Computers in Biology and Medicine 43 (2013) 1382–1389

Contents lists available at ScienceDirect

Computers in Biology and Medicine

journal homepage: www.elsevier.com/locate/cbm

GPU-accelerated 3D mipmap for real-time visualization of ultrasound volume data

Koojoo Kwon, Eun-Seok Lee, Byeong-Seok Shin *

Department of Computer and Information Engineering, Inha University, Incheon, Republic of Korea

ARTICLE INFO

Article history:
Received 22 February 2013
Accepted 13 July 2013

Keywords:
Ultrasound data
Volume rendering
3D noise filtering
Mipmap

ABSTRACT

Ultrasound volume rendering is an efficient method for visualizing the shape of fetuses in obstetrics and gynecology. However, in order to obtain high-quality ultrasound volume rendering, noise removal and coordinates conversion are essential prerequisites. Ultrasound data needs to undergo a noise filtering process; otherwise, artifacts and speckle noise cause quality degradation in the final images. Several two-dimensional (2D) noise filtering methods have been used to reduce this noise. However, these 2D filtering methods ignore relevant information in-between adjacent 2D-scanned images. Although three-dimensional (3D) noise filtering methods are used, they require more processing time than 2D-based methods. In addition, the sampling position in the ultrasonic volume rendering process has to be transformed between conical ultrasound coordinates and Cartesian coordinates. We propose a 3D-mipmap-based noise reduction method that uses graphics hardware, as a typical 3D mipmap requires less time to be generated and less storage capacity. In our method, we compare the density values of the corresponding points on consecutive mipmap levels and find the noise area using the difference in the density values. We also provide a noise detector for adaptively selecting the mipmap level using the difference of two mipmap levels. Our method can visualize 3D ultrasound data in real time with 3D noise filtering.

1. Introduction

Visualization of ultrasound data is a technique used to image ultrasonic echo signals. An ultrasound device has several advantages compared with other medical imaging modalities such as CT and MRI. The patient can be diagnosed comfortably with ultrasound imaging since it is a non-invasive method. In addition, the procedure used to acquire the data is faster than that of other imaging techniques, and it offers interactive visualization of the underlying anatomy with the capacity to represent dynamic structures such as cardiac motion. Three-dimensional (3D) visualization of ultrasound data is a helpful method in the field of clinical imaging. While early applications were focused on cardiac, obstetric, and gynaecological applications, its capabilities continue to expand throughout the field of clinical imaging.

Two significant problems have to be surmounted in order to achieve efficient 3D ultrasound data visualization: noise reduction and coordinate conversion between conical ultrasound coordinates and Cartesian coordinates. First, noise filtering is used to reduce or eliminate noise since ultrasound data typically contains speckle noise and fuzzy boundaries. Pre- and post-processing of the ultrasound images can improve the quality of the final image. Several methods apply noise removal filters to the entire volume dataset in the pre-processing step using two-dimensional (2D) filter kernels [1,2]. However, 2D filtering methods ignore relevant information provided by adjacent 2D-scanned images. Second, CT and MR data are reconstructed in terms of a regular 3D voxel dataset by stacking parallel cross-sections. However, nonparallel slices are not suitable for this representation since ultrasound volume data are acquired using ultrasound probes. They require conversion between ultrasound coordinates and Cartesian coordinates during visualization in real time. Most conventional methods perform coordinates conversion between two spaces in the pre-processing stage, as it is a time-consuming task. 3D ultrasound imaging is often used in obstetric ultrasonography. It can provide real-time diagnosis of the anatomical region of interest and can be used to efficiently visualize object characteristics and diagnostic quality [3]. Due to the phase sensitive detection of ultrasound devices, ultrasound images are characterized by patterns of white and dark spots. These patterns are known as *speckles* and are forms of multiplicative noise [4]. Speckles are often deemed undesirable; therefore, several noise removal filters have been proposed to eliminate them. In particular, several adaptive filters have been presented for speckle reduction by Lee et al., Frost et al., and Kuan et al. [5–7]. These filters have a moving kernel on the image area. The balance and size of the window kernel is an

* Corresponding author. Tel.: +82 32 860 7452.
E-mail address: bsshin@inha.ac.kr (B.-S. Shin).

0010-4825/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compbiomed.2013.07.014

important factor, and it requires more processing time since it uses a large kernel. Kim et al. [8] proposed a noise filtering method for 2D ultrasound images that uses a median filter; however, it is required in the pre-processing stage. Burckhardt [9] presented a theoretical analysis of noise as an interference phenomenon. A number of researchers have used Gaussian-weighted averaging to smooth images and the Laplacian-of-Gaussian (LoG) to detect tissue boundaries [10]. Evans and Nixon [11] proposed the use of a variation of the median filter to estimate the local mode within ultrasound images. However, it is difficult to determine the mode within a small population such as a 9×9 filter kernel. As a result, they use a truncated median filter to estimate the mode of the local distribution. Although these filtering methods are good for image improvement, they are not suitable for volumetric ultrasound data as they ignore possibly relevant information provided by adjacent 2D-scanned images while using these 2D filters. Sun et al. [12] proposed an anisotropic diffusion method for reducing speckle on 3D ultrasound images. Their proposed method extends the 2D speckle reduction method to 3D data, but it takes several minutes for computing time. Morihisa et al. [13] provided an expansion from 2D to 3D filtering methods to improve image quality for cone-beam computed tomography. The effect of their noise reduction filter is to provide improvements in morphological structure depiction. It can be used to eliminate the effects of speckles and impulsive noise; however, its processing time is not suitable for real-time visualization. The visualization of 3D ultrasound has to be processed as fast as the associated data are acquired. Several approaches for visualizing volumetric data have been proposed [14]. Volume ray casting, in particular, provides superior image quality but comes at a high computational cost. The first implementations to use programmable graphics hardware were those by Krüger [15] and Röttger et al. [16] in 2003. They introduced volume visualization using a consumer PC in combination with acceleration techniques such as early ray termination and empty-space skipping. To enhance rendering performance while maintaining or even improving the image quality, Scharsach [17] presented a cached blocking GPU-based ray casting scheme. As the performance of GPUs improves, volume rendering for object transformation is now being performed in the rendering stage rather than the pre-processing stage [18–20]. To avoid dealing with complex meshes in proxy-based space warping, transformation can be defined as a point-wise warping of the volume. It is an inverse map method in which a viewing ray is warped from the original sampling position to a transformed one during ray traversal. Spatial transfer function [21] and displacement mapping [20] are used for object transformation. However, they need more computation time for transformation because the transformation function is complicated and most of the operations are executed in the fragment shader.

To visualize ultrasound data in real time with less noise, we propose a 3D-mipmap-based approach that performs noise filtering on GPUs. First, we generate point primitives from voxels of input volume data. Next, we select non-transparent point sets by applying an opacity transfer function and generate a 3D mipmap on the vertex shader and the geometry shader using these point sets. The density values of corresponding points on two mipmap levels are then compared, and the noise region is found using the difference in the density values. The difference signifies the disproportion of uniformity in the same area. The 3D mipmap has to be constructed within a short period of time before the fragment processing performs noise reduction. While conventional noise filtering methods rely solely on the fragment shader, our method reduces the burden placed on the fragment shader by increasing the use of the vertex shader for 3D mipmap generation. This improves the overall rendering speed since it balances the load of vertex and fragment processing. The main contribution of our work is that we make fast 3D filtering method using mipmap structure on GPU shader for ultrasound data. Using the proposed method, it is possible to obtain a real-time 3D filtering result better than the quality of applying 2D filter.

2. Method

We propose a noise reduction method for 3D ultrasound data that uses 3D mipmaps on the vertex shader, the geometry shader, and the fragment shader and reduces noise without destroying the fine details of the object. Our method comprises a two-pass approach using shaders. The first pass comprises vertex processing in which we generate a points set from the volume data. In this pass, several levels of the 3D mipmap are generated from the point primitive set on the vertex shader and the geometry shader. The second pass is implemented on the fragment shader. We provide a noise detector that adaptively selects the mipmap level using the difference in the sample values of two mipmap levels.

2.1. 3D mipmap generation

A mipmap is a pre-calculated, optimized collection of images accompanying an original texture and is intended to increase rendering speed and reduce aliasing. Most graphics hardware supports mipmaps because they can be implemented easily and do not require a lot of storage. However, the supported mipmaps are only 2D-based mipmap structures. We need to provide mipmaps for 3D textures on the vertex shader since recent GPU architectures can easily remove and create vertices at runtime. This can be performed repetitively at each shader phase in the graphics pipeline.

Fig. 1 depicts the 3D mipmap generation process. First, vertex buffer VB_i $(0 \le i \le n)$ and stream output buffer SO_i are generated in accordance with the mipmap levels. The number of mipmap levels generated by the user is represented as n. We store three voxels for one vertex, as this helps to reduce the amount of memory required and is suitable for representing voxel coordinates XYZ. An average of eight contiguous voxels for the current level are stored in the vertex buffer for the next level while a 3D mipmap is being generated. The i-th voxel position of mipmap level $n-1$, corresponding to the i-th voxel on mipmap level i, is addressed using Eq. (1) since the texture coordinates of each mipmap level is normalized from zero to one. The output of each step of SO corresponds to each level of the 3D mipmap. This mipmap generation process is completed after these processes have been repeated n times. The results are then loaded into video memory, and the final image is rendered in the fragment shader using the 3D mipmap texture

$$\begin{aligned} x_{n-1} &= x/xSize_{n-1} \\ y_{n-1} &= y/ySize_{n-1} \\ z_{n-1} &= z/zSize_{n-1} \end{aligned} \tag{1}$$

where, x, y, and z are the specific position of the 3D mipmap and $xSize$, $ySize$, and $zSize$ are the sizes on the n-th level of the current 3D mipmap. In our method, there are two ways to generate the SO: either a thread is assigned to a single vertex (Fig. 2(a)) or the thread has three channels for one vertex (Fig. 2(b)). We reduce the number of threads using the method depicted in Fig. 2(b), which we call *compressed SO*, instead of the method shown in Fig. 2(a). When the method depicted in Fig. 2(a) is used, all the vertices are processed at each thread in parallel. However, indices u, v, and w have to be calculated on the volume texture in each thread since u, v, and w signify the position of the 3D texture. In addition, it is not suitable for volume data since most hardware have limitations on the maximum number of threads. On the other hand, we can