18-551: Digital Signal Processing and Communication Design

Group 6: Get In My Belly!

Final Report

Geoffrey-Dixon Ernst

Sanghamitra Gogoi

Wendy Shutt

Introduction

While we were reviewing the past projects, we came across two projects, one in Image Segmentation and the second in Fourier Volume Rendering and it occurred to us the usefulness of a system which could read in a stack of images, process the images with the capability of choosing one organ and then being able to view a dissection of the organ at any chosen angle. This system has a great future in the field of medicine in improving and helping make a better medical diagnosis.

Past Solutions

Our first step was to look at the algorithms and any hardware and software the prior groups used for their projects.

The segmentation project was done by Group 5 of Spring 2004. Their project aimed to identify four major organs in an abdominal CT image scan and color each one a different color. They targeted the kidneys, liver, and spleen. In order to accomplish this task they used median filtering, the mean-shift algorithm, thresholding, and blob labeling.

The Volume Rendering project was done by Group 3 of Spring 2001. To summarize, they first read in images using a Fast Fourier Transform which performs a 3D forward FFT and returns a 3D object with frequency domain data. They then perform a Fourier Projection Slice Theorem which returns a 2D object also in frequency domain. The data is sent though the FFT function again to perform an Inverse 2D FFT. This now results in a volume rendered image.

Our Solution

Descriptions of algorithms:

Watershed

In the watershed segmentation algorithm, grayscale values of each pixel can be viewed as heights. This way, an image can be viewed as a topographical map. Now we will consider rain falling onto the image. When it collects, it will collect in the basins, or the areas with the lowest heights. Flooding will then occur in these areas. The overall goal is to "change the starting image into another image whose catchment basins are the objects or regions we want to identify" (Gonzalez, 418). Segments of the image are where distinct lakes form, thus areas/lines of high intensity become the edges of the segments. One disadvantage to this algorithm is that noisy material can lead to over-segmentation. Because of this, pre-processing or the merging of basins on certain criterion will be necessary afterward. Other approaches can be explored as well, such as the use of gradients, smoothing, and marker-controlled segmentation. Gradients prove useful because the gradient magnitude image has "high pixel values along object edges, and low pixel values everywhere else" (Gonzalez, 420). Smoothing the gradients can help as well, which can be accomplished by a series of opening and closing operations. Finally, marker-controlled watershed segmentation can be used. A marker, or a connected component belonging to an image, is desirable in each of the components of interest in the photograph, as well as one in the background of the image. The markers can be used to then modify the gradient image.

In our implementation of the watershed algorithm, we approximated the gradient using two Sobel edge detectors, one in the x direction and one in the y direction. Then we moved onto an indexing step in which each minimum was given a distinct value, allowing us to discern which blob a particular

pixel belonged to. In this step, you basically follow the path to the local minimum at each point. The reason why we decided to use gradients was to ensure that areas that someone looking at the original image would see as one distinct area was in fact detected by the watershed algorithm as one area. This is shown quite well in the pictures below. In the basic watershed (figure 2), areas that one would see as a single organ from figure 1 are split into 2, 5, or even 10 different regions by the watershed. By looking at figure 4 one can see that most, if not all, of this over-segmentation is avoided.

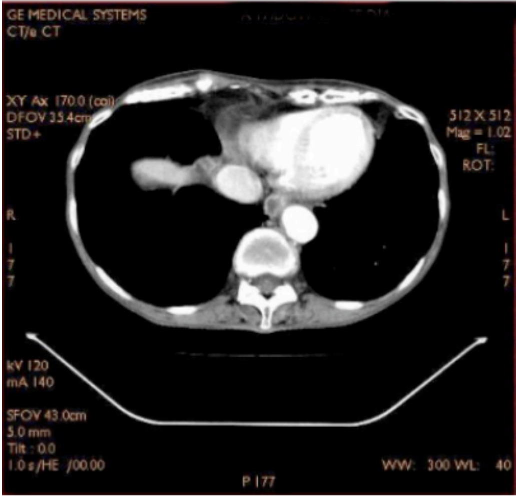


Figure 1. Original Image

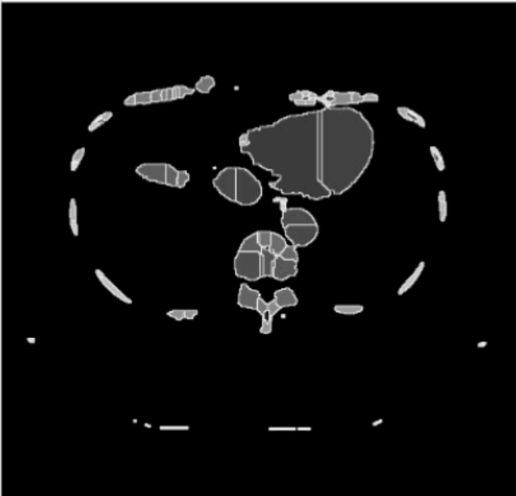


Figure 2. Basic Watershed

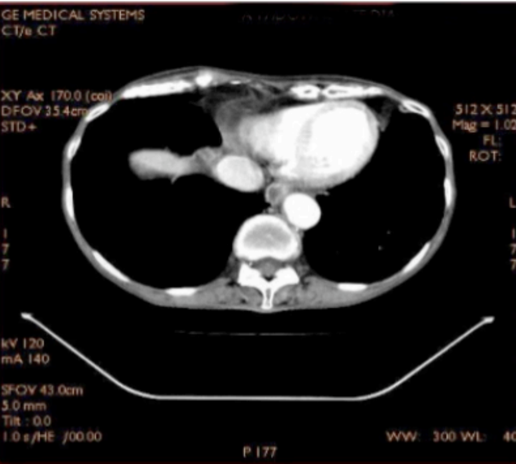


Figure 3. Same Original Image

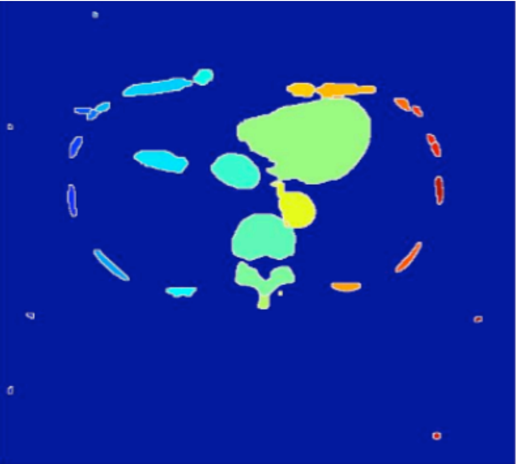


Figure 4. Gradient-based Watershed

Mean-shift

The mean-shift algorithm basically aims to segment the image into regions by setting a region that has similar color and intensity levels to the same level throughout. Any segment with a number of pixels that share a number of similar features will be grouped together by the algorithm and form clusters. There will be a densely populated center in the feature space. The feature space is basically comprised of the original image data, represented as the (x, y) location of each pixel, plus its color in the L*u*v* space (Pantofaru, 3). Once this feature space is obtained, the mean-shift filtering step can actually be applied. This entails finding the "modes of the underlying pdf and associating with them any points in their basin of attraction" (Pantofaru, 3). The next step is clustering. At this point, each data point in the feature space has been replaced by its corresponding mode. One method of clustering is to group together any modes which are less than one kernel radius apart, and merge their basins of attraction.

Volume Rendering & Interpolation:

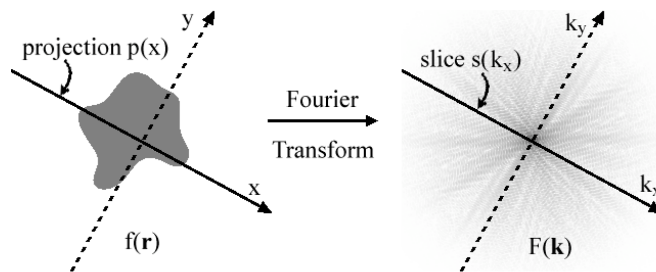
The objective behind the volume rendering part of the project is to take the data set of the images that have gone through segmentation and then ultimately generate a three-dimension model of the chosen organ.

The stack of segmented images that were created after the segmentation part will now go through the Hartley transform which is basically the modified version of the Fast Fourier Transform and considers real values exclusively, unlike the FFT which considers both real and imaginary values. This is an improvement since images only consist of real data and therefore we save up on processing power and memory space. The same butterfly structures used in DIT and DIF FFT algorithms can be applied to DHT to make it fast. The FHT is O(N logN) similar to FFT however, the FFT code for the DSK is highly

tuned and in ASM while we had to write the FHT code and so out FHT implementation is computed more slowly than FFT. However, real output means lower memory cost.

$$H_k = \sum_{n=0}^{N-1} x_n \left[\cos\left(\frac{2\pi}{N}nk\right) + \sin\left(\frac{2\pi}{N}nk\right)\right] \qquad k = 0, \ldots, N-1.$$

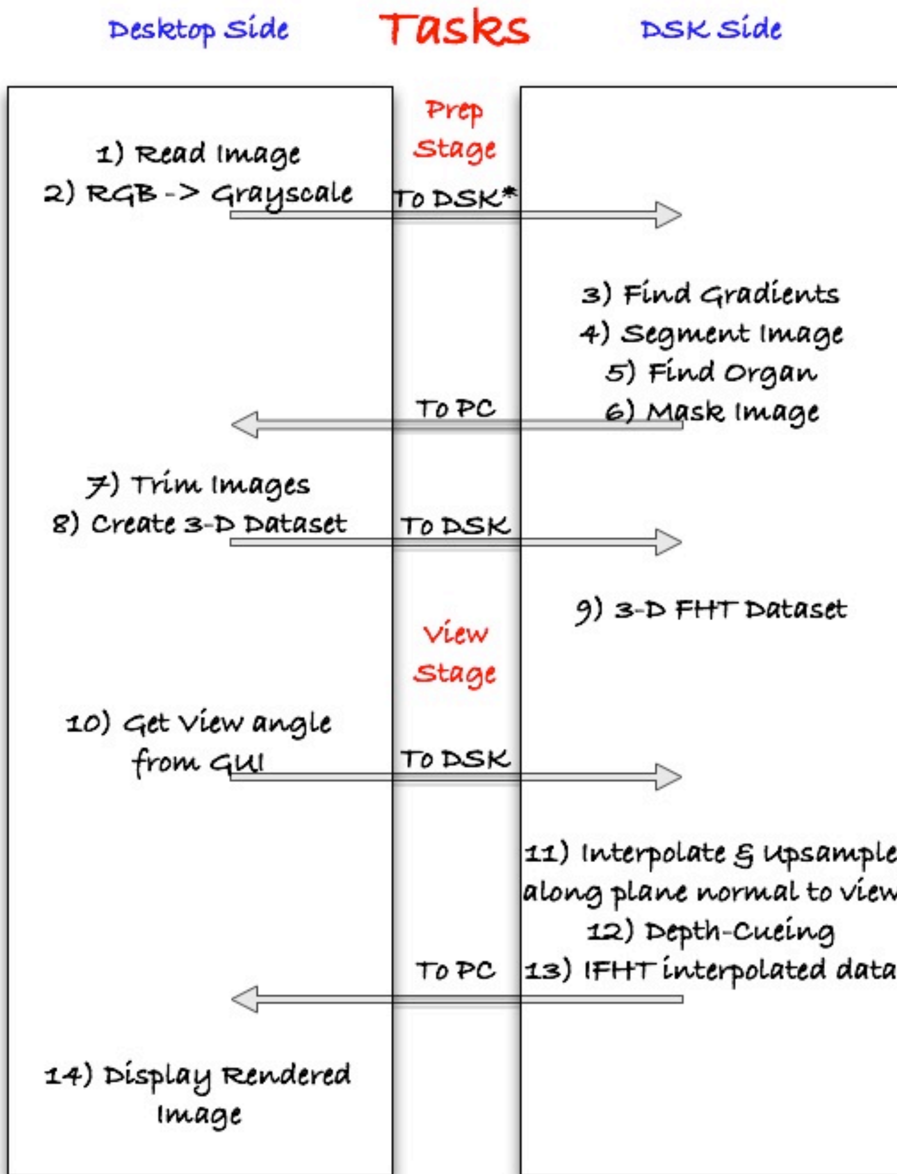
After this step, the images are then processed in the Slicer function, which implements the Fourier Slice theorem, with the desired view angle. The theorem says that a two-dimensional Fourier Transform of the 2D projection of a 3D object at an angle Θ is the same as a two-dimensional plane passing through the origin of the three-dimensional Fourier Transform (3D FT) of that three- dimensional object at an angle Θ. The Fourier Projection Slice Theorem is performed by taking 3D Fast Fourier Transform (3D FFT) on the data set and then the slice extraction is a simple matrix algebra cross product calculation where the perpendicular unit vectors on desired plane is found and is used as a basis to determine the remaining points on the slice.



Cubic interpolation attempts to reconstruct a continuous function from a set of discrete samples by fitting 3rd order polynomials (cubic functions) to the discrete samples. Cubic interpolation functions are piecewise defined over a finite extent; the function we used three different cubic polynomials. Interpolation evaluates the reconstructed continuous function at a new set of points, thus giving the

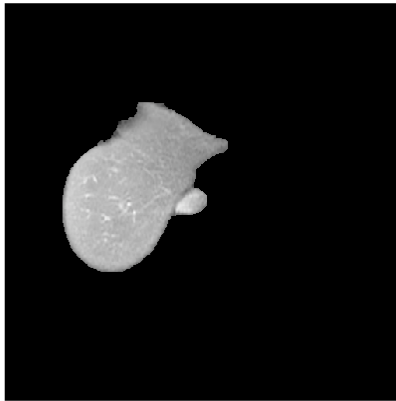
interpolated or re-sampled image. In effect, for each sample around each interpolated point, the polynomials of the interpolation function are evaluated; because these functions are often piecewise defined, a different cubic polynomial is used for each sample. We used an interpolation function was the C2-continuous, meaning that if the interpolation function was plotted, the function, as well as its first and second derivatives would be continuous. Interpolation functions of this variety were found to have optimal Fourier properties (meaning their most closely resembled the rect function of the ideal interpolator), and were also quick to compute.

Finally, an inverse 2D FFT is performed which will finally give the volume rendered image.



The DSK software includes many capabilities and performs many functions, especially above and beyond the previous DSK software used by the two previous groups we have built from. The DSK includes: network code to communicate with the PC-side user interface; an implementation of a

gradient-based watershed algorithm; an implementation of the Fast Hartley Transform (FHT) in 1-, 2- and 3-D; logic that identifies & masks an organ of interest from a abdominal slice (a medical image of an abdomen); and code to compile a set of slices (images taken in the same orientation, but at different places in the abdomen) into a single 3-D representation of an abdomen. While the two previous groups combined implemented all the functions our project implemented to a greater or lesser degree that, the only parts previously implemented on the DSK were a 1- and 2-D FHT and organ finding.



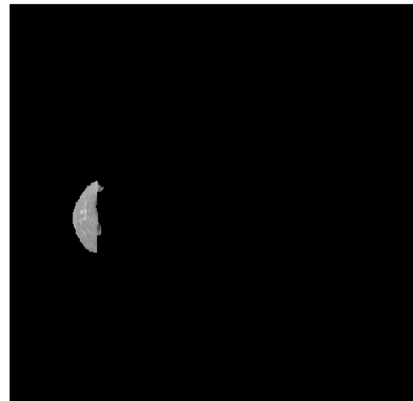
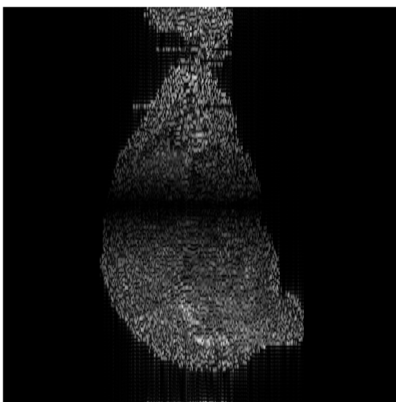


Figure 5 Figure 6 Figure 7

Sample Images after segmentation



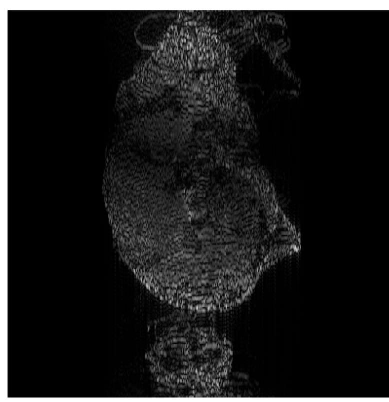
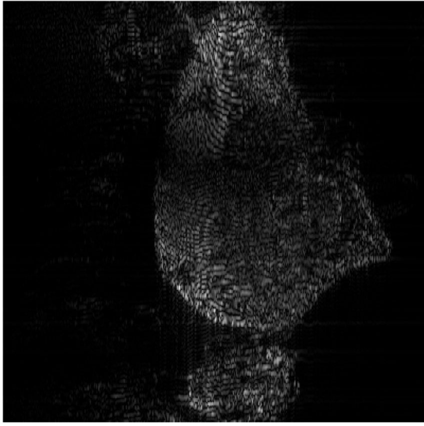


Figure 8 Figure 9



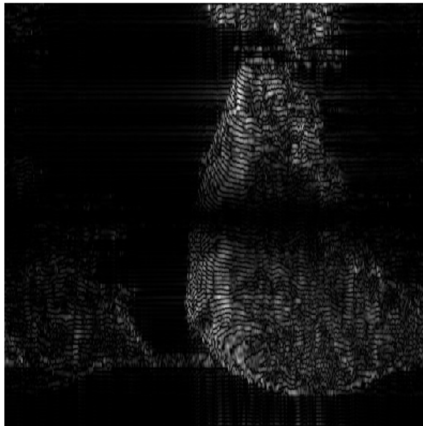


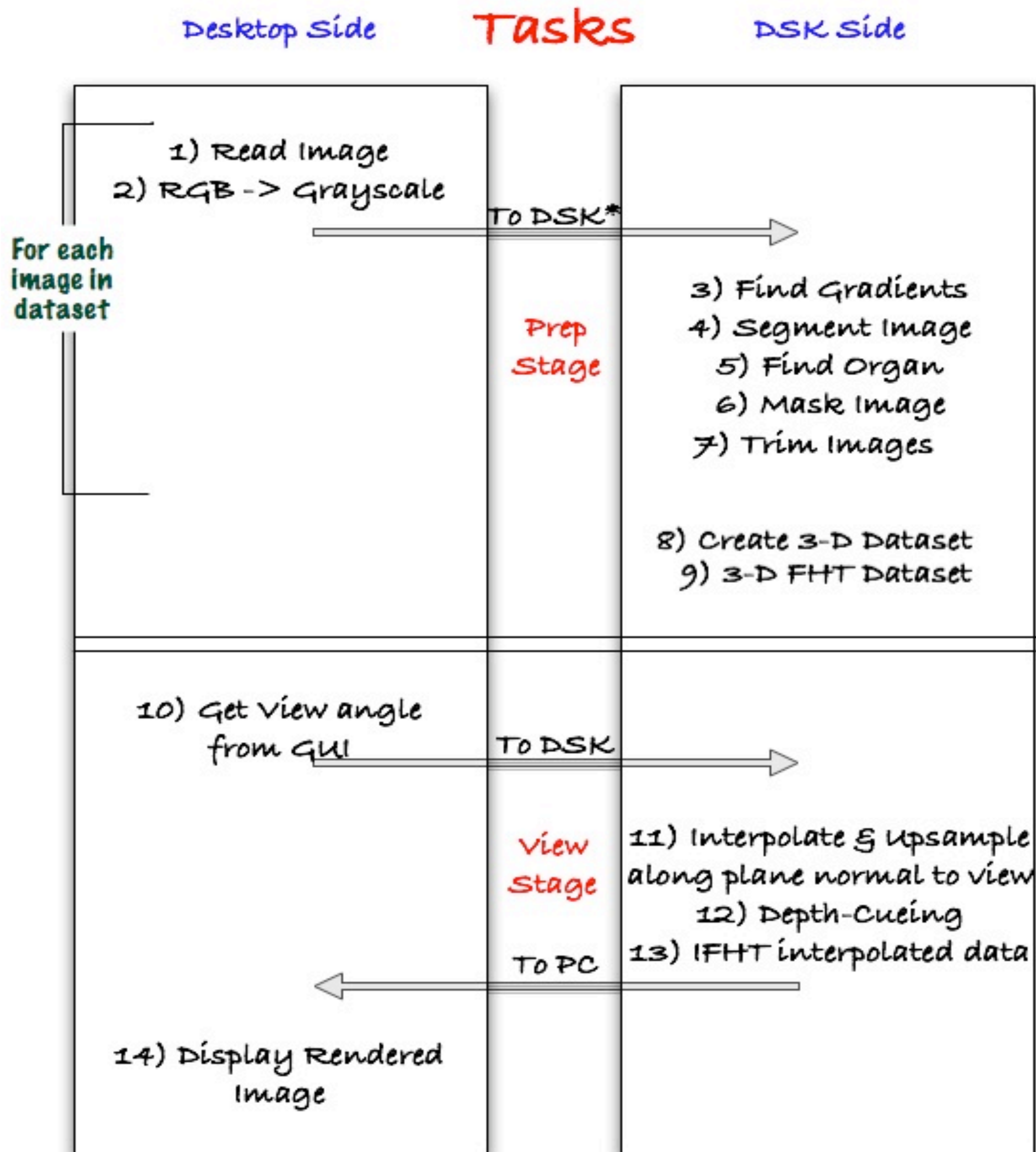
Figure 10 Figure 11

Images after Volume Rendering

Memory Issues

In our project, all of the 3-D dataset must reside in SDRAM. As well, we temporarily store some of the image buffers in SDRAM and use DMAs to shuttle blocks of data back and forth. All of our other data can be stored on chip, with most memory accesses being sequential. As well, many operations are performed on characters which amounts to 1.5 cycles/word or 0.375 cycles/ byte. For each of our input images, 120,000 bytes need to be copied from SDRAM onto the chip twice. Because of this we spend 67,500 cycles * 2 = 145,000 cycles paging data. However, for much of this, DMAs can be done in the background.

For our 3-D FHT the data resides off chip, with a non-sequential algorithm, meaning that it does not operate on contiguous bits of memory. This means that we cannot DMA all the data. The 3-D FHT performs $3N^2$ N-FHTs and can page in the data, with DMAs, for $2N^2$. The size of a DMA is $128* N^2$= 16384, which amounts to about 9216 cycles per DMA. The total number of cycles for moving data on and off chip, in the fast case, is 4,718,592. Then, a 1.5 cycle/word L2 to L1 cache transfer applies. As well, we have 7.125 cycles/word for non-contiguous transfers, which are direct CPU to SDRAM. This amounts to 14.25 cycles/word * 128 words = 1824 cycles/N-DHT. Then there are N^2 N-DHTs = 29,884,416 cycles for memory accesses. Finally, for our 2-D IFHT everything is on-chip and the array is 64KB, thus we need 24,567 cycles to accesses it, which is minimal.



The DSK software includes many capabilities and performs many functions, especially above and beyond the previous DSK software used by the two previous groups we have built from. The DSK includes: network code to communicate with the PC-side user interface; an implementation of a gradient-based watershed algorithm; an implementation of the Fast Hartley Transform (FHT) in 1-, 2-

and 3-D; logic that identifies & masks an organ of interest from a abdominal slice (a medical image of an abdomen); and code to compile a set of slices (images taken in the same orientation, but at different places in the abdomen) into a single 3-D representation of an abdomen. While the two previous groups combined implemented all the functions our project implemented to a greater or lesser degree that, the only parts previously implemented on the DSK were a 1- and 2-D FHT and organ finding.

Data Rates

Network Data Rates (network code was not stable between Java and DSK):

PC → DSK:

2.5 MB/s Estimated

1.42 MB/s initially measured (256 KB in 18 ms)

Need to send 15.36 MB, would take 10.74 seconds

DSK → PC:

10.0 MB/s Estimated

Were not able to make a measurement.

Send back minimal amount of data (~16 KB), would take 1.7 ms using estimated speed.

Calculation:

Gradient Calculation:

Estimate: 4 multiplies per 119301 pixels (and 16 additions) → 3 cycles/pixel = 357903 cycles / image = 1.6 ms / image = 203.6 ms total

Actual: 390283 cycles / image = 1.7 ms / image → 222.0 ms total (extrapolated from time for a single image)

Segmentation:

Estimate: 56 additions + 20 conditionals → will be really slammed in discontinuous memory accesses from conditionals → computations will be negligible in face of these, but estimate 14 cycles / pixel = 1680000 cycles / image = 7.5 ms / image = 955.7 ms total. This estimate is extremely optimistic.

Actual: 3875928 cycles / image = 17.2 ms / image = 2204 ms total (close to memory estimate)

Masking:

Estimate: Like segmentation, many conditionals → negligible computations → just those necessary to increment loop variables

Actual: 673027 cycles / image = 3.0 ms / image = 382.9 ms total (close to memory estimate)

3-D FHT:

Estimate: Assume FHT takes about as long as FFT = 1904 cycles for 128-FHT * 49152 128-FHTs = 93585408 cycles = 415.9 ms. This assumption does not take into account that 16384 of these FHTs cannot be moved on-chip.

Actual: 874892473 cycles = 3888.4 ms. This is so off because of the memory accesses and also because FHT is not as tuned as the FFT and so takes (much) longer than 1904 cycles/computation

2-D FHT (technically inverse):

Estimate: As before, same estimates, but need 256 128-FHTs = 487424 cycles = 2.2 ms. Same caveats apply.

Actual: 4018372 cycles = 17.8 ms. Same reasons for the discrepancy apply.

Interpolation:

Estimate: 36 multiplies / pixels → 18 cycles / pixels, 16384 pixels = 294912 cycles = 1.3 ms. Interpolation code is not completely memory-contiguous, so this estimate, which does not include memory rates, is slightly optimistic.

Actual: 492838 cycles = 2.2 ms.

Memory:

Gradient:

Estimate: All image data is on-chip, algorithm is memory-contiguous; ~1.5 cycles/pixel.

Segmentation:

Estimate: Instructions discontinuous thanks to conditionals. However, there are very few instructions in loop. However, data accesses are also somewhat discontinuous. Estimate roughly 4 cache misses / pixel.

Masking:

Estimate: Data is continuous. Instructions discontinuous. Negligible memory delay.

3-D FHT:

Estimate: 16384 continuous memory accesses, 16384 discontinuous L2 memory accesses, 16384 discontinuous off-chip memory accesses. Once cached, algorithm is memory-local.

2-D FHT:

Estimate: 16384 continuous memory accesses, 16384 discontinuous L2 memory accesses.

Interpolation:

Estimate — Same pattern of memory accesses (on average) as gradient, however, some orientations are continuous, while others are not. Estimate ~1.5 cycles/pixel.

Demo

The Java-DSK network code was not stable. Initially, it worked to a point, but as we continued to debug it, performance got worse and worse. This rendered the DSK useless, as we could not get data to it. Therefore, we demonstrated the entire project on the PC, but still using two pieces, a Java GUI and a C program that emulated the DSK. The C program running on the computer was the version of our software before we ported it to the DSK. Data transfer between the two was performed via files rather than potentially running into errors with the network again. The demo worked insofar that it generated a volume rendering of the 3-D dataset. The outline of our target organ, the liver, was visible from all viewpoints. However, the image was not as smooth and continuous as we were expecting; it closely resembled the output of the previous volume rendering group despite the use of a more complicated interpolation algorithm. We now postulate that the reason for the graininess of the output is the low resolution in the z-direction (i.e. the distance between slices); interpolating along the z-axis may yield better results.

Recommended Future Directions

We would recommend interpolating in the z-direction to maybe achieve a smooth, attractive picture and also possibly the implementation of depth-cueing after finding a suitable filter.

Semester Schedule

October

Week	Sun	Mon	Tues	Wed	Thurs	Fri	Sat
1							
2	19	20	21	22 Segmentation – Matlab code and research (Wendy)	23	24 Segmentation - Matlab code and research (Wendy)	25
3	26	27 Segmentation – C code (Group)	28	29 Volume rendering (Matlab) and memory estimations (Group)	30	31 Volume rendering (C code) and depth cueing research (Group)	1

November

Week	Sun	Mon	Tues	Wed	Thurs	Fri	Sat
4	2	3 Volume rendering clean-up and depth cuing (Geoff and Sangha)	4	5 Depth cuing and unit testing (Group)	6	7 Depth cuing and unit testing (Group)	8
5	9	10 **CHECK POINT**	11	12 Integration – All C	13	14 Integration and Port	15

				Code working (Group)		code to DSK (Group)	
6	16	17 Port to DSK (Geoff) GUI (Sangha)	18	19 Testing (Group) GUI (Sangha and Geoff)	20	21 Testing (Group) GUI (Sangha and Geoff)	22
7	23	24 Network Testing (Group)	25	26	27	28	29
8	30	1 Network Testing (Geoff)	2	3 **PROJECT DEMO (Group)**	4	5	6

December

Week	**Sun**	**Mon**	**Tues**	**Wed**	**Thurs**	**Fri**	**Sat**
9	7	8 **PROJECT WRITE-UP DUE**	9	10	11	12	13

Databases

We obtained three datasets from UPMC, two of which were used for testing and one was used for our demo.

References

Code References

- C Watershed:

  http://infocom.cheonan.ac.kr/~nykwak/kuim/index.htm

- C FHT

  http://www.geocities.com/ResearchTriangle/8869/fft_source.tar.gz

- JAVA mean shift

  http://rsbweb.nih.gov/ij/plugins/mean-shift.html

- Wrote code for interpolation, finding target organ, and masking image

Algorithm References

[1] Totsuka, T. and Levoy, M. 1993. Frequency domain volume rendering. In *Proceedings of the 20th Annual Conference on Computer Graphics and interactive Techniques* (Anaheim, CA, August 02 - 06, 0093). SIGGRAPH '93. ACM, New York, NY, 271-278. DOI=http://doi.acm.org/10.1145/166117.166152

[2] Bracewell, R. N. 1986 *The Hartley Transform* . Oxford University Press, Inc.

[3] Levoy, M. 1992. Volume rendering using the Fourier projection slice theorem. In *Proceedings of the Conference on Graphics interface '92* (Vancouver, British Columbia, Canada). K. S. Booth and A. Fournier, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 61-69.

(http://portal.acm.org/citation.cfm?id=155302&dl=GUIDE&coll=GUIDE&CFID=5058263&CFTOKEN=20296582)

Explanation of how to use Volume Rendering with the Slicer function

[4] Malzbender, T. 1993. Fourier volume rendering. *ACM Trans. Graph.* 12, 3 (Jul. 1993), 233-250. DOI= http://doi.acm.org/10.1145/169711.169705

Includes explanation of Fourier Volume Rendering

[5] Pantofaru, Caroline and Hebert, Martial. A Comparison of Image Segmentation Algorithms http://www.ri.cmu.edu/pub_files/pub4/pantofaru_caroline_2005_1/pantofaru_caroline_2005_1.pdf

Includes description of 3 segmentation algorithms : a graph-based approach, mean-shift, and a hybrid of the two.

[6] Gonzalez, Woods, and Eddins. Digital Image Processing Using Matlab, 2003.

Includes description of Watershed segmentation algorithm and basic Matlab code