journal homepage: www.intl.elsevierhealth.com/journals/cmpb

Visualization and GPU-accelerated simulation of medical ultrasound from CT images

Oliver Kutter[a,*], *Ramtin Shams*[b], *Nassir Navab*[a]

[a] *Computer Aided Medical Procedures (CAMP), Technische Universität München, Boltzmannstr. 3, 85748 Garching bei München, Germany*
[b] *Research School of Information Sciences and Engineering (RSISE), The Australian National University, Canberra*

ARTICLE INFO

Article history:
Received 21 August 2008
Received in revised form
18 December 2008
Accepted 19 December 2008

Keywords:
Ultrasound
Simulation
Real-time
GPU
Visualization

ABSTRACT

We present a fast GPU-based method for simulation of ultrasound images from volumetric CT scans and their visualization. The method uses a ray-based model of the ultrasound to generate view-dependent ultrasonic effects such as occlusions, large-scale reflections and attenuation combined with speckle patterns derived from pre-processing the CT image using a wave-based model of ultrasound propagation in soft tissue. The main applications of the method are ultrasound training and registration of ultrasound and CT images.

1. Introduction

1.1. *Ultrasound simulation*

Ultrasound as an imaging modality is desirable from many perspectives: (a) it is real-time with a high temporal resolution, (b) it is risk-free (radiation-free and non-hazardous), (c) the ultrasound devices are relatively cheap, (d) ultrasound devices are portable and relatively small, and (e) ultrasound probes can be used to target small tissue interfaces in endoscopic, laparoscopic and intravascular applications. The main drawback, however, is the quality of the acquired images and a low signal to noise ratio (SNR), which makes navigation and interpretation of the acquired images, particularly challenging. Ultrasound simulation systems have been shown to improve the performance and skills of users, significantly (e.g. see [1]). This is due to the fact that the trainees can practice localization and acquisition of ultrasound without the time-constraints imposed by such practice on the patients and can also access a variety of cases which have been collected and stored in the simulation system's database over time. Other studies by Knudson and Sisley [2] and Terkamp et al. [3] found use of an ultrasound simulator compared favorably with traditional hands-on training on patients. Knudson also reports simulator training to be a convenient and objective method of introducing ultrasound to surgery residents.

A number of systems including commercial products are available for ultrasound training (e.g. [3–11]). These systems allow navigation with a virtual probe within the space of pre-recorded ultrasound images. The acquisition protocol is typically 3D freehand ultrasound with an off-line compounding stage where 2D ultrasound images are combined to create a 3D volume (some directly acquire 3D ultrasound). At runtime, during training sessions, the position and orientation of the virtual probe is tracked and the relevant ultrasound planes are re-sliced from the previously computed volumes.

* *Corresponding author*. Tel.: +49 89 289 19403; fax: +49 89 289 17059.
E-mail address: kutter@in.tum.de (O. Kutter).
0169-2607/$ – see front matter © 2009 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.cmpb.2008.12.011

UltraSim [4] is a commercially available ultrasound training simulator by MEDSIM. Aiger and Cohen-Or [5] describe the system in detail. A dummy ultrasound probe, with an integrated tracking sensor, is moved over a mannequin while simulated ultrasound images are generated in real-time by re-slicing a pre-computed large volumetric ultrasound data-set from pre-acquired freehand 3D ultrasound for the selected virtual scan geometry.

Ehricke [6] presents SONOSim3D, a simulator for sonography training and education. Simulated 2D ultrasound images are generated by re-slicing a pre-acquired 3D ultrasound data-set from an extendable database of cases. A digital patient model with 3D image volumes (CT and MR data-sets and pathology information from the Visible Human project) is used to help trainees with anatomical interpretation.

Henry et al. [7] describe an ultrasound imaging simulation for diagnosis of deep venous thromboses. A pre-recorded ultrasound volume in combination with a deformable model of the anatomy is used to generate the simulated ultrasound images. UltraTrainer by Stallkamp and Walper [8] replaces the probe with a magnetic tracker and uses 3D data from a real examination registered with a phantom. Weidenbach et al. [9] present an augmented reality based simulator for training in echocardiography. Pre-recorded freehand 3D echocardiography data is registered to a surface model of the human heart. During a training session the simulated ultrasound data can be displayed together with the heart model to provide anatomical context to the trainee. Heer et al. [10] evaluate the use of a training system for gynecological sonography. Re-slicing of a pre-acquired 3D freehand ultrasound data-set is directly used for simulation of 2D ultrasound image. Tahmasebi et al. [11] present a generic framework for ultrasound simulation. Simulated ultrasound images are generated by re-slicing a pre-acquired 3D ultrasound data-set. The authors incorporate real-time haptic feedback in conjunction with a deformable model of the data-set to provide a realistic scanning experience. Furthermore simultaneous display of simulated ultrasound images and co-registered CT or MRI data is supported. The framework features a flexible design that can be extended to other medical simulators.

Fully synthetic simulation of ultrasound has been proposed by Jensen et al. [12–14] based on an acoustic wave-propagation model and using the concept of spatial impulse response [15,16] which is implemented in a program called Field II [13]. The program can be used to simulate any linear ultrasound system with single or multi-element transducers, any given apodization, focusing, pulse excitation scheme and aperture geometry [17]. The program requires location and strength of scatterers as input and gives best results with carefully designed and synthetically generated scattering patterns. As such, the program is mostly used to determine the effects of various parameters on transducer design. Additionally, the simulations for even a single B-mode image take an extremely long time and need to be parallelized (the execution time for a B-mode image with 128 RF scan lines and 1,000,000 point scatterers is in the order of 2 days on a single CPU), which makes it impractical for real-time simulation and in training applications.

More recently, simulation of ultrasound from CT volumes has attracted interest. Authors in [18] briefly discuss simulation of ultrasound from CT volumes without providing much detail on their ultrasound modeling. In [19], the authors discuss a system for ultrasound guided needle insertion and use CT images for patient specific training. Our team has investigated ultrasound simulation using a simple ray-based modeling of ultrasound for registration purposes in [20], which uses a simple ultrasound simulation. In [21], we proposed an enhanced modeling of ultrasound, which results in realistic ultrasound simulations from CT images suitable for ultrasound training (see Fig. 1(c)). In this paper, we present a framework for ray-based simulation of ultrasound on the GPU and a visualization software. The framework accommodates ultrasound simulations with varying degrees of complexity and is fast enough to produce interactive simulation and visualization for training purposes.

One common problem with traditional ultrasound simulation systems (e.g. [3–11]) is that the simulation is realistic as long as the operator remains within close vicinity of originally acquired positions and orientations. As the probe is navigated further away from the acquisition positions, the images become less realistic, since view-dependent ultrasound effects are no longer accurately represented. The acquisition protocol is also complicated and requires the volume of interest to be imaged from various positions and not

Fig. 1 – (a) Schematic of a convex array transducer with a multi-element active aperture; (b) ultrasound field of view superimposed on the liver of a human subject; (c) our simulated ultrasound (combined reflection and scattering images), for the region of interest shown in (b). Also notice shadowing on the right-hand side due to an air–tissue interface and in the middle–bottom of the image due to a a bone–tissue interface.

to contain view-dependent artifacts such as shadowing, and the effect of a fixed gain and focus. Then there is, of course, the issue of compounding the images and accumulated errors due to mis-registration and accumulation of intensity values with varying intensities due to view-dependant artifacts.

Use of CT images as the basis for simulations not only avoids the aforementioned drawbacks but also has the advantage of allowing for patient specific simulations, ease of navigation for novice users as they can practice ultrasound navigation with the help of corresponding CT information (this extra assistance is obviously turned off at later stages of training). It also provides easier access to raw data for simulation, as CT images are routinely acquired for diagnostic and planning and the acquisition protocol is uncomplicated and streamlined.

1.2. General purpose programming on the GPU

General purpose programming on the GPU (GPGPU) has been around for more than a decade. Up until recently, GPGPU programs had to be implemented in programmable vertex and fragment shaders of the graphics pipeline mostly in an OpenGL or Direct3D environment using computer graphics shading languages, e.g. *OpenGL's built-in Shading Language* (GLSL), *Microsofts High Level Shading Language* (HLSL), or *NVIDIA's C for Graphics* (Cg). The implementation of a non-graphics algorithm is therefore not straightforward as it has to be reformulated to be executed by the graphics pipeline. Additionally one has to circumvent the limitations imposed by the computer graphics environment, e.g. no random access writes (scatter writes) to the global GPU memory.

Since late 2006, however, NVIDIA and ATI have introduced *Compute Unified Device Architecture* (CUDA) and *Close to Metal* (CTM), application programming interfaces (API) on their latest generation of graphics cards, respectively. The technology allows programming of the GPU in a C-like fashion and offers access to the GPU hardware as a streaming multi-processor architecture. Some limitations for general purpose computations on the GPU hardware are addressed by these APIs, e.g. scatter writes to the global memory are possible.Hence these approaches provide more flexibility. However, parallelization and setting up an optimal execution configuration is left to the programmer.

Despite the benefits offered by CUDA and CTM, we chose an OpenGL/GLSL GPGPU environment for our GPU-based ultrasound simulation. As we will show, the ultrasound simulation can be formulated as a ray casting problem. Thus, we can benefit from the availability of an established framework for direct volume rendering and GPU-based image processing in OpenGL/GLSL. Use of OpenGL/GLSL also ensures that our implementation is relatively independent of the graphics hardware and can be run on a wider range of devices. The only requirement is a graphics card that supports *Shader Model* 3.0 and OpenGL 2.1.

2. Method

2.1. A ray-based model for ultrasound

When an ultrasound beam travels through a piecewise homogenous medium, it gets partially reflected at the interface between two media with differing *acoustic impedances*. The change in acoustic impedance is the main physical interaction that makes ultrasound visualization possible [22]. The amount of energy which is reflected is determined by the *reflection coefficient*, α_R, given by

$$\alpha_R = \left(\frac{Z_2 - Z_1}{Z_2 + Z_1}\right)^2, \tag{1}$$

where Z_1 and Z_2 are the acoustic impedances of the media in question. The acoustic impedance itself depends on the speed of sound in a medium and the density and is given by $Z = \rho c$. The remaining energy that passes through the interface is characterized by the transmission coefficient $\alpha_T = 1 - \alpha_R$.

The ultrasound simulation from CT images is based on the premise that there is an approximately linear relationship between CT Hounsfield values and the acoustic impedance for soft-tissue [20]. We perform automatic segmentation of bone and air interfaces in order to calculate the reflection coefficient for air-tissue and bone-tissue interfaces where CT values cannot be directly used in (1). The reflection of ultrasound at tissue interfaces is non-specular and subject to scattering. We use a Lambertian scattering model where the intensity of the scattered signal depends on the incidence angle and can be written as

$$R(\mathbf{x}) = \alpha_R(\mathbf{x}) I_i(\mathbf{x}) |\mathbf{r}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x})|, \tag{2}$$

where $I_i(\cdot)$ is the intensity of the incident beam at the interface, $\mathbf{r}$ is the unit vector in the direction of the beam, $\mathbf{n}$ is the surface normal, $|\cdot|$ is the absolute value operator, and $R(\cdot)$ is the intensity of the reflected signal. According to a Lambertian scattering model, the intensity of the signal, as perceived by an arbitrary viewer, is independent of the viewing angle and only depends on the angle of incidence. If we show the initial intensity of the ultrasound by I_0, and the incident intensity at spatial location $\mathbf{x}$ by $I_i(\mathbf{x})$, the accumulative attenuation at point $\mathbf{x}$ will be given by $I_i(\mathbf{x})/I_0$. The reflected signal travels back through the same attenuating medium (ignoring any refraction), and as such the intensity of the signal as sensed by the receiver, $I_r(\mathbf{x})$, is attenuated by the same coefficient as in the forward path and can be written as

$$I_r(\mathbf{x}) \propto \alpha_R(\mathbf{x}) \frac{I_i^2(\mathbf{x})}{I_0} |\mathbf{r}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x})|. \tag{3}$$

The effect of a finite beam width produced by multiple transducer elements is modeled by integrating the perceived intensities along the active wavefront at a specified depth using a suitable window function [21]. For a linear array transducer we can write

$$I_r(x, y) \propto \int_{x-\ell}^{x+\ell} \alpha_R(u, y) \frac{I_i^2(u, y)}{I_0} |\mathbf{r}(u, y) \cdot \mathbf{n}(u, y)| \omega(u)\, du, \tag{4}$$

where $\omega(\cdot)$ is the window function, and ℓ is the length of the active aperture, given by $\ell = n_a(w_e + s_e)$, where n_a is the number of active elements, w_e is the width of each element, and s_e is the spacing between adjacent elements. A square, triangular, or a Hanning window can be used for apodization

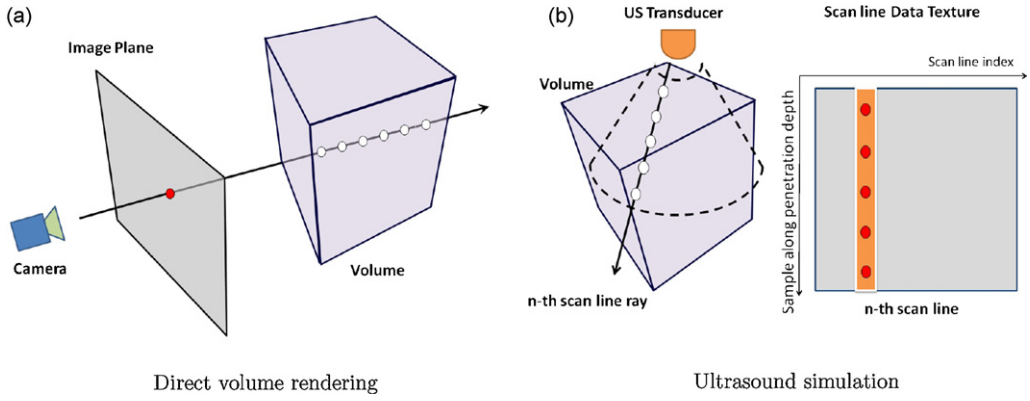

Fig. 2 – Difference of ray casting algorithms for (a) direct volume rendering and (b) ultrasound simulation: (a) for DVR, multiple samples along a ray require a single storage (red dot in the image plane); (b) for ultrasound simulation, every sample along a ray requires a corresponding storage. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

depending on how the transducer elements are being activated.

2.2. *GPU-accelerated ultrasound simulation*

We formulate the ultrasound simulation problem on the GPU as a ray casting problem. The (virtual) ultrasound transducer is positioned within the space of the CT volume. For every transducer element, and depending on the geometry of the probe (i.e. linear or curvilinear), an ultrasound beam is cast and multiple rays are processed in parallel by the GPU. For each sample along a ray, Eq. (3) is computed inside a *fragment shader*. The results are stored as a measure of the acoustic intensity received by a transducer element from a point at a given depth in the anatomy along an ultrasound beam and displayed as an image.

The algorithm is implemented in C++, OpenGL, and GLSL. The OpenGL FramebufferObject (FBO) Extension[1] is employed to efficiently render to off-screen render targets (i.e. 2D and 3D textures in GPU memory).

A key difference of our ray casting algorithm for ultrasound simulation compared to traditional ray casting algorithms (e.g. for direct volume rendering (DVR) [23]) is the need to store sample values along each ray. In a standard ray casting algorithm, based on the light propagation model, the output for each ray is a single value which is the result of combining the color and intensity contribution of each sampled element along a ray (see Fig. 2(a)). As shown in Fig. 2(b), for ultrasound, an acoustic echo is returned from each sample along a ray and needs to be stored separately. For a high quality simulation, we need 256 or more samples along each ray. This largely exceeds the number of output channels per fragment. As such, we use a multi-pass algorithm for efficient implementation of ultrasound ray casting on the GPU.

Various data structures are allocated and loaded during the initialization stage and an optimal memory layout is determined. CT data, ultrasound ray start and end vectors are stored in textures. Ray start and end vectors are used to compute position of samples within the CT volume at each pass of the algorithm. Ray start and end vectors have to be re-initialized, every time that the user changes the orientation or position of the probe.

The ray casting algorithm is designed to be independent of the probe geometry and ultrasound dimensions. Scan line information is stored in 2D textures for both 2D and 3D ultrasound images. This is a major benefit and allows us to use the same algorithm for simulation of 2D, 3D, linear, curvilinear and freehand ultrasound. The original dimension and shape of the ultrasound image are restored in the scan conversion stage.

We need three render targets for storage of intermediate results and acoustic intensities. This is to store $I_i(\mathbf{x})$ and $I_r(\mathbf{x})$ (refer to Section 2.1). $I_i(\cdot)$ is calculated recursively

$$I_i(\mathbf{x}) = I_i(\mathbf{x} - \Delta\mathbf{d})(1 - \alpha_R(\mathbf{x} - \Delta\mathbf{d})), \tag{5}$$

where $\Delta\mathbf{d}$ is the incremental sampling vector along a given ray. Storage of $I_i(\cdot)$ scan line data requires two textures to avoid read/write conflicts and synchronization issues. The algorithms interleaves data read/writes for even/odd rows of the scan lines (ping-pong rendering). This is to ensure that all fragment shaders finish writing into row k, before starting row $k+1$ which requires values of the previous row.

A practical consideration in allocating textures is the memory layout. GPUs typically have an upper-bound for the width and height of the textures. Regardless of the available memory, one cannot allocate a texture that exceeds the limit in one or multiple dimensions. Performance-wise, GPUs typically perform better with square textures whose dimensions are a power of 2. We need a texture of size $n \times d$ for simulating an ultrasound with n transducer elements and d samples along each ray. This is not a problem for 2D ultrasound as the number of scan lines hardly exceeds 256. However, for 3D ultrasound the number of elements and as a result scan lines can easily exceed the limit. Therefore, the memory layout is optimized to

[1] http://www.opengl.org/registry/specs/EXT/framebuffer_object.txt.

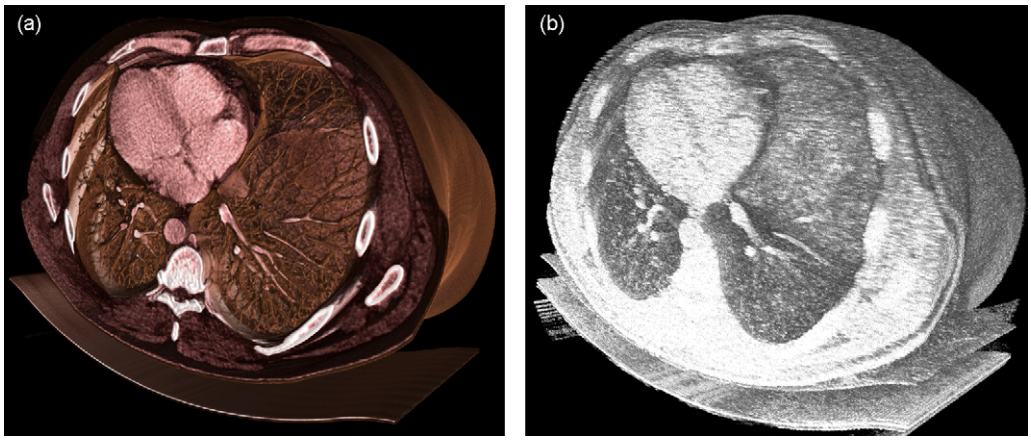

Fig. 3 – (a) Volume rendering of the CT image; (b) volume rendering of the corresponding scattering image.

be close to square and several tiles of scan lines are arranged within the texture, as needed.

2.3. *Creating the ultrasound image*

Using the model presented in Section 2.1, we generate an image called the reflection image from CT data. The reflection image simulates view-dependent ultrasonic effects due to reflection and attenuation of the signal. Tissue boundaries are emphasized in the image and shadows due to large impedance mismatches between tissue–bone and tissue–air interfaces are simulated.

We also generate a scattering image using Field II by preprocessing a CT volume from a fixed view-point as described in [21]. Fig. 3 shows a volume rendered CT image of the abdomen with the corresponding scattering image.

The scattering artifacts in Ultrasound are view-dependent. In a real ultrasound, the speckle pattern which results from the scattering looks to be elongated in a direction perpendicular to the ultrasound beam. We have modeled this in (4) for the reflection image. Applying the same principle to the scattering image has the effect of elongating the speckle and improves visual quality of the ultrasound simulation and somewhat compensates for the fixed-view computation of the scattering image. Also note that the scattering image is multiplied by the respective transmission coefficients in (6). This prevents formation of speckle in areas of the image where a shadow is expected and the intensity of the incident signal is low.

The reflection and scattering images are combined using the following formula:

$$I_{us}(\mathbf{x}) = (G_{\sigma_1}(\mathbf{x}) * I_r(\mathbf{x}) + \alpha G_{\sigma_2}(\mathbf{x}) * \alpha_T(\mathbf{x}))I_s(\mathbf{x}), \tag{6}$$

where $I_{us}(\cdot)$ is the ultrasound image, $I_r(\cdot)$ is the reflection image, $I_s(\cdot)$ is the elongated scattering image, α is a blending coefficient, and G is a Gaussian filter with 0 mean and adjustable standard deviation (σ_1 and σ_2) used to smooth the output of the image fusion process. The blending parameters, α, σ_1 and σ_2 are adjusted by the operator for best viewing results.

The resulting image has a large dynamic range which far exceeds the dynamic range of the display and range of intensities that can be identified by the human eye. To reduce the dynamic range, we compress the signal using the following log-compression method

$$I_c(\mathbf{x}) = \begin{cases} 0, & I_{us}(\mathbf{x}) < \max\{I_{us}(\mathbf{x})\}10^{-\beta/10}, \\ 10\log_{10}^{I_{us}(\mathbf{x})/\max\{I_{us}(\mathbf{x})\}} + \beta, & \text{otherwise}, \end{cases} \tag{7}$$

where β is the dynamic range of the compressed signal.

2.4. *The simulation pipeline*

Our simulation pipeline as shown in Fig. 4 consists of five stages: the scan line traversal, pre-scan conversion, scan conversion, post-scan conversion and compositing stages.

The pre-scan conversion, post-scan conversion and compositing stages are optional and are executed if required by the underlying ultrasound model (e.g. the model in [20] utilizes the scan line and scan conversion stages only, while the more complex in [21] invokes all stages of the pipeline).

- *Scan line traversal stage*: As the first stage of the pipeline, the 3D data-set is sampled along each scan line and the values are stored in a 2D texture. Each time probe-related parameters are varied by the user, scan line data has to be recomputed. For simulation of an ultrasound image with d samples (pixels) along each beam, the algorithm requires exactly d render passes. For simulation of 2D ultrasound, typically a single line primitive is executed at each pass. However, for 3D ultrasound or simulation of multiple 2D ultrasound images, where scan lines are tiled within the texture memory, we run m parallel line primitives, where m is the number of tiles. Running multiple line primitives typically provides a better utilization of the GPU resources. This means that our algorithm reaches its full capacity (throughput) for larger simulations (i.e. multiple 2D and 3D ultrasound).
  The ultrasound simulation may require a re-mapping of the CT values so that they can be directly used in Eq. (1). A transfer function lookup texture is used for efficient re-mapping of CT values.
- *Pre-scan conversion stage*: For efficient computation, Eq. (4) can be reformulated as the convolution of the scan line data

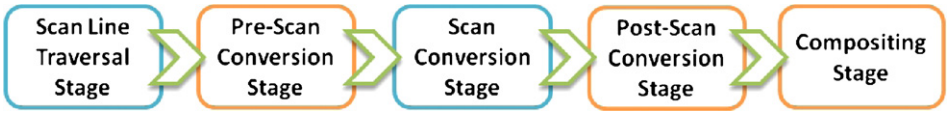

Fig. 4 – Various stages of the simulation pipeline on the GPU. The scan line and scan conversion stages are always executed. Other stages (orange boxes) are optional and only executed if required by the simulation model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

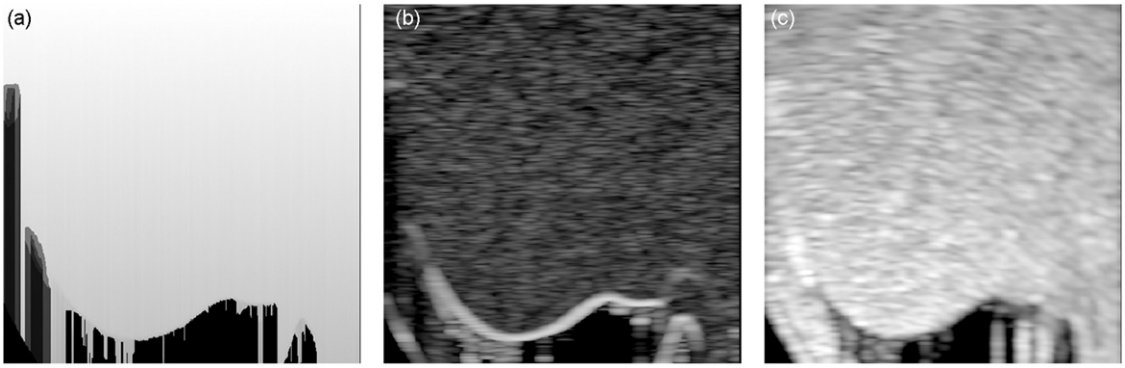

Fig. 5 – Intermediate results from pre-scan conversion stage: (a) transmission coefficients scan line image, no filtering applied (transmission texture); (b) reflection scan line image, Hanning window applied (reflection texture); (c) scattering scan line image, Hanning window applied (scattering texture).

with an appropriate 1D window function. This computation is performed in the pre-scan conversion stage. Fig. 5 shows the content of various textures resulting from pre-scan conversion stage for a sample ultrasound simulation of the liver.

- *Scan conversion stage*: This stage is used to convert scan line data into a 2D or 3D Cartesian representation. Scan conversion is implemented by backward warping on the GPU using a specialized fragment shader for each probe geometry and dimension. We use the GPUs built-in bilinear interpolation for maximum performance.
- *Post-scan conversion stage*: 2D and 3D simulated images may have to be filtered for improved visual quality according to (6). This requires convolution with the appropriate 2D or 3D filter which is implemented by a fragment shader on the GPU. Separable kernels are used in conjunction with two/three render passes for 2D/3D filtering, where possible, to improve the performance. Fig. 6 shows the content of various textures following the post-scan conversion stage for a sample ultrasound simulation of the liver.
- *Compositing stage*: In the compositing stage, intermediate results from various sources are combined in a fragment shader according to (6) and (7), which computes the final value for each pixel and prepares the data for visualization.

2.5. *Real-time visualization*

A key component of our real-time visualization is the concurrent display of the simulated ultrasound images within the CT data-set using GPU-Accelerated direct volume rendering (see Fig. 7). We use the emission-absorption model of light propagation through a translucent volume [24], which is based on the assumption that the volume is filled with light emitting particles and ignores scattering of light. For a ray of light traveling along a direction, parameterized by a variable

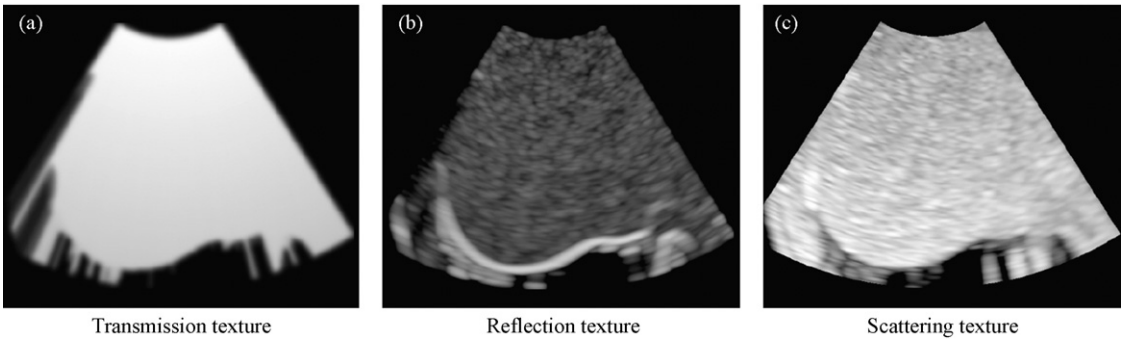

Fig. 6 – Resulting Cartesian images after scan conversion and post-scan conversion stages: (a) smoothed transmission coefficients (transmission texture); (b) smoothed reflection image (reflection texture); (c) scattering image (scattering texture).

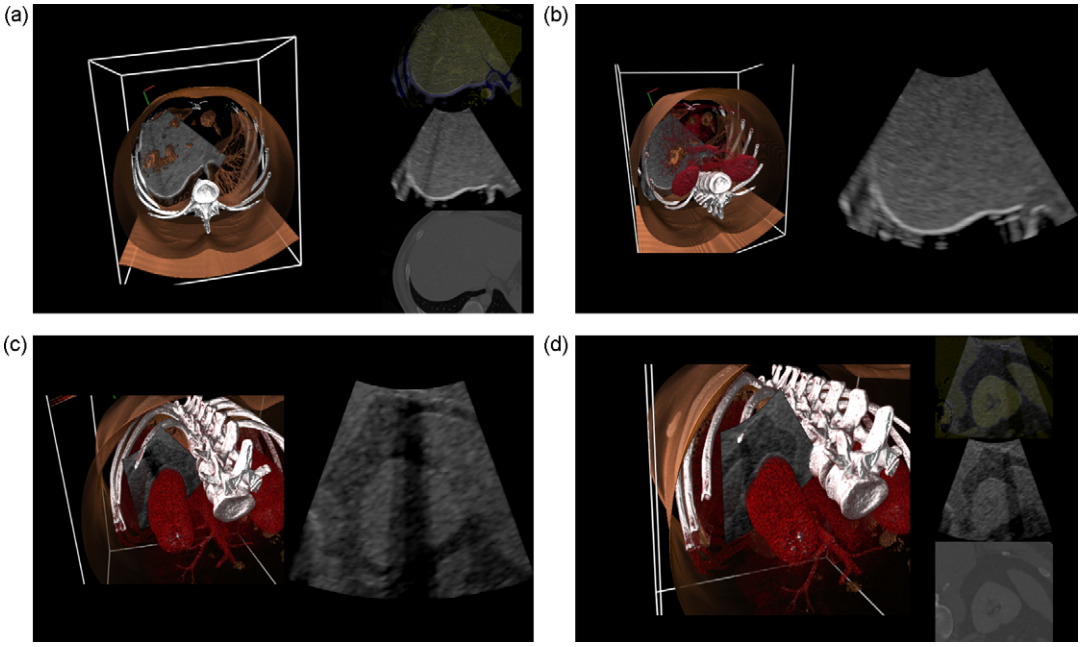

Fig. 7 – Real-time direct visualization of a CT volume and simulated 2D ultrasound. (a, b) Simulation with a wide angle curvilinear transducer scanning the liver–lung boundary. (c, d) Simulation with a narrow angle curvilinear transducer scanning the left kidney. Note the occlusion artifacts due rays intersecting the ribs.

such as s, the volume rendering integral can be written as

$$I(s) = I(s_0)e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(\tilde{s})e^{-\tau(\tilde{s},s)}d\tilde{s}, \tag{8}$$

with

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(d)ds, \tag{9}$$

where $I(\cdot)$ is the intensity of light at a given point along the ray, s_0 denotes the point where the beam enters the volume, and q and τ are emission and absorption coefficients, respectively. The first term in (8) describes the background light attenuation by the volume and the second term accounts for the contribution of emitting particles along the ray while taking the distance dependent attenuation of light into account.

In practice, a numerical approximation of the analytical volume rendering integral is used compute the integral iteratively while traversing the volume, either in a front-to-back or back-to-front fashion along the viewing direction using alpha blending. For a detailed treatment of the subject the reader is referred to [23].

In recent years, GPU-accelerated ray casting has emerged as the de facto standard for high quality real-time direct volume rendering [23,25–27]. The algorithm owes its popularity to its easy and straightforward implementation on modern GPUs compared to other volume rendering techniques such as texture slicing [23]. Furthermore, the algorithm lends itself to optimization well and is highly adaptable for various visualization tasks.

Despite an exponential improvement in computing capability of GPUs in recent years, volume rendering of 3D medical images remains a computationally expensive task. Various techniques need to be employed in order to achieve real-time high quality rendering. We briefly describe the methods, we employed for achieving interactive frame rates.

- *Deferred rendering*: Our renderer is implemented by a multistage rendering pipeline. To improve the overall performance each stage of the pipeline is updated only if parameters affecting the stage itself are changed or any previous stage is updated. Costly operations in the shaders are deferred to the latest possible phase or avoided completely if their contribution is negligible for the final image.
- *Early termination*: We use front-to-back compositing along the viewing ray when computing the volume rendering integral which allows us to terminate the computations if the accumulated opacity is saturated.
- *Volume culling*: With a typical transfer function, a large number of voxels in a volume are fully transparent, and thus do not contribute to the finally rendered image. To improve the performance, no computations should be performed for these voxels. We employ an octree to partition the voxels of the volume into cells. Each cell is initialized with the intensity range of the enclosed voxels. For a given transfer function non-contributing cells are culled when computing the CT volume ray entry and exit positions. Thus, the ray casting algorithm only renders the volume enclosed by the active octree cells and skips the invisible areas in front and behind the visible volume.
- *Sampling frequency*: The sampling frequency along a ray, is the basic parameter that provides a trade-off between the computational performance versus quality. The quality

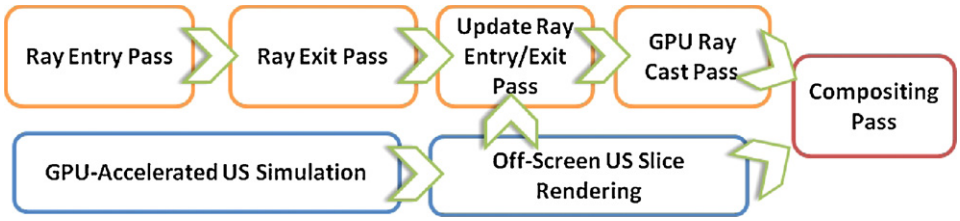

Fig. 8 – Visualization pipeline.

cannot be improved arbitrarily by increasing the sampling frequency. As such, we use the lowest sampling frequency, beyond which quality improvement is not noticeable. One way to achieve a higher quality at a lower sampling rate is to avoid using a regular sampling grid. A low sampling rate with equidistance samples exhibits visually displeasing grid artifacts also known as rings. Using a random offset, on an otherwise equidistance sampling, removes this artifact and allows the sampling rate to be reduced, without loss of quality, in the interest of improved performance.

- *Classification*: The appearance and visibility of voxels is computed in the classification stage of the volume rendering

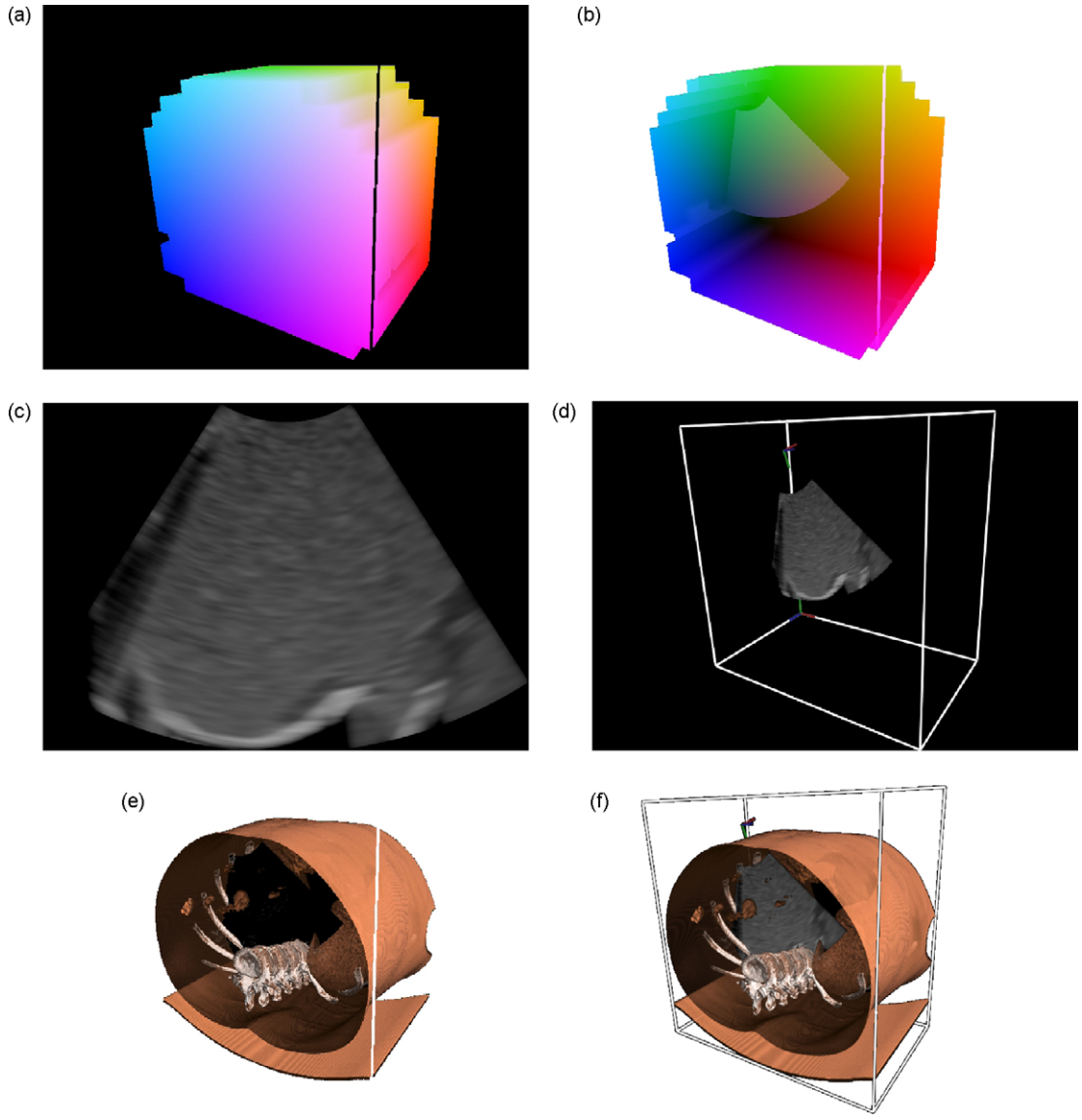

Fig. 9 – Different stages of the visualization pipeline. (a) Ray start and (b) ray end coordinates stored in RGB textures. Note ray stop positions for pixels on the ultrasound image plane. (c) Simulated 2D ultrasound image. (d) Rendering of the texture-mapped ultrasound image in 3D. (e) Direct volume rendering of the CT image. (f) Composition of CT DVR and simulated ultrasound.

algorithm. Classification is implemented by texture lookups in transfer function lookup textures. Traditional 1D transfer function tables require a high sampling frequency of the volume data due to high frequencies introduced by the transfer function, e.g. in semi-transparent rendering of tissue interfaces. To deal with this problem and to reduce the volume sampling frequency pre-integrated [28] and post-color attenuated [29] classification techniques have been introduced. Both approaches pre-compute the volume integral between each two sample values and thus allow a reduction of the sampling frequency while maintaining a high quality. Pre-integrated transfer functions give the best visual results, however the lookup table update is computationally expensive, thus for interactive classification we use post-color attenuated transfer functions.

2.6. *Visual consistency of ultrasound rendering*

The simulated ultrasound image is rendered as an opaque plane/volume within the CT DVR space. Missing or incorrect interaction of an opaque geometry, e.g. ultrasound image plane, with the volume-rendered CT data disturbs the visual perception of the anatomy and its depth. To correctly integrate the simulated ultrasound within the CT volume, we adjust the CT volume entry and exit positions of each ray in the corresponding textures prior to the ray casting pass. For every valid ray entry position in normalized CT volume coordinates a fragment shader checks whether it is occluded by the ultrasound plane by comparing the ray entry position and the ultrasound plane depth values. If the ultrasound plane is closer to the viewer than the ray volume entry position it is cleared with zero, so that no ray is cast. For the CT ray exit positions a similar test is performed. If the ultrasound plane depth value is smaller than the ray exit position, the ultrasound plane depth value is back-projected to the normalized CT volume coordinates and replaces the CT volume ray exit position. Thereby, rays are correctly stopped at the first opaque geometry surface in the viewing direction. This is demonstrated in Fig. 9(b).

2.7. *Visualization pipeline*

The described techniques are utilized in the rendering pipeline for displaying the simulated ultrasound images and the volume rendered CT image. Fig. 8 and Fig. 9 depict various stages of the rendering pipeline and a sample visualization of the pipeline, respectively. In the ray entry and ray exit passes (Fig. 9(a) and (b)), the front and back faces of the volume's bounding geometry are rendered into two textures that store the ray entry and exit positions in normalized volume coordinates at each texel. In the next pass, the ultrasound field of view is rendered in the same 3D space as the volume. The ultrasound field of view is texture mapped with the result of the ultrasound simulation (Fig. 9(c) and (d)). The depth image from this pass is used to update the ray entry and exit positions stored in the corresponding textures prior to the ray casting pass (Fig. 9(e)). The rendering results of the ray casting pass and the ultrasound plane are composited in the final rendering pass using alpha blending to yield the final image (see Fig. 9(f)).

3. User interface

We have developed an application for the GPU-accelerated ultrasound simulation to display the simulated ultrasound

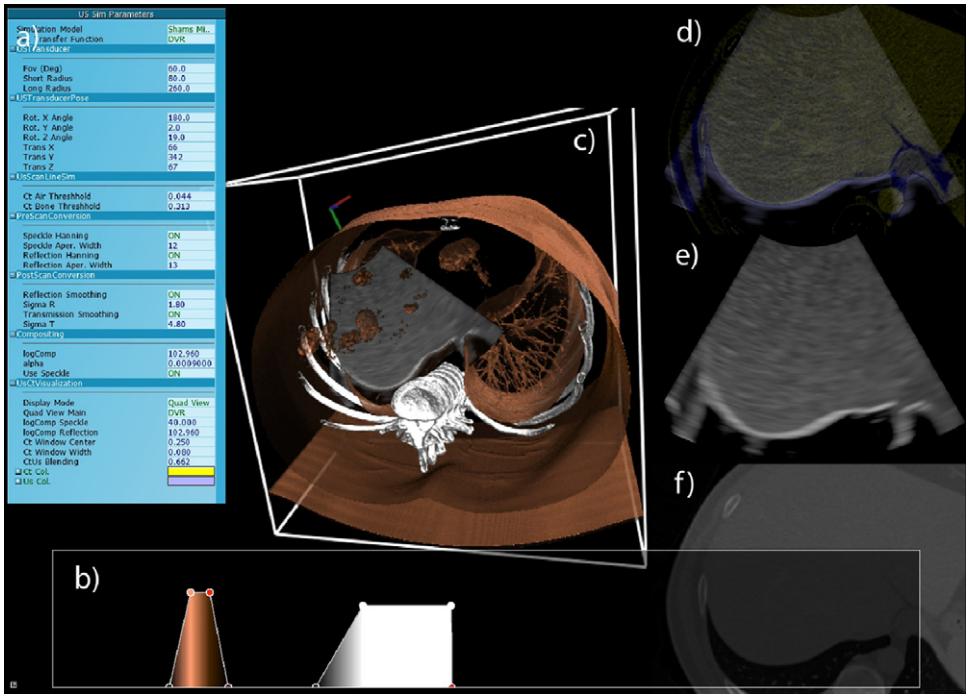

Fig. 10 – Screen-shot of the application: (a) application options, (b) transfer function widget, (c) 3D view, depicting shaded DVR of the CT data and the 2D ultrasound image, (d) blending of the simulated 2D ultrasound image and corresponding CT MPR, (e) simulated 2D ultrasound image, and (f) CT MPR corresponding with the ultrasound image plane.

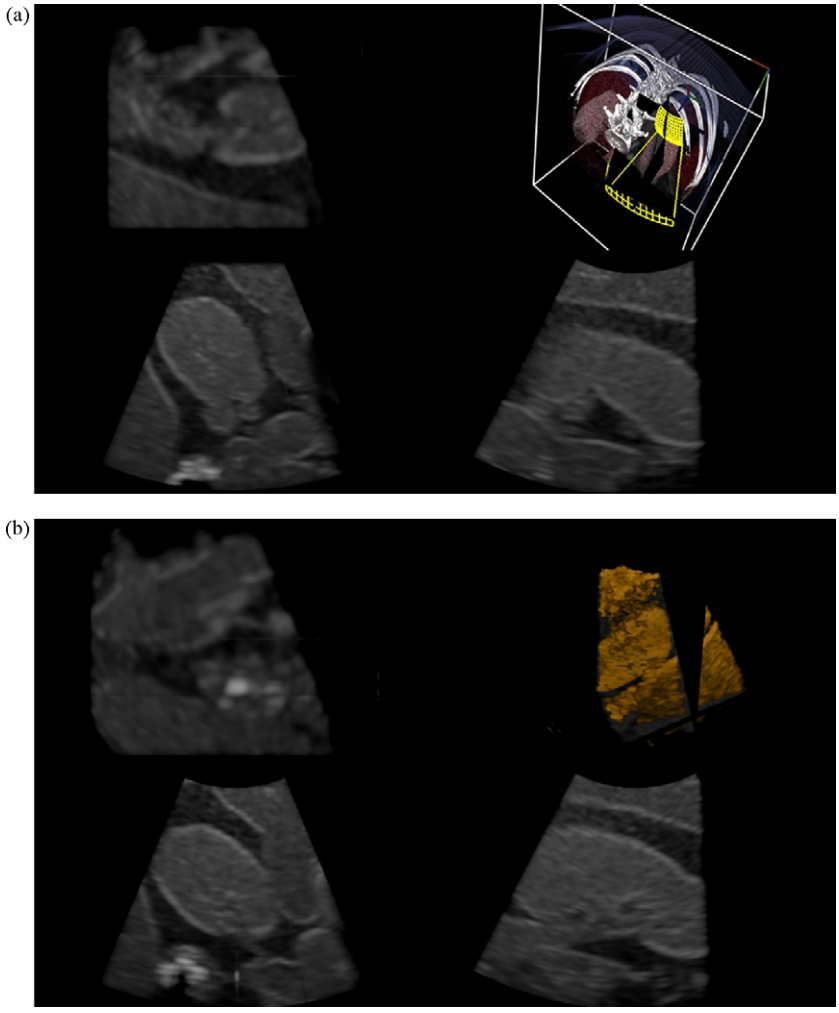

Fig. 11 – Screen shots from visualization of simulated 3D ultrasound volume. (a) Coronal, sagittal and axial MPRs extracted from simulated ultrasound volume. Upper-right quadrant: Volume rendering of CT data with wire frame rendering of ultrasound field of view and ultrasound MPRs inside the CT volume. (b) Coronal, sagittal and axial MPRs extracted from simulated ultrasound volume. Upper-right quadrant: Volume Rendering of simulated ultrasound volume with texture mapped MPR planes.

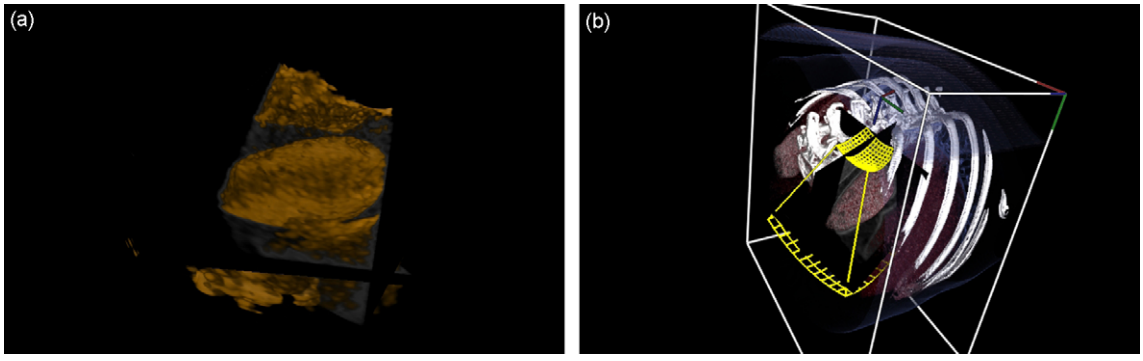

Fig. 12 – Screen shots from visualization of simulated 3D ultrasound volume. (a) Volume rendering of CT data with wire frame rendering of ultrasound field of view and ultrasound MPRs inside the CT volume. (b) Coronal, sagittal and axial MPRs extracted from simulated ultrasound volume. (b) Upper-right quadrant: Volume Rendering of simulated ultrasound volume with texture mapped MPR planes.

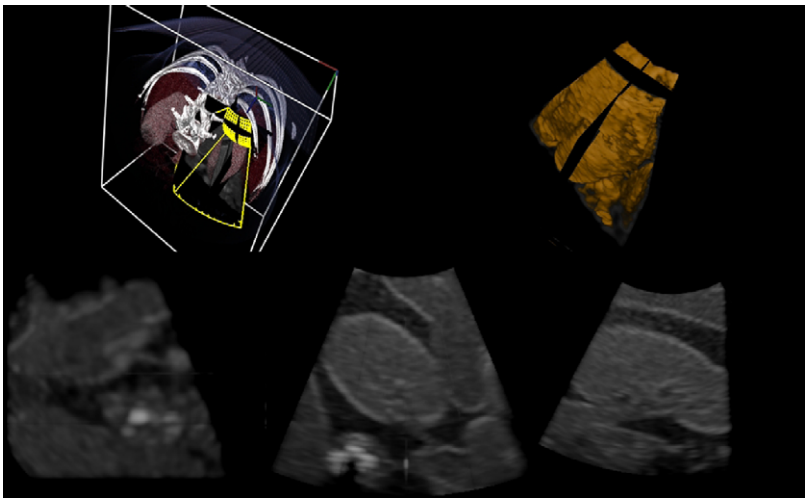

Fig. 13 – Screen shot from visualization of simulated 3D ultrasound volume. Top row, from left to right: Volume rendering of CT data with wire frame rendering of ultrasound field of view and ultrasound MPRs inside the CT volume. Volume Rendering of simulated ultrasound volume with texture mapped MPR planes. Bottom row, from left to right: Axial, sagittal and coronal MPR planes extracted from simulated ultrasound volume.

Table 1 – Performance in frames per second for the combined simulation and visualization.

Param. change	Radeon 1950Pro	Quadro FX3600M	Quadro FX5600
None—Simulation only	45	79	162
Volume pose	5	15	32
Transducer pose	4	16	35
Transducer shape	4	15	34
Sim. param.	40	76	157
Transfer function	9	31	47

Table 2 – Benchmark configuration parameters for performance evaluation of single 2D ultrasound image simulation.

Benchmark index	Scan lines	Depth samples	Ultrasound image resolution
1	256	256	256 × 256
2	512	512	512 × 512
3	512	512	640 × 480
4	512	512	800 × 600
5	1024	1024	1024 × 1024

images in 2D and 3D using different visualization techniques in real-time. A screen-shot of the application's user interface is shown in Fig. 10. The user interface consists of four main views and two widgets for adjusting the simulation parameters and the direct volume rendering transfer function:

- *3D view*: The 3D view displays 3D CT volume and the ultrasound plane within the 3D volume. The 3D image is rendered using standard volume rendering techniques and the ultrasound image is texture-mapped onto the corre-

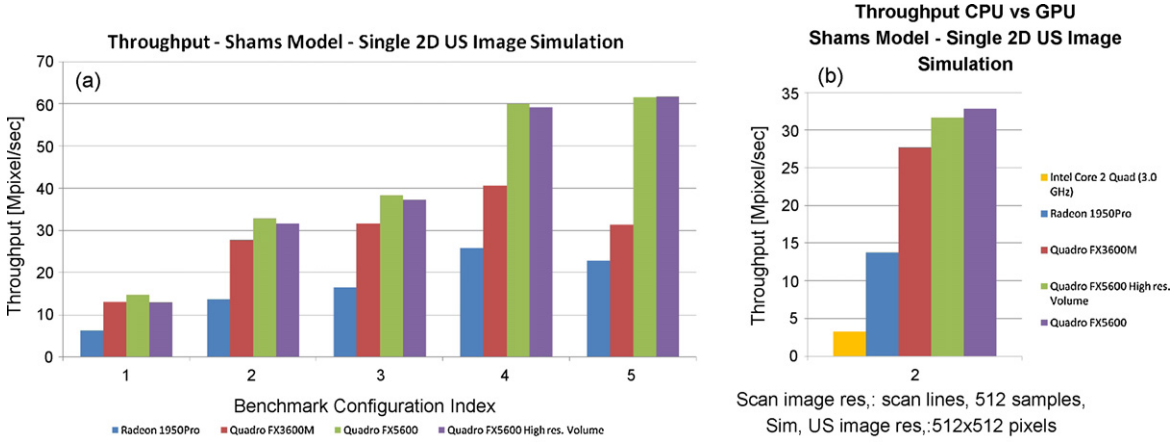

Fig. 14 – (a) Benchmark results for simulation of a single 2D ultrasound image using the simulation model by Shams et al. for benchmark configurations denoted in Table 2. (b) CPU versus GPU performance comparison for simulation of a single 2D ultrasound image: 512 scan lines with 512 samples, 512 × 512 pixels image resolution.

sponding plane within the 3D volume. The user can change the details of the volume rendering (e.g. display internal organs, vasculature, bones or skin surface) in real-time by changing transfer function parameters.

- *Ultrasound View*: displays the simulated 2D/3D ultrasound image. For 3D ultrasound, the user can choose between the 2D display of coronal, sagittal or axial MPR reconstructions or a 3D direct volume rendering of the simulated ultrasound volume (see Figs. 11–13).
- *CT view*: The CT view displays a multi-planar reconstruction (MPR) of a CT plane that corresponds with the current position, orientation and field of view of the ultrasound image.
- *Combined view*: shows the fusion of the ultrasound and CT images and allows the user to easily compare ultrasound and CT features.

Ultrasound simulation and visualization parameters can be adjusted interactively (see Fig. 10(a), the depicted simulation parameters are for the model by Shams et al. [21]). The parameters are organized in groups. Certain groups are shared among all simulation models, others are specific to a particular ultrasound simulation model.

- *Transducer geometry and pose*: allows for the selection of the probe geometry (i.e. linear or curvilinear), setting the probe position and orientation, field-of-view, and minimum and maximum penetration depth.
- *Scan line traversal*: parameters in this group affect the scan line traversal stage. For instance, for the model in [21] these are the air and bone segmentation thresholds.
- *Pre/post-scan conversion*: the options include type of filters, window sizes, standard deviation of the filters, etc.
- *Compositing*: the options include log compression, blending factors and boolean flags denoting whether certain operations should be executed in the compositing shaders.
- *Visualization*: the options include CT window level and window width for 2D CT slice visualization, and blending factors and colors for the combined CT/ultrasound visualization.

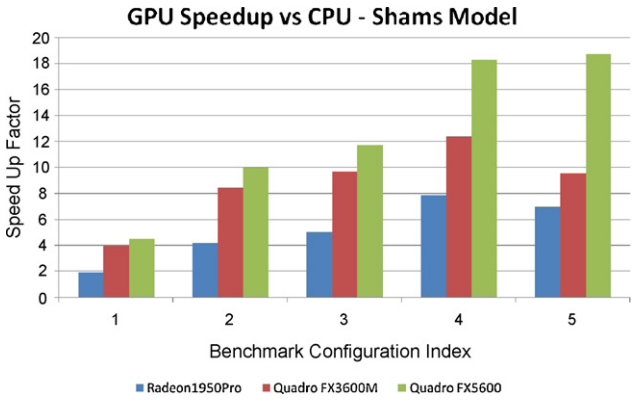

Fig. 15 – GPU speedups for the simulation of a single 2D ultrasound image using the model by Shams et al. for benchmark configurations in Table 2.

4. Computational performance

The ray-based simulation of ultrasound is very efficient on the GPU. In this section, we present detailed performance results for the two main application areas of the simulation framework, simulation and visualization in ultrasound training and simulation for registration of ultrasound and CT images. The requirements for the two applications are different. Ultrasound training requires more realistic simulation of ultrasound images and uses a more accurate simulation model model, whereas for registration only a few ultrasound specific effects have to be simulated and a more simplified ultrasound model can be employed. We first describe the test environment and the data-sets and parameters used for the performance evaluation. Then, we describe the performance of the simulation and simultaneous visualization for ultrasound training using the ultrasound model presented in this paper. We conclude with an analysis of the throughput performance of the ultrasound simulation for registration of ultrasound and CT images using the ultrasound model by Wein et al. [20].

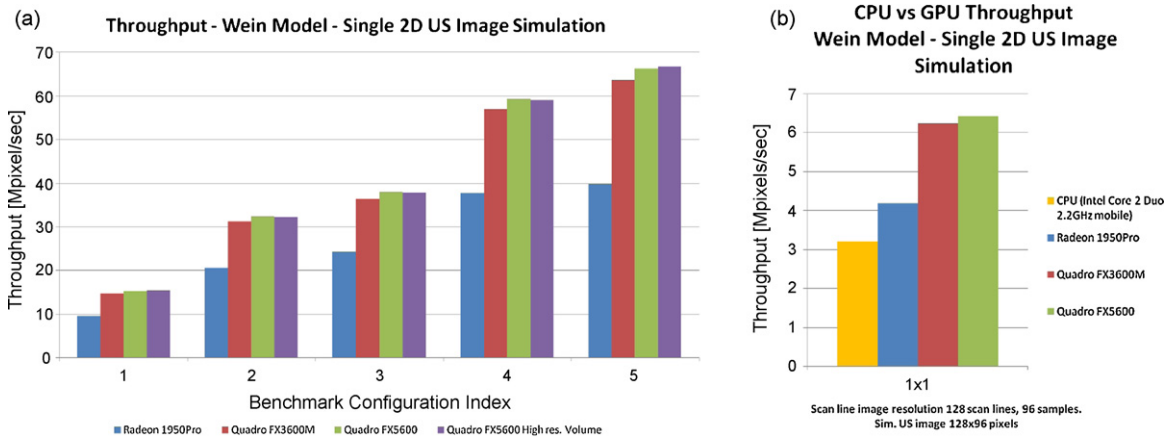

Fig. 16 – (a) Simulation performance for a single 2D ultrasound image using the benchmark configuration parameters specified in Table 2. (b) Comparison of GPU and CPU throughput for simulating a single 2D ultrasound image of 128 × 96pixels.

4.1. *Test environment*

The performance of the ultrasound simulation and visualization was evaluated on three computers:

(1) AMD Opteron 165 CPU (2 × 1.8 GHz), 2 GB RAM, AMD/ATI Radeon 1950Pro with 256 MB RAM (Shader Model 3.0), Win. XP (32-bit)
(2) Intel Core Duo 2 CPU (2×2.66 GHz, mobile), 4 GB RAM, NVIDIA Quadro FX3600M with 512 MB RAM (Shader Model 4.0), Win. Vista (64-bit)
(3) Intel Core Duo 2 CPU (2× 2.66 GGHz), 4 GB RAM, NVIDIA Quadro FX5600 with 1.5 GB RAM (Shader Model 4.0), Win. Vista (64-bit)

For our performance measurements, we used a CT volume of the abdomen of a human subject with a resolution of 512 × 512 × 484 voxels (16-bit, 242 MB). A speckle volume of the same size was pre-computed from the CT data (32-bit float, 484 MB). The full-size volumes were used with Quadro FX5600 (1.5 GB RAM) but the volumes were down-sampled for Quadro FX3600M and Radeon 1950Pro cards to fit within the GPU's memory.

4.2. *Performance of simulation and visualization for training applications*

Training applications require interactive frame rates for operation of a virtual transducer and provision of a smooth

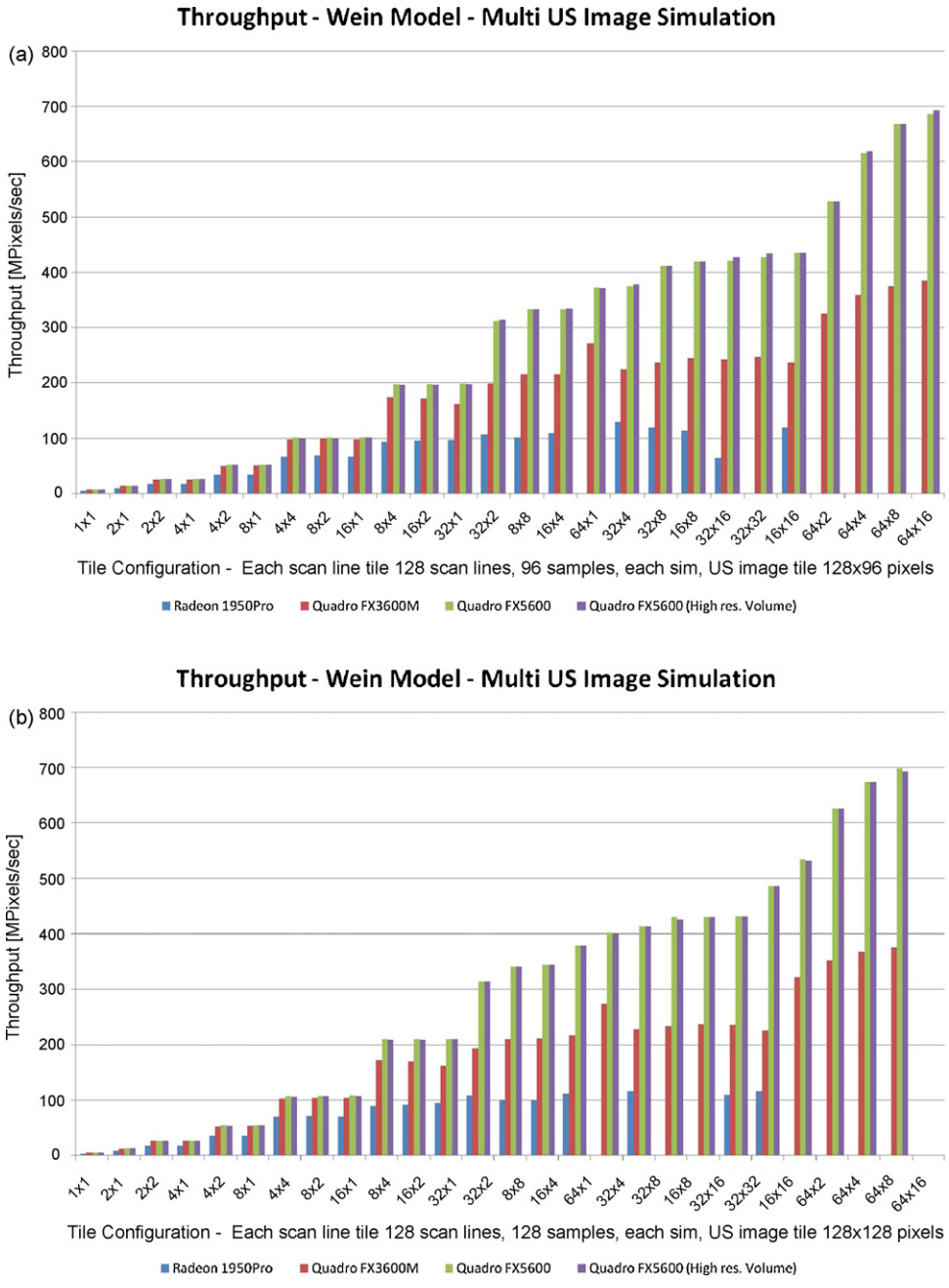

Fig. 17 – Throughput [MPixels/s] for different image tile configurations: the throughput increases with the number of tiles and is typically optimal for square configurations.

and uninterrupted visual feedback. In this section, we first evaluate the combined performance of the simulation and visualization and then compare the performance of simulations on the different GPUs and with a CPU implementation.

Visualization is typically the more time-consuming part of the algorithm. The performance of visualization is dependent on many parameters, e.g. direct volume rendering technique, local illumination, and the chosen transfer function. For our experiments, we adjusted the parameters for high quality visualization using pre-integrated classification for DVR, local illumination with Blinn-Phong shading and on the fly-gradient evaluation. A resolution of 640 × 480 pixels was used for rendering, the number of samples per ray was set to 512, early ray termination and empty space leaping optimizations were also activated.

Table 1 shows the average frame rate of the combined simulation and visualization as the operator varies simulation and visualization parameters. Changes in relative orientation or position of the volume with respect to the camera, the ultrasound transducer with respect to the volume, or transducer geometry affect the performance. This is due to the fact that these changes require the entire rendering pipeline to be re-executed. As can be seen, the algorithm performs interactively on higher end GPUs such as FX5600, under all conditions. The performance is equally good for mainstream GPU models (e.g. 8800GTX/GTS and 9800GTX) that have around the same number of stream processors as FX5600. However, for lower end GPUs the rendering quality has to be reduced in order to achieve interactive frame rates, under all conditions.

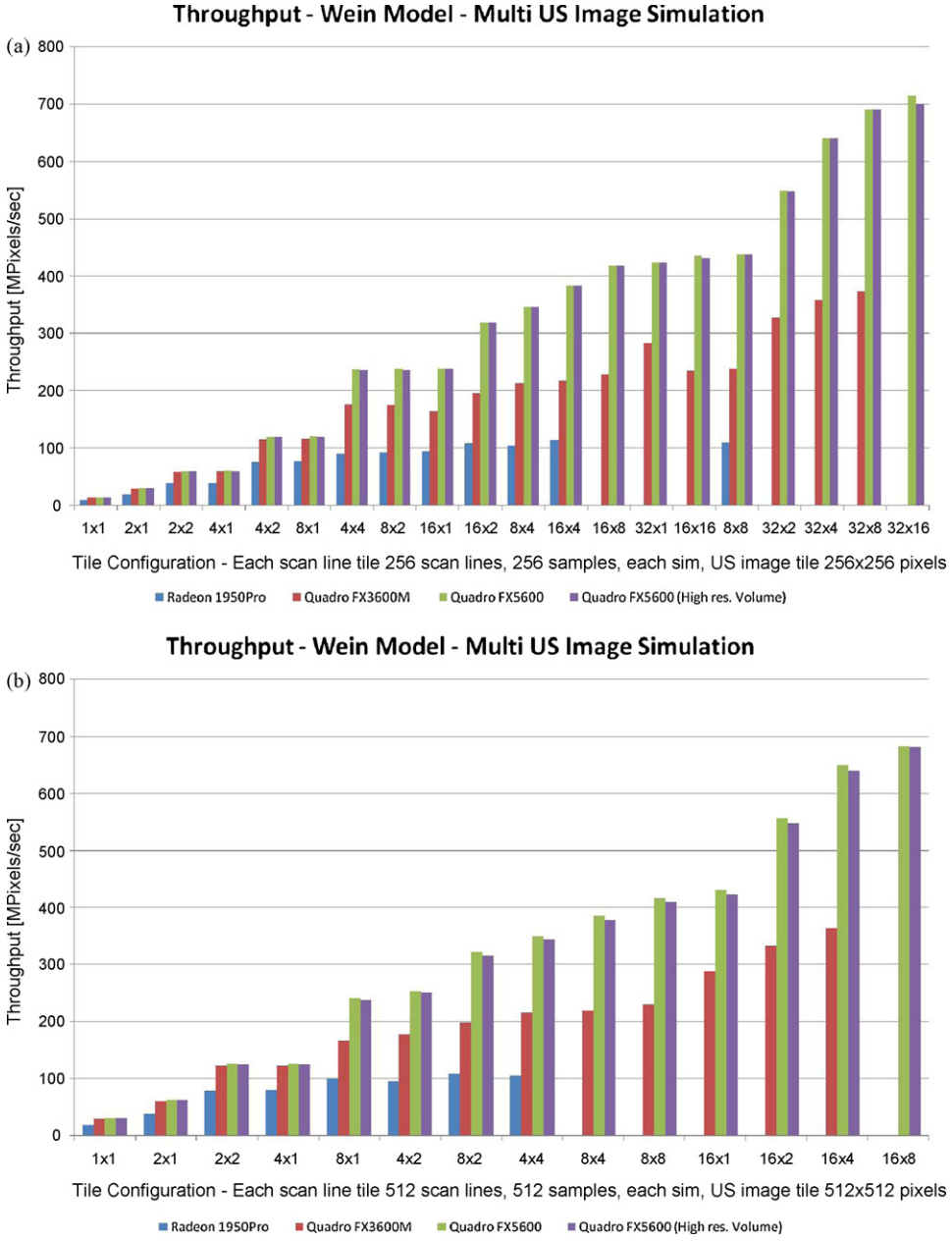

Fig. 18 – Throughput [MPixels/s] for different image tile configurations: the throughput increases with the number of tiles and is typically optimal for square configurations.

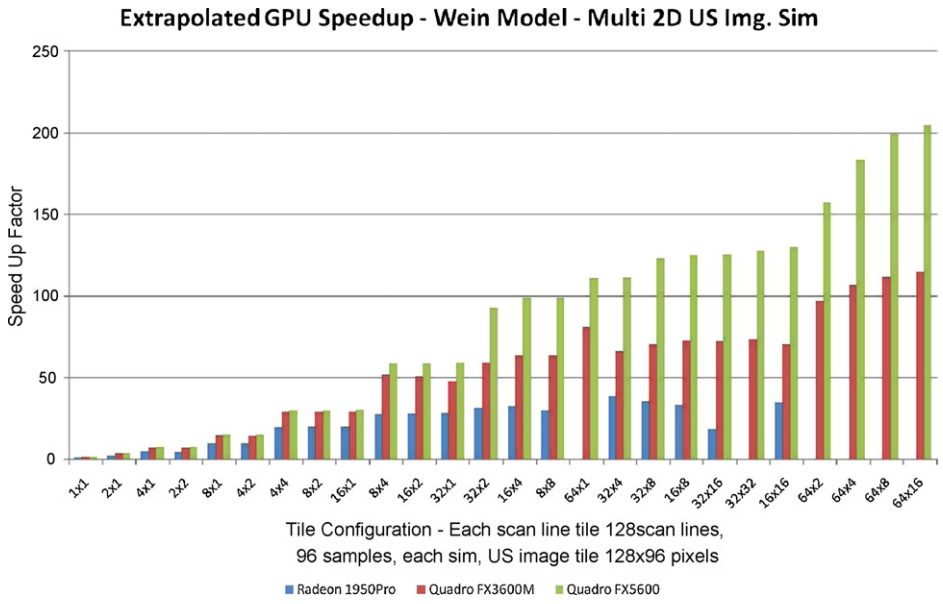

Fig. 19 – Speedups for GPU implementation of Wein's model and simulation of multiple images of resolution 128 × 96 compared to performance of CPU implementation for simulation of the same total number of images. CPU implementation performance values estimated from timings provided by Wein et al.

The performance of the simulations were also measured by throughput in mega-pixels rendered per second for varying numbers of scan lines and samples, and ultrasound image resolutions (see Table 2 for benchmark configuration details). The results were compared with the throughput of a CPU implementation measured on an Intel Core 2, Quad 3.0 GHz processor. The results are given in Fig. 14. Unlike the GPU version, the throughput for the CPU implementation does not (noticeably) vary with the image size. The GPU implementation outperforms the CPU by up to ∼ 20 times (see Fig. 15).

4.3. *Performance of simulation for registration applications*

Registration of ultrasound and CT images requires the repeated simulation of ultrasound images at various orientations and positions from the CT data during optimization of the registration parameters. The simulation of a single 2D ultrasound image using the model in [20] barely utilizes the computational resources of the GPU. Fig. 16(b) depicts the throughput for a single 2D ultrasound image using the scan line and ultrasound image resolutions given in Table 2. The throughput is limited by the number of active fragment shaders/stream processors and control program overhead in the scan line traversal stage. As can be seen, the throughput improves as the size of the ultrasound image increases, since GPU resources are being more optimally utilized for larger images. The authors of [20] kindly provided us with the timings of their CPU ultrasound simulation C++ implementation on a 2.2 GHz Intel Core 2 Duo mobile processor. The simulation of a single 2D ultrasound image, 128 × 96 took ∼ 3.5 ms. To compare the performance, we used this value for estimation of the GPU speedups compared to the CPU for the simulation of single 2D ultrasound images (see Fig. 16(b)) and multiple 2D ultrasound images on the GPU (see Fig. 19) of the same resolution.

The key to increase the throughput of the simulation is to process more fragments in a single pass of the scan line simulation stage. We achieve this by packing multiple ultrasound images into tiles of a large texture on the GPU. In each simulation pass, multiple ultrasound images are processed resulting in an improved GPU hardware utilization and increased data throughput per second.

Figs. 17 and 18 depict the throughput achieved by the parallel simulation of multiple ultrasound images. The throughput increases by the number of image tiles. Using a tile configuration of 32 × 16 images, each with a 256 scan lines with 256 samples and an ultrasound image resolution of 256 × 256 pixels, we achieved a throughput of > 700 MPixels/s on an NVIDIA Quadro FX5600 board.

Fig. 19 depicts the speedup for our multi-image GPU simulation compared to the CPU implementation by Wein et al. for the simulation of ultrasound images of 128 × 96 pixels resolution and 128 scan lines with 96 samples. With a Quadro FX5600 a speedup of more than 200 times can be achieved.

5. Conclusion

In this paper, we presented a novel framework for ray-based simulation of ultrasound and its visualization in real-time, which can be run on a wide range of standard GPU hardware. The modular design of the simulation and rendering pipeline support ultrasound models with various degrees of complexity.

We demonstrated the superior performance of our method for two main applications of ultrasound simulation, ultrasound training and registration of CT and ultrasound images

with significant speedups (up to 200 times) compared to CPU-based implementations.

The main benefit of a GPU implementation is improved performance of computations which allows us to run the combined simulation and visualization at interactive frame-rates. This is possible due to higher processing power of the GPU and the fact that simulation on the GPU removes the need for costly data transfers between the host and the GPU for visualization purposes. Registration applications also benefit from the improved performance of the simulations on the GPU.

We chose to implement our method in OpenGL instead of CUDA/CTM for a number of reasons: (a) our problem could be formulated as a ray casting problem which is well studied in the OpenGL domain, (b) hardware independence, and (c) visualization is more suited to OpenGL and there are still issues with interoperability between CUDA and OpenGL. We expect interoperability issues to be resolved in the future. OpenCL (Open Computing Language) which promises to provide platform independent access to GPU hardware comparable to CUDA/CTM will also become available. We expect OpenCL to provide a more generic programming platform which will be more suitable for implementation of more complex ultrasound models. Conceptually, the simulation pipeline, as described in this paper, can be implemented without much change. The scan line stage can be further improved as it will be able to run in a single pass in a CUDA-like kernel.

Ongoing work deals with the integration of the presented work into an augmented reality framework for evaluation of ultrasound acquisition performance by physicians. The framework is also being integrated into an ultrasound/CT registration application. The method will be used for multimodal registration of freehand 3D ultrasound data to CT and simultaneous mono-modal registration of a set of ultrasound volumes. In the future, we plan to use fusion of CT and ultrasound data for an improved reconstruction of large ultrasound volumes incorporating knowledge of the anatomy from the CT image and from ultrasound artifacts.

Acknowledgment

Part of this work was supported by funding of the European Committee within the passport project – http://www.passport-liver.eu/.

REFERENCES

[1] H. Maul, A. Scharf, P. Baier, M. Wüstemann, H.H. Günter, G. Gebauer, C. Sohn, Ultrasound simulators: experience with the SonoTrainer and comparative review of other training systems, Ultrasound Obstet. Gynecol. 24 (5) (2004) 581–585.

[2] M.M. Knudson, A.C. Sisley, Training residents using simulation technology: experience with ultrasound for trauma, J. Trauma: Injury Infect. Critical Care 48 (4) (2000) 659–665.

[3] C. Terkamp, G. Kirchner, J. Wedemeyer, A. Dettmer, J. Kielstein, H. Reindell, J. Bleck, M. Manns, M. Gebel, Simulation of abdomen sonography. evaluation of a new ultrasound simulator, Ultraschall. Med. 24 (2003) 239–244.

[4] UltraSim: Ultrasound training simulator, MedSim Advanced Medical Simulations, Ltd., http://www.medsim.com/, 2008.

[5] D. Aiger, D. Cohen-Or, Real-time ultrasound imaging simulation, Real-Time Imag. 4 (4) (1998) 263–274.

[6] H.-H. Ehricke, SONOSim3D: a multimedia system for sonography simulation and education with an extensible case database, Eur. J. Ultrasound 7 (1998) 225–300.

[7] D. Henry, J. Troccaz, J.L. Bosson, O. Pichot, Ultrasound imaging simulation: application to the diagnosis of deep venous thromboses of lower limbs, in: W.M.W. Iii, A.C.F. Colchester, S.L. Delp (Eds.), Medical Image Computing and Computer Assisted Intervention (MICCAI), Lecture Notes in Computer Science, vol. 1496, 1998, pp. 1032–1040.

[8] J. Stallkamp, M. Walper, UltraTrainer—a training system for medical ultrasound examination, in: Medicine Meets Virtual Reality (MMVR), IOS Press, 1998, pp. 298–301.

[9] M. Weidenbach, C. Wick, S. Pieper, K.J. Quast, T. Fox, G. Grunst, D.A. Redel, Augmented reality simulator for training in two-dimensional echocardiography, Comput. Biomed. Res. 33 (1) (2000) 11–22.

[10] I.M. Heer, K. Middendorf, S. Müller-Eglo, M. Dugas, A. Strauss, Ultrasound training: the virtual patient, Ultrasound Obstet. Gynecol. 24 (2004) 440–444.

[11] A. Tahmasebi, K. Hashtrudi-Zaad, D. Thompson, P. Abolmaesumi, A framework for the design of a novel haptic-based medical training simulator, IEEE Trans. Informat. Technol. Biomed. 12 (5) (2008) 658–666.

[12] J.A. Jensen, N.B. Svendsen, Calculation of pressure elds from arbitrarily shaped, apodized, and excited ultrasound transducers, IEEE Trans. Ultrason. 39 (1992) 262–267.

[13] J.A. Jensen, Field: a program for simulating ultrasound systems, in: 10th Nordic-Baltic Conference on Biomedical Imaging Published in Medical & Biological Engineering & Computing, 1996, pp. 351–353.

[14] J.A. Jensen, S.I. Nikolov, Fast simulation of ultrasound images, in: IEEE Ultrasonics Symposium, 2000, pp. 1721–1724.

[15] G.E. Tupholme, Generation of acoustic pulses by baffled plane pistons, Mathematika 16 (1969) 209–224.

[16] P.R. Stepanishen, Transient radiation from pistons in an in nite planar baffle, J. Acoust. Soc. Am. 49 (1971) 1629–1638.

[17] J.A. Jensen, Simulation of advanced ultrasound systems using Field II, in: IEEE Int. Symp. on Biomedical Imaging (ISBI), 2004, pp. 636–639.

[18] A. Hostettler, C. Forest, A. Forgione, L. Soler, J. Marescaux, Real-time ultrasonography simulator based on 3D CT-scan images, Medicine Meets Virtual Reality (MMVR) vol. 111 (2005) 191–193.

[19] F.P. Vidal, N.W. John, A.E. Healey, D.A. Gould, Simulation of ultrasound guided needle puncture using patient specic data with 3d textures and volume haptics, Comput. Animat. Virtual Worlds 19 (2) (2008) 111–127.

[20] W. Wein, A. Khamene, D.-A. Clevert, O. Kutter, N. Navab, Simulation and fully automatic multimodal registration of medical ultrasound, in: N. Ayache, S. Ourselin, A. Maeder (Eds.), Medical Image Computing and Computer Assisted Intervention (MICCAI), Lecture Notes in Computer Science, Springer, 2007, pp. 136–143.

[21] R. Shams, R. Hartley, N. Navab, Real-time simulation of medical ultrasound from CT images, in: D. Metaxas, L. Axel, G. Fichtinger, G. Szekely (Eds.), Medical Image Computing and Computer Assisted Intervention (MICCAI), Lecture Notes in Computer Science, Springer, New York, USA, 2008, pp. 734–741.

[22] W. Hedrick, D. Hykes, D. Starchman, Ultrasound Physics and Instrumentation, 3rd edition, Mosby-Year Book, Inc., 1995.

[23] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, Real-Time Volume Graphics, AK Peters, Ltd, 2006.
[24] P. Sabella, A rendering algorithm for visualizing 3D scalar elds, Comput. Graphics 22 (4) (1988) 51–58.
[25] J. Krüger, R. Westermann, Acceleration techniques for GPU-based volume rendering, in: G. Turk, J.J. vanWijk, R.J. Moorhead II (Eds.), Proceedings of the 14th IEEE Visualization 2003 (VIS'03), IEEE Computer Society, 2003, pp. 287–292.
[26] S. Stegmaier, M. Strengert, T. Klein, T. Ertl, A simple and flexible volume rendering framework for graphics-hardware-based raycasting, in: Proceedings of the International Workshop on Volume Graphics, vol. 5, 2005, pp. 187–195.
[27] H. Scharsach, M. Hadwiger, A. Neubauer, S. Wolfsberger, K. Buhler, Perspective isosurface and direct volume rendering for virtual endoscopy applications, in: Proceedings of Eurovis/IEEE-VGTC Symposium on Visualization, 2006, pp. 315–322.
[28] K. Engel, M. Kraus, T. Ertl, High-quality pre-integrated volume rendering using hardware-accelerated pixel shading, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, 2001, pp. 9–16.
[29] Q. Zhang, R. Eagleson, T. Peters, Rapid voxel classification methodology for interactive 3D medical image visualization, in: N. Ayache, S. Ourselin, A. Maeder (Eds.), Medical Image Computing and Computer Assisted Intervention (MICCAI), Lecture Notes in Computer Science, Springer, 2007, pp. 86–93.