Advanced mixing audio streams for heterogeneous computer systems in telecommunications

A.A. Kolpakov[1], Ju.A. Kropotov[1]

[1]*Murom institute (branch) VlSU, Orlovskaya st., 23, 602264, Murom, Vladimir region, Russia*

Abstract

This paper presents an algorithm enhanced mixing of audio streams for computation on GPUs, which combines multiple stages of mixing by using two-pass rendering, which significantly reduces the switching time between buffers. Methods of computer experimental comparative studies were carried out evaluating the performance of the developed algorithm. The purpose of the present paper is development of an efficient algorithm for mixing audio streams for processing on GPUs. The method of transfer operations computing on graphics processors with the use of Shader programs was developed. Novel features presented solutions is using a two-pass rendering. The results showed that the application of the developed algorithm leads to an increase in computational performance up to 6 times. Presented solution can be implemented as software in the telecommunications multiprocessor systems.

Keywords: two-pass rendering; algorithm for increasing productivity; parallel computing; heterogeneous computing systems; graphics processors; mixing of audio data

1. Introduction

The development of information and telecommunication systems of digital in-house operational, command, loudspeaker and telephone communication on their basis becomes especially urgent with the development of modern computer networks. It is justified to use panel or tablet computers based on ready-made solutions to ensure optimum performance when designing such systems. Since voice communication plays an important role, the voice conferences mode with several participants is the main one in such systems.

The simplest scenario of organizing voice communication is that each sound source sends its audio stream to each receiver independently. This method is simple and convenient, but it requires high network bandwidth, which is not always possible. Therefore, the best method is audio mixing, which means combining the audio streams from each source into one. Based on the possibility of sound waves superimposed on each other, this method can provide an acceptable sound quality, while ensuring a decrease in network congestion. However, the application of this method can significantly increase the load on the server CPU, which can negatively affect the overall system performance. The way out of this situation may be the use of GPUs to solve this problem [1].

2. The problems with using graphics processors for audio mixing

Although the graphics processors are quite efficient, there are several problems in using them for audio mixing, which are related to the architecture and the limited functionality [2].

The first problem is the bus bandwidth between the GPU and the main memory, which is smaller than between the main processor and the main memory. For example, the Intel 975X chipset provides theoretical bandwidth for a CPU of 10.7 GB/s, and for the GPU only 8 GB/s. Practice shows that the lack of support for asynchronous I/O requires a lot of time for additional operations, such as locking / unlocking the buffer. Since the general calculations on the GPU are based on 3D rendering, the write speed is usually higher than the read speed. This asymmetry makes the procedure for reading the result sufficiently long [3, 4].

Secondly, the general calculations on the GPU are based on 3D models, different tasks require different GPU settings, such as 3D models, transforming matrices and shader programs. During the loading of the settings, the computational flows of the GPU are not involved. Worst of all, the GPU does not tell the CPU when the task is completed, so the CPU must periodically check the status of the GPU. This is quite a time-consuming operation, as it breaks the parallelism between the GPU and the CPU.

Third, the disadvantage of the GPU is its performance in logical operations. As you know, CPU tracks branches, GPU works differently: each branch is first executed, and then the desired result is selected. This makes parallelization easier, but requires more resources.

Finally, the GPU instruction set is incompatible with the CPU. In addition, execution time and code length are limited. All this makes it difficult to transfer existing algorithms to graphics processors [5].

3. The structure of the developed algorithm

The basic algorithm of audio mixing consists of five steps. The first step is to summarize the audio samples from different sources, which can be represented by the following formula [6,7]:

$$\overrightarrow{M}_t = \sum_{k=0}^{n} \vec{u}_{k,t}, \quad (1)$$

где $\vec{u}_{k,t}$ – the sample vector; i.e. the vector of samples obtained by microphone *k* in time *t*;

$\overrightarrow{M}_t$ – resulting mixing vector.

The second stage is echo cancellation, which in its basic form consists in excluding the sample of the *i*-th device from the final vector [8]. This stage is represented in the form

$$\overrightarrow{M}_{i,t} = \overrightarrow{M}_t - \vec{u}_{i,t}, \quad (2)$$

где $\overrightarrow{M}_{i,t}$ – resulting mixing vector for the *i*-th device;

$\vec{u}_{i,t}$ – sample of the *i*-th device.

For the correctness of the resulting vector $\overrightarrow{M}_{i,t}$, it is necessary that its dimension be equal to the dimension of the incoming vectors. However, after stages 1 and 2, the vector $\overrightarrow{M}_{i,t}$ may be overcrowded, leading to unwanted noise [9]. In order to use the original vector of large dimensions $\overrightarrow{M}_{i,t}$, it is necessary to compress it for further use. This is done in the third stage by the formula

$$\overrightarrow{M}_{i,t} = \overrightarrow{M}_{i,t} \times frac_{i,t}, \quad (3)$$

where $frac_{i,t}$ – attenuation factor for the *i*-th device. This coefficient must be calculated automatically, starting from the maximum compressed sample. The search for the maximum among the compressed samples occurs in the fourth stage, and the correction of the attenuation coefficient is made on the fifth stage.

A block diagram of the basic mixing algorithm is shown in fig. 1.

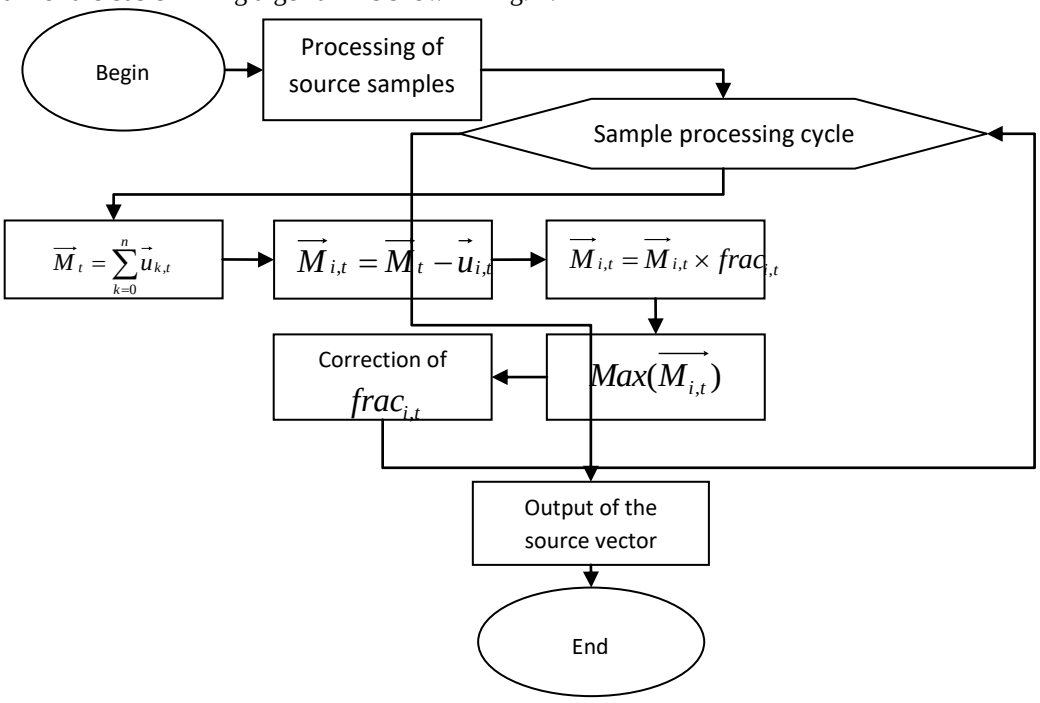

Fig. 1. A block diagram of the basic mixing algorithm.

For a better description of the algorithm, the GPU capabilities are represented as an *f*-function of the *X* texture and pixel coordinates represented by the formula

$$f(X,P) = \{y_i \mid \forall p_i \in P_i y_i = f(X, p_i)\}, \quad (4)$$

где y_i – projection of pixel p_i on the texture *X*,

P – 3D model projection.

For each pixel in the 3D model projection *P*, the function (4) will be calculated and then the result will be written to the render buffer.

From the formula (4) the sample of calculations on GPU can be presented as *A*=(*X*,*P*,*f*).

Suppose that *n* is the total number of sound sources in the session, and *L* – the length of one audio sample. Each of the first three mixing steps (sample accumulation, echo cancellation and compression) yields *n* sequences of *L* bytes. At the same time, the last two steps (searching for the maximum sample and adapting the attenuation coefficient) yield only *n* integers. Since the first three steps can be performed within one projection for computing on GPU, they can be combined into one step, as shown below

$$m_i = \left(\sum_{j=0}^{n-1} u_j - u_i \right) \times frac_i . \quad (5)$$

Since the fourth step uses a different measurement than the first three, it cannot be combined within the framework of formula (5). A block diagram of the extended mixing algorithm is shown in fig. 2.

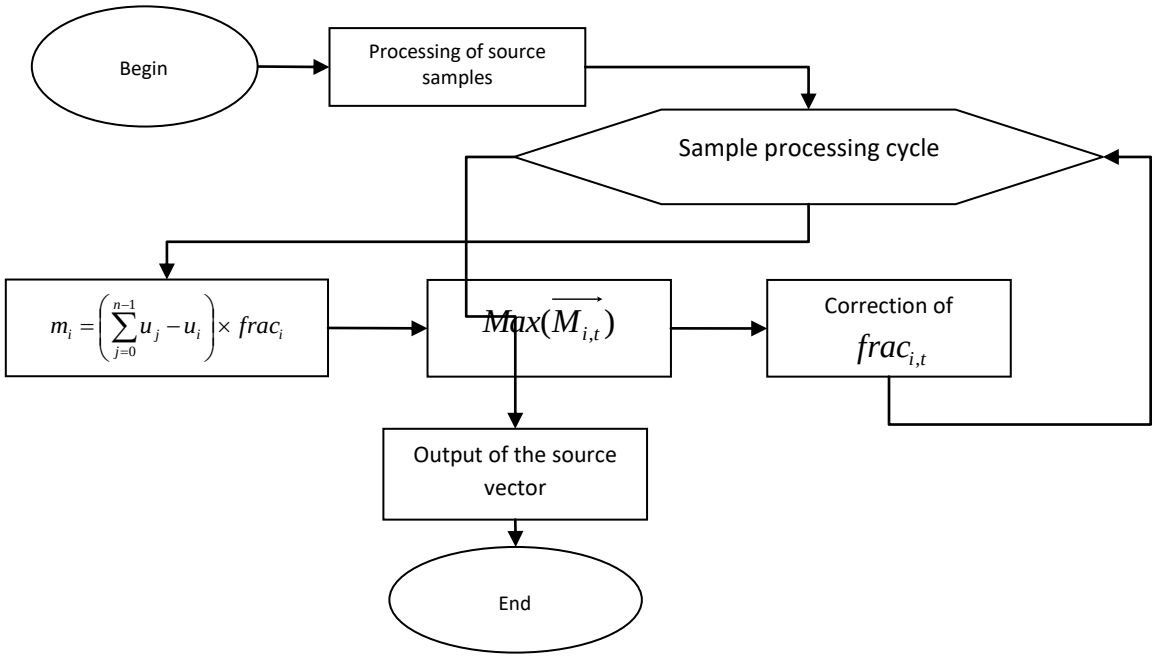

Fig. 2. A block diagram of the extended mixing algorithm.

Usually, calculations can be combined according to general criteria when their projections do not coincide and they do not have intersections. Therefore, if these calculations are denoted as *A1*=(*Xl*,*P1*,*f1*) and *A2*=(*X2*, *P2*, *f2*), then they can be combined as shown below.

$$X = \langle X_1, X_2 \rangle, P = P_1 \cup P_2, \; f(\langle X_1, X_2 \rangle, p) = \begin{cases} f_1(X_1, p), p \in P_1, \\ f_2(X_2, p), p \in P_2. \end{cases} \quad (6)$$

As seen from (6), the main algorithm is checking the coordinates of each pixel to select the module's execution function. Since the GPU executes all branches before selecting the desired one, each branch will be executed for each pixel, which will take a long time.

The developed algorithm presents an alternative method for executing a large number of functions, which outputs data of different lengths to a single render buffer using multithreaded rendering. Here the main rule is to move the projection by modifying the projection matrix, so that the computational area of each function is limited to the required area, rather than the entire buffer.

Depending on the pixel, it can have a different amount of information without loss of versatility. *w* denote the total number of pixels needed to store L bytes. Thus, the render buffer can be represented as (*w+1*)*n*. 3D model of the developed algorithm is a rectangle, which lies in the plane Z. It has the dimensions: *2w/*(*w+1*) units in width and 2 units in height. Coordinates of vertices – (-1,-1,0), (1-2/(*w*+1),-1,0), (-1,1,0), (1-2/(*w*+1),1,0).

At the first rendering pass, a unit matrix is chosen as the projection matrix. This ensures that the projection matches the 3D model. After converting the visible area, in this pass the formula (5) is applied to perform the first three steps of the mixing. Transformations of the 3D model, produced in the first three steps, are shown in Fig. 3.

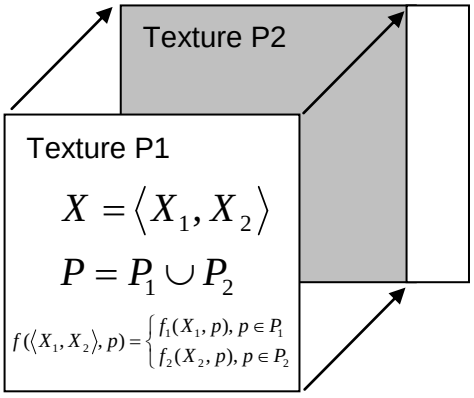

Fig. 3. The first pass of the algorithm.

In the second rendering pass, the projection is shifted to the left by *L* pixels. At the same time, the shader program switches to the search mode of the maximum sample. In this pass, only one column can be written, since most parts of the projection lie outside the render buffer and will be automatically ignored by the GPU. Since the clipping of the projection was performed at the beginning of the render, this method calls the function only for the correct pixels, and not for the entire buffer. The actions performed on the second pass of the algorithm are shown in fig. 4.

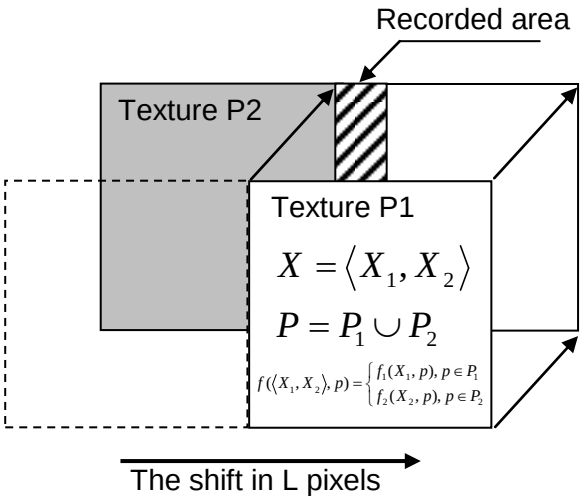

Fig. 4. The second pass of the algorithm.

4. Using a single-texture algorithm

As noted above, both pass algorithms requires as input data samples every *n* sequences. Each sequence must have its own unique texture. Total needed *n* texture dimension *L*. This multi-textural technology is not suitable for audio mixing. First, each sound source requires its own texture. Therefore, additional passes may be required. Secondly, loading a lot of small textures is much slower than loading a single large one.

In the developed algorithm, it is proposed to load a single texture. As an example, the first pass of the algorithm is presented. The input contains *n* samples of sequences of size *L* bytes and *n* attenuation coefficients. Two textured RGBA format buffers are used. Texture T1 has the dimension [*L*/4]**n* and stores all sample sequences in a line. Texture T2 has a dimension of 1**n* and stores the attenuation coefficients. The coordinates of all the textures are given in Table 1.

Table 1. Coordinates of textures used in the developed algorithm.

Vertexes	Coordinates of texture T1	Coordinates of texture T2
(-1, -1, 0)	(1/2*w*, 1)	(0.5, 1)
(1-2/(*w*+1), -1, 0)	(1, 1)	(0.5, 1)
(-1, 1, 0)	(1/2*w*, 0)	(0.5, 0)
(1-2/(*w*+1), 1, 0)	(1, 0)	(0.5, 0)

Audio mixing is performed independently for each pixel. The pixel's texture coordinates (*x, y*) are calculated by interpolating the vertex texture coordinates. Based on Table 1, the texture coordinate for T1 or pt.t0 should be (*X, Y*), and for T2 or pt.t1 – (0.5, *Y*). Pt.t0 refers to the sample for which the current mixing transformations are made, this sample is called the "aiming point". To exclude the appearance of an echo, the other texture coordinate of ptCur is restricted by accessing T1 instead of pt.t0. The *x* component of the ptCur texture is identical to the pt.t0 texture, and the *y* component is calculated from a cyclic variable that designates each sound source. In the loop, the position register *v* is used to skip the "aiming point". Finally, the attenuation coefficients are read from the texture pt.t1. Since the coefficient is stored in the first byte, the sample is produced only by the blue component.

5. Experimental study of the developed algorithm

Test input data for mixing are sequences of 320 samples. All samples are generated randomly. Test bench configuration: CPU Intel Core i3-4130, 4 GB RAM, graphics card NVIDIA GeForce GT730. The number of *M* sequences was varied. The results for the basic and developed algorithms at the output are identical. The results of an experimental study of the dependence of the mixing time *t* on the number of sequences *M* are shown in fig. 5.

As can be seen from the test results shown in Fig. 5, application of the advanced algorithm of audio mixing allows to increase productivity of computer system in 5-6 times [9].

Table 2 shows the results of measuring the average running time of the algorithm t for 8 random sequences of 320 samples. During the measurement, 1000 processing cycles were carried out, for each of which new random sequences of samples were

generated. For the results obtained, the dispersion of the output sequences is calculated, the values of which are also given in Table 2.

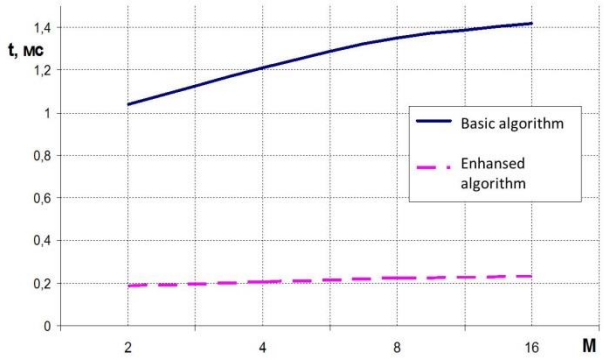

Fig. 5. The results of an experimental study of the dependence of the mixing time *t* on the number of sequences *M*.

Table 2. Results of the study of the algorithm for extended audio mixing for 8 sample sequences.

The used algorithm	Average time of the algorithm, ms	Dispersion of output Sequences
Basic algorithm	1,351	3,757
Developed algorithm	$2{,}226 \times 10^{-1}$	$1{,}426 \times 10^{-3}$

As can be seen from the test results presented in Table 2, the variance of the output sequences for the basic algorithm is substantially higher than for the developed one [12,13]. This is due to the fact that 32-bit numbers are used for the calculations in the GPU, whereas for the calculations on the central processor – 64-bit. Application of the developed algorithm in a heterogeneous computer system reduces the processing time to 0.22226 x 10-3 s instead of 1.351 x 10-3 s – the time of data processing by the basic algorithm.

6. Conclusion

This paper presents an algorithm for the extended downmix audio streams for computing on GPU. Its main advantage is the combination of multiple stages of mixing by using a two-pass rendering, which significantly reduces the switching time between buffers. The use of one texture for calculations increases the efficiency of I/O operations. Although I/O operations take approximately half the computation time, experimental studies of the developed algorithm showed an increase in performance up to 6 times.

References

[1] Lindholm E, Nickolls J , Oberman S , Montrym J. NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 2008; 28(2): 39–55.
[2] Luebke D, Harris M, Kruger J, Purcell T, Govindaraju N, Buck I, Woolley C, Lefohn A. GPGPU: general purpose computation on graphics hardware. ACM SIG-GRAPH. New York, USA: Course Notes, 2004; 33 p. DOI: 10.1145/1103900.110393.
[3] Kolpakov AA. Theoretical evaluation of growth performance computing systems from the use of multiple computing devices. V mire nauchnykh otkrytii 2012; 1: 206–209. (in Russian)
[4] Boreskov AV. Shaders development and debugging. Sankt-Peterburg: BHV-Peterburg, 2006; 488 p. (in Russian)
[5] Nekrasov KA, Potashnikov SI, Boyarchenkov AS, Kupryazhkin AYa. Parallel computing for general purpose graphics processors: text book. Ministry of Education and Science of the Russian Federation, Ural Federal University. Ekaterinburg : Publishing house of the Ural University, 2016; 104 p. (in Russian)
[6] Kropotov JuA. Experimental study of the law of distribution of probability of amplitudes of signals of systems of transmission of voice information. Proektirovanie i tekhnologiia elektronnykh sredstv 2006; 4: 37–42. (in Russian)
[7] Belozyorov AS, Korobicyn VV. Implementation of computations on a graphics processor using Nvidia CUDA platform . Programmnye produkty i sistemy 2010; 1: 62–64. (in Russian)
[8] Kropotov JuA, Belov AA, Proskuryakov AJu, Kolpakov AA. Methods of Designing Telecommunication Information and Control Audio Exchange Systems in Difficult Noise Conditions. Sistemy upravleniia, sviazi i bezopasnosti 2015; 2: 165–183. (in Russian)
[9] Ermolaev VA, Kropotov JuA. Algorithms for processing acoustic signals in telecommunication systems by local parametric methods of analysis. Proceedings International Siberian Conference on Control and Communications (SIBCON), 2015. URL: http://ieeexplore.ieee.org/document/7147109/.