Cornell University

M.Eng Project Final Report

3D Landmark Detection:
Detecting the Center of Vertebral Bodies

Daniel Park (dp435)

supervised by
Professor Anthony P. Reeves (apr5)

May 29, 2017

Abstract

Automatic segmentation of the bone from CT images plays a vital role in the implementation of computer-assisted diagnosis. The following project attempts to explore and develop methods to automatically determine the 3D location of the center of each vertebrae by analyzing a set of 3D human CT images of the chest. The proposed algorithm first generates an image where the intervertebral discs are emphasized and places max-cut planes between each column. This yields us with the z-coordinates of the upper and lower surfaces of the vertebral columns, which we then use to calculate the midpoint of to get the z-coordinates of the respective centers. The final step is to look up the the x,y-coordinates of the spinal canal centerline at the z-coordinates of those midpoints. The results produced by the algorithm were evaluated using the 3D Euclidean distance in order to determine how close the identified centers were to their respective ground truths. If the difference was below the specified threshold distance, it was labeled as a successful localization for that center. Success rate for each dataset was defined by the ratio of number of successful VB localization to the total number of VBs.

1 Introduction

1.1 Background

The objective of this project is to design and implement a fully automatic framework that identifies the 3D location of the center of each vertebrae in human CT images of the chest. Often times, the first steps for most diagnosis tasks for the vertebral region involve either localizing or segmenting the vertebrae and the intervertebral discs (Ghosh 1). Such outputs would then further be analyzed to detect deformities or fractures present within the vertebral region. Thus, our framework may potentially be used to aid in automatically localizing, segmenting or labeling the vertebral structures.

1.2 Potential Issues

Complexity is added to the problem of automated analysis of the vertebrae due to the variability in the bone structures of each person and variability in the image acquisition itself, which results in different resolutions and noise. In addition, detecting the intervertebral discs may prove to be difficult since the intevertebral discs and the spongy tissue within the vertebral bodies share similar intensity values. In some cases, it may even be possible for certain regions of the intervertebral disc to not clearly show up in the CT images if the patient suffers from physical deformities or fractures. Thus, an analysis method depending solely on intensity values would likely have to address some of these issues.

1.3 Previous Work

1. **Automated analysis of anatomical structures from low-dose chest computed tomography scans.**
   The dissertation by Lee (2011)[1] provides a method to separate the vertebral bodies

from each other. In order to separate the anterior portion of the vertebrae, an enhanced image of the intervertebral discs is generated by applying a closing operation on the bone segmentation with a spherical kernal expanded the Z-direction and then, subtracting the original segmentation from it. This yields an output where the regions containing intervertebral discs are emphasized. Tangent planes are placed along the centerline of the spinal canal, and sum the voxels they pass through is calculated for each plane. Local maximas are identified, which yields the division between vertebral bodies in the anterior section. The intersection points with the centerline and these plane also serve as seed points when determining the division for the posterior section. A rectangular probe is placed at the seed points and the direction in which it yields the min-cut is determined. This min-cut plane serves as the splitting surface for the spinous processes. From a dataset of 50 CT images, the algorithm successfully segmented 99.7% of the vertebral bodies and 96.3% of the spinous processes.

2. **A fully automatic vertebra segmentation method using 3D deformable fences**

   The paper by Kim et al. (2009)[2] provides a method to automatically detect, identify, and segment each vertebrae in CT images. The first step involves generating a valley image; that is, a 3D greyscale closing morphological operation is first applied to the original image, which then, the original image is subtracted from. Afterwards, a 3D Gaussian kernel ($\sigma = 5mm$) is applied. This is done to not only reduce noise, but to generate an image where the division between two adjacent vertebra is emphasized. The spinal cord is extracted by initializing a 3D sphere inside the spinal canal and propagating it along the axial axis. The path traced by the center of this sphere during propagation defines the centerline. The intervertebral disc is detected by generating ray vectors of length r at varying panning and tilting angles with the points on the centerline defined as its endpoint. For each ray, the maximum distance that it can travel before hitting bone or the centerline point itself is calculated. The intervertebral disc regions are then found by finding the local maximas of these distances. Each maximum ray serves as the initial plane that passes through the intervertebral disc, which is then deformed until the model reaches convergence. The proposed algorithm was run on 50 data sets (293 vertebral columns) and its output was subjectively evaluated. It was determined through manual inspection that approximately 80% of the vertebral columns were successfully segmented.

3. **Automated detection of spinal centrelines, vertebral bodies and intervertebral discs in CT and MR images of lumbar spine**

   The paper by Stern et al. (2009)[3] provides a way to automatically extract the spinal centerline, and the centers of vertebral bodies/intervertebral discs. The centerline is determined by first running the image through a 3D Canny and Sobel detectors. This provides the edge and gradient vectors. Opposite edge points are used to estimate the center points while a 3D accumulator keeps track of these estimations. The region with the highest accumulator value is determined to be the center point. After the centerline is extracted, the image intensity and gradient magnitude profile along it is calculated. Local extremas from both profiles are identified, which then are passed

through a correlation function in order to classify them as either the center of the intervertebral disc or the vertebral bodies. For identifying the center of the intervertebral discs and vertebral bodies, the mean difference was $2.7 \pm 1.9mm$ between the actual output and their respective ground truths.

4. **Fully Automatic Localization and Segmentation of 3D Vertebral Bodies from CT/MR Images via a Learning-Based Method**

   This paper by Chu et al. (2015)[4] describes a learning-based approach to segmenting 3D vertebral bodies and identifying the center of each body. Although our project will not employ a learning-based approach, we use the evaluation metrics described in the paper. Chu et al. (2015)[3] computes the localization distance R for each VB (vertebral body) center using:

$$R = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}$$

   where Δx is the absolute difference between the X axis of the identified center and the X axis of the center of the ground truth. Δy and Δz follow the same logic. Chu et al. (2015)[4] classifies the VB center identification as successful if the corresponding R value is less than t mm. "The successful localization rate P_t with accuracy of less than t mm is formulated as follows" (Chu 12):

$$P_t = \frac{\text{number of successful VB localization}}{\text{number of VBs}}$$

1.4 Overview

In Section 2, we describe the algorithm that we implemented to determine the center locations of each vertebral body. We state our expectations for the outcome based on results from previous work, and also present our experiment procedure: we describe the dataset we will use for our training and testing sets, the evaluation metrics that will be used to evaluate the performance, and possible critical parameters that can be adjusted, if necessary. In Section 3, we present the results generated by the algorithm. In Section 4, we analyze and outline the results. We explore potential strengths and weaknesses of the algorithm, and also compare the our findings with our initial hypothesis. In Section 4, we conclude the report by providing a brief summary of our findings and also suggesting possible avenues for future works.

2 Methods

2.1 Algorithm

The high-level logic of the algorithm is that we first generate an image where the intervertebral discs are emphasized. We then identify the 13 intervertebral discs by finding the max-cut planes that pass through them. We find the intersection points between these planes and the spinal canal centerline, and then, take the midpoint between each adjacent intersection

point. This yields us with the z-coordinate of the center of each vertebral body. We can then obtain the remaining x,y-coordinates by looking up the coordinates of the centerline at the corresponding midpoints. The algorithm is as follows:

1. Extract the masks for the vertebrae and spinal canal centerline.
2. Apply a morphological closing operation on the vertebrae mask with an ellipsoidal kernel that is widest in the z-direction. This is done to capture the regions belonging to the intervertebral discs.

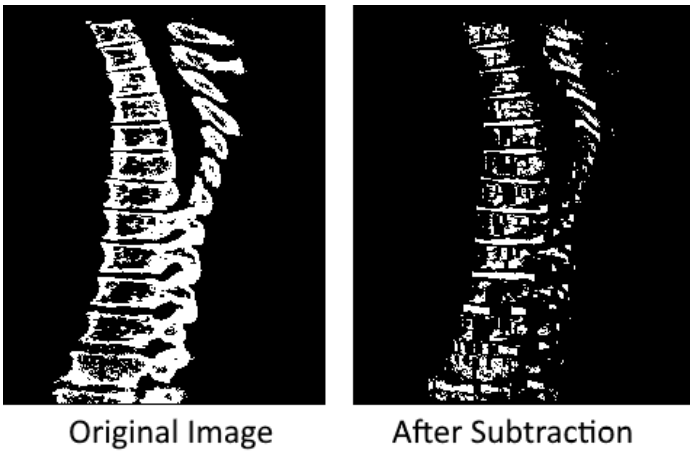

Figure 1: Effect of "emphasizing" intervertebral disc regions

3. Subtract the original vertebrae mask from the output of Step 2 to obtain a binary image where the intervertebral disc regions are emphasized.

   Figure 1 shows the image before and after the subtraction is applied.

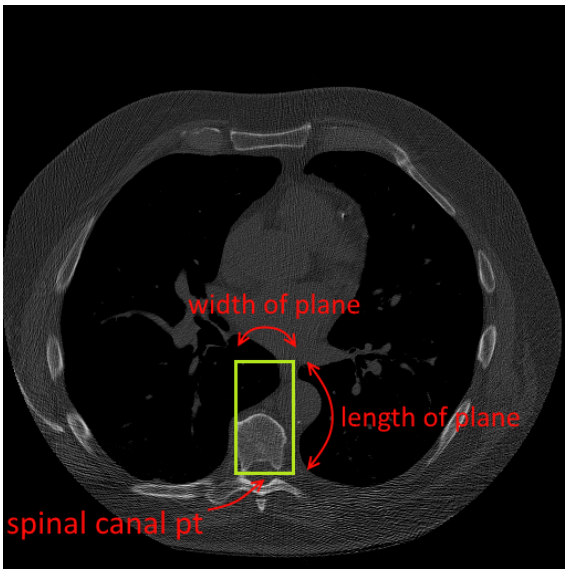

Figure 2: Example of plane initialization

4. Using the "intervertebral disc emphasized" image, for each point on the spinal canal centerline:

   (a) we place a plane toward the anterior direction that contains that point and is parallel to the XY-axis.

   For example, if we look at Figure 2, the outer edges of the plane are indicated by the yellow box. The plane is placed so that the spinal canal point lies on the middle of the bottom edge of the plane that we are considering.

   (b) we rotate this plane by $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ with respect to the X-axis and then, determine the maximum sum of the voxel intensities that this plane passes through.

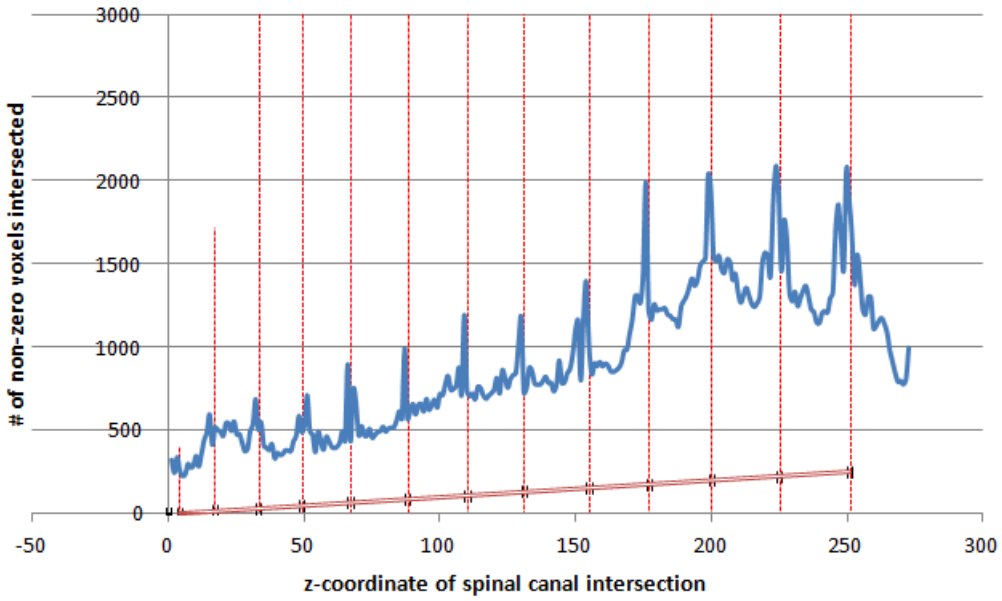

Figure 3: W0002: detected local maximas

5. After determining the maximum sum of the voxel intensities for each plane, we identify 13 local maximas. Then, we store the Z-coordinates of the intersection point between each max-cut plane and the spinal canal centerline into a "intervertebral disc candidate list":

   (a) At this step, we place constraints that the local maximas have to be at least 15 pixels away to avoid selecting multiple planes that pass through the same intervertebral disc.

6. If the algorithm fails to detect some local maximas, we find the two adjacent planes whose intersection with the centerline are furthest apart in the z-direction, and place a plane in the center of the two. Once again, we store the z-coordinate of the intersection point between this new plane and the centerline into the candidate list.

   Figure 3 shows an example of the local maximas being detected for the dataset, W0002.

7. Sort the candidate list in increasing order.

8. Take the midpoint of each adjacent candidate in the candidate list. Taking the midpoint between the adjacent intervertebral discs yields us the z-coordinate corresponding to the center of each vertebral body.

9. Output the centerline coordinates whose z-coordinates correspond to the midpoints calculated in Step 7.

2.2 Experiment

2.2.1 Hypothesis and evaluation function

To measure the accuracy of the identified centers of the vertebral bodies, we used the evaluation metric discussed by Chu et al. (2015)[4]. The localization distance R for each VB (vertebral body) center is calculated by the following:

$$R = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}$$

The VB center identification is classified as successful if the corresponding R value is less than t mm. "The successful localization rate P_t with accuracy of less than t mm is formulated as follows" (Chu 12):

$$P_t = \frac{\text{number of successful VB localization}}{\text{number of VBs}}$$

Based on qualitative analysis, we observed the height of each vertebral column to be approximately 15-25 pixels high in the z-direction. Thus, the distance we defined for a center point to be labeled as correct was 5 pixels within the z-coordinate of the expected ground truth. We chose a slack of 5 pixels to account for user error since we manually marked the ground truths. We also chose to assume the correctness of the x,y-coordinates since our program just inherited and outputted those values from the label map of the spinal canal centerline, which was marked by a professional.

Our algorithm follows a similar approaches taken by Lee (2011)[1] and Kim et al. (2009)[2]. Lee (2011)[1] was able to successfully segment 99.7% of the vertebral bodies, and Kim et al. (2009)[2] claimed that around 80% of their segmentations were classified as "good". Thus, we expect our algorithm to be at least 90% successful (i.e. approximate midpoint between the two) in identifying the center locations based on their results.

2.2.2 Documented Data Set

The 3D CT images come from the SIMBA Framework Public Access Database. For fifty of the images in this data set, label maps for different segmented image regions are provided. To establish the ground truths for each data set, we identify the z-coordinates of the upper and lower surfaces of each vertebral column. We then take the midpoint of the two to get the z-coordinates of the center of each vertebral column. To get the x,y-coordinates, we find

the coordinates for the center of the spinal canal at the corresponding z-levels. Each data set will have ground truths containing 12 center points. If more than 12 columns are present, we will only consider the first 12 center points with the lowest z-coordinates.

2.2.3 Experiment Procedure

Our training and testing set will each contain 10 unique data sets. After the ground truths are manually marked, we will first run our algorithm on the training set and evaluate the results using the evaluation function mentioned in Section 2.2.1. The following critical parameters will be readjusted to yield better results, if necessary:

- the size of the kernel used when applying morphological filter operations
- length and width of the max-cut plane.
- the angle of tilt of the max-cut planes: $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ radians at intervals of $\frac{\pi}{64}$.
- the minimum distance that the local maximas have to be apart from each other in Step 5 of the algorithm.

After the optimization step is complete, we will run our algorithm on the testing set and evaluate the final results.

3 Results

Each entry in the following tables is defined by: $\frac{\#\text{ OF CORRECT CENTER POINTS}}{\#\text{ OF CENTER POINTS}}$

Table 1: Effects of changing kernel size for morphological operations

Kernel Size (x,y,z)	W0001	W0002	W0003	W0004	W0005	W0006	W0007	W0008	W0009	W0010	Avg. Score
(2,2,4)	0.33333	0.66666	0.91666	0.75	0.41666	0.58333	0.66666	0.83333	0.58333	0.58333	0.63333
(1,1,6)	1.0	1.0	0.83333	0.83333	1.0	1.0	0.91666	0.66666	1.0	0.75	0.9
(2,2,6)	0.75	1.0	0.83333	0.83333	1.0	0.75	0.83333	0.66666	0.91667	0.58333	0.81666
(1,1,8)	1.0	1.0	0.83333	0.83333	1.0	1.0	0.91666	0.66666	1.0	0.91666	0.91666
(2,2,8)	1.0	1.0	0.91666	1.0	1.0	1.0	1.0	0.58333	1.0	0.83333	0.9
(1,1,10)	1.0	1.0	0.83333	0.83333	1.0	1.0	0.91666	0.75	0.83333	0.75	0.89166
(2,2,10)	1.0	1.0	0.83333	0.75	1.0	1.0	0.91666	0.75	1.0	0.91667	0.91666

Table 2: Effects of tilting plane

Tilt Angle	W0001	W0002	W0003	W0004	W0005	W0006	W0007	W0008	W0009	W0010	Avg. Score
$[-\pi/4, \pi/4]$	0.91666	1.0	0.75	0.83333	1.0	1.0	0.75	0.5	1.0	1.0	0.875
$[-\pi/6, \pi/6]$	1.0	1.0	0.91666	0.83333	1.0	1.0	0.66666	0.75	0.83333	0.83333	0.88333
$[-\pi/8, \pi/8]$	1.0	1.0	0.83333	0.75	1.0	1.0	0.91666	0.75	1.0	0.91667	0.91666
$[-\pi/10, \pi/10]$	1.0	1.0	0.75	0.83333	1.0	1.0	0.75	0.75	1.0	0.75	0.88333
$[-\pi/12, \pi/12]$	1.0	1.0	0.83333	0.83333	1.0	1.0	0.83333	0.58333	1.0	0.75	0.88333

Table 3: Effects of plane size

Tilt Angle	W0001	W0002	W0003	W0004	W0005	W0006	W0007	W0008	W0009	W0010	Avg. Score
l=50, w=50	1.0	1.0	0.83333	0.75	1.0	1.0	1.0	0.91666	1.0	0.83333	0.93333
l=50, w=75	1.0	1.0	0.91666	0.83333	1.0	1.0	1.0	0.66666	1.0	0.833333	0.925
l=75, w=75	1.0	1.0	0.833333	0.83333	1.0	1.0	1.0	0.83333	1.0	0.83333	0.93333
l=100, w=50	1.0	1.0	0.83333	0.75	1.0	1.0	0.91666	0.75	1.0	0.91667	0.91666
l=100, w=75	1.0	1.0	0.833333	0.83333	1.0	1.0	0.91666	0.66666	1.0	1.0	0.925
l=100, w=100	1.0	1.0	0.833333	0.83333	1.0	1.0	0.91666	0.66666	1.0	1.0	0.925

Table 4: Restricting how close local maximas can be initially

minimum distance	W0001	W0002	W0003	W0004	W0005	W0006	W0007	W0008	W0009	W0010	Avg. Score
d=10	1.0	0.83333	0.83333	0.83333	1.0	1.0	0.83333	0.83333	1.0	0.66666	0.88333
d=13	1.0	1.0	0.83333	0.83333	1.0	1.0	1.0	0.83333	1.0	0.83333	0.93333
d=15	1.0	1.0	0.83333	0.83333	1.0	1.0	1.0	0.83333	1.0	0.83333	0.93333
d=17	1.0	1.0	0.83333	0.75	1.0	1.0	1.0	0.58333	1.0	0.83333	0.9
d=20	1.0	1.0	0.91666	0.91666	1.0	1.0	1.0	0.58333	1.0	0.83333	0.925

Table 5: Best Achieved Training Set Score

caseID	W0001	W0002	W0003	W0004	W0005	W0006	W0007	W0008	W0009	W0010	Avg. Score
FRACTION CORRECT	1.0	1.0	0.833333	0.83333	1.0	1.0	1.0	0.83333	1.0	0.83333	0.93333

Parameters used:

- Morphological kernel size set to (2,2,10)
- Tilting angle of plane ranges from $[-\pi/8, \pi/8]$ radians at intervals of $\dfrac{\pi}{64}$
- Plane size set to length=75, width=75
- Each local maxima must *initially* be at least 15 pixels away in the z-direction. (Planes can be fit between adjacent planes later on, if necessary.)

Table 6: Testing Set Results (% correct)

caseID	W0011	W0012	W0013	W0016	W0017	W0018	W0019	W0020	W0021	W0022	Avg. Score
FRACTION CORRECT	1.0	1.0	0.83333	1.0	1.0	0.66666	1.0	1.0	0.75	0.66666	0.89166

4 Discussion

Our algorithm had 4 critical parameters: the kernel size of the morphological closing operation, the maximum tilt of the max-cut planes, the size of the max-cut planes, and the minimum distance between each adjacent plane (i.e. local maximums). Based on the results from Table 1, we found that a spherical kernel with a height of at least 8 pixels was sufficient

enough to fill in the intervertebral discs. There was no significant change when we increased the kernel height from 8 to 10 pixels, which was expected since most regions belonging to the intervertebral discs were already filled. However, we speculate that continually increasing the size would actually hurt performance since this pre-processing operation would "emphasize" the spongy tissue within the vertebral bodies even more, potentially causing them to be mistaken for discs when taking the max-cut planes. Next, we varied the angle at which we tilted the planes and found that a maximum tilt of $\pm\pi/8$ radians produced the best result (average score of 0.91666). However, the other angles that we tested for did not produce significantly different outcomes (Table 2). In addition, varying the width and length of the plane had minimal effect on the overall scores (Table 3). Lastly, constraining each local maximum to be at least 13-15 pixels away from each other produced the best average score of 0.93333 for the training set (Table 3).

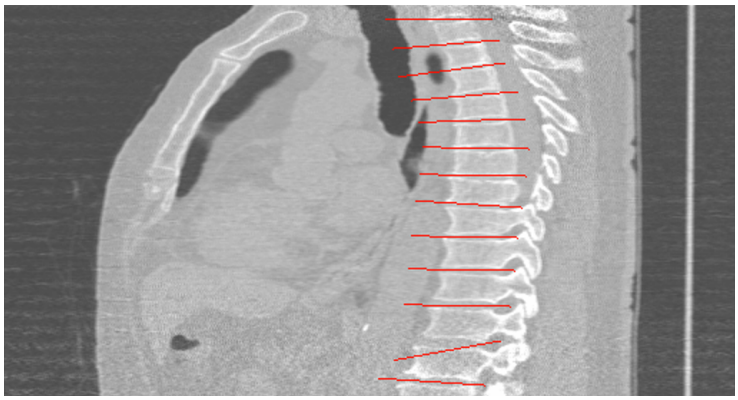

Figure 4: Max-cut planes in sagittal view (W0001)

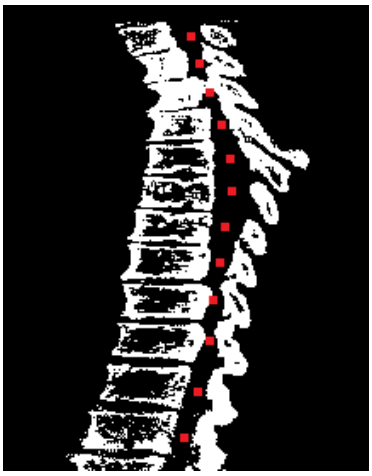

Figure 5: Successful detection of all center points (W0001)

The best average score obtained for the training set was 93.333% correctness (i.e. 112 center points out of 120 were identified correctly), and the score obtained for the testing set was 89.166% (i.e 107 center points out of 120 were identified correctly). Our algorithm performed well when there was a significant enough gap between each column; that is, the intervertebral discs were easily detectable. In addition, our algorithm produced good results for cases where there wasn't a significant curvature in the spine. Even in cases where some of the discs were not detected, placing a plane in between two adjacent planes that were farthest away from each other often helped resolve this issue.

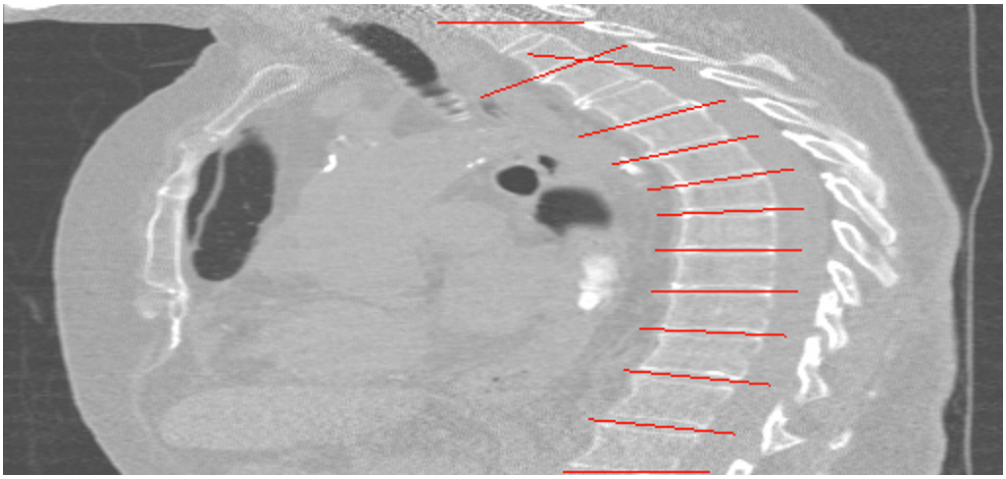

Figure 6: Max-cut planes in sagittal view (W0004)

Unfortunately, we were not able to achieve our goal of obtaining a success rate of over 90% on our testing set. There were a couple factors that prevented us from achieving the target score. The first potential cause was that the parameters (specifically plane size and tilt) that we optimized for our training set did not translate that well into our testing set. For example, there were a few images that experienced far greater curvature in the spine (Figure 6). Using the optimized parameters either "over-tilted" or "under-tilted" the plane in some cases, causing us to miss the intervertebral disc when fitting the plane.

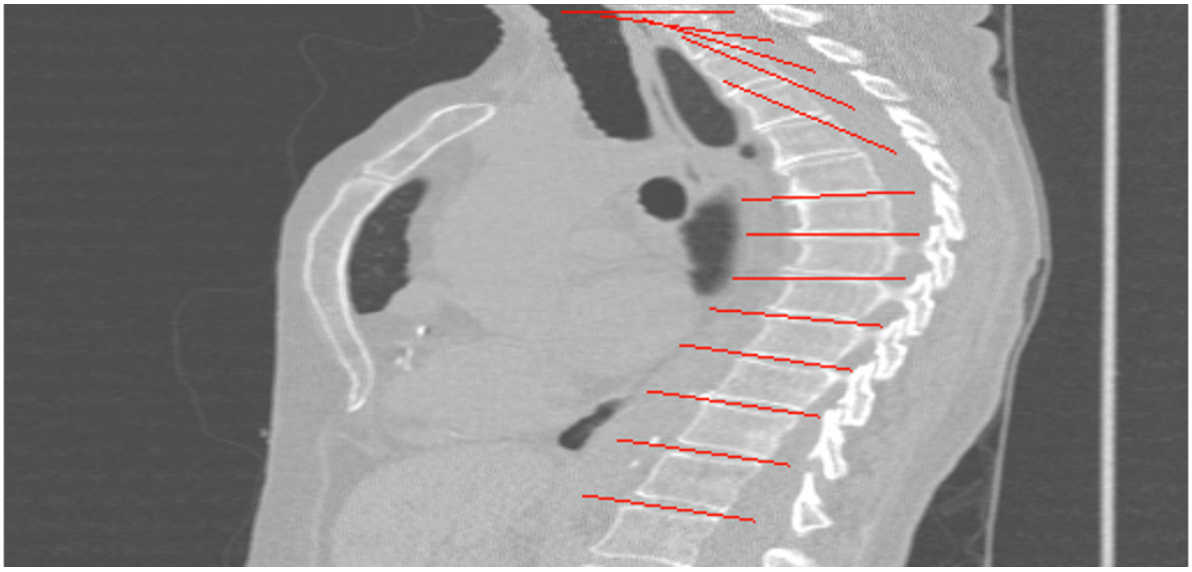

Figure 7: Max-cut planes in sagittal view of CT image (W0016)

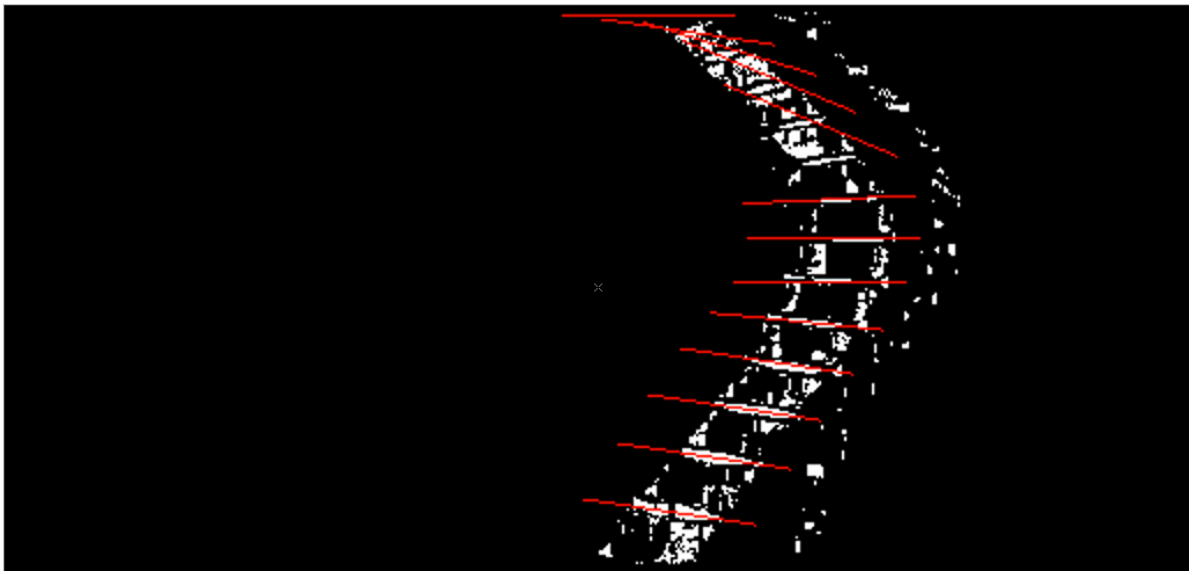

Figure 8: Max-cut planes in sagittal view of inververtebral disc-emphasized image (W0016)

In some of the scans, the close proximity of adjacent vertebral bodies caused the discs to not show up clearly, even after we pre-processed it to emphasize them. In addition, this pre-processing operation also emphasized the spongy tissue within the vertebral body (Figure 8). As a result, this caused us to select the incorrect max-cut planes and misidentify them as the ones that passed through the intervertebral discs. Further refining the upper and lower bounds of the tilt angles may have helped to mitigate these issues.

5 Conclusion

Our algorithm first creates an image where the intervertebral discs are emphasized. Afterwards, it attempts to fit planes across these discs by finding the max-cut planes. It then takes the midpoint of each adjacent plane and uses that to find the centerpoints. Using this algorithm, we were able to achieve a success rate of 93.333% on our training set and 89.166% on our testing set.

One of the major challenges we faced during this project was the vast variability in bone structures between each patient. This caused the parameters (i.e. morphological filter size, plane size and tilt, etc.) that we optimized for our training set to not translate as well as we had hoped for our testing set. In addition, the morphological closing operation that emphasized the intervertebral discs also emphasized the spongy bone inside the vertebral bodies, which caused incorrect planes to be selected and further lowered performance.

Possible future work may include using a similar approach taken by this max-cut plane algorithm in conjunction with a 3D region growing algorithm to isolate each vertebral body. It may then be possible to perform individual analysis on them, such as measuring volume, or even detecting fractures.

6 References

1. Lee, Jaesung. Automated analysis of anatomical structures from low-dose chest computed tomography scans. Diss. Cornell University, 2011.

2. Kim, Yiebin, and Dongsung Kim. "A fully automatic vertebra segmentation method using 3D deformable fences." *Computerized Medical Imaging and Graphics* 33.5 (2009): 343-352.

3. Stern, Darko, et al. "Automated detection of spinal centrelines, vertebral bodies and intervertebral discs in CT and MR images of lumbar spine." Physics in medicine and biology 55.1 (2009): 247.

4. Chu, Chengwen, et al. "Fully Automatic Localization and Segmentation of 3D Vertebral Bodies from CT/MR Images via a Learning-Based Method." *PloS one* 10.11 (2015): e0143327.

5. Ghosh, Subarna, et al. "Automatic lumbar vertebra segmentation from clinical CT for wedge compression fracture diagnosis." SPIE Medical Imaging. International Society for Optics and Photonics, 2011.

7 Program Documentation

NAME

link - imglnks all cases specified in input file and extracts necessary masks for each case.

SYNOPSIS

link [case file]

DESCRIPTION

link is a bash script that imglnks the cases specified in the input file, and also extracts the masks for the vertebra and spinal canal.

CONSTRAINTS

Each line in the text file contains a single case ID.

OPTIONS

case file=	file containing case IDs

AUTHOR(S)

Daniel Park

NAME

maxcut - outputs the centerpoints of the vertebral bodies in .csv format.

SYNOPSIS

maxcut [if=image][ig=inmap][sp=inmap][of=outfile][-v]

DESCRIPTION

maxcut identifies max-cut planes that passes through the intervertebral discs in order to identify the upper and lower surfaces of the anterior portion of each vertebral body. Once the intersection points with the planes and the spinal canal centerline is determined, the midpoint for each adjacent point is calculated. The center points are determined by looking up the coordinates of the centerline existing at the z-coordinates specified by the midpoints.

CONSTRAINTS

Input file if is the original CT scan. ig is a mask where the intervertebral disc is emphasized (apply closing operation on vertebrae mask and subtract the original vertebrae mask from it). fp is the label map with the segmented spinal canal. of is the output .csv file.

OPTIONS

if=	input scan
ig=	intervertebral disc emphasized mask file
sp=	spinal canal mask file
of=	output .csv file
-v	verbose mode

AUTHOR(S)

Daniel Park

NAME

exec - runs maxcut against all cases specified in input file, and also evaluates result using compare.py

SYNOPSIS

exec [case file]

DESCRIPTION

Compiles maxcut.c and executes the program for all cases specified in the case file.

CONSTRAINTS

Each line in the text file contains a single case ID.

OPTIONS

case file= file containing case IDs

SEE ALSO

maxcut, compare

AUTHOR(S)

Daniel Park

NAME

compare.py - compares output to ground truth

SYNOPSIS

compare [generated output] [ground truth]

DESCRIPTION

A program that evaluates the output of the maxcut program against respective ground truth. Uses a greedy matching algorithm, and thus, may not produce optimal matchings (i.e. lower rate than actual success rate).

CONSTRAINTS

Each .csv file has 12 coordinates in increase z-coordinate order.

OPTIONS

generated output= .csv file containing output of mincut.

ground truth= .csv file containing respective ground truths.

SEE ALSO

maxcut, exec

AUTHOR(S)

Daniel Park

8 Program Code

Listing 1: link

```
#!/bin/sh

# imglnk all cases listed in the file specified by $1
for i in `cat $1`
do
  echo $i
  if [ ! -d $i ]; then
    #link case
    imglnk SR $i -m
  fi
  cd $i
  vlmap if=s1map.vx of=spcanal.vx -e o=spcanal
  vlmap if=s1map.vx of=vert.vx -e o=vertebrae
  vmorph -de if=vert.vx of=morph.vx s=2,2,10 t=s
  vop -sub if=morph.vx ig=vert.vx of=disc.vx
  rm morph.vx
  cd ..
done
echo "Linking completed."
```

Listing 2: exec

```
#!/bin/sh

# compile .c program
vcc maxcut.c -o maxcut

# imglnk all cases listed in the file specified by $1
for i in `cat $1`
do
# run program
  maxcut if=$i/s1.if ig=$i/disc.vx sp=$i/spcanal.vx of=$i/center.csv
  # python compare.py $i/center.csv $i/truth
done
echo "Execution completed."
```

Listing 3: compare.py

```
import sys
import csv
import copy
from math import sqrt

def score(c1, c2):
    distance = sqrt((c2[2]-c1[2])**2.0)
    return distance

if __name__ == "__main__":

    maxDist = 5

    f1 = open(sys.argv[1],'rb')
    f2 = open(sys.argv[2],'rb')

    userCoords = []
    solnCoords = []

    # load arrays.
    reader1 = csv.reader(f1)
    for row in reader1:
        floatArray = [float(float_string) for float_string in row]
        userCoords.append(floatArray)

    reader2 = csv.reader(f2)
    for row in reader2:
        floatArray = [float(float_string) for float_string in row]
        solnCoords.append(floatArray)

    # compare distances.
    success = 0.0
    tmpCoords = copy.copy(solnCoords)
    for k in range(0, len(userCoords)):
        for j in range(0, len(solnCoords)):
            eval_score = score(userCoords[k], solnCoords[j])
            if (eval_score < maxDist):
                if (solnCoords[j] in tmpCoords):
                    tmpCoords.remove(solnCoords[j])
                    success += 1.0
                    break

    #print("# of success: " + str(int(success)))
    print(str(success/len(userCoords)))
    f1.close()
    f2.close()
```

Listing 4: maxcut.c

```
/*************************************************************/
/* VisX4 program maxcut                                      */
/*                                                           */
/* Finds max-cut plane that passes between each vertebral    */
/* column, and uses that to find the center of each vertebra.*/
/*************************************************************/
#include "VisXV4.h"          /* VisX structure include file  */
#include "Vutil.h"           /* VisX utility header files    */
#include "math.h"

extern char *VisXhist;

char * pname = "maxcut";

VisXfile_t *VXin,*VXin2, *VXin3;      /* input file structure
*/
FILE * fp; VisXfile_t *VXout;         /* output file structure
*/
VisXelem_t *VXlist,**VXpt;            /* VisX data structure
*/
VisXelem_t *mlist, *splist;           /* VisX data structure
*/

VisX3dim_t  sim;                      /* source image structure
*/
VisX3dim_t  rim;                      /* result image structure
*/
VisX3dim_t  mim;                      /* mask image structure
*/
VisX3dim_t  spim;                     /* spcanal mask image structure
*/
VisXiinfo_t imginfo;
float xres,yres,zres,ri,rs;

void VX3frameset(VisX3dim_t *is, VisX3dim_t *ir);
void quicksort(int cand[], int first, int last);
void rotateX(float arr[3], float angle);
int findPlaneCoords(float normal[3], int point[3], int x, int y);
void limitSearchSpace(int searchSpace[], int lowerBound, int upperBound);

VXparam_t par[] = {
  {"if=",    0,   "input CT image"},
  {"ig=",    0,   "vertebrae mask file"},
  {"sp=",    0,   "spinal canal mask file"},
  {"of=",    0,   "output file"},
```

```
  {"-v",      0,   "verbose flag"},
  { 0,        0,   0},
};

/* Command line parameters are accessed in code by vars below */
#define  IVAL    par[0].val
#define  MVAL    par[1].val
#define  SPVAL   par[2].val
#define  OVAL    par[3].val
#define  VFLAG   par[4].val

int
main(argc, argv)
int argc;
char *argv[];
{
  int i,j,k,x,y,z;
  int xmin,xmax,ymin,ymax,zmin,zmax;

  int sp_arr[400][3];

  int cand[13];                  /* array holding Z-coords of top/bottom
                                    surfaces of each vertebral body. */
  int candIdx = 0;

  /* control param for how close candidates can be. */
  int restrictionWindow = 15;

  /* control params for plane width & length*/
  int width = 75;
  int length = 75;

  VisXelem_t *vptr = NULL, *mptr = NULL, *sptr = NULL;

  VXparse(&argc, &argv, par);     /* parse the command line      */

  VXin  = VXopen(IVAL, 0);        /* open input file             */
  VXout = VXopen(OVAL, 1);        /* open the output file        */

  VXlist = VXread(VXin);          /* read input file             */
  VXgetresinfo( &imginfo);
  VXgetrescale( &imginfo);
  xres = imginfo.xres;
  yres = imginfo.yres;
  zres = imginfo.zres;
  ri = imginfo.ri;
  rs = imginfo.rs;
```

```
if( VFLAG) {
  fprintf(stderr, "img res = %f x %f x %f ri %f rs %f\n",
    imginfo.xres, imginfo.yres, imginfo.zres, imginfo.ri, imginfo.rs);
}

if(VXNIL == (vptr = VXfind(VXlist, VX_PSHORT))){
  fprintf(stderr, "%s: no acceptable input image found, exiting.\n",pname);
  exit(1);
}

/* Initialize input image structure */
VXset3dim(&sim, vptr, VXin);
if(sim.chan != 1){
  fprintf(stderr, "%s: Multi-channel images are not supported.\n",pname);
  exit(1);
}

if ( SPVAL ) {
  if ( VFLAG ) {
    fprintf(stderr, "%s: Mask file specified, applying mask...\n",pname);
  }
  VXin3 = VXopen(SPVAL, 0);        /* open mask file                */

  /* Read spinal canal mask file */
  splist = VXread(VXin3);
  if (VXNIL == (sptr = VXfind(splist,VX_PBYTE)) ) {
    fprintf(stderr, "%s: Invalid format for mask file.\n",pname);
    exit(1);
  }

  VXset3dim(&spim, sptr, VXin3);

  /* Check if image and mask have same bounding box, warn if not
     Note that this is not a problem for this program, so we don't
     do anything.                                                  */
  if ( (sim.xlo != spim.xlo) || (sim.xhi != spim.xhi) ||
       (sim.ylo != spim.ylo) || (sim.yhi != spim.yhi) ||
       (sim.zlo != spim.zlo) || (sim.zhi != spim.zhi) ) {
    fprintf(stderr, "%s: bounding boxes do not match.\n",pname);
  }

  /* Load spinal canal coords into array. */
  int sp_idx = 0;
  for (k = spim.zlo; k <= spim.zhi; k++) {
    for (j = spim.ylo; j <= spim.yhi; j++) {
      for (i = spim.xlo; i <= spim.xhi; i++) {
        if (spim.u[k][j][i] == 255) {
```

```
            sp_arr[k][0] = i;
            sp_arr[k][1] = j;
            sp_arr[k][2] = k;
          }
        }
      }
    }
  }

  /* Apply mask if mask image is specified*/
  if ( MVAL ) {
    if ( VFLAG ) {
      fprintf(stderr, "%s: Mask file specified, applying mask...\n",pname);
    }
    VXin2 = VXopen(MVAL, 0);        /* open mask file                */

    /* Read mask file */
    mlist = VXread(VXin2);
    if (VXNIL == (mptr = VXfind(mlist,VX_PBYTE)) ) {
      fprintf(stderr, "%s: Invalid format for mask file.\n",pname);
      exit(1);
    }

    VXset3dim(&mim, mptr, VXin2);

    /* Check if image and mask have same bounding box, warn if not
       Note that this is not a problem for this program, so we don't
       do anything.                                                  */
    if ( (sim.xlo != mim.xlo) || (sim.xhi != mim.xhi) ||
         (sim.ylo != mim.ylo) || (sim.yhi != mim.yhi) ||
         (sim.zlo != mim.zlo) || (sim.zhi != mim.zhi) ) {
      fprintf(stderr, "%s: bounding boxes do not match.\n",pname);
    }

    /* Determine what regions overlap */
    if (sim.xlo > mim.xlo) {
        xmin = sim.xlo;
    } else {
        xmin = mim.xlo;
    }
    if (sim.xhi < mim.xhi) {
        xmax = sim.xhi;
    } else {
        xmax = mim.xhi;
    }
    if (sim.ylo > mim.ylo) {
        ymin = sim.ylo;
```

```
  } else {
      ymin = mim.ylo;
  }
  if (sim.yhi < mim.yhi) {
      ymax = sim.yhi;
  } else {
      ymax = mim.yhi;
  }
  if (sim.zlo > mim.zlo) {
      zmin = sim.zlo;
  } else {
      zmin = mim.zlo;
  }
  if (sim.zhi < mim.zhi) {
      zmax = sim.zhi;
  } else {
      zmax = mim.zhi;
  }
  /* Apply mask to source image */
  for (k = zmin; k <= zmax; k++) {
    for (j = ymin; j <= ymax; j++) {
      for (i = xmin; i <= xmax; i++) {
        if (mim.u[k][j][i] == 0) {
          /* set pixels to lowest value for short */
          sim.s[k][j][i] = -32767;
        }
      }
    }
  }
}

/* Create result image structure */
VXmake3dim(&rim, VX_PBYTE, sim.bbx, sim.chan);

/* Find first frame of interest. */
int isFound = 0;
int frameStart = 0;

for (k = zmin; k <= zmax; k++) {
  x = sp_arr[k][0];
  y = sp_arr[k][1];
  z = sp_arr[k][2];
  for (j=0; j<length; j++) {
    for (i=-width/2; i<= width/2; i++) {
      if (mim.u[z][y+j][x+i] == 255 && isFound == 0) {
        frameStart = k;
        isFound = 1;
```

```
      }
      if (isFound == 1)
        break;
    }
    if (isFound == 1)
      break;
  }
  if (isFound == 1)
    break;
}

int searchSpace[spim.zhi];
/* Initialize potential search space to all true.*/
for (i=0; i<spim.zhi; i++)
  searchSpace[i] = 1;

/* Add first candidate. */
cand[0] = sp_arr[frameStart][2];
limitSearchSpace(searchSpace, 0, frameStart+restrictionWindow);

int px,py,pz;                          /* plane coords.            */
float angle;                           /* angle of tile of plane.  */
signed long int acc;                   /* accumulator for intensity of voxels
                                          that the plane intersects. */
signed long int maxcut[spim.zhi];      /* max-cut array */

/* initialize maxcut array with max signed long. */
for (k=0; k<spim.zhi; k++)
  maxcut[k] = -2147483647;

for (k=frameStart+1; k<spim.zhi; k++) {
  x = sp_arr[k][0];
  y = sp_arr[k][1];
  z = sp_arr[k][2];
  /* tilt the plane by angle */
  for (angle=-M_PI/8.0; angle<=M_PI/8.0; angle+=M_PI/64.0) {
    acc = 0;
    float normal[3] = {0.0,0.0,1.0};
    rotateX(normal, angle);

    for (j=0; j<length; j++) {
      px = sp_arr[k][0];
      py = sp_arr[k][1]+j;
      pz = findPlaneCoords(normal, sp_arr[k], px, py);

      /* Out-of-bounds: just skip iteration. */
      if (pz < frameStart+1 || pz >= sim.zhi) {
```

```
        acc = -2147483647;
        break;
      }

      for (i=-width/2; i<= width/2; i++) {
        /* Skip non-vertebrae voxels. */
        if (mim.u[pz][py][px+i] != 0) {
          acc += 1;
        }
      }
    }
    maxcut[k] = MAX(maxcut[k], acc);
  }
  if (VFLAG)
    fprintf(stderr,"FRAME:%d, MAXCUT:%d\n",k,maxcut[k]);
}

/* Search for local maximas. */
for (i=1; i<=13; i++) {
  acc = -2147483647;
  isFound = 0;
  for (k=0; k<spim.zhi; k++) {
    if (searchSpace[k] == 1 && maxcut[k] > acc) {
      acc = maxcut[k];
      candIdx = k;
      isFound = 1;
    }
  }
  if (isFound == 1) {
    cand[i] = candIdx;
    if (((float) candIdx) > ((float)spim.zhi)*0.5) {
      limitSearchSpace(searchSpace, MAX(0,candIdx-20),
        MIN(spim.zhi,candIdx+18));
    }
    else
      limitSearchSpace(searchSpace, MAX(0,candIdx-restrictionWindow),
        MIN(spim.zhi,candIdx+restrictionWindow));
  } else {
    cand[i] = -2147483647;
  }
}

quicksort(cand,0,12);
int numMissed = 0;              /* number of local maximas missed. */
for (k=0; k<13; k++) {
  if (cand[k] < 0)
    numMissed++;
```

```
}

int missedDist, potentialCand, maxCand;
/* Extrapolate unidentified maximas. */
for (i=0; i<numMissed; i++) {
  missedDist = -2147483647;
  candIdx = 0;
  /* Search upper portions of vertebrae. */
  for (k=0; k<6; k++) {
    if (cand[k] < 0)
      continue;
    if (cand[k+1] - cand[k] > missedDist) {
      missedDist = cand[k+1] - cand[k];
      candIdx=k;
    }
  }

  /* Include searching for lower portions of vertebrae. */
  for (k=0; k<12; k++) {
    if (cand[k] < 0)
      continue;
    if (cand[k+1] - cand[k] > 35) {
      missedDist = cand[k+1] - cand[k];
      candIdx=k;
    }
  }

  /* Calculate midpoint of widest gap, and check nearby pixels of the mdpt. */
  potentialCand = (cand[candIdx]+cand[candIdx+1])/2;
  maxCand = potentialCand;
  for (k=potentialCand-5; k>= potentialCand+5; k++) {
    if (maxcut[k] < maxcut[maxCand])
      maxCand = k;
  }
  cand[0] = maxCand;
  quicksort(cand,0,12);
}

/* Check to see if there's a signficant gap in lower portion of the
   vertebrae. This is done to consider the lowest 12 center points,
   in case there are more than 12 columns in the CT image. */

numMissed = 0;      /* number of local maximas missed in lower region. */
for (k=0; k<3; k++) {
  if (cand[k+1] - cand[k] > 28)
    numMissed++;
}
```

```
  for (i=0; i<numMissed; i++) {
    candIdx = -1;
    /* Search upper portions of vertebrae. */
    for (k=0; k<3; k++) {
      if (cand[k+1] - cand[k] > 28) {
        candIdx=k;
      }
    }
    potentialCand = (cand[candIdx]+cand[candIdx+1])/2;
    maxCand = potentialCand;
    for (k=potentialCand-5; k>= potentialCand+5; k++) {
      if (maxcut[k] < maxcut[maxCand])
        maxCand = k;
    }
    cand[12] = maxCand;
    quicksort(cand,0,12);
  }


  if (VFLAG)
    for (k=0; k<13; k++)
      fprintf(stderr,"CANDIDATE ID:%d, Z:%d\n",k,cand[k]);

  fp = fopen(OVAL,"w");

  for (k = 0; k < 13-1; k++) {
    candIdx = (cand[k+1]+cand[k])/2;
    x = sp_arr[candIdx][0];
    y = sp_arr[candIdx][1];
    z = sp_arr[candIdx][2];
    fprintf(fp, "%d,%d,%d\n",x,y,z);
  }
  fclose(fp);

  VX3frameset(&sim,&rim);

//  VXwrite(VXout, rim.list);      /* write data                    */
  VXclose(VXout);                 /* close files                   */
  VXclose(VXin);

  exit(0);
}

/* Method to sort 1D-array in increasing order. */
void quicksort(int cand[], int first, int last) {
  int pivot,j,temp,i;
```

```
  if(first<last){
    pivot=first;
    i=first;
    j=last;

    while(i<j){
      while(cand[i]<=cand[pivot] && i<last)
        i++;
      while(cand[j]>cand[pivot])
        j--;
      if(i<j) {
        temp=cand[i];
        cand[i]=cand[j];
        cand[j]=temp;
      }
    }

    temp=cand[pivot];
    cand[pivot]=cand[j];
    cand[j]=temp;

    quicksort(cand,first,j-1);
    quicksort(cand,j+1,last);
    }
}

/* Rotates a 3D vector by <angle> degrees (in radians) about the X-axis. */
void rotateX(float arr[3], float angle) {
  float y = arr[1];
  float z = arr[2];
  arr[1] = cos(angle) * y - sin(angle) * z;
  arr[2] = sin(angle) * y + cos(angle) * z;
}

/* Finds plane with normal that intersects point, and then find z-coords at
   (x,y) on the plane. */
int findPlaneCoords(float normal[3], int point[3], int px, int py) {
  float intercept = (normal[0] * ((float) point[0])
    + normal[1] * ((float) point[1]) + normal[2] * ((float) point[2]));
  return (int) ((intercept - normal[0] * ((float) px) - normal[1]
    * ((float) py))/normal[2]);
}

/* Restrict search space for potential candidates. */
void limitSearchSpace(int searchSpace[], int lowerBound, int upperBound) {
  int idx;
```

```
  for (idx = lowerBound; idx <= upperBound; idx++)
    searchSpace[idx] = 0;
}

void VX3frameset(VisX3dim_t *is, VisX3dim_t *ir) {
/* VX3frameset: inserts frame markers in ir in the same location
   as in is

   This function assumes that both 3D images have the same number
   of images. If not, it will print an error and quit.
*/
    VisXelem_t *src, *dest;
    VisXelem_t *fptr,*sptr;     /* ptrs into src list */
    VisXelem_t *dptr;           /* ptrs into dst list */

    /* ensure at the beginning of each list */
    src=VXfirst(is->list);
    dest=VXfirst(ir->list);

    if ( VXNIL == (fptr = VXfind(src, VX_FRAME)) ) {
        /* No frames, don't do anything */
        return;
    }
    /* start searching after the first frame marker */
    sptr = src;
    dptr = dest;
    while(VXNIL != (sptr = VXfind(sptr, VX_BBX)) ) {
        if (VXNIL == (dptr = VXfind(dptr, VX_BBX)) ) {
            fprintf(stderr, "Error: Image count not equal!\n");
            exit(1);
        }
        fptr = VXbfind(sptr, VX_FRAME);
        /* Go back one element from BBX to add frame marker */
        dptr = VXaddelem(dptr->prev,VX_FRAME,"",fptr->size);

        fptr = VXfind(fptr, VX_EFRAME);
        /* Set pointer to the image (bbx, then image) */
        /* TODO: Implement variable number of element skipping */
        dptr = dptr->next->next;
        dptr = VXaddelem(dptr, VX_EFRAME, "", fptr->size);
        sptr = sptr->next;
    }
    if (VXNIL != VXfind(dptr, VX_BBX)) {
        fprintf(stderr, "Error: Image count not equal at end!\n");
        exit(1);
    }
}
```