6-1-2011

CUDA accelerated cone-beam reconstruction

Albert Sze

CUDA Accelerated Cone-Beam Reconstruction

by

Albert W. Sze

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of Master of Science
in Computer Engineering

Supervised by

Professor and Department Head Dr. Andreas Savakis
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
June 2011

Approved by:

Dr. Andreas Savakis, Professor and Department Head
Thesis Advisor, Department of Computer Engineering

Dr. Jay S. Schildkraut,
Committee Member, Carestream Health, Inc.

Dr. Muhammad E. Shaaban, Professor
Committee Member, Department of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

CUDA Accelerated Cone-Beam Reconstruction

I, Albert W. Sze, hereby grant permission to Wallace Memorial Library to reproduce my thesis in whole or part.

Albert W. Sze

Date

DEDICATION

I dedicate this thesis to my parents, for their unending support.

ACKNOWLEDGMENTS

I would like to thank my committee members Dr. Andreas Savakis for introducing me to this project and advising me, my supervisor Dr. Jay Schildkraut for his direct cooperation and support, and Dr. Muhammad Shaaban for being fundamental to my education.

I would also like to thank Rick Tolleson for being the best lab manager, ever.

Lastly, this project would have not been possible without the support of RIT, Carestream Health, and my family.

ABSTRACT

CUDA Accelerated Cone-Beam Reconstruction

Albert W. Sze

Supervising Professor: Dr. Andreas Savakis

Cone-Beam Computed Tomography (CBCT) is an imaging method that reconstructs a 3D representation of the object from its 2D X-ray images. It is an important diagnostic tool in the medical field, especially dentistry. However, most 3D reconstruction algorithms are computationally intensive and time consuming; this limitation constrains the use of CBCT.

In recent years, high-end graphics cards, such as the ones powered by NVIDIA graphics processing units (GPUs), are able to perform general purpose computation. Due to the highly parallel nature of the 3D reconstruction algorithms, it is possible to implement these algorithms on the GPU to reduce the processing time to the level that is practical.

Two of the most popular 3D Cone-Beam reconstruction algorithms are the Feldkamp-Davis-Kress algorithm (FDK) and the Algebraic Reconstruction Technique (ART). FDK is fast to construct 3D images, but the quality of its images is lower than the quality of ART

images. However, ART requires significantly more computation. Material ART is a recently developed algorithm that uses beam-hardening correction to eliminate artifacts.

In this thesis, these three algorithms were implemented on the NVIDIA's CUDA platform. These CUDA based algorithms were tested on three different graphics cards, using phantom and real data. The test results show significant speedup when compared to the CPU software implementation. The speedup is sufficient to allow a moderate cost personal computer with NVIDIA graphics card to process CBCT images in real-time.

CONTENTS

DEDICATION iii

ACKNOWLEDGMENTS iv

ABSTRACT v

Chapter 1: Introduction 1

Chapter 2: Cone-Beam Reconstruction Algorithms 4

2.1 The Projection Model 4

2.2 FDK Algorithm 5

2.3 ART Algorithm 11

2.4 Material ART Algorithm 16

Chapter 3: NVIDIA CUDA 18

3.1 Brief History of GPUs 18

3.2 Main G80 “Tesla” Architectural Features 20

3.3 NVIDIA CUDA 23

3.4 Fermi Architecture 24

3.5 Unique CUDA Features 27

Chapter 4: Previous Work 29

4.1 Previous Reconstruction Algorithms 29

4.2 Previous CUDA Accelerations 29

Chapter 5: CUDA Implementations 31

5.1 Overview 31

5.2 Important Points about CUDA 31

5.3 CUDA Accelerated FDK 32

5.4 CUDA Accelerated ART 35

5.5 CUDA Accelerated Material ART 37

5.6 Development 40

5.6.1 ART vs. Material ART 41

Chapter 6: Test Methodology 44

6.1 CUDA Graphics Cards 45

6.2 Data Sets 46

6.3 Expected Results 49

Chapter 7: Results and Analysis 50

7.1 Image Quality 50

7.1.1 GPU vs. CPU 51

7.1.2 ART vs. FDK 52

7.1.3 Comparison of Material ART Versions 56

7.1.4 Confirmation of Material ART 56

7.1.5 Truncation Artifacts 59

7.2 Runtime Results 61

7.2.1 Overview 62

7.2.2 Runtime Confirmations 63

7.2.3 Varying Problem Size 64

7.2.4 Overall Performance Gains 65

7.3 Unexpected Results 66

7.3.1 Comparing Data Sets 66

7.3.2 Observation 68

7.3.3 ART and Material ART 70

7.3.4 GPU Time Breakdown 72

Chapter 8: Conclusions and Future Directions 77

8.1 Conclusions 77

8.2 Future Directions 77

BIBLIOGRAPHY 80

APPENDIX A Reconstructed Images from the Runs of Reconstruction Algorithms on CPU & GTX560 Graphics Card 82

APPENDIX B Reconstructed Images from the 20 Iteration Runs of Reconstruction Algorithms on GTX560 Graphics Card..... 90

APPENDIX C Runtime Results from the 20 Iteration Runs of Reconstruction Algorithms on GTX560 Graphics Card 109

LIST OF TABLES

Note: Tables in the Appendices are not listed here.

Table 1 – GT 330M Specifications 40

Table 2 – Summary of Test Plan 44

Table 3 – Benchmark Workstation Specifications 45

Table 4 – Testing CUDA Graphics Cards Specifications 45

Table 5 – Data Sets Description 46

Table 6 – Image Comparison for Appendix A 51

Table 7 – Images Comparison for Appendix B 52

Table 8 – Runtimes for Reconstruction Algorithms ran on CPU & Graphics Cards 61

Table 9 – Runtime Scaling for P1, P2, and P3 64

Table 10 – Runtimes for Reconstruction Algorithms ran on CPU & Graphics Cards 65

Table 11 – GPU Time Breakdown 72

Table 12 – Percentage Breakdown of GPU Time 73

LIST OF FIGURES

Note: Figures in the Appendices are not listed here.

Figure 1 – KODAK 9500 Cone-Beam 3D System [8] 2

Figure 2 – Model of How the X-ray Projection Data is Collected [9] 4

Figure 3 – Three Dimensional Coordinate System [1] 5

Figure 4 – Parallel Beam Projections [5] 6

Figure 5 – Fan Beam Projections [5] 7

Figure 6 – The Fourier Slice Theorem [5] 8

Figure 7 – CPU Implementation of the FDK Algorithm 10

Figure 8 – Object and Projection Representation for ART[5] 12

Figure 9 – The Kaczmarz Method, an Iterative Method to Solve a System of Equations [5] 13

Figure 10 – CPU Implementation of ART Algorithm 15

Figure 11 – CPU Implementation of Material ART Algorithm 17

Figure 12 – GeForce 6800 Architecture [10] 19

Figure 13 – GeForce 8800 Architecture [11] 20

Figure 14 – The Two Streaming Multiprocessors (SM) in a Texture/Processor Cluster (TPC). 21

Figure 15 – CUDA Hierarchy of Threads, Blocks, and Grids [12] 23

Figure 16 – The Fermi Architecture [12] 24

Figure 17 – Streaming Multiprocessor for the Fermi Architecture [12] 25

Figure 18 – Comparison between CPU and GPU in FLOPs [13] 26

Figure 19 – Comparison between CPU and GPU in Memory Bandwidth [13] 26

Figure 20 – CUDA Implementation of FDK Algorithm 33

Figure 21 – CUDA Implementation of ART Algorithm 35

Figure 22 – CUDA Implementation of Material ART V1..... 37

Figure 23 – CUDA Implementation of Material ART V2..... 38

Figure 24 – The Shepp-Logan Phantom 47

Figure 25 – Carestream Health CS 9300 System..... 48

Figure 26 – B1 (R1, FDK, Slice 322) [left]; B5 (R1, ART Iteration 20, Slice 322) [right] 53

Figure 27 – B16 (R1, FDK, Slice 364) [left]; B20 (R1, ART Iteration 20, Slice 364) [right] . 54

Figure 28 – Elimination of Vertical Streaking..... 55

Figure 29 – B25 (R1, MART1 Iteration 20, Slice 364) [left]; B30 (R1, MART2 Iteration 20, Slice 364) [right] 56

Figure 30 – Comparison between FDK and ART 57

Figure 31 – Comparison between FDK and Material ART V1 58

Figure 32 – Comparison between FDK and Material ART V2 58

Figure 33 – Comparison between Material ART V1 and V2 59

Figure 34 – Truncation Artifact in B31 (R2, FDK, Slice 135) [left]; B35 (R2, ART Iteration 20, Slice 135) [right] 59

Figure 35 – Truncation Scenario [22]..... 61

Figure 36 – Reconstruction Algorithms on CPU..... 68

Figure 37 – Reconstruction Algorithms on GTX295..... 68

Figure 38 – Reconstruction Algorithms on GTX560..... 69

Figure 39 – ART Algorithm Comparison..... 69

Figure 40 – Material ART V2 Algorithm Comparison..... 70

Figure 41 – GPU Time Percentages for ART on Data Sets P1 and P2 75

LIST OF EQUATIONS

Equation 1 – Fourier Slice Theorem [5] 7

Equation 2 – Filtered Backprojection [5] 8

Equation 3 – The System of Equations for ART[5] 12

Equation 4 – Formula to Find the Next Iteration [5] 13

Chapter 1: Introduction

Computed Tomography (CT) and Cone-Beam Computed Tomography (CBCT) are two of many medical imaging technologies that allow doctors and dentists to visualize what is inside of the human body in three dimensions. Both technologies allow medical professionals to see all parts of the human body without injuring it.

The main components of Computed Tomography or Cone-Beam Computed Tomography are a rotating X-ray machine and a very powerful digital computer. The X-ray machine takes thousands of two-dimensional "pictures" of the object or the body; digital computer puts these images together into a highly detailed three-dimensional image of what is inside the object. However, 3-D image reconstruction is computationally intensive. There are two challenges to computer scientists and engineers: to reconstruct the 3-D image in real-time and to complete it on commodity hardware.

The CBCT scanner is smaller, faster, and safer than the CT scanner. Due to the cone shaped X-ray beam, the size of the scanner, amount of X-rays, and scanning time are reduced greatly. These can be very beneficial to patients; therefore CBCT scanners are more popular than the CT scanners.

Figure 1 – KODAK 9500 Cone-Beam 3D System [8]

Scientists and engineers have been developing many algorithms to reconstruct 3-D images from 2-D CBCT images. Two of the most popular CBCT algorithms are the Feldkamp-Davis-Kress algorithm (FDK) and the Algebraic Reconstruction Technique (ART). FDK is the most basic and common reconstruction algorithm. It uses the method of filtered-backprojection to reconstruct the original object. FDK is faster than ART but not as accurate as ART. ART is an iterative method that builds upon the FDK. It uses projection in conjunction with backprojection to correct for reconstruction artifacts.

Recently developed, Material ART is an improvement over ART, as Material ART will correct for beam-hardening. Beam-hardening is an artifact that occurs when the X-ray source is polychromatic. It produces reconstruction artifacts that make the center of the reconstruction volume denser than it should be. This is because the lower energy levels get absorbed faster, thus distorting the energy distribution.

Modern NVIDIA GPUs are scalable processor arrays that excel at computing parallel tasks. Compute United Device Architecture (CUDA) technology allows general purpose programs to run on NVIDIA GPUs. Since the reconstruction algorithms are easy to "parallelize," image reconstruction of CBCT could be implemented on the CUDA platform.

The objective of this thesis is to implement three reconstruction algorithms, FDK, ART, and Material ART, for Cone-Beam Computed Tomography on NVIDIA GPUs, using CUDA. Three different NVIDIA GPUs are compared: the high-end GeForce GTX 560 with 2 GB graphics memory, the mid-range GeForce GTX 295 with 1 GB graphics memory, and the low-end GeForce G210 with 512 MB graphics memory. These GPUs should provide a good estimation of the cost-to-performance ratio of the CUDA based reconstruction algorithms.

In the remainder of the thesis, Chapter 2 provides background information on the CBCT and on the three reconstruction algorithms considered. Chapter 3 describes the hardware and software architectures of the NVIDIA's CUDA enabled GPUs. Chapter 4 discusses various implementations of the reconstruction algorithms found in published scientific papers. Chapter 5 details the implementation of these three algorithms on the GPU. The test methodology is discussed in Chapter 6, and Chapter 7 reports and analyzes results. Finally, Chapter 8 discusses final conclusions and future work.

Chapter 2: Cone-Beam Reconstruction Algorithms

2.1 The Projection Model

The model assumed for the collected projection data is that an X-ray source and detector plane are rotated about an object. The object could be a patient's head, as shown in Figure 2.

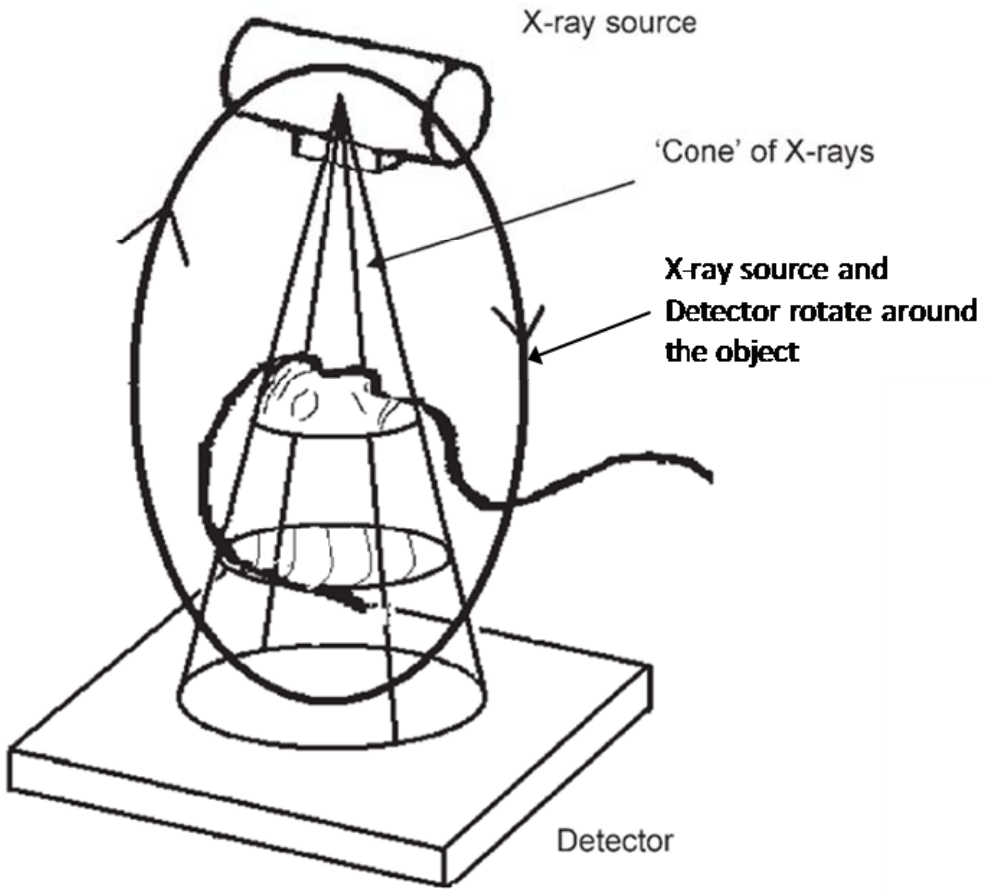


Figure 2 – Model of How the X-ray Projection Data is Collected [9]

Based on this, a coordinate system can be created based on the source, object, the axis of rotation, and the location of the detection plane.

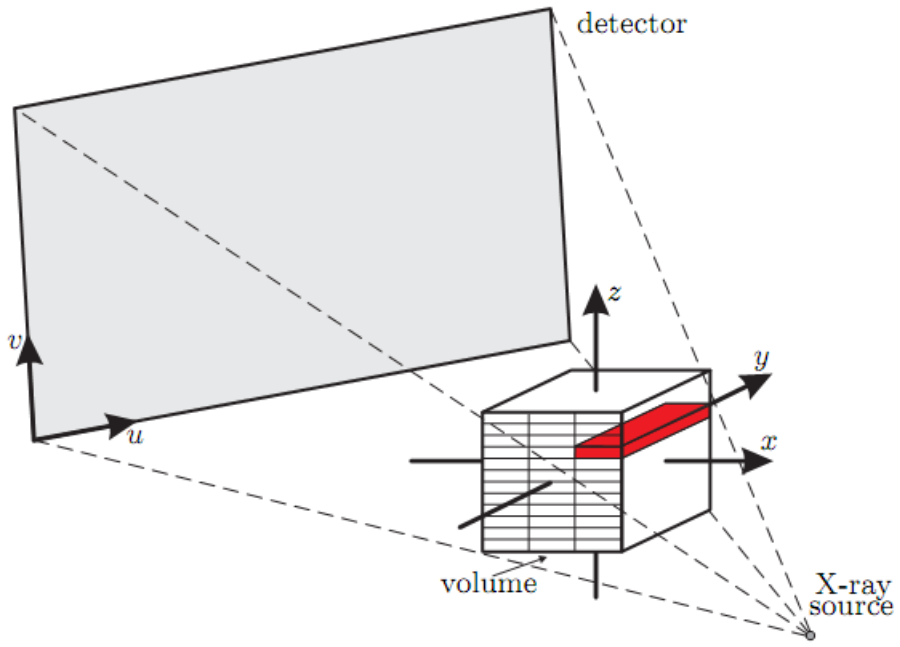


Figure 3 – Three Dimensional Coordinate System [1]

Figure 3 demonstrates this coordinate system. The reconstruction volume can have an arbitrary coordinate system, but each projection will have its own detector coordinate system (illustrated by the *u* and *v* axes) and current source location. The source location will be in volume coordinates. In the implementations, locations are often translated between these coordinate systems.

2.2 FDK Algorithm

The FDK algorithm, named after L. A. Feldkamp, L. C. Davis, and J. W. Kress, is one of the most popular algorithms used for CBCT [4]. It is based on filtered backprojection.

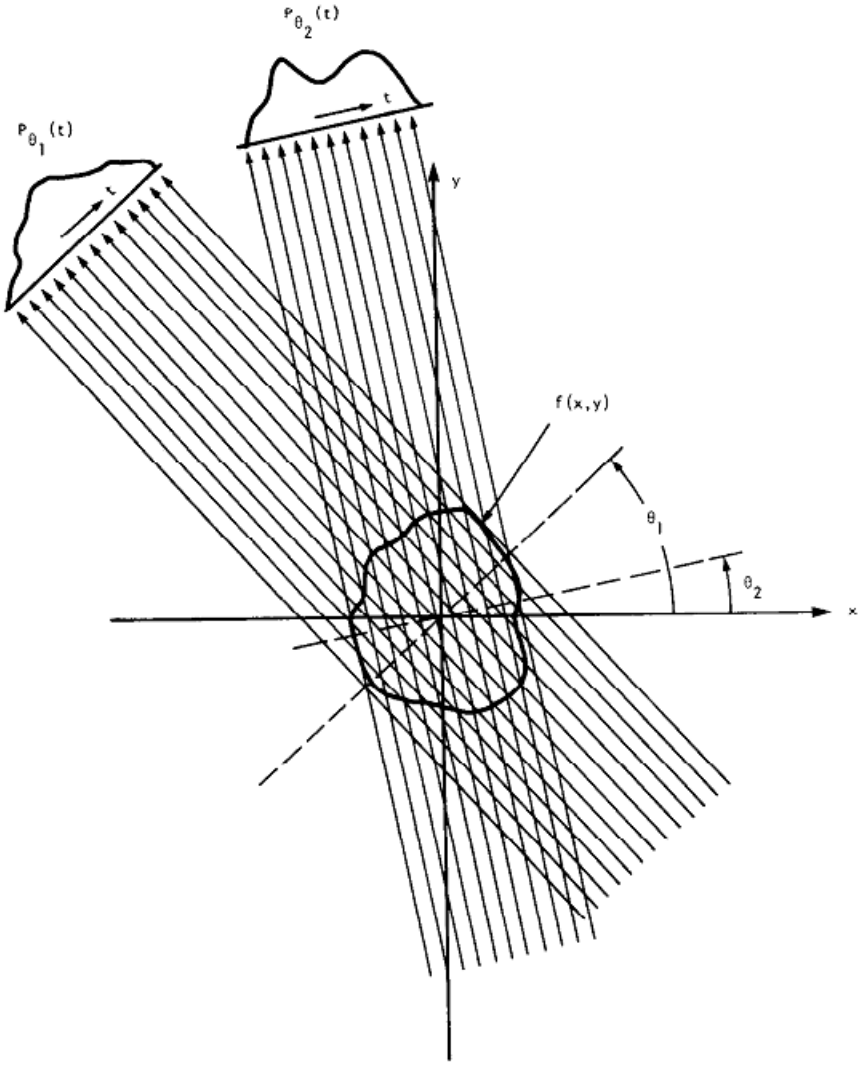


Figure 4 – Parallel Beam Projections [5]

A projection can be modeled as the line integrals of an object. The diagram at the top shows the projections formed by parallel lines through an object.

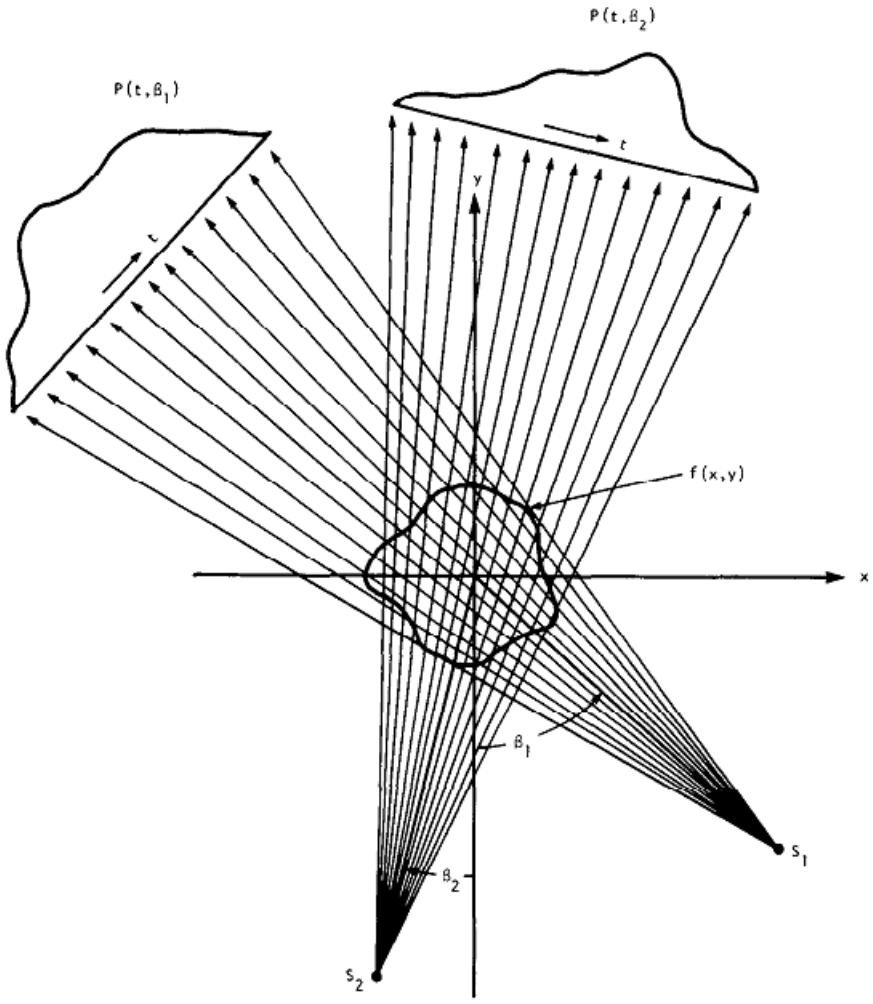


Figure 5 – Fan Beam Projections [5]

A more accurate model for the projections of X-rays from a single source is if the lines form a fan rather than being parallel with respect to each other.

The foundation of tomographic imaging reconstruction is based on the Fourier Slice Theorem. The Fourier Slice Theorem is:

$$F(u,v) = \int_{-\infty}^{\infty} P_\theta(t) e^{-j2\pi wt} dt$$

Equation 1 – Fourier Slice Theorem [5]

Where $F(u, v)$ is the 2-dimensional Fourier transform of the object and $P_\theta(t)$ is the projection of the object at angle θ. What this means is the Fourier Transform of a projection is equal to the slice of the object in the frequency domain.

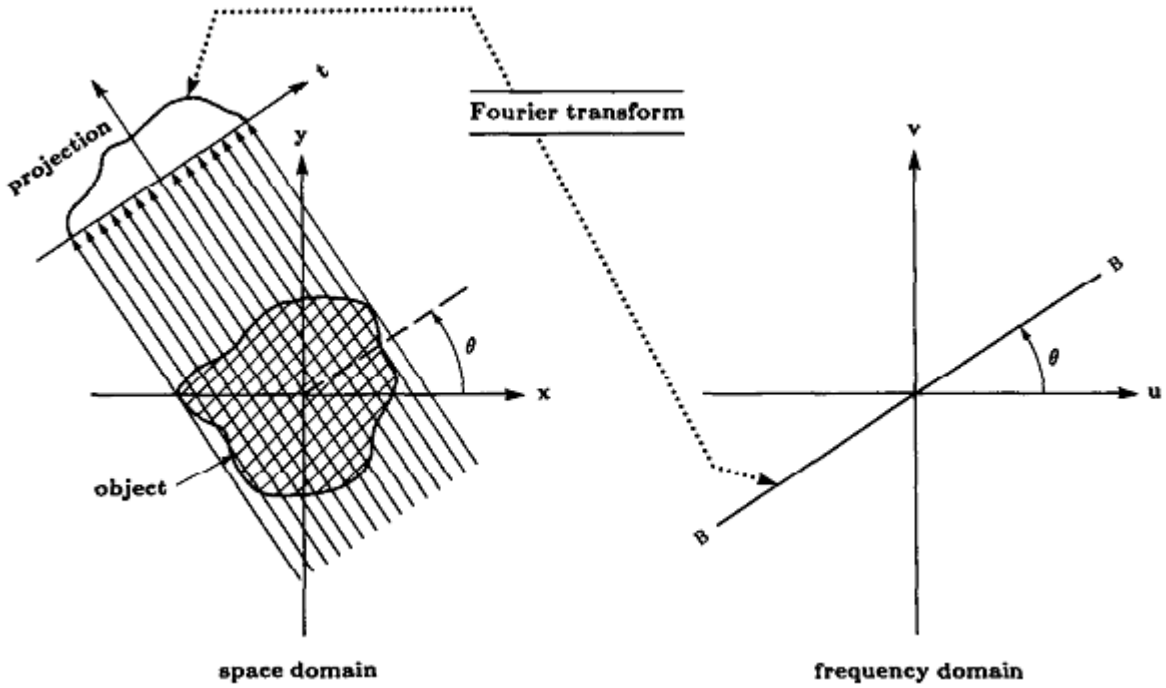


Figure 6 – The Fourier Slice Theorem [5]

Thus, for the case of parallel line projections, the original object can be found through the polar integral of the weighed (or filtered) inverse Fourier transform of the projection slices, or:

$$f(x,y) = \int_0^\pi \left[\int_{-\infty}^{\infty} S_\theta(w)|w| e^{j2\pi t} dw \right] d\theta$$

Equation 2 – Filtered Backprojection [5]

Where $S_\theta(w)$ is the Fourier transform at angle θ.

The case of fan-beam projections can be derived by reorganizing the projection lines into the parallel-line equivalent coordinate system. This introduces another weight into the formula. The algorithm is completed once it is extended to the 3rd dimension.

Figure 7 gives a high level overview of how the FDK reconstruction works. For each projection that gets loaded into the program, the projection goes through a Fourier filter. Then each voxel in the reconstruction object gets modified by backprojection of the projection data.

The FDK reconstruction has very large amounts of parallelism. Each voxel-backprojection operation is independent from each other; therefore, each can run in parallel. The difference between Figure 7 and Figure 20 shows how a CUDA implementation of the algorithm can take advantage of this parallelism.

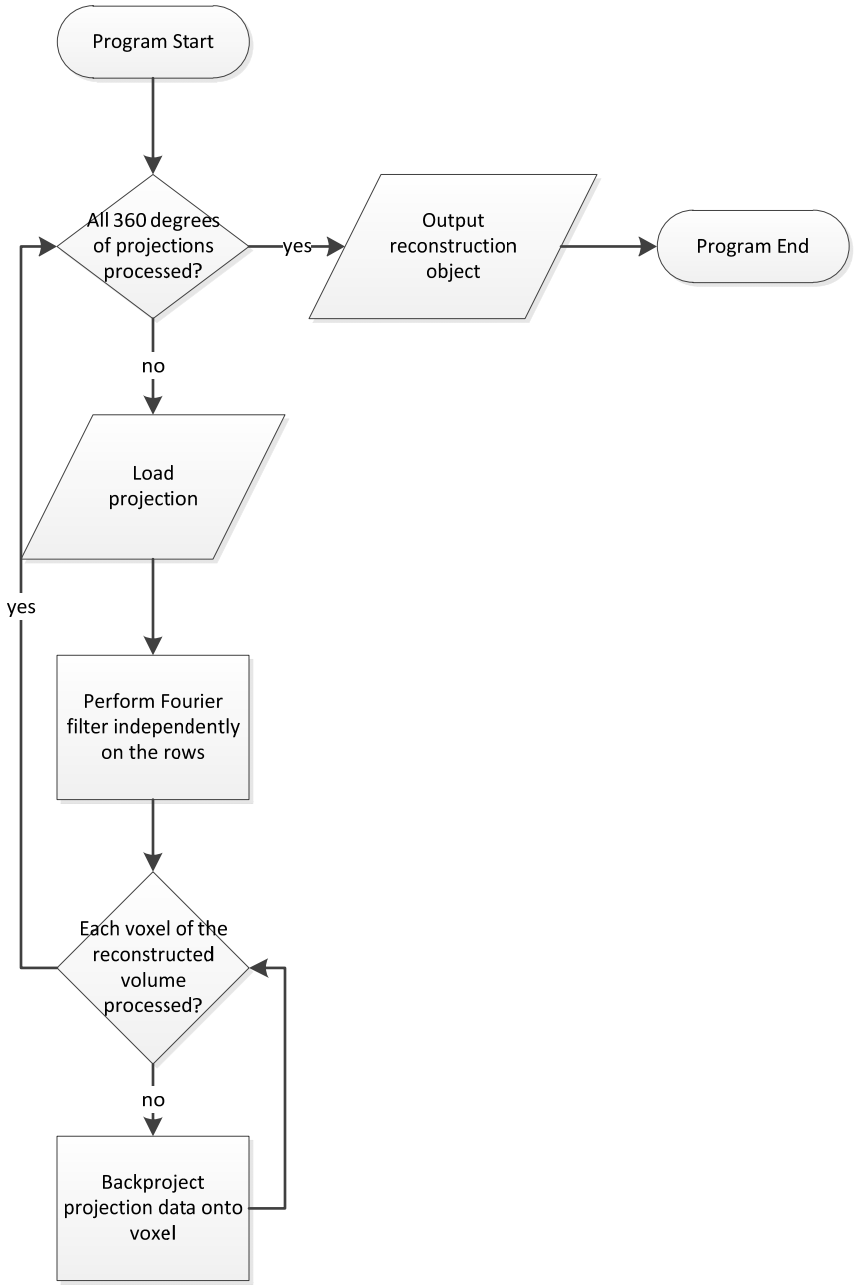


Figure 7 – CPU Implementation of the FDK Algorithm

2.3 ART Algorithm

There are three specific computer implementations of algebraic reconstruction algorithms. They are:

- Algebraic Reconstruction Technique (ART)
- Simultaneous Iterative Reconstructive Technique (SIRT)
- Simultaneous Algebraic Reconstructive Technique (SART)

This thesis implemented SART for CUDA; therefore, we will simply refer to it as ART.

Theoretically, ART approaches the reconstruction in the reverse to FDK. It sets up systems of equations that describe the reconstruction and uses an iterative method to reconstruct the original object. Figure 8 and Equation 3 set up the systems of equations, where N is the number of rays in a projection and M is the number of cells in the object. Figure 9 and Equation 4 demonstrate the iterative calculations. In practice, the ART is used to calculate the differences between the currently reconstructed model and the input projections and uses that to correct any artifacts produced.

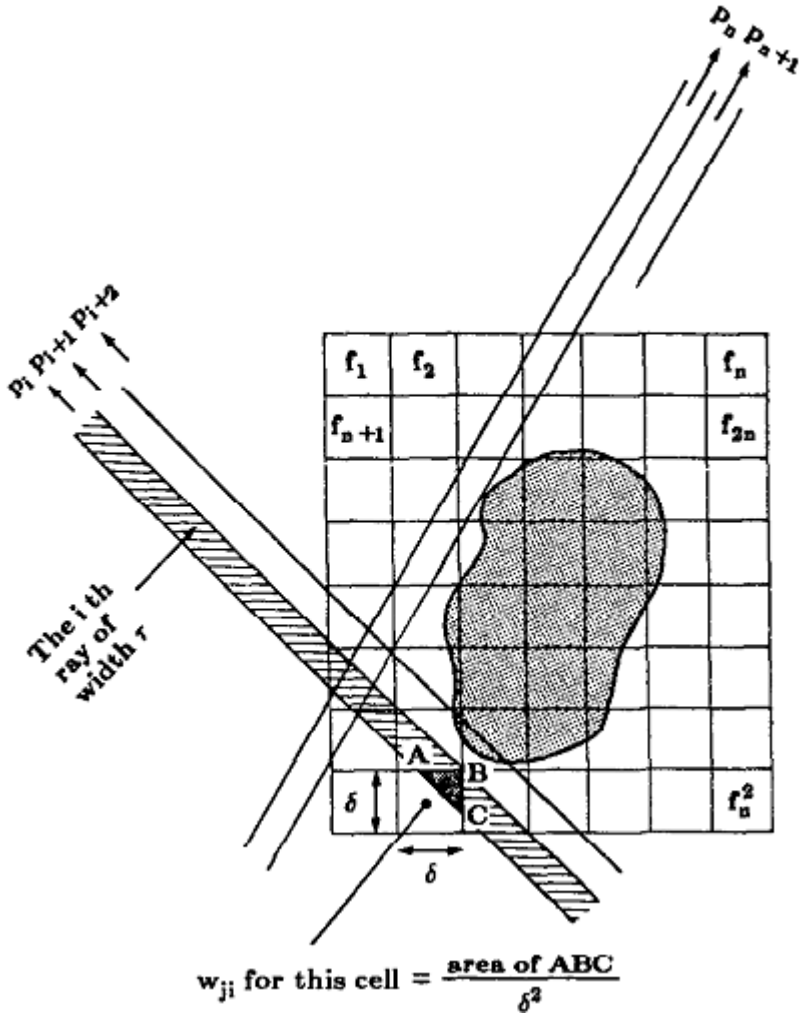


Figure 8 – Object and Projection Representation for ART[5]

$$w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + \cdots + w_{1N}f_N = p_1$$

$$w_{21}f_1 + w_{22}f_2 + w_{23}f_3 + \cdots + w_{2N}f_N = p_2$$

$$\vdots$$

$$w_{M1}f_1 + w_{M2}f_2 + w_{M3}f_3 + \cdots + w_{MN}f_N = p_M$$

Equation 3 – The System of Equations for ART[5]

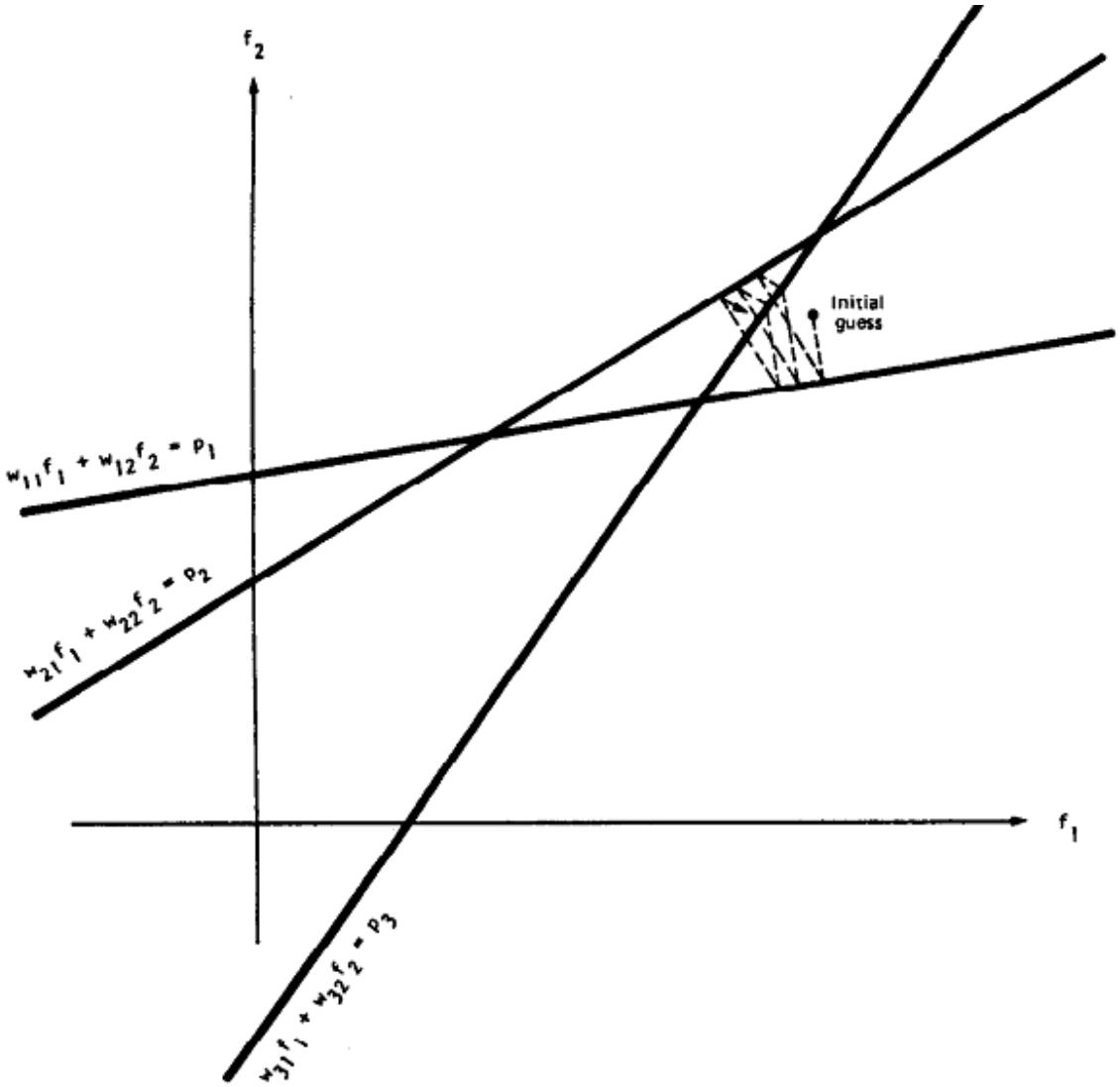


Figure 9 – The Kaczmarz Method, an Iterative Method to Solve a System of Equations [5]

$$\vec{w}_i = (w_{i1}, w_{i2}, \cdots, w_{iN})$$

$$\vec{f}^{(i)} = \vec{f}^{(i-1)} - \frac{(\vec{f}^{(i-1)} \cdot \vec{w}_i - p_i)}{\vec{w}_i \cdot \vec{w}_i} \vec{w}_i$$

Equation 4 – Formula to Find the Next Iteration [5]

The objective is to improve upon FDK. Since ART is an iterative method, it can be initialized to the FDK reconstruction as the initial guess. Additionally, because ART needs to converge onto a solution, FDK provides a good initialization; Figure 9 demonstrates this. Therefore, ART is used to produce correction factors upon FDK reconstruction, to remove the image artifacts and to improve the reconstruction quality.

Figure 10 visualizes ART. The CPU version gives a high level overview of the ART. After the projection is loaded, the reconstruction object gets forward projected to calculate what that projection should look like. Then the difference between the projection data and the forward projection is calculated. This difference is then backprojected onto the reconstruction object to correct for the difference between the projection data and the current reconstruction volume.

Similarly to the FDK, the ART has a large amount of parallelism. The backprojection has the same amount of parallelism compared to the FDK. The forward projection is detector pixel based, as compared to voxel based backprojection, but it still presents an obvious target for parallelization.

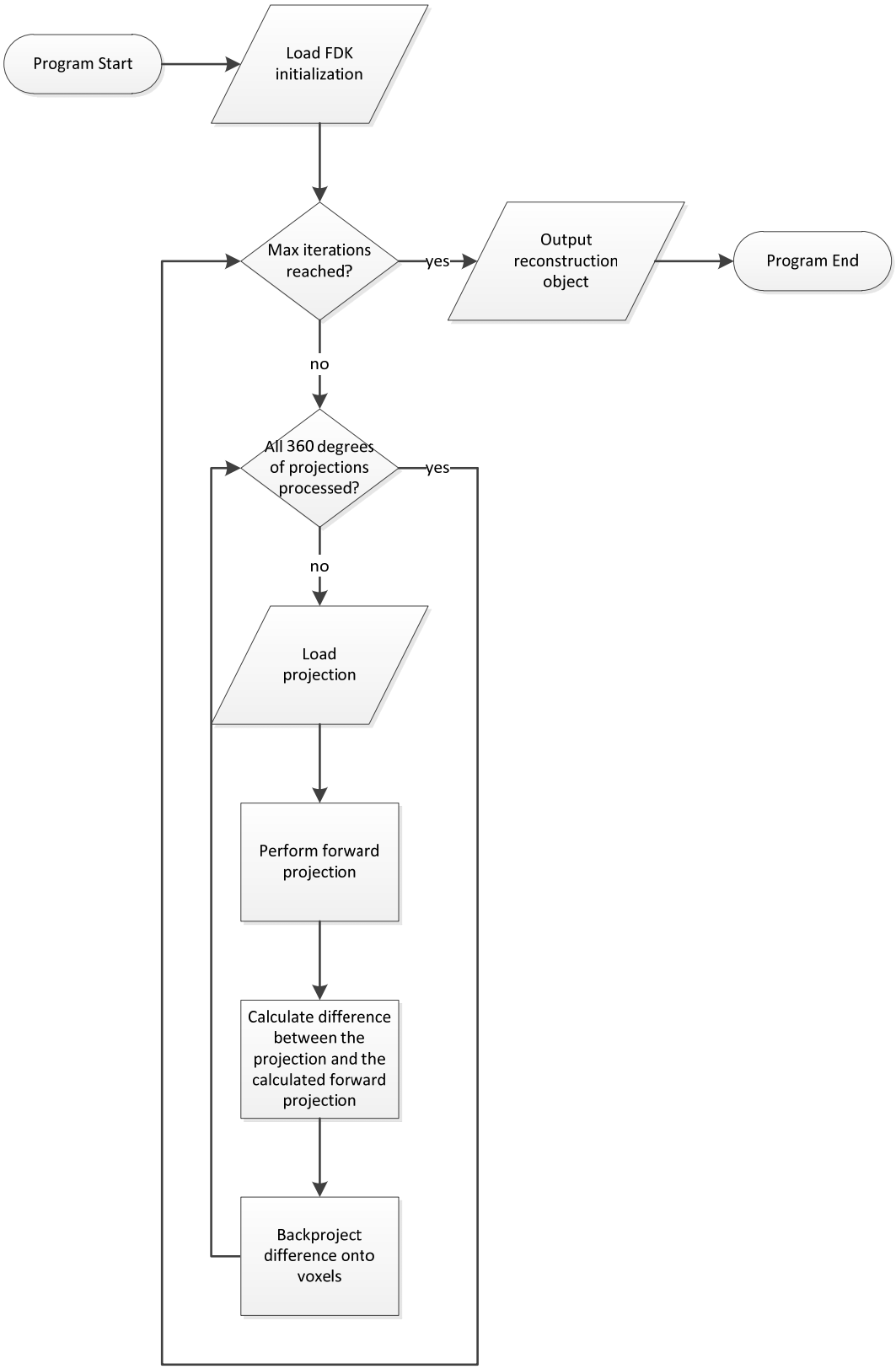


Figure 10 – CPU Implementation of ART Algorithm

2.4 Material ART Algorithm

A variation of the ART that uses “Material Projection” attempts to correct for beam-hardening, a characteristic of X-rays passing through material. In medical imaging, when an X-ray passes through a material, such as flesh or bone, its energy decreases. The FDK and ART assume that the X-ray spectrum is either comprised of a single energy level or comprised of an unchanging distribution of energy levels. However, in real life, neither is true. As an X-ray spectrum passes through material, its lower level energies get absorbed faster than its higher level energies; its energy level distribution changes. This has the effect of making the interior of the scanned object seem denser than it actually is and producing additional reconstruction artifacts. This is called beam-hardening.

Material ART solves this by maintaining an energy distribution count during the forward projection process, and using it to produce a beam-hardening correction factor for the backprojection process. The downside is that it increases the amount of calculations by the number of energy levels maintained for the forward projection process and another volume object has to be maintained for the beam-hardening correction factors. The beam-hardening correction factors pose a problem because of the memory costs of maintaining a large 3D floating-point volume and the limited amount of memory in CUDA cards.

The scope of this thesis is to implement CUDA accelerations for all three algorithms. The next chapter covers the CUDA platform.

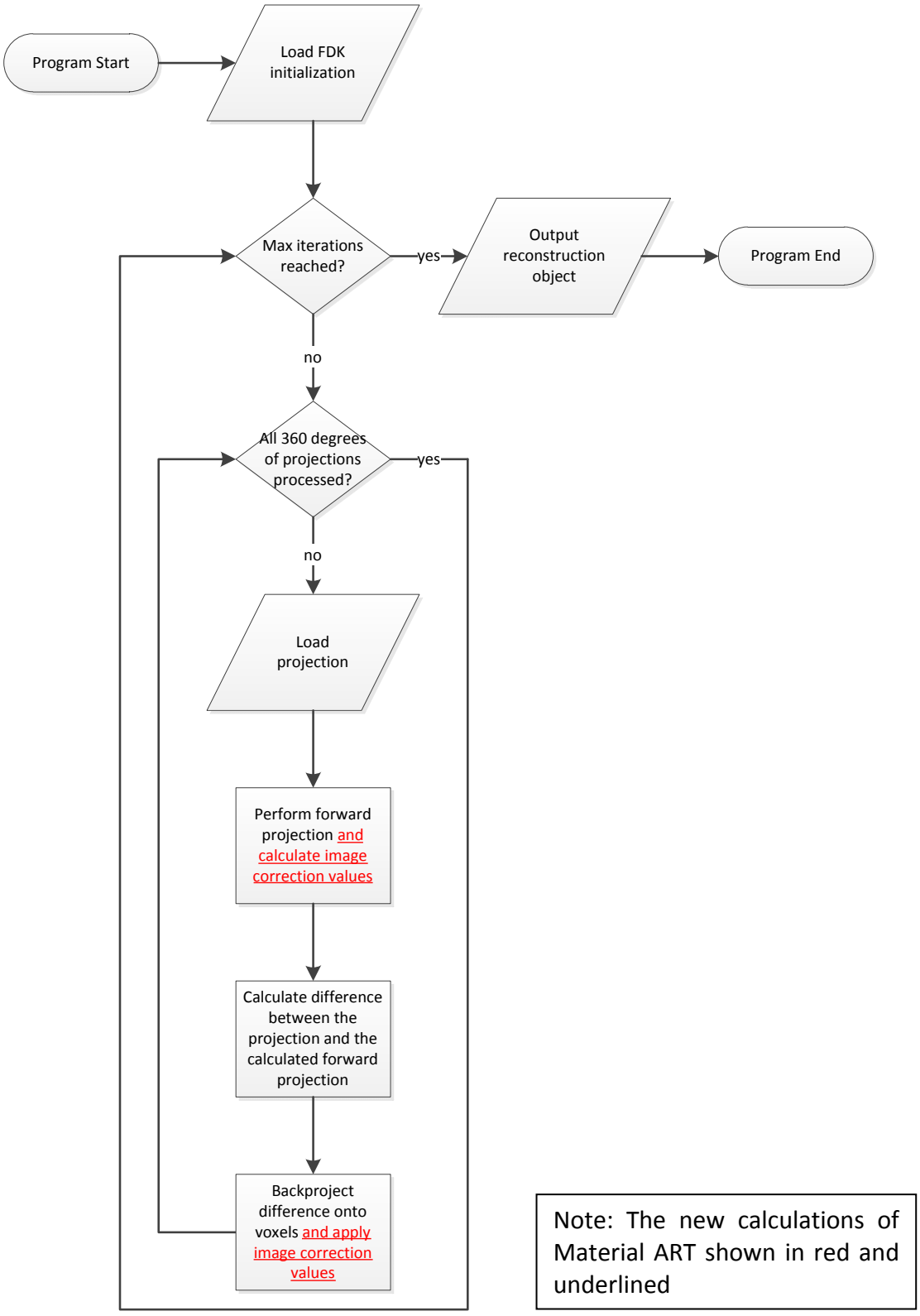


Figure 11 – CPU Implementation of Material ART Algorithm

Chapter 3: NVIDIA CUDA

In 2007, the NVIDIA Corporation introduced the CUDA platform to harness the computational power of its GPUs. Due to their nature of being large arrays of processors and their SIMT nature, CUDA can accelerate parallelized computation.

In this chapter, we will review a brief history of the evolution of the GPUs. Next, we review the G80 "Tesla" architecture and relate it to CUDA. Lastly we discuss the architectural improvements of the Fermi architecture.

3.1 Brief History of GPUs

The primary purpose of GPUs is to accelerate the computation intensive graphics pipeline, typically for graphics rendering and video games. The earliest GPUs had fixed function, but the vertex and fragment stages of the graphics pipeline required the hardware to become programmable. Also, heavy demand from increasingly sophisticated video games placed even greater performance demands from GPUs. Early efforts produced architectures with greater numbers of programmable Vertex and Fragment processors. However, because graphics workloads were typically unbalanced between the Vertex and Fragment processors and the increased cost and difficulty in designing and maintaining two different processors designs, this led to the unified G80 "Tesla" architecture, that was the first CUDA enabled GPU. The subsequent G200 and Fermi architectures added features and improved upon this.

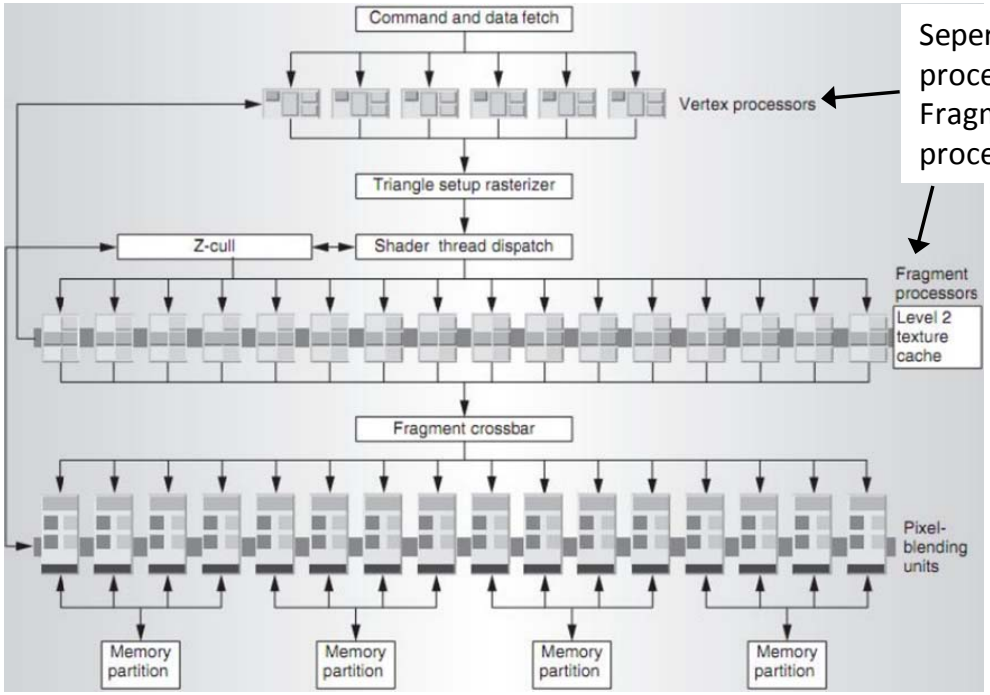


Figure 12 – GeForce 6800 Architecture [10]

Figure 12, Figure 13, Figure 14, Figure 16, and Figure 17 are excerpts from previous papers that illustrate the evolution of the architecture. Figure 12 visualizes the GeForce 6800 architecture, a typical GPU architecture before the unified architecture; note the separate discrete vertex and fragment processors. Compare this to the unified architecture of the GeForce 8800, the first CUDA-enable GPU. Figure 13 illustrates the scalable processors array. Figure 16 illustrates the continued evolution and further improvements made to the architecture; these advancements will be discussed in subsequent sections.

3.2 Main G80 “Tesla” Architectural Features

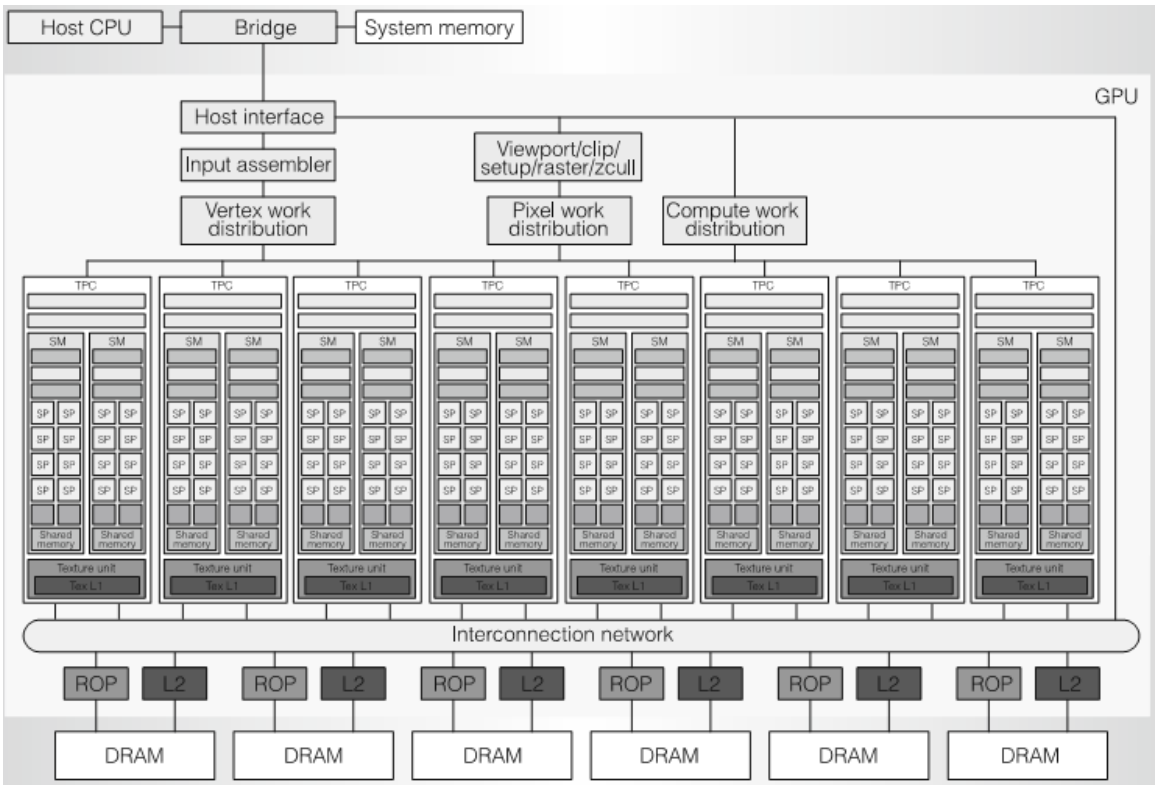
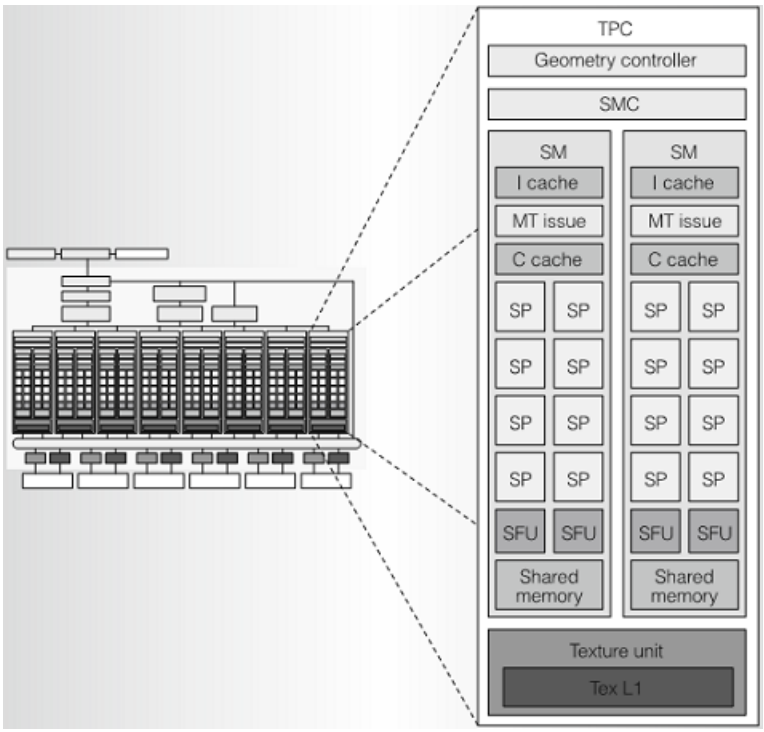


Figure 13 – GeForce 8800 Architecture [11]

The G80 was the first unified architecture GPU. Instead of the separate vertex and fragment processors, it is a “Scalable Processor Array (SPA).” This means that they are comprised of very large arrays of general processing. Figure 13 illustrates the scalable processor array architecture. Instead of running specialized vertex and fragment programs or the graphics pipeline, it can run general programs. This is the basis of CUDA.



Note each SM has 8 Streaming Processors (SP)

Figure 14 – The Two Streaming Multiprocessors (SM) in a Texture/Processor Cluster (TPC).

The main characteristic of modern GPUs is their parallel nature. They are made of multiple streaming processors, with each containing eight processing elements called "CUDA cores." Figure 11 shows a Texture/Processor Cluster, which contains two Streaming Multiprocessors (SM). One CUDA core is considered to be a stream processor, a processor meant to deliver high performance by repeating a set of instructions or kernel over a batch of data. Another example of a stream processor is the Cell Broadband Engine Architecture (CBEA).

The CUDA-enabled GPUs are SIMT in nature, which means Single Instruction, Multiple Thread. This is similar to SIMD, or Single Instruction, Multiple Data. The difference is subtle but important. Instead of just simply executing the same set of instructions over a batch of data, a GPU program will use the parallel architecture to launch a batch of

threads to operate in parallel. Each streaming multiprocessor contains a multithreaded instruction unit that creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps and manages a pool of 24 warps for a total of 768 threads, although for CUDA a block is limited to only 512 threads in Compute Capability 1.x.

Because each streaming multiprocessor contains 8 CUDA cores, it makes each streaming multiprocessor very efficient at processing calculations. Their SIMT nature also makes the multiprocessor very efficient in utilizing the memory bandwidth and hiding the latencies. One weakness of the design is that for maximum efficiency, all threads in a warp must be running the same set of instructions, or divergent threads would occur and take a performance hit. This also means that the GPU is not very effective at handling conditional statements, as it lacks the specialized hardware of traditional processors. Another weakness is that it lacks a unified address space, so it cannot formally use pointers in the C programming language and any function calls must be in-lined. A number of these problems were solved in the Fermi architecture [12].

3.3 NVIDIA CUDA

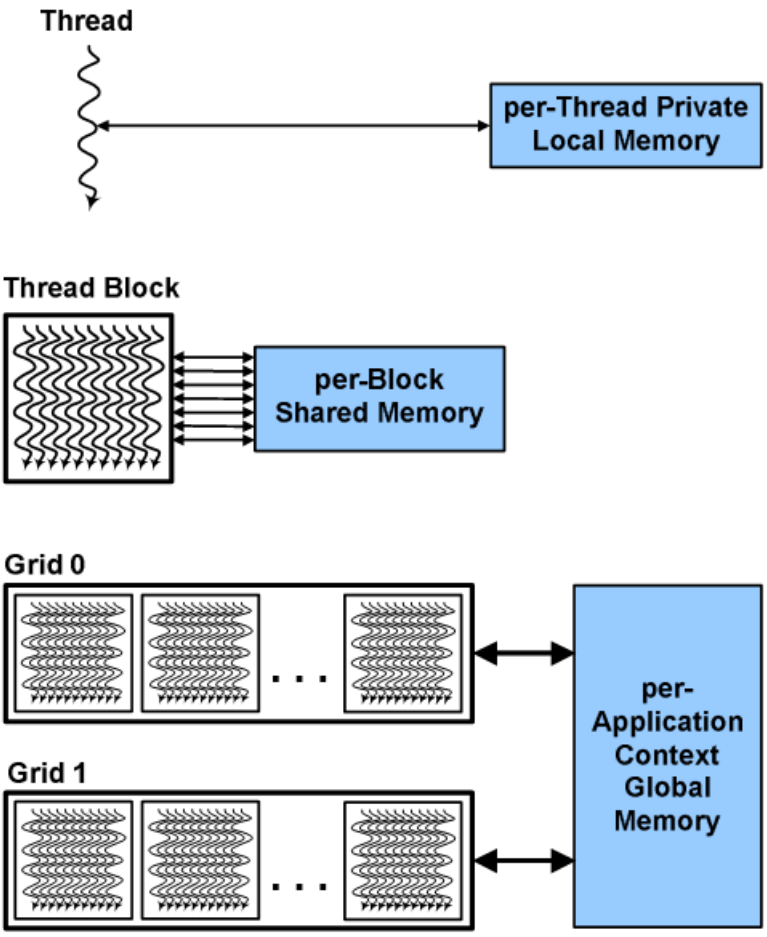


Figure 15 – CUDA Hierarchy of Threads, Blocks, and Grids [12]

Compute Unified Device Architecture (CUDA) is the computing engine in NVIDIA GPUs that allow general programs to run on GPUs. Keeping the G80 Tesla architecture in mind, it is easy to understand how CUDA takes advantage of the hardware to run general programs.

It is the programmer's responsibility to specify explicit parallelism in a program. The programmer does this by programming a C program to run on the GPU, or a "kernel." Then the programmer specifies how many threads to run in that kernel. The threads are

organized by blocks, which are organized into a grid. Figure 15 provides a visual companion for this. Each block of threads corresponds to a SM during execution. The SM automatically manages the pool of threads in hardware.

3.4 Fermi Architecture

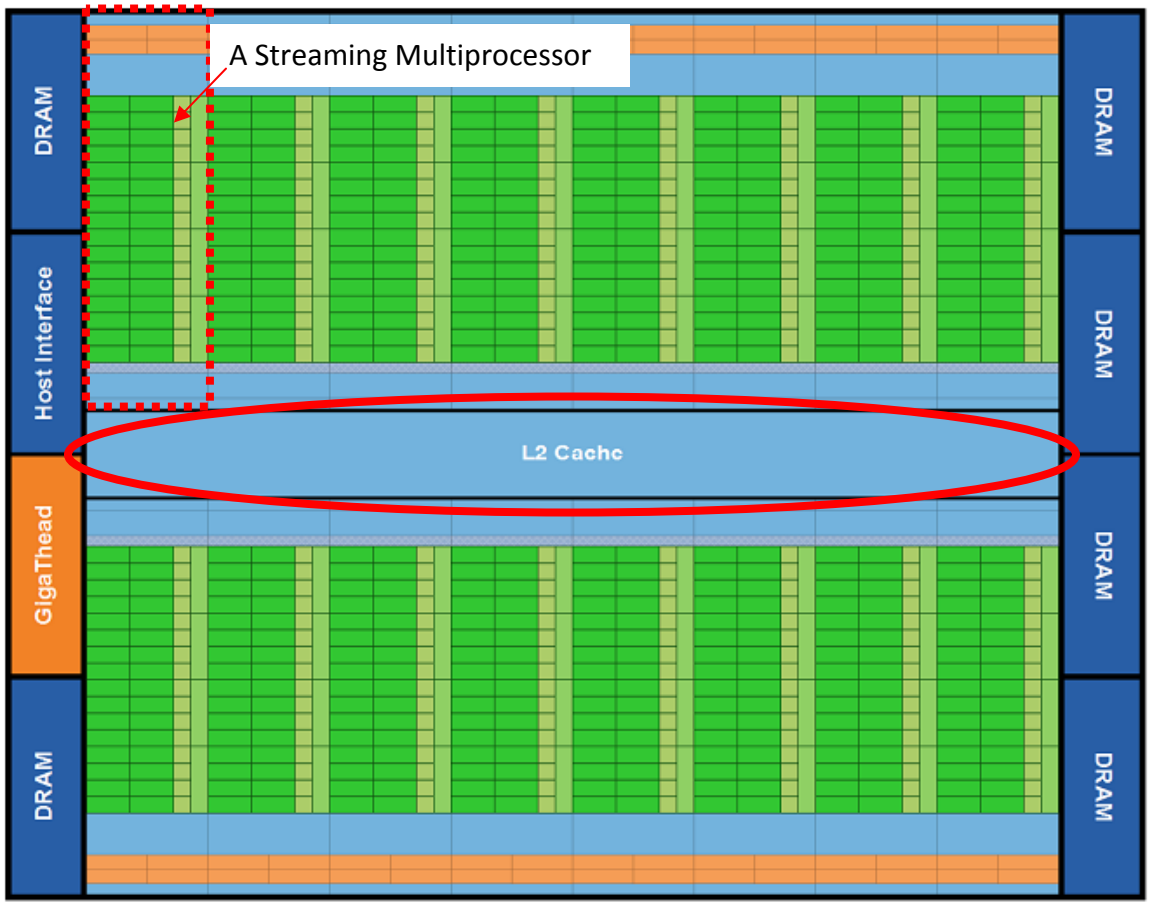


Figure 16 – The Fermi Architecture [12]

The Fermi Architecture is the most current architectural leap in NVIDIA GPU design. Its design improvements include increased CUDA cores per multiprocessor, an increase of CUDA cores in the GPU overall, a dual-warp scheduler for each streaming multiprocessor, a unified address space to support formal pointers, a proper memory hierarchy that includes L1 and L2 caches, and a Giga thread scheduler that can run

kernels concurrently. The important feature to note in Figure 16 is the L2 cache (circled in solid red) and the streaming multiprocessors.

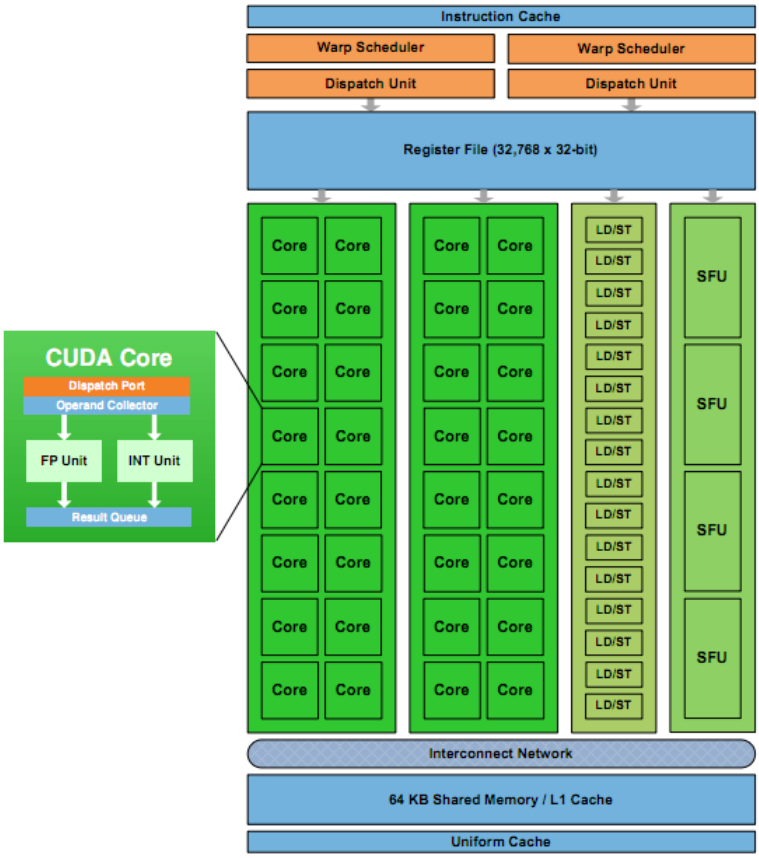


Figure 17 – Streaming Multiprocessor for the Fermi Architecture [12]

Figure 17 is a detailed look at the Streaming Multiprocessor that demonstrates the greater number of CUDA cores per SM, the L1 cache, and the dual-warp scheduler and dispatch unit.

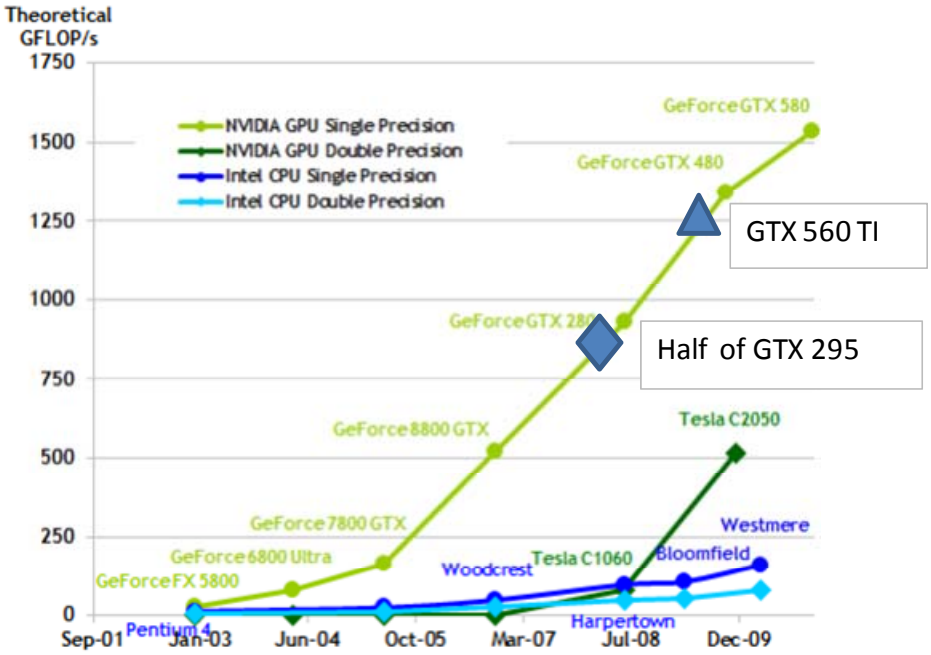


Figure 18 – Comparison between CPU and GPU in FLOPs [13]

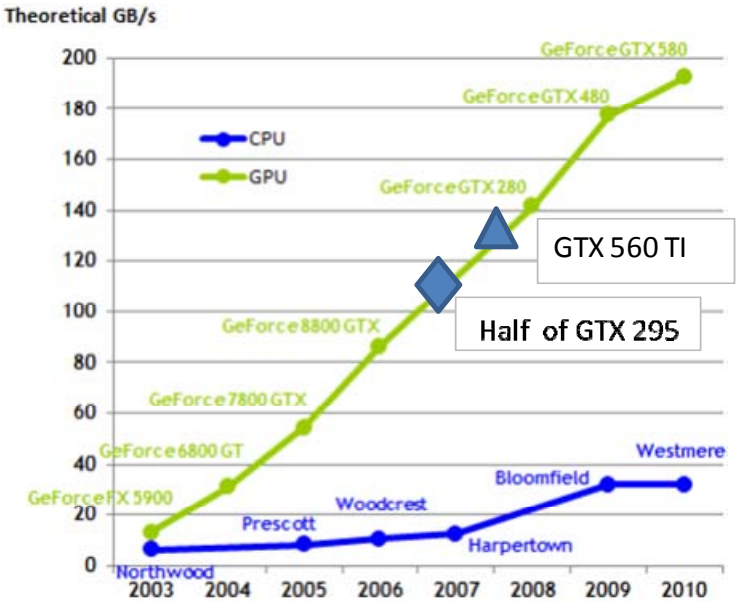


Figure 19 – Comparison between CPU and GPU in Memory Bandwidth [13]

The end result of the evolutionary pressures on GPU is the current, massive gap in computational power and memory bandwidth between CPUs and GPUs. Figure 18 and Figure 19 illustrate this gap.

Due to its highly parallel nature, it is difficult to predict the performance of a CUDA program. For example, sometimes the CUDA card has enough streaming multiprocessors to be able to compute all the blocks at once. Another scenario is that the CUDA program provides too many blocks at once and some of the blocks have to be serialized. Then there is yet another consideration that the kernel threads use so little registers that multiple blocks can run in a single multiprocessor. These factors are examples of what makes performance prediction difficult.

3.5 Unique CUDA Features

Before the Fermi Architecture, NVIDIA GPUs did not have traditional caches to reduce the effect of access to on-card memory. Instead, it uses "shared memory." This is on-chip memory that is shared between threads within a block. It operates like a programmer managed cache, having a read time like a register. Because it is handled by the programmer rather than automatically handled by the hardware, it is more difficult to use.

Another unique feature of the GPUs is the texture memory, a cache that stores spatially local data. It is useful for accessing spatially local data, and it can linearly interpolate between data points. One disadvantage of texture memory is that, to bind data to texture memory, the data must either be a simple 1D memory space or an opaque CUDA array. This means that a separate CUDA array must be allocated each time texture memory is to be used. This becomes a problem with large reconstruction

volumes and low amounts of CUDA global memory. Another weakness of texture memory is that it is read only, which limits its use.

A potential feature for the future is using surface memory. Surface memory can be written to as well as read from. But this new feature is only available in CUDA Compute Capability 2.0. Also, it is limited to 2D, not 3D.

Another unique feature to note is constant memory. As the name implies, constant memory cannot be modified during a CUDA kernel's runtime. It is stored off-chip in the CUDA card's RAM, but it is cached and optimized for when all the threads in a kernel access the same memory location simultaneously. There is a pre-fetch penalty for the first access, but subsequent accesses only have a latency of one cycle.

Now that all the necessary background material has been sufficiently covered, the next chapter reviews the previous work on Cone-Beam Reconstruction Algorithms.

Chapter 4: Previous Work

4.1 Previous Reconstruction Algorithms

Medical Imaging Cone-Beam Reconstruction has been an active topic of research. Due to its computationally intensive nature, a great deal of effort has been put into creating more efficient algorithms. The algorithm proposed by L. A. Feldkamp, L. C. Davis, and J. W. Kress is one of the most popular algorithms used for this field [4]. Cone-Beam reconstructions are known to be computationally intensive, so previous research has focused on acceleration. These efforts include Cone-Beam Reconstruction implementations on FPGAs [16][17][18] and on the CBEA (Cell Broadband Engine Architecture) [14][15]. There has also been work in using GPUs for Cone-Beam Reconstruction using shader languages and OpenGL [19][20]. OpenCL is a competing general purpose GPU (GPGPU) standard, and it has been applied to Cone-Beam Reconstruction [21].

4.2 Previous CUDA Accelerations

NVIDIA CUDA has been a popular research project since its introduction in 2007. This is due to the possibilities opened by its computational power and its widespread application in scientific computing [6]. CUDA has been applied to Cone-Beam reconstruction before. The FDK is a common and basic algorithm, so CUDA accelerations have been done [1]. The computation was executed on a data set of 414 projections of 1024 x 1024 pixels for a 512^3 volume running on a GTX 8800 which had an

execution time of 12 seconds. It also compared the runtimes to the CBEA implementations.

A version of ART called SART (Simultaneous Algebraic Reconstruction Technique) also has been implemented in CUDA [2], although it is less common. The work in [2] used a Quadro FX 5600 on a data set of 228 projections of 256 x 128 pixels and a 512 x 512 x 350 volume. The result is a runtime of 844 seconds for 20 iterations. Also recommended in [2] is the use of 3D texture memory, although the results obtained in this thesis show it is not always beneficial. Note that the ART implementation in this thesis uses FDK as an initialization.

Although there has been research on Beam-Hardening [3], Material Projection is a recently developed technique so there are no CUDA implementations yet.

This concludes the background of this thesis. The next chapter covers the details of the CUDA implementations done in this thesis, followed by test methodology and the results and analysis.

Chapter 5: CUDA Implementations

5.1 Overview

One defining characteristic and limitation of all the reconstruction algorithms is the reconstruction volume size, which, by definition, grows cubically in relation to reconstruction dimension size. This is the limiting factor when considering what kind of data sets can be run on the reconstruction programs. Simply put, a CUDA card cannot run a program that requires more device RAM than what the card has. Typically this is a problem for reconstruction volumes.

Another characteristic that is common for all CUDA implementations is that they all use constant memory for the kernel parameters. One challenge is that due to the complexity of the kernels, a very large number of parameters were needed. Putting them into the parameter list of the kernels would increase the number of registers used and is likely to cause the use of shared memory. Constant memory is a way around that. This is also one of the tips included in the CUDA Best Practices [23].

5.2 Important Points about CUDA

There are a number of possible bottlenecks that can severely reduce parallel computation and performance gains. Often, these are memory bottlenecks. CUDA's extreme parallel nature often turns compute bound problems into I/O bound problems. This is also a descriptive definition of a supercomputer.

The first bottleneck is the memory transfer between the CUDA device and the host system. This is usually the largest bottleneck, because CUDA cards have a limited amount of DRAM/global memory and the PCI-Express bus is the slowest component between the host system and the CUDA device.

The second bottleneck is the Global memory access. Just like in conventional computer systems, accesses to RAM are extremely costly. For CUDA, accesses to the global memory cost hundreds of cycles in latency. CUDA uses the multiple warps in a block to attempt to hide the latency, but optimization typically involves reducing Global memory accesses. Other types of memory such as texture memory, constant memory, and shared memory can be used. Compute capability CUDA 2.0 made the architectural improvement of formal caches, so it reduces the penalty of Global memory accesses.

5.3 CUDA Accelerated FDK

As discussed previously, FDK is based on filtered backprojection. This means FDK can be divided into two parts: Fourier filtering and backprojection.

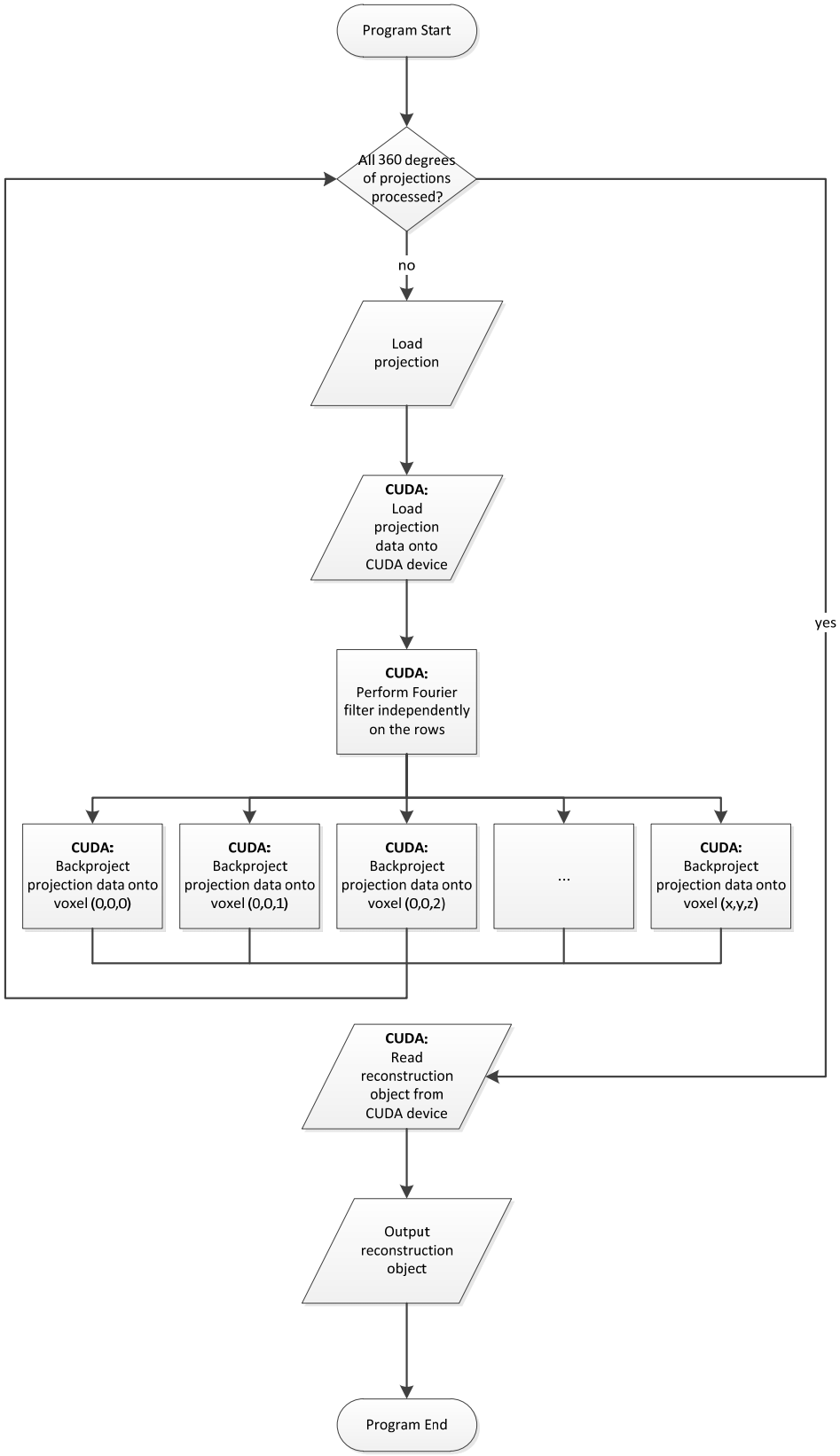


Figure 20 – CUDA Implementation of FDK Algorithm

Fourier filtering is implemented by using the Fourier transform to transform the individual rows of the detector image, multiplying the Fourier signal with the filter, then performing the reverse transformation to get back the filtered signal. CUDA already has an implementation of Fast Fourier Transformation in its SDK, called the CUFFT. A simple signal multiplying kernel and the CUFFT are used to implement the CUDA version of Fourier filtering.

The backprojection is a relatively straight port from the original C++ code. As discussed previously, the code was parallelized along individual voxels. In detail, each voxel would calculate the direction of their X-ray and use that to calculate where it would hit the detector. Then the kernel would use the built-in bilinear interpolation in the texture memory. This feature is important for a couple of reasons: it is a built-in function so it reduces the demand for registers and CUDA cores and it reduces Global memory accesses because the texture memory is cached for spatially localized data.

An attempted version of FDK was the "reworked FDK." This was developed in an effort to achieve the same speedup that was seen in [1]. It achieved greater efficiency by loading all the projection data to the CUDA card in the beginning, and running one kernel for all projections instead of one kernel run for each projection. This was discontinued because it could only work for data set with 360 projections (due to the use of constant memory) and thus it was less flexible. And even though it resulted in a greater speed up, this factor was not enough [1].

5.4 CUDA Accelerated ART

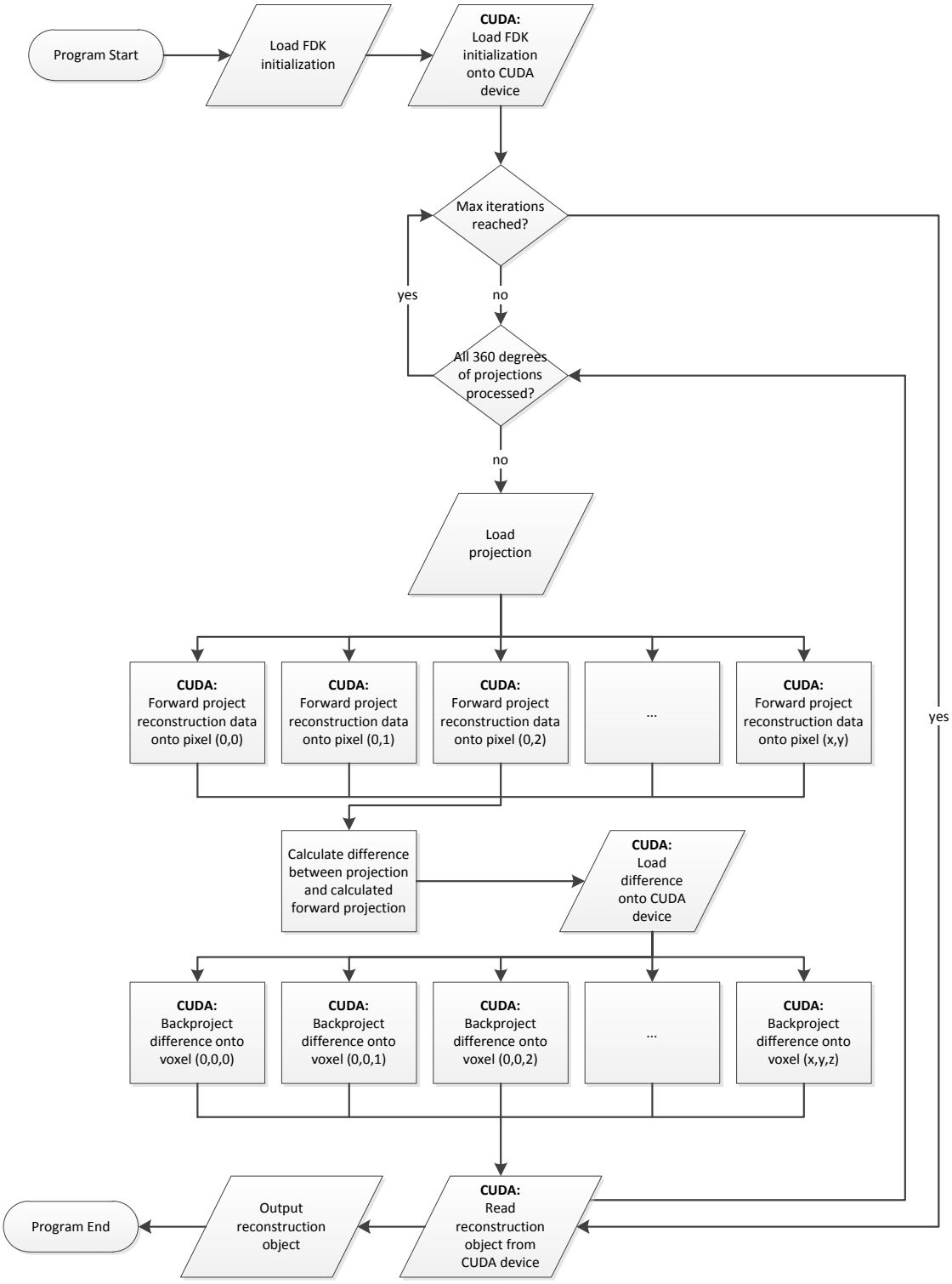


Figure 21 – CUDA Implementation of ART Algorithm

Remember that ART is the theoretical opposite of FDK. As it has been discussed in previous literature [15], using Forward Projection to correct for backprojection, or implementing SART, has been shown to be effective in reducing and eliminating reconstruction artifacts, increasing image sharpness, and increasing overall reconstruction quality.

Forward projection works by calculating the direction of the X-ray for each pixel of the detector, and using ray casting to calculate the attenuation value of that detector pixel. Then the backprojection step uses the difference between the input projection data and the calculated projection data to correct for any differences found between them.

Similar to using texture memory for the 2D detector data, forward projection uses a 3D texture for the reconstruction data. This is done for the same reasons. Unfortunately, due to the previously mentioned reconstruction volume size problem and for CUDA cards with low amounts of global memory, a problem becomes apparent. For the typical reconstruction size of 512^3, a 1 GB CUDA card cannot maintain both the normal reconstruction volume and the CUDA array reconstruction volume that is bound to the texture memory at the same time. What happens then is that the volume has to be copied back to the host each time for the backprojection step. This is not a major problem for reconstruction sizes of 256^3, but the memory transfer takes up far more significant time for 512^3.

5.5 CUDA Accelerated Material ART

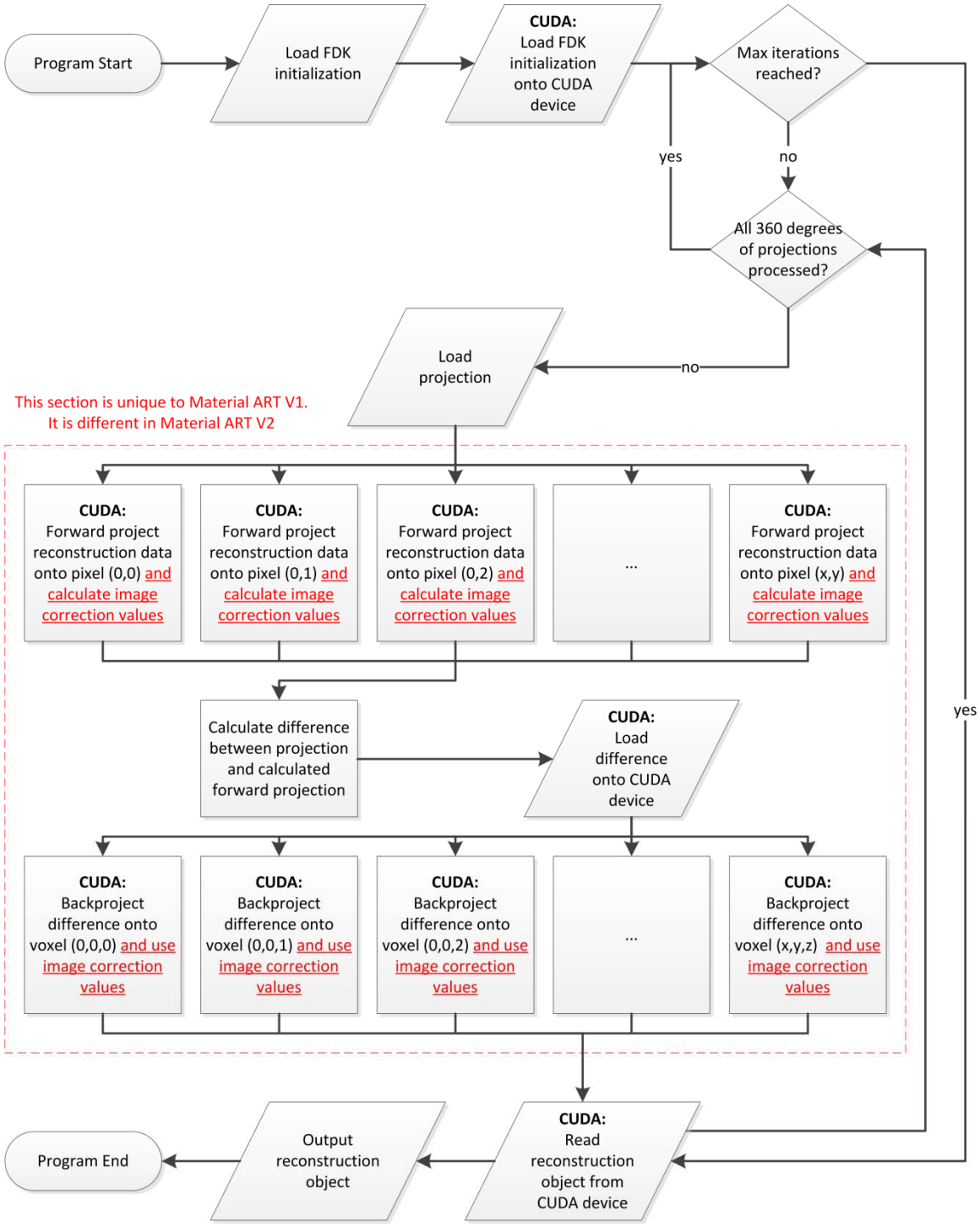


Figure 22 – CUDA Implementation of Material ART V1

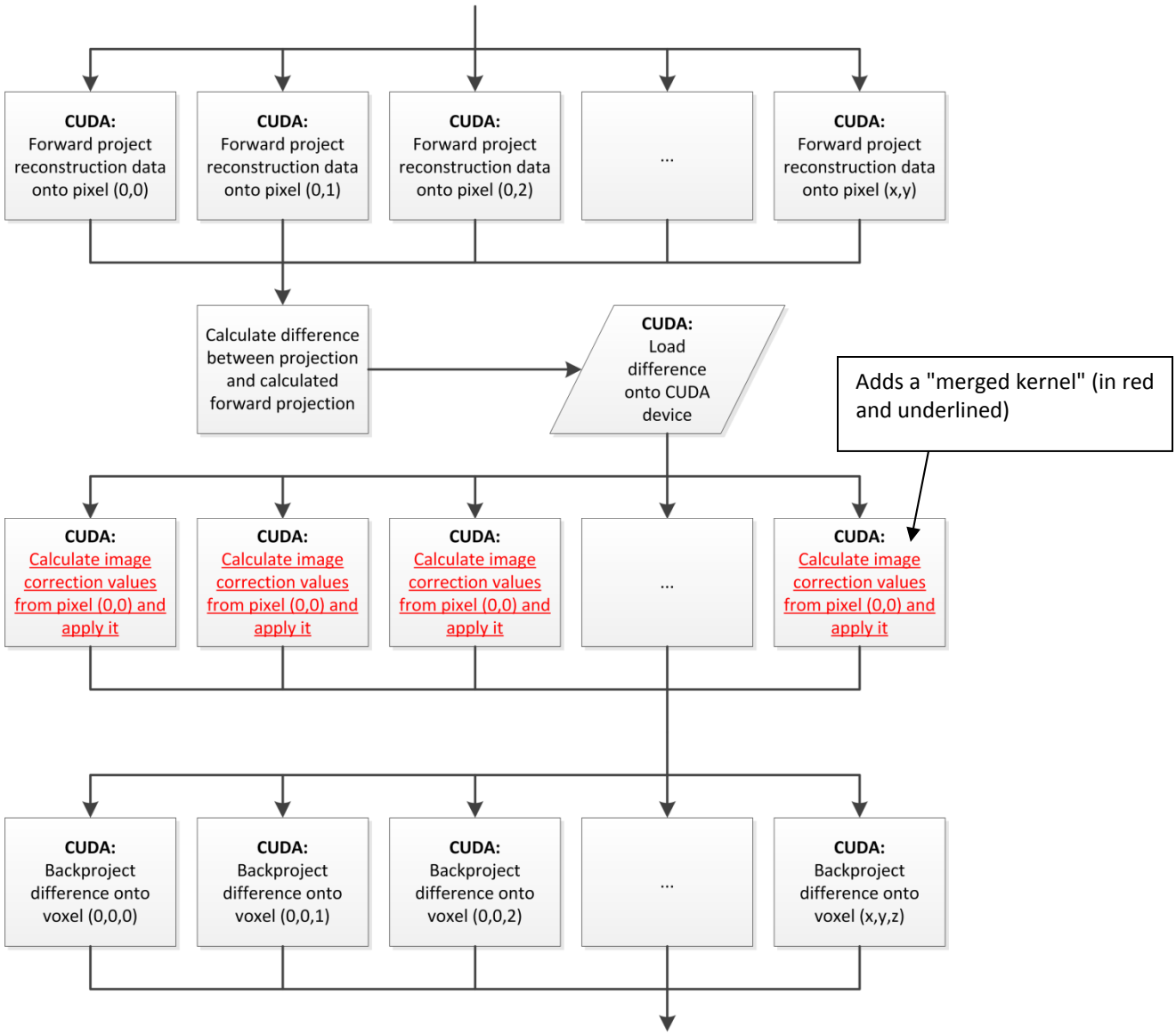


Figure 23 – CUDA Implementation of Material ART V2

Material-ART has its own unique challenges. The Image Correction Factors volume, by definition, is the same size as the reconstruction object. The typical case size for reconstruction volumes is 512^3 floating point values, or 512 MB. However, the typical CUDA card only has less than 1 GB of RAM. This would make it impossible to maintain both the reconstruction volume and the image correction volume in memory at the same time. There are two solutions to this. The first is to use a CUDA card with more

RAM. The second solution is to modify the implementation of the algorithm, or make minor changes to the algorithm itself to eliminate the need to maintain the Image Correction Factors in the main memory. This second solution can be implemented by recalculating the Image Correction Factors on the fly for the reconstruction step. This modification becomes the basis of the "merged kernel" implementation. The merged kernel ray casts from the detector pixel just like the forward projection step, and calculates and applies the image correction values. The merged kernel is separate from the forward projection and back projection, as shown in Figure 23. The disadvantage of this method is the increase in calculation and possible synchronization errors.

There are further notes on the implementation of Material ART. As mentioned before, Material ART is more complex and requires more computation. It also requires additional parameters. This is because during the forward projection step, at the section when Material ART reads from the reconstruction volume, it takes the attenuation of the voxels and classifies them to materials from a materials list received in the parameters. This means that a material ART program would require both a Materials list and the expected attenuation values for the expected materials. The attenuation values come from the National Institute of Standards and Technology.

Ideally, the energy levels per detector pixel and the additional parameters for Material ART should be placed in shared memory. However, they were not. The primary reason is that there is uncertainty in how large of memory space the additional parameters

would take up, and there is a limited amount of memory space of shared memory per each block. Additionally, energy levels must be maintained per ray casted. Because a large amount of threads were grouped per block and the energy spectrum can be divided into an arbitrary number of levels, the memory requirements would exceed shared memory capacity. These reasons combined with the difficulty of developing in CUDA and the generous amount of space in global memory, these additional parameters were delegated to global memory. The cache of the Fermi architecture makes this debate moot, as the cache effectively eliminates the global memory penalty.

One potential source of error is that the actual energy levels of the X-ray sources are not known; since proper calibrations were not done. The energy levels provided for the real data reconstructions are estimates. Even if copper was used to filter the X-ray source, the X-ray source is still polychromatic.

5.6 Development

Development of the CUDA accelerations was done on a TOSHIBA Satellite A665-S6050 Laptop with a GT 330M. The specifications are below:

Table 1 – GT 330M Specifications

Graphics Card	Compute Capability	Number of multiprocessors	Number of CUDA cores	RAM	GFLOPS
GT 330M	1.2	6	48	1GB	182

As running typical size data sets (512^3 reconstruction dimensions) was time consuming and impractical, development was done on the small size (256^3 reconstruction

dimensions) instead, with the assumption that the runtime characteristics remain the same.

In implementation, data transfers between the host and device and global memory accesses were minimized. There are only two exceptions:

1. Because in-development profiling indicated that calculation dominated GPU time, so texture memory was utilized to reduce global memory accesses and overall kernel time. This comes at the cost of increased memory transfers between the host and device. More details are discussed in the analysis of section 7.3.3 ART and Material ART and section 7.3.4 GPU time breakdown.
2. The size of Material ART parameters are not constant and are not guaranteed to keep the same dimensions. Due to these reasons and the low amount of available shared memory, the parameters were placed in global memory.

5.6.1 ART vs. Material ART

ART implements the use of texture memory to both calculate the bilinear and trilinear interpolation and to cache the memory accesses. The Material ART cannot use the interpolation of the texture memory, because it has to classify the accessed voxels before calculating the effective attenuation value. This was considered to be a disadvantage with an expected effect of slower kernel execution due to increased global memory accesses. Furthermore, Material ART has additional parameters that are stored

in global memory as stated before, so the additional global memory accesses are expected to reduce kernel execution performance.

5.6.1.1 Memory Swapping

By necessity of using the texture memory and the size of the typical usage case, the reconstruction volume has to be swapped between the system and the CUDA card. As previously mentioned, the typical dimension size is 512^3 or 512 MB and the older CUDA cards only have 1 GB of memory.

The typical steps of runtime case are:

1. Allocate 3D CUDA array for reconstruction volume
2. Bind 3D CUDA array to texture memory
3. Execute Forward Projection Kernel
4. Free CUDA array
5. Allocate 3D memory space for Backprojection
6. Copy over the reconstruction volume from the system to the CUDA device
7. Execute the Backprojection Kernel
8. Copy back the reconstruction volume from the CUDA device
9. Free the 3D memory space
10. Go back to step 1 if it is not the final projection and iteration

The problem is that because of the limited RAM in the CUDA card, it is not possible to maintain both the 3D volume and the 3D CUDA array at the same time; thus there is a

need for swapping. Material ART is different because it allocates the 3D volume at the start and maintains it throughout the entire runtime, it does not swap the volume.

In-developing profiling shows that the swapping of a 256^3 volume does not significantly impact runtime. More information on this is presented in the results and analysis of section 7.3.3 -- ART and Material ART and section 7.3.4 -- GPU time breakdown.

The next chapter discusses the test to confirm the CUDA implementation of the algorithm.

Chapter 6: Test Methodology

Table 2 – Summary of Test Plan

<table>
<tr><th rowspan="2">Dataset</th><th rowspan="2">Actual Name</th><th rowspan="2">Data Source</th><th colspan="4">Hardwares / Algorithms to be tested</th><th rowspan="2">Remarks</th></tr>
<tr><th>CPU</th><th>G210</th><th>GTX295</th><th>GTX560</th></tr>
<tr><td>P1</td><td>headphantomTiny</td><td rowspan="4">Computer generated</td><td rowspan="9">FDK, ART, MART</td><td>FDK, ART, MART1, MART2</td><td>FDK, ART, MART1, MART2</td><td rowspan="8">FDK, ART, MART1, MART2</td><td rowspan="4">All 1 iteration</td></tr>
<tr><td>P2</td><td>headphantomTiny</td><td>FDK, ART, MART1, MART2</td><td>FDK, ART, MART1, MART2</td></tr>
<tr><td>P3</td><td>headphantomTiny</td><td></td><td>FDK, ART, MART2</td></tr>
<tr><td>P4</td><td>hardeningphantom</td><td></td><td>FDK, ART, MART2</td></tr>
<tr><td>R1</td><td>9300_Patient_321</td><td rowspan="4">CBCT Data</td><td></td><td>FDK, ART, MART2</td><td rowspan="4">All 1 iteration. GTX295 to run 20 iterations on ART, MART1, & MART2</td></tr>
<tr><td>R2</td><td>9300_Patient_349</td><td></td><td>FDK, ART, MART2</td></tr>
<tr><td>R3</td><td>Bacelone_head_R23</td><td></td><td></td></tr>
<tr><td>R4</td><td>Rod_corticalbone</td><td>FDK, ART, MART1, MART2</td><td>FDK, ART, MART1, MART2</td></tr>
</table>

Testing and benchmarking was completed across three graphic cards and eight data sets on a workstation that can support all three graphics cards. Table 2 provides a summary of the test plan. It will be covered in further detail in the following sections. The workstation is a Windows XP SP3 system with an AMD Athlon 64 X2 Dual Core 5600+ processor. Further details are shown in Table 3.

Table 3 – Benchmark Workstation Specifications

System Information	
Operating System	Windows XP Professional SP3
System Manufacturer	Gigabyte Technology Co., Ltd.
System Model	GA-MA790X-DS4
Processor	AMD Athlon(tm) 64 X2 Dual Core Processor 5600+, MMX, 3DNow (2 CPUs), ~2.9GHz
Memory	2814MB RAM
PCI Express 2.0 x16	2
Power Supply	Antec True Power Trio TP3-650 650W Power Supply with Three 12V Rails

6.1 CUDA Graphics Cards

Three CUDA cards were selected to test the performance and behavior of the CUDA programs across different CUDA cards. These cards are:

Table 4 – Testing CUDA Graphics Cards Specifications

card name	compute capability	Number of multiprocessors	Number of CUDA cores	RAM	GFLOPS
G210	1.2	2	16	512MB	69
GTX 295	1.3	2x30	2x240	2x1GB	2x894
GTX 560	2.1	8	384	2GB	1263

Three CUDA graphics cards, G210, GTX 295, and GTX 560, were used to test the performance and behavior of the reconstruction algorithms on CUDA. At present, G210 is a low-end graphics card with 512 MB graphics memory, GTX 295 is a mid-range card with 1 GB graphics memory, and GTX 560 is a high-end card with 2 GB graphics memory. They have different amounts of memory in order to demonstrate the memory limit on the reconstruction algorithms. The GTX 560 has Compute Capability 2.0, compared to Compute Capability 1.2 or 1.3 for other graphics cards; this is a significant jump in

architectural improvements. The GTX 295 is different from others in that it combines two physical NVIDIA GPUs into one graphic card. However, this thesis' CUDA implementation of reconstruction algorithms only utilized one GPU because programming of two GPUs is beyond the scope of this thesis. The CUDA card used for development, the GT 330M, is closer to the G210 rather than the GTX 295.

6.2 Data Sets

Table 5 – Data Sets Description

Dataset	No. of Projections	Detector				Volume				
		Column	Row	Size (KB)	Problem Size (MB)	Column	Row	Slice	Size (MB)	Problem Size (GB)
P1	360	128	128	64	23	128	128	128	8	3
P2	360	256	256	256	90	256	256	256	64	23
P3	360	512	512	1024	360	512	512	512	512	180
P4	360	512	512	1024	360	512	512	512	512	180
R1	246	575	640	1438	345	576	576	450	570	137
R2	449	960	768	2880	1263	576	576	450	570	250
R3	360	600	950	2227	783	608	608	608	857	301
R4	360	600	950	2227	783	300	300	200	69	24

Eight data sets, provided by Carestream Health, Inc., were used for testing. They are divided between computer generated data and real data. The real data refers to projection data produced from CT scans of patients and phantoms. Computer generated data are not produced from X-ray machines. The real data are further distinguished between the patients and phantoms. Phantoms are manufactured objects that test reconstruction algorithms. A well known and commonly used phantom is the Shepp-Logan Phantom. Figure 24 shows the Shepp-Logan Phantom. Computer generated data P1, P2, and P3 are based on the Shepp-Logan Phantom.

Figure 24 – The Shepp-Logan Phantom

Two types of phantoms were used for the computer generated data: the standard phantom and the Material-phantom, a phantom specifically used to test cone-beam correction. Also, because they are generated, different sizes of these phantoms will be used. For real data, CT-scans of patients were used as the final test and benchmark to verify that the algorithms work, and that the ART and Material-ART improve reconstruction quality. Also, CT data of a real phantom was used to detect beam-hardening correction. Four sets of real data were used. Two data sets were selected to have manageable dimensions, while the third was selected because its dimensions are too large for the program to handle; these data sets were selected to verify the memory-bound problem.

Figure 25 – Carestream Health CS 9300 System

The real data sets R1 and R2 were scanned in the CS 9300 system, while the real data set R3 was scanned in the KODAK 9500 Cone Beam 3D System. The real data set R4 was scanned in an unspecified scanner.

The real data sets were used to test for image quality. Visual inspection was used to determine image quality; the results from multiple programs running on the same data set were compared. The FDK results served as the baseline, as it has the fastest runtime but the poorest image quality. The FDK results were compared to the ART and Material ART results. Between the FDK and ART results, specific details were examined, such as image sharpness of image details and the correction of artifacts. Within the ART and Material ART, the image qualities between different numbers of iterations were compared to find the number of iterations needed to produce optimal image quality.

The results of the two Material ART implementations were compared to confirm the validity of the alternate Material Art implementation and to make sure the second Material ART implementation did not produce additional artifacts.

6.3 Expected Results

Before benchmarking was performed, there were a number of expectations for the results. These expectations are as follows:

- The newer GPU should have faster runtimes than the older ones.
- FDK will be the fastest algorithm, followed by ART, Material ART V1, and Material ART V2.
- The main limitation of what data sets can be run on which CUDA cards will be dictated by the amount of RAM the CUDA card has. It is expected that the G210 can only run the smallest reconstruction sizes (P1, P2, and R4) since it has only 512 MB of RAM. GTX 295 is expected to be able to run almost all of the cases except R3 due to its extremely large reconstruction size.

There are additional expectations for Material ART. Because the real data sets R1, R2, and R3 have their source filtered through copper, they effectively have minimal beam-hardening; Material ART will not produce a significant difference compared to the ART results. This is why R4 was selected to confirm beam-hardening.

All the necessary data was collected by following this test plan. The next chapter displays the test results and analysis.

Chapter 7: Results and Analysis

7.1 Image Quality

The real data sets R1, R2, and R3 are CT scans of patient's heads. Of these, the features of interest are the molars, molar implants (e.g., dental fillings), sinus bones, and ear bones. The reconstruction slices are shown because they either have a feature of interest or they showcase the elimination of an artifact. R4 is a slab of acrylic with six rods of bone-like material. It was made to be the same as the Hardening Phantom P4.

The first criterion for the results of the CUDA accelerations is the image quality of the output. As stated by the proposed testing, there are several points that have to be confirmed. These are:

1) Confirm that the GPU produces images that are the same quality or better quality compared to the CPU output
2) Confirm the quality improvement of ART over FDK and determine how many iterations are needed
3) Confirm that Material ART confirms beam-hardening
4) Confirm Material ART V2 produces the same output images as Material ART V1

Because there are a very large number of images to be presented, the images are placed in Appendix A and Appendix B. The image quality analysis is done through visual

inspection; what we are interested in is the removal of image artifacts and overall increase in image quality.

7.1.1 GPU vs. CPU

Table 6 – Image Comparison for Appendix A

Visual Image Comparison on Reconstruction Algorithms Ran on CPU & GTX560

Dataset	Algorithm	Image Slice	Image Comparsion		
			CPU		GTX560
R1	FDK	322	A1	=	A2
R1	ART	322	A3	<	A4
R1	MART1	322	A5	=	A6
R1	MART2	322	A7	=	A8
R2	FDK	135	A9	=	A10
R2	ART	135	A11	<	A12
R2	MART1	135	A13	=	A14
R2	MART2	135	A15	=	A16
R4	FDK	38	A17	=	A18
R4	ART	38	A19	<	A20
R4	MART1	38	A21	=	A22
R4	MART2	38	A23	=	A24

= Images are similar per visual comparison

< GPU Image is better than CPU Image per visual comparison

Appendix A contains selected slices of the reconstructed volumes for the real data sets. The CPU results are on the left side. The GPU results are on the right side. The point of Appendix A is to confirm that the GPU implementations either produce images that are

the same output or of better quality than images of CPU implementations. Through visual inspection, we can confirm this is true.

7.1.2 ART vs. FDK

Table 7 – Images Comparison for Appendix B

Visual Image Comparison of 20 Iterations of Iterative Reconstruction Algorithms Ran on GTX560

Dataset	Algorithm	Image Slice	Image Comparsion								
			FDK		Iteration 1		Iteration 5		Iteration 10		Iteration 20
R1	ART	322	B1	<<	B2	<<	B3	<<	B4	<	B5
R1	MART1	322	B6	<<	B7	<<	B8	<<	B9	<	B10
R1	MART2	322	B11	<<	B12	<<	B13	<<	B14	<	B15
R1	ART	364	B16	<<	B17	<<	B18	<<	B19	<	B20
R1	MART1	364	B21	<<	B22	<<	B23	<<	B24	<	B25
R1	MART2	364	B26	<<	B27	<<	B28	<<	B29	<	B30
R2	ART	135	B31	<<	B32	<<	B33	<<	B34	<	B35
R2	ART	190	B36	<<	B37	<<	B38	<<	B39	<	B40
R3	ART	443	B41	<<	B42	<<	B43	<<	B44	<	B45
R3	ART	527	B46	<<	B47	<<	B48	<<	B49	<	B50
R4	ART	38	B51	<<	B52	<<	B53	<<	B54	<	B55
R4	MART1	38	B56	<<	B57	<<	B58	<<	B59	<	B60
R4	MART2	38	B61	<<	B62	<<	B63		B64		B65
R4	ART	150	B66	<<	B67	<<	B68	<<	B69	<	B70
R4	MART1	150	B71	<<	B72	<<	B73		B74		B75
R4	MART2	150	B76	<<	B77	<<	B78		B79		B80

<< Image of higher iteratives is much better than the image of lower iteratives per visual comparison

< Image of higher iteratives is better than the image of lower iteratives per visual comparison

Artifacts in the images prevented the visual comparison

Appendix B confirms the image quality improvement that comes with the iterative methods: ART and Material ART. The results show that single iteration is insufficient, and there are diminishing returns past five iterations. By the twentieth iteration, the reconstruction has converged and the noise becomes accentuated.

Specific cases are examined in detail.

7.1.2.1 ART improvement in Slice 332 of R1 Data set

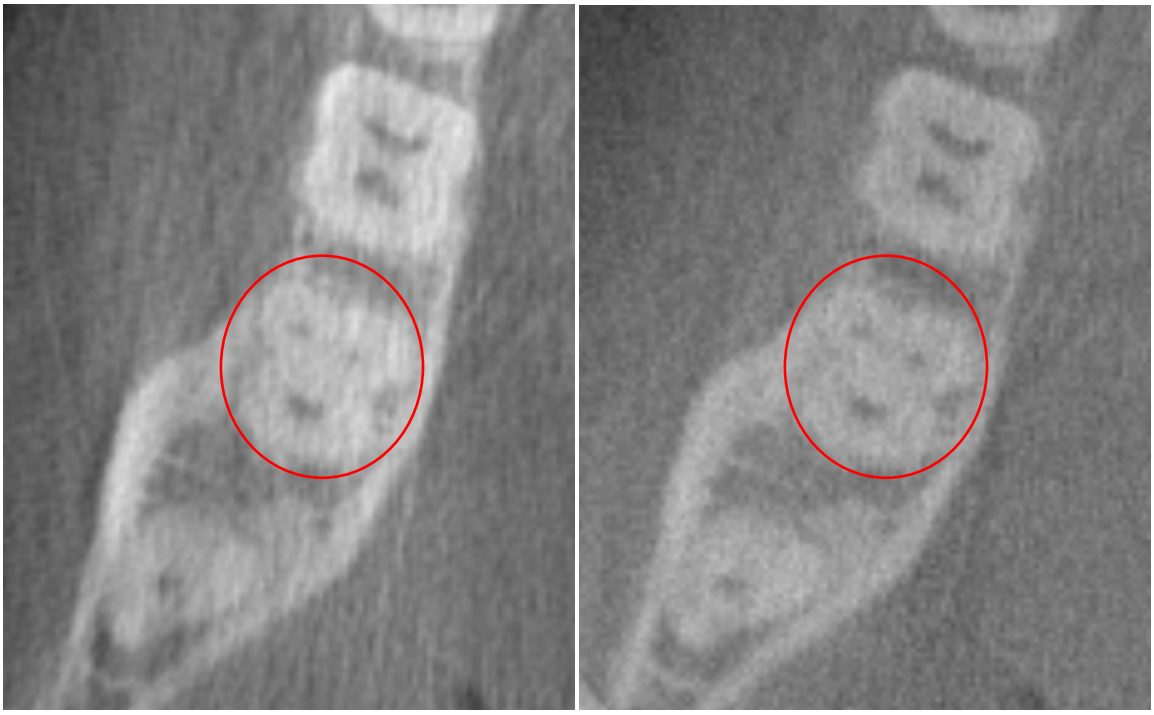


Figure 26 – B1 (R1, FDK, Slice 322) [left]; B5 (R1, ART Iteration 20, Slice 322) [right]

The first image quality comparison is between the FDK reconstruction and the 20th iteration of ART. Overall, there is an increase of image sharpness and a reduction of artifacts. An example of this is Figure 26. Figure 26 shows the reduction of image streaking around the teeth and the increase of sharpness of the "monkey face."

The focus of this comparison is the "monkey face," or the distinctive molar at the bottom right section of the slice. The monkey face is encircled in red ink in Figure 27. Compared to the FDK, the monkey's right eye in the ART is a lot clearer and is representative of an overall increase in sharpness. Also, please notice the elimination of

the vertical streaking in the ART. This elimination is important for medical imaging reconstruction.

7.1.2.2 Image improvement for Slice 364 of R1 Data set

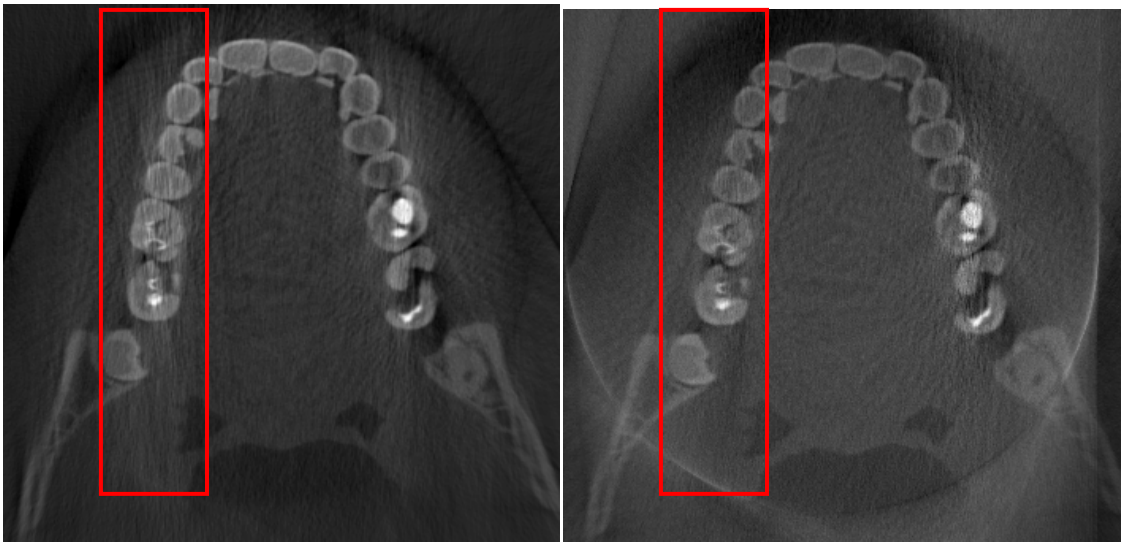


Figure 27 – B16 (R1, FDK, Slice 364) [left]; B20 (R1, ART Iteration 20, Slice 364) [right]

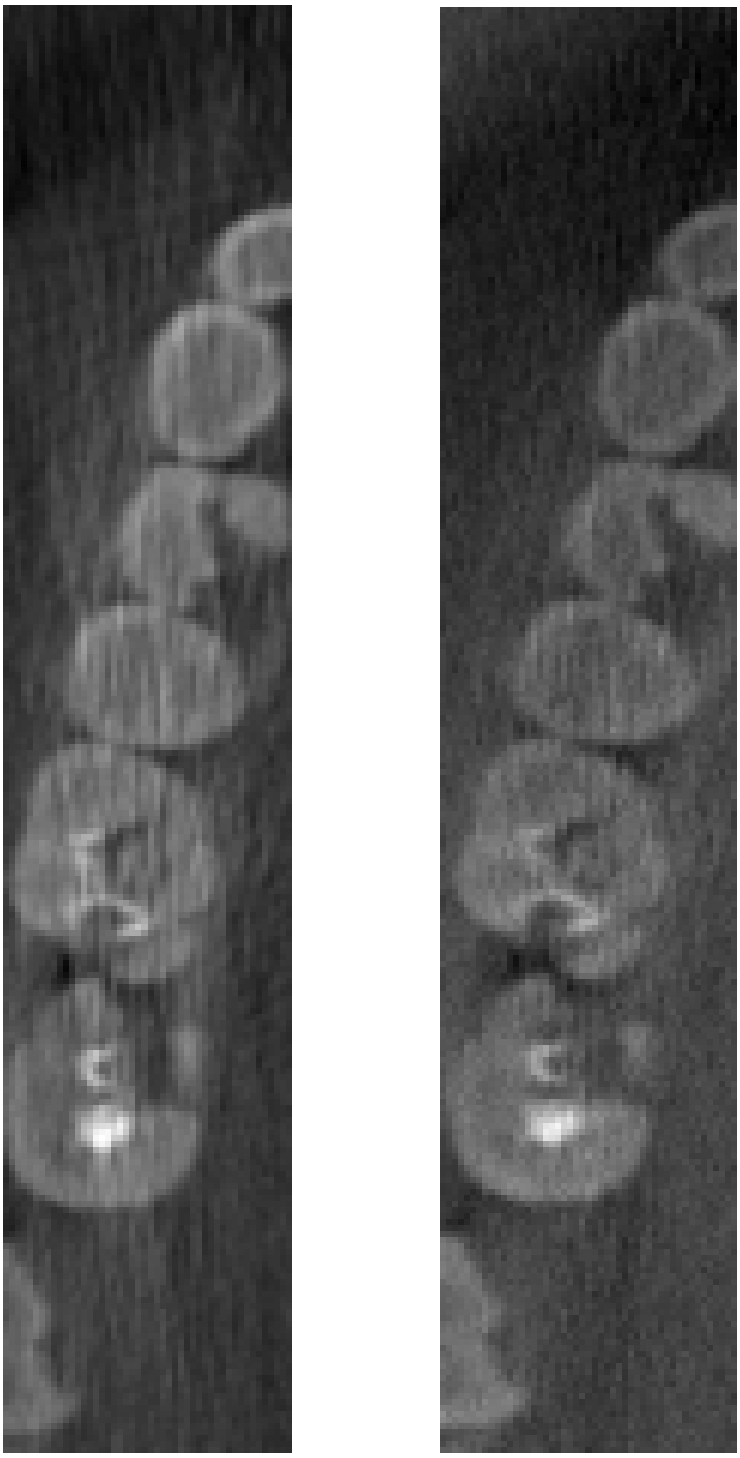


Figure 28 – Elimination of Vertical Streaking

The slice 364 provides another example of streaking reduction; it is the grey-white vertical pattern that is outlined in the red rectangle and eliminated as seen in Figure 27

and Figure 28. Also, ART cannot correct for the scattering of the molar implant, or the tooth filling in the molar on the right side. It produces a strong scattering artifact because it is made of metal.

7.1.3 Comparison of Material ART Versions

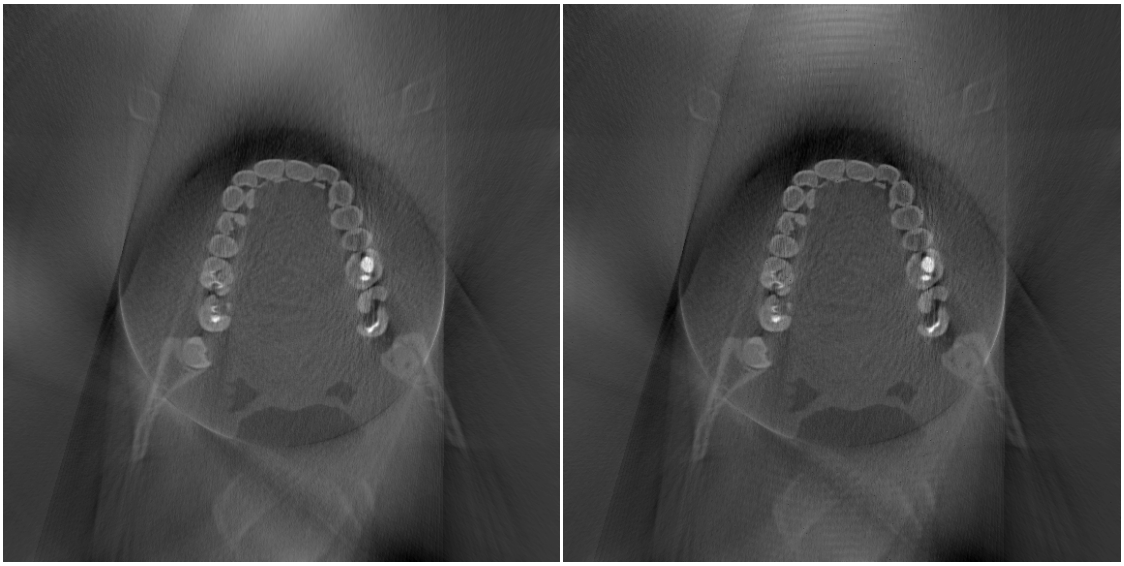


Figure 29 – B25 (R1, MART1 Iteration 20, Slice 364) [left]; B30 (R1, MART2 Iteration 20, Slice 364) [right]

A comparison between these two images in Figure 29 shows that material ART for both versions produce the same output, with the exception of minor ring artifacts produced by version 2. However, the ring artifacts are outside the truncation artifact. All images in Appendix A and Appendix B confirm that both Material ART V1 and Material ART V2 produce the same output.

7.1.4 Confirmation of Material ART

As expected from the R1 results displayed in Appendix B, Material ART did not produce a significant difference from ART output.

What was surprising is that ART appears to have corrected artifacts thought to have been caused by beam-hardening in R4. See images of slice 38 and 150 in Appendix B. The results for R4 are inconclusive, since an induced ring artifact prevents proof by inspection.

Therefore, P4 is the only data set that can conclusively confirm the beam-hardening of Material ART.

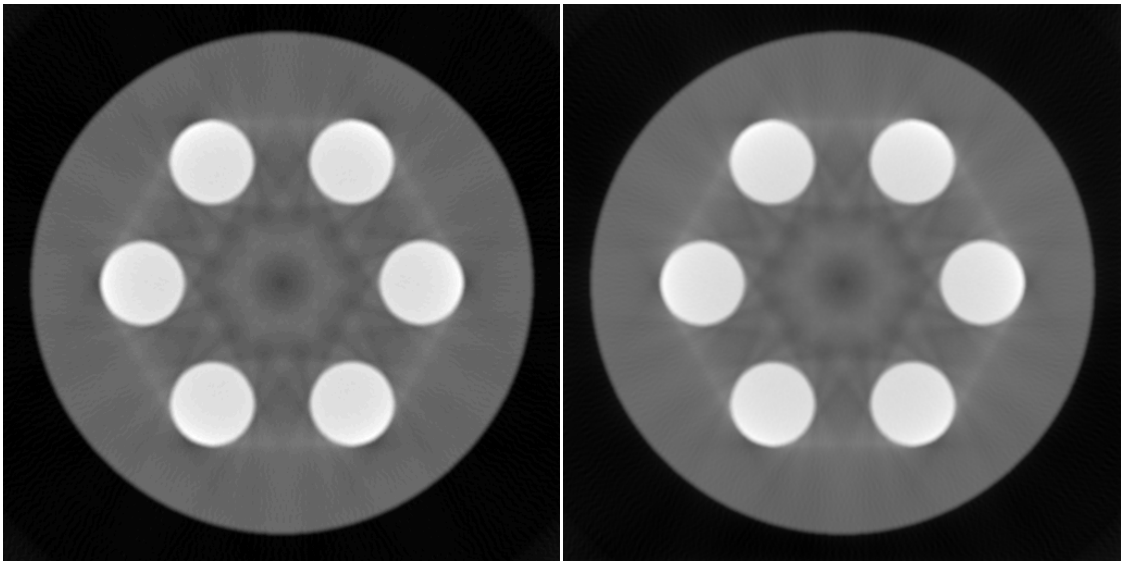


Figure 30 – Comparison between FDK and ART

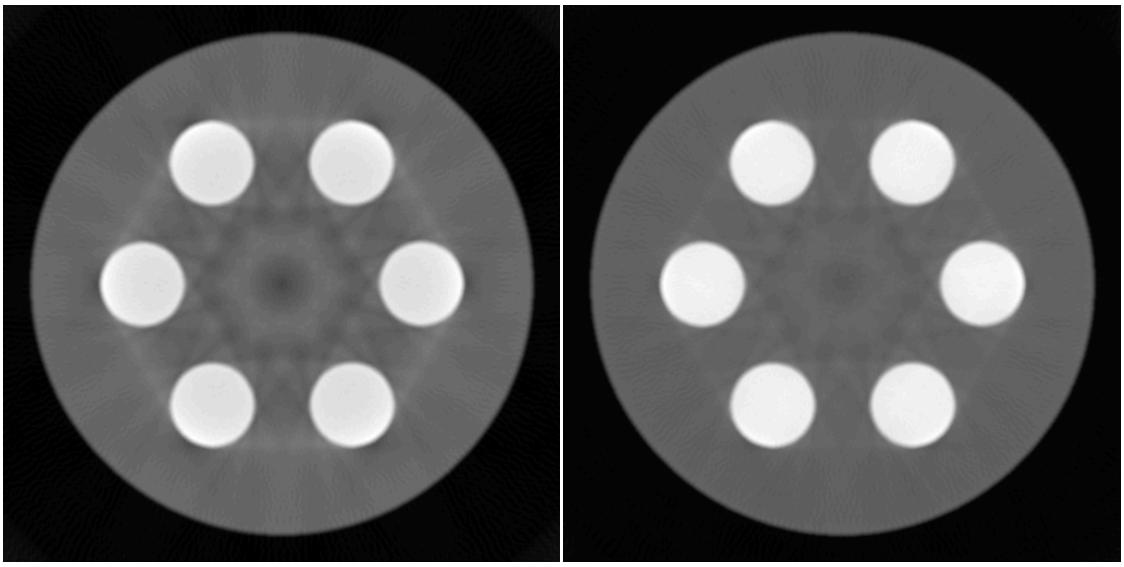


Figure 31 – Comparison between FDK and Material ART V1

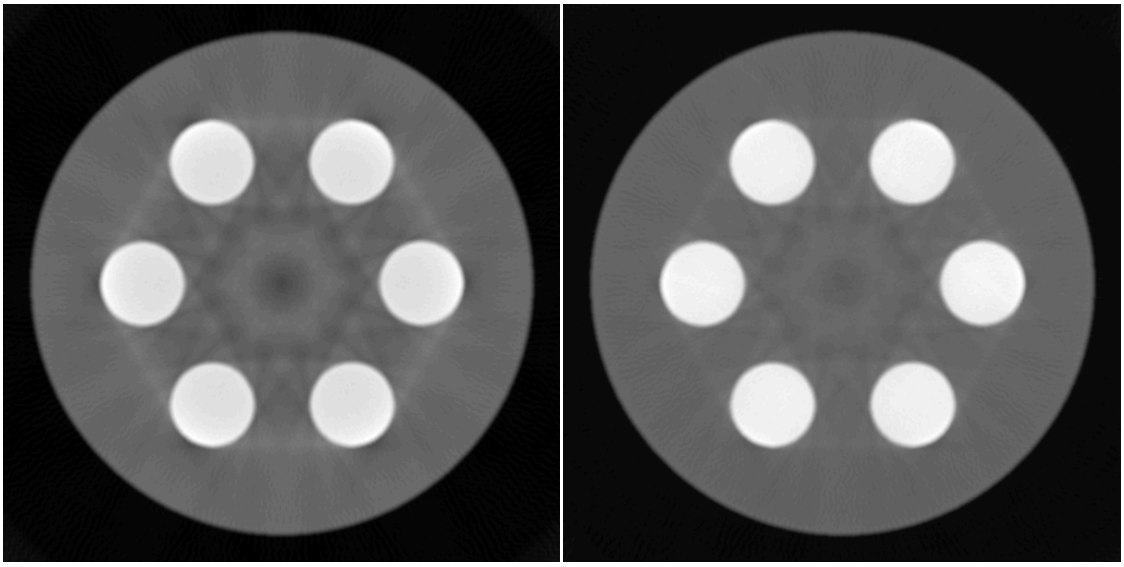


Figure 32 – Comparison between FDK and Material ART V2

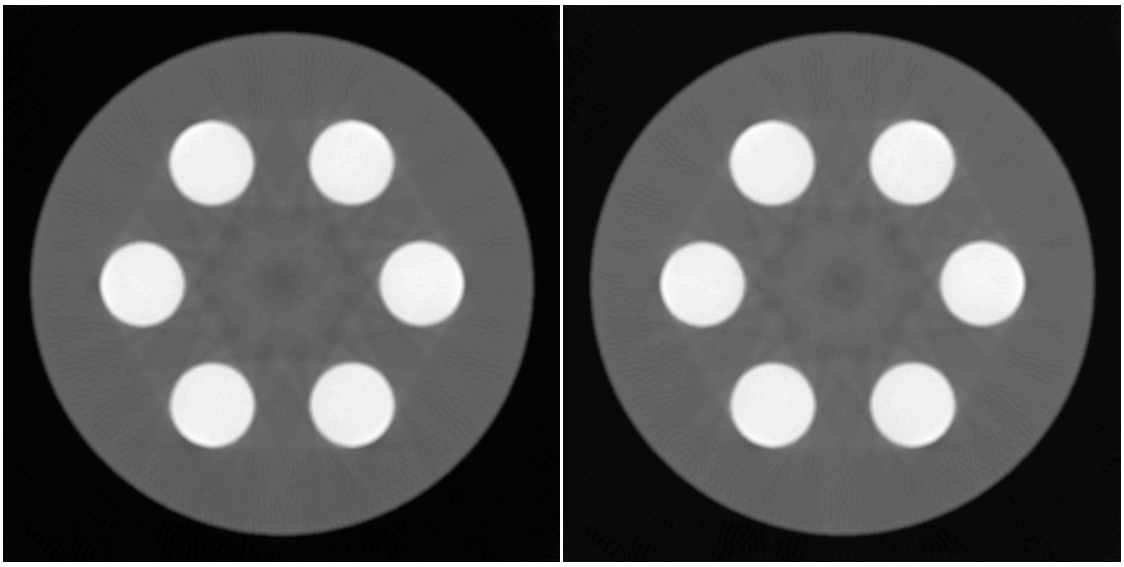


Figure 33 – Comparison between Material ART V1 and V2

Here, the results in Figure 30, Figure 31, Figure 32, and Figure 33 are conclusive. The FDK reconstruction displays strong beam-hardening artifacts that ART cannot fix, but the Material ART versions can, even after a single iteration. Figure 33 demonstrates that both implementations produce the same result.

7.1.5 Truncation Artifacts

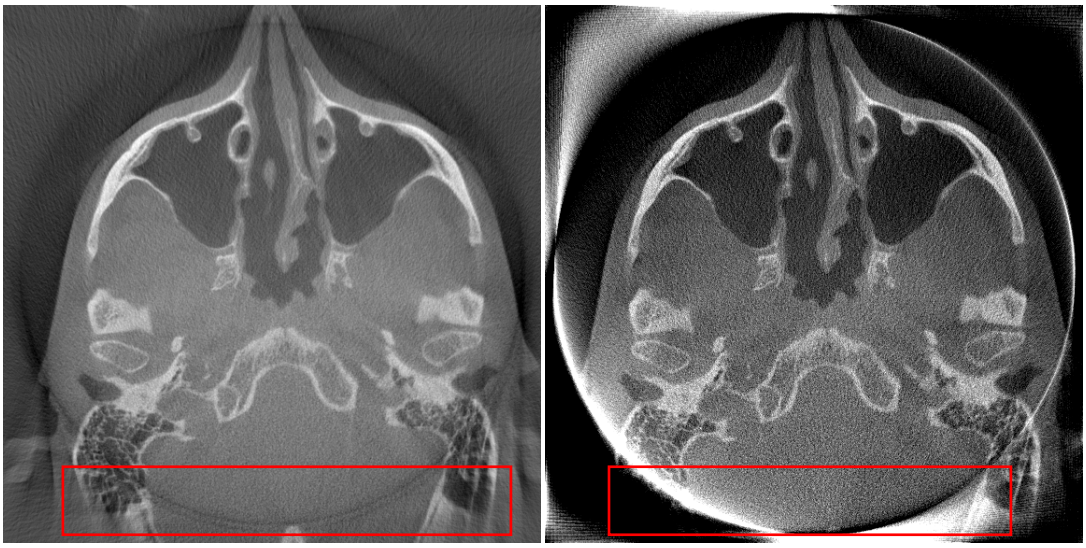


Figure 34 – Truncation Artifact in B31 (R2, FDK, Slice 135) [left]; B35 (R2, ART Iteration 20, Slice 135) [right]

One way to classify artifacts is by the cause of the artifact. Physics-based artifacts result from the physical processes involved in the acquisition of CT data. Patient-based artifacts are caused by factors related to the patient's form or function. Scanner-based artifacts result from imperfections in scanner function.

Another artifact that appears in the images of Appendix B is the truncation artifact. Truncation artifacts are a known phenomenon; they have appeared in previous literature as a scanner-based artifact [22]. Figure 35 shows a scenario where truncation artifacts can occur; the detector is too small, so parts of the object are outside the field of view. This means that in the reconstruction, there is insufficient information for those parts of the object. Only the parts of the object that are in the field of view are fully reconstructed.

Figure 34 shows this truncation artifact; it is the circle that gets stronger for the ART image. FDK has this truncation artifact, but it is far stronger in the iterative methods. This is because ART and Material ART use forward projection to apply image correction, and that accentuates the object within the field of view at the cost of what is outside the field of view.

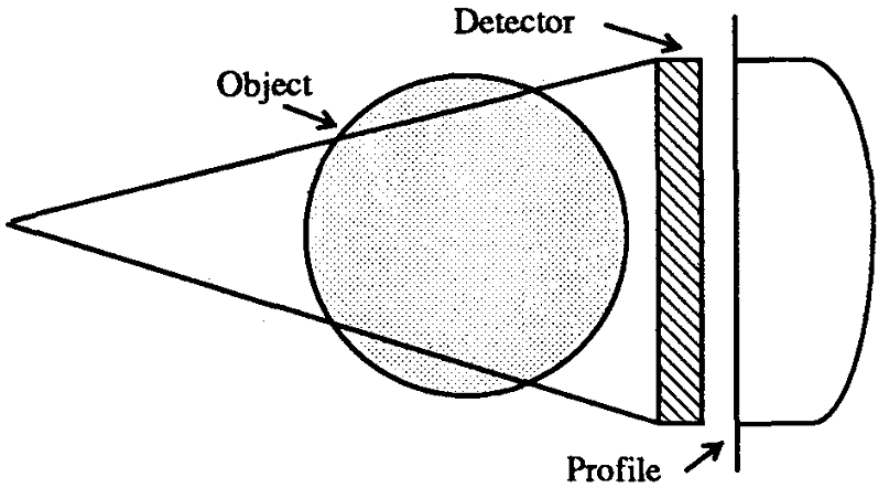


Figure 35 – Truncation Scenario [22]

7.2 Runtime Results

Table 8 – Runtimes for Reconstruction Algorithms ran on CPU & Graphics Cards

Reconstruction Algorithms on CPU and Various Graphics Cards
One iteration

	FDK Algorithm			
Dataset	cpu	g210	gtx295	gtx560
P1	259	18	8	7
P2	1,996	75	23	20
P3	15,823		99	87
P4	15,915		108	95
R1	6,355		107	87
R2	20,479		239	200
R3	12,341			164
R4	1,592	150	89	92

	ART Algorithm			
Dataset	cpu	g210	gtx295	gtx560
P1	746	56	30	27
P2	5,894	272	127	120
P3	50,909		851	830
P4	50,946		841	806
R1	38,105		660	599
R2	149,988		1,328	1,192
R3	91,864			1,239
R4	23,430	688	299	269

	Mtl ART Algorithm, Ver. 1			
Dataset	cpu	g210	gtx295	gtx560
P1	1,044	95	26	20
P2	8,295	556	107	70
P3	76,984			492
P4	77,257			505
R1	69,413			318
R2	279,566			929
R3	173,746			672
R4	44,620	3,487	568	239

	Mtl ART Algorithm, Ver. 2			
Dataset	cpu	g210	gtx295	gtx560
P1	1,044	143	30	20
P2	8,295	938	147	76
P3	76,984		1,256	607
P4	77,257		1,280	518
R1	69,413		1,286	351
R2	279,566		4,998	1,244
R3	173,746			752
R4	44,620	6,841	979	283

Data point is the process time of one iteration in seconds
Indicates the workstation could not run the algorithm on the data set

The runtime results of Table 8 represent the runtimes of each test case. The runtimes are defined as the time it takes for the program to run for the test case, from the beginning to the end. This gives us a very high level view of the performance gain, as it gives the actual performance gain instead of just focusing on the accelerated portions of the programs.

7.2.1 Overview

First, the runtimes are relatively consistent, as Appendix C demonstrates. Appendix C is the graphs and tables that detail the multiple iteration runs. The consistencies of the runtimes show that the runtimes are unlikely to change significantly between different runs. This allows us to run each case once with confidence that the runtime obtained is representative.

Second, the runtimes between the reconstruction algorithms are as expected, with the exceptions having interesting implications. The FDK is the fastest algorithm overall, with ART being the next fastest and the Material ART being the slowest. However, the architectural improvements in GTX 560 change this relationship. These architectural improvements will be discussed in further depth later.

Third, the runtimes between the different CUDA cards are as expected. The weakest card preformed the worst, while the GTX 560 performed the best. In fact, the performance of the GTX 560 exceeded expectations, highlighting the architectural

design improvements made between compute capabilities. This will be discussed further.

Fourth, the memory limitations of the data sets were as expected. These limitations were predicted in advance, and the ones predicted to crash due to memory limits did so.

7.2.2 Runtime Confirmations

As mentioned before, the results and graphs in Appendix C confirm the constant runtime for program runs. An additional check is P3 and P4. These two data sizes are exactly the same, but they reconstruct different volumes. This also confirms that it is the data set dimensions that determine the runtime, not the contents of the data set.

7.2.3 Varying Problem Size

Table 9 – Runtime Scaling for P1, P2, and P3

Reconstruction Algorithms on CPU and Various Graphics Cards
One iteration

	FDK Algorithm			
Dataset	**cpu**	**g210**	**gtx295**	**gtx560**
P2/P1	7.7	4.2	2.9	2.9
P3/P2	7.9		4.3	4.4
P3/P1	61.1		12.4	12.4

	ART Algorithm			
Dataset	**cpu**	**g210**	**gtx295**	**gtx560**
P2/P1	7.9	4.9	4.2	4.4
P3/P2	8.6		6.7	6.9
P3/P1	68.2		28.4	30.7

	Mtl ART Algorithm, Ver. 1			
Dataset	**cpu**	**g210**	**gtx295**	**gtx560**
P2/P1	7.9	5.9	4.1	3.5
P3/P2	9.3			7.0
P3/P1	73.7			24.6

	Mtl ART Algorithm, Ver. 2			
Dataset	**cpu**	**g210**	**gtx295**	**gtx560**
P2/P1	7.9	6.6	4.9	3.8
P3/P2	9.3		8.5	8.0
P3/P1	73.7		41.9	30.4

Data point is the process time of one iteration in seconds
Indicates the workstation could not run the algorithm on the data set

As the computer generated phantoms P1 through P3 demonstrated, the problem size exhibits cubic growth with increasing dimension size; this is proven by the CPU runtime increasing by eight every time the dimensions were doubled. The scaling for GPU runtime is more complicated, but it is always less than the CPU scaling factor. The implication of this is that the GPU scales better than the CPU, so there is more performance improvement for data sets with high dimensions than smaller dimensions.

7.2.4 Overall Performance Gains

Table 10 – Runtimes for Reconstruction Algorithms ran on CPU & Graphics Cards

Speed up of Reconstruction Algorithms on Various Graphics Cards over CPU

	FDK Algorithm		
Dataset	g210	gtx295	gtx560
P1	14	32	37
P2	27	87	100
P3		160	182
P4		147	168
R1		59	73
R2		86	102
R3			75
R4	11	18	17

	ART Algorithm		
Dataset	g210	gtx295	gtx560
P1	13	25	28
P2	22	46	49
P3		60	61
P4		61	63
R1		58	64
R2		113	126
R3			74
R4	34	78	87

	Mtl ART Algorithm, Ver. 1		
Dataset	g210	gtx295	gtx560
P1	11	40	52
P2	15	78	119
P3			156
P4			153
R1			219
R2			301
R3			258
R4	13	79	187

	Mtl ART Algorithm, Ver. 2		
Dataset	g210	gtx295	gtx560
P1	7	35	52
P2	9	56	109
P3		61	127
P4		60	149
R1		54	198
R2		56	225
R3			231
R4	7	46	158

Data point is the process time of CPU / the process time of the graphics card
Indicates the workstation could not run the algorithm on the data set

As previously discussed, the runtime results fulfilled the expectation that each more powerful card will produce faster runtimes, with a possible exception being ART. The runtime results also fulfilled the expectation that the FDK is the fastest algorithm. The relation between ART and Material ART (both versions) is more complicated and is a topic for discussion.

Speaking at a very high level and general trend, the G210 experiences a 16x speedup, the GTX 295 an 50x speedup, and the GTX 560 experiences speedups that range from 100x to 297x.

One way to judge if a speedup is sufficient for a CUDA card is to compare it to the number of CUDA cores of that card. This is because the number of CUDA cores represents the number of operations that can run simultaneously in parallel. The overall analysis is that the G210 reached sufficient throughput, the GTX 295 needs improvement, and the GTX 560 also reached sufficient throughput. This is because the GTX 295 had reached only 50x improvement for a 128 card. The reasons for this include the unoptimized sections of the CUDA accelerations. However, the GTX 560 was able to run at near peak efficiency due to its cache being able to blunt the penalty of unoptimized code.

7.3 Unexpected Results

7.3.1 Comparing Data Sets

As discussed previously, the relation between the different computer generated data sets has fulfilled predictions.

As mentioned before, the real data sets were selected for specific reasons. R1 and R2 were selected to have reconstruction volume dimensions to the specified typical size,

and R3 was selected for having above typical reconstruction dimension size. R4 was selected to confirm beam-hardening for real data. However, unexpected runtime behavior arose.

7.3.1.1 Comparison of Real Data Sets R1, R2, & R3

Initially, it was not expected for R2 runtimes to be so large, and even comparable to R3. However, closer inspection reveals the reasons why. Out of all the real data sets of patient data, R1 is the absolute smallest. Although it has the same reconstruction dimensions of R2, it has much smaller detector size and much lower number of projections. These two have an additive effect to reduce runtime. The detector size is especially influential to the runtime for the iterative methods.

By definition, R2 has the opposite relationship to R1. R2 has 449 projections compared to the 246 projections of R1. The difference between the numbers of projections is a factor of 1.8; this relationship cannot account for the entire performance gap. Therefore, only the detector size can account for the discrepancy.

R3 has too many parameter differences to R2 to effectively compare to it. The different number of projections, different detector size, and different reconstruction size makes it impossible to determine its relation with R2. This follows the point made previously; it is difficult to predict the runtime of a CUDA program before the fact.

7.3.2 Observation

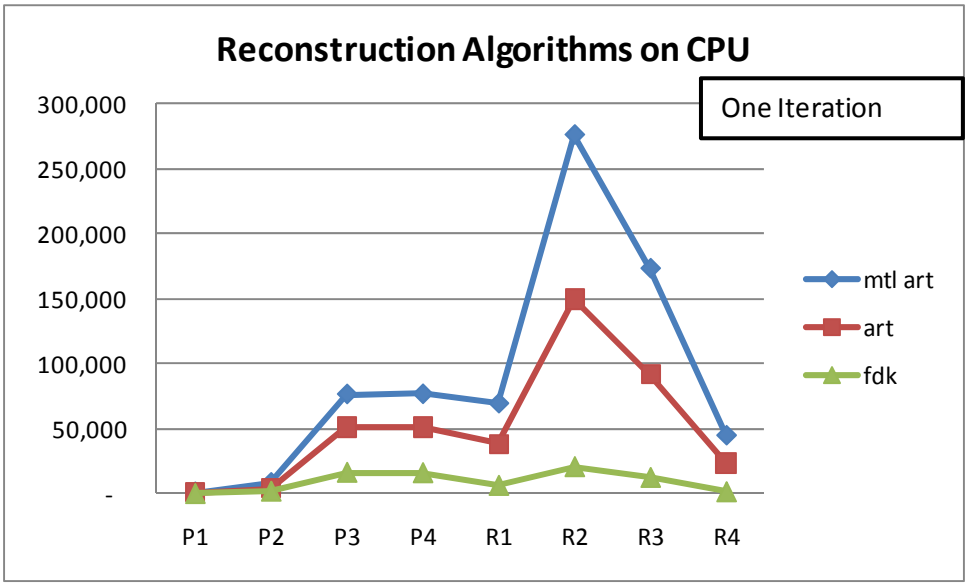


Figure 36 – Reconstruction Algorithms on CPU

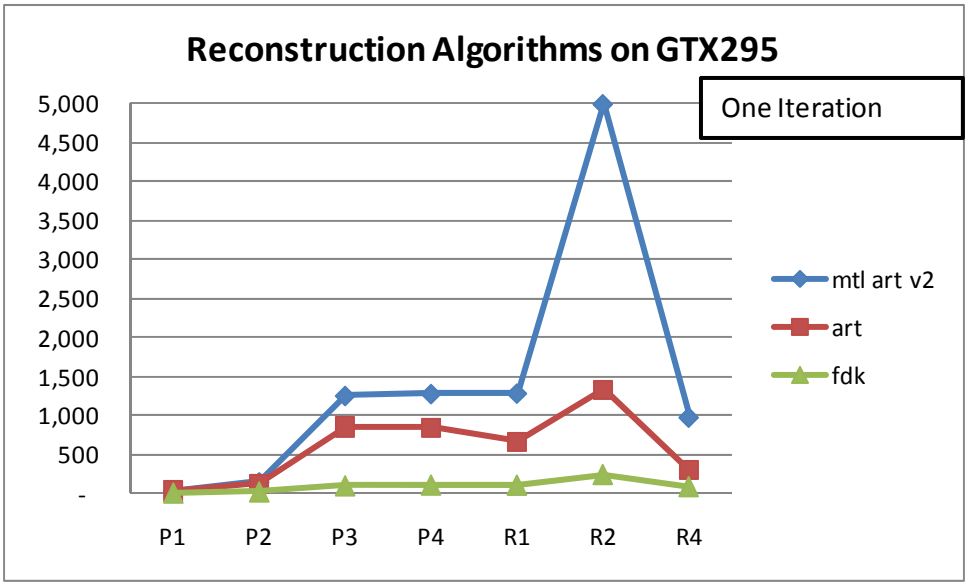


Figure 37 – Reconstruction Algorithms on GTX295

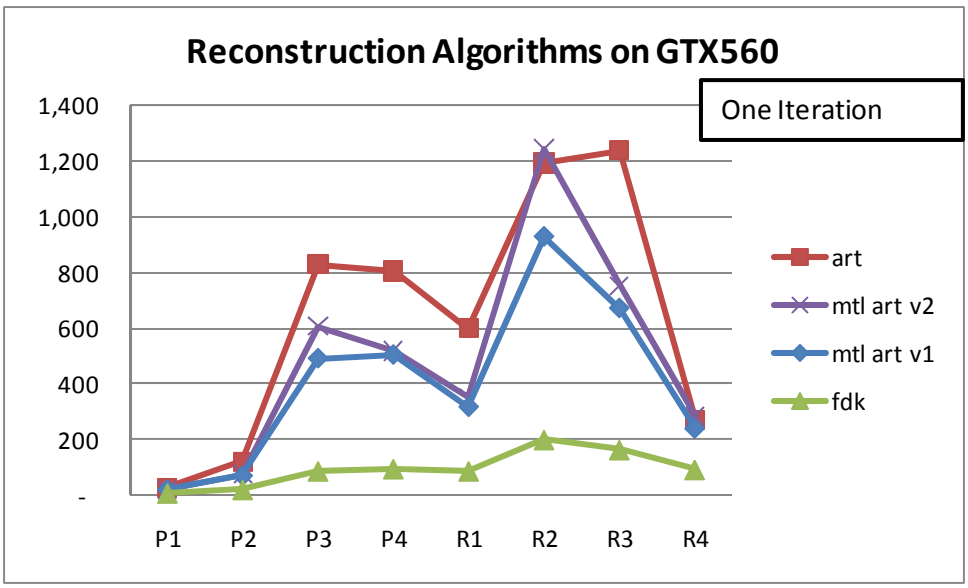


Figure 38 – Reconstruction Algorithms on GTX560

Running algorithms on CPU or GTX 295, the runtime result is as expected. FDK is fastest and Material ART is slower than ART. On the GTX 560 Ti, there were unexpected results. FDK still is fastest, but ART is slower than Material ART.

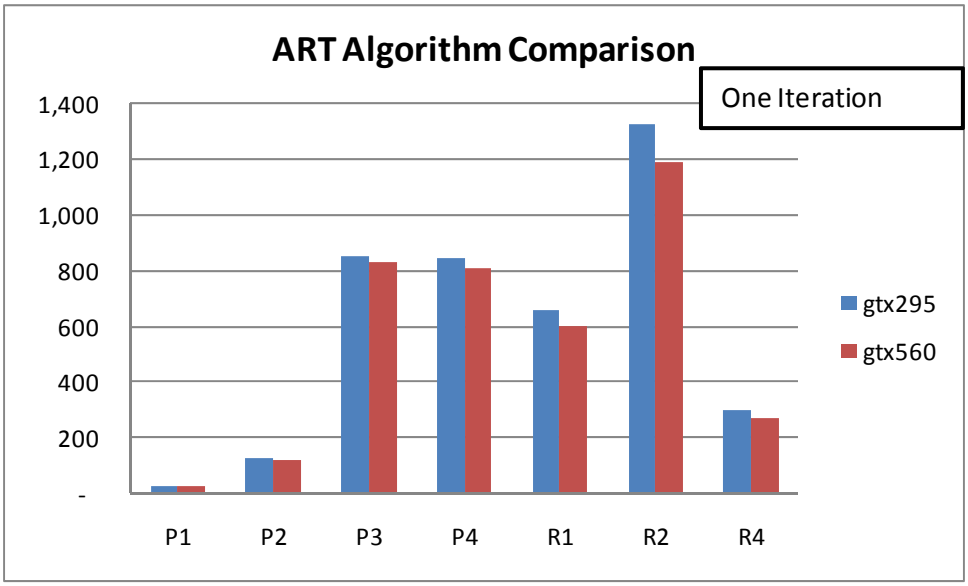


Figure 39 – ART Algorithm Comparison

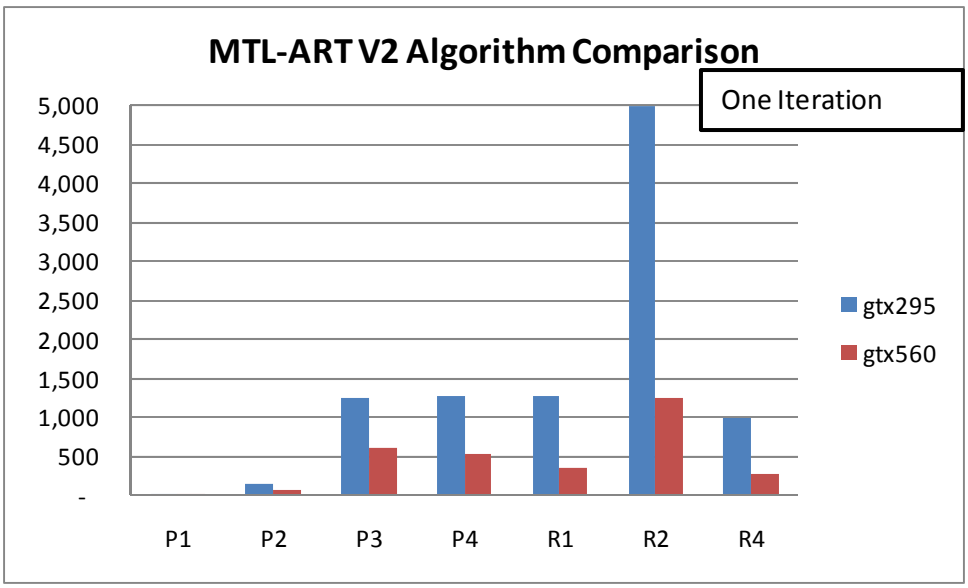


Figure 40 – Material ART V2 Algorithm Comparison

The expectation that Material ART runs slower holds true for the older CUDA cards: the G210 and the GTX 295 (for the limited data sets the programs can run on). However, this trend breaks for the GTX 560. This happens for several reasons.

7.3.3 ART and Material ART

7.3.3.1 Memory Swapping

The CUDA accelerations were developed on the 256^3 reconstruction sizes, or 64 MB, for the weaker CUDA cards such as the G210 and the GTX 295. For this problem size, the swapping took an insignificant amount of time and the actual calculation dominated the runtime; hence the effort to reduce the Kernel execution time and the reason texture memory is useful. The plan was to use the scalability inherent to CUDA to scale the program to larger reconstruction sizes. It was expected that the characteristics of the runtime breakdown remain the same; however, this is not the case.

For the GTX 560, it became apparent that the Material ART is faster than ART. When this happened, it was discovered that the swap time had dramatically increased between the 256^3 reconstruction size and the 512^3 reconstruction size. Because ART swaps and Material ART does not swap the reconstruction volume, Material ART runtime becomes faster than the ART. This will be further examined in detail.

7.3.3.2 Caching

Another irregularity is the relatively large runtime increase between the ART and Material ART V2 for the GTX 285, as compared to the actual decrease in runtime for the GTX 560. This can be explained by the new caching features of the GTX 560.

Material ART is another leap in complexity as compared to ART, and Material ART V2 uses more calculation as compared to Material ART V1. A number of parameters for Material ART are stored in global memory, and the CUDA implementation of ART uses so many registers that it spills into local memory. For the GTX 285, this means performance hits when running Material ART V2. But because the GTX 560 has an actual cache hierarchy, these performance penalties are blunted. This is why the GTX 560 experience a nominal performance hit for the Material ART V2.

7.3.4 GPU Time Breakdown

Due to the previously mentioned unexpected runtime results, the CUDA accelerations were run through the CUDA Visual Profiler provided in the CUDA toolkit. This was done to analyze the details of the GPU runtimes.

Table 11 – GPU Time Breakdown

GPU time (seconds)						
Dataset	**Computation**			**Memory Operations**		
FDK	**G210**	**GTX 295**	**GTX 560**	**G210**	**GTX 295**	**GTX 560**
P1	11.5	0.8	0.5	0.2	0.2	0.2
P2	59.0	4.2	2.3	0.6	0.6	0.6
P3		25.7	14.0		2.7	2.7
P4		25.7	14.0		2.9	2.7
ART	**G210**	**GTX 295**	**GTX 560**	**G210**	**GTX 295**	**GTX 560**
P1	18.2	1.4	0.8	7.1	6.3	5.9
P2	98.7	7.3	3.9	49.9	49.4	48.7
P3		52.2	27.5		386.9	386.4
P4		52.2	27.5		386.9	386.4
Material ART V1	**G210**	**GTX 295**	**GTX 560**	**G210**	**GTX 295**	**GTX 560**
P1	67.5	7.7	3.3	0.1	0.1	0.1
P2	454.0	56.6	23.2	0.3	0.3	0.3
P3			259.8			1.6
P4			259.6			1.6
Material ART V2	**G210**	**GTX 295**	**GTX 560**	**G210**	**GTX 295**	**GTX 560**
P1	118.2	14.4	5.8	0.1	0.1	0.1
P2	848.0	110.5	41.7	0.3	0.3	0.2
P3		1109.2	473.1		1.2	1.2
P4		1124.8	473.4		1.2	1.2

Table 12 – Percentage Breakdown of GPU Time

GPU Time (%)						
Dataset	**G210**		**GTX 295**		**GTX 560**	
FDK	**computation**	**memory**	**computation**	**memory**	**computation**	**memory**
P1	98.5	1.4	84.0	16.0	76.6	23.3
P2	98.9	1.1	87.1	12.9	79.1	20.9
P3			90.5	9.4	83.6	16.4
P4			89.9	10.1	83.7	16.2
ART	**computation**	**memory**	**computation**	**memory**	**computation**	**memory**
P1	72.0	28.0	17.7	82.3	11.3	88.7
P2	66.4	33.6	12.8	87.1	7.4	92.6
P3			11.9	88.1	6.6	93.3
P4			11.9	88.1	6.6	93.3
Material ART V1	**computation**	**memory**	**computation**	**memory**	**computation**	**memory**
P1	99.9	0.1	99.1	0.9	98.4	1.6
P2	99.9	0.1	99.5	0.5	98.9	1.1
P3					99.4	0.6
P4					99.4	0.6
Material ART V2	**computation**	**memory**	**computation**	**memory**	**computation**	**memory**
P1	99.9	0.1	99.4	0.6	99.1	0.9
P2	100.0	0.0	99.8	0.2	99.5	0.5
P3			99.9	0.1	99.7	0.2
P4			99.9	0.1	99.7	0.2

Table 11 and Table 12 show the breakdown of all the computer generated cases. Only the computer generated cases were profiled because we have direct control of their dimensions; this is not the case for the real data.

In Table 11 and Table 12, the GPU time refers to the amount of the runtime spent on the GPU, either running kernels or transferring data between the host and the device. Because these are the most significant components of the GPU time, the GPU time was categorized between the Computation time and the Memory Operations time. Computation time refers to the time spent on the reconstruction kernels

(backprojection, forward projection, and merged), as well as the time spent on the FFT and filtering. In contrast, the Memory Operations time refers to the time spent on transferring data between the host and device. The Memory Operations time also includes the time spent on memory set operations.

7.3.4.1 ART vs. Material ART

Table 11 and Table 12 confirm that, for the GTX 295 and GTX 560, ART is dominated by memory operations and Material ART is dominated by calculation. The implication is that the current texture memory usage is a liability rather than an improvement. A better solution would have been to forgo the use of texture memory, or to use a CUDA card with more memory so host system to device swapping is not necessary.

7.3.4.2 ART Runtime across Graphics Cards

The runtime characteristics of ART change across different CUDA cards. Table 12 shows that for the P1 and P2 data sets, the percentage breakdown for the G210 conflicts with the breakdown for the GTX 295 and GTX 560 Ti. For the G210 case the computation time dominated over the memory transfer time. For the faster cards, the memory operations dominated in contrast.

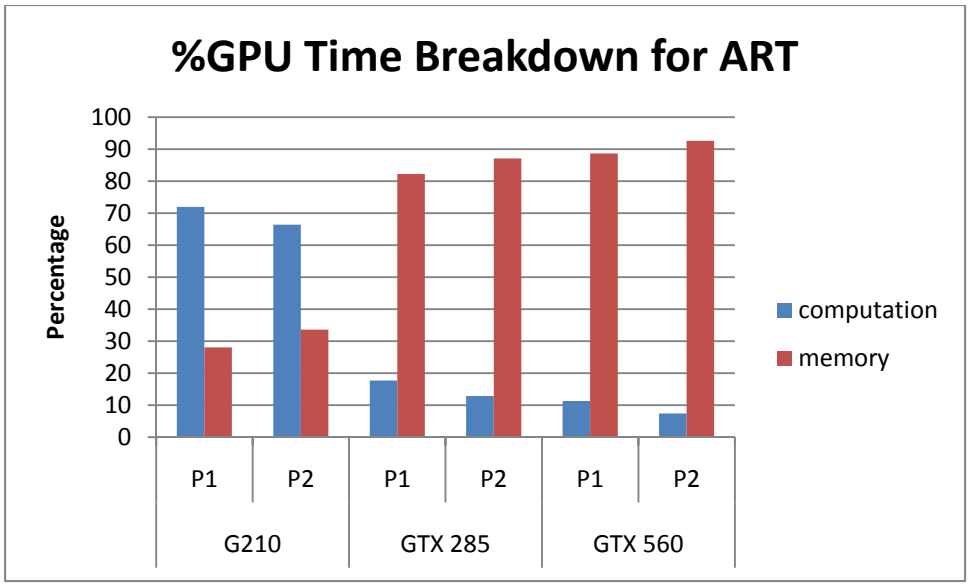


Figure 41 – GPU Time Percentages for ART on Data Sets P1 and P2

Figure 41 demonstrates the change of the runtime characteristics of the problem. Because the implementations were developed on the GT 330M (which is closer to the G210 rather than the GTX 295), the in-development profile shows emphasis on the kernel computation as opposed to the memory transfer and it was desired to use the texture memory; the added memory transfer was less important. However, the above graph showed that this changed for the more powerful GPUs: the computation time was reduced and the memory operation time remained the same, but took up a greater share of the runtime. The lesson here is that the significance between the calculations and memory transfer can change with the CUDA card.

7.3.4.3 Further Analysis of GPU Time Breakdown

As expected, from the results of Table 11, computation time decreases with increasingly powerful cards, while the memory operations time remains constant across different

cards. This is because the memory transfers have to go across the PCI-Express bus and this bottleneck does not change.

Table 11 shows a clear speedup between the computation times of the GTX 295 and the GTX 560. In fact, there is approximately a 2x speedup between the GTX 295 and the GTX 560 computation times. Because the GTX 295 has 240 cores (per GPU) and the GTX 560 has 384 cores, the increase in the number of cores is insufficient to explain the speedup – the GTX 560 has only 60% more cores than GTX 295. The new formal cache hierarchy is the feature in the new architecture that explains the speedup. The CUDA version should be upgraded for the programs so the recent Compute Visual Profiler can analyze the cache benefit.

In the results and analysis, both the reconstruction images and runtimes were collected and analyzed. The image quality was confirmed and the runtimes produced interesting results. The next chapter concludes the thesis.

Chapter 8: Conclusions and Future Directions

8.1 Conclusions

The image comparisons show that CUDA accelerations produce images that are as good as or better quality than the CPU implementation. The FDK and ART implementation has proven to be solid, and the CUDA accelerations of Material ART have shown that more testing and development are needed.

All CUDA accelerations have also demonstrated speedup. Texture memory usage for ART is shown to be flawed, and solutions have been proposed to fix it. The speedup provided for Material ART will enable and make further research practical.

8.2 Future Directions

Near the end of the project, development and debugging tools were released for CUDA. Future work should take advantage of these tools to reduce the difficulty of CUDA programming. This would require upgrading the operating systems from Windows XP to Windows 7 and upgrading the CUDA version of project. NVIDIA's Compute Visual Profiler should be used to properly measure the effect of the cache on runtime. NVIDIA Parallel Nsight should be used to debug the kernels running on the GPU and aid further CUDA development work.

Material ART is only part of the strategy of improving reconstruction quality. The next step is to implement scatter correction, or to correct for the scattering of X-rays. Material ART is still a very immature algorithm, and needs to be developed further. Proper calibrations need to be taken of the X-ray machines (the 9300 and the 9500).

The memory swapping in ART has been discussed in depth, and either of the proposed solutions should be implemented:

- Roll-back texture memory usage and maintain static memory spaces again
- Use a CUDA card with sufficient global memory to maintain both the CUDA array and reconstruction volume at the same time

The second solution would be ideal, as it would have the benefits of texture memory without the current cost of memory swapping.

This thesis is part of a larger goal in the medical imaging community to eventually reach real-time CT reconstruction. Commodity GPUs have shown performance speedups not seen outside of supercomputers, cluster computing, or FPGAs. Yet, GPUs are far cheaper and more economical than both supercomputers and cluster computing and far more flexible and easier to develop than FPGAs. Because the reconstruction algorithms are highly parallel, they make a good acceleration target for NVIDIA CUDA. The CUDA accelerations have demonstrated speedups up to 297x, and the image output confirmed increased image quality. The end result is that the CUDA accelerations of the

reconstruction algorithms are enabling technologies; what was previously impractical is now feasible and it allows further research and faster, better image reconstructions.

BIBLIOGRAPHY

[1] H. Scherl et al., "Fast GPU-based CT reconstruction using the common unified device architecture (CUDA)," in *Proc. Nuclear Science Symp. and Medical Imaging Conf. (NSS/MIC'07)*, Oct. 2007, pp. 4464-4466.

[2] B. Keck et al., "GPU-accelerated SART reconstruction using the CUDA programming environment," Proc. SPIE 7258, 72582B (2009), DOI:10.1117/12.811559

[3] C. H. Yan et al., "Reconstruction algorithm for polychromatic CT imaging: application to beam hardening correction," *Medical Imaging, IEEE Transactions on* , vol. 19, no. 1, pp. 1-11, Jan. 2000. doi: 10.1109/42.832955 Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=832955&isnumber=18026 [May 22, 2011]

[4] L. A. Feldkamp et al., "Practical cone-beam algorithm," J. Opt. Soc. Am. A 1, 612-619 (1984). Available: http://www.opticsinfobase.org/abstract.cfm?URI=josaa-1-6-612 [May 22, 2011]

[5] A. C. Kak and Malcolm Slaney, *Principles of Computerized Tomographic Imaging,* IEEE Press, 1988. Available: http://www.slaney.org/pct/pct-toc.html [May 22, 2011]

[6] What is CUDA? Internet: http://www.nvidia.com/object/what_is_CUDA_new.html. [May 22, 2011]

[7] "CS 9300 System* for Point-of-Care CT Imaging." Internet: Internet: http://carestream.com/cs-9300-system.html [May 13, 2011]

[8] "The KODAK 9500 Cone Beam 3D System." Internet: Internet: http://eamer.carestreamdental.com/en/digital-imaging/3d-imaging/Kodak%209500.aspx [May 13, 2011]

[9] C. H. Kau et al., "Current Products and Practice Three-dimensional cone beam computerized tomography in orthodontics". *Journal of Orthodontics*, vol. 32, 281–292, 2005h.

[10] J. Montrym and H. Moreton. 2005. The GeForce 6800. IEEE Micro 25, 2 (March 2005), 41-51. DOI=10.1109/MM.2005.37 Available: http://dx.doi.org/10.1109/MM.2005.37 [May 22, 2011]

[11] E. Lindholm et al., 2008. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro* vol. 28, no. 2 (March 2008), pp. 39-55. DOI=10.1109/MM.2008.31 Available: http://dx.doi.org/10.1109/MM.2008.31 [May 22, 2011]

[12] Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Fermi. 17 Nov. 2010 Internet: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf [May 22, 2011]

[13] "NVIDIA CUDA C Programming Guide." Internet: http://developer.download.nvidia.com/compute/CUDA/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf, March 25, 2011 [May 13, 2011]

[14] H. Scherl et al., "Implementation of the FDK algorithm for cone-beam CT on the Cell Broadband Engine Architecture," in *Proceedings of SPIE Medical Imaging 2007: Physics of Medical Imaging*, J. Hsieh and M. Flynn, Eds., vol. 6510, San Diego, February 2007, p. 651058.

[15] M. Kachelriess et al., "Hyperfast perspective cone-beam backprojection," in *IEEE Nuclear Science Symposium and Medical Imaging Conference*, San Diego, 2006, m01-7.

[16] M. Trepanier and I. Goddard, "Adjunct processors in embedded medical imaging systems," *SPIE Medical Imaging Proc.*, vol. 4681, pp. 416–424, 2002.

[17] I. Goddard and M. Trepanier, "High–speed cone–beam reconstruction: An embedded systems approach," *SPIE Medical Imaging Proc.*,vol. 4681, pp. 483–491, 2002.

[18] I. Goddard and M. Trepanier, "The role of FPGA–based processing in medical imaging," *VMEbus Systems*, Apr. 2003.

[19] K. Mueller et al., "Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?" in *SPIE Electronic Imaging Conference*, San Diego, 2007, (Keynote, Computational Imaging V).

[20] M. Churchill, "Hardware-accelerated cone-beam reconstruction on a mobile C-arm," in *Proceedings of SPIE*, J. Hsieh and M. Flynn, Eds., vol. 6510, 2007.

[21] B. Wang et al. , "Accelerated cone beam CT reconstruction based on OpenCL," *Image Analysis and Signal Processing (IASP), 2010 International Conference on* , vol., no., pp.291-295, 9-11 April 2010. doi: 10.1109/IASP.2010.5476110.
Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5476110&isnumber=5476039 [May 22, 2011]

[22] G. L. Zeng et al., "A study of reconstruction artifacts in cone beam tomography using filtered backprojection and iterative EM algorithms," *Nuclear Science, IEEE Transactions on* , vol. 37, no. 2, pp. 759-767, Apr 1990. doi: 10.1109/23.106711.
Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=106711&isnumber=3253 [May 22, 2011]

[23] "CUDA C BEST PRACTICES GUIDE." Internet: http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf, March 2011 [May 23, 2011]

Appendix A

Reconstructed Images from the Runs of Reconstruction Algorithms on CPU & GTX560 Graphics Card

This appendix includes only a representative sample of images that may be of interest and provides more detailed insight

List of Figures

Figure A1: R1, FDK, Slice 322, CPU 84

Figure A2: R1, FDK, Slice 322, GTX560 84

Figure A3: R1, ART, Slice 322, CPU 84

Figure A4: R1, ART, Slice 322, GTX560 84

Figure A5: R1, MTL-ART, Slice 322, CPU 85

Figure A6: R1, MTL-ART V1, Slice 322, GTX560 85

Figure A7: R1, MTL-ART, Slice 322, CPU 85

Figure A8: R1, MTL-ART V2, Slice 322, GTX560 85

Figure A9: R2, FDK, Slice 135, CPU 86

Figure A10: R2, FDK, Slice 135, GTX560 86

Figure A11: R2, ART, Slice 135, CPU 86

Figure A12: R2, ART, Slice 135, GTX560 86

Figure A13: R2, MTL-ART, Slice 135, CPU 87

Figure A14: R2, MTL-ART V1, Slice 135, GTX560 87

Figure A15: R2, MTL-ART, Slice 135, CPU 87

Figure A16: R2, MTL-ART V2, Slice 135, GTX560 87

Figure A17: R4, FDK, Slice 38, CPU 88

Figure A18: R4, FDK, Slice 38, GTX560 88

Figure A19: R4, ART, Slice 38, CPU 88

Figure A20: R4, ART, Slice 38, GTX560 88

Figure A21: R4, MTL-ART, Slice 38, CPU 89

Figure A22: R4, MTL-ART V1, Slice 38, GTX560 89

Figure A23: R4, MTL-ART, Slice 38, CPU 89

Figure A24: R4, MTL-ART V2, Slice 38, GTX560 89

Figure A1: R1, FDK, Slice 322, CPU

Figure A2: R1, FDK, Slice 322, GTX560

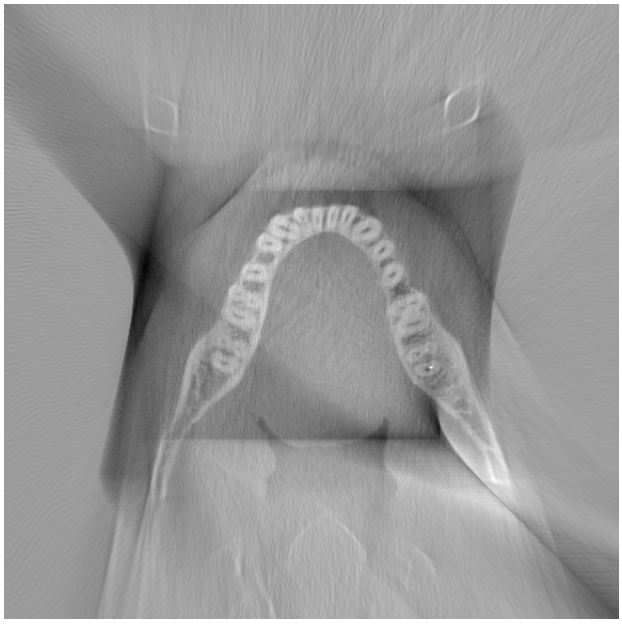


Figure A3: R1, ART, Slice 322, CPU

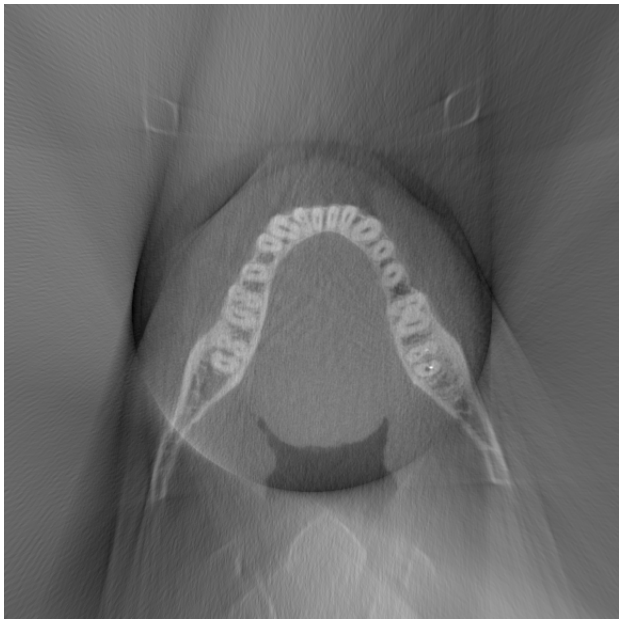


Figure A4: R1, ART, Slice 322, GTX560

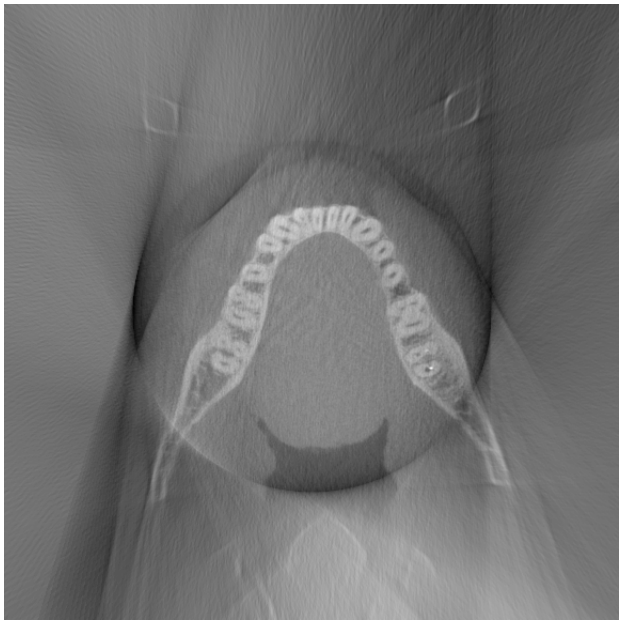


Figure A5: R1, MTL-ART, Slice 322, CPU

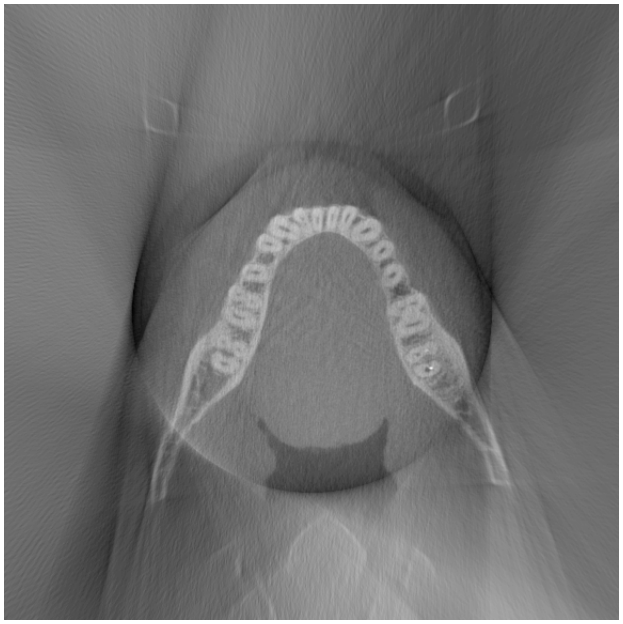


Figure A6: R1, MTL-ART V1, Slice 322, GTX560

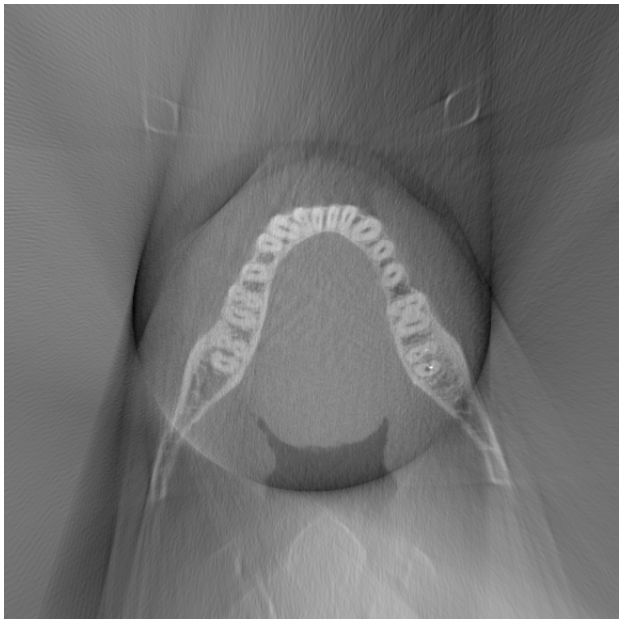


Figure A7: R1, MTL-ART, Slice 322, CPU

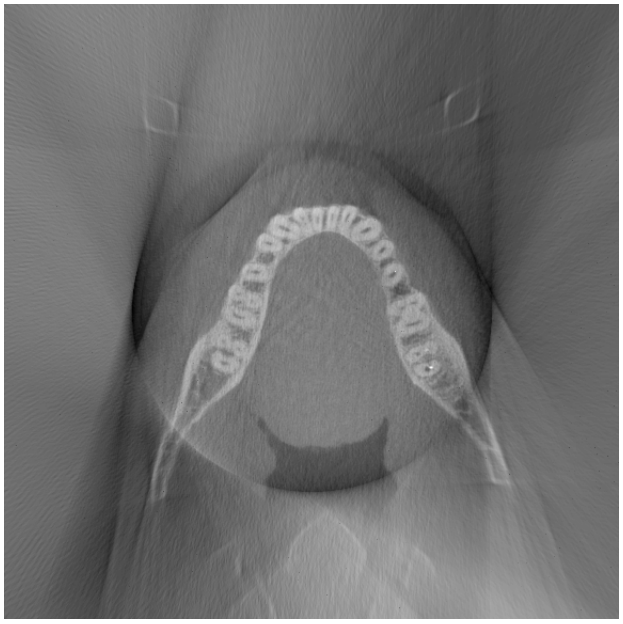


Figure A8: R1, MTL-ART V2, Slice 322, GTX560

Figure A9: R2, FDK, Slice 135, CPU

Figure A10: R2, FDK, Slice 135, GTX560

Figure A11: R2, ART, Slice 135, CPU

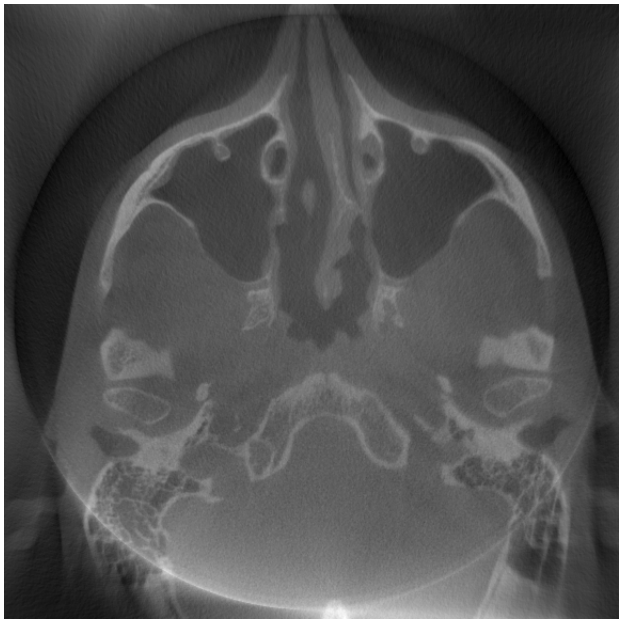


Figure A12: R2, ART, Slice 135, GTX560

Figure A13: R2, MTL-ART, Slice 135, CPU

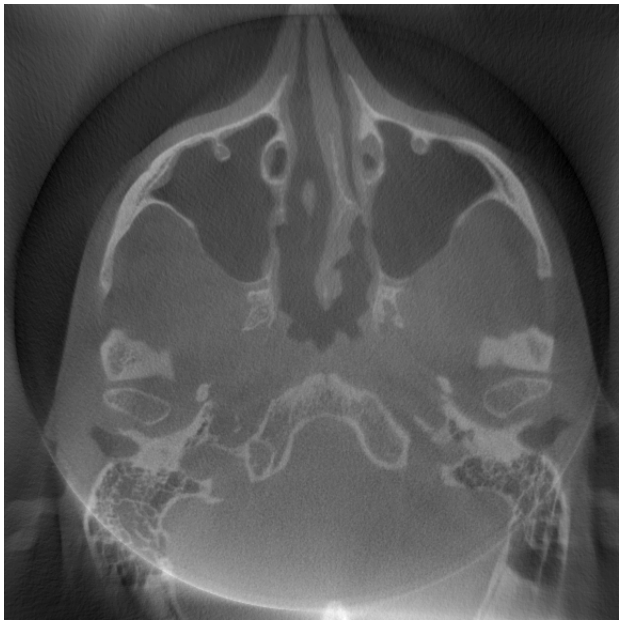


Figure A14: R2, MTL-ART V1, Slice 135, GTX560

Figure A15: R2, MTL-ART, Slice 135, CPU

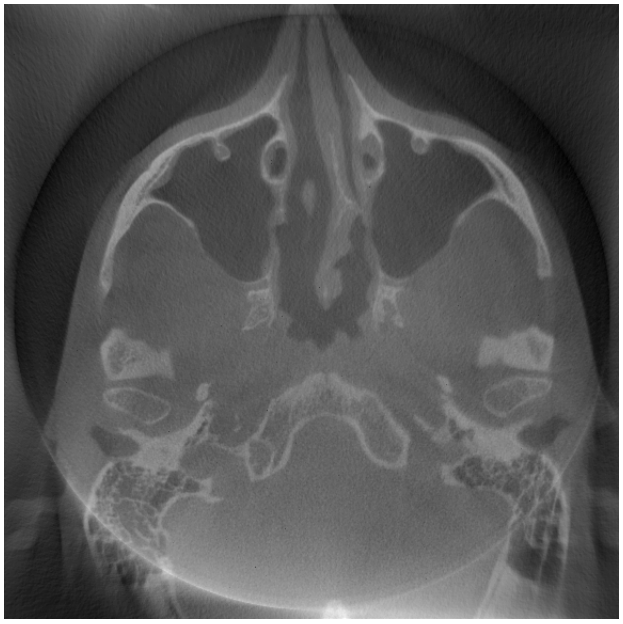


Figure A16: R2, MTL-ART V2, Slice 135, GTX560

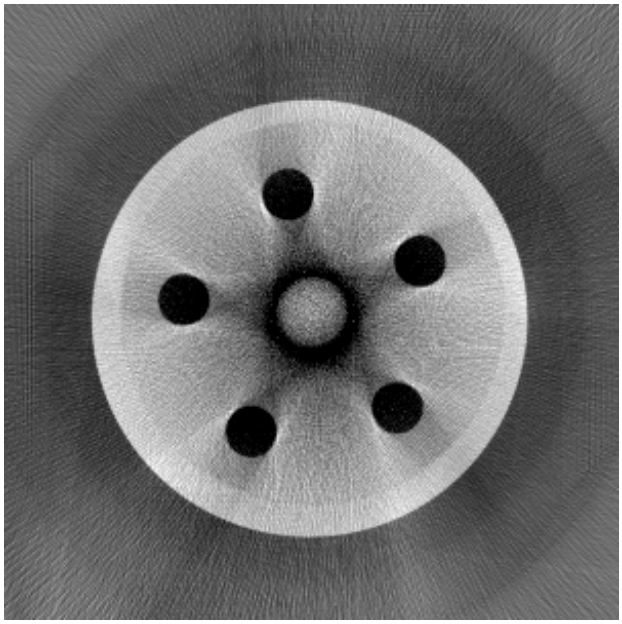


Figure A17: R4, FDK, Slice 38, CPU

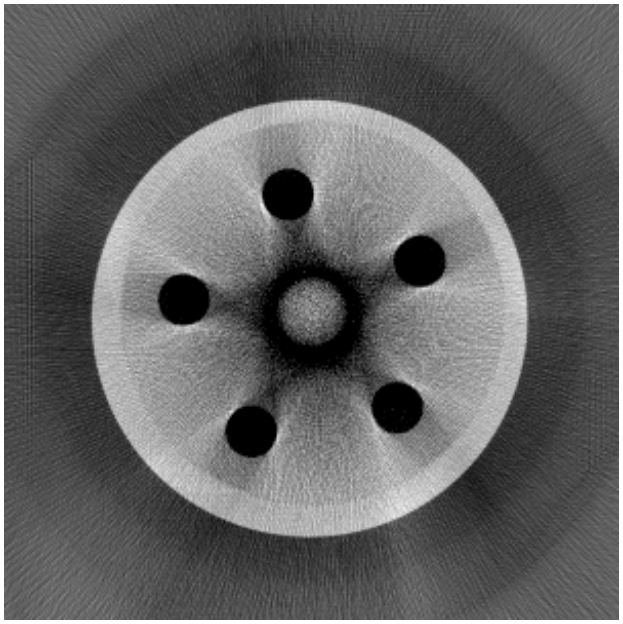


Figure A18: R4, FDK, Slice 38, GTX560

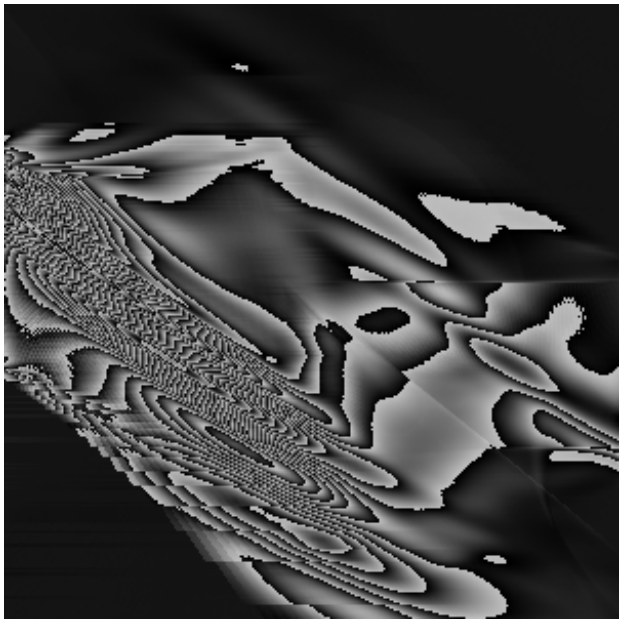


Figure A19: R4, ART, Slice 38, CPU

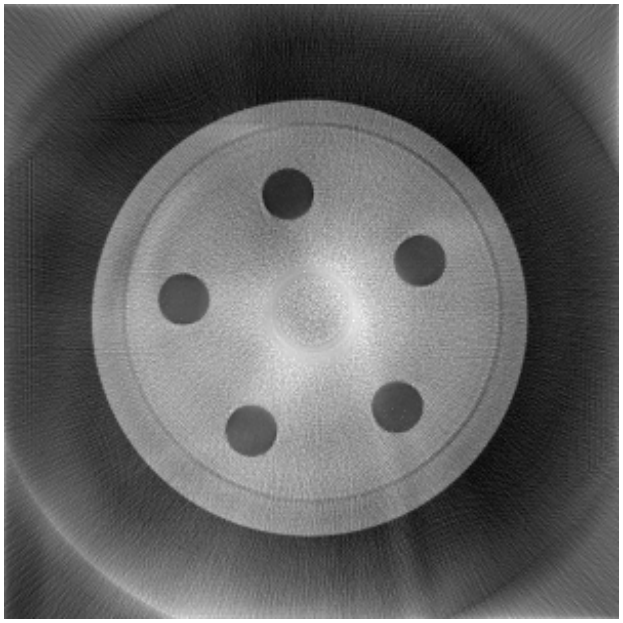


Figure A20: R4, ART, Slice 38, GTX560

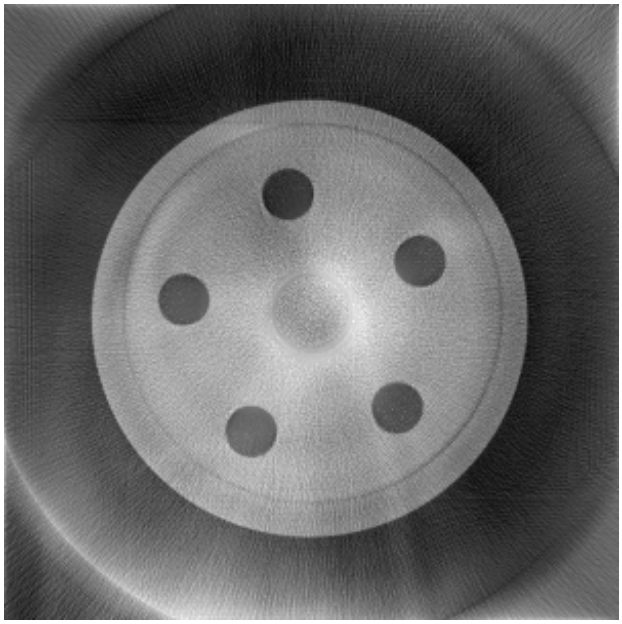


Figure A21: R4, MTL-ART, Slice 38, CPU

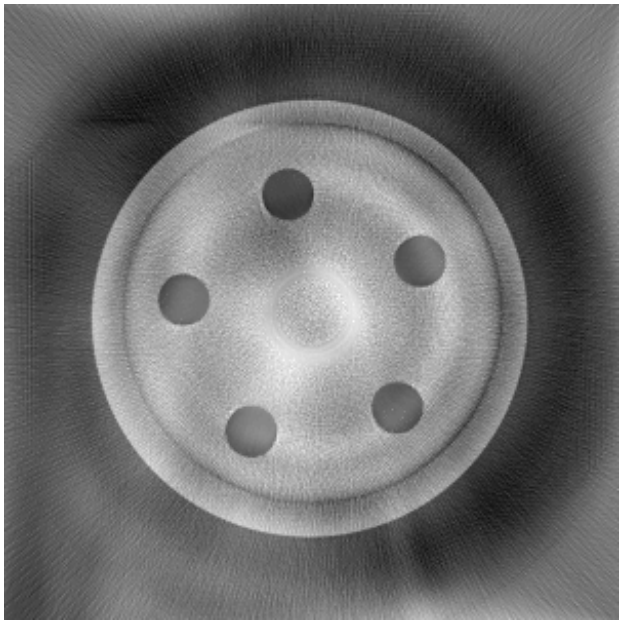


Figure A22: R4, MTL-ART V1, Slice 38, GTX560

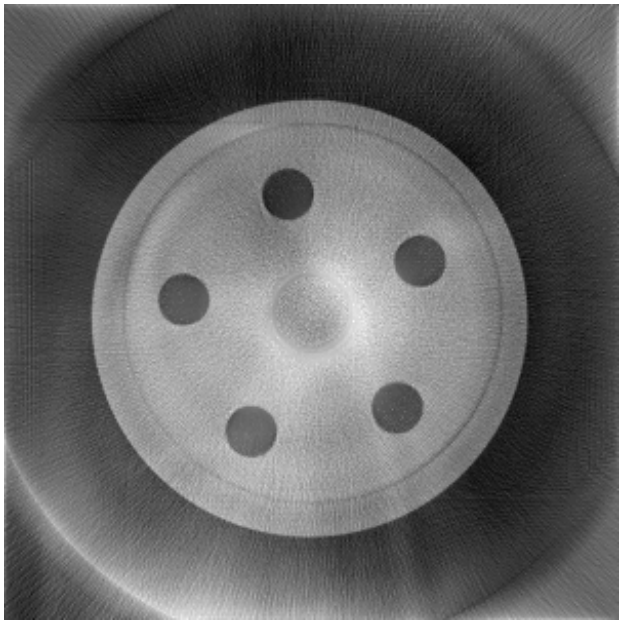


Figure A23: R4, MTL-ART, Slice 38, CPU

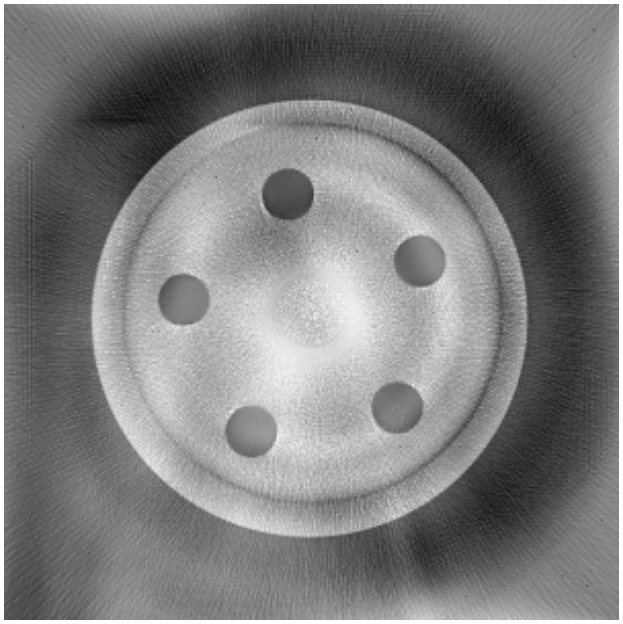


Figure A24: R4, MTL-ART V2, Slice 38, GTX560

Appendix B

Reconstructed Images from the 20 Iteration Runs of Reconstruction Algorithms on GTX560 Graphics Card

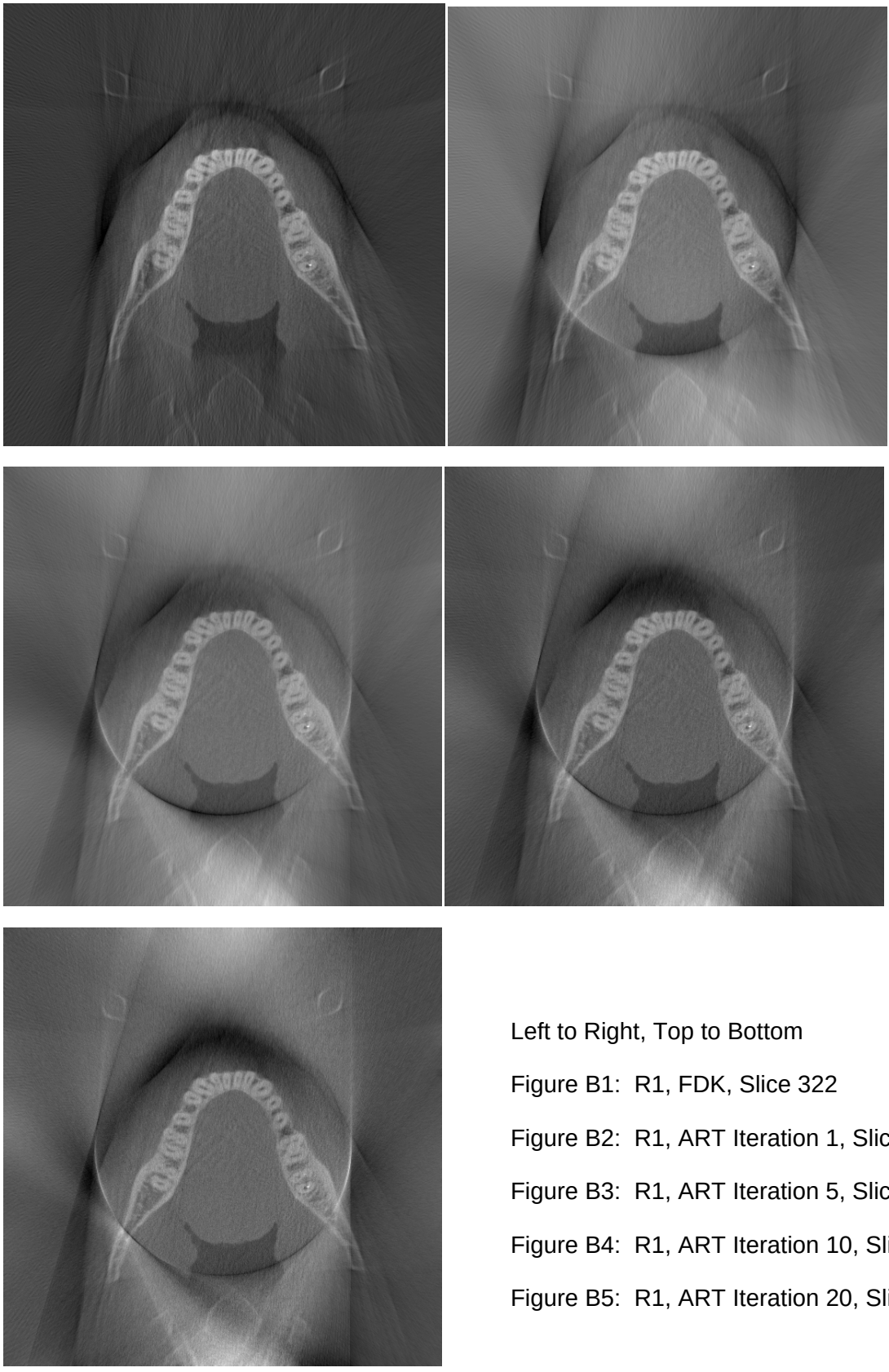
This appendix includes only a representative sample of images that may be of interest and provides more detailed insight

List of Figures

Figure B1: R1, FDK, Slice 322 93
Figure B2: R1, ART Iteration 1, Slice 322 93
Figure B3: R1, ART Iteration 5, Slice 322 93
Figure B4: R1, ART Iteration 10, Slice 322 93
Figure B5: R1, ART Iteration 20, Slice 322 93
Figure B6: R1, FDK, Slice 322 94
Figure B7: R1, MART1 Iteration 1, Slice 322 94
Figure B8: R1, MART1 Iteration 5, Slice 322 94
Figure B9: R1, MART1 Iteration 10, Slice 322 94
Figure B10: R1, MART1 Iteration 20, Slice 322 94
Figure B11: R1, FDK, Slice 322 95
Figure B12: R1, MART2 Iteration 1, Slice 322 95
Figure B13: R1, MART2 Iteration 5, Slice 322 95
Figure B14: R1, MART2 Iteration 10, Slice 322 95
Figure B15: R1, MART2 Iteration 20, Slice 322 95
Figure B16: R1, FDK, Slice 364 96
Figure B17: R1, ART Iteration 1, Slice 364 96
Figure B18: R1, ART Iteration 5, Slice 364 96
Figure B19: R1, ART Iteration 10, Slice 364 96
Figure B20: R1, ART Iteration 20, Slice 364 96
Figure B21: R1, FDK, Slice 364 97

Figure B22: R1, MART1 Iteration 1, Slice 364 97
Figure B23: R1, MART1 Iteration 5, Slice 364 97
Figure B24: R1, MART1 Iteration 10, Slice 364 97
Figure B25: R1, MART1 Iteration 20, Slice 364 97
Figure B26: R1, FDK, Slice 364 98
Figure B27: R1, MART2 Iteration 1, Slice 364 98
Figure B28: R1, MART2 Iteration 5, Slice 364 98
Figure B29: R1, MART2 Iteration 10, Slice 364 98
Figure B30: R1, MART2 Iteration 20, Slice 364 98
Figure B31: R2, FDK, Slice 135 99
Figure B32: R2, ART Iteration 1, Slice 135 99
Figure B33: R2, ART Iteration 5, Slice 135 99
Figure B34: R2, ART Iteration 10, Slice 135 99
Figure B35: R2, ART Iteration 20, Slice 135 99
Figure B36: R2, FDK, Slice 190 100
Figure B37: R2, ART Iteration 1, Slice 190 100
Figure B38: R2, ART Iteration 5, Slice 190 100
Figure B39: R2, ART Iteration 10, Slice 190 100
Figure B40: R2, ART Iteration 20, Slice 190 100
Figure B41: R3, FDK, Slice 443 101
Figure B42: R3, ART Iteration 1, Slice 443 101
Figure B43: R3, ART Iteration 5, Slice 443 101
Figure B44: R3, ART Iteration 10, Slice 443 101
Figure B45: R3, ART Iteration 20, Slice 443 101
Figure B46: R3, FDK, Slice 527 102
Figure B47: R3, ART Iteration 1, Slice 527 102
Figure B48: R3, ART Iteration 5, Slice 527 102
Figure B49: R3, ART Iteration 10, Slice 527 102
Figure B50: R3, ART Iteration 20, Slice 527 102
Figure B51: R4, FDK, Slice 38 103

Figure B52: R4, ART Iteration 1, Slice 38 103
Figure B53: R4, ART Iteration 5, Slice 38 103
Figure B54: R4, ART Iteration 10, Slice 38 103
Figure B55: R4, ART Iteration 20, Slice 38 103
Figure B56: R4, FDK, Slice 38 104
Figure B57: R4, MART1 Iteration 1, Slice 38 104
Figure B58: R4, MART1 Iteration 5, Slice 38 104
Figure B59: R4, MART1 Iteration 10, Slice 38 104
Figure B60: R4, MART1 Iteration 20, Slice 38 104
Figure B61: R4, FDK, Slice 38 105
Figure B62: R4, MART2 Iteration 1, Slice 38 105
Figure B63: R4, MART2 Iteration 5, Slice 38 105
Figure B64: R4, MART2 Iteration 10, Slice 38 105
Figure B65: R4, MART2 Iteration 20, Slice 38 105
Figure B66: R4, FDK, Slice 150 106
Figure B67: R4, ART Iteration 1, Slice 150 106
Figure B68: R4, ART Iteration 5, Slice 150 106
Figure B69: R4, ART Iteration 10, Slice 150 106
Figure B70: R4, ART Iteration 20, Slice 150 106
Figure B71: R4, FDK, Slice 150 107
Figure B72: R4, MART1 Iteration 1, Slice 150 107
Figure B73: R4, MART1 Iteration 5, Slice 150 107
Figure B74: R4, MART1 Iteration 10, Slice 150 107
Figure B75: R4, MART1 Iteration 20, Slice 150 107
Figure B76: R4, FDK, Slice 150 108
Figure B77: R4, MART2 Iteration 1, Slice 150 108
Figure B78: R4, MART2 Iteration 5, Slice 150 108
Figure B79: R4, MART2 Iteration 10, Slice 150 108
Figure B80: R4, MART2 Iteration 20, Slice 150 108



Left to Right, Top to Bottom

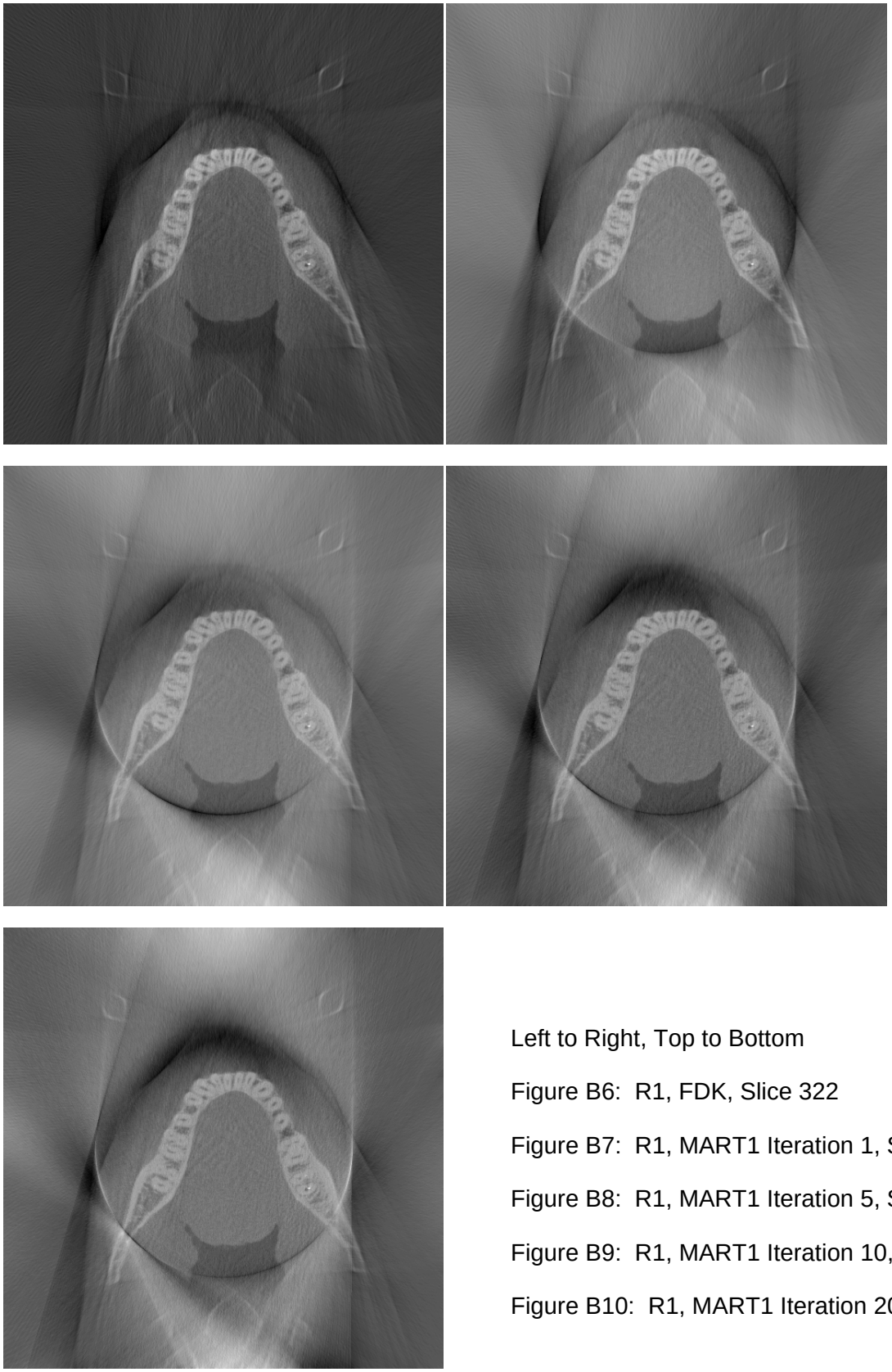
Figure B1: R1, FDK, Slice 322

Figure B2: R1, ART Iteration 1, Slice 322

Figure B3: R1, ART Iteration 5, Slice 322

Figure B4: R1, ART Iteration 10, Slice 322

Figure B5: R1, ART Iteration 20, Slice 322



Left to Right, Top to Bottom

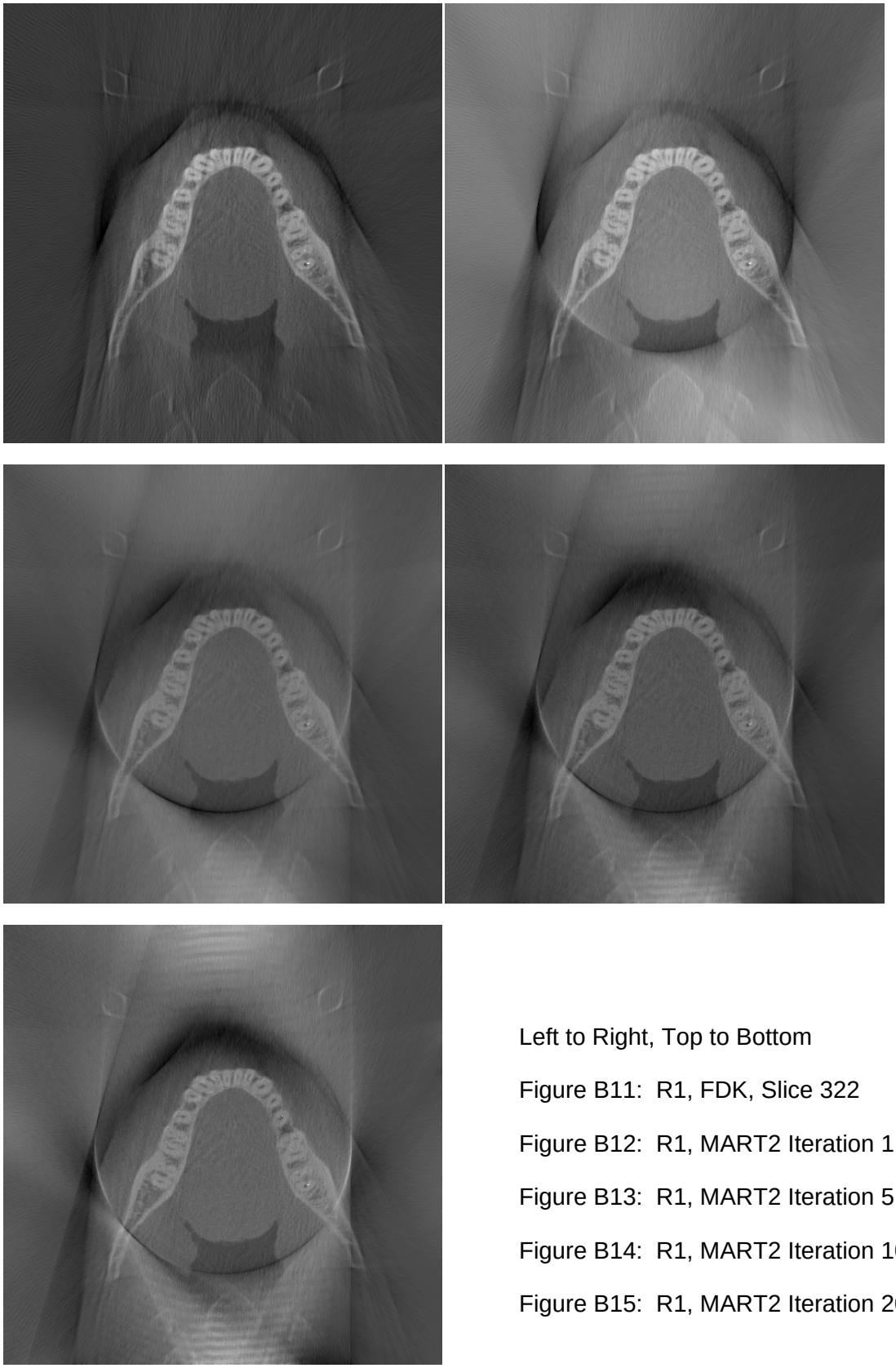
Figure B6: R1, FDK, Slice 322

Figure B7: R1, MART1 Iteration 1, Slice 322

Figure B8: R1, MART1 Iteration 5, Slice 322

Figure B9: R1, MART1 Iteration 10, Slice 322

Figure B10: R1, MART1 Iteration 20, Slice 322



Left to Right, Top to Bottom

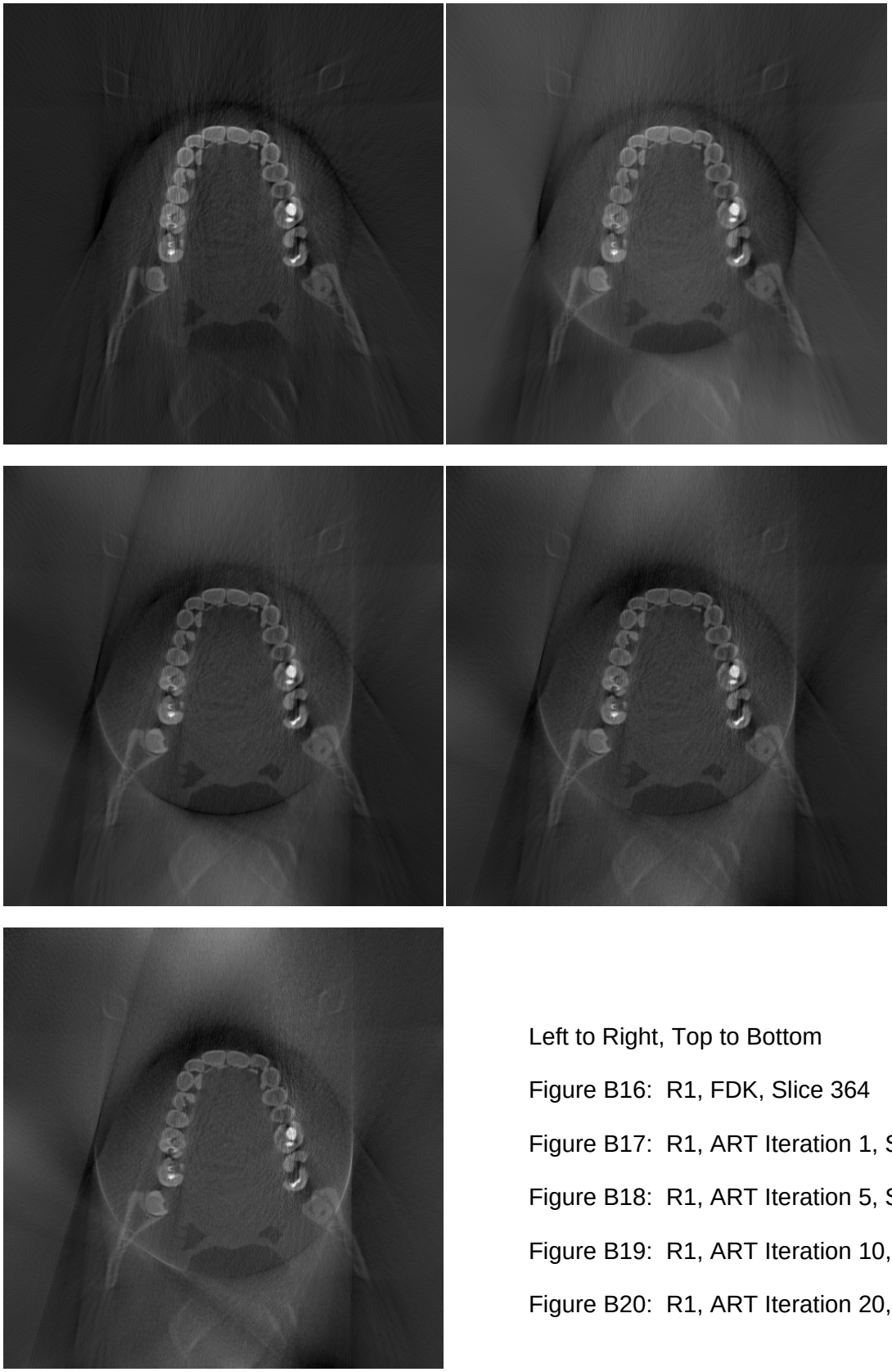
Figure B11: R1, FDK, Slice 322

Figure B12: R1, MART2 Iteration 1, Slice 322

Figure B13: R1, MART2 Iteration 5, Slice 322

Figure B14: R1, MART2 Iteration 10, Slice 322

Figure B15: R1, MART2 Iteration 20, Slice 322



Left to Right, Top to Bottom

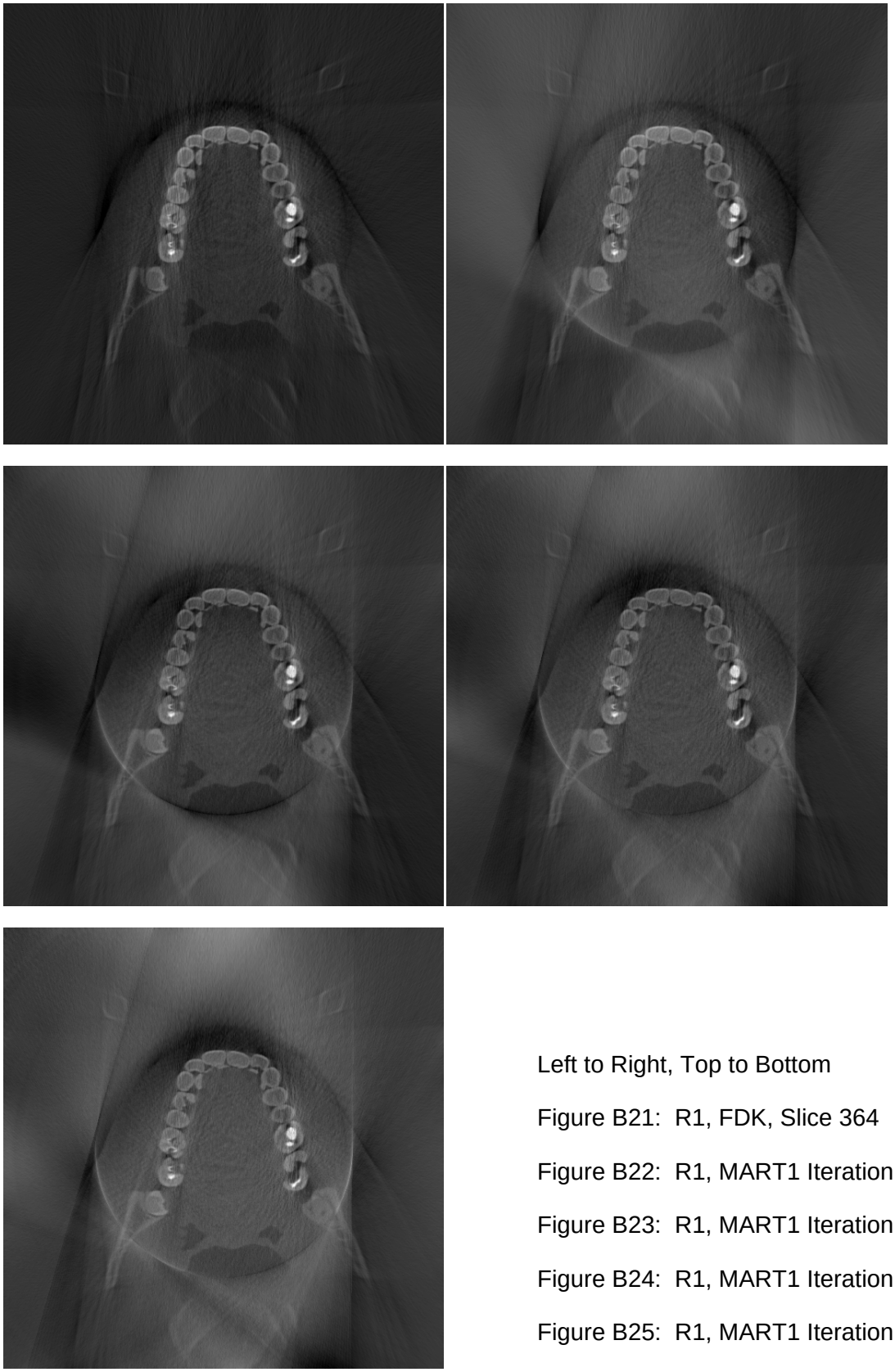
Figure B16: R1, FDK, Slice 364

Figure B17: R1, ART Iteration 1, Slice 364

Figure B18: R1, ART Iteration 5, Slice 364

Figure B19: R1, ART Iteration 10, Slice 364

Figure B20: R1, ART Iteration 20, Slice 364



Left to Right, Top to Bottom

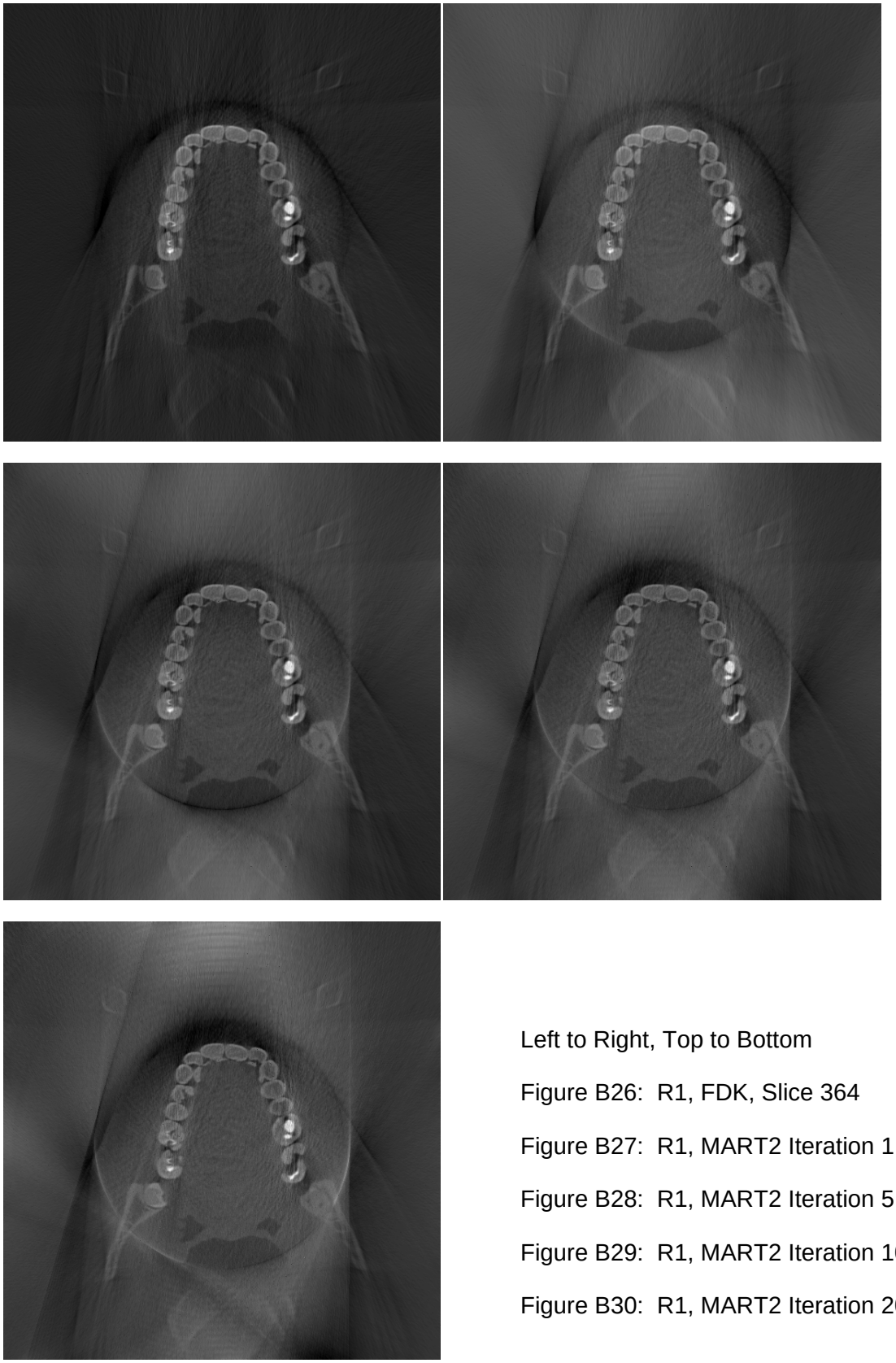
Figure B21: R1, FDK, Slice 364

Figure B22: R1, MART1 Iteration 1, Slice 364

Figure B23: R1, MART1 Iteration 5, Slice 364

Figure B24: R1, MART1 Iteration 10, Slice 364

Figure B25: R1, MART1 Iteration 20, Slice 364



Left to Right, Top to Bottom

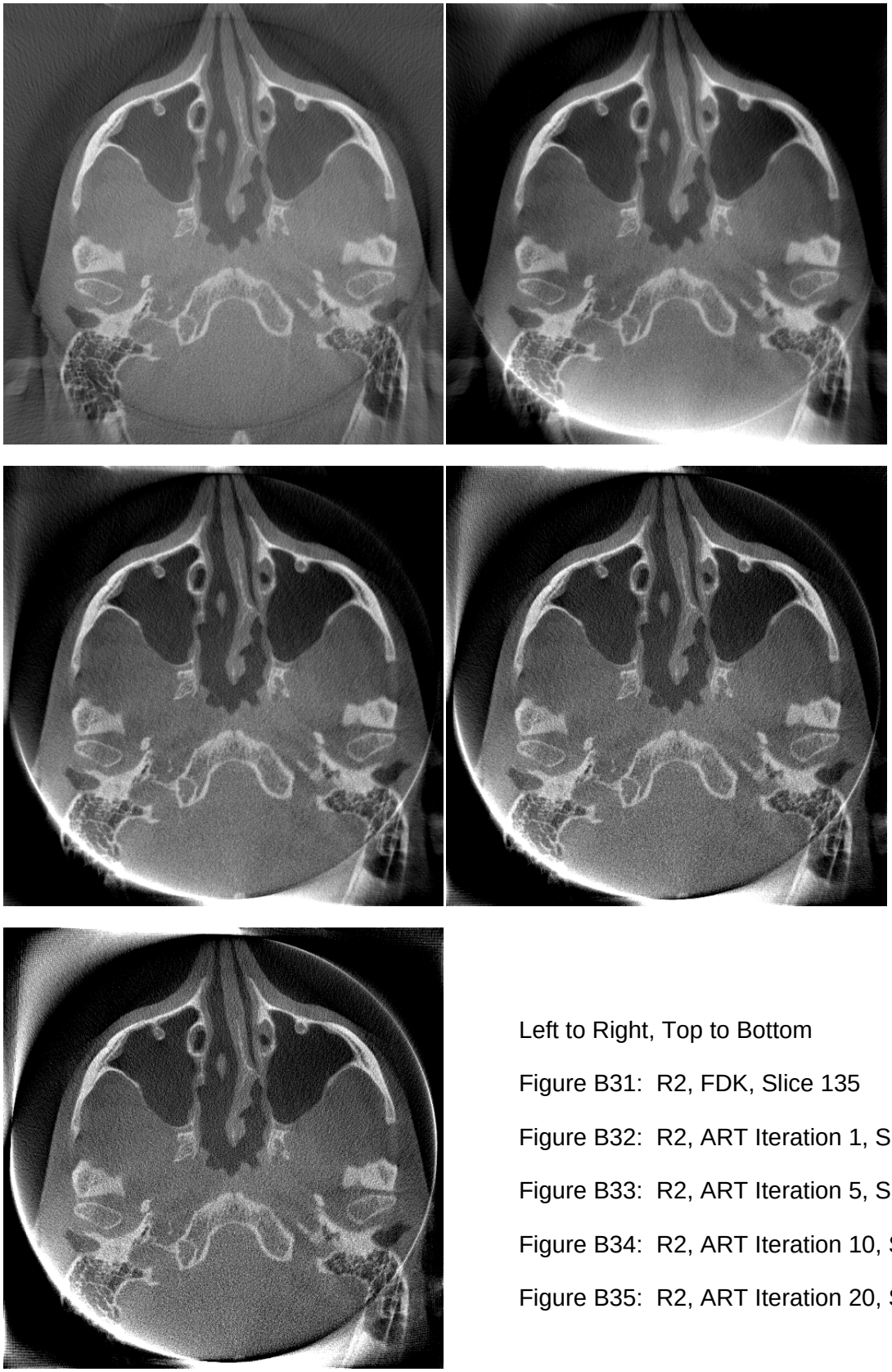
Figure B26: R1, FDK, Slice 364

Figure B27: R1, MART2 Iteration 1, Slice 364

Figure B28: R1, MART2 Iteration 5, Slice 364

Figure B29: R1, MART2 Iteration 10, Slice 364

Figure B30: R1, MART2 Iteration 20, Slice 364



Left to Right, Top to Bottom

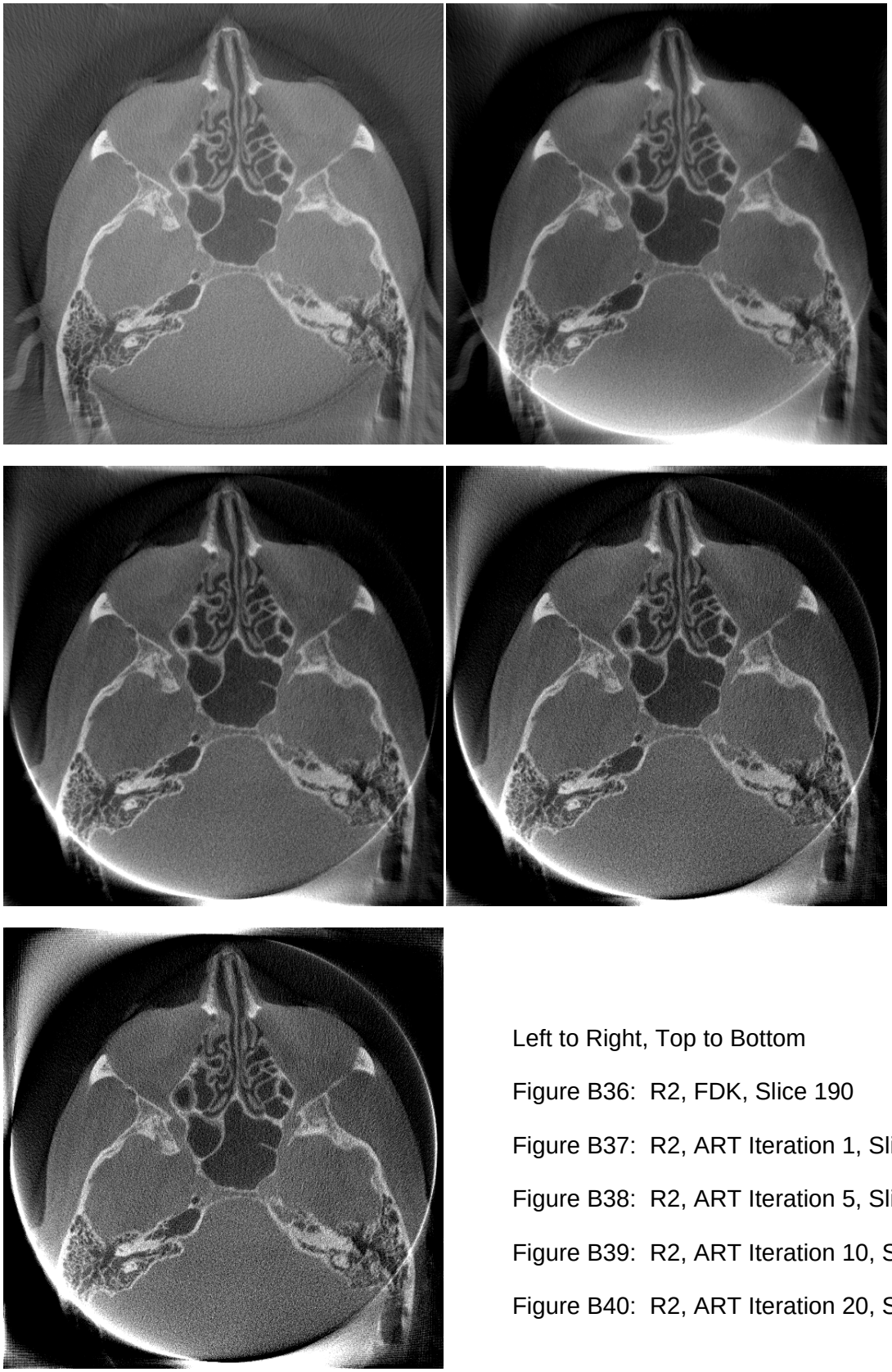
Figure B31: R2, FDK, Slice 135

Figure B32: R2, ART Iteration 1, Slice 135

Figure B33: R2, ART Iteration 5, Slice 135

Figure B34: R2, ART Iteration 10, Slice 135

Figure B35: R2, ART Iteration 20, Slice 135



Left to Right, Top to Bottom

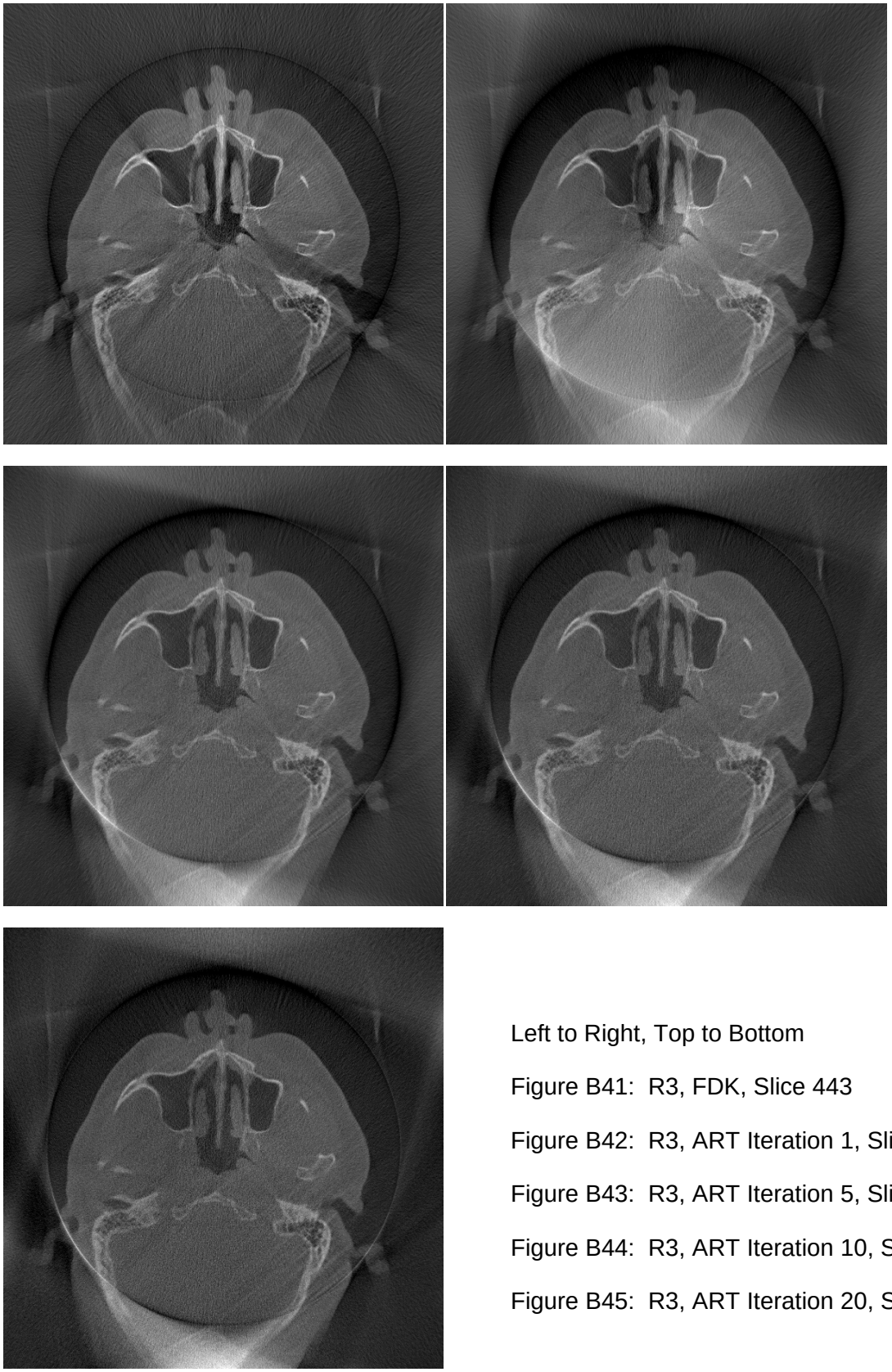
Figure B36: R2, FDK, Slice 190

Figure B37: R2, ART Iteration 1, Slice 190

Figure B38: R2, ART Iteration 5, Slice 190

Figure B39: R2, ART Iteration 10, Slice 190

Figure B40: R2, ART Iteration 20, Slice 190



Left to Right, Top to Bottom

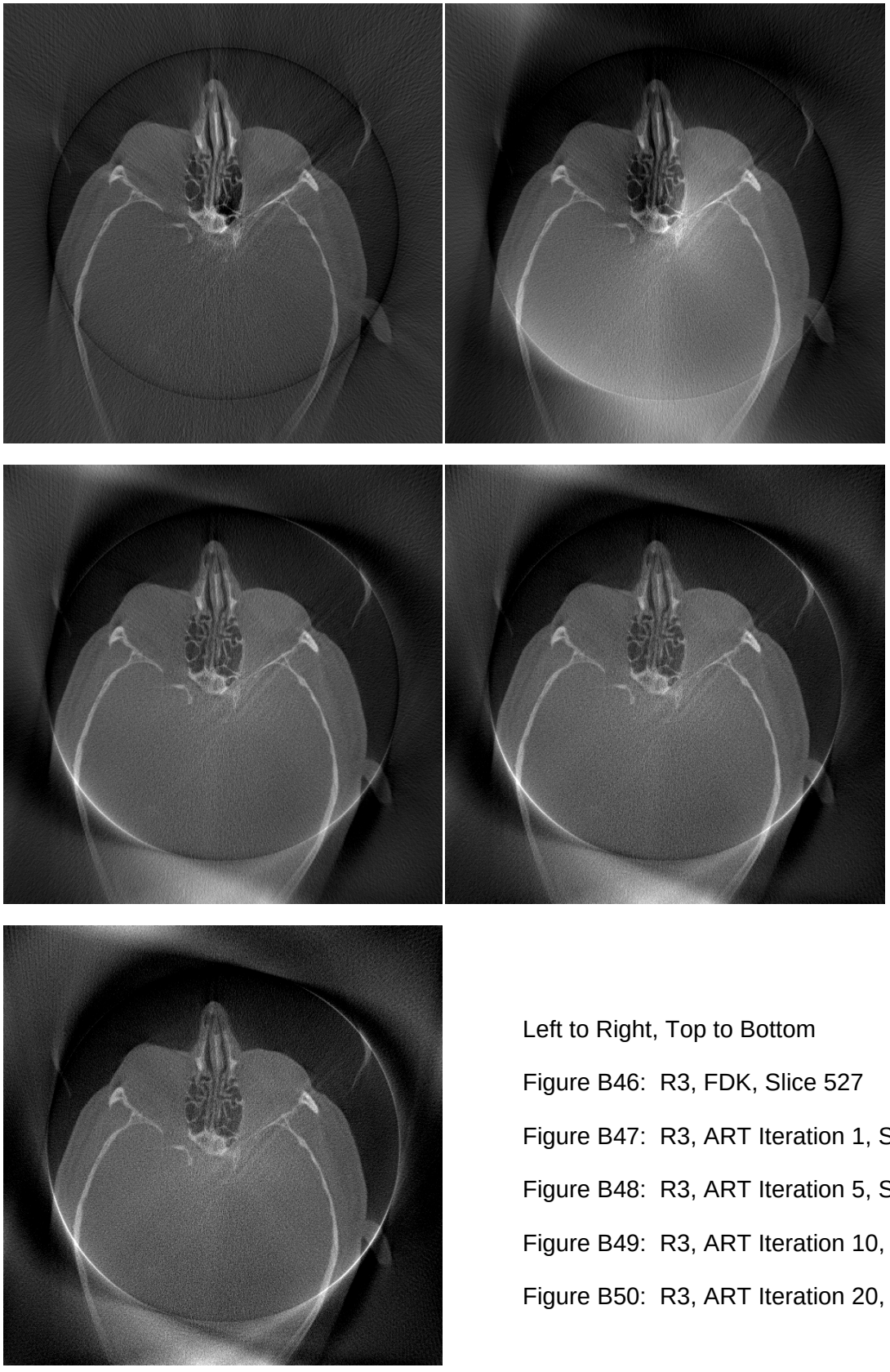
Figure B41: R3, FDK, Slice 443

Figure B42: R3, ART Iteration 1, Slice 443

Figure B43: R3, ART Iteration 5, Slice 443

Figure B44: R3, ART Iteration 10, Slice 443

Figure B45: R3, ART Iteration 20, Slice 443



Left to Right, Top to Bottom

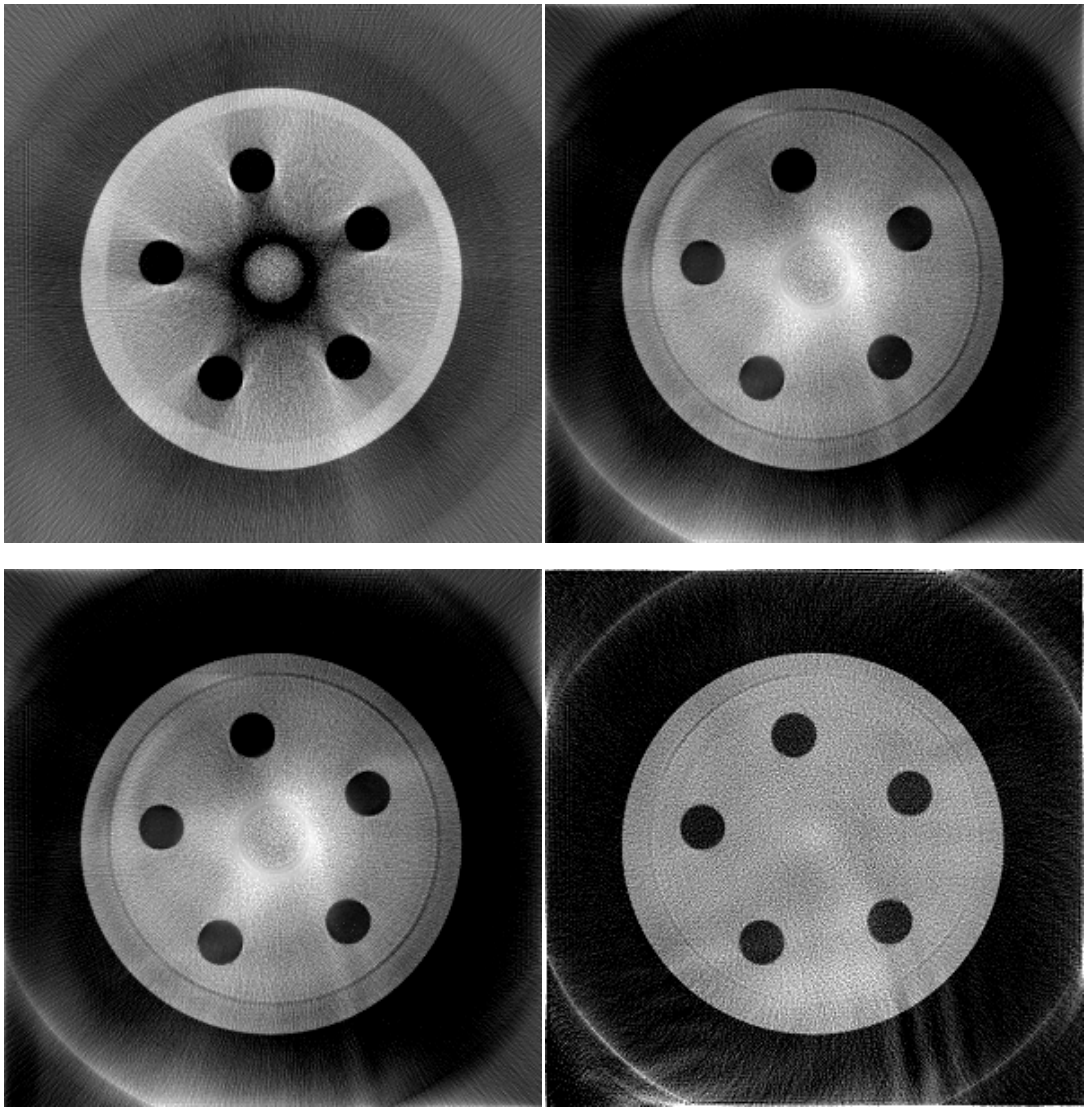
Figure B46: R3, FDK, Slice 527

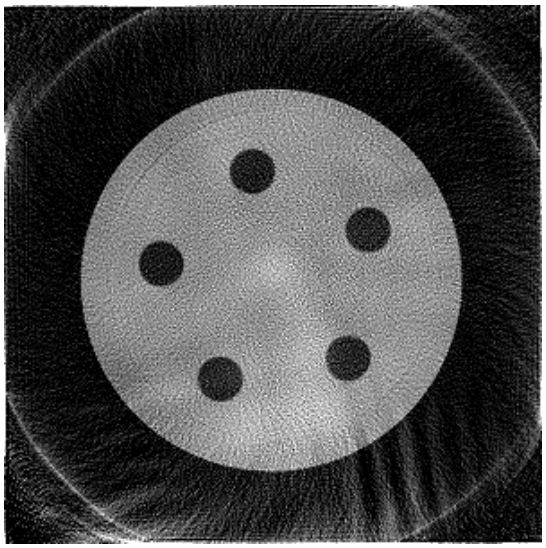
Figure B47: R3, ART Iteration 1, Slice 527

Figure B48: R3, ART Iteration 5, Slice 527

Figure B49: R3, ART Iteration 10, Slice 527

Figure B50: R3, ART Iteration 20, Slice 527





Left to Right, Top to Bottom

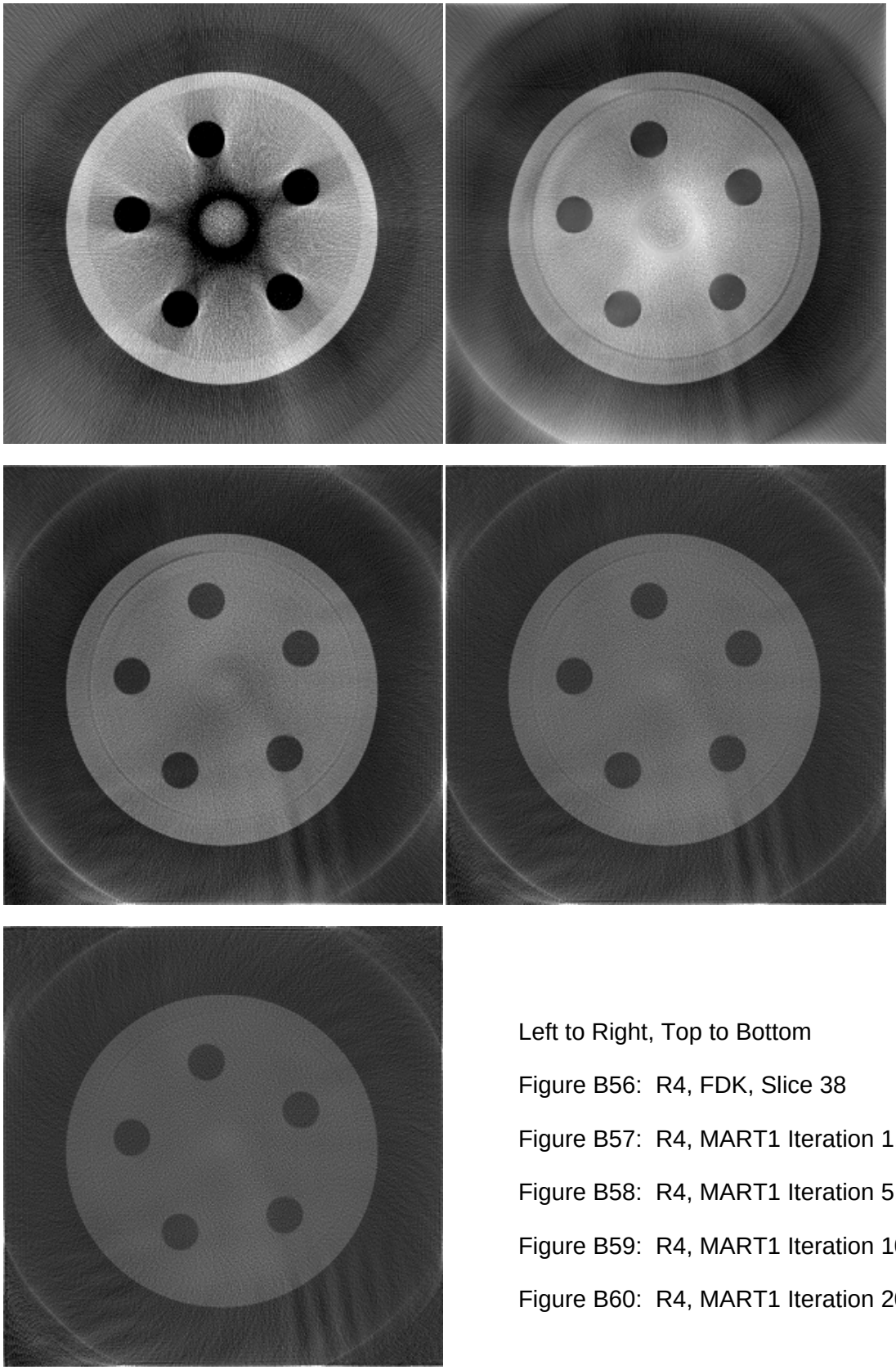
Figure B51: R4, FDK, Slice 38

Figure B52: R4, ART Iteration 1, Slice 38

Figure B53: R4, ART Iteration 5, Slice 38

Figure B54: R4, ART Iteration 10, Slice 38

Figure B55: R4, ART Iteration 20, Slice 38



Left to Right, Top to Bottom

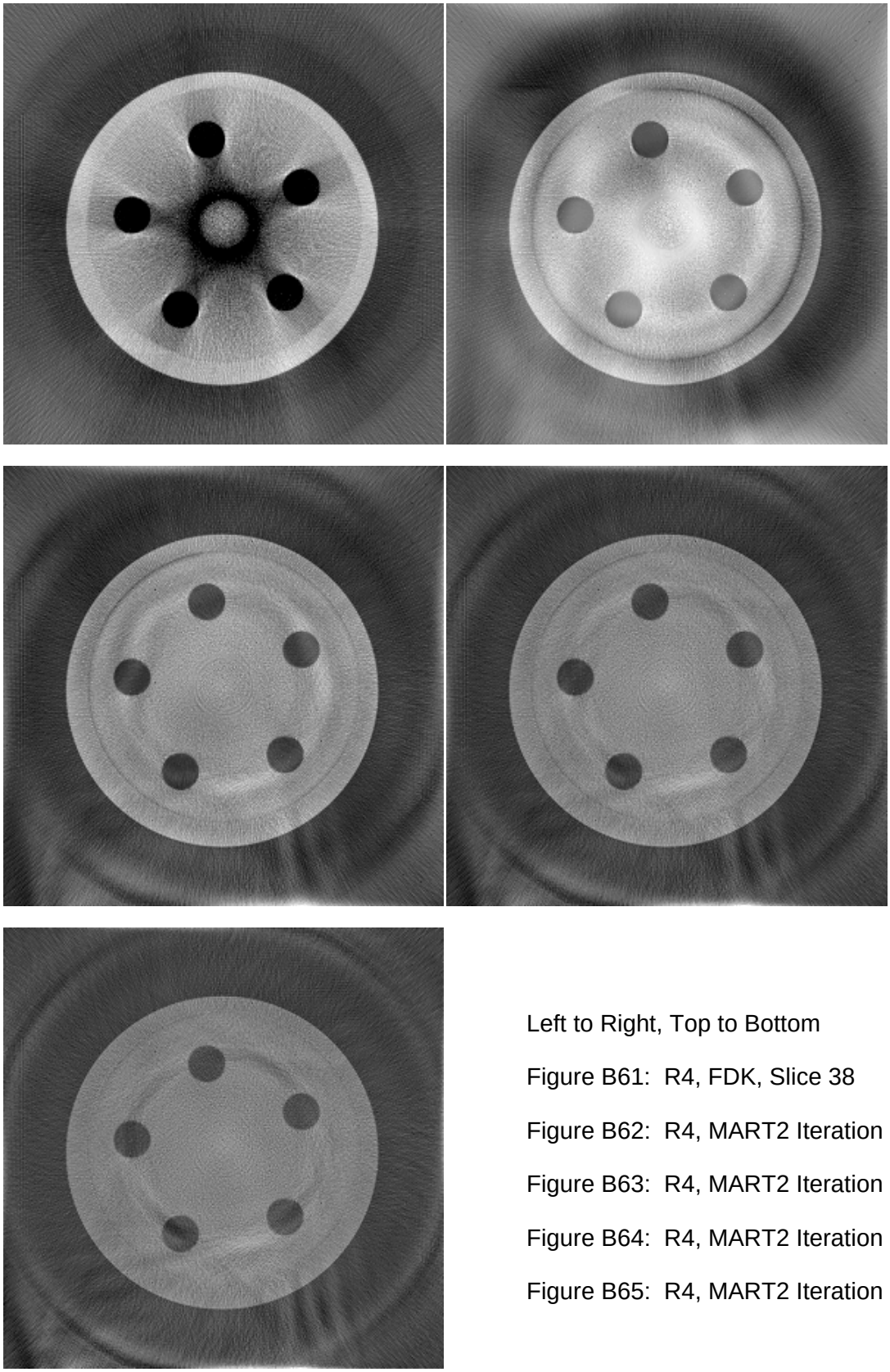
Figure B56: R4, FDK, Slice 38

Figure B57: R4, MART1 Iteration 1, Slice 38

Figure B58: R4, MART1 Iteration 5, Slice 38

Figure B59: R4, MART1 Iteration 10, Slice 38

Figure B60: R4, MART1 Iteration 20, Slice 38



Left to Right, Top to Bottom

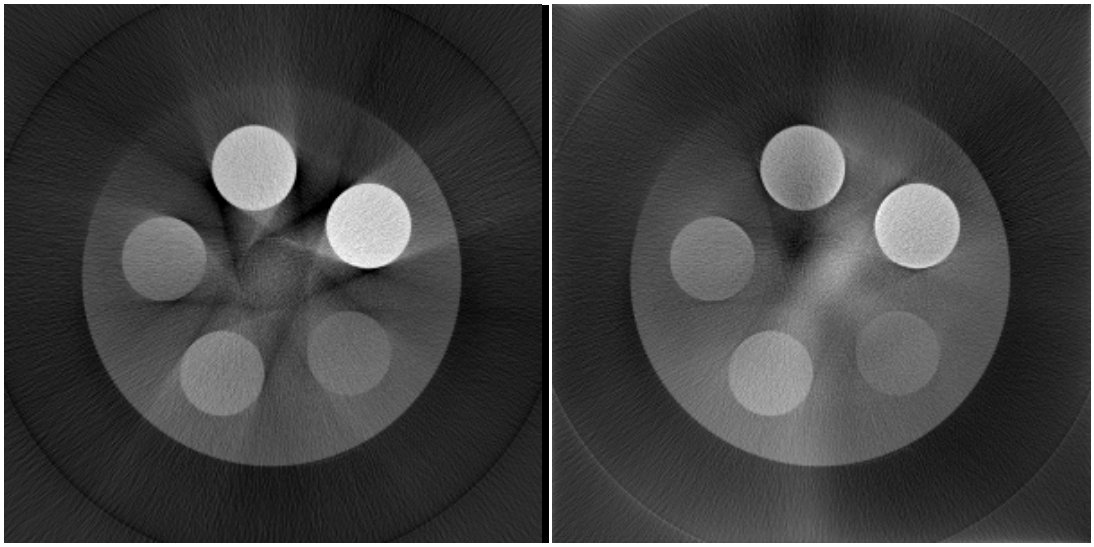
Figure B61: R4, FDK, Slice 38

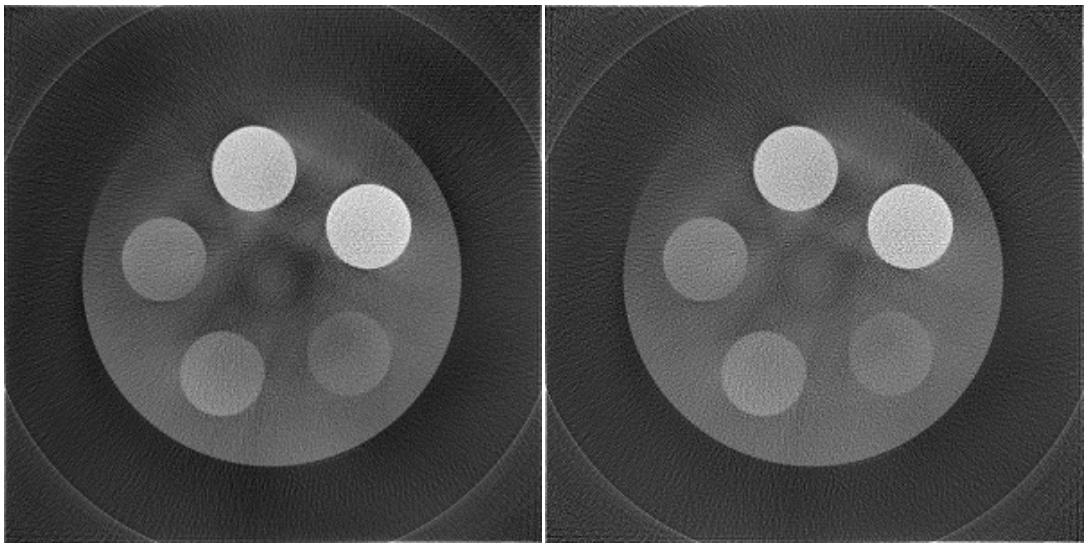
Figure B62: R4, MART2 Iteration 1, Slice 38

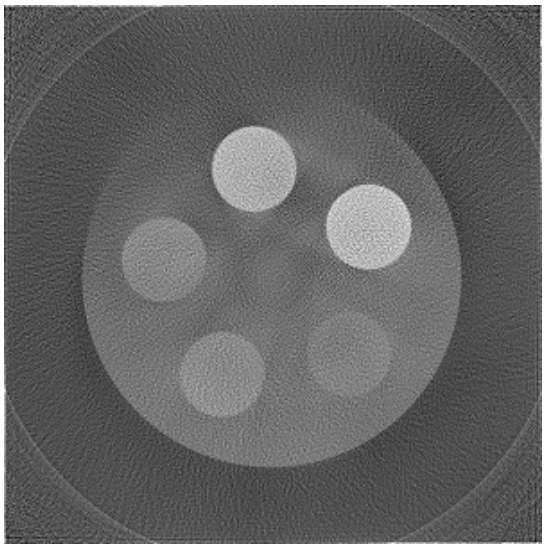
Figure B63: R4, MART2 Iteration 5, Slice 38

Figure B64: R4, MART2 Iteration 10, Slice 38

Figure B65: R4, MART2 Iteration 20, Slice 38







Left to Right, Top to Bottom

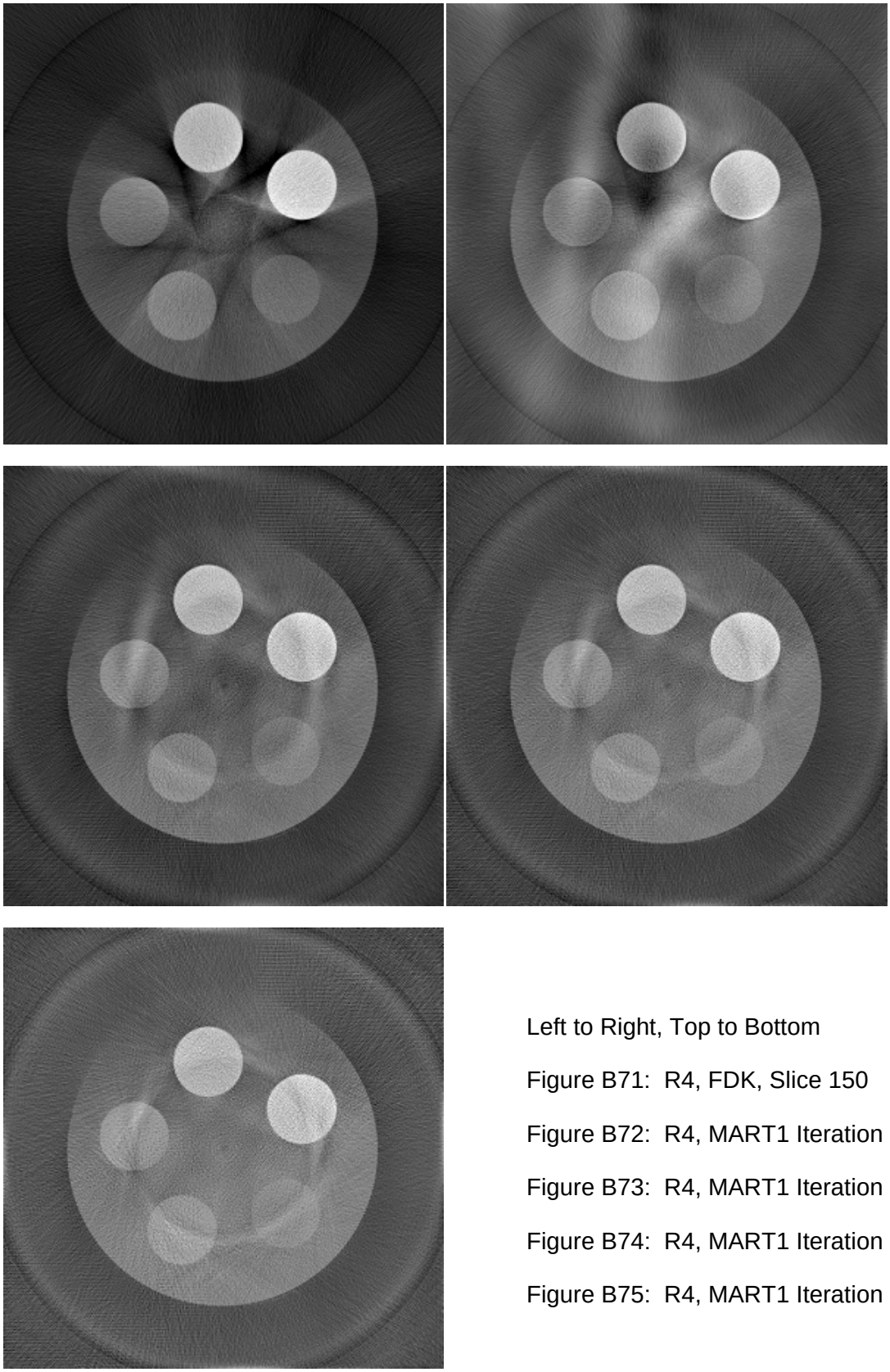
Figure B66: R4, FDK, Slice 150

Figure B67: R4, ART Iteration 1, Slice 150

Figure B68: R4, ART Iteration 5, Slice 150

Figure B69: R4, ART Iteration 10, Slice 150

Figure B70: R4, ART Iteration 20, Slice 150



Left to Right, Top to Bottom

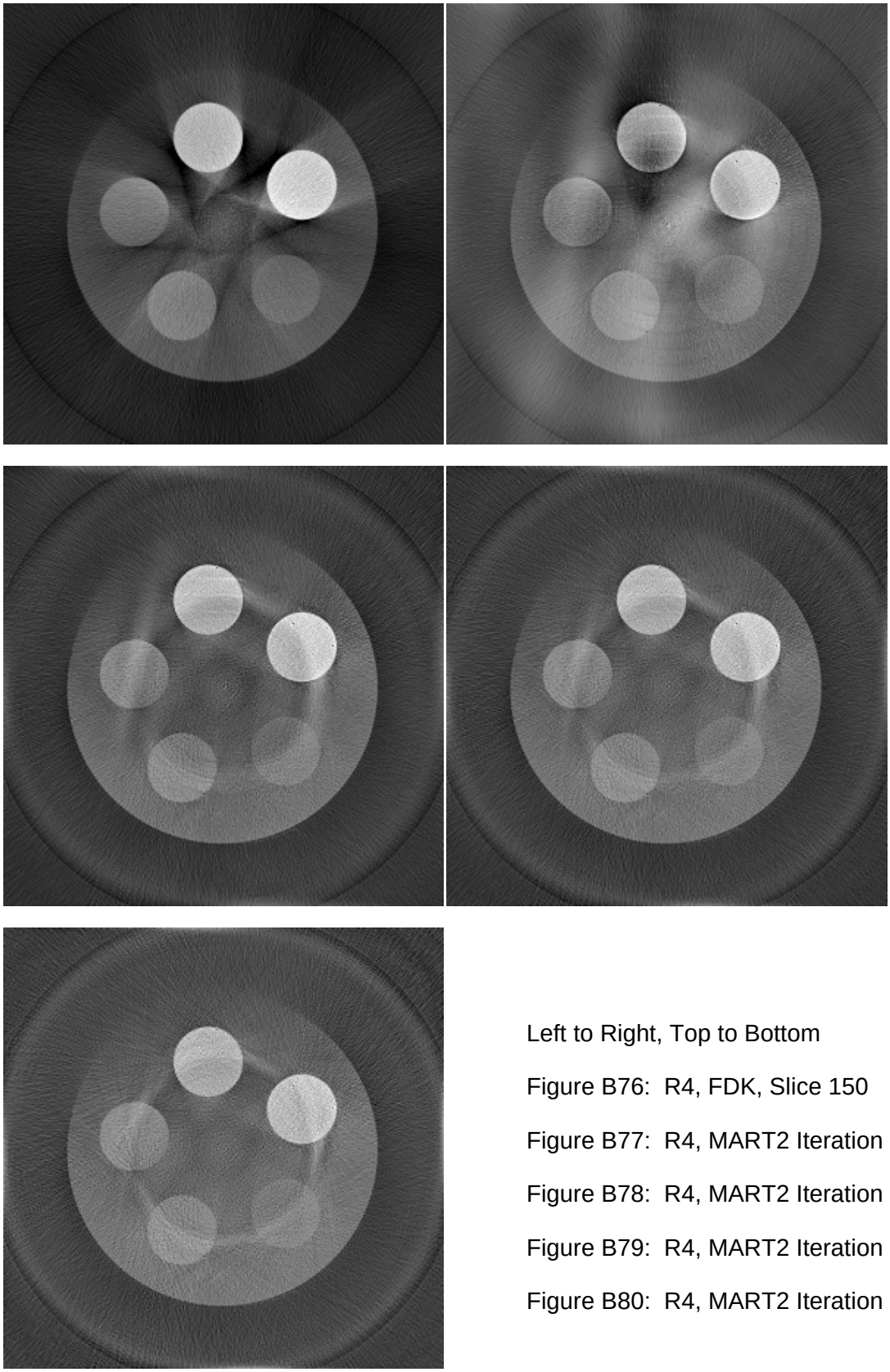
Figure B71: R4, FDK, Slice 150

Figure B72: R4, MART1 Iteration 1, Slice 150

Figure B73: R4, MART1 Iteration 5, Slice 150

Figure B74: R4, MART1 Iteration 10, Slice 150

Figure B75: R4, MART1 Iteration 20, Slice 150



Left to Right, Top to Bottom

Figure B76: R4, FDK, Slice 150

Figure B77: R4, MART2 Iteration 1, Slice 150

Figure B78: R4, MART2 Iteration 5, Slice 150

Figure B79: R4, MART2 Iteration 10, Slice 150

Figure B80: R4, MART2 Iteration 20, Slice 150

Appendix C

Runtime Results from the 20 Iteration Runs of Reconstruction Algorithms on GTX560 Graphics Card

Note: All runtime results are in seconds

List of Figures

Figure C1: Runtimes of 20 Iteration Runs of R1 Data Set by Reconstruction Algorithms on GTX560 114

Figure C2: Runtimes of 20 Iteration Runs of R2 Data Set by Reconstruction Algorithms on GTX560 114

Figure C3: Runtimes of 20 Iteration Runs of R3 Data Set by Reconstruction Algorithms on GTX560 115

Figure C4: Runtimes of 20 Iteration Runs of R4 Data Set by Reconstruction Algorithms on GTX560 115

List of Tables

Table C1: Runtimes of 20 Iteration Runs of ART Algorithm on GTX560 111

Table C2: Runtimes of 20 Iteration Runs of Material ART Algorithm Version 1 on GTX560 112

Table C3: Runtimes of 20 Iteration Runs of Material ART Algorithm Version 2 on GTX560 113

Table C1: Runtimes of 20 Iteration Runs of ART Algorithm on GTX560

	20 Iterations of ART algorithm on GTX560							
	R1		R2		R3		R4	
0	84.3		183.4		157.6		80.1	
1	598.5	514.2	1191.5	1008.1	1239.0	1081.4	268.7	188.6
2	1160.3	561.8	2199.7	1008.2	2339.2	1100.2	457.9	189.2
3	1814.8	654.5	3208.0	1008.3	3439.1	1099.9	646.6	188.7
4	2419.4	604.6	4216.1	1008.1	4515.4	1076.3	835.2	188.6
5	2921.4	502.0	5259.8	1043.7	5593.6	1078.2	1023.9	188.7
6	3438.8	517.4	6568.4	1308.6	6615.2	1021.6	1214.9	191.0
7	3940.8	502.0	7588.8	1020.4	7692.2	1077.0	1403.4	188.5
8	4442.9	502.1	8603.7	1014.9	8768.6	1076.4	1591.8	188.4
9	4944.9	502.0	9612.6	1008.9	9844.6	1076.0	1779.5	187.7
10	5447.1	502.2	10621.4	1008.8	10921.6	1077.0	1967.0	187.5
11	5964.1	517.0	11644.9	1023.5	12022.9	1101.3	2157.2	190.2
12	6466.3	502.2	12653.5	1008.6	13100.4	1077.5	2353.3	196.1
13	6971.9	505.6	13662.0	1008.5	14176.8	1076.4	2540.8	187.5
14	7692.7	720.8	14670.6	1008.6	15254.2	1077.4	2728.4	187.6
15	8229.8	537.1	15686.5	1015.9	16331.0	1076.8	2915.8	187.4
16	8747.3	517.5	16713.0	1026.5	17431.5	1100.5	3105.5	189.7
17	9249.4	502.1	17721.6	1008.6	18508.4	1076.9	3293.5	188.0
18	9751.5	502.1	18730.3	1008.7	19585.9	1077.5	3482.0	188.5
19	10253.5	502.0	19738.9	1008.6	20583.0	997.1	3670.4	188.4
20	10755.7	502.2	20747.6	1008.7	21660.2	1077.2	3859.1	188.7
AVERAGE		533.6		1028.2		1075.1		189.0

Table C2: Runtimes of 20 Iteration Runs of Material ART Algorithm Version 1 on GTX560

	20 Iterations of Mtl ART algorithm, ver. 1 on GTX560							
	R1		R2		R3		R4	
0	82.5		184.4		159.2		81.5	
1	317.5	235.0	929.4	745.0	672.3	513.1	239.2	157.7
2	542.8	225.3	1608.9	679.5	1177.1	504.8	347.7	108.5
3	766.0	223.2	2283.1	674.2	1682.2	505.1	456.4	108.7
4	974.8	208.8	2942.1	659.0	2164.0	481.8	562.8	106.4
5	1183.6	208.8	3601.0	658.9	2646.4	482.4	669.2	106.4
6	1407.1	223.5	4275.1	674.1	3151.3	504.9	777.8	108.6
7	1615.6	208.5	4934.1	659.0	3633.5	482.2	884.2	106.4
8	1823.8	208.2	5592.9	658.8	4115.4	481.9	990.7	106.5
9	2032.1	208.3	6251.6	658.7	4597.9	482.5	1097.2	106.5
10	2241.4	209.3	6910.4	658.8	5080.5	482.6	1203.6	106.4
11	2465.1	223.7	7584.2	673.8	5572.1	491.6	1312.0	108.4
12	2674.6	209.5	8243.2	659.0	6039.9	467.8	1418.1	106.1
13	2884.0	209.4	8902.2	659.0	6508.5	468.6	1524.3	106.2
14	3093.5	209.5	9561.1	658.9	6976.9	468.4	1630.4	106.1
15	3302.9	209.4	10220.0	658.9	7445.8	468.9	1736.6	106.2
16	3527.3	224.4	10903.9	683.9	7938.0	492.2	1845.3	108.7
17	3736.9	209.6	11572.8	668.9	8406.4	468.4	1951.5	106.2
18	3946.5	209.6	12241.8	669.0	8874.6	468.2	2057.7	106.2
19	4156.0	209.5	12911.0	669.2	9343.1	468.5	2164.0	106.3
20	4366.1	210.1	13580.5	669.5	9746.8	403.7	2270.3	106.3
AVERAGE		214.2		669.8		479.4		109.4

Table C3: Runtimes of 20 Iteration Runs of Material ART Algorithm Version 2 on GTX560

	20 Iterations of Mtl ART algorithm, ver. 2 on GTX560							
	R1		R2		R3		R4	
0	85.8		184.1		156.3		82.3	
1	350.6	264.8	1243.9	1059.8	752.3	596.0	283.0	200.7
2	605.7	255.1	2242.7	998.8	1332.7	580.4	435.0	152.0
3	862.1	256.4	3231.7	989.0	1911.0	578.3	586.3	151.3
4	1102.1	240.0	4204.0	972.3	2452.8	541.8	735.5	149.2
5	1342.2	240.1	5176.3	972.3	2996.1	543.3	884.6	149.1
6	1597.6	255.4	6164.0	987.7	3574.8	578.7	1036.3	151.7
7	1837.5	239.9	7136.4	972.4	4114.5	539.7	1185.6	149.3
8	2077.8	240.3	8108.8	972.4	4654.8	540.3	1334.9	149.3
9	2317.9	240.1	9081.2	972.4	5195.3	540.5	1484.1	149.2
10	2558.0	240.1	10053.6	972.4	5735.6	540.3	1633.4	149.3
11	2813.6	255.6	11042.5	988.9	6313.6	578.0	1785.2	151.8
12	3053.7	240.1	12014.8	972.3	6863.3	549.7	1934.3	149.1
13	3293.7	240.0	13004.2	989.4	7415.0	551.7	2084.0	149.7
14	3533.7	240.0	13976.4	972.2	7967.9	552.9	2233.7	149.7
15	3773.8	240.1	14948.7	972.3	8521.1	553.2	2383.4	149.7
16	4028.8	255.0	15937.6	988.9	9099.8	578.7	2535.5	152.1
17	4268.8	240.0	16909.7	972.1	9653.3	553.5	2685.0	149.5
18	4508.8	240.0	17881.8	972.1	10143.0	489.7	2834.8	149.8
19	4748.9	240.1	18853.8	972.0	10695.6	552.6	2984.7	149.9
20	4989.5	240.6	19826.5	972.7	11247.7	552.1	3134.7	150.0
AVERAGE		245.2		982.1		554.6		152.6

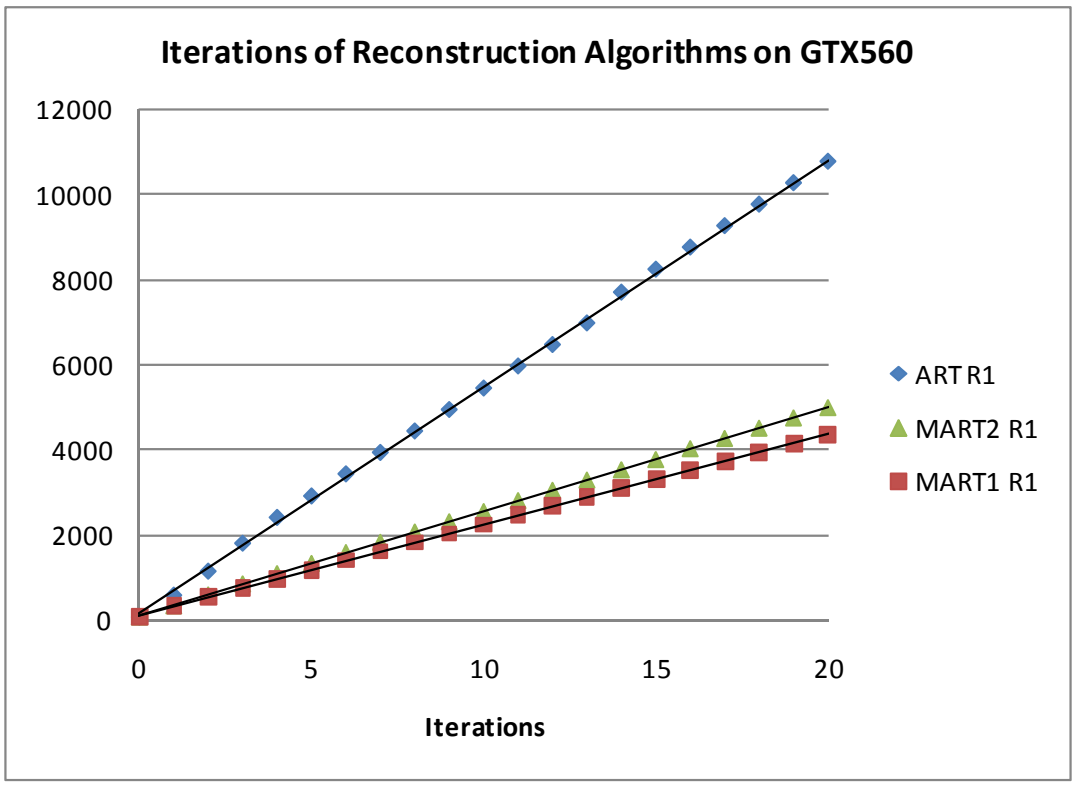


Figure C1: Runtime of 20 Iteration Runs of R1 Data Set by Reconstruction Algorithms on GTX560

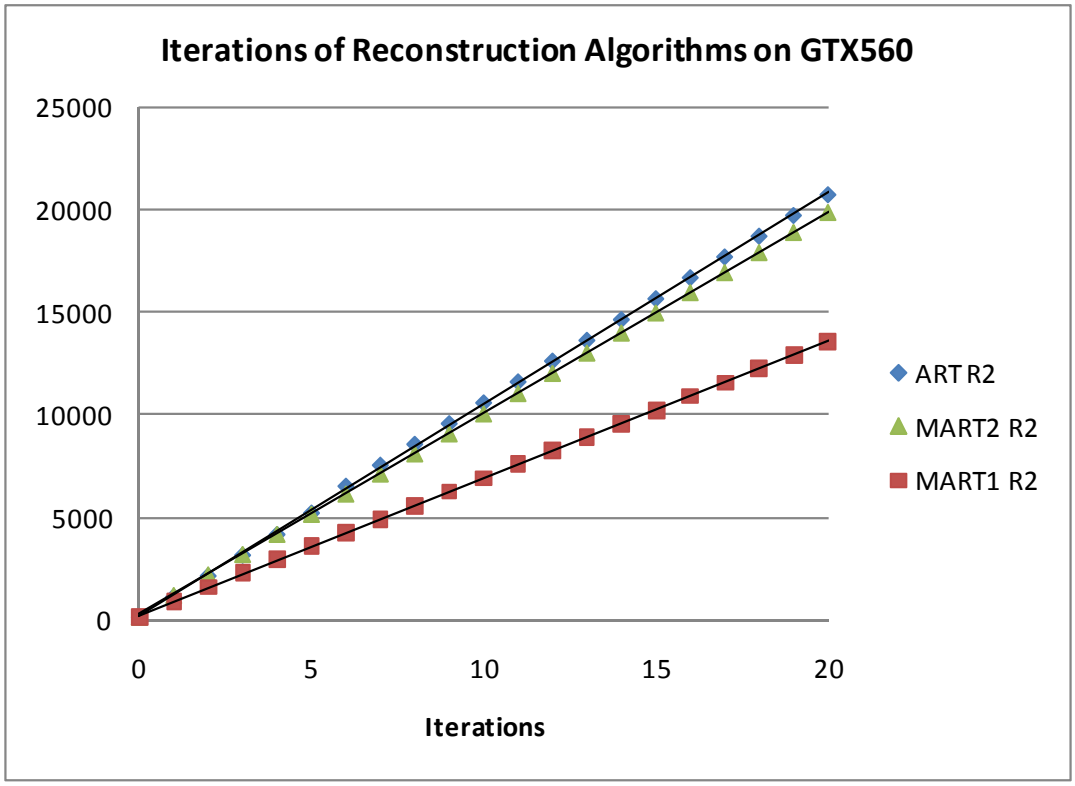


Figure C2: Runtime of 20 Iteration Runs of R2 Data Set by Reconstruction Algorithms on GTX560

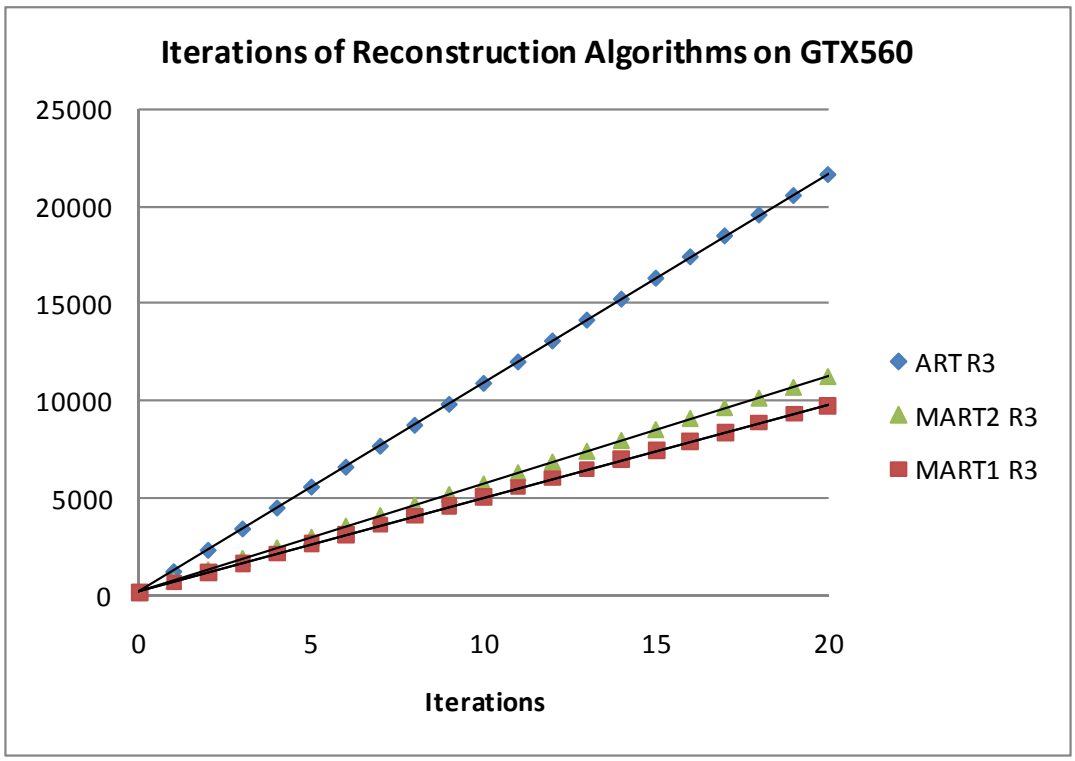


Figure C3: Runtime of 20 Iteration Runs of R3 Data Set by Reconstruction Algorithms on GTX560

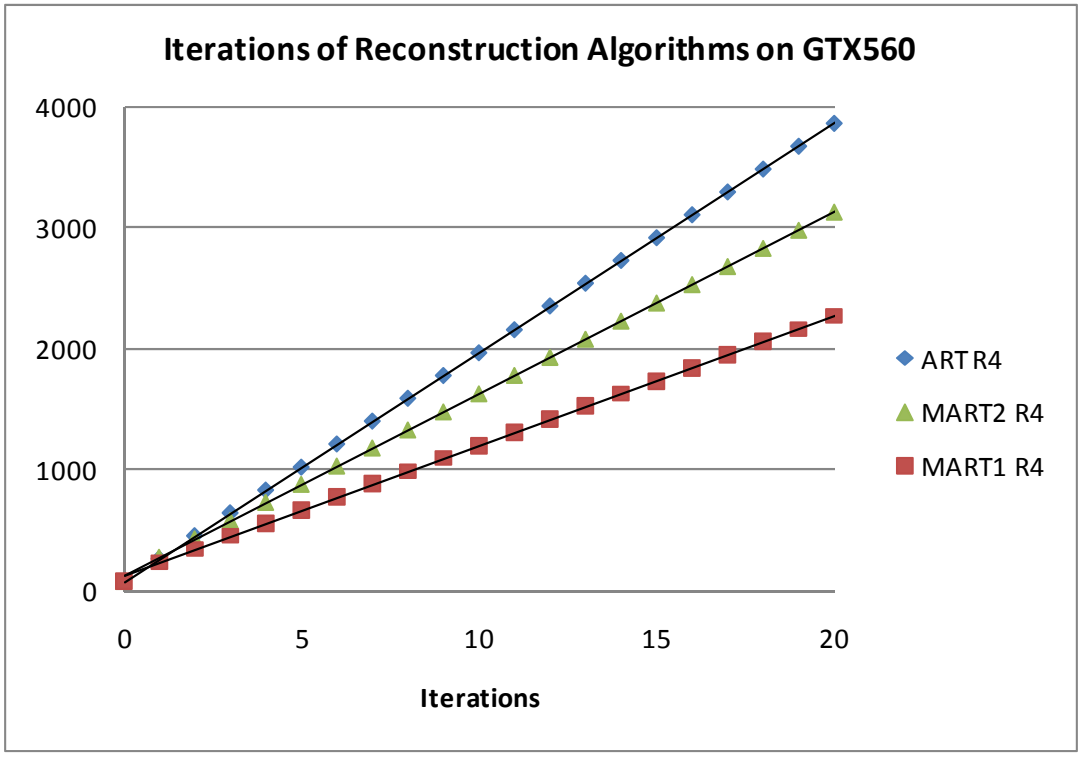


Figure C4: Runtime of 20 Iteration Runs of R4 Data Set by Reconstruction Algorithms on GTX560