Filip Malawski
Łukasz Czekierda

COMPRESSION OF IMAGE SEQUENCES IN INTERACTIVE MEDICAL TELECONSULTATIONS

Abstract *Interactive medical teleconsultations are an important tool in modern medical practice. Their applications include remote diagnostics, conferences, workshops, and classes for students. In many cases, standard medium or low-end machines are employed, and the teleconsultation systems must be able to provide a high quality of user experience with very limited resources. Particularly problematic are large datasets consisting of image sequences that need to be accessed fluently. The main issue is insufficient internal memory; therefore, proper compression methods are crucial. However, a scenario where image sequences are kept in a compressed format in the internal memory and decompressed on-the-fly when displayed is difficult to implement due to performance issues. In this paper, we present methods for both lossy and lossless compression of medical image sequences that only require compatibility with the Pixel Shader 2.0 standard, which is present even on relatively old, low-end devices. Based on the evaluation of quality, size reduction, and performance, these methods have been proven to be suitable and beneficial for medical teleconsultation applications.*

Keywords image compression, medical teleconsultations, telemedicine

Citation Computer Science 18 (1) 2017: 95–114

1. Introduction

Medical imaging is an essential element of diagnosing patients in almost all medical fields. In recent years, the amount of imaging equipment and number of performed examinations has increased more rapidly than the number of radiologists qualified to analyze the data. In particular, remote hospitals with limited human resources are frequently dependent on cooperation with major medical centers. In order to make this collaboration effective, dedicated teleconsultation systems have been developed.

Most advanced systems allow for interactive, synchronous, online collaboration among multiple participants. Due to their usefulness, these systems can also be utilized in other applications, such as medical conferences, workshops, and classes for students. In many cases, professional diagnostic equipment is not available, and standard medium or low-class machines are used. Medical imaging data such as ultrasound (US), computed tomography (CT), and magnetic resonance (MR) examinations are often very extensive since they include time dimension (US) or 3D spatial information (CT/MR), both stored as sequences of images. On the other hand, users of teleconsultation systems expect to be able to quickly browse all of the data. Therefore, even though the data is sent to all participants of a teleconsultation prior to its start, issues arise with handling the data locally, as the machines may be unable to load and display the images with sufficient speed. Moreover, the collaboration process is only as effective as the weakest machine allows, since all participants need to wait for it to synchronize the displayed data.

The most important reason for issues with loading and displaying imaging data effectively is insufficient internal memory. The datasets often exceed available internal memory resources; therefore, the data must be partially stored on the hard drive during a teleconsultation session (which results in unacceptable delays). Keeping the image sequences in a compressed format and decompressing each frame only when it is displayed can greatly reduce memory requirements. On the other hand, on-the-fly decompression is computationally very expensive. Sequential methods running on the CPU are not sufficient in this case. Parallelized algorithms that run on the GPU are much more effective; however, they usually require dedicated devices that are compatible with GPGPU frameworks such as CUDA or OpenCL. These are often unavailable on medium- or low-class devices that can be employed for teleconsultations.

For this reason, we propose two novel methods for lossy and lossless compression of medical images tailored especially for teleconsultation applications that require only a compatibility with the Pixel Shader 2.0 standard (a feature present even in relatively old, low-class machines). We provide an evaluation of both methods in terms of size reduction and efficiency that indicates that they are well-suited for medical teleconsultation systems. The methods were designed for and tested with an actual telemedical system – TeleDICOM [3, 9].

2. Background

2.1. Medical teleconsultations

Medical images constitute a very important element of medical documentation in contemporary medicine. Continuous progress in the areas of medical research and the development of diagnostic equipment allows us to gather more and more information describing the human body and, as a result, treat it more efficiently. The proper analysis of medical images requires significant knowledge and experience. Unfortunately, the number of experts able to properly interpret medical images has increased much more slowly than the amount of imaging equipment has in recent years. It is often the case that small, remote hospitals lack such experts and must rely on cooperation with larger medical centers.

Digital representation of medical data enables us to replicate and transfer it easily, therefore allowing for remote analysis and processing. Such a process is commonly referred to as a medical teleconsultation and is often supported by dedicated systems. The architecture of a typical system devoted to remote diagnostics is relatively simple: it includes sending appropriate data, analyzing it on the remote side, and returning the diagnosis – usually in the form of a textual descriptions and image annotations. Examples of such a system can be found in [6, 16, 19]. Advanced systems allow for interactive, synchronous, online collaborations. In this case, many users concurrently participate in a consultation session and are able to share the view of the session as fully as possible. This approach mirrors the traditional consultation process performed locally much more accurately than simple asynchronous systems. Moreover, interactive systems often provide various diagnostic tools, which allows for the efficient collaboration of medical experts. Synchronous interactive teleconsultation systems are discussed in [3, 8, 17].

Although teleconsultation systems were primarily devised for cooperation between medical institutions, new applications have emerged in recent years. Interactive teleconsultation systems can be employed for:

- Remote diagnostics – teleconsultations between medical institutions are currently quite common, although it is worth mentioning that interactive collaboration is mostly employed in emergency situations. In standard consulting, asynchronous systems are still preferred due to the much-lower organizational overhead.
- Medical conferences – an expert can present a case remotely or comment on presented cases without traveling to the conference (which is often problematic due to the high workload of medical doctors).
- Workshops – an expert can remotely teach medical doctors how to analyze specific cases. In this scenario, the interactive tools are particularly useful.
- Teaching students – some medical universities have adopted teleconsultation systems as a part of their classes. It is worth noting than, even when students are in the same room, such systems improve the classes by providing each student with a terminal with a common view of the analyzed images.

The analysis of the medical images should generally be performed using professional diagnostic machines. Specific requirements vary significantly, depending on the modality. Diagnostics of CT or MR scans requires certified medical displays with appropriate grayscale depth. On the other hand, in the case of US modality, these requirements are not so strong, and standard machines are often sufficient. For teleconsultations, such non-professional devices are frequently employed, regardless of the modality of the data. Particularly in the case of conferences, workshops, and instruction of students, dedicated diagnostic machines are often unavailable, and it is up to the expert to decide how to present the given data considering the limited capabilities of the equipment. In the case of remote consultations between medical institutions, the use of non-professional machines is much less frequent (although these lesser machines can be utilized in some cases when the expert decides that they are sufficient for the particular data).

An important conclusion from the practices of using medical teleconsultation systems is that it is desirable for these systems to work efficiently not only with professional diagnostic machines but also with standard machines (which are often middle- or even low-class devices). In public healthcare, these low- class devices are quite common and not likely to be replaced soon, particularly in the case of remote hospitals that are often underfunded.

2.2. Medical imaging data

The formally adopted representation format of medical images is DICOM [23], which includes both imaging data as well as metadata such as information about the patient, parameters of the acquisition, calibration data, etc. The images are coded using various algorithms, such as RLE [12], JPEG-Lossless [13], or JPEG 2000 [14]. Due to the employed compression, the size of a DICOM file can be significantly decreased, which is important when it comes to storage and transfer. Nevertheless, the image must be decompressed prior to being displayed; therefore, teleconsultation systems must handle the full size of the images.

Particularly complex from the visualization point of view are files containing a sequence of images that are displayed as a video clip. Not only do they have significant volume, but they also must be presented fast enough (usually 25 to 50 fps). A good example is US files forming a part of an echocardiography examination, describing a period of heart operation (typically gated using an ECG signal). Also, in the case of 3D data such as CT or MR examination that contains a series of 2D images called slices, a frequent approach is to present them as a video clip as well. The diagnosing doctor may then browse through the slices or play them as a movie – in this case, 25 fps is usually sufficient.

The volume of a typical dataset prepared for consultation varies depending on the modality, the configuration of the acquisition machines, and other factors. Usually, a US examination may contain multiple sequences with up to 100 frames, and a CT/MR study includes a few series with even 1000 slices altogether. Typically, the resolution of a single image is 512×512 pixels or similar, and a single pixel is represented on 8 bits (US) or 12 bits (CT/MR). In many cases, more than a single examination is used, since a combination of many modalities or comparison of different examinations may be needed. Therefore, in a single teleconsultation session, hundreds of megabytes of imaging data are often used, which may be difficult to handle by the teleconsultation system.

In order to provide good user experience, the time needed for loading an examination should not exceed one second, and the users should be able to browse or play the image sequences in real time. Moreover, teleconsultation systems are expected to meet additional requirements: a) multiple image sequences may need to be opened concurrently for comparison; b) when a CT/MR scan series is presented as a video sequence, immediate access to all slices is necessary; c) in the case of a collaborative medical session, each participant's machine should display images at the same, or at least very similar, time in order to not slow down or hinder the entire interaction. Also, images and image sequences often need to be processed while being displayed. For instance, in the case of CT, the very basic requirement is providing a real-time manual adjustment of the currently displayed range of Hounsfield Units (HU) – an operation computationally similar to manipulating the contrast and brightness of an image.

Meeting these requirements is often problematic, mostly because the volume of a session dataset usually significantly exceeds the size of the internal memory of the machine that is expected to analyze the data. In such cases, the operating system starts to use the hard drive as virtual memory, which considerably slows down displaying images and significantly decreases the quality of the collaboration. One of the most efficient manners of addressing this issue is keeping the imaging data in the internal memory in a compressed format and decompressing each frame only when it is accessed. The frame is kept in the decompressed format only while being displayed (which is memory-efficient) but requires decompression each time the frame is loaded on the screen. On-the-fly decompression is computationally very expensive; therefore, performance concerns must be addressed while implementing such an approach (particularly in the case of medium- and low-class machines), and proper compression methods must be applied.

2.3. Image compression

Methods for image compression have been developing for decades. The most popular compressed formats (such as PNG [10] or JPEG [11]) are commonly used for web graphics and photography, respectively. In order to avoid any loss of relevant information, however, medical images are usually compressed using lossless compression (such as JPEG-LS [13] or JPEG 2000 [14]). In some cases, lossy compression of medical imaging data is acceptable – medical experts can decide if a compressed image still provides the proper diagnostic quality. In this work, we consider both lossy and lossless methods.

Algorithms for medical image compression have been widely investigated, and many different approaches can be found in the literature. Modifications of JPEG-LS and Discrete Cosine Transform (DCT) are discussed in [20] and [32], respectively. Contextual vector quantization is employed in [7], and 3D wavelet coders are used in [29]. A symmetry-based approach is discussed in [1, 28]. Although these methods have been proven to be efficient for medical image compression, it is not possible to run them on a CPU fast enough to provide 25 fps for on-the-fly decompression, particularly in the case of medium or low-end devices.

In order to achieve significant acceleration, parallel computations on GPUs are often employed. Methods devised for general image compression include parallelized variants of JPEG 2000 [18], LZ77 [2], LZSS [26], and 3D fast wavelets [5]. Multiple parallel algorithms targeted for medical images are available as well. These employ (among others) Discrete Wavelet Transform (DWT) [30], least-squares prediction [31], or Medical Image Lossless Compression (MILC) [27]. A common factor for all of these methods is that they require dedicated hardware that is compatible with CUDA [24] or OpenCL [15] frameworks. This requirement renders them impractical in the discussed scenario, since medium or low-end machines seldom have such capabilities.

For this reason, we decided to create compression methods tailored for such devices. In this work, we assume that a minimum configuration capable of running parallelized image processing algorithms on a GPU requires compatibility with Pixel Shader 2.0 (PS2.0). This feature is available even in low-end, several-year-old graphics cards. On the other hand, it allows for reasonable custom image manipulation on the GPU, therefore constituting a vital tool for our methods.

Pixel shader is a small piece of code that is run on the GPU directly before displaying the image. The code is run separately for each pixel, in parallel. A pixel shader instance has access to the whole input image and knows the coordinates of the pixel (whose final value it has to compute). It cannot, however, communicate with the other instances executed for other pixels. As a result, pixel shaders are very fast but limited to the algorithms where each pixel value may be computed independently. In this work, we used the Windows Presentation Foundation (WPF) framework [22], which enables us to employ pixel shaders in business applications.

Based on opinions gathered from doctors from John Paul II Hospital in Krakow, we decided to develop both lossless and lossy compression methods dedicated for medical teleconsultations. Lossy compression is intended for US modality, while lossless compression is employed for CT/MR examinations. For lossy compression, we adopted the DXT1 [21] compression scheme (which was accepted for US image compression by the doctors with whom we consulted). Although DXT1 is natively supported in most graphics cards (even relatively old ones), there are reasons for custom implementation. First of all, access to native support of the DXT1 format is dedicated for the game development frameworks and is very limited in business applications. Mixing various programming frameworks results in a significant decrease in performance. Secondly, native support is dedicated for color images only. Better size reduction may be achieved for grayscale images when using a custom implementation. Issues and proposed solution for the implementation of DXT1 with PS2.0 are discussed in Section 3.1. The proposed lossless compression method is based on the division of data in both the intra-frame and inter- frame domains. This is discussed in Section 3.2.

3. Methods

In this section, we describe the proposed methods for both lossy and lossless compression devised specifically for medical teleconsultation sessions. We assume that the imaging data is transferred prior to a scheduled meeting and, therefore, all necessary data is already present once the session commences (which is typical for teleconsultation systems operating with large datasets [4]). The compression stage is intended to be run after the data transfer and before the start of the session and, therefore, can be treated as part of the session initialization. Each participating machine runs the compression stage by itself, locally. Crucial for the whole process is the decompression stage, performed during the session.

Our main concern was to achieve real-time on-the-fly decompression with a target frame rate of 25 fps on medium and low-end machines. This implied a requirement of compatibility with PS2.0, which provides access to custom image processing on the GPU even for such devices. It is worth mentioning that the proposed methods are devised strictly for image sequences. Static images usually need no compression, since a single image requires much less memory than a sequence. In the case of high-resolution static images, longer loading times are acceptable (up to one sec); therefore, slower yet more-efficient compression methods may be applied.

3.1. Lossy compression

DXT Compression DXT, also known as S3 Texture Compression (S3TC) or Block Compression (BC), is a relatively simple algorithm originally created for the compression of textures in computer games. The main advantage of this method is that it operates on fixed sized blocks, therefore making it easily parallelizable. There are five variations of the algorithm (DXT1-DXT5), each tailored for different handling of the alpha channel. Since medical images are opaque, we only focus on the DXT1 variation (as the other variations are designed to handle transparency).

The DXT1 divides the image into independently processed blocks of 4×4 pixels. For each block, two colors that best represent the block are computed (c0 and c1). The colors are stored in the BGR565 format, requiring a total of 32 bits to store both of them. For opaque images, an additional two colors are computed at the decompression stage by the interpolation of c0 and c1:

$$\begin{aligned} c2 &= \frac{2}{3} * c0 + \frac{1}{3} * c1 \\ c3 &= \frac{1}{3} * c0 + \frac{2}{3} * c1 \end{aligned} \tag{1}$$

The pixels themselves are stored as a lookup table, with 2 bits per pixel indicating one of the four colors. An array of 16 pixels requires 32 bits; therefore, the total amount of memory required for a single block is 64 bits (32 bits for colors c0 and c1, and 32 bits for the lookup table). The memory layout of a single block is presented in Figure 1. DXT1 can handle both RGB24 and RGBA32 formats as input data. In the first case, the compression ratio is 6:1 (384 bits to 64 bits); in the second, it is 8:1 (512 bits to 64 bits).

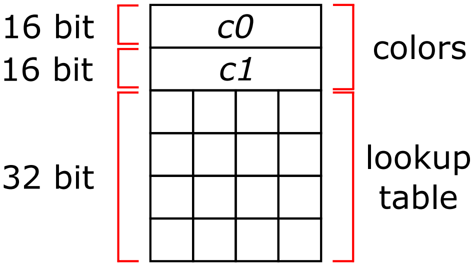

Figure 1. DXT1 single block memory layout.

Pixel Shader implementation: DXT1-ps

Due to the reasons explained in Section 2.3, we decided to create a custom implementation of the DXT1 algorithm using pixel shaders. These allow us to define how to compute output pixels that are displayed from input pixels of the loaded image. It is worth mentioning that the output pixel value may be computed using multiple input pixels. Since the compression stage is not time- critical in our scenario, we will only discuss the decompression stage. We first consider color images, as they are the more-difficult case. Adaptation to grayscale images will be discussed at the end of this section. We will refer to our pixel-shader modification of the DXT1 method as *DXT1-ps*.

Although DXT1 decompression seems simple enough to be easily implemented with pixel shaders, difficulties arise due to the requirement of compatibility with the PS2.0 standard. There are two important limitations that must be considered: 1) PS2.0 allows only for 32 texture instructions and 64 arithmetic instructions; and 2) the width and height of the compressed image must equal the width and height of the original image; otherwise, scaling artifacts would be introduced when loading the image to the video memory. We discuss several possible approaches in the context of these limitations in order to properly justify the final solution.

A natural approach would be to use a 4 bits-per-pixel (bpp) image format for storing a compressed image, since this would provide exactly the required amount of memory – 64 bits per each 16-pixel block. In this case, each color c0 and c1 would be coded on four input pixels. However, for the RGB565 format that is used in DXT1, retrieving each channel of colors would introduce substantial instruction overhead, since each color channel would be spanned over two input pixels (see Figure 2a). Moreover, PS2.0 does not support bitwise operations; therefore, an expensive modulo operation would be necessary to access parts of the input pixels corresponding to each channel. Additional costs would be computing the position of and accessing the input pixels required for computing the currently-processed output pixel. The limit for the number of instructions makes this solution unviable.

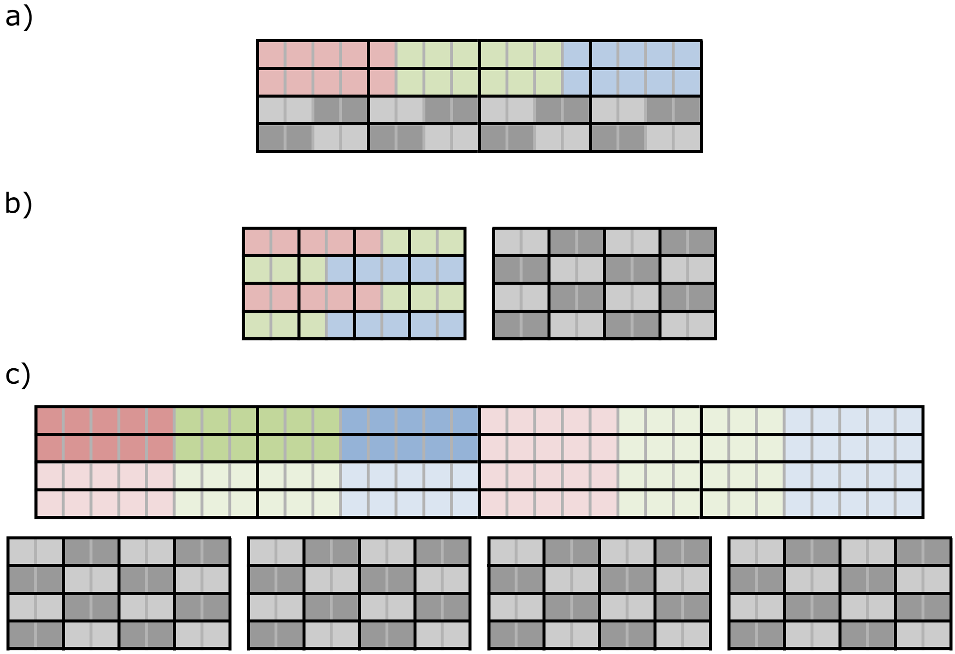

Figure 2. Potential DXT1-ps implementations for the color images – different memory layouts for a single 16-pixel block. Each array represents one PS2.0 texture. Black lines indicate the input pixels of the textures; gray lines indicate bits; red, green, and blue fields represent channels of the c0 and c1 colors; light- and dark-gray fields represent the lookup table. Layouts: a) a single 4-bpp texture; b) two 2-bpp textures (one for the colors, one for the lookup table); c) one 8-bpp texture per four consecutive frames for the colors, one 2-bpp texture per each frame for the lookup table.

Another approach is to employ multiple textures per image. PS2.0 allows us to load up to four input textures for one output image. DXT1 has a natural division of each block to the color and the lookup table parts. The lookup table can be easily represented with a 2-bpp format, with the output pixel positions aligned with the input image and, therefore, requiring no coordinate computations. This solution, however, would require the color part to be represented as a 2-bpp format as well, which would further increase the complexity of reconstructing the colors. As shown in Figure 2b, red and blue channels would be spanned over three input pixels and the green channel over four input pixels. The required number of instructions would greatly exceed the limit.

One possible solution to handle the color part would be to load the color part texture with a smaller dimension and 24-bpp format. Although this would substantially facilitate access to the colors and sufficiently reduce the number of required instructions, the color part texture would be automatically scaled up to the size of the lookup table texture before the pixel shader processing stage, therefore introducing additional interpolation errors.

In order to avoid interpolation in our final DXT1-ps method, we pack the color parts of four consecutive frames into one texture with 8-bpp format and dimensions equal to the lookup table texture. The 8-bpp format is sufficient to reconstruct the colors within the instruction limit. Each image is, therefore, represented as a lookup table texture (unique for each image) and a color texture shared among four images. The memory layout is presented in Figure 2c.

In the case of 8-bit grayscale images, the algorithm is slightly modified. The memory required for a single block in this case is reduced to 48 bits (16 bits for the two gray values, 32 bits for the lookup table). Since grayscale images have only one channel (as opposed to three channels in RGB images), reconstruction of the stored gray values is computationally much less complex. Therefore, it is possible to use a 2-bpp format for the gray values part and remain within the limit for the number of instructions. Nevertheless, since the gray values part occupies half the space needed for the lookup table part, we still need to combine gray values parts from two consecutive frames into one texture in order to preserve equal texture sizes and avoid scaling artifacts. The memory layout is presented in Figure 3.

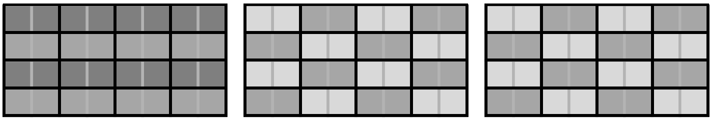

Figure 3. DXT1-ps implementation for grayscale images – the memory layout of a single block. Black lines indicate the input pixels of the image; gray lines indicate bits; dark- and medium-gray fields indicate the two coded gray values; light- and medium-gray fields indicate the lookup tables. Left array – 2-bpp texture holding the gray values for two consecutive frames; middle and right array – 2-bpp textures with the lookup tables for each of the two frames.

3.2. Lossless compression

The proposed method for lossless compression is based on differential coding between frames as well as division to blocks inside the frames. This exploits the nature of the considered data – 3D medical images such as CT or MR consist of a series of consecutive images taken in the transverse plane. As a result, any frame is frequently similar to the ones immediately preceding and following it. In general, the further from the considered frame, the lower the similarity is – frames that are far apart may, in fact, be significantly different. Nevertheless, when a few subsequent frames are considered, the similarity is sufficient enough in most cases to exploit differential coding. Division to blocks inside the frames is complementary to differential coding, and allows us to employ it efficiently with pixel shaders (as will be explained further in this section).

The conceptual model for differential coding is as follows. The series of images is divided into batches, which are compressed separately. Each batch has one *reference image*, which is coded directly – the value of the pixel in the image is the value that will be displayed on the screen. The rest of the images in the batch are coded differentially (*differential images*) – the value of each pixel in the image is the difference between the value that should be displayed and the value of the corresponding pixel in the reference image. An example of differential coding is presented in Figure 4. In standard coding, image B would require an 8-bpp format (values ranging from 0 to 63); however, in differential coding, it requires only 4 bpp (values ranging from -7 to 8).

60	31	52
51	63	45
31	27	39

Image A

Reference image

63	27	57
51	60	52
37	20	33

Image B

3	-4	5
0	-3	7
6	-7	-6

Differential image

Figure 4. Differential coding: Left – first image used as a reference; middle – second image coded directly; right – second image coded differentially.

The problem in this approach is the maximum difference of corresponding pixels. The bpp value in PS2.0 is set for the whole image; therefore, the pixel that has the highest difference sets the bpp required to code the whole differential image. In an extreme case when all pixels are the same except one, requiring n bits to code its difference from the reference image, the whole differential image would have to be coded with n bits, even though the images are almost identical. On the other hand, if we were able to set different bpp for each pixel, we could fully benefit from differential coding.

In order to verify the potential of this method, we measured both the maximum and average pixel differences between corresponding pixels of consecutive frames in publicly-available CT and MR datasets [32]. 10 CT and 10 MR datasets were employed, all images having a 12-bpp format. We conducted tests with different distances between frames, where d = 1 indicates two subsequent frames, d = 2 indicates frames that have one frame between them, and so on. Results are shown in Figure 5. The difference between two frames is expressed in bits per pixel, which is computed as a ceiling of log2 of the difference in the pixel values. The plot shows values averaged from all datasets. Results for the maximum difference (Figure 5 left) correspond to the whole image having the same bpp, and results for the average difference (Figure 5 right) correspond to the hypothetical situation of setting different bpp for each pixel.

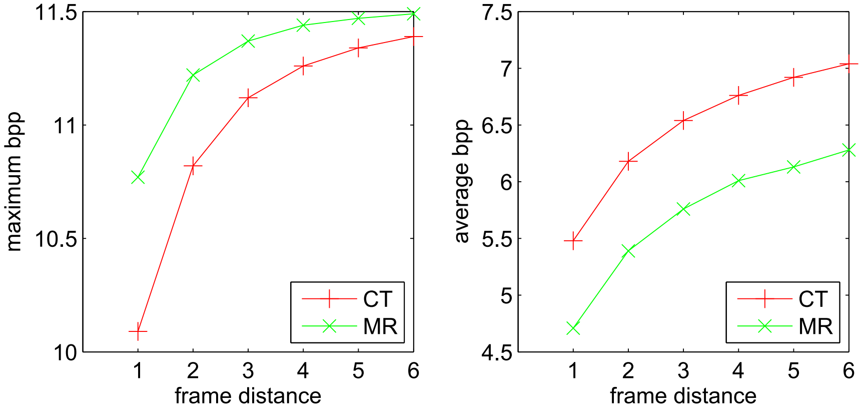

Figure 5. The maximum (left) and average (right) bpp required to code the difference of corresponding pixels in consecutive frames (distance 1–6), computed for publicly-available 10 CT and 10 MR datasets (12 bpp each).

As we can see, using differential coding with the same bpp for the whole image would result in almost no compression at all. On the other hand, using different bpp for each pixel would provide significant size reduction. However, as mentioned before, pixel shaders allow us to only use the same bpp for the whole image. Therefore, in order to take advantage of differential coding, we decided to additionally employ division to blocks inside each frame. The idea is to divide the image into several smaller images, each being coded as a separate block. The final image is then displayed as a set of adjoining smaller images – subimages. In each block (subimage), the differential frames can have different bpp. The limitation of such a solution is the overhead caused by displaying simultaneously multiple subimages. Using a separate subimage for each pixel would create hundreds of thousands of subimages, which would have to be reloaded with each frame. On the other hand, the larger the blocks, the less efficient the compression is, because each block is coded with the bpp required to code the maximum difference in this block. Therefore, a trade-off must be found between the subimage loading performance and compression efficiency. In Section 4.2, we analyze different block sizes.

The composition of the sequence compressed with the proposed method is presented in Figure 6. The input sequence is divided into batches of equal length (except for the last one, which may be shorter). Each batch is encoded as one reference frame and a set of differential frames. Each frame (reference and differential) is divided into equal-sized square blocks. The blocks in the reference frame are encoded with full bit depth in order to provide full information. The blocks in the differential frames are encoded with a bit depth sufficient to code the maximum difference between this block and the corresponding block in the reference image. Since PS2.0 can handle up to four input textures and each can have a different bpp value, it is possible to achieve any required bpp for the block by distributing the values to multiple textures. It is worth mentioning that the block size should be selected in such a manner that the width and height of the image are multiples of this size – otherwise, the image must be padded with empty pixels (which results in poorer size reduction).

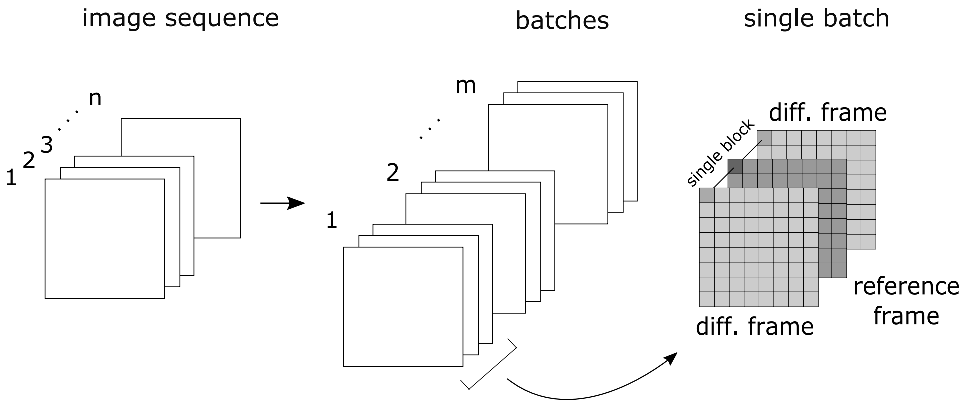

Figure 6. The proposed lossless compression method. A sequence of n images is divided into m batches. Each batch consists of a single reference frame and multiple differential frames. For legibility, batches of three elements are used (although, in general, the batches can have any arbitrary length). Each frame is additionally divided into blocks (subimages). Each block in each batch can use different bpp for coding differential frames.

4. Results and discussion

In order to properly verify the feasibility of the proposed methods, several experiments were conducted. For the performance experiments, we employed a low-end several-year-old machine with a configuration typical for a public healthcare hospital: 2GHz CPU, 2GB RAM, and an integrated graphics card compatible with the PS2.0 standard. In this section, we use size reduction (also known as the space savings) as the basic metric of compression efficiency, defined as:

$$size\ reduction = 1 - \frac{compressed\ size}{uncompressed\ size} \tag{2}$$

4.1. Lossy compression: DXT1-ps

Quality evaluation of the reconstructed DXT1-ps image sequences was performed by cardiologists from John Paul II Hospital in Krakow, Poland. Using the echocardiography data of their patients, they assessed the differences between the uncompressed and compressed image sequences. In their report, they stated that the differences are barely noticeable and that the compressed image sequences still provide proper diagnostic quality. Based on their opinion, we claim that this method is viable for use with ultrasound medical images.

Size reduction of the DXT1-ps method is independent from the contents of the image sequence. This depends mostly on the bit depth of the data and (to a minor degree) on the image size and length of the image sequence. The size reduction of a single block for both DXT1-ps and DXT1 is presented in Table 1. In the case of 24-bit RGB images, the size reduction of a single block in DXT1-ps is 83% (64 bits instead of 384), similar to basic DXT1. In the case of 8-bit grayscale images, the size reduction of a single block in DXT1-ps is 62,5% (48 bits instead of 128), which is better than in basic DXT1 (which treats grayscale images the same as RGB) and results in a 50% size reduction (64 bits instead of 128).

Both DXT1 and DXT1-ps work on 4-by-4 pixel blocks; therefore, the size reduction decreases slightly when the image width and/or height is not a multiple of 4 (since the image must be extended to the proper size with empty pixels). Nevertheless, this is rarely the case; also, the difference is barely noticeable, as the width and height of the image is much larger than 4. DXT1-ps compression ratio also decreases when the sequence length is not a multiple of 4 (or 2, in the case of the grayscale sequences); this is due to the joined colors part for consecutive frames. However, this does not constitute a problem since, for long sequences, the decrease is insignificant compared to the entire sequence size, and in the case of short sequences, the total size is small enough for the decrease to be irrelevant as well.

Table 1
Size reduction of a single block in DXT1-ps and DXT1.

	DXT1-ps	**DXT1**
24-bit RGB	83%	83%
8-bit Grayscale	62,5%	50%

The performance of the DXT1-ps method was measured on the test machine using the data provided by the cardiologists from John Paul II Hospital. An RGB sequence with the largest resolution (768×576) consisting of 76 frames was played fluently with 25 frame-per-second speed (using, on average, 2% of the CPU). This result is more than sufficient for use in telemedical applications, even when multiple sequences are loaded simultaneously.

4.2. Lossless compression

The main focus of the experiments with lossless compression was to find an optimal balance between size reduction and performance. Publicly-available datasets [25] (10 CT and 10 MR datasets) were employed. Several block sizes (16×16, 32×32, 64×64, 128×128) as well as several batch lengths (3, 5, 7, 9, 11) were tested with the given datasets. Tables 2 and 3 present the results for compression efficiency (expressed in averaged bpp) for the CT and MR datasets, respectively. As expected, smaller blocks resulted in better size reduction. In the case of batch length, longer batches provided more differential images and fewer reference images; on the other hand, however, maximum pixel difference increased with the frame distance, making differential coding less efficient. For better readability, the results are also plotted in Figure 7.

Table 2

Lossless compression: average bits per pixel of the compressed image sequences for the CT datasets.

	Block size			
Batch length	**16×16**	**32×32**	**64×64**	**128×128**
3	9,34	9,71	10,10	10,51
5	9,03	9,47	9,94	10,46
7	8,96	9,43	9,94	10,50
9	8,96	9,45	9,98	10,57
11	8,93	9,45	9,99	10,61

Table 3

Lossless compression: average bits per pixel of the compressed image sequences for the MR datasets.

	Block size			
Batch length	**16×16**	**32×32**	**64×64**	**128×128**
3	9,07	9,37	9,48	9,85
5	8,63	9,00	9,16	9,65
7	8,48	8,89	9,08	9,62
9	8,41	8,86	9,06	9,64
11	8,38	8,82	9,07	9,65

The performance of the lossless compression method depends not only on the selected block size and batch length but also on the size of the image. The performance measurement was done with a 512×512 image sequence, since this is the most common size in the employed datasets (13 out of 20 sets). Only two datasets had slightly larger images (576×576); the others had smaller images, with the smallest being 192×256. In the experiments, the sequences were played at 25 frames per second, which corresponds to the actions of quickly browsing the data or playing the data as a video. Table 4 presents average CPU usage, including standard deviation. It is obvious that smaller blocks result in much-greater CPU usage. One interesting fact is that longer batches require less CPU usage – this is due to the fact that changing the differential image is much quicker than changing the reference image. Apart from CPU usage, we also assessed how fluently the images were displayed. With 64×64 and 128×128 block sizes, the animation was fluent; but with smaller sizes (even though the CPU was running at 60–70%), the animation had short pauses, considerably degrading the user's experience.

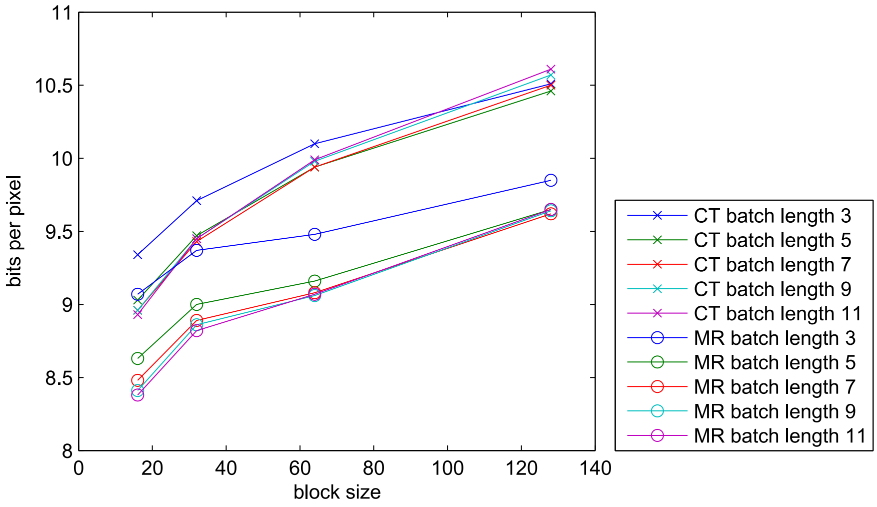

Figure 7. Lossless compression: average bits per pixel for the compressed image sequences for the CT and MR datasets.

Table 4

Lossless compression: average CPU usage [%] when playing a sequence of 512×512 images with 25 fps.

Block size	**16×16**	**32×32**	**64×64**	**128×128**
#Blocks	1024	256	64	16
Batch length 3	75,1 ± 14,0	63,3 ± 6,4	43,8 ± 4,6	35,9 ± 4,6
Batch length 5	66,0 ± 16,0	61,6 ± 8,4	44,5 ± 4,4	32,0 ± 6,0
Batch length 7	59,1 ± 17,1	60,6 ± 8,3	42,8 ± 4,2	29,4 ± 5,0
Batch length 9	53,9 ± 19,1	63,6 ± 13,2	41,8 ± 5,1	27,7 ± 5,5
Batch length 11	46,4 ± 16,7	57,9 ± 9,4	42,2 ± 6,4	27,1 ± 6,5

Based on the above data, we conclude that the optimal configuration is a block size set to 64×64 (64 subimages for a 512×512 image) and a batch length set to 9, as it provides fluent animation and good size reduction. It is worth noting that, even though they occupy only 12 bits, CT and MR images are typically stored in 16 bits, since most programming frameworks do not provide a 12-bit format. Therefore, our method allows for an overall average reduction of data size by 38% for CT data (9,98 bpp instead of 16) and 44% for MR data (9,06 bpp instead of 16). For comparison, state-of-the-art methods based on the CUDA framework [31] report achieving an average of 5,80 bpp for 10 CT datasets. Is it worth noting that a telemedical system may implement multiple algorithms and use them with regard to the available hardware.

5. Conclusions

In this work, we presented both lossy and lossless methods for the compression of medical image sequences in teleconsultation applications on medium and low- end devices. The methods provide real-time on-the-fly image decompression by employing parallel computations on the GPU. They require only compatibility with the PS2.0 standard, which is an important advantage over other parallel methods that require compatibility with CUDA or OpenCL frameworks – a feature not available on medium or low-end machines. The lossy compression method is based on the DXT1 format. The pixel shader implementation (DXT1-ps) enables an efficient cooperation with business application development frameworks (such as WPF) and provides better size reduction for grayscale images than the standard DXT1. The lossless compression method integrates differential coding between the frames and the division into blocks inside the frames, which allows us to take advantage of the similarities in 3D medical data, despite being limited to per-pixel computations, as required for employing the pixel shaders. Both methods provide significant size reduction with sufficient performance and allow for the storage of more imaging data in the internal memory. Additional verification with actual telemedical system TeleDICOM [3, 9] (for which the methods were designed) proved that they considerably improved the quality of medical teleconsultations.

Acknowledgements

This research was partially supported by grants no. POIG.01.03.01-00-008/08 and 11.11.230.124. We would like to express our thanks to prof. Andrzej Gackowski MD, PhD, as well as other physicians from John Paul II Hospital in Krakow for their invaluable help in evaluating our research.

References

[1] Bairagi V.K.: Symmetry-Based Biomedical Image Compression. *Journal of Digital Imaging*, vol. 28(6), pp. 718–726, 2015, `http://link.springer.com/10.1007/s10278-015-9779-3`.

[2] Chłopkowski M., Walkowiak R.: A general purpose lossless data compression method for GPU. *Journal of Parallel and Distributed Computing*, vol. 75, pp. 40–52, 2015, `http://dx.doi.org/10.1016/j.jpdc.2014.09.016`.

[3] Czekierda Ł., Malawski F., Wyszkowski P.: Holistic approach to design and implementation of a medical teleconsultation workspace. *Journal of Biomedical Informatics*, vol. 57, pp. 225–244, 2015.

[4] Czekierda Ł., Masternak T., Zieliński K.: Evolutionary approach to development of collaborative teleconsultation system for imaging medicine. *IEEE transactions on information technology in biomedicine: a publication of the IEEE Engineering in Medicine and Biology Society*, vol. 16(4), pp. 550–60, 2012, `http://www.ncbi.nlm.nih.gov/pubmed/22510953`.

[5] Franco J., Bernabe G., Fernandez J., Ujaldon M.: Parallel 3D fast wavelet transform on manycore GPUs and multicore CPUs. *Procedia Computer Science*, vol. 1(1), pp. 1101–1110, 2010.

[6] Gennari J.H., Weng C., Benedetti J., McDonald D.W.: Asynchronous communication among clinical researchers: a study for systems design. *International Journal of Medical Informatics*, vol. 74(10), pp. 797–807, 2005, `http://www.ncbi.nlm.nih.gov/pubmed/16023408`.

[7] Hosseini S.M., Naghsh-Nilchi A.R.: Medical ultrasound image compression using contextual vector quantization. *Computers in Biology and Medicine*, vol. 42(7), pp. 743–50, 2012, `http://www.ncbi.nlm.nih.gov/pubmed/22608347`.

[8] Hsu Y.C., Lin P.Y., Hsu S.J., Chan C.T.: Design and Implementation of Teleconsultation System for Instant Treatment. *2008 2nd International Conference on Bioinformatics and Biomedical Engineering*, pp. 1355–1358, 2008, `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4535548`.

[9] ICS-AGH: TeleDICOM. `http://www.teledicom.pl/index.php/en/`.

[10] ISO/IEC-15948:2004: Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=29581`.

[11] ITU: JPEG Standard – JPEG ISO/IEC 10918-1 ITU-T Recommendation T.81. `http://www.w3.org/Graphics/JPEG/itu-t81.pdf`.

[12] ITU: Recommendation T. 45: Run-length Colour Encoding. `http://www.itu.int/rec/T-REC-T.45-200002-I/en`.

[13] ITU: T.87 Information technology – Lossless and near-lossless compression of continuous-tone still images – Baseline. `http://www.itu.int/rec/T-REC-T.87/en`.

[14] Joint-Photographic-Experts-Group: JPEG2000. `https://jpeg.org/jpeg2000/index.html`.

[15] Khronos-Group: OpenCL. `https://www.khronos.org/opencl/`.

[16] Lasierra N., Alesanco A., Gilaberte Y., Magallón R., García J.: Lessons learned after a three-year store and forward teledermatology experience using internet: Strengths and limitations. *International Journal of Medical Informatics*, vol. 81(5), pp. 332–43, 2012, `http://www.ncbi.nlm.nih.gov/pubmed/22425394`.

[17] Lee J.S., Tsai C.T., Pen C.H., Lu H.C.: A real time collaboration system for teleradiology consultation. *International Journal of Medical Informatics*, vol. 72(1-3), pp. 73–79, 2003, `http://linkinghub.elsevier.com/retrieve/pii/S1386505603001308`.

[18] Lee J.W., Kim B., Yoon K.S.: CUDA-based JPEG2000 encoding scheme. *16th International Conference on Advanced Communication Technology*, pp. 671–674, 2014, `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6779047`.

[19] Luccichenti G., Cademartiri F., Pichiecchio A., Bontempi E., Sabatini U., Bastianello S.: User interface of a teleradiology system for the MR assess-

ment of multiple sclerosis. *Journal of Digital Imaging*, vol. 23(5), pp. 632–8, 2010, `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3046674{\&}tool=pmcentrez{\&}rendertype=abstract`.

[20] Miaou S.G., Ke F.S., Chen S.C.: A lossless compression method for medical image sequences using JPEG-LS and interframe coding. *IEEE transactions on information technology in biomedicine: a publication of the IEEE Engineering in Medicine and Biology Society*, vol. 13(5), pp. 818–21, 2009, `http://www.ncbi.nlm.nih.gov/pubmed/19447732`.

[21] MSDN: DXT1 texture format. `https://msdn.microsoft.com/en-us/library/windows/desktop/bb147243(v=vs.85).aspx`.

[22] MSDN: Windows Presentation Foundation (WPF). `https://msdn.microsoft.com/en-us/library/aa663364.aspx`.

[23] NEMA: DICOM standard. `http://dicom.nema.org/standard.html`.

[24] NVIDIA: CUDA. `http://www.nvidia.com/object/cuda_home_new.html`.

[25] OsiriX: DICOM sample image sets. `http://www.osirix-viewer.com/datasets/`.

[26] Ozsoy A., Swany M., Chauhan A.: Optimizing LZSS compression on GPGPUs. *Future Generation Computer Systems*, vol. 30(1), pp. 170–178, 2014, `http://dx.doi.org/10.1016/j.future.2013.06.022`.

[27] Pizzolante R., Castiglione A., Carpentieri B., Santis A.D.: Parallel low-complexity lossless coding of three-dimensional medical images. *Proceedings – 2014 International Conference on Network-Based Information Systems, NBiS 2014*, pp. 91–98, 2014.

[28] Sanchez V., Abugharbieh R., Nasiopoulos P.: Symmetry-based scalable lossless compression of 3D medical image data. *IEEE Transactions on Medical Imaging*, vol. 28(7), pp. 1062–1072, 2009.

[29] Sriraam N., Shyamsunder R.: 3-D medical image compression using 3-D wavelet coders. *Digital Signal Processing*, vol. 21(1), pp. 100–109, 2011, `http://linkinghub.elsevier.com/retrieve/pii/S1051200410001442`.

[30] Wang B., Govindan P., Gonnot T., Saniie J.: Acceleration of ultrasonic data compression using OpenCL on GPU. In: *2015 IEEE International Conference on Electro/Information Technology (EIT)*, pp. 305–309, 2015.

[31] Weinlich A., Rehm J., Amon P., Hutter A., Kaup A.: Massively parallel lossless compression of medical images using least-squares prediction and arithmetic coding. *2013 IEEE International Conference on Image Processing*, pp. 1680–1684, 2013, `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6738346`.

[32] Wu Y.G., Tai S.C.: Medical image compression by discrete cosine transform spectral similarity strategy. *IEEE Transactions on Information Technology in Biomedicine*, vol. 5(3), pp. 236–243, 2001.

Affiliations

Filip Malawski
AGH University of Science and Technology, Department of Computer Science, Krakow, Poland, fmal@agh.edu.pl

Łukasz Czekierda
AGH University of Science and Technology, Department of Computer Science, Krakow, Poland, luke@agh.edu.pl

Received: 30.05.2016
Revised: 5.09.2016
Accepted: 6.09.2016