PRACTICAL CONSIDERATIONS FOR GPU-ACCELERATED CT

Klaus Mueller and Fang Xu
Center for Visual Computing   Computer Science   Stony Brook University

ABSTRACT

The introduction of programmability into commodity graphics hardware (GPUs) has enabled their use much beyond their native domain of computer graphics, in many areas of high performance computing. We have shown in previous work that many types of CT algorithms, both iterative and non-iterative, can also greatly benefit from the high degree of SIMD (Same Instruction Multiple Data) parallelism these platforms provide. In this paper, we extend this work by describing how one can deal with a number of challenges that frequently arise in practical application settings using the Feldkamp algorithm: large data, angle-dependent projection geometry, and the need for higher accuracy without compromising speed. For this, we combine our fast hardware-native 8-bit interpolation scheme with a higher precision dual-pass mechanism. This latest version of our RapidCT system runs on the most current GPU hardware, nearly eight times faster than the previous version.

1. INTRODUCTION

Computed Tomography has become one of the most popular medical diagnostic modalities since its invention thirty years ago. Various exact or non-exact, analytical or iterative tomographic algorithms have been designed and investigated. Among all methods, the filtered backprojection algorithm devised by Feldkamp, Davis and Kress (FDK) [4] for 3D reconstruction from cone-beam projections still remains the most widely used approach. Here, the most time-consuming part, the backprojection procedure which has a complexity of O(N^4) in the spatial domain constitutes the bottleneck for all software solutions. Custom chips and boards, such as ASICs (Application Specific Integrated Circuits) and FPGAs (Field Programmable Logic Arrays) have become available to overcome these high computational demands and deliver impressive speeds. However, the high cost of these hardware solutions and their inflexibility for reprogramming and generalization limit their application in experimental clinical and research settings.

In recent papers we have described [9]-[11] a framework called RapidCT (soon freely available at http://www.rapidCT.com), which takes advantage of the immense processing power of programmable commodity graphics hardware, produced by NVidia and ATI. It achieves reconstruction speeds of 1-2 orders of magnitude greater than CPU-based applications, both for iterative algorithms, such as SART [1] and OS-EM [5], as well as for analytical approaches, such as FDK [4]. This gap is bound to widen when considering that GPU performance growth has consistently tripled Moore's law. In the first incarnation of our system [9][11], we have focused solely on the floating point processing pipeline since it naturally provides for the best possible accuracy, for a given reconstruction algorithm. In [10] we then discovered that higher speed (nearly 5 times higher) can be obtained by using an integer-based fixed-point implementation, which enables the more costly interpolation operations to be conducted on the much faster hardwired facilities of the boards (in contrast to a program run in the GPU's fragment processor). Despite the reduction in bit-precision in some parts of the pipeline, the reconstruction result is still satisfactory. While not fit for very low-contrast diagnostic reconstructions, a system in this mode would be ideal for rapid CT-guided 3D positioning, biopsies, and other tasks where cone-beam scanning is used for guiding procedures. In this paper, we describe an only slightly slower approach, which produces a quality similar to that of the original floating point approach, but still employs the faster byte-oriented GPU pipelines. In our effort to provide a system that fits more closely the needs of an actual practical application, we also devise techniques that deal with large data in the presence of the restricted GPU memory caches and with un-constant projection geometries, which frequently arise with slightly unstable cone-beam gantries. While there are many more practical issues, such as the handling of noise, scattering, beam hardening, and others, which could be incorporated into the GPU pipeline, here we are only focusing on the issues purely related to the backprojection task, which is a centerpiece of FDK. In this regard, the outline of our paper is as follows: In Sections 2 and 3, we will mention previous relevant work and provide more detail on GPU-based reconstruction. Then, in Section 4, we will describe our new contributions, while Sections 5 will present results and conclusions.

2. PREVIOUS RELATED WORK

Cabral et. al [2] were the first to employ texture mapping hardware for tomographic reconstruction from cone-beam data. Mueller et al. [7] also investigated the application of graphics hardware for cone-beam CT, in conjunction with an iterative method, SART [1]. Finally, Chidlow et al. [3] implemented emission tomography with the OS-EM algorithm [5] on a NVidia GeForce 4 GPU card. Although good speed-ups and quality reconstructions were achieved, the potential of these solutions was limited, due to the hardware's limited precision and capabilities - many operations still had to be performed on the slower CPU, which also required expensive data transfers. To overcome the precision limitations, techniques for virtually extending the limited framebuffer (8 or 12 bits) to higher fixed-point precision were proposed, using a decomposition into the RGBA color channels. Mueller and Yagel used a two-channel approach to virtually extend the precision to 16 bits, while Chidlow et al. later employed a similar approach to split a 16-bit dataword into all four channels. In both applications, the dataword assembly and backprojection accumulations had to be performed on the CPU. The emergence of full GPU programmability and floating point precision enabled GPU-resident backprojection interpolations [11] and accumulations at high precision [10]. So far, none of the previous work has dealt with larger CT projection data, exceeding the size of the resident memory, nor were the performance-limiting issues that arise with large data explored.

3. BACKGROUND

The components of a GPU processing pipeline are illustrated in Fig. 1. The vertices of a polygon enter the *geometry stage* where they are transformed into screenspace. More advanced operations can be programmed in the *vertex shader*. A set of vertices (usually 3 or 4) forms a polygon and once its vertices are transformed into screenspace, the *rasterizer* maps the polygon to the screen by producing an array of *fragments*, one per screen pixel. The *fragment* (or *pixel*) *shader* is then available to perform more advanced operations on these fragments. One of these is texture mapping, where an image (the *texture*), pre-associated with the polygon, is interpolated at a fragment's relative position (by associating each polygon vertex with an image such mapping can be uniquely defined). Since a fragment does not always fall on an image pixel, bilinear texture interpolation is performed. For byte textures this is done in fast hardwired circuitry, while for floating point textures it must be performed in a somewhat slower *fragment program*. We exploited these faster circuits in [10]. Once the fragments have been processed they are composited with the existing framebuffer image, which can also act as a texture in the fragment processing stage.

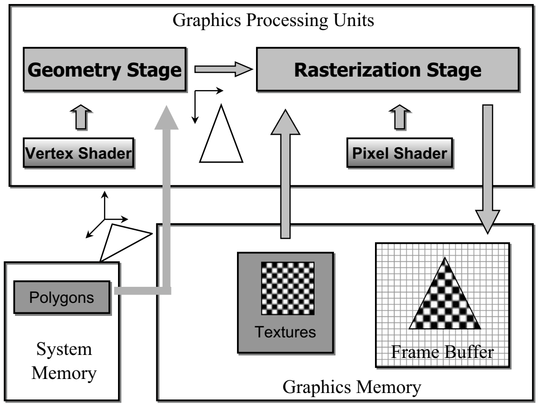


Figure 1: The GPU rendering pipeline.

The main computational effort in FDK stems from the backprojector, written as:

$$v_j = \sum_{p_i \in P_\phi} p_i w_{ij} \qquad (1)$$

where the v_j are the voxels to be reconstructed, the p_i are the pixels in the projection images P_ϕ (acquired at angle ϕ), and the w_{ij} are the weights relating a voxel-pixel pair and determined by the interpolation filter. A filtering of the projections prior to backprojection can be achieved with a suitable GPU algorithm, using [6] for the FFTs and a simple texture multiplication for the filter weighting.

4. METHODS AND IMPLEMENTATIONS

4.1. Original 8-Bit Approach with 4-Channel Processing

One way to implement backprojection is via *projective textures* [8]. An FDK implementation using projective textures requires two volume slice stacks, one for each major direction, *X* and *Z* (*Y* is the axis about which the scanner rotates, see Fig. 2). As described in [11], the hardware's texture matrix is set up in such a way that a backprojection image can be mapped and added to a vertical (*Y*-axis parallel) volume slice using a pixel shader program. The texture interpolations associated with the mapping are done in (hardwired) 8-bit arithmetic [10]. We perform this operation for all images in the set for this specific volume slice. Since the operation is done in 8-bit precision, we cannot accumulate the result directly (it would overflow quickly). Instead, we subdivide the output texture into *M* tiles (*M* being the number of projections in the set) and map each backprojection to an individual tile. We then use a multipass accumulation step in floating point precision to sum the tile textures into the volume slice. Since accumulation is a simple texture streaming step, there is only little overhead incurred by the floating point precision, and we obtained (in [10]) an over 4-fold speedup over the full floating point implementation of [11].

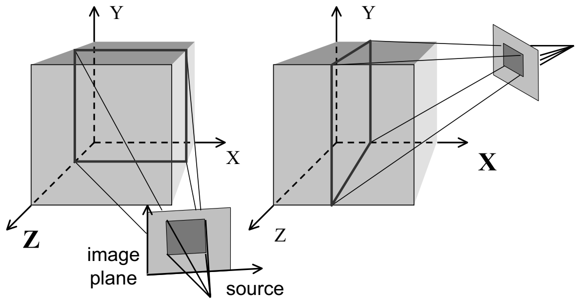


Figure 2: The projective texture approach.

We perform the backprojection in the four color channels RGBA in parallel [11], one projection angle per channel, written to a 4-channel (RBGA) tile. This yields a speedup of nearly 4, with respect to a single channel approach.

4.2. Incorporating Varying Projector Geometries

In practice, cone-beam geometries do not follow a perfect circular trajectory and source-detector axes do not fall within a common plane, therefore each channel must carry its own projection matrix to accomplish texture indexing. We therefore introduce a vertex program which takes the four projection matrices as parameters to perform per-channel projective texture sampling. Here, we should mention that this strategy only works for the (vertical) projective texture-based backprojection described above. It does not work well with the (horizontal) texture spreading approach described in [10], since it is difficult to find the appropriate mapping function between the texture coordinates on the projection images and the sampling locations on the spread area. But as is also discussed there, both approaches run at the same speed for FDK.

4.3. Dual-Pass Approach for Higher Precision

Narrowing down the dynamic range of the filtered CT images from 32 to 8 bits for the backprojection procedure just described can result in a loss of detail. We have used the 3D extension of the Shepp-Logan brain phantom (SLP) to test our algorithms. In this phantom, the contrast of the three small "tumors" near the bottom of the phantom (see Fig. 6) is only 0.5% of the full dynamic range, which is [0.0, 2.0] (note that for display, the dynamic range of the phantom features was stretched to fit into the displayable interval). Fig. 6a shows a slice across the phantom at the original contrast, created with the 8-bit backprojector. We observe some streak artifacts, and the three small tumors are also hard to distinguish. Fig. 6c shows the result obtained with the contrast constraint slightly weakened. For this purpose, we obtained analytical projections of the SLP with the feature contrast doubled, and used those in the reconstruction. The tumors can now be well separated and the streak artifacts are reduced. Thus, the regular algorithm is quite useful for object reconstruction, as long as the contrast requirements are not strict. But we need do better for a diagnostic setting.

Since in our rendering framework, the RGBA channels have been fully occupied, and we do not want to lose the speed gained from the fast 8-bit rasterization, we devised a "double-precision" scheme as follows. For this, we first perform a compression of the dynamic range of the original floating point SLP projection data into 16-bit fixed point words. This is justifiable since data obtained from commercial scanners are usually never wider than 16 bits. Although filtering may result in a higher precision depth, we have not noticed any adverse problems due to this 16-bit compression.

We then render these 16-bit words in two passes. First, every 16-bit word is divided into two bytes and stored in a 2-component texture, containing the higher 8-bit and lower 8-bit words, respectively. Two spreading passes are then performed on each 8-bit texture individually to obtain two tiled sheets, which are streamed into the floating point accumulation pipeline in turn. Here, we should note that the interpolation result obtained solely with the higher 8-bit texture will lose the lower 8 bits, if generated there. This is similar to the scheme presented in [7]. When assembling the two results, an additional shifting operation needs to be executed on the sheet containing the higher 8-bit information before summation (see Fig. 3). Fig. 6b (d) illustrates for the SLP at 0.5% (1%) contrast that this scheme delivers excellent reconstruction results, despite its slightly reduced accuracy compared to a fully 16-bit approach. Also, by design, the total reconstruction time required with the extra spreading pass performed is just slightly over twice as long as that required for the single precision reconstruction.

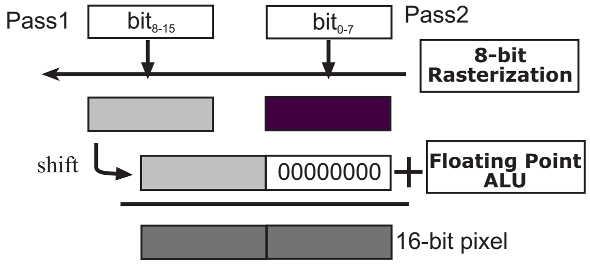


Figure 3: Virtual double precision through data splitting, individual rendering and summation.

4.4. Support for Larger Datasets

All popular PC graphics cards on the market have limited on-board memory (256MB on the NVidia GeForce FX and ATI Radeon series). Given the potentially large magnitude of CT projection images (360 images of size 512^2 or 1024^2) and the resulting volume at a matching resolution, a trivial upscaling of the current implementation to these larger datasets will not scale linearly in performance. In our experiments, we have found that the increase in time spent for the reconstruction volumes of linearly increasing size is not linear (see Fig. 4). In simple cases, the time required for the backprojection onto a slice of a 256^3 volume is much higher than the expected four fold increase of the time incurred for the same operation for a 128^3 volume. A particularly dramatic increase can be observed near a specific dataset size (a volume size of 202^3 in Fig. 4). This can be explained by an excessive number of cache misses, followed by texture reloads.

Texture compression strategies are often adopted to address problems due to insufficient memory space. The existing OpenGL compression extension (ARB_texture_compression) has been designed to boost pipeline efficiency. However, this lossy scheme does not satisfy the strict quality requirements of CT. Pre-compressing the CT images in a lossless way before texture loading, by ways of a user-defined compression scheme, is not practical either, due to the unavailability of corresponding decompression mechanism on the card, although this could be user-implemented.

We therefore adopt a scheme that alleviates the burden imposed by large images and volumes through partitioning and subsequent stitching (assembly) of the results. The idea is to control the scale of computation on the GPU in order to maximize the efficiency of the memory operations, while avoiding superfluous traffic incurred by large data. More concretely, the reconstruction task of a N^3 volume can be decomposed into 8 smaller volumes of size $(N/2)^3$ (see Fig. 1). Reconstruction of each smaller volume still needs the participation of all projection images and will go through the same procedures with regards to projection and accumulation.

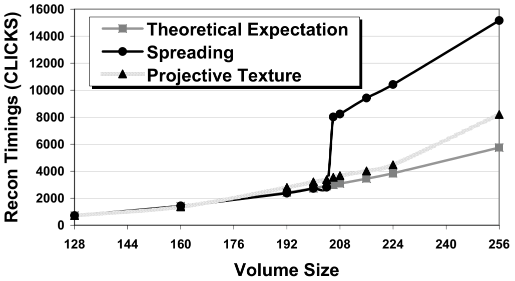


Figure 4: Non-linear timing curve as a function of volume size (the size numbers are to be taken cubed).

But the complexity of all texture-related operations is kept at $O(N^3/8)$ and the required bandwidth is thus bounded. This scheme can be performed recursively until an appropriate size of the partial volume is found. We observed in our experiments on the current platforms that texture sizes of 128 exhibit good cache behavior. Thus, our decomposition with the current GPU platforms uses volume blocks of size 128^3. As a simple example, the top and bottom partial volumes shown in Fig. 1 consist of reconstructions and assemblies from blocks 0-3 and 4-7, respectively, and are exported to the CPU in two transfers, if no subsequent visualization is needed. Else, we compress the volume section in a quick run length encoding (RLE), which compresses away the blank voxels near the boundaries of the volume.

For extra large projections exceeding the physical graphics memory, a straightforward compromise is to divide projection images into subsets, perform the reconstruction set by set and combine the individual results in the end. Another approach is to only reconstruct a certain range of volume slices at a time. This requires the decomposition of the projections into a corresponding set of segments. One can still use the volume block-scheme described above to keep below the cache limits, and for both strategies the projections for the next slice range can be streamed in while the previous reconstruction is still in progress, since loading and GPU calculations can occur simultaneously. Currently, we always decompose our volume into their upper and lower halves, due to the independent calculation of these volumes.

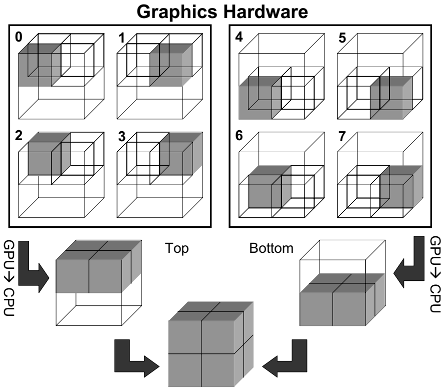


Figure 1: Strategies for texture partitioning and assembly.

5. RESULTS AND CONCLUSIONS

Fig. 6 compares some reconstructions of the SLP at 2 different contrasts (the original 0.5% and a more moderate 1%), obtained with both the single and the dual-pass approach. The line plots show the intensity profile of a 1D cut across the 3 small tumors near the bottom of the 3D SLP. We refer to Section 4.3 for a discussion of the reconstruction results. The tables on the right in Fig. 6 show the performance obtained with the different settings discussed here (for an NVidia 7800 FX GPU), all using the 4-channel parallelism (RGBA) for the reconstruction of a 256^3 volume. We see that the better data caching achieved with the data partitioning (P) allows speedups of 2-3, both at floating and at fixed-point precision. We also observe that while the double-pass approach (DP) does run about 1.5 times slower than the single-pass approach (SP), it is still 14 times faster than the floating point approach, while producing an image that is visually very close to the original. It is also 60 times faster than a pixel-driven CPU floating point implementation. Although there is a small amount of noise that can be detected in the line plot, the level is too small to show up in the image. We should also mention that our partitioning strategy provides the desired linear scaling in dataset size - a reconstruction into a 128^3 volume completes at 0.2s in the SP mode and at 0.4s in DP. Finally, the table on the bottom right in Fig. 6 shows the running speed for reconstructions from projection images increasing in size. Although we did not increase the size of the reconstructed volume, the larger resolution of the projections yields better interpolation results in the reconstructions, which leads to higher reconstruction fidelity, when the object requires it (the SLP can be reconstructed well with the 256^2 projections). We observe that the reconstruction runs at the expected speed as long as all projections fit into memory. Extra large images that exceed the physical onboard memory are divided into subsets small enough to fit, as described in Section 4.4. The reconstruction is then run individually and the results are summed together.

Our results indicate that GPU-based CT has shown convincing potential for clinical practice. Next, we plan to reconstruct a volume from real cone-beam projections, and we also plan to incorporate some of the standard noise models, non-linear rays in iterative methods, and others.

6. ACKNOWLEDGEMENTS

This work was partially funded by NIH grant R21 EB004099-01 and Agard Lab, Department of Biochemistry and Biophysics, University of California at San Francisco.

7. REFERENCES

[1] A. Andersen, A. Kak, “Simultaneous Algebraic Reconstruction Technique (SART): a superior implementation of the ART algorithm,” *Ultrasonic Imaging*, 6:81–94, 1984.

[2] B. Cabral, N. Cam, J. Foran, “Accelerated volume rendering and tomographic reconstruction using texture mapping hardware,” *Proc. Symp. on Volume Visualization 1994*, pp. 91-98.

[3] K. Chidlow, T. Möller, “Rapid emission tomography reconstruction,” *Proc. Volume Graphic Workshop 2003*, pp. 15–26.

[4] L. Feldkamp, L. Davis, and J. Kress, “Practical cone beam algorithm,” *J. Optical Society America*, pp. 612–619, 1984.

[5] H. Hudson and R. Larkin, “Accelerated image reconstruction using ordered subsets of projection data,” *IEEE Trans. of Medical Imaging*, 13:601–609, 1994.

[6] T. Jansen, B. von Rymon-Lipinski, N. Hanssen, and E. Keeve, "Fourier Volume Rendering on the GPU Using a Split-Stream-FFT," *Vision, Modeling, Visualization '04*, pp. 395-403, 2004.

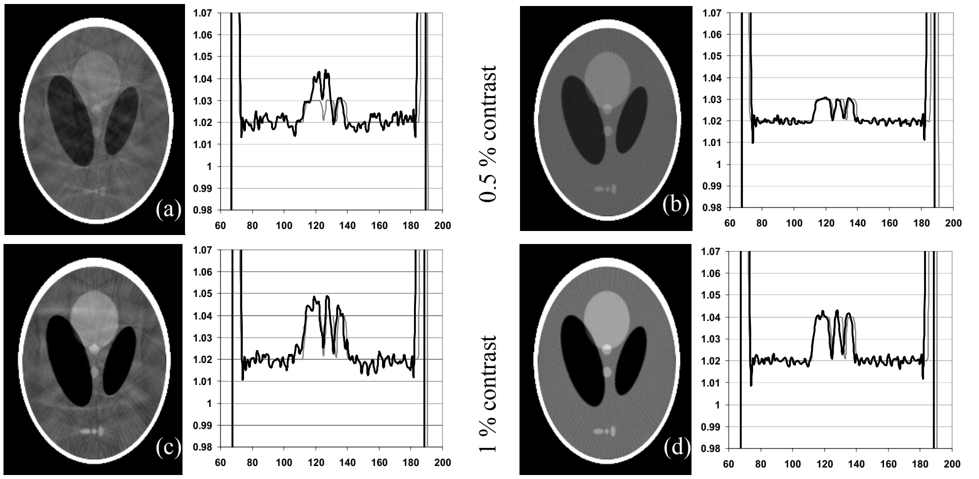
[7] K. Mueller and R. Yagel, “Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2-D texture mapping hardware,” *IEEE Trans. on Medical Imaging*, 19(12):1227–1237, 2000.

[8] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli, “Fast shadows and lighting effects using texture mapping,” *Proc Siggraph ‘92*, pp. 249–252, 1992.

[9] F. Xu and K. Mueller, “A Unified Framework for Rapid 3D Computed Tomography on Commodity GPUs,” *IEEE Medical Imaging Conference* 2003.

[10] F. Xu and K. Mueller, "Ultra-fast filtered backprojection on commodity graphics hardware", *2004 IEEE International Symposium on Biomedical Imaging*, April 2004.

[11] F. Xu and K. Mueller, “Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware,” *IEEE Trans. on Nuclear Science*, 52(3):654-663, 2005.



8-bit precision in interpolation (single-pass)

16-bit precision in interpolation (dual-pass)

Mode	Time
CPU float	180s
RGBA float	42s
RGBA float + P	20s (2)
RGBA SP	6.1s (7)
RGBA SP + P	1.9s (21)
RGBA DP	9.0s (5)
RGBA DP + P	3.0s (14)

Projections	Memory	Time
160 256x256	20 MB	3.0s
160 1024x768	420 MB	4.0s
320 1024x768	540 MB	6.7s

Figure 6: **Left.** A slice of the 3D SLP reconstructed at 256^3 resolution from 160 projections. **Right top.** Timings (speedups in parentheses) with the various conditions discussed in the paper (RGBA: 4-channel approach - sect. 4.1, P: partitioning - sect. 4.4, SP/DP: single/dual pass approach - sect. 4.3) **Right bottom.** Performance with larger input datasets, typically used in clinical practice