

**PROCESS-LEVEL PARALLELIZATION  
OF  
SPATIAL REFERENCING**

By

Justin M. LaPre

A Thesis Submitted to the Graduate  
Faculty of Rensselaer Polytechnic Institute  
in Partial Fulfillment of the  
Requirements for the Degree of  
MASTER OF SCIENCE

Approved:

---

Christopher D. Carothers  
Thesis Adviser

Rensselaer Polytechnic Institute  
Troy, New York

March 2005  
(For Graduation May 2005)

Permission to reproduce this work, in part or in whole, is hereby granted.

# CONTENTS

List of Figures . . . . .	v
Acknowledgments . . . . .	vi
Abstract . . . . .	vii
1. Introduction and Historical Review . . . . .	1
1.1 Introduction . . . . .	1
1.2 Historical Review . . . . .	2
2. Spatial Referencing . . . . .	4
2.1 Lin’s Algorithm . . . . .	4
2.2 O’Neil’s Improvements . . . . .	7
2.2.1 Software Complexities . . . . .	8
2.2.2 Lock Contention . . . . .	8
2.2.3 Hyper-Threading . . . . .	8
3. Process-Level Parallelization of Spatial Referencing . . . . .	10
3.1 Algorithmic Improvements . . . . .	10
3.1.1 High Level High Priority Boxes . . . . .	11
3.1.2 High Level Random Boxes . . . . .	11
3.1.3 Low Level High Priority Boxes . . . . .	12
3.1.4 Quadrants . . . . .	13
3.2 Results . . . . .	14

4. Discussions and Conclusions . . . . .	19
4.1 Achievements . . . . .	19
4.2 Tradeoffs . . . . .	19
4.3 Initial Motivations . . . . .	19
4.4 Future Work . . . . .	20
4.5 Summary . . . . .	20
Literature Cited . . . . .	21
Appendices	
A. Software Modifications . . . . .	24
A.1 Low Level High Priority Algorithm . . . . .	24
A.2 Quadrants . . . . .	25
B. How to Build and Run the Software . . . . .	28

## List of Figures

1.1	A physician holding a lens over the patient's eye while performing the surgery. . . . .	1
2.1	An example mosaic. . . . .	5
2.2	An illustration of the progression of Lin's algorithm. . . . .	6
3.1	A flow chart of the Low Level High Priority algorithm. . . . .	12
3.2	Pseudocode for re-prioritizing the boxes. When a box has been traced, its priority is lowered. . . . .	13
3.3	Pseudocode for re-prioritizing the boxes. If a box has not been traced, its priority is its quality multiplied by its weight. . . . .	14
3.4	A flow chart of the Quadrant algorithm. . . . .	14
3.5	Four images from the quadrant algorithm. Notice they are all operating in their own corner of the image. . . . .	15
3.6	The number of boxes processed vs. the number of images that registered for the two algorithms of interest and the baseline. . . . .	16
3.7	Improvements of the two algorithms vs. the baseline. . . . .	17

## Acknowledgments

My biggest thanks should go to my advisor, Christopher Carothers. He had faith in me when I had lost faith in myself, and for that I shall forever be indebted to him. Thank you, Chris.

I would also like to thank Dr. Badrinath Roysam and Dr. Charles Stewart for giving me an opportunity to work on their project.

Additionally, I would like to thank Alex Tyrrell for his insight and guidance through this work as well as Richard O’Neil for laying the foundation for this thesis. I would also like to thank Omar Ahmed Al-Kofahi and Muhammad-Amri Abdul-Karim — our conversations were invaluable. I would also like to thank Dr. Gang Lin, for allowing me to modify the algorithm which he developed.

To my friends Angela, Chris, Dave, Greg, Katie, Mike, and finally my girlfriend, Deana, thank you. You have put up with far too much from me these past few years.

Lastly, I would like to thank my family (and my dogs!), for supporting me in these trying times. I know it was not easy, but thank you.

## Abstract

Laser retinal surgery is often used to treat vision-threatening diseases such as diabetic retinopathy and Age-related Macular Degeneration (AMD). The failure rate for such surgeries is approximately 50% [4, 8, 9, 12, 13, 14], both for the initial surgery as well as for each follow-up surgery. The spatial referencing algorithm, developed by Lin *et al.* [10] was developed to aid the physician responsible for those surgeries. Spatial referencing allows the surgeon to pinpoint the area over which the laser is aimed, thereby minimizing any errors during the procedure and decreasing the rate of failure.

In turn, the work of O’Neil [15] tried to further enhance the algorithm by parallelizing it. While his approach with two threads had some positive speedup, his approach with four threads suffered a drastic slowdown, ultimately performing worse than the non-parallelized version of the same code.

The aim of this work is to improve upon Lin’s algorithm, again by attempting to parallelize it, though taking a different approach than O’Neil. Our approach prefers *process-level* parallelism as opposed to *thread-level* parallelism for reasons that will be made clear in the coming chapters.

Additionally, some degree of a real-time deadline was imposed in this code, while in most of the cited literature it is not taken into consideration. Instead of a time-based deadline, however, we have opted for a computational deadline.

This thesis is motivated by the need to build real-time tools to aid physicians performing laser retinal surgery.

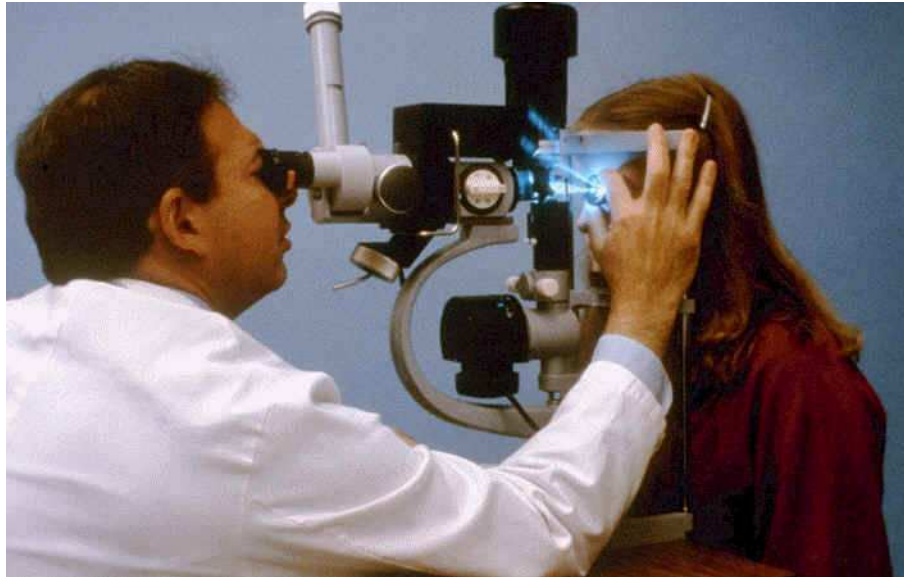
# CHAPTER 1

## Introduction and Historical Review

### 1.1 Introduction

Laser retinal surgery is often used as treatment for various forms of disease, such as Choroidal Neovascularization (CNV), Age-related Macular Degeneration (AMD), degenerative myopia, and diabetic retinopathy, for example. However, failure rates are high due to the capacity of the eye to make rapid movements. The reaction time of even the fastest of surgeons is orders of magnitude slower than the speed at which the eye can move.

The current procedure involves the physician holding a focusing lens over the area to be treated, aiming the laser with a joystick, and firing with a foot pedal — clearly this procedure is error-prone (figure 1.1). Using computers to aid the physician is the logical conclusion, and is the basis for our group’s ongoing work. Concerted



**Figure 1.1:** A physician holding a lens over the patient’s eye while performing the surgery.

efforts are being made to ensure the safety of the patient and his vision.

The first step is to simply create a device which keeps track of the position of the eye with respect to where the laser is being fired. Should the laser drift away from the target area, the laser should immediately turn off. The next step would be to further remove the physician from the equation, and allow the computer to control where and when to fire the laser — undoubtedly this goal is still years away.

The goal of this thesis is to express the impracticality of using thread-level parallelism in this domain. Instead, we choose to introduce process-level methods for parallelization of spatial referencing. The benefits to this approach will be elaborated upon in coming chapters.

## 1.2 Historical Review

Many of the references cited in this thesis are from earlier members of this group, including Balduf, Becker, Can, Lin, O’Neil, Shen and Tyrrell. Their work collectively is the intellectual foundation of this thesis.

The work of Becker [2, 3] was the catalyst that generated interest in the realm of retinal laser surgery within our group. Becker realized that the algorithms were challenging due to their conflicting goals of accuracy, speed, and robust operation in spite of poor image quality.

Following that was Balduf’s attempt to parallelize the work of Becker [1], the first of many attempts to speed up registration through the use of multiple processors. Balduf’s results were promising; though his performance gains were not enough to make real-time control of retinal laser surgery feasible, this work encouraged further investigation through O’Neil as well as the author.

Next was Can’s improvement to tracing [5] through the use of exploratory algorithms, in which the tradeoff between optimal results and computation time was made in favor of shorter computational time. Additionally, Can [6, 7] introduces a new hierarchical algorithm for registering a pair of images using a 12-parameter

transformation with as little as 20% overlap.

The next generation of registration algorithms is based on the work of Shen *et al.* [16, 18]. While prior registration algorithms would prefer to register once and then track retinal movements for as long as possible, this new group would *always* start from the beginning and find the retina's position in absolute terms. This newer form of the algorithm was always more attractive to the author due to the fact that errors would not compound over time. Another important point from Shen's work was that optimal scheduling could only be achieved with perfect hindsight, and is therefore impossible [17]. It is also of note that until Shen, all of the previous work had been done on SGI Workstations, such as Indys, O2s, and Octanes, all of which are fairly expensive. Shen moved the algorithms to lower-cost commodity hardware, such as the Intel Pentium line.

We now find ourselves at the current iteration of this work, that done by Lin *et al.* [10], which is the basis for this thesis. Using hierarchies of progressively smaller and smaller grid boxes, the algorithm recursively attempts to find landmarks, and ultimately, constellations (groups) of landmarks. O'Neil [15] attempted to make Lin's algorithm run in parallel with multiple threads, but met with limited success due to the complexities of the software in which the code was written.

## CHAPTER 2

### Spatial Referencing

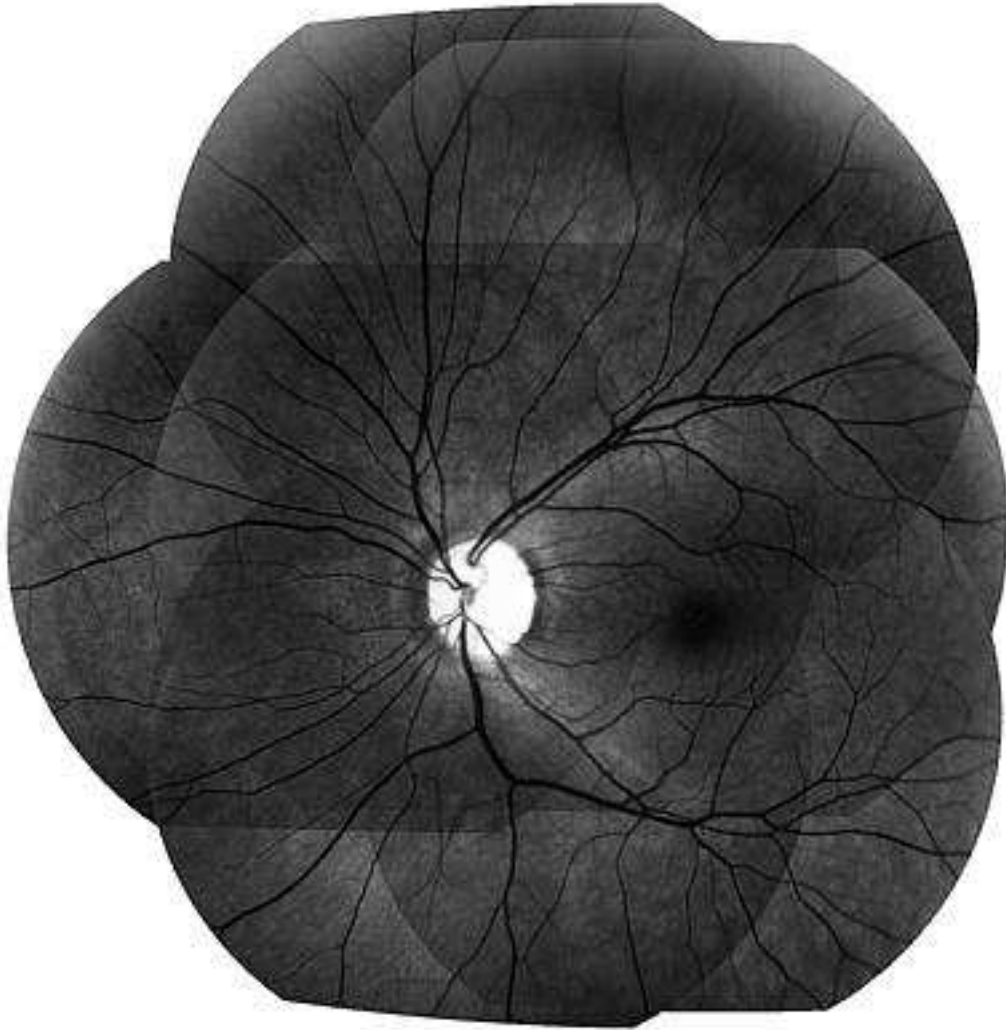
The next few sections will give a general overview of Lin's spatial referencing algorithm, followed by a brief outline of the work of O'Neil. Next, some failed attempts to improve the algorithm will be covered. The next chapter will discuss the author's improvements to the spatial referencing algorithm.

#### 2.1 Lin's Algorithm

Before the algorithm can even be used, an off-line spatial map must be created and verified (figure 2.1). This precomputed map is used to store as much information as possible to make the on-line spatial referencing fast. Spatial maps are generated from high-quality still images of the retina that will be undergoing the surgery.

Once the off-line spatial map is created, the on-line exploratory algorithm can be used. The exploratory algorithm has five steps:

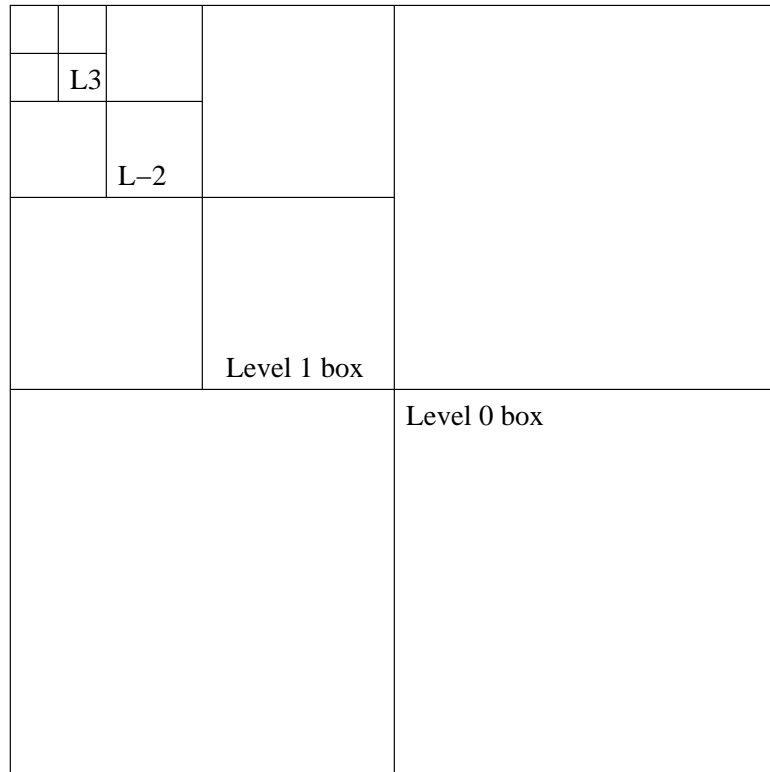
1. Grid analysis and seed point detection: a grid of evenly spaced horizontal and vertical lines are surveyed across the image. Local intensity minima crossing these grid lines are interpreted as blood vessels and the location is stored. These locations are known as seed points.
2. Pre-emptive tracing: starting at a seed point, the algorithm traces along the pair of parallel edges until it finds a bifurcation in the vessel. The bifurcation is known as a landmark, and its location is stored. Tracing is pre-empted when a landmark constellation is found.
3. Indexing: because of the fact that landmark constellations are only approximate at this stage in the algorithm, they are considered quasi-invariant. The



**Figure 2.1: An example mosaic.**

QIFV, or quasi-invariant feature vector, is computed, finding the five closest constellation matches. Of these, the two best are refined and verified immediately.

4. Refinement and verification: the location of the landmark at this point is only approximate. Using the bifurcation and crossover locations, the actual location can be determined down to sub-pixel accuracy. Each constellation match corresponds to a set of landmarks, which are used to generate an initial transformation to the closest off-line diagnostic image. If the match is considered



**Figure 2.2: An illustration of the progression of Lin’s algorithm.**

too inaccurate, the mapping is rejected another match is considered.

5. Termination: the algorithm repeats steps 2–4 until a constellation has been found and globally verified or until the priority queue is out of boxes.

The most important contribution of Lin’s work, however, is the prioritization of which grid boxes to trace. Lin’s algorithm routinely predicts the best grid boxes to explore approximately 90% of the time on video-stream quality data, which as we will see is not nearly of the quality used in Lin’s study [10]. This typically outperforms Shen’s [17] priority scheme, which often predicts good locations to search for individual landmarks, but is not as effective at finding constellations.

The core of this improvement comes from the decision to use hierarchical grid analysis. First, it divides the image into (typically)  $4 \times 4$  grid boxes. These are the coarsest level. It then prioritizes these boxes based on the number of seed points

detected at their borders and on  $q(b)$ , where  $q(b)$  is defined as

$$q(b) = \sum_{i=1}^N s_i - \left\| \sum_{i=1}^N s_i [\cos 2\theta_i, \sin 2\theta_i] \right\| + N$$

Let  $N$  be the number of seeds,  $b$  be the grid box,  $s_i$  denotes the strength of each seed point, and  $\theta_i$  be the estimated blood vessel tangent direction at each seed.  $q(b)$  is known as the strength-weighted angular diversity measure, and it is  $q(b)$  that we use to prioritize the grid boxes. Grid boxes are recursively subdivided down to a configurable level, and tracing only occurs at the finest (deepest) level. One priority queue is associated with each level.

## 2.2 O’Neil’s Improvements

O’Neil took the algorithm developed by Lin, described in the previous section, and attempted to multi-thread it. This attempt was partially successful, however it did not yield the performance desired.

O’Neil’s approach was basically to bisect the image along the X-axis when using two threads, and to split the image up into quadrants when using four threads. 80 pixels of overlap were allotted on all boundaries.

The median speedup using two threads was 41.73% with an average speedup of 33.95%. However, these results were taken from the modified code. The modified code had a median slowdown of 153.66%, and an average slowdown of 117.28% with respect to the unmodified code. When using the modified code on four threads, the median speedup was -55.02% and the average was -50.31%. In other words, the four-way threaded code took *longer* to finish than even the single-threaded version. There are several possible reasons for this, which will be elaborated upon in the following sections.

### 2.2.1 Software Complexities

VXL, or the Vision *Something* Library, is an extremely complex software library. With it, one can develop GUI-based imaging programs, video manipulation software, or even retinal laser surgery applications. The robustness of the library is burdened with untold layers of software indirection, however, these can be contained.

The probable source of problems for O’Neil was the numeric library in VXL. It was originally written in Fortran, though instead of re-writing it in C or C++, which is the language of choice for the remainder of the software, it was instead simply converted to C using a program like `ftoc`. Unfortunately, to be on the safe side, the converter made all of the local variables `static`. `static` variables are hardly ever a problem until multi-threading comes into the picture, and then it becomes a very big problem. There were literally thousands of `static` keywords across hundreds of files. Though an attempt was made to do a blind search and replace, ultimately that proved fruitless — apparently at least some of the `static` keywords *are* necessary.

### 2.2.2 Lock Contention

O’Neil’s attempt to use as few locks as possible was definitely a good decision. The numeric library, as described above, had to be locked before any portion of it could be used. Unfortunately, the math library is used extensively in this code base and the performance is therefore significantly degraded. Additionally, his occasional crash was caused by an unknown race condition that was never found.

### 2.2.3 Hyper-Threading

While the operating system in question (Linux) believed there to be four distinct processors for it to schedule, in reality the machine O’Neil was using had only two. The reason that the OS believed there to be four was that these processors were *hyper-threaded*. This simply means that one processor is capable of running two processes at one time, though it is merely an illusion. Intel has shown some speedup using these processors, but it is only under specifically-tailored workloads

that any speedup is gained. The 2.4 Linux process scheduler was considered naive with respect to hyper-threading — it was not until version 2.5 that a more intelligent approach was devised to take advantage of hyper-threading.

What most likely happened was that all four processors were making data requests on the bus, thereby overloading it and causing a slowdown across all processors. Hyper-threading may be appropriate for certain multimedia applications, however its limitations make it inappropriate for the application at hand.

## CHAPTER 3

### Process-Level Parallelization of Spatial Referencing

This chapter introduces improvements to the algorithm developed by Lin [10] that overcomes some of the shortcomings of O’Neil’s work [15].

#### 3.1 Algorithmic Improvements

Originally, the author was working on the same hardware O’Neil had been working on. Recently, however, the machine would inexplicably hang. The decision was made to move from the dual hyper-threaded Xeon to the new four processor 2.2 GHz Opteron with 16 GB of RAM.

After many trials at threading the application, it was decided that the software was far too complex to be well-suited to threading. Instead, in place of threads, processes were used. Processes were decided upon for several reasons. First, they are encapsulated in their own address space — no single process can overwrite another. This relieves the necessity for locking. Second, in the event that inter-process communication is necessary, System V IPC [19] is more sound and solid than any threading implementation, as well as more cross-platform.

Also worth note is the fact that the code base was changed midway through the investigation. Some of the files had been modified over and over to suit the needs of whoever was working on it at the time. This proved less than optimal due to the fact that the author had to not only read and understand the original code base, but all of the modifications layered on top of it, such as the work of Tyrrell *et al.* [21]. It was decided to instead revert back to a simpler version of the code, one that only operated on a single image at a time instead of a stream of images, thereby making the need for System V IPC unnecessary.

It should be noted that for all of the implementations below, no attempt was made to prevent the individual processes from working in the same area of the image. That may involve locking to some degree, which we would like to avoid. Some of the algorithms below exhibit more overlapping than others. Ideally, an algorithm would never do the same work twice, and that is what we strive for. The last two algorithms perform fairly well in this respect, although sometimes the overlap may be significant.

Of course, whenever dealing with multi-threading or multiple processes, it is always possible to get more than one result. We can choose the best result returned by comparing the average or median errors of the results. Typically, the median error is used. Again, System V IPC could be used to decide which registration had the least amount of error, then continue on with that information.

### **3.1.1 High Level High Priority Boxes**

Initially, attempts were made to parallelize the highest level (smallest, most refined) boxes. This was a sub-optimal approach, however, due to the fact that the next highest priority box would be adjacent to the current box, thereby clustering all of the computational work in a single region. The tracing would ultimately overlap, and little would be gained from having four processors work in the same area.

### **3.1.2 High Level Random Boxes**

The next logical step was to choose non-adjacent boxes. This approach yielded the ability to successfully locate landmarks, although similar to the work of Shen [17], could not often construct constellations from them. This path was therefore abandoned as well.

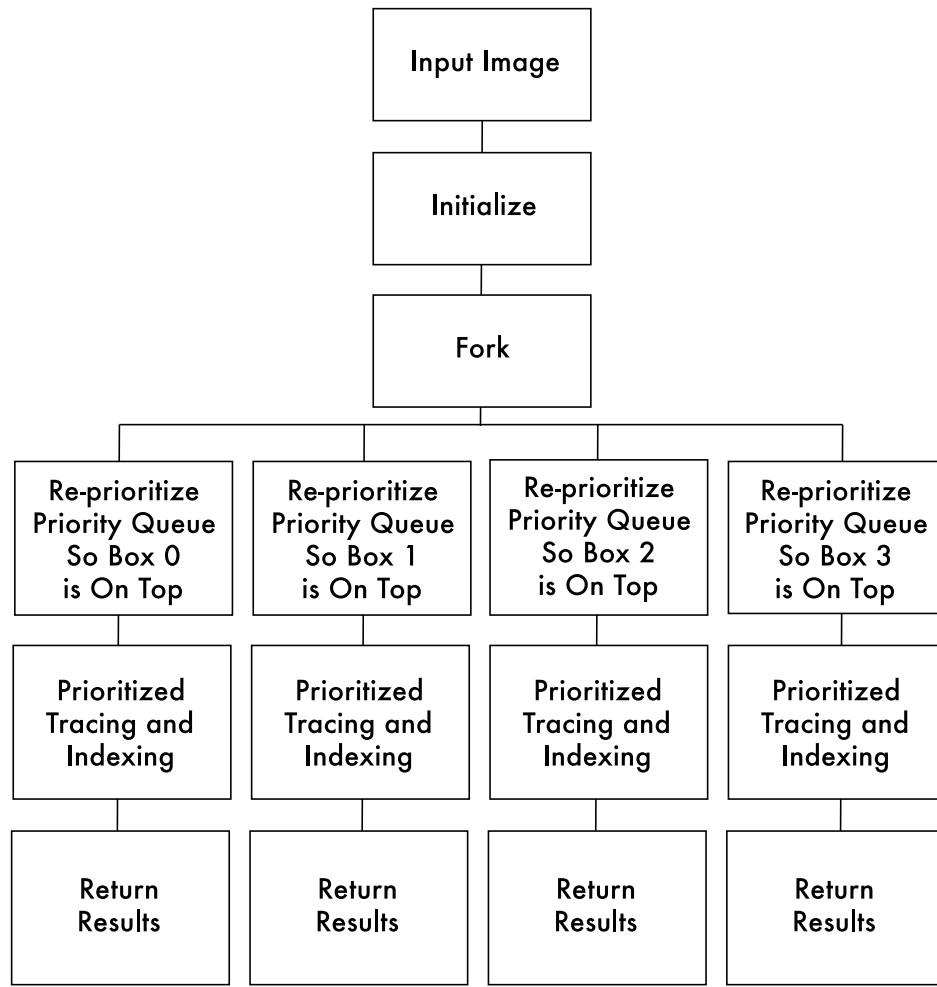


Figure 3.1: A flow chart of the Low Level High Priority algorithm.

### 3.1.3 Low Level High Priority Boxes

Following that, we basically replicated Lin’s algorithm for each processor. In other words, we would take the four highest priority low level (largest, least refined) boxes and run the hierarchical spatial referencing algorithm (figure 3.1). This is essentially the same as running the unmodified algorithm four times independent of one another. The only difference is the re-prioritization of the boxes after the call to `fork`.<sup>1</sup> After the `fork`, there are now four copies of the process, all with priority

<sup>1</sup>After `forks`, the address space is identical in every way. However, once a change is made in any of the `forked` processes, the page holding the changed data is replicated and then modified. This is known as “Copy-On-Write”, or COW, and more details can be found in most textbooks on operating systems.

queues identical to their parent. We change the priorities by pushing all unwanted boxes to the bottom of the priority queue by marking them as traced, which in turn lowers their priority. See the pseudocode for re-prioritization in figure 3.2.

This version of the algorithm gave exceedingly good results. However, one might consider using O'Neil's approach and just give each processor four low level boxes. This is, in fact, what we do and it is described in the next section.

```
for  $i < 4$  do  
  if  $i == \text{current process}$  then  
    continue ;  
  else  
     $\text{box}[i] \rightarrow \text{has traced} = \text{true}$  ;  
  end  
end  
 $\text{heapify}(\text{boxes})$  ;
```

Figure 3.2: Pseudocode for re-prioritizing the boxes. When a box has been traced, its priority is lowered.

#### 3.1.4 Quadrants

This is the last step in the progression of algorithms pertaining to this thesis. Since there were  $4 \times 4 = 16$  boxes to divide by four processors, each processor gets four boxes. Intuitively, we give each processor the four boxes corresponding to one particular corner. This basically gives us the same overall layout that O'Neil was striving for without the race conditions (figure 3.4). Once again, the program **forks** and we have four identical priority queues. However, this time we want each process to trace one quadrant, and must therefore raise the priorities of four boxes making up that quadrant. For all of the boxes in our quadrant, we elevate both the **quality** and the **weight** by 100, which are then multiplied together to determine its priority. We elevate them instead of setting them to 100 because, to some degree, they have already been prioritized, and we would like to not discard that information. In theory, these values could be higher than 100, but in practice their result is never above several hundred. See the pseudocode for re-prioritization in figure 3.3.

```

foreach box in Quadrant do
  box->quality += 100.0 ;
  box->weight += 100 ;
end

```

Figure 3.3: Pseudocode for re-prioritizing the boxes. If a box has not been traced, its priority is its quality multiplied by its weight.

## 3.2 Results

Out of 944 retinal images, the largest number of images that were ever successfully registered was 876 (without any computational limits on the number of boxes) using the algorithm described in 3.1.4. Keep in mind, once again, that these are *video-*

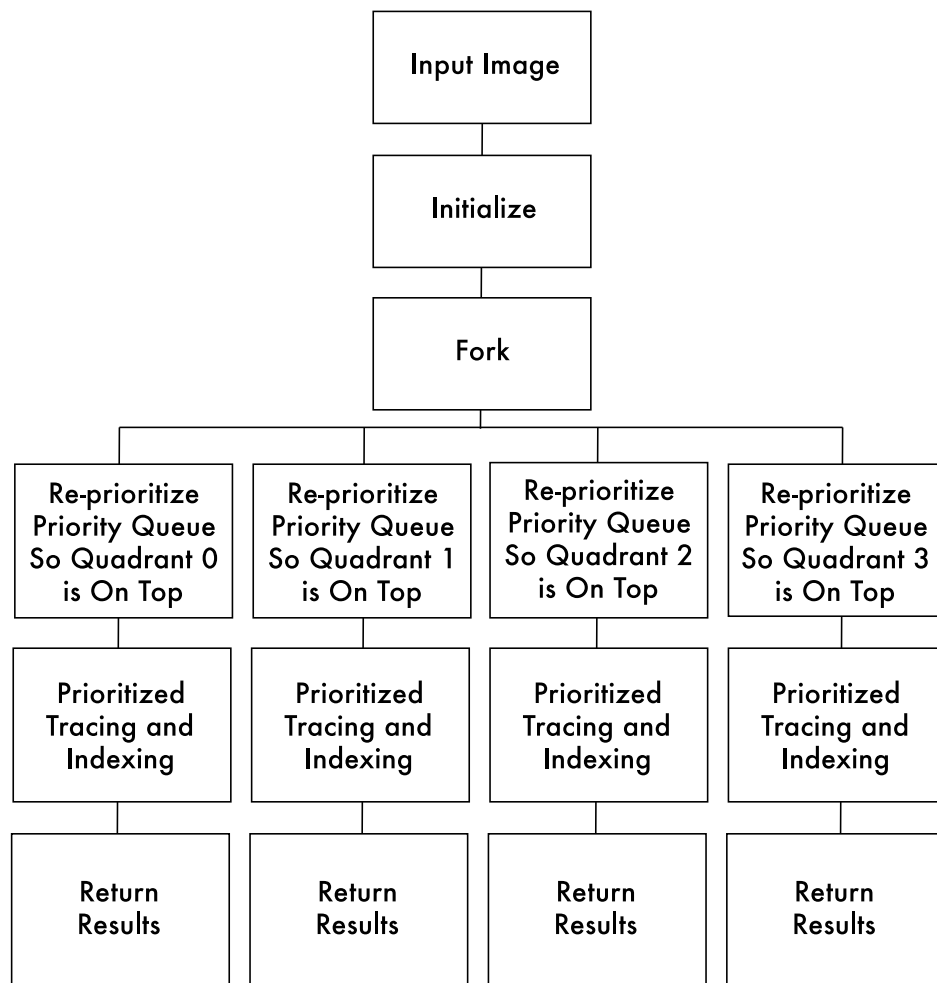
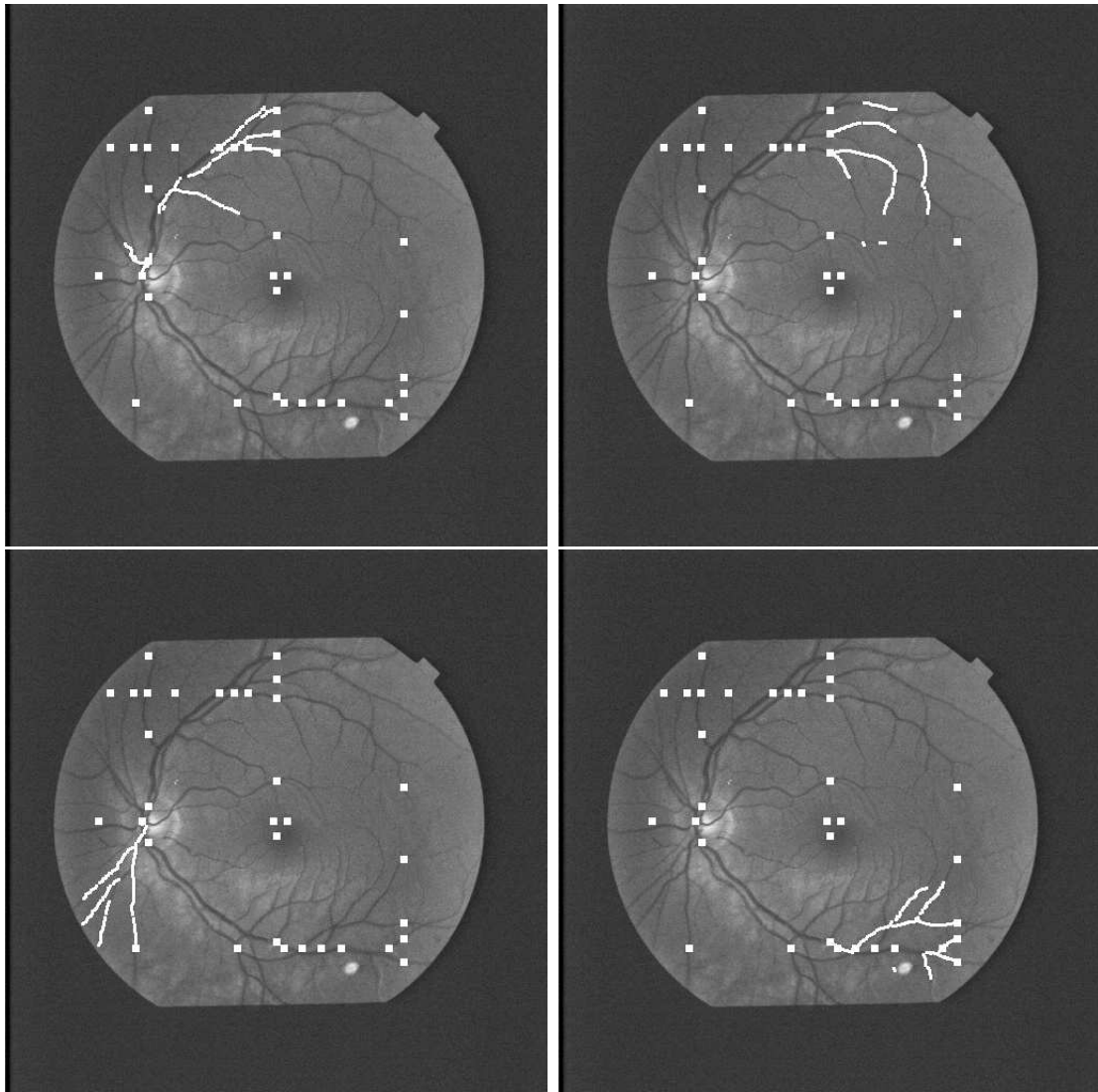


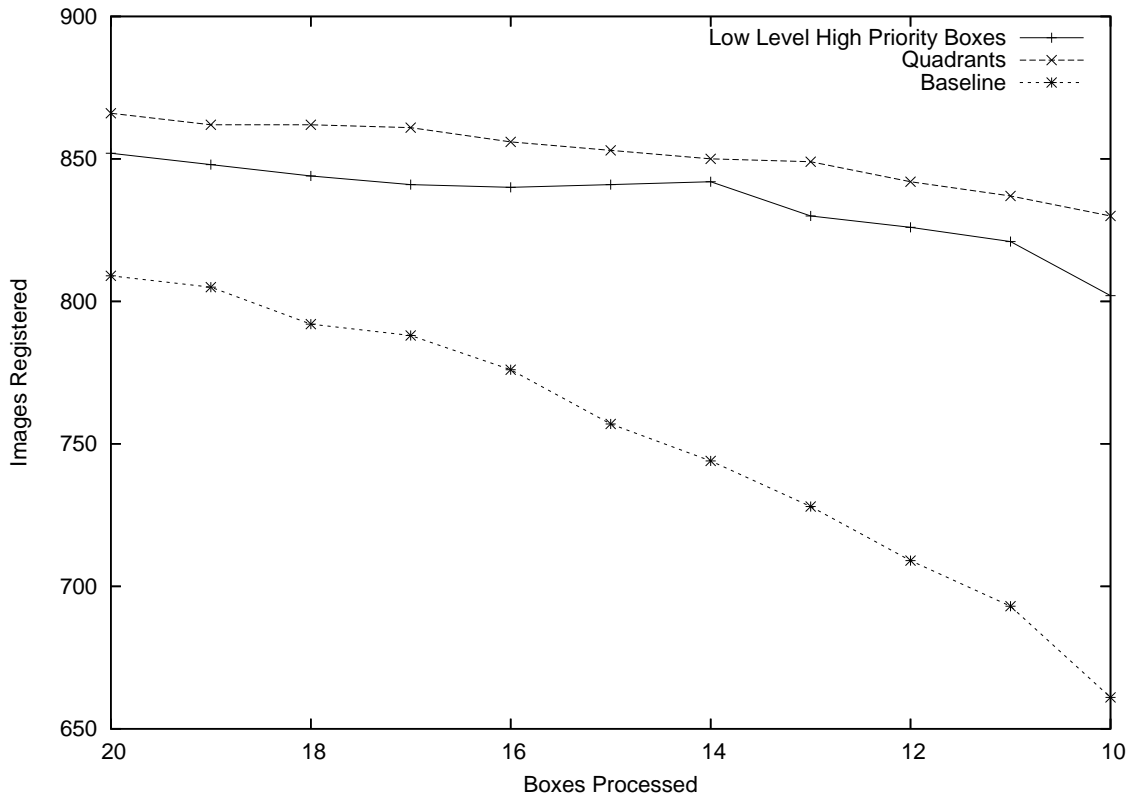
Figure 3.4: A flow chart of the Quadrant algorithm.



**Figure 3.5: Four images from the quadrant algorithm. Notice they are all operating in their own corner of the image.**

*quality* images as opposed to retinal *photographs*, and therefore the quality is far less. The 68 retinal images which never registered suffered from motion blur, which is a limitation that Lin's algorithm could not overcome [22].

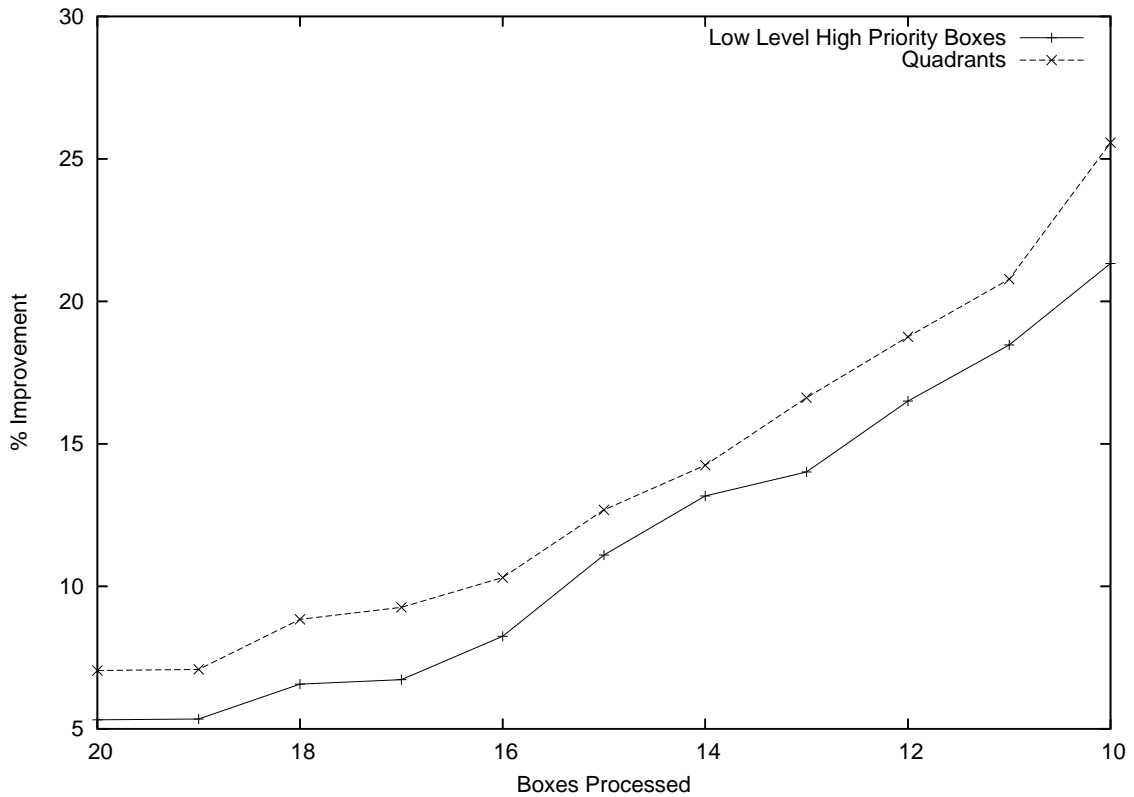
While the number of boxes in figure 3.6 range from 20 to 10, we are primarily interested in the number of images which registered using only 10 boxes. Ideally, all of our registration work would complete within the 10-box limit. The rest of the data is for comparison purposes only.



**Figure 3.6: The number of boxes processed vs. the number of images that registered for the two algorithms of interest and the baseline.**

Of the 876 images that could possibly be registered, only 661 of them could successfully register within the 10 box tracing limit using the unmodified, single-threaded version of the code. For this test, this was the only modification that was made. These values will be used as our baseline for performance.

With the algorithm described in section 3.1.4, 830 images successfully registered, showing a 25% improvement over the unmodified code with a 10 box limit. The algorithm described in section 3.1.3 also performed well, yielding a 21% improvement over the baseline code (figure 3.6). In figure 3.7, one can view the improvements vs. the baseline performance. It is evident from this figure that the improvement is inversely proportional to the number of boxes allowed to be processed. This explains why parallelization is well-suited to this application.



**Figure 3.7: Improvements of the two algorithms vs. the baseline.**

We can clearly see that the number of registrations in the baseline algorithm decreases linearly with the reduction in the boxes that are allowed to be processed. The fact that the baseline algorithm is suboptimal across the length of the graph is important, however, the more important feature to notice is the rapid decrease in the performance of the baseline algorithm. While the proposed algorithms decrease slightly with a reduced allotment of boxes, the rate at which the baseline algorithm decreases is startling. It drops to approximately 75% of the images when 10 box quotas are used, which is a rather alarming statistic. The proposed algorithms, meanwhile, maintain over a 90% registration rate despite the same box constraints.

Worth noting is the “knee” in the graph of the low level high priority boxes. This is a side-effect of the `while` loop — sometimes it directly exits the loop without determining whether the image registered, though if the `while` condition fails, it

always checks before returning.

## CHAPTER 4

### Discussions and Conclusions

#### 4.1 Achievements

The quality of the results is of no importance if they are computed after the real-time deadline. Sometimes it is necessary to take the suboptimal but computationally advantageous route. While the algorithm developed by Lin is quite cleverly designed, improvements from parallel computing can be achieved. While limiting the number of boxes we operate on per process, we can divide up all of the boxes between the four processors, thereby yielding a noticeable improvement. This thesis has demonstrated two algorithms that suit this purpose, both of which have improvements above 20%.

#### 4.2 Tradeoffs

As figure 3.6 shows, computing a greater number of boxes yields images that register more often. Approximately 95% of the images register using 10 boxes, while nearly 99% of the images register when using a 20-box quota. A 4% increase in productivity does *not* merit a doubling of the tracing effort in the author's opinion.

#### 4.3 Initial Motivations

This thesis evolved from some different, though related, goals and intentions. Initially the author was driven to enhance the real-time capabilities of the code base. To that end, the new real-time functionality in the 2.6 Linux kernel was investigated and subsequently implemented. This allows normal, user-land processes to be given real-time priorities under the Linux scheduler, which depreciates the techniques presented in Tyrrell *et al.* [21].

Additionally, it was thought that the processor-swapping, common among CPU-intensive jobs under the 2.4 kernel, was hurting our performance. The 2.6 Linux kernel added *processor affinity* [11], the ability to restrict a job to a specific subset of processors. The speedup gained from this was negligible, though it did help to make our results more predictable.

However, once our goals changed, we transitioned to newer hardware as well as the original code base. These new enhancements were not re-implemented.

#### **4.4 Future Work**

Applying these algorithms to streams of data is the next logical step in developing a prototype for computer-assisted retinal laser surgery. Using these enhancements to Lin's algorithm would, due to the use of process-level parallelism over thread-level parallelism, require System V IPC [19]. Communications between the processes would require at a bare minimum shared memory and semaphores. This would allow the processes that completed to elect the "best" solution to use and continue on based upon that registration.

#### **4.5 Summary**

This thesis has displayed the capability of Lin's software to be applied to a parallel computing paradigm. Improvements were observed and tradeoffs were made with regard to quality and computational quotas. Two algorithms were presented which resulted in more image registrations than produced by the baseline code: one using high priority low-level boxes, the other using quadrants. Both improved upon the number of images registered by the baseline code by more than 20%, bringing us closer to building real-time retinal laser surgery tools.

## Literature Cited

- [1] E. Balduf, “Improved Algorithms for Automatic Retinal Mapping and Real-Time Location Determination for Retinal Laser Surgery Systems,” Masters Thesis, Rensselaer Polytechnic Institute, July, 1998.
- [2] D.E. Becker, “Algorithms for Automatic Retinal Mapping and Real-Time Location Determination for an Improved Retinal Laser Surgery System,” Ph.D. Thesis, Rensselaer Polytechnic Institute, August 1995.
- [3] D. Becker, A. Can, J.N. Turner, H.L. Tanenbaum, B. Roysam, “Image Processing Algorithms for Retinal Montage Synthesis, Mapping, and Real-Time Location Determination,” *IEEE Transactions on Biomedical Engineering*, vol. 45, pp. 105–118, January 1998.
- [4] N.M. Bressler, S.B. Bressler, E.S. Gragoudas, “Clinical Characteristics of Choroidal Neovascular Membranes,” *Archives of Ophthalmology*, vol. 105, pp. 209–213, 1987.
- [5] A. Can, H. Shen, J.N. Turner, H.L. Tanenbaum, B. Roysam, “Rapid Automated Tracing and Feature Extraction from Retinal Fundus Images Using Direct Exploratory Algorithms,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 3, no. 2, June 1999.
- [6] A. Can, C.V. Stewart, B. Roysam, H.L. Tanenbaum, “A Feature-Based Algorithm for Joint, Linear Estimation of High-Order Image-to-Mosaic Transformations: Mosaicing the Curved Human Retina,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 412–419, March 2002.
- [7] A. Can, C.V. Stewart, B. Roysam, H.L. Tanenbaum, “A Feature-Based, Robust, Hierarchical Algorithm for Registering Pairs of Images of the Curved Human Retina,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 347–364, March 2002.
- [8] S. Fine, “Observations Following Laser Treatment for Choroidal Neovascularization,” *Archives of Ophthalmology*, vol. 106, pp. 1524–1525, 1988.
- [9] J.M. Krauss, C.A. Puliafito, “Lasers in Ophthalmology,” *Lasers Surg. Med.*, vol. 17, pp. 102–159, 1995.

- [10] G. Lin, C.V. Stewart, B. Roysam, K. Fritzsche, G. Yang, H.L. Tanenbaum, "Predictive Scheduling Algorithms for Real-Time Feature Extraction and Spatial Referencing: Application to Retinal Image Sequences," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 1, pp. 115–125, January 2004.
- [11] Love, Robert. *Linux Kernel Development, Second Edition*, Novell Press, Indianapolis, Indiana, 2005.
- [12] P.N. Monahan, K.A. Gitter, J.D. Eichler, G. Cohen, "Evaluation of Persistence of Subretinal Neovascular Membranes Using Digitized Angiographic Analysis," *Retina — J. Retinal Vitreous Diseases*, vol. 13, no. 3, pp. 196–201, 1993.
- [13] P.N. Monahan, K.A. Gitter, J.D. Eichler, G. Cohen, K. Schomaker, "Use of Digitized Fluorescein Angiogram System to Evaluate Laser for Subretinal Neovascularization: Technique," *Retina — J. Retinal Vitreous Diseases*, vol. 13, no. 3, pp. 187–195, 1993.
- [14] R. Murphy, "Age-Related Macular Degeneration," *Ophthalmology*, vol. 9, pp. 696–971, 1986.
- [15] R.A. O'Neil, "Parallel Spatial Referencing of Retinal Fundus Images: Application for Real-Time Laser Retinal Surgery," Masters Thesis, Rensselaer Polytechnic Institute, April 2003.
- [16] H. Shen, "Indexing Based Frame-Rate Spatial Referencing Algorithms: Application to Laser Retinal Surgery," Ph.D. Thesis, Rensselaer Polytechnic Institute, November 2000.
- [17] H. Shen, B. Roysam, C.V. Stewart, J.N. Turner, H.L. Tanenbaum, "Optimal Scheduling of Tracing Computations for Real-Time Vascular Landmark Extraction from Retinal Fundus Images," *IEEE Transactions on Information Technology in Biomedicine*, vol. 5, no. 1, pp. 77–91, March 2001.
- [18] H. Shen, C.V. Stewart, B. Roysam, G. Lin, H.L. Tanenbaum, "Frame-Rate Spatial Referencing Based on Invariant Indexing and Alignment with Application to On-Line Retinal Image Registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 379–384, March 2003.
- [19] W.R. Stevens, *Advanced Programming in the Unix Environment*, Addison-Wesley, Boston, Massachusetts, 1993.
- [20] C.V. Stewart, C.-L. Tsai, B. Roysam, "The Dual-Bootstrap Iterative Closest Point Algorithm With Application to Retinal Image Registration," *IEEE Transactions on Medical Imaging*, vol. 22, no. 11, pp. 1379–1394, November 2003.

- [21] J.A. Tyrrell, J.M. LaPre, C.D. Carothers, B. Roysam, C.V. Stewart, "Efficient Migration of Complex Off-Line Computer Vision Software to Real-Time System Implementation on Generic Computer Hardware," *IEEE Transactions on Information Technology in Biomedicine*, vol. 8, no. 2, pp. 142–153, June 2004.
- [22] *Personal correspondence with Alex Tyrrell.*

## Appendix A

### Software Modifications

#### A.1 Low Level High Priority Algorithm

Anything between `<JUSTIN>` and `</JUSTIN>` is code added by the author. In `rtl_online_vessel_tracer.txx`, the function `initialize_box` was modified:

```
for( int i=0;i<box_level_;++i )
    box_heap_[i].Heapify( grid_box_sptr_[i], (int)pow(4,i)*grid_box_sptr_[0].size()

//<JUSTIN>
for(int q = 0; q < THREADS - 1; q++) {
    if(fork() != 0) ++thread_number;
    else break;
}

switch(thread_number) {
case 0:
    switch_boxes(1, 2, 3);
    break;

case 1:
    switch_boxes(0, 2, 3);
    break;

case 2:
    switch_boxes(0, 1, 3);
    break;
```

```

    case 3:
        switch_boxes(0, 1, 2);
    }
//</JUSTIN>

// subdivide the boxes
if( box_level_>1 ) add_finetest_box();

```

and the function switch\_boxes was added:

```

//<JUSTIN>
template <class pix_type>
void
rtl_online_vessel_tracer< pix_type > ::
switch_boxes(int box1, int box2, int box3)
{
    box_heap_[0][box1]->has_traced = true;
    box_heap_[0][box2]->has_traced = true;
    box_heap_[0][box3]->has_traced = true;
    box_heap_[0].Heapify(box_heap_[0].heap_array, box_heap_[0].size());
}
//</JUSTIN>

```

## A.2 Quadrants

Again, in rtl\_online\_vessel\_tracer.txx, the function initialize\_box was modified:

```

// calculate initial quality of the boxes
for( int i=0;i<(int)grid_box_sptr_[0].size();++i )
    quality_of_box( grid_box_sptr_[0][i] );

```

```
//<JUSTIN>

for(int q = 0; q < THREADS - 1; q++) {
    if(fork() != 0) ++thread_number;
    else break;
}

switch(thread_number) {

case 0:
    switch_boxes(0, 1, 4, 5);
    break;

case 1:
    switch_boxes(2, 3, 6, 7);
    break;

case 2:
    switch_boxes(8, 9, 12, 13);
    break;

case 3:
    switch_boxes(10, 11, 14, 15);
}

//</JUSTIN>

// build heap at different levels
for( int i=0;i<box_level_;++i )
    box_heap_[i].Heapify( grid_box_sptr_[i], (int)pow(4,i)*grid_box_sptr_[0].size()
```

and the function switch\_boxes was added:

```
//<JUSTIN>
template <class pix_type>
void
rtl_online_vessel_tracer< pix_type > ::
switch_boxes(int i1, int i2, int i3, int i4)
{
    grid_box_sptr_[0][i1]->quality += 100.0;
    grid_box_sptr_[0][i1]->weight += 100;

    grid_box_sptr_[0][i2]->quality += 100.0;
    grid_box_sptr_[0][i2]->weight += 100;

    grid_box_sptr_[0][i3]->quality += 100.0;
    grid_box_sptr_[0][i3]->weight += 100;

    grid_box_sptr_[0][i4]->quality += 100.0;
    grid_box_sptr_[0][i4]->weight += 100;
}
//</JUSTIN>
```

## Appendix B

### How to Build and Run the Software

Excellent documentation for building the software already exists at the following URL:

<http://www.cs.rpi.edu/research/groups/vision/doc/>

Our group uses a tool known as CMake which is a more advanced analog to the familiar Unix tool “make.” Configuring and building the software is simple — just follow these steps:

1. Install CMake.
2. Get the VXL and RPI source and organize them accordingly.
3. Create a directory for the binaries.
4. In the binary directory, run “ccmake <source directory>.” “ccmake” is the GUI front-end to “cmake.”
5. Change the line “CMAKE\_BACKWARDS\_COMPATIBILITY” from 2.0 to 1.2.
6. Type “c” once and wait. When it stops, type “c” again and wait. Once that finishes, type “g” to generate the Unix makefiles.
7. Once “ccmake” exits, change directories to “retl/rtx” from the current binary directory.
8. Type “make online\_indexing\_with\_leave\_out” to create the binary.

Shell scripts often come in handy for running the binary “online\_indexing\_with\_leave\_out” on a large number of images.