Real-time GPU implementation of a weighted filtered back-projection algorithm for stationary gantry CT reconstruction

William Thompson[a,*], **Edward Morton**[a], **Alexander Katsevich**[b,c], **Seongjin Yoon**[b], **Michael Frenkel**[b]
[a]Rapiscan Systems, 2805 Columbia St, Torrance, CA 90503, USA
[b]iTomography Corp., Texas Medical Center Innovation Institute, 2450 Holcombe Blvd, Houston, TX 77021, USA
[c]University of Central Florida, Mathematics Department, Orlando, FL 32816, USA

Abstract. We present details of a real-time implementation of a new algorithm designed to reduce streak artifacts in switched-source stationary gantry CT reconstruction. The algorithm is of the filtered back-projection type, and uses a voxel-specific weighting function to account for the non-uniform distribution of illumination angles caused by such a scanning geometry. The main challenge in developing a real-time implementation is the storage and memory bandwidth requirements imposed by the weighting function. This has been addressed by storing weights at a low precision and reduced resolution, and using interpolation to recover weights at the full resolution. Results demonstrate real-time performance of the algorithm at a realistic problem size, running on a low-cost consumer grade laptop.

Keywords: image reconstruction, weight function, real-time, GPU.

*William Thompson, withompson@rapiscansystems.com

1 Introduction

The Real Time Tomography (RTT) system[1] is a family of fast cone beam CT scanners developed by Rapiscan Systems, designed for applications where scanning rate and/or continuous operation are of critical importance. The system achieves its high speed by employing a stationary gantry, containing a fixed ring of discrete x-ray sources. By switching the sources in order, the speed of this design is constrained only by data acquisition bandwidth and available photon flux. The system uses several fixed rings of detectors, offset from the plane of sources, as shown in Fig. 1.

Sources can be fired in almost any sequence. As explained in Ref. 2, for optimal data acquisition, the ideal firing order is non-sequential; this presents challenges for reconstruction algorithm design due to the discontinuous virtual source trajectory. An additional challenge is that the distribution of illumination angles varies considerably over the field of view. The RTT system design has also been extended to include non-circular system geometries, further increasing this variability.

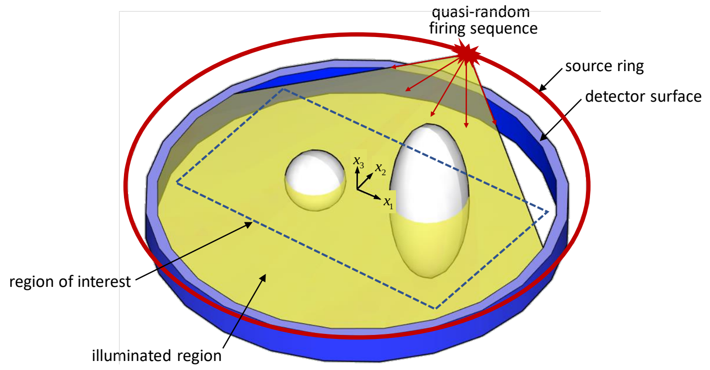

Fig 1 Conceptual drawing of the RTT system geometry.

In order to support continuous real-time reconstruction, it is necessary to implement the algorithm as a pipeline, capable of running at data rates of several hundred MB/s. As an example, for the commercially available single-energy RTT110 scanner, the required reconstruction rate is 480 slices per second, which results in a sustained output data rate of over 400MB/s. For multi-energy systems, this will be even higher. The analysis in Ref. 2 was based on an iterative reconstruction algorithm, easily adaptable to any system geometry. However, such algorithms are still not computationally feasible at the high data rates necessary for real-time use. Therefore, an analytical algorithm of sufficiently low computational complexity to be capable of being implemented in real-time, yet still giving good image quality with unusual system geometries, was sought. The work presented in this paper gives details of the real-time implementation of such an algorithm, developed by Katsevich et al., presented in Ref. 3.

2 Mathematical formulation of the algorithm

The implemented algorithm is based on the formula (3) in Ref. 3, and is described in detail in that paper. The algorithm is defined in the 2D sense and extended to 3D using an FDK-type cone beam weighting. After performing integration by parts with respect to s, in a similar manner to Ref. 4, the full reconstruction formula, defined in the flat detector geometry, reads:

$$\begin{aligned} f(\vec{x}) = \frac{1}{2\pi}\int_S & \left(\frac{w(\vec{x},s)\big(\vec{y}'(s)\cdot(\vec{x}-\vec{y}(s))\big)}{|\vec{x}-\vec{y}(s)|^3} + \frac{w(\vec{x},s)'}{|\vec{x}-\vec{y}(s)|} \right) \\ & \cdot \sqrt{R^2+u_0^2(\vec{x},s)} \int_{-\infty}^{\infty} \frac{\hat{f}(\vec{y}(s),\gamma(u))/\sqrt{R^2+u^2}}{u-u_0(\vec{x},s)} du\, ds \\ + \frac{1}{2\pi}\int_S & \frac{w(\vec{x},s)|\vec{y}'(s)\times(\vec{x}-\vec{y}(s))|}{|\vec{x}-\vec{y}(s)|^3} \\ & \cdot \frac{\sqrt{R^2+u_0^2(\vec{x},s)}}{R} \int_{-\infty}^{\infty} \frac{\partial_u \hat{f}(\vec{y}(s),\gamma(u))\sqrt{R^2+u^2}}{u-u_0(\vec{x},s)} du\, ds. \end{aligned} \tag{1}$$

Here S is the parametric interval that describes the source trajectory $\vec{y}(s)$, and $w(\vec{x},s)$ is a weighting function, chosen as in Ref. 3 based on the sparsity of the angular illumination pattern.

3 Implementation details

3.1 General approach

The inversion formula of (1) essentially breaks down into the sum of two weighted back-projection terms, the first term operating on Hilbert transformed data, and the second one on ramp filtered data. The ramp filter term is implemented as the difference of a ramp filter and Hilbert transform term in order to avoid a finite difference operation. Fig. 2 shows data flow through the implemented algorithm pipeline, which forms part of a wider real-time data acquisition and imaging pipeline. The pipeline runs entirely on the GPU, and is coded in CUDA. Data are copied to and from the device using asynchronous functions, to allow potential overlap of copying and computation.

The pipeline operates with a granularity of one revolution, defined as the set of all projections from every source having been fired once. A single revolution of input projection data corresponds to an integer number of output reconstructed slices; in this way, the reconstruction geometry is periodic, and the pipeline operations on each revolution are identical.

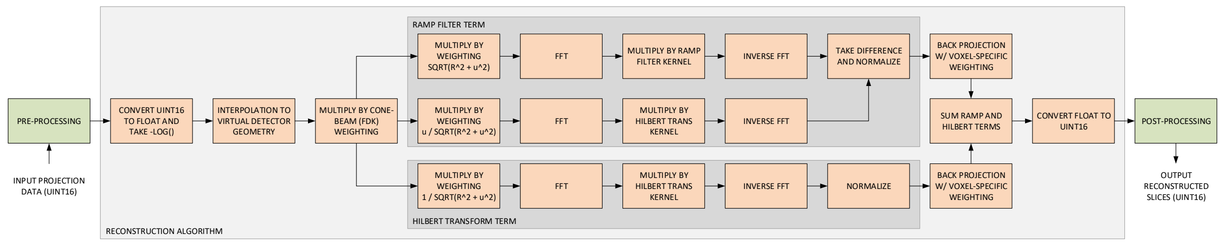
RAMP FILTER TERM
PRE-PROCESSING
INPUT PROJECTION DATA (UINT16)
CONVERT UINT16 TO FLOAT AND TAKE -LOG()
INTERPOLATION TO VIRTUAL DETECTOR GEOMETRY
MULTIPLY BY CONE-BEAM (FDK) WEIGHTING
MULTIPLY BY WEIGHTING SQRT(R^2 + u^2)
FFT
MULTIPLY BY RAMP FILTER KERNEL
INVERSE FFT
TAKE DIFFERENCE AND NORMALIZE
BACK PROJECTION W/ VOXEL-SPECIFIC WEIGHTING
MULTIPLY BY WEIGHTING u / SQRT(R^2 + u^2)
FFT
MULTIPLY BY HILBERT TRANS KERNEL
INVERSE FFT
SUM RAMP AND HILBERT TERMS
CONVERT FLOAT TO UINT16
POST-PROCESSING
OUTPUT RECONSTRUCTED SLICES (UINT16)
MULTIPLY BY WEIGHTING 1 / SQRT(R^2 + u^2)
FFT
MULTIPLY BY HILBERT TRANS KERNEL
INVERSE FFT
NORMALIZE
BACK PROJECTION W/ VOXEL-SPECIFIC WEIGHTING
HILBERT TRANSFORM TERM
RECONSTRUCTION ALGORITHM

Fig 2 Data flow through the implemented algorithm pipeline.

3.2 Back-projection

The back-projection kernel is the most computationally expensive step in the algorithm pipeline. This is based on a texture memory implementation, where the coordinates on a virtual flat detector are calculated, and the projection data values read using built-in linear interpolation. Data for each term of (1) are stored using a CUDA built-in vector type, which allows the values for the two terms to be loaded using a single texture read. To compensate for the doubled memory bandwidth, the filtered projection data are stored in the 16 bit half-precision floating-point format, and converted to standard 32 bit floating-point values in hardware during the texture read. As observed in Ref. 5, this results in negligible loss of accuracy, but significant increase in performance.

3.3 Weighting functions

Calculation of the weighting function $w(\vec{x}, s)$ is computationally expensive, and it is therefore necessary to pre-compute and store the values for every source-voxel combination. In our implementation, $w(\vec{x}, s)$ is combined with the numerical quadrature weight for integration over s in (1), to form a single compound weighting function. At the typical problem size for the commercially available RTT110 scanner, it would take approximately 24GB to store the complete set of weighting values for both terms of (1) in single-precision floating-point format. Although this is feasible with current generation high-end GPU hardware, it is desirable to reduce this, since loading the values from GPU memory in the back-projection kernel is a significant performance bottleneck.

In practice, it is not necessary to store the weighting functions at high precision, and using the 16 bit half-precision floating-point format yields negligible differences, giving a factor of two reduction. The in-plane weighting functions are generally smooth, but with some discontinuities; therefore, we compute and store the weighting functions for each source-slice pair at a reduced resolution, and use GPU hardware texture interpolation to scale back up to the output reconstructed slice resolution. Empirically, we have found that using a resolution reduction factor of 4 results in no major image quality degradation, but reduces memory bandwidth and storage requirements by a factor 16. Fig. 3 shows that differences at the image center are negligible, but increase towards the edge regions. In practical use, such differences will likely be below the noise level.

4 Performance analysis

The implementation was tested using data from a Rapiscan RTT110 scanner operating at 30 revolutions per second, using a reconstructed slice size of 864×512 voxels, and 16 slices per reconstructed volume chunk. The test hardware was an Alienware 17 R4 laptop, with nVidia GTX1080 GPU, containing 2560 CUDA cores and 8GB on-board RAM. Fig. 4 shows a timeline view of

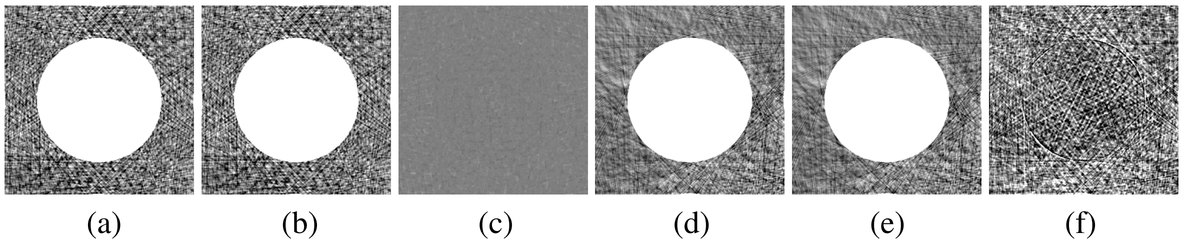

Fig 3 Results of reconstruction from simulated, noise-free data, with and without interpolation of the weighting functions; all images using grayscale window of ±10% of full scale: (a) – (c) center region, (d) – (f) edge region; (a, d) no interpolation, (b, e) 4× interpolation, (c, f) difference.

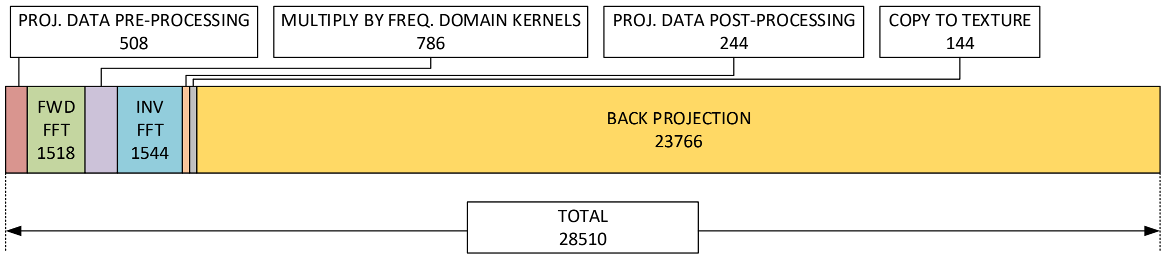

Fig 4 Timeline depiction of the reconstruction pipeline, with timings in microseconds.

the complete reconstruction pipeline, demonstrating a total execution time significantly less than the 33.3ms required for real-time reconstruction. Note that this excludes the time taken for the host-to-device and device-to-host memory copies, since these are asynchronous.

5 Conclusions

We have presented an efficient implementation of an algorithm designed for optimal image quality from stationary gantry CT systems with arbitrary geometry. Implementation challenges caused by the weighting functions and dual back-projection steps were overcome using reduced precision storage with minimal impact on image quality. Our implementation has been demonstrated to give real-time performance on low-cost hardware, potentially increasing the reach of RTT technology.

References

1 E. Morton, K. Mann, A. Berman, *et al.*, "Ultrafast 3D reconstruction for x-ray real-time tomography (RTT)," in *Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE*, 4077–4080 (2009).
2 W. M. Thompson, W. R. B. Lionheart, and D. Öberg, "Reduction of periodic artefacts for a switched-source x-ray CT machine by optimising the source firing pattern," in *Fully 3D 2013*, 345–348 (2013).
3 A. Katsevich, S. Yoon, M. Frenkel, *et al.*, "Reduction of irregular view-sampling artifacts in a stationary gantry CT scanner," in *Fully 3D 2019*, (Submitted 2019).
4 A. Katsevich, K. Taguchi, and A. Zamyatin, "Formulation of four Katsevich algorithms in native geometry," *IEEE Transactions on Medical Imaging* **35**, 855–868 (2006).
5 N. Maass, M. Baer, and M. Kachelrieß, "CT image reconstruction with half precision floating-point values," *Medical Physics* **38**, 656–667 (2011).