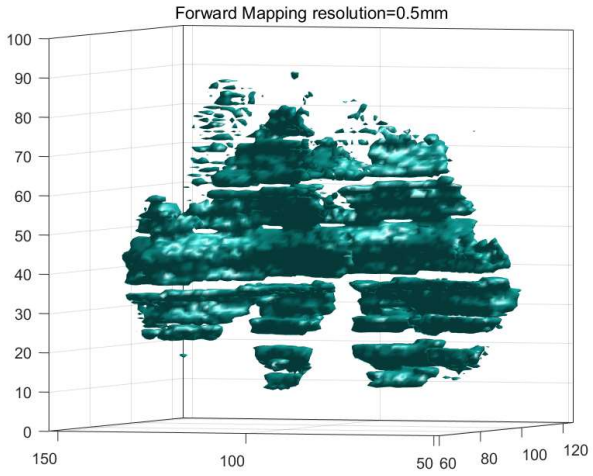
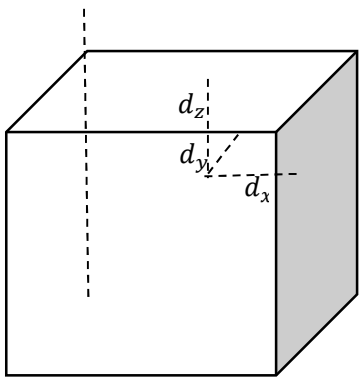


Fig 3: the rough result

It has already shown a much better result than stack-by-stack method. However, the surface is still rough with some bulges. Notice that there are some gaps, they are the results of the loss of some frame data

To solve this problem, weighted average should be considered. The pixels should be ranked by *the level of nearness*, which can be a lot kinds of functions of the distance. Here, trilinear interpolation is chosen. In explanation, if the edge length of the small cube, or the resolution is r, and the vector from pixel to the nearest voxel is (d_x, d_y, d_z), then the weight is $(r - d_x)(r - d_y)(r - d_z)$.



The weight is easy to calculate. Though it is not linear to the 3D distance of the two points, it is largest when the voxel be the nearest. Later in inverse mapping method subsection, it is shown that bilinear interpolation is a good choice for weighted average. The final result is shown in fig(4)

Final results:

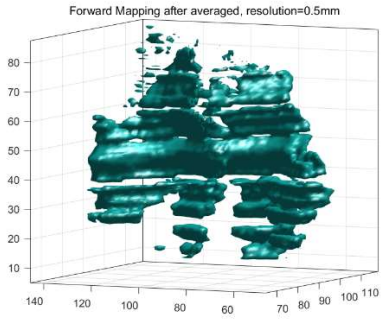
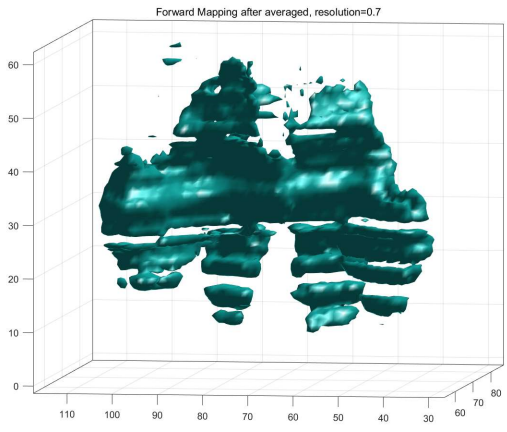
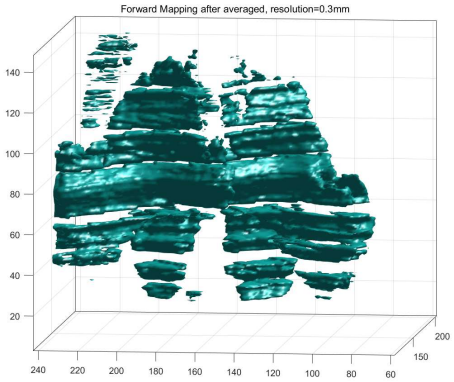


Fig 4: the trilinear interpolated result

The result is smoother now.

Here is a comparison of different resolution.





Subsection2: Inverse Mapping Method

Method:

In details, the forward mapping can be summarized in steps:

(i) Rebuild a regular value;

(ii) Project the voxel onto the nearest frame plane and get the projection points on the plane;

(iii) Assign the value of pixel to the voxel by using the nearest neighbor interpolation method in frame plane;

(iv) Display the 3D matrix by matlab.

first step is the same as forward mapping method.

To find the nearest frame plane of the specific voxel, we need to do the projection. For the first time we projected the voxel onto every frame plane but the amount of calculation is too high. Then we can do a sifting before we calculate the distance. We have found that the z-coordinates of the four vertexes of every frame plane is very close. We write some lines of code and found the biggest difference between the z-coordinates of 2 vertexes of the same frame plane is just 9.1187. So, if the difference between the z-coordinates of a vertex and the voxel is bigger than 10, we do not need to calculate the distance.

Then we can calculate the distance of the voxel and the sifted frame planes. Firstly, we choose a vertex of a frame plane and find the vector $\overrightarrow{p_1}$ from the voxel to the vertex so that we can do the projection easier. Secondly, we calculate the normalized (norm = 1) normal vector $\vec{n}$. And the absolute value of the inner product between $\overrightarrow{p_1}$ and $\vec{n}$ is just the distance d between the voxel and the frame plane.

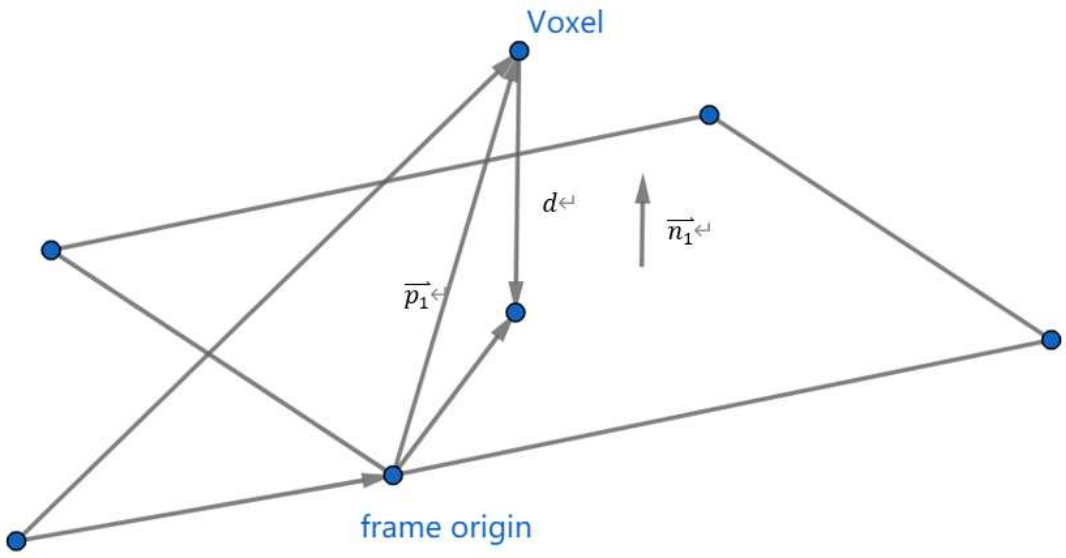


Fig 5

Perform the process for all sifted planes then we can find the smallest distance and the corresponding nearest frame plane.

Base on the descriptions above, we do the projection. We have calculated the corresponding nearest frame plane whose normalized normal vector is $\vec{n}_0$, the vector from the voxel to the origin vertex $\overrightarrow{p_0}$ (in the codes it is $\overrightarrow{p_2}$) and the distance between the voxel and the frame plane d_0. Then, because we don't know the angle between $\vec{n}_0$ and $\overrightarrow{p_0}$, the projection vector may be $\overrightarrow{p_0} - d_0\vec{n}_0$ or $\overrightarrow{p_0} + d_0\vec{n}_0$, depending on whose norm is smaller. We name the projection vector $\vec{r}$.

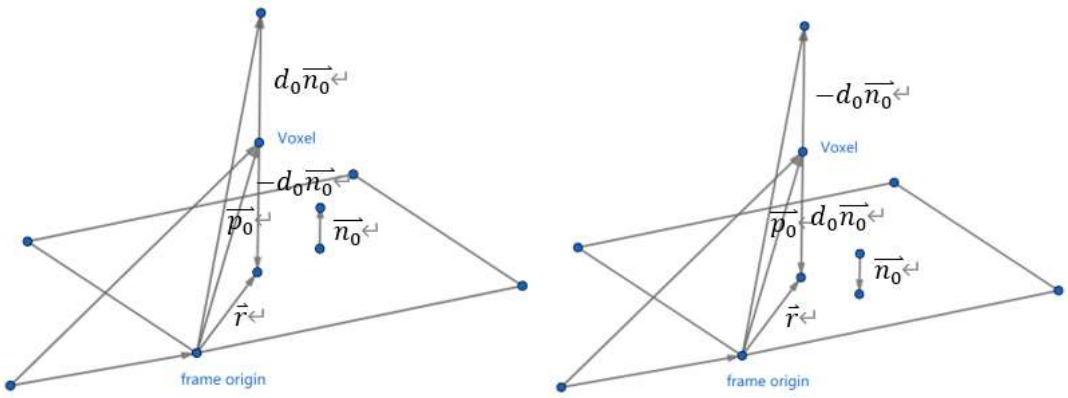


Fig 6: the different angle causes different $\vec{r}$

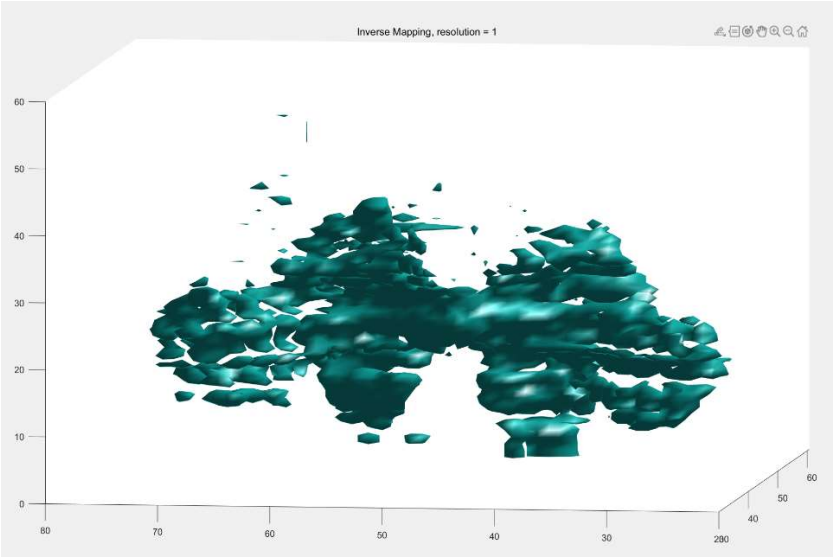
To find the nearest pixel on the frame plane, we express $\vec{r}$ by two linearly independent vectors $\overrightarrow{v_x}$ and $\overrightarrow{v_y}$, which are chosen as the two sides of the frame plane.

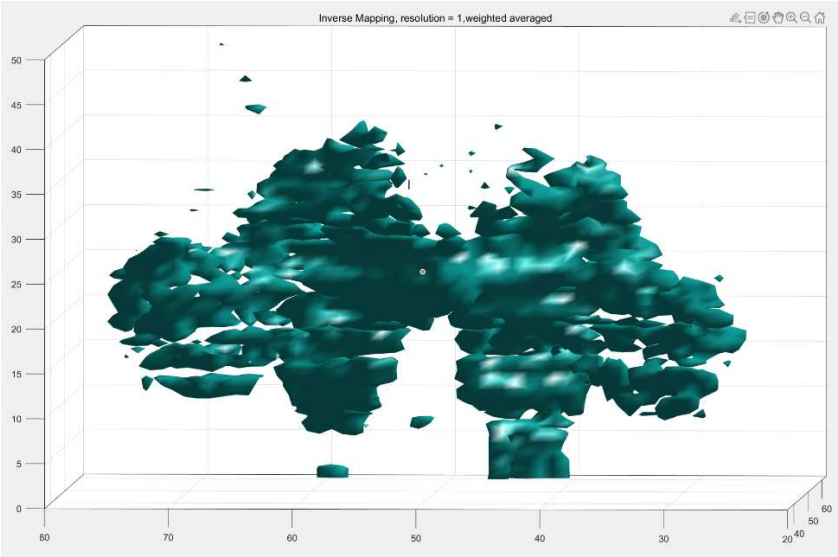
$$\vec{r} = r_x\overrightarrow{v_x} + r_y\overrightarrow{v_y}$$

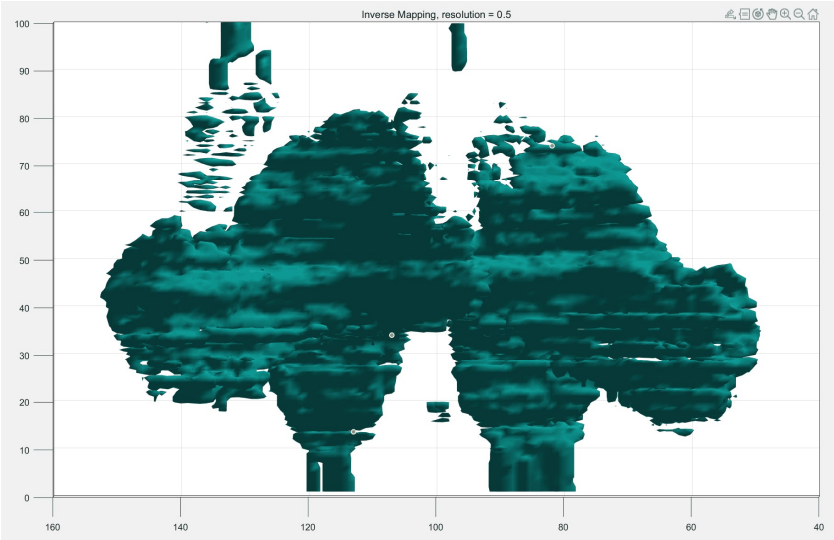
We know that there are 240 pixels on the x-direction and 320 pixels on the y-direction. Then $240r_x$ and $320r_y$ show where the projection point is. Directly, $floor(240r_x), floor(320r_y), ceil(240r_x), ceil(320r_y)$ determine the four nearest pixels (the projection point is just in the square of those four pixels). Then just calculate the distance between the projection point and pixel and find the nearest pixel. then, just assign the value of the pixel to the voxel and perform this process for all voxels and display the 3D matrix

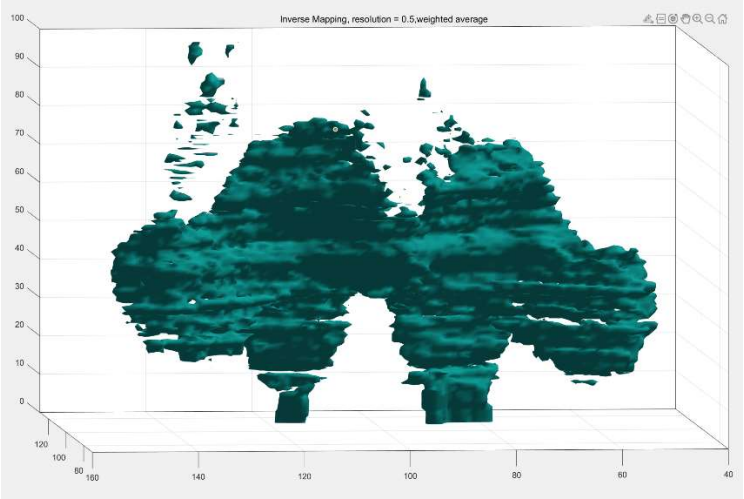
Also, we can give these four pixels a weight determined by the distance and do the weighted average. This way can be a useful improvement.

Final results:









It is found that for the result without averaged, there is some strange columnar volume, which comes from the same isolated pixel. For result after averaged, such volume has decreased, and when the resolution is large, it can avoid a loss of voxel.

Discussion: Comparison of different methods

(i) For stack-by-stack method, there is no gap in the figure, but the edge of the figure is not smooth and this method provides a very low precision to construct a 3D image. It has ignored the differences of the distance and angle of different frame plains.

(ii) For forward mapping method, the independent variable is pixel, the dependent variable is voxel. So, all the value of pixel will be in use. But if there is deficiency in the data of 2D frame (the data is all zero), then there would be gaps in 3D image.

(iii) For inverse mapping method, the independent variable in voxel, the dependent variable is pixel. So, all the voxel could find a corresponding value, and so that it could help to reduce the gaps in the 3D image. However, the shortcuts of this method are （1）The speed of constructing 3D image is slow; (2) Not all the value of pixel would be in use.

Appendix: A check of the resolution of the given pixels

The given data of step2 is theoretically has a resolution 0.296*0.296mm^2, however, if the distances of the neighboring points in GpsData are calculated, it is found that there are two value, 94.70 and 67.33. Which means the resolutions are $\frac{94.70}{320} = 0.296\text{mm}$ and $\frac{67.33}{240} = 0.281\text{mm}$ respectively. Fortunately, the difference of the two sides make no different to the result. The codes for this check is shown below.

```
X=zeros(694*4,1);
Y=zeros(694*4,1);
Z=zeros(694*4,1);
for i=1:1:694
    newgps = T*[gpsData(:,:,i);a];
    for j=1:1:4
        X(4*(i-1)+j)=newgps(1,j);
        Y(4*(i-1)+j)=newgps(2,j);
        Z(4*(i-1)+j)=newgps(3,j);
    end
end



r=zeros(694*4-1,1);
for i=1:1:length(X)-1
    if i-4*floor(i/4) ~= 0
        r(i)=sqrt((X(i+1)-X(i))^2+(Y(i+1)-Y(i))^2+(Z(i+1)-Z(i))^2);
    else
        r(i)=sqrt((X(i-3)-X(i))^2+(Y(i-3)-Y(i))^2+(Z(i-3)-Z(i))^2);
    end
end
scatter(1:1:length(r),r,'.')
```