Digital Image Processing Report
Project 1: Reconstruction

2019233181 Jiale Xu

10/12/2019

1. Intruction to Problems

- Reconstruct a 3D volume by putting the 2D *FrameData* stack by stack and display it.
- Reconstruct a 3D volume using forward mapping method and display it.
- (Optional) Reconstruct a 3D volume using inverse mapping method and display it.
- Discuss the difference of the reconstruction results using different methods.

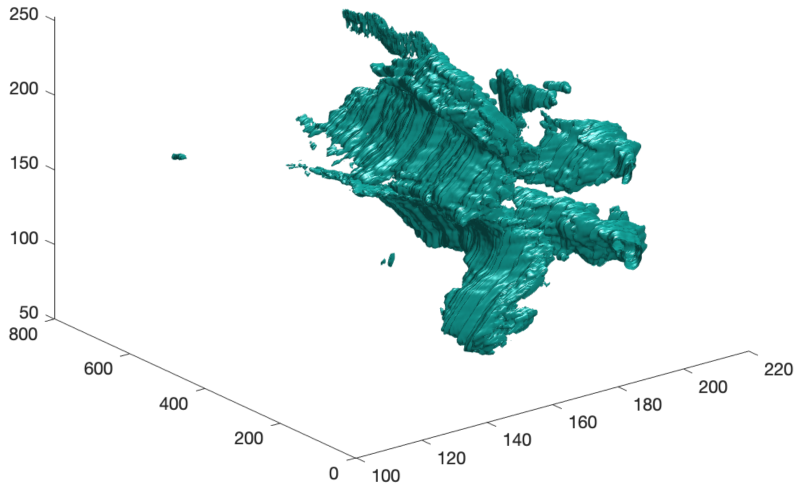
2. Project Content

Problem 1

Code (*stack_and_display.m*):

```
load frameData;
data = zeros(694, 240, 320);
for i = 1:694
    data(i, :, :) = frameData{i}(:, :);
end
isosurface(data);
```

Result:



Problem 2: forward mapping

Procedure:

- Rebuild a regular volume according to the *GpsData*.
- Find the nearest voxel for each pixel.
- Assign the value of pixel to the voxel.
- Display the 3D matrix by using Matlab function *isosurface()*.

First of all, we use *CalibrationPoints.mat* to obtain transformation matrix between GPS transmitter space and world space.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$$

In this case, we have four points in GPS transmitter space and four points in world space which are stored in *CalibrationPoints.mat.* We transform the coornidates into homogeneous coordinates and represent them by $\boldsymbol{G} \in \mathbb{R}^{4\times4}$ and $\boldsymbol{W} \in \mathbb{R}^{4\times4}$ respectively.

Since

$$\boldsymbol{W} = \boldsymbol{T}\boldsymbol{G}$$

We can get the transformation matrix by

$$\boldsymbol{T} = \boldsymbol{W}\boldsymbol{G}^{-1}$$

Code (*getTransformationMatrix.m*):

```
function matrix = getTransformationMatrix(gpsSpaceCoordinates,worldSpaceCoordinates)
gpsSpaceCoordinates = cat(1, gpsSpaceCoordinates, [1 1 1 1]);
matrix = worldSpaceCoordinates / gpsSpaceCoordinates;
matrix = cat(1, matrix, [0 0 0 1]);
end
```

After calculating, the transformation matrix is

$$\boldsymbol{T} = \begin{bmatrix} 0.5634 & 0.6436 & 0.5180 & -165.4164 \\ -0.7077 & 0.0525 & 0.7045 & -329.0293 \\ 0.4264 & -0.7637 & 0.4849 & 231.8552 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

Once we have the transformation matrix, we can transform the points in *GpsData.mat* into world space to get the range of coordinates on each axis, and construct a regular volume accordingly.

```
load calibrationPoints;
load frameData;
load gpsData;
%obtain transformation matrix
matrix = getTransformationMatrix(calibrationPoints(:, :, 1), calibrationPoints(:, :, 2));
%construct regular volume
worldData = matrix * cat(1, reshape(gpsData, 3, 4*694), ones(1, 4*694));
worldData = worldData(1:3, :);
xMin = min(worldData(1, :), [], 'all');
xMax = max(worldData(1, :), [], 'all');
yMin = min(worldData(2, :), [], 'all');
yMax = max(worldData(2, :), [], 'all');
zMin = min(worldData(3, :), [], 'all');
zMax = max(worldData(3, :), [], 'all');
voxelSize = 0.5;
xSize = ceil((xMax-xMin) / voxelSize);
ySize = ceil((yMax-yMin) / voxelSize);
zSize = ceil((zMax-zMin) / voxelSize);
forwardVoxel = zeros(xSize, ySize, zSize);
```

Here, *forwardVoxel* is the constructed volume. As multiple pixels may be mapped to one voxel, we use a variable *count* to record how many pixels are mapped to each voxel, it has the same size as *forwardVoxel.*

```
count = zeros(xSize, ySize, zSize);
```

Then we need to transform pixels in each frame to GPS transmitter space. As we know the coordinates of four corner points of each frame in GPS transmitter space, it's easy to calculate the coordinate of each pixel (here i, j are the indices of a pixel in a frame).

```
rec = gpsData(:, :, img);
m = ((240-i)*rec(:,1)+(i-1)*rec(:,4)) / 239;
n = ((240-i)*rec(:,2)+(i-1)*rec(:,3)) / 239;
gpsPoint = ((320-j)*m+(j-1)*n) / 319;
```

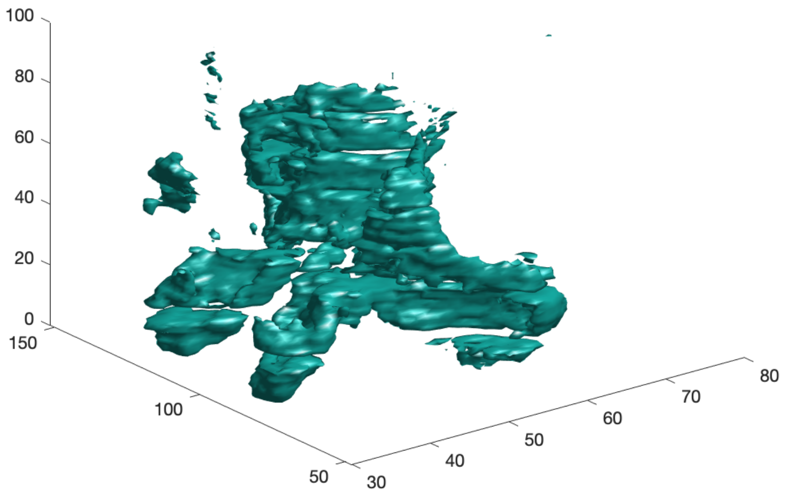
After that, we transform the coordinates of pixels into world space, and assign its intensity value to the nearest voxel.

```
worldPoint = matrix * cat(1, gpsPoint, 1);
worldPoint = worldPoint(1:3);
XYZ = floor(abs(worldPoint - [xMin; yMin; zMin])/voxelSize) + [1; 1; 1];
forwardVoxel(XYZ(1), XYZ(2), XYZ(3)) = forwardVoxel(XYZ(1), XYZ(2), XYZ(3)) + frameData{img}(i,j);
count(XYZ(1), XYZ(2), XYZ(3)) = count(XYZ(1), XYZ(2), XYZ(3)) + 1;
```

We get the intensity values accumulately since multiple pixel values may be assigned to one voxel, so finally we need to take the average on each voxel.

```
for i = 1:xSize
    for j = 1:ySize
        for k = 1:zSize
            if ~count(i, j, k) == 0
                forwardVoxel(i, j, k) = forwardVoxel(i, j, k) / count(i, j, k);
            end
        end
    end
end
```

Result:



Problem 3: inverse mapping

Procedure:

- Rebuild a regular volume according to the *GpsData*.
- Project the voxel onto the nearest frame plane and get the projection points on the plane.
- Assign the value of pixel to the voxel by using the nearest neighbor interpolation method in frame plane.
- Display the 3D matrix by using Matlab function *isosurface()*.

Similar to problem 2, we construct a regular volume first.

```
load calibrationPoints;
load frameData;
load gpsData;
%obtain transformation matrix
matrix = getTransformationMatrix(calibrationPoints(:, :, 1), calibrationPoints(:, :, 2));
%construct regular volume
worldData = matrix * cat(1, reshape(gpsData, 3, 4*694), ones(1, 4*694));
worldData = worldData(1:3, :);
xMin = min(worldData(1, :), [], 'all');
xMax = max(worldData(1, :), [], 'all');
yMin = min(worldData(2, :), [], 'all');
yMax = max(worldData(2, :), [], 'all');
zMin = min(worldData(3, :), [], 'all');
zMax = max(worldData(3, :), [], 'all');
voxelSize = 0.5;
xSize = ceil((xMax-xMin) / voxelSize);
ySize = ceil((yMax-yMin) / voxelSize);
zSize = ceil((zMax-zMin) / voxelSize);
inverseVoxel = zeros(xSize, ySize, zSize);
```

Here, *inverseVoxel* is the constructed volume. Now our goal is to find the corresponding pixels in the GPS transmitter space for each voxel in the screen space.

For each voxel (i, j, k), we first transform it into GPS transmitter space.

```
x = xMin + (i-1)*voxelSize;
y = yMin + (j-1)*voxelSize;
z = zMin + (k-1)*voxelSize;
gpsPoint = matrix \ [x; y; z; 1];
gpsPoint = gpsPoint(1:3);
```

Then we need to calculate the distance between the transformed point to every frame plane in GPS transmitter space. We can achieve this by

$$d = \left| \overrightarrow{AB} \cdot \frac{\vec{n}}{|\vec{n}|} \right|$$

Where $\overrightarrow{AB}$ denotes a vector from point A to B, A is a point outside the plane, B is any point in the plane. $\vec{n}$ is the normal vector of the plane.

```
m = squeeze(gpsData(:, 2, :)) - squeeze(gpsData(:, 1, :));
n = squeeze(gpsData(:, 4, :)) - squeeze(gpsData(:, 1, :));
normals = cross(m, n);
norms = zeros(1, 694);
for l = 1:694
    norms(l) = norm(normals(:, l));
end
distances = abs(dot(squeeze(gpsData(:,1,:))-gpsPoint, normals, 1) ./ norms);
```

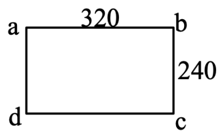
After that, we need to project the point onto the nearest frame plane. We can do this by

$$X_{proj} = \boldsymbol{B} \left(\boldsymbol{B}^T \boldsymbol{B} \right)^{-1} X$$

Where $\boldsymbol{B}$ represents a basis of the plane. Here we can get $\boldsymbol{B}$ by simply concatenate the two vectors of $b - a$ and $d - a$.

```
[minDistance, index] = min(distances);
B = cat(2, m(:, index), n(:, index));
projPoint = B / (B'*B) * B' * gpsPoint; %projection
```

Finally, we assign the value of pixel to the voxel by using the nearest neighbor interpolation method in frame plane. To get the frame indices (row, column) of point X_{proj}, we can calculate the distance between the point and rectangle edges (ab and ad). Then calculate the indices according to proportion.



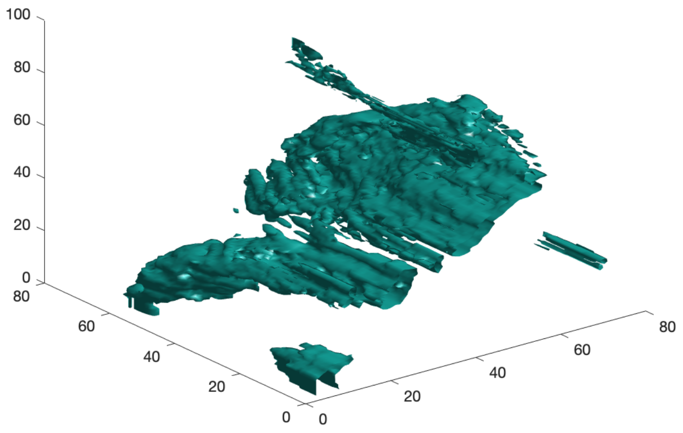
Suppose A, B, C, D represent the four vertices a, b, c, d of frame rectangle, then we can get the distance between point X_{proj} and edges AB, AD by

$$d(X_{proj}, AB) = \frac{|\overrightarrow{AB} \times \overrightarrow{AX_{proj}}|}{|\overrightarrow{AB}|}$$

$$d(X_{proj}, AD) = \frac{|\overrightarrow{AD} \times \overrightarrow{AX_{proj}}|}{|\overrightarrow{AD}|}$$

```
%calculate the distances from projected point to ab and ad
abDistance = norm(cross(gpsData(:,1,index)-projPoint, m(:, index))) / norm(m(:, index));
adDistance = norm(cross(gpsData(:,1,index)-projPoint, n(:, index))) / norm(n(:, index));
abLength = norm(m(:, index));
adLength = norm(n(:, index));
row = round(abDistance / abLength * 239 + 1);
column = round(adDistance / adLength * 319 + 1);
if and(row <= 240, column <= 320)
    inverseVoxel(i, j, k) = frameData{index}(row, column);
end
```

Result:



Problem 4

Discuss the difference of the reconstruction results using different methods:

1. *forward mapping*: there are many blank voxels in the reconstruction result, because forward mapping is likely to map multiple pixels onto one voxel while some voxels are never mapped onto.
2. *inverse mapping*: the result is more solid and watertight compared to forward mapping's result, because every voxel is mapped to a pixel.