Hindawi Publishing Corporation
ISRN Biomedical Engineering
Volume 2013, Article ID 892967, 9 pages
http://dx.doi.org/10.1155/2013/892967

Research Article

High-Fidelity Visualization of Large Medical Datasets on Commodity Hardware

Luigi Gallo and Alessio Pierluigi Placitelli

National Research Council of Italy, Institute for High Performance Computing and Networking (ICAR-CNR), Via Pietro Castellino 111, 80131 Naples, Italy

Correspondence should be addressed to Luigi Gallo; luigi.gallo@na.icar.cnr.it

Received 19 March 2013; Accepted 9 June 2013

Academic Editors: M. Deng and D. Giansanti

Recent advances in CT and MRI static and dynamic scanning techniques have led to great improvements in the resolution and size of volumetric medical datasets, and this trend is still ongoing. However, the explosion of dataset size prevents clinicians from taking advantage of an interactive, high-resolution exploration of volumetric medical data on commodity hardware, due to the memory constraints of modern graphics cards. This paper presents a hybrid CPU-GPU volume ray-casting method and some hybrid-based inspection tools aimed at providing interactive, medical-quality visualization using an ordinary desktop PC. Experimental results show that the hybrid method provides a near-interactive high-fidelity visualization of large medical datasets even if only limited hardware resources are available.

1. Introduction

Over the past few years, direct volume rendering (DVR) has stood out as a powerful means to visualize volumetric medical data. The main benefit of volume rendering techniques is that they allow you to capture all the 3D data in a single 2D image, conveying much more information than indirect rendering techniques [1]. Nonetheless, only recently we have been experiencing a broader adoption of DVR techniques in daily clinical practice. For a long time, insufficient image quality [2] and poor interactivity [3] had been the main barriers to the adoption of volume visualization in the clinic. Medical visualization requires interactive response times in image generation, but until a few years ago interactivity was achievable only if high-end, highly expensive graphics workstations were used.

The turning point was the dramatic increase in the programmability and computational precision of modern graphics processing units (GPUs), which have made it possible to visualize volumetric medical data with a high degree of accuracy and at an interactive frame rate. Vertex and fragment units of the new generation of GPUs are now user programmable, allowing applications to take advantage of their massive many-core architectures to speed up data parallel tasks. Moreover, whereas few years ago graphics hardware supported only 32-bit color and so provided only 8-bit values to store the voxel intensities, now the GPU's pixel depth reaches 128 bits per pixel, which means that each component (red, green, blue, and alpha) has a 32-bit floating point precision throughout the graphics pipeline. As a consequence, in recent years much research activity has been concentrated on the design of GPU-accelerated DVR techniques suitable for the visualization in real time of 3D [4–7] and 4D volumetric medical data [8, 9].

However, great advances have also been made in CT and MRI static and dynamic scanning techniques. Fully isotropic three- and four-dimensional images are nowadays of common use in clinical practice. These developments have led the creation of higher resolution and larger volumetric datasets, presenting challenging issues for GPU-based visualization. In the next few years, datasets used in clinical practice are expected to reach 2048^3 voxels, against the 64^3 produced by the first commercial CT scanner [2].

Whereas modern 64-bit high-performance computing processors allow us to add sufficient main memory to contain entire 3D datasets of any size, today's GPUs are severely

memory constrained. Such memory limitations force users to discard significant portions of collected data or to view large visualizations in small sections. In fact, the 10 GB/s bandwidth of PCIe interfaces, available on most off-the-shelf personal computers, makes it impossible to preserve an interactive visualization by transferring "on demand" portions of the dataset to the graphics memory. Since swapping the data from system to video memory causes a severe performance degradation, the entire dataset has to be entirely contained in the GPU's memory to avoid the bandwidth bottleneck. As a result, interactive GPU-based volume rendering is limited to datasets that fit into the graphics memory.

To meet these challenges, in this paper we present a hybrid DVR method aimed at providing medical-quality rendering at interactive frame rates using commodity hardware. Such a hybrid approach takes advantage of the heterogeneity of the resources available on off-the-shelf computers: the large availability of system memory and the parallel-oriented architecture of modern GPUs.

Furthermore, we introduce three interactive tools based on the hybrid CPU-GPU ray-casting method aimed at easing the exploration of large volumetrical data: a gaze-directed volume rendering tool, which provides images, the maximum resolution of which is set according to the user's gaze direction; an inner structure tool, which allows to interactively inspect data by using two different transfer functions at the same time; and a localized oversampling tool, which allows users to interactively execute oversampling, time-consuming, and antialiasing technique, only on user-specified regions of the volume (See video in Supplementary Material available online at http://dx.doi.org/10.1155/2013/892967).

Both the visualization method and the interactive exploration tools have been implemented and released as open source software to promote their use and evolution [10].

2. Background

To speed up the rendering of large volumetric medical data, a well-established approach is to render different blocks of the volume at different resolutions. As reported in [11], while interpreting the imaging data, radiologists usually follow a model for visual search that postulates a preattentive global analysis followed by fixations and discovery scanning. Therefore, only some structures, which typically occupy a small percentage of the data, are of interest, but their analysis requires contextual information like locations within a specific organ or adjacency to sensitive structures [12]. Therefore, although interesting features are typically localized and appear in a small region of interest (ROI), also the context, which comprises all that is not in the ROI, is equally important for medical inspection. Nonetheless, it is acceptable to display objects in the ROI at full resolution and to omit details for objects outside the ROI. This leads to a reduction in information and thus, potentially, lowers the computation and communication requirements.

Multiresolution volume rendering [13–15] is basically a solution to reduce the computational cost of DVR by rendering the high-priority regions with the highest accuracy and the lower-priority regions with progressively less accuracy and progressively faster. First, a multiresolution data hierarchy composed of multiple spatially partitioned blocks is created. Then, at runtime, various amounts of data at different levels of detail are extracted and used for the rendering as the user navigates through the hierarchy. However, this technique is unsuitable for clinical applications, because the hierarchical data structure is built before the rendering process, making it difficult to interactively change the high resolution region. As a consequence, dynamic multiresolution approaches have recently been proposed to enable the interactive selection of the region of interest without compromising the display speed [9, 16]. The aforementioned multiresolution approaches tackle the performance issue of volume rendering but do not deal with the memory limitations of GPUs.

The most common approach to cope with the memory limitations of GPUs is to downsample the data to reduce its extent. However, this approach is not suitable for medical data visualization, because data downsampling drastically decreases the quality of the image and so produces visualization artifacts or hides relevant information. To reduce the number of samples without compromising image quality, Kraus et al. in [17] have recently proposed a GPU-based volume rendering method with adaptive sampling in all three spatial directions. Such an approach can avoid most of the data transfer between main memory (RAM) and graphics memory (VRAM) but at the cost of additional data decompression by the GPU. Compression-based multiresolution rendering approaches are instead aimed at allowing an interactive walkthrough of large medical datasets on commodity hardware. In [18, 19], input data are first converted into a compressed hierarchical wavelet representation. Then, the wavelet representation is decompressed on the fly and rendered on the GPU.

All the previously mentioned approaches share the concept of foveated volume, namely a nonuniform sampled volume, whose density of samples is highest at a point called the fovea. Whereas an ideal foveated volume has a smooth transition from the highest to the lowest resolution, in practice only a small number of levels are presented [20]. In this work, we present an implementation of a two-level DVR technique. As in [21–23], the approach consists in dividing volume data into a context and a focus region, leaving the user free to change the position and size of the focus region so as to explore thoroughly different structures in volume data. However, in this work, rather than focusing on data compression or downsampling procedures, we concentrate on taking advantage of the heterogeneity of processing resources to allow an interactive and high-fidelity volume rendering of large medical datasets also on commodity hardware.

3. Materials and Methods

3.1. Volume Ray Casting. Volume rendering describes a wide range of techniques for generating images "directly" from three-dimensional scalar data. Among the many DVR techniques, we have chosen ray casting [24] to implement the hybrid CPU-GPU visualization method. Ray casting is the most direct volume-rendering approach [25], and a recent imaging scientist-guided evaluation has confirmed

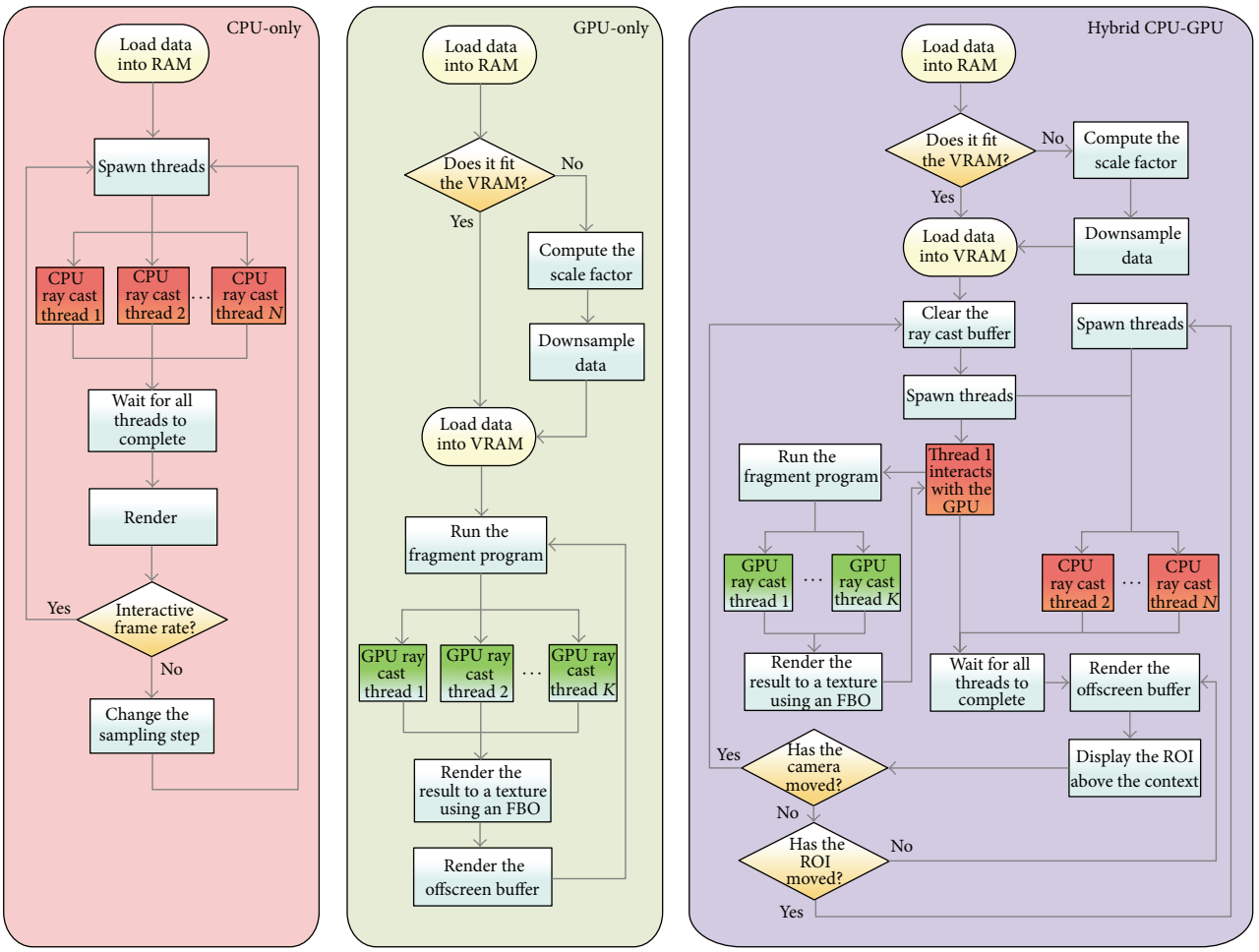


Figure 1: Dataflows that depict memory transfers and code executions of the CPU-only, GPU-only and hybrid CPU-GPU ray casting methods.

that it is the most acceptable technique for the high-fidelity requirements of medical imaging [2]. Basically, ray casting is a process that, for each pixel in the image to render, casts a single ray from the eye through the pixel's center into the volume and integrates the optical properties obtained from the volume densities encountered along the ray. Rays advance a unit vector step by step, and a value is derived by trilinear interpolation from the eight corners of the cell containing the point. Volume rendering via ray casting is an inherently parallelizable task, because the rays are processed independently. Since modern GPUs are able to take advantage of a massive parallelism thanks to their parallel-oriented hardware architecture, many methods have been proposed to provide a high-quality and real-time ray casting executed on a per-fragment level completely on GPUs.

To better describe the hybrid method, we firstly introduce CPU- and GPU-only ray-casting methods specifically focusing on memory transfers. In a typical CPU-only ray-casting pipeline, volume data is first loaded entirely into the RAM, then the software ray-casting method is run. The workload on the CPU is generally balanced by assigning different lines of the image plane to different threads. The result of this elaboration is saved in a buffer which is shared between all the threads (see Figure 1, CPU-only). Each thread T_i runs the software ray-casting method for a line R_k, only if the following expression is true:

$$R_k \bmod (N_T - 1) = T_i, \tag{1}$$

where N_T is the total number of threads. To avoid possible collisions and the need for synchronization primitives, the location of the pixel data inside the shared buffer is determined by considering its position in the viewport. When all the threads terminate, the elapsed rendering time is pushed in a vector. This value is used to adjust the sampling distance in order to achieve an interactive frame rate. Usually, to enhance the interactivity of the image analysis, a high resolution image of the volume is rendered only when the user is not interacting with the scene, whereas just a low-resolution image is provided otherwise.

In GPU-only ray-casting methods, rather than the interactivity, the main issue is the fitting of the dataset into the graphics memory. If the volume data does not fit the video memory, a scale factor is computed. The dataset is uniformly downsampled according to such a scale factor and is then loaded as a 3D texture into the VRAM. The GPU processes the data through a fragment program executed

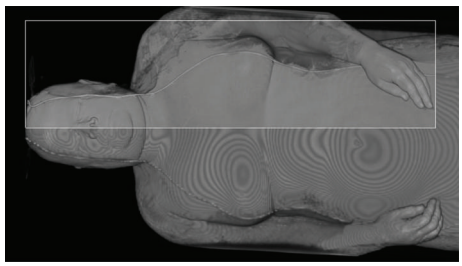


Figure 2: Multiresolution volume visualization with a different level of detail for the ROI (inside the white box) and for the context (outside the white box).

using all the available GPU cores. The result of this parallel elaboration is stored in a 2D texture and then displayed (see Figure 1, GPU-only). Mechanisms similar to those of CPU-only implementations for preserving an interactive frame rate can also be present.

3.2. The Hybrid Volume Ray Casting. The proposed hybrid volume ray-casting method produces a multiresolution visualization of the data, in which only the ROI is always rendered at high resolution, whereas the context is rendered at a variable resolution according to the availability of resources. The dataset is first downsampled in order to fit into the graphics memory. At the same time, a fully detailed version of the dataset is stored into the system main memory. Then, the CPU is used to render at high resolution only the region of interest, whereas the GPU is used to render the context.

The resolution of the context depends on the availability of video memory on the graphics card. If the entire dataset fits inside the VRAM, a full resolution image both of the ROI and of the context is visualized. Figure 2 shows the result of an application of the hybrid volume ray-casting method to render a 1 GB CT dataset on a graphics card equipped with 512 MB of dedicated video memory.

In more detail, the hybrid method starts by computing the scale factor used to downsample all the volumetric data in order to make them fit the VRAM (see Figure 1, Hybrid CPU-GPU). To compute this value, both the maximum texture size allowed on the graphics card and the available dedicated graphics memory are analyzed. Next, the volume is downsampled by scaling it uniformly along the three axes. Then, the downsampled data is loaded as a 3D texture into the graphics memory to provide a fast and direct access from the GPU cores. Therefore, at this time, the volume is both present in the RAM and, in a downsampled version, in the VRAM.

Each time a viewport update is requested, the most recent view matrix is compared with the previous one. If a change in the position or orientation of the camera is detected, the shared buffer which holds the result of the CPU-based ray-casting of the context is cleared. Then, the context part of the image is recomputed by using the GPU. When no changes in camera data are detected, the shared buffer is not cleared, and only the ROI is recomputed by the CPU. By acting in this way, when the ROI is resized or moved by the user, just the high-resolution region of the image is recomputed. As a consequence, the user can progressively extend the high-resolution part of the image in an interactive way.

To compute the final image, a thread is launched for each available CPU core. If the context needs to be recomputed, one CPU thread is used to interact with the graphics hardware in order to control and handle the results of the ray-casting elaboration. Such a CPU thread instructs the GPU to execute the fragment program that implements the ray-casting method and then stores the results in a 2D texture. The remaining CPU threads run the ray-casting method in order to draw the high-resolution area. In accordance with the CPU-only pipeline, we balance the workload on the CPU by assigning different lines of the image plane to different threads (see (1)). Only those casting rays whose starting points are inside the ROI are assigned to CPU threads. The color of each pixel is finally written into a buffer which is shared by all the threads.

Let p be a pointer to the first memory location of the aforementioned buffer. For each processed pixel located at (x, y) we access the corresponding data structure inside the buffer following this indexation:

$$i = p \cdot \left[y \cdot \text{pixels Per Row} \cdot \text{size of } (i) + x \cdot \text{size of } (i) \right]. \quad (2)$$

As a result, different threads write to different memory locations. The method waits for all the threads to terminate and then draws the context information from the GPU to the viewport. The buffer containing the high-resolution result coming from the CPU-based ray casting is then rendered above the context.

3.3. The Inspection Tools. The novel hybrid visualization method allows us to implement advanced medical image exploration tools that are commonly used only on high-end graphics workstations. In the following, three of these inspection tools for 3D medical image exploration are introduced.

3.3.1. Gaze-Based Volume Rendering. The gaze-based volume rendering approach [26, 27] consists in letting a high-resolution sweet spot follow the user's gaze to achieve a trade-off between the computation resource and the represented details. In more details, the eye tracker is used to collect information so as to calculate where to place the region of interest. The gaze-based approach is aimed at keeping at high resolution just the part of the image which is currently under inspection (fovea), so allowing users to benefit from both high resolution and interactive frame rate.

In our implementation, the gaze tracking is smoothed, as described in [28, 29]. Moreover, to preserve an interactive frame rate, the system automatically reduces the ROI size so as to keep the frame rate at a user defined level. In this way, both the position and the size of the region of interest are entirely controlled by the application, leaving the user free to simultaneously manipulate (e.g., rotate, zoom, etc.) the volume.

3.3.2. Virtual Lens. The virtual lens tools provide a highly interactive simultaneous visualization of interior and exterior structures (see Figure 3). As in the Virtual Magic Lantern

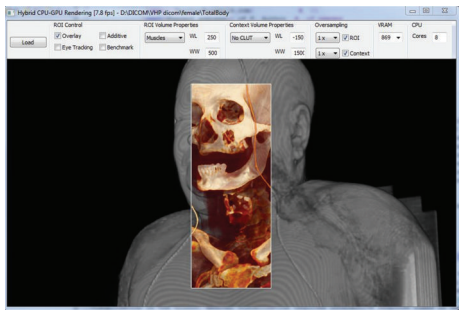


Figure 3: The virtual lens tool: ROI and context are visualized with different transfer functions and color lookup tables. ROI position and size can be controlled interactively by the viewer.

described in [30], the virtual lens allows the focal region to be visualized using a secondary transfer function and/or a different color lookup table.

Real-time modifications of the transfer function and of the color lookup table are featured as well (and applied simultaneously to the ROI, generated by the CPU, and to the context, generated from the GPU).

3.3.3. Localized Oversampling. Real-world objects have continuous, smooth curves and lines, whereas monitors can only display discrete points of light. This causes a problem known as aliasing. *Oversampling* is one of the ways of reducing the aliasing problem. This antialiasing approach consists in casting additional rays and then averaging the computed values. Samples are taken at several instances inside the pixel (and not just at the center). In concrete terms, instead of sampling one ray per pixel, 4/16/64 rays are sampled per pixel. Moreover, the number of sample points on each ray can vary too. In fact, to better simulate the continuous integration of the light effects in a discrete space, a high sampling rate can be used, together with opacity correction techniques to avoid over-composited values.

Figure 4 shows the enhancements in image quality due to the application of oversampling techniques to volume ray casting. In particular, the stripes pattern is not visible even at high zoom levels. However, the rendering time considerably increases if oversampled volume ray casting is used, since the computational load increases proportionally to the number of rays casted through the volume and to the number of sample points on each ray. The localized oversampling tool takes advantage of the hybrid rendering method to apply oversampling only on a region of interest. In this way, the total rendering time is reduced so as to preserve the interactive inspection of medical images.

4. Results and Discussion

4.1. Experimental Testbed. The hybrid CPU-GPU volume ray-casting method has been implemented entirely in the C++ language as an extension of the open-source visualization toolkit (VTK) library [31]. Then, the CPU-, GPU-, and hybrid-based ray-casting methods have been integrated into a DICOM viewer developed entirely in C++ by using the open-source libraries VTK, for the 3D rendering and visualization, and wxWidgets [32], for the user interface. The source code of both the extension of the VTK library and the experimental testbed application is released as open source software and is available for download [10]. The tools have been implemented by extending the Medical Imaging Toolkit (MITO) open-source software [33].

To evaluate the hybrid CPU-GPU ray-casting method, we compared the rendering times obtained by using the hybrid CPU-GPU with the CPU-only and GPU-only ray-casting methods as implemented in the VTK library. The performance measures were computed by averaging the rendering times obtained in twelve trials. For the hybrid technique, in each trial the ROI size was increased gradually from 0% to 100% of the rendering window.

Two sets of human 16-bit CT data in DICOM format were used in our evaluation: Cenovix, a renal angio-CT composed of 361 slices, and visible human female (VHF), a complete, anatomically detailed, three-dimensional representation of the female human body from the National Library of Medicine of the USA [34] composed of 1734 slices. The resolution of the slices was, for both datasets, 512×512 pixels. In all the experiments, the two datasets were rasterized with the same transfer function. Specifically, window level and width settings were set to 50 and 350, respectively.

The sampling distance of the casting rays was set to the diagonal of the voxels that compose the dataset. Specifically, the sampling distance was set to 0.973 and to 1.73 for the Cenovix and the VHF dataset, respectively. Rendering times vary drastically with rendering window size, viewing direction, and distance of the camera from the volume. In all the experiments, we rendered the volumetric data in two windows, the size of which was 720×380 pixels and 1440×760 pixels, respectively. We used axial as the viewing direction and performed a dolly in of the camera so as to let the rendered image fill the entire window.

The performance measures have been gathered on three machines.

Mobile (MO). A notebook equipped with a single CPU Intel Mobile Core 2 Duo P8600 at 2.40 GHz, 4 GB RAM DDR2 at 400 MHz, a graphics card NVIDIA GeForce 9500 M GT (32 stream processors, 256 MB dedicated video memory GDDR2, and 128-bit memory interface), and an OS Windows 7 32 bits.

Consumer Level (CL). A desktop PC equipped with a single CPU Intel Core i7 860 at 2.80 GHz, 8 GB RAM DDR3 at 667 MHz, a graphics card Nvidia Quadro FX 580 (32 stream processors, 512 MB dedicated video memory GDDR3, 128-bit memory interface), and an OS Windows 7 64 bits.

High End (HE). A graphical workstation equipped with two CPUs Intel Xeon W5590 at 3.33 GHz, 24 GB RAM DDR3 at 667 MHz, a graphics card Nvidia Quadro FX 5800 (240 stream processors, 4 GB dedicated video memory GDDR3, 512 bit memory interface), and an OS Windows XP x64 edition.

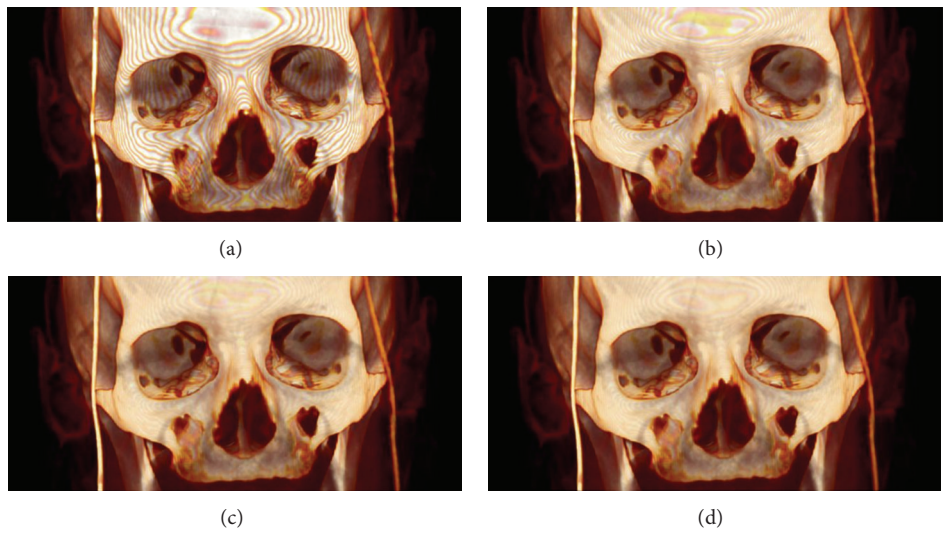
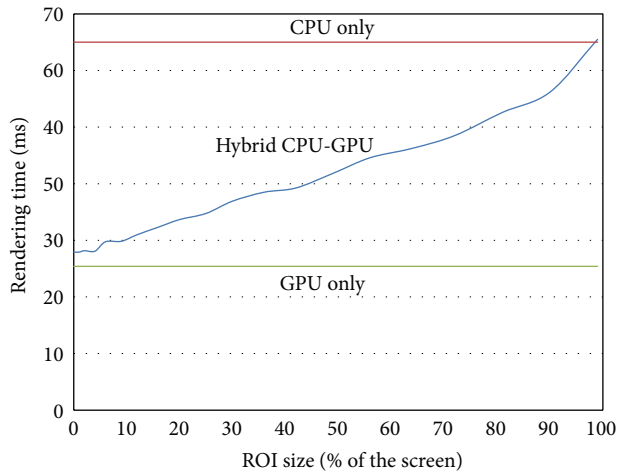
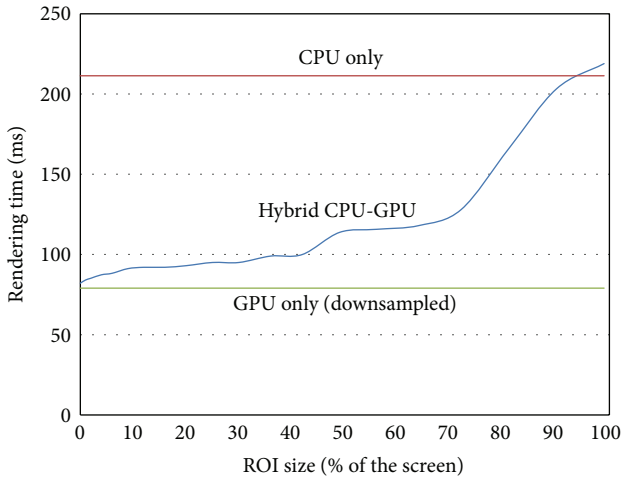


Figure 4: Oversampled volume ray casting (a) not used, (b) low, (c) medium, and (d) high.



(a) Cenovix dataset

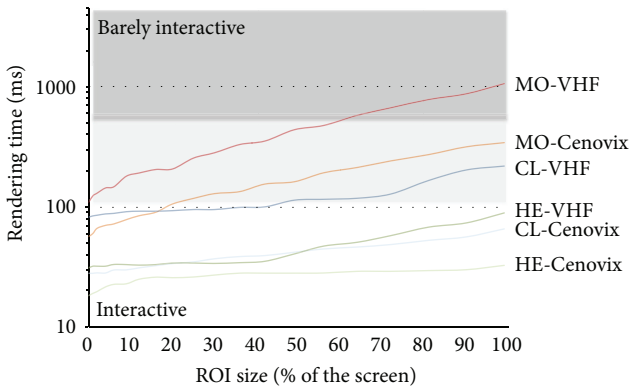


(b) VHF dataset

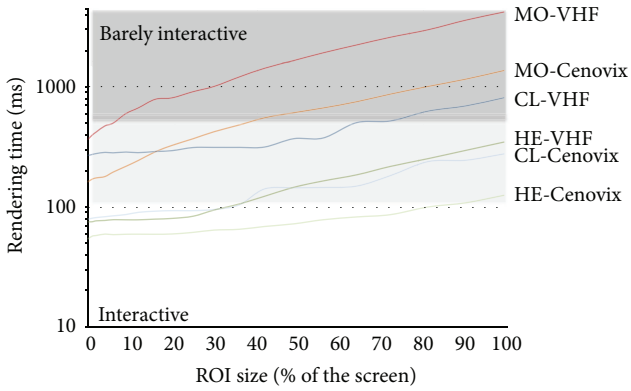
Figure 5: Rendering times of GPU-, CPU-, and hybrid-based ray-casting methods on the consumer-level hardware.

4.2. Experimental Results and Discussion. On the mobile machine, both the datasets must be heavily downsampled to fit the 256 MB of video memory available. As a consequence, the GPU-only volume rendering is not suitable for a medical quality exploration of the data. As the ROI size increases, the CPU-only method shows lower rendering times than the hybrid one, providing an average frame rate of 6.2 frames per second (fps) with a rendering window size of 720×380. This is due to the fact that the notebook used was equipped with only two CPU cores. Whereas the CPU-only method exploits both these cores in the rendering computation, the hybrid method uses one core for the rendering of the high-resolution area and the other one to control the rendering of the context performed by the GPU. Notwithstanding this, the results show that the hybrid method allows a much more interactive visualization (fps $\geq$ 10) of the Cenovix dataset for a high-resolution area smaller than 20% of the screen. Considering the larger VHF dataset, the hybrid method provides an fps $\geq$ 5 for high-resolution areas covering up to the 20% of the screen, whereas the CPU-only method provides an fps $\simeq$ 2.2.

When using the consumer-level hardware, the hybrid CPU-GPU method allows us to keep an interactive visualization (fps $\geq$ 10) of the Cenovix dataset independently of the ROI size (see Figure 5(a)). Moreover, it allows a highly-interactive visualization (fps $\geq$ 25) as long as the ROI size is smaller than 45% of the screen. This result was to be expected since, in these experimental conditions, the CPU-only rendering time also allows an interactive frame-rate (fps $\simeq$ 15). For an ROI larger than 99% of the screen, the CPU-only method produces rendering times slightly lower than the ones produced by the hybrid one due to the overhead introduced into the latter to handle both the CPU and the GPU rendering processes. When switching to the large VHF dataset, the hybrid CPU-GPU method still allows us to keep an interactive frame rate until the ROI size covers up to 40% of the screen (see Figure 5(b)). It is worth noting that the GPU-only rendering method, even if it allows an interactive



(a) Rendering window size 720 × 380



(b) Rendering window size 1440 × 760

FIGURE 6: Comparison of the hybrid volume ray-casting rendering times with different sizes of the rendering window.

visualization of data (fps ≃ 12), requires a downsampling of the dataset in order to fit the available dedicated video memory (512 MB nominally, 487 MB actually available), so it cannot be used for medical quality inspection of data. For an ROI size larger than 40% of the screen, the hybrid method provides a near-interactive visualization ($4 < \text{fps} < 10$).

On the high-end machine, thanks to the sixteen CPU cores available (eight physical and eight logical due to hyperthreading), all the volume ray-casting methods allow us to keep an interactive visualization of both the Cenovix and the VHF datasets. However, also in this case, the hybrid method produces lower rendering times than the CPU-only method as long as the high-resolution area is smaller than 98% of the screen. Since the high-end hardware we used is equipped with sufficient VRAM (4 GB) to avoid the data downsampling step of the ray-casting pipeline, the GPU-only method can provide a high-fidelity volume rendering with the highest frame rate (fps ≃ 34 with the VHF dataset).

Finally, Figure 6 shows a comparison of average rendering times obtained with the hybrid CPU-GPU method with the mobile, consumer-level, and high-end machine with a rendering window size of 720 × 380 and 1440 × 760. In this figure, rendering times are reproduced in a logarithmic scale with base 10. Basically, the slope of the hybrid method curve is mainly given by the CPU contribution to the ray-casting computation, whereas the y-intercept is mainly given by the GPU contribution. In fact, only the GPU works when the ROI size is zero, whereas the CPU rendering time becomes progressively greater than the GPU one as the ROI size increases. The performance gap between the different machines becomes more evident with the larger dataset. With the mobile machine, the VHF dataset can be visualized only at a near-interactive frame rate (fps ≥ 2), with an ROI size smaller than 58% of the 720 × 380 rendering window (see Figure 6(a)) and smaller than 6% of the screen of the 1440 × 760 rendering window (see Figure 6(b)). With a consumer-level machine, instead, an interactive visualization of the VHF dataset is possible with the small rendering window if the ROI size is smaller than 45% of the screen. The high-end machine, in the worst combination of dataset and rendering window, allows an interactive visualization until the region of interest is lower than 38% of the screen.

Figure 6 also shows that the hybrid method scales almost linearly with the total number of pixels to render. In more detail, doubling the sides of the rendering window (and so quadrupling the number of pixels to render), we have observed that rendering times are almost quadrupled. The results also show that, whereas an increase in the number of CPU cores produces a linear increase in rendering performance (58% of average reduction between the 8 logical CPU cores at 2.80 GHz of the consumer level machine and the 16 at 3.33 GHz of the high-end machine), an increase in the number of GPU cores produces a gain that varies with the size of the dataset and especially with the number of pixels to render. In fact, the reduction of rendering times obtained with the 240 GPU cores of the HE machine compared to the 32 GPU cores of the CL machine varies from 40%, obtained with the Cenovix dataset and a window size of 720 × 380, to 68%, obtained with the VHF dataset and a quadrupled window size. The latter result confirms that, when fragment programs are used, the more there are pixels to render and so CPU threads simultaneously in flight, the more the overall efficiency increases due to a better hiding of memory access latency.

5. Conclusions

The explosion of medical dataset size prevents clinicians from taking advantage of an interactive, high-resolution GPU-based 3D visualization on commodity hardware. To provide an interactive visualization, the entire dataset has to be entirely contained in the graphics memory, but today's GPUs are severely memory constrained. As a result, interactive volume rendering on commodity hardware is limited to datasets that fit into the GPU's memory.

In this paper, a 3D volume visualization method has been proposed to visualize interactively large medical datasets on machines with limited hardware resources. The rendering method is based on a hybrid approach that takes advantage of the heterogeneity of the resources available on off-the-shelf computers: the large availability of system memory and the parallel-oriented architecture of graphical processing units. Specifically, the CPU is used to render a fully detailed image

of a user-controlled region of interest, whereas the GPU is used to render the remaining part of the image by using a downsampled version of the dataset.

Furthermore, three inspection tools based on the hybrid rendering method have been introduced: (i) a gaze-directed volume rendering tool, which provides images the maximum resolution of which is set according to the user's gaze direction; (ii) an inner structure tool, which allows to inspect volumetric data by using two different transfer functions at the same time; and (iii) a localized oversampling tool, which allows users to interactively enhance the image quality only on a region of interest.

The experimental results indicate that the hybrid rendering method and the inspection tools can be profitably used for the interactive exploration of large medical dataset on machines on which, due to limited hardware resources, CPU- and GPU-based rendering methods cannot provide an interactive, high-fidelity visualization. Future work will focus on designing alternative ways to divide the workload between CPU and GPU and on testing the approach also for gradient-shaded rendering. The visualization method and the inspection tools have been implemented and issued as open source software, which may promote their use and evolution.

Disclosure

The Cenovix dataset is from http://www.dicom7.com/. The VHF dataset is from https://mri.radiology.uiowa.edu/VHDicom.

References

[1] K. Mueller and A. E. Kaufman, "Volume visualization in medicine," in *Handbook of Medical Image Processing and Analysis*, I. N. Bankman, Ed., chapter 46, pp. 785–816, Academic Press, San Diego, CA, USA, 2nd edition, 2008.

[2] M. Smelyanskiy, D. Holmes, J. Chhugani et al., "Mapping high-fidelity volume rendering for medical imaging to CPU, GPU and many-core architectures," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1563–1570, 2009.

[3] L. Gallo, "A glove-based interface for 3D medical image visualization," in *Intelligent Interactive Multimedia Systems and Services: Proceedings of the 4th International Conference on Intelligent Interactive Multimedia Systems (Smart Innovation, Systems and Technologies)*, vol. 6, pp. 221–230, Springer, Berlin, Germany, 2010.

[4] S. X. Qin, X. B. Geng, and Y. S. Jiang, "Automatic dynamic task distribution between CPU and GPU for VR systems," *Applied Mechanics and Materials*, vol. 157-158, pp. 1324–1330, 2012.

[5] T.-H. Lee, J. Lee, H. Lee, H. Kye, Y. G. Shin, and S. H. Kim, "Fast perspective volume ray casting method using GPU-based acceleration techniques for translucency rendering in 3D endoluminal CT colonography," *Computers in Biology and Medicine*, vol. 39, no. 8, pp. 657–666, 2009.

[6] K. Xie, G. Sun, J. Yang, and Y. M. Zhu, "Interactive volume cutting of medical data," *Computers in Biology and Medicine*, vol. 37, no. 8, pp. 1155–1159, 2007.

[7] D. Levin, U. Aladl, G. Germano, and P. Slomka, "Techniques for efficient, real-time, 3D visualization of multi-modality cardiac data using consumer graphics hardware," *Computerized Medical Imaging and Graphics*, vol. 29, no. 6, pp. 463–475, 2005.

[8] M. Howison, E. W. Bethel, and H. Childs, "Hybrid parallelism for volume rendering on large-, multi-, and many-core systems," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 1, pp. 17–29, 2012.

[9] Q. Zhang, R. Eagleson, and T. M. Peters, "Dynamic real-time 4D cardiac MDCT image display using GPU-accelerated volume rendering," *Computerized Medical Imaging and Graphics*, vol. 33, no. 6, pp. 461–476, 2009.

[10] iHealth lab, "Hybrid CPU-GPU volume ray-casting," 2013, http://ihealthlab.icar.cnr.it.

[11] S. Piccand, R. Noumeir, and E. Paquette, "Region of interest and multiresolution for volume rendering," *IEEE Transactions on Information Technology in Biomedicine*, vol. 12, no. 5, pp. 561–568, 2008.

[12] J. E. van der Heyden, K. M. Inkpen, M. S. Atkins, and M. S. T. Carpendale, "Exploring presentation methods for tomographic medical image viewing," *Artificial Intelligence in Medicine*, vol. 22, no. 2, pp. 89–109, 2001.

[13] R. Westermann, "A multiresolution framework for volume rendering," in *Proceedings of the symposium on Volume visualization (VVS '94)*, pp. 51–58, ACM, New York, NY, USA, 1994.

[14] E. C. la Mar, B. Hamann, and K. I. Joy, "Multiresolution techniques for interactive texture-based volume visualization," in *Proceedings of the 10th IEEE Visualization Conference (VIS '99)*, pp. 355–361, IEEE Computer Society, Los Alamitos, CA, USA, October 1999.

[15] S. Guthe, M. Wand, J. Gonser, and W. Strasser, "Interactive rendering of large volume data sets," in *Proceedings of the 13th IEEE Visualization Conference (VIS '02)*, pp. 53–60, IEEE Computer Society, Los Alamitos, CA, USA, November 2002.

[16] C. Wang, A. Garcia, and H.-W. Shen, "Interactive level-of-detail selection using image-based quality metric for large volume visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 1, pp. 122–134, 2007.

[17] M. Kraus, M. Strengert, T. Klein, and T. Ertl, "Adaptive sampling in three dimensions for volume rendering on GPUs," in *Proceedings of the 6th Asia-Pacific Symposium on Visualisation (APVIS '07)*, pp. 113–120, Los Alamitos, CA, USA, February 2007.

[18] S. Guthe and W. Strasser, "Advanced techniques for high-quality multi-resolution volume rendering," *Computers and Graphics*, vol. 28, no. 1, pp. 51–58, 2004.

[19] K. Xie, J. Yang, and Y. M. Zhu, "Real-time rendering of 3D medical data sets," *Future Generation Computer Systems*, vol. 21, no. 4, pp. 573–581, 2005.

[20] A. Corcoran, N. Redmond, and J. Dingliana, "Perceptual enhancement of two-level volume rendering," *Computers and Graphics*, vol. 34, no. 4, pp. 388–397, 2010.

[21] J. Zhou, M. Hinz, and K. Tonnies, "Focal region-guided feature-based volume rendering," in *Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission (3DPTV '02)*, pp. 87–90, IEEE Computer Society, Los Alamitos, CA, USA, 2002.

[22] J. Zhou, C. Xiao, Z. Wang, and M. Takatsuka, "A concept of volume rendering guided search process to analyze medical data set," *Computerized Medical Imaging and Graphics*, vol. 32, no. 2, pp. 140–149, 2008.

[23] Y.-S. Wang, C. Wang, T.-Y. Lee, and K.-L. Ma, "Feature-preserving volume data reduction and focus+context visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 2, pp. 171–181, 2011.

[24] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29–37, 1987.

[25] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *Computer Graphics*, vol. 22, no. 4, pp. 65–74, 1988.

[26] M. Levoy and R. Whitaker, "Gaze-directed volume rendering," *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 2, pp. 217–223, 1990.

[27] A. Lu, R. Maciejewski, and D. S. Ebert, "Volume composition and evaluation using eye-tracking data," *ACM Transactions on Applied Perception*, vol. 7, no. 1, pp. 1–20, 2010.

[28] L. Gallo, M. Ciampi, and A. Minutolo, "Smoothed pointing: a user-friendly technique for precision enhanced remote pointing," in *Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '10)*, pp. 712–717, February 2010.

[29] L. Gallo and A. Minutolo, "Design and comparative evaluation of smoothed pointing: a velocity-oriented remote pointing enhancement technique," *International Journal of Human Computer Studies*, vol. 70, no. 4, pp. 287–300, 2012.

[30] E. Monclús, J. Díaz, I. Navazo, and P.-P. Vázquez, "The virtual magic lantern: an interaction metaphor for enhanced medical data inspection," in *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology (VRST '09)*, pp. 119–122, ACM, November 2009.

[31] W. Schroeder, K. Martin, and B. Lorensen, *Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*, Kitware, Clifton Park, NY, USA, 4th edition, 2006.

[32] wxWidgets, "Cross-platform GUI library," 2010, http://www.wxwidgets.org.

[33] L. Gallo, G. de Pietro, A. Coronato, and I. Marra, "Toward a natural interface to virtual medical imaging environments," in *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '08)*, pp. 429–432, May 2008.

[34] M. J. Ackerman, "Accessing the visible human project," Tech. Rep., Corporation for National Research Initiatives, 1995.