5 ТЕОРИЯ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ЯЗЫКИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Fominykh M., Smorkalov A., Morozov M.

EXTENDED STREAM PROCESSORS TEXTURE GENERATION MODEL FOR 3D VIRTUAL WORLDS: EVALUATION RESULTS

Review: *In this paper, we present an extended stream processors texture generation model for displaying educational content in 3D virtual worlds. The model suggests conducting image-processing tasks on stream processors in order to reduce the load on CPU. The main objective of the paper is to provide the evaluation results of the suggested extended model based on a series of tests. The extension of the model consists of using fixed pipeline features of stream processors. The obtained results of performance evaluation confirm high efficiency and veracity of the generalized mathematical and programming models for image processing. High performance can be explained by specificity problem of generating educational content for virtual words because the source data for the synthesis of images and the data area for the resultant images are in the local memory of stream processors.*

Аннотация: *В этой статье представлена расширенная модель генерация текстур на потоковых процессорах для отображения образовательного контента в трехмерных виртуальных мирах. Модель предполагает перенос выполнения задач по обработке изображений на потоковые процессоры с целью уменьшения нагрузки на центральный процессор. Главная цель статьи предоставить результаты апробации предложенной расширенной модели, основанные на серии тестов. Расширение модели включает в себя использование возможностей фиксированного конвейера потоковых процессоров. Полученные результаты оценки производительности подтверждают высокую эффективность и достоверность расширенной математической и программной модели для обработки изображений. Высокая производительность объясняется спецификой задачи генерации образовательного контента для виртуальных миров, т.к. исходные данныедля синтеза изображений и блок памяти для результирующих изображений располагаются в локальной памяти потоковых процессоров.*

Keywords: *3D virtual worlds, image processing, stream processors, educational content, vAcademia, performance evaluation, mathematical model, programming model, performance, image synthesis*

Ключевые слова: *трехмерные виртуальные миры, обработка изображений, потоковые процессоры, образовательный контент, vAcademia, оценка производительности, мате-*

DOI: 10.7256/2305-6061.2014.3.13384

матическая модель, программная модель, производительность, синтез изображений

1. Introduction

Three-dimensional virtual worlds (3D VWs) provide a unique set of features that can be used for serious applications, such as learning and collaboration. Such features include low cost and high safety, 3D representation of learners and objects, interaction in simulated contexts with high immersion [1, 2]. Processing large amounts of images in 3D VW is often required when working on serious tasks. In other tasks, displaying images, video or interactive elements is also required; however, the amount of the content is smaller. Usually, an image is calculated on a CPU on client side (e.g., in Second Life™ and Blue Mars™) or server side (e.g., in Open Wonderland™) and then loaded into the stream processor memory as a texture.

Stream Processors (SPs) are specialized processors characterized by a very high data parallelism [3]. SPs are most widely applied in graphics adapters, and therefore, their main tasks are related to processing 3D graphics, such as a high-efficiency rasterization of polygons with texture mapping, fast affine transformations of the vertex data flow, interpolation of scalars, vectors and matrices on the surface of polygons, and calculation of lighting.

Due to the focus on the highly parallel computing tasks in 3D graphics, these devices have many hardware constraints [4, 5] and their cores have relatively simple architecture [6]. Therefore, most of the classical algorithms that can be executed on CPU cannot be executed on SPs without modification.

The work presented in this paper is devoted to designing and implementing a method for texture generation of educational content using SPs. Although image processing and texture generation on SPs has been used in computer games for several years, we consider textures of educational materials in particular that are characterized by meaningful content aimed by a teacher. In addition, we present an extended mathematical and a programming models for the method. Using programmable and fixed pipeline abilities of stream processors give us ability to provide effective implementation of wide range of educational tools for 3D virtual world vAcademia. In terms of scientific contribution, this work can be interpreted as solving a new problem using a well-known method.

2. Stream Processors Texture Generation Model

In the modern systems, many image-processing tasks are not suitable for being calculated on CPU, as it is loaded with other tasks or excessive computation time is required. In addition, processing some of such tasks using CPU is inefficient, as the source data for the synthesis of images and the data area for the resultant images are in the local memory of other devices. Usually, the data communication between the main and the device memories is done through the data bus, which has a limited capacity. The data-communication rate limits the performance

DOI: 10.7256/2305-6061.2014.3.13384

of the image processing using stream processors significantly.

The types of tasks described include, for example, image processing for subsequent use as textures for rendering 3D scenes in virtual environments. 3D visualization in such applications is hardware-based and conducted on SPs. The source data are in the local memory of the graphics card and the CPU heavily loaded with calculations related to the maintenance of the virtual environment.

This implies that processing images using the capabilities of SPs directly can be efficient, especially given the fact that their computing power usually exceeds the capabilities of CPUs tenfold. However, the SPs have some serious hardware limitations due to their architecture. These limitations do not allow to use them for implementing most of the classical image processing algorithms. Some of them require completely new approaches.

Thus, there is a need for a theoretical framework in image processing that considers the limitations imposed by SPs. In addition, software tools for modifying the algorithms (without the necessity for a deep understanding of their architecture) to be executed on the SPs are needed.

In order to formalize the domain, we have developed the mathematical model of image processing on SPs, based on the specifics of the SP architecture and hardware constraints. The mathematical apparatus of processing 3D graphics on SPs was simplified to focus only on processing images.

The model introduces the basic terms, objects, and their transformations. An image is represented in the RGBA format.

$$U(x, y) = \{f_R(x, y), f_G(x, y), f_B(x, y), f_A(x, y)\} \quad (1),$$

where $f_R(x, y), f_G(x, y), f_B(x, y), f_A(x, y)$ are discrete functions defined by tabular procedure and corresponding to the color channel with values in the range [0, 1].

The result of transformation G of image A based on image B is a modification of the image function (1):

$$R = G(A, B, x, y) \quad (2),$$

A geometrical figure is defined as a set of two-dimensional vectors of vertices V, a set of indexes of vertices F, and a color in $\{r, g, b, a\}$ format.

$$S = \{V, F, \{r, g, b, a\}\} \quad (3),$$

Rasterization is a transformation of a geometrical figure that has an image as a result.

$$U(x, y) = G_R(G_P(S, M_P)) \quad (4),$$

where G_R is a rasterizing transformation, M_P is a projective matrix, and G_P is a projection transformation.

DOI: 10.7256/2305-6061.2014.3.13384

The result of a projection transformation G_P is a projected figure.

$$S_P = G_P(S, M_P) \quad (5),$$

In addition, we applied the mathematical formalization to the configurable functionality (of the SPs) that was suitable for image processing tasks. As one of the specifics, SPs have some configurable (not programmed) functionality, which was formalized and linked to the programmed functionality. This included a mechanism for texture sampling, color mask, hardware cut of the rasterization area, hardware-based blending of the source image and the rasterized image.

The suggested model allows defining the color of each pixel of the resultant image separately as the resultant image is the function of pixel coordinates. This makes it possible to calculate parts of an image or even single pixels instead of the whole image. The general nature of the model allows comparing the efficiency of different approaches to any specific image processing task, such as dynamic texture generation, using the formula for image generation time:

$$T = T_C + T_{TR} + T_1 * W * H \quad (6),$$

In this equation, T – is the time of processing the whole image, T_C – compilation time of the image processing program (shader) that performs a transformation, T_{TR} – time of preparation for transformation, T_1 – time of calculating the color of one pixel, i.e. calculating (2), and W and H – width and height of the processed image.

Hardware scissors of the rasterization area defines a rectangular area of an image so that rasterization is applied within the boundaries and not applied outside, cutting the rest of the pixels. This principle allows extending (2) as follows:

$$R = G(A, B, x, y, x_1, y_1, x_2, y_2) = \begin{cases} G(A, B, x, y), \ if \ \ x \geq x_1, x \leq x_2, y \geq y_1, y \leq y_2 \\ A(x, y) \ otherwise \end{cases} \quad (7),$$

Another configurable feature of SPs is hardware blending of the original image and the rasterized figure [7]. Let A be the original image, B – the rasterized figure, and R – the resultant image. Then hardware mixing principle can be displayed by the following:

$$R_Q(x, y) = \min(1, \max(0, A(x,y)_Q * Ks_Q + B(x,y)_Q * Kd_Q)) \quad (8),$$

In this equation, Q is the color or alpha channel, Ks_Q is the mixing coefficient of the original image A, Kd_Q in the mixing coefficient of the image of the rasterized figure B.

The most popular algorithm that uses hardware mixing is alpha blending. This algorithm can be described by the following equation:

$$R_Q(x, y) = \min(1, \max(0, A(x,y)_Q * B(x,y)_A + B(x,y)_Q * (1 - B(x,y)_A))) \quad (9),$$

DOI: 10.7256/2305-6061.2014.3.13384

In this equation Ks_Q is equal to $B(x,y)_A$ – the value of the alpha channel of the rasterized figure, while Kd_Q is equal to the inverted value of $B(x,y)_A$. This coefficients corresponds to the hardware transformation with the following parameters: source factor = SRC_ALPHA, destination factor = ONE_MINUS_SRC_ALPHA, and blend mode = FUNC_ADD.

The mathematical model presented above was used as a base for the programming model and architecture based on four main objects (Texture, Drawing Target, Filter, and Filter Sequence) and a limiting condition <β>.

Texture is an image in format (1) stored in the SP memory. The image can be loaded to a texture and obtained from a it asynchronously (using extension GL_ARB_pixel_buffer_object [8]) with the possibility to check data availability, reducing the expenses of the communication through the data bus.

Drawing Target is an object that defines the resultant image, color mask, scissors, blend settings and settings of other configurable features of SPs.

Filter is a subroutine with the main function GetColor that defines the image transformation (2) and returns the color of a point for the given coordinates. This function has predefined and custom (user-defined) parameters. The GetColor function is defined in GLSL-like language extended with additional functions for image processing. Its parameters are strongly typed, have unique names, and are defined in the form of XML. GetColor provides the multipurpose model, allowing, for example, to program texture generation as a method of procedural materials and as a method of full periodical texture regeneration.

Filter Sequence (FS) is a sequence of Filters with parameters to be used for complex transformations.

<β> is a limitation introduced to the model for securing independent parallel processing of any image blocks.

The programming model is illustrated in the example below. Images are processed by applying FS which contains two consequently working Filters with mutually changing source image and resultant image (Fig. 1). Texture T1 is the parameter of Filter F1 which is applied to the Drawing Target DT. Texture T2 is the resultant image of applying the F1 to DT. At the same time, T2 is the parameter of F2 which is applied to DT. T1 is the resultant image of applying the F2 to DT. First, F1 is applied, then F2. The source image is taken from T1, while the resultant image becomes the same.

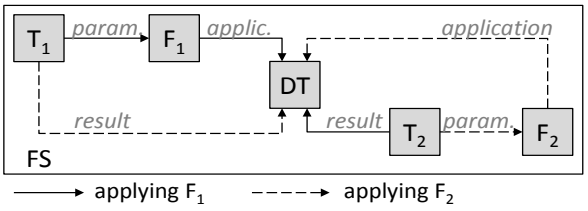

Figure 1 The relation of the objects of the model, Filter Sequence example of ping-pong technique

DOI: 10.7256/2305-6061.2014.3.13384

When using the suggested programming model, and on the first use of the transformation (Filter), a translation of the subroutine that converts an image from an XML description and extended GLSL into a program in standard GLSL is required. On that basis, formula (6) needs an additional component – translation time T_T

$$T = T_T + T_C + T_{TR} + T_1 * W * H \quad (10),$$

If the Filter is simple, $T_T + T_C$ may be much less than $T_{TR} + T_1 * W * H$. This limitation was eliminated by using extension GL_ARB_get_program_binary [9] that allows using pre-compiled shader programs. Moreover, not only the source code is translated into binary data, but also all related attributes of the XML description of the. This allows excluding translation and compilation stages from the process of loading the image processing program (shader).

Applying this method, translation and compilation of the Filter's code can be done only once e.g., when installing the software. In this case, $T_T + T_C$ in formula (10) is replaced by the time of loading a pre-compiled shader program T_{CL}:

$$T = T_{CL} + T_{TR} + T_1 * W * H \quad (11),$$

In this equation, $T_{CL} << T_T + T_C$

Further, we describe an original modification of the Discrete Wavelet Transformation (DWT) algorithm to run on SPs based on the models described above. We implemented the same algorithm to run on CPU for the case of inability to use the SP. We applied the method of 2D DWT filter cascade. It enabled dividing the forward DWT into two phases, in contrast to the lifting scheme which would require more phases and consequently more time. We implemented it using a FS consisting of two Filters.

The first phase in implemented as a complex Filter. The color space is transformed from RGB to YUV. Then, image components that contain color values (U and V) are scaled down N times, averaging the color. It allows compressing the image with minimum effect, as the human eye distinguishes the color differences worse than brightness (luminance). Parameter N is a positive integer that defines the image quality (quality grows when N is reduced). In the end of the first phase, the values of chrominance components of each pixel are changed in the way that none of them satisfies (12).

$$C_Q / D = 0 \quad (12),$$

In this equation, C_Q is the value of chrominance component and D – color quantization coefficient.

The second phase starts with applying 2D DWT FS for each color component. Then, the values of all components of the acquired image are quantized using coefficient *D*. The criteria for rejecting a pixel with coordinates (x,y) as insignificant on cascade step l is satisfying (13) or (14). The choice between (13) and (14) is made based on minimizing l. If l has the same value, (13) is used. When the checking described above is running, no conflicts with other DWT filter cascade steps occur, as the limitation <β> ensures that the resultant image and parameter

DOI: 10.7256/2305-6061.2014.3.13384

images are different.

$$\begin{cases} \left| C(x,y) - (C(x - 2^{I+1}, y) + C(x + 2^{I+1}, y)) \right| < E \\ \text{x mod } 2^{I+1} = 2^{I} \end{cases} \quad (13),$$

$$\begin{cases} \left| C(x,y) - (C(x, y - 2^{I+1}) + C(x, y + 2^{I+1})) \right| < E \\ \text{y mod } 2^{I+1} = 2^{I} \end{cases} \quad (14),$$

In these equations, $C(x,y)$ – is the value of the processed color component at the point (x, y) and E – acceptable variation of the approximated value of the pixel's color component from its true value.

If the value of the pixel's color component is rejected, it is zeroed (zero is written into the resultant image). As a prediction operator, we use a linear interpolation function. In the end of the second phase, the values of the color components are quantized with coefficient D.

$$C_Q = C_Q / D \quad (15),$$

For each pixel, the algorithm is finding the minimal cascade step I that satisfies (13) or (14) by direct enumeration, which is justified as the cost of superfluous arithmetical operations is significantly less than that of additional phases (which corresponds to the time TTR in the mathematical model describer above).

The inverse DWT should consist of (2 * K + 1) and (K + 1) phases for 2D and 1D cases correspondingly, where K is the number of DWT cascade steps).

In 1D case, the first K phases are a FS that implements K steps of the filter cascade of the inverse DWT. As it is necessary to use the results of earlier steps on the later steps and <β>, the inverse DWT cannot be implemented in a single phase. Instead, we use a ping-pong FS (Fig. 1), applying the same Filter with various cascade steps I and interleaved source and resultant images for all K phases.

If an integer I that satisfies the following equation exists,

$$x \bmod 2^{I+1} = 2^{I} \quad (16),$$

the color of the pixel with coordinates (x,y) for the step I is defined by the following equation.

$$C(x,y) = \begin{cases} C(x,\ y),\ if\ C(x,\ y)\ >\ 0 \\ (C(x - 2^{I+1}, y) + C(x + 2^{I+1}, y) / 2, if\ C(x,y) = 0 \end{cases} \quad (17),$$

During the last phase, the UV components are dequantized and the initial sized of them are recovered. The color space is transformed from YUV back to RGB.

Modifying formula (11), we can get the formulas for the wall time for the forward DWT for the lifting scheme:

$$T = T_{TR} * (K + 1) + T_1 * W * H \quad (18),$$

DOI: 10.7256/2305-6061.2014.3.13384

and for the filter cascade scheme:

$$T = T_{TR} + (T_1 + T_2) * W * H \quad (19),$$

In this equation, T_2 is the average calculation time for minimal I that satisfies (13) or (14) for a given pixel. As the resource intensiveness of T_2 is small, $T_2 * W * H$ is significantly smaller than $T_{TR} * K$.

The mathematical model suggested for efficient rasterization of attributed vector figures [10, 11]. Geometric figure cannot be rasterized for its analytical description using SPs without shader programs, as SPs are able to deal only with vertexes and triangles. Theoretically, geometrical figure can in fact be rendered by its analytical description using shaders, however, the efficiency of such approach is low, as the area of the rendered figure can be much smaller than the area of rasterized triangulated figure. The efficiency remains low even if using a bounding box instead of the actual figure.

3. Interactive Virtual Whiteboard of vAcademia

Interactive virtual whiteboard (VWB) is the main tool (or a container of tools) for collaborative work on 2D graphical content in vAcademia. The mathematical and the programming models presented in the previous section were used for implementing the VWB tools. The tools classified into three groups by the method of generating the resultant image: Sharing Changing Blocks, Sharing Attributed Vector Figures, and Processing Static Images [10].

Multiple VWBs of different sizes can be set up in any location of vAcademia (Fig. 2). In addition, every participant can set up an extra VWB during a class and present some content on it. Multiple users can stream or share their content simultaneously by simple mechanisms such as drag-and-drop. A colored laser pointer can be used for focusing attention on a certain element of the board. Additional auxiliary mechanisms allow user to switch easily between the displayed data for better overview.

Figure 2 Multiple VWBs and virtual laser pointers in use

vAcademia has a dedicated interface for placing objects in virtual locations (Fig 3). VWBs

DOI: 10.7256/2305-6061.2014.3.13384

of different sizes and designs are stored in the Object Gallery and available for every user [12]. User-created whiteboards are put on the ground, but can be moved and rotated (Fig. 3). VWBs that are part of Standard Locations cannot be relocated during a live session, however, they can be modified in advance and saved as templates to be used in specific settings (for example, a large number of whiteboards or specific placement pattern) [12].

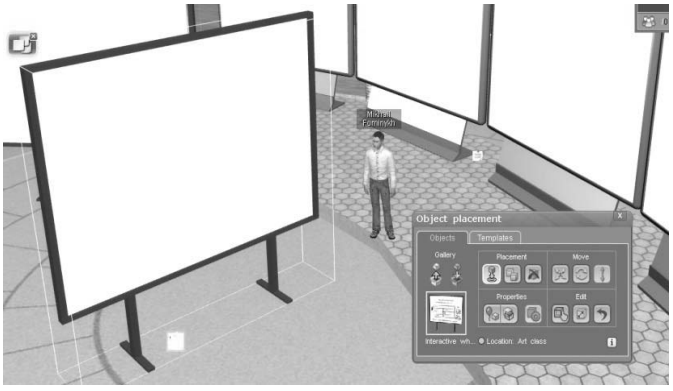

Figure 3 Placing a VWB in a virtual classroom

VWB can be used for displaying and annotating slides, sharing the desktop, sharing the application that runs on the teacher's or student's computer, sharing the web camera, drawing figures, typing text, and inserting text or graphics from the clipboard. The VWB's tools can be accessed from the context menu (Fig 3). The image shows the full menu (Fig. 4), but it can be extended or reduced if a specific tool is used or if the VWB is occupied by another user. The first row of the menu contains only one button – a laser pointer that can be pointed to any place on any VWB or other objects. The second and third rows contain all the VWB's tools. The forth row contains buttons that focus the user's view on a specific VWB, on all VWBs in the virtual location, or sets the view into free camera mode. The fifth row has a button to search through the previous contents on the specific VWB (Fig. 4).

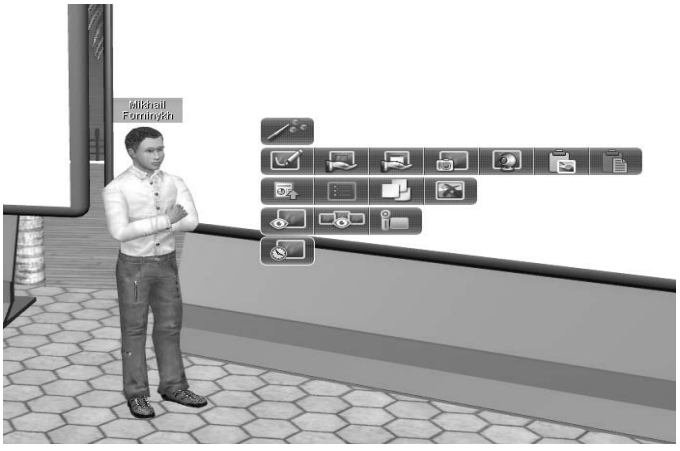

Figure 4 Accesssing VWB's tools

DOI: 10.7256/2305-6061.2014.3.13384

In addition to the context menu, the content on a VWB can be placed by drag-and-drop. Images and slide presentations can be uploaded to a Resource Collection in advance and dragged to a VWB during a live class (Fig. 5). Test and quiz results can be displayed on a VWB using drag-and-prop as well [12].

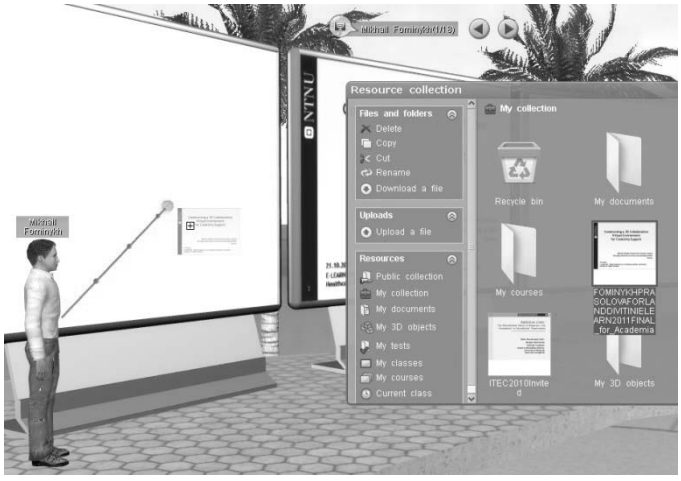

Figure 5 Placing content on a VWB

4. Performance evaluation results

In this section, we present the results of testing the tools for collaborative work on 2D graphical content and the underlying mechanisms. First, we compare the performance of the algorithms using SPs and CPU. Second, we explore the general efficiency of the system, i.e. performance degradation when using many tools simultaneously, measuring the average and peak values. In both cases, we present the average results acquired by running the system on 20 different hardware configurations with Intel CPU and NVidia / ATI graphics adapters from the same price range. On each hardware configuration 10 runs were conducted for each image size. Third, we conducted a user evaluation among a group of students.

4.1. Performance of attributed vector figures methods

The mathematical model presented in Section 2 suggests operating vector figures by their triangulation. We evaluated this method by conducting an experiment, theorizing that it is superior to the others. Three approaches were compared. The first approach suggests rasterizing vector figures directly by their analytical descriptions using shaders which reduces the task to filtration and is implemented using Filter (See the programming model in Section 2). The second approach considers that the actual area of the rasterized figure can be much smaller than the area of the rasterized triangulated rectangle. Therefore, a bounding box is used as such rectangle. The third approach is an original triangulation method we suggest.

DOI: 10.7256/2305-6061.2014.3.13384

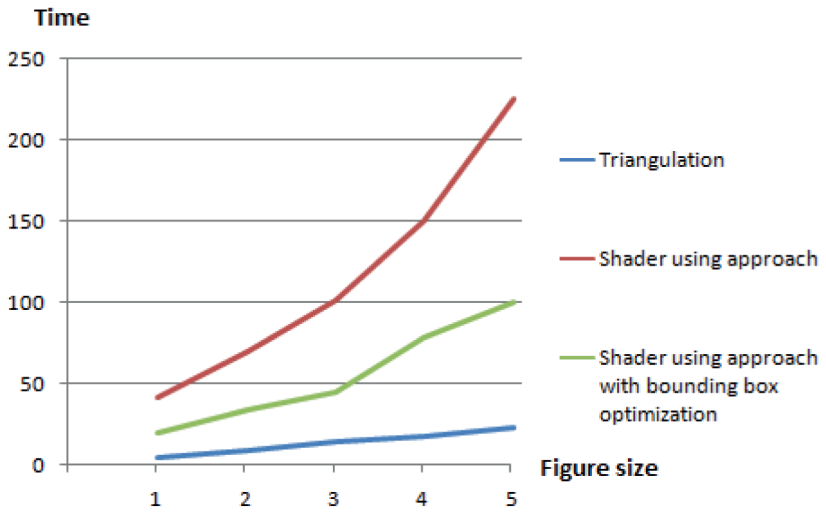

Figure 6 Performance of attributed vector figures methods

The results acquired in the experiment demonstrate that our original method provides the best performance (Fig. 6). This is caused by the fact that basic vector figures can be triangulated easily without using complex algorithms. At the same time, this approach significantly reduces the number of rasterized pixels.

4.2. Performance of the Algorithms on SPs and CPU

We compared the performance of the algorithms by SPs and CPU to confirm the rationale behind using SPs (instead of CPU) for image processing in vAcademia. Although such comparison data do not provide insight into the overall improvement in the software performance, they can be used for evaluating the power of SPs. The relation between execution time of the forward and inverse DWT on CPU and on SPs for different image sizes is shown below (Fig. 7).

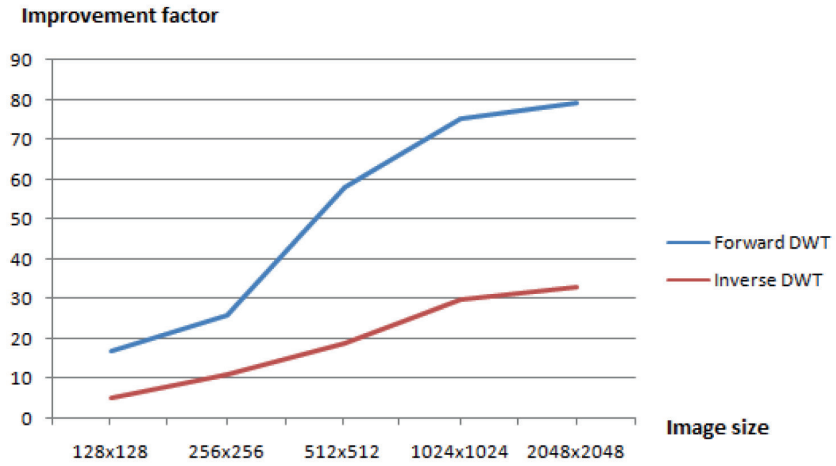

Figure 7 Discrete wavelet transformations on CPU and on SPs

DOI: 10.7256/2305-6061.2014.3.13384

A similar relation for rasterization of attributed vector figures is presented below (Fig. 8).

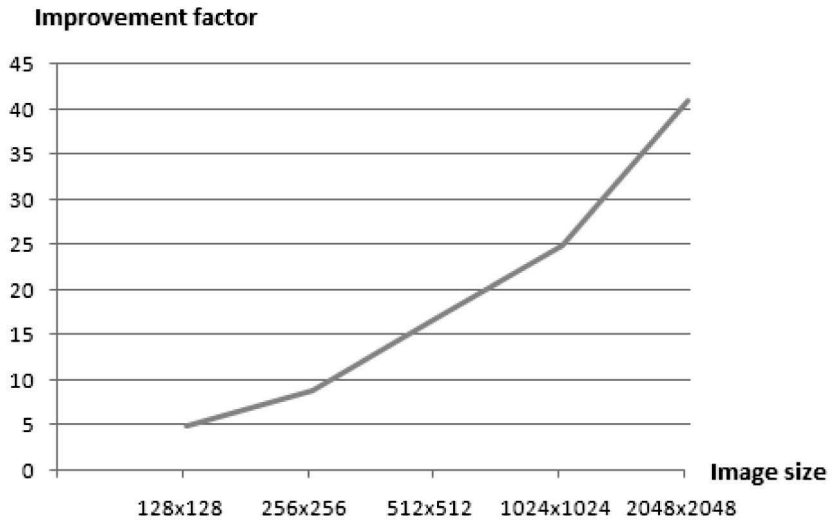

Figure 8 Rasterization of attributed vector figures on CPU and on SPs

The results presented above demonstrate that the advantage of SPs over CPUs is about 70 times for the forward DWT, about 28 times for the inverse, and 42 times for rasterization of attributed vector primitives. The maximum benefit is achieved for large images, as in this case the time of image processing is more significant in comparison with the length of time T_{TR} (formula 6). The rate of the inverse DWT is lower as the large number of passes is required and correspondingly the greater total value of T_{TR}.

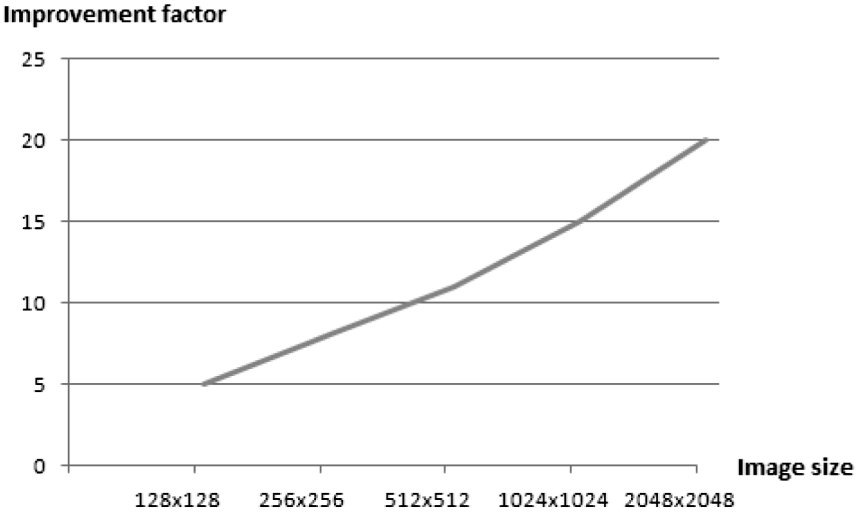

Figure 9 Performance of the Backchannel tool

DOI: 10.7256/2305-6061.2014.3.13384

In addition, we compared the performance of texture generation for a VWB with the Backchannel tool on using CPU and SPs (Fig. 9). The results demonstrate that the advantage of SP is growing with the increase of the image size. The advantage is growing slower than in the case of DWT (Fig. 7), as displaying the Backchannel tool is a less complex computation task, and therefore, T_{TR} contributes more to the overall image processing time.

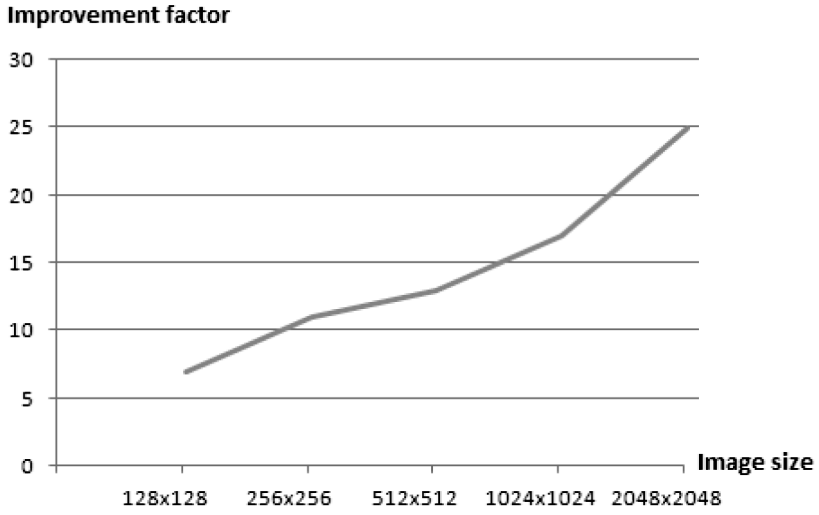

Figure 10 Performance of the Sticky Notes tool

Similar results were acquired when comparing the performance of texture generation for a VWB with the Backchannel tool on using CPU and SPs (Fig. 10). The advantage of SP over CPU is growing at the rate slightly faster than for the Backchannel tool, but still slower than for the DWT algorithm. This result confirms the relation between the computational complexity of the texture generation task, time T_{TR}, and the advantage of SPs over CPU.

Overall, the improvement acquired by using SPs differs from the ratio of the peaking performance of SPs to the peaking performance of CPU not more than twofold, which can be considered satisfactory.

4.2. General Efficiency of the System

The data acquired from comparing the performance the algorithms on SPs and CPU do not fully demonstrate how the performance of the whole system changes. Therefore, we tested the general efficiency of the system when the suggested approaches were implemented and applied.

We collected the data for the practical evaluation of the system by selective replaying 3D recordings. The number of simultaneously working tools was regulated by excluding the required number of VWBs from a re-played 3D recording. Both layers of the VWB were utilized (Fig. 11).

DOI: 10.7256/2305-6061.2014.3.13384

Figure 11 The process of testing performance degradation as a function of the number of VWBs

We present the results by demonstrating the ratio of the average and peaking performance degradation to the number of simultaneously working VWBs (Fig. 12). The analysis of data reveals that 50 simultaneously working VWBs reduce the performance of the client software not more than by 7%, which is a satisfactory result (Fig. 12).

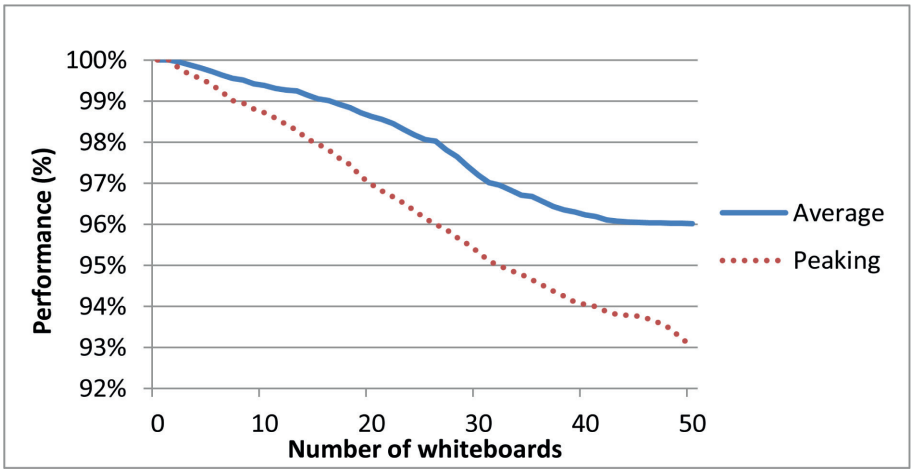

Figure 12 Average and peaking performance degradation as a function of the number of VWBs

In the next case, we used the same settings, but up to 25 VWBs that were actively used – constantly changing images on the upper layer. We present the average and peaking performance degradation as a function of the number of actively used VWBs (Fig. 13). The analysis of data reveals that the peaking performance degradation riches 22% when the number of actively used VWBs is 25. The average performance degradation riches 13% (Fig. 13).

DOI: 10.7256/2305-6061.2014.3.13384

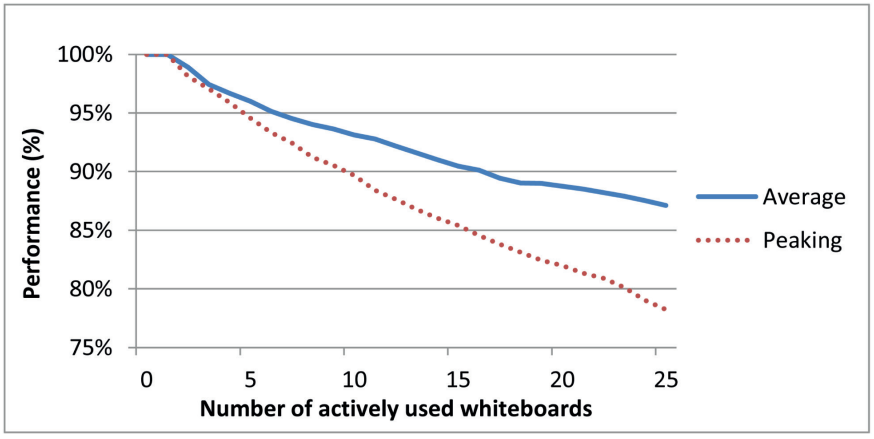

Figure 13 Average and peaking performance degradation as a function of the number of actively used VWBs

The results acquired during testing indicate that the implementation of image processing algorithms on SPs excels the performance using only CPU computational power tenfold and more.

5. Conclusion

In this paper, we presented a SPs-based texture generation model that allows designing convenient and sophisticated tools for collaborative work with graphics inside a 3D VW. The developed system allows solving the tasks of raster image processing, such as the DWT, texture generation, image filtering on SPs for further use as 2D images in 3D space visualization. The suggested model can benefit any 3D space where processing large numbers of 2D graphics is required. Using programmable and fixed pipeline abilities of stream processors give us ability to provide effective implementation of wide range of educational tools in vAcademia virtual world.

Библиография :

1. C. Dede, "Immersive Interfaces for Engagement and Learning," Science, vol. 323(5910), 2009, pp. 66–69, doi:10.1126/science.1167311.
2. R. Mckerlich, M. Riis, T. Anderson, and B. Eastman, "Student Perceptions of Teaching Presence, Social Presence, and Cognitive Presence in a Virtual World," Journal of Online Learning and Teaching, vol. 7(3), 2011, pp. 324–336.
3. R. Marroquim and A. Maximo, "Introduction to GPU Programming with GLSL," in Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing, 2009, pp. 3-16, doi:10.1109/SIBGRAPI-Tutorials.2009.9.
4. K. Fatahalian and M. Houston, "A closer look at GPUs," Communications of the ACM, vol. 51(10), October 2008 2008, pp. 50–57, doi:10.1145/1400181.1400197.

5. D. B. Kirk and W.-m. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach. New York, USA: Morgan Kaufmann, 2012.
6. K. Fatahalian, "From Shader Code to a Teraflop: How a Shader Core Works," in Beyond Programmable Shading Course New York, NY, USA: ACM SIGGRAPH, 2010.
7. OpenGL machine, http://www.opengl.org/documentation/specs/version1.1/state.pdf.
8. O. Harrison and J. Waldron, "Optimising data movement rates for parallel processing applications on graphics processors," in Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks, Innsbruck, Austria, 2007, pp. 251–256.
9. A. Pooley, A. S. Christensen, B. Merry, D. Garcia, E. Werness, G. Kolling, G. Roth, J. Green, J. Bolz, J. Sandmel, J. Blankenship, J. Leech, M. Callow, P. Brown, R. Simpson, and T. Olson, OpenGL extension ARB_get_program_binary specification, 2010, http://www.opengl.org/registry/specs/ARB/get_program_binary.txt.
10. A. Smorkalov, M. Fominykh, and M. Morozov, "Stream Processors Texture Generation Model for 3D Virtual Worlds: Learning Tools in vAcademia," in 9th International Symposium on Multimedia (ISM), Anaheim, CA, USA, 2013, pp. 17–24, doi:17 10.1109/ISM.2013.13.
11. A. Smorkalov, M. Fominykh, and M. Morozov, "Collaborative Work and Learning with Large Amount of Graphical Content in a 3D Virtual World Using Texture Generation Model Built on Stream Processors," International Journal of Multimedia Data Engineering and Management (IJMDEM), vol. 5(2), 2014, pp. 18–40, doi:10.4018/ijmdem.2014040102.
12. M. Morozov, A. Gerasimov, M. Fominykh, and A. Smorkalov, "Asynchronous Immersive Classes in a 3D Virtual World: Extended Description of vAcademia," in Transactions on Computational Science XVIII. vol. 7848, M. Gavrilova, C. J. K. Tan, and A. Kuijper, Eds.: Springer Berlin Heidelberg, 2013, pp. 81–100, doi:10.1007/978-3-642-38803-3_5.

References:

1. C. Dede, "Immersive Interfaces for Engagement and Learning," Science, vol. 323(5910), 2009, pp. 66–69, doi:10.1126/science.1167311.
2. R. Mckerlich, M. Riis, T. Anderson, and B. Eastman, "Student Perceptions of Teaching Presence, Social Presence, and Cognitive Presence in a Virtual World," Journal of Online Learning and Teaching, vol. 7(3), 2011, pp. 324–336.
3. R. Marroquim and A. Maximo, "Introduction to GPU Programming with GLSL," in Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing, 2009, pp. 3-16, doi:10.1109/SIBGRAPI-Tutorials.2009.9.
4. K. Fatahalian and M. Houston, "A closer look at GPUs," Communications of the ACM, vol. 51(10), October 2008 2008, pp. 50–57, doi:10.1145/1400181.1400197.
5. D. B. Kirk and W.-m. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach. New York, USA: Morgan Kaufmann, 2012.
6. K. Fatahalian, "From Shader Code to a Teraflop: How a Shader Core Works," in Beyond Programmable Shading Course New York, NY, USA: ACM SIGGRAPH, 2010.
7. OpenGL machine, http://www.opengl.org/documentation/specs/version1.1/state.pdf.

DOI: 10.7256/2305-6061.2014.3.13384

8. O. Harrison and J. Waldron, "Optimising data movement rates for parallel processing applications on graphics processors," in Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks, Innsbruck, Austria, 2007, pp. 251–256.
9. A. Pooley, A. S. Christensen, B. Merry, D. Garcia, E. Werness, G. Kolling, G. Roth, J. Green, J. Bolz, J. Sandmel, J. Blankenship, J. Leech, M. Callow, P. Brown, R. Simpson, and T. Olson, OpenGL extension ARB_get_program_binary specification, 2010, http://www.opengl.org/registry/specs/ARB/get_program_binary.txt.
10. A. Smorkalov, M. Fominykh, and M. Morozov, "Stream Processors Texture Generation Model for 3D Virtual Worlds: Learning Tools in vAcademia," in 9th International Symposium on Multimedia (ISM), Anaheim, CA, USA, 2013, pp. 17–24, doi:17 10.1109/ISM.2013.13.
11. A. Smorkalov, M. Fominykh, and M. Morozov, "Collaborative Work and Learning with Large Amount of Graphical Content in a 3D Virtual World Using Texture Generation Model Built on Stream Processors," International Journal of Multimedia Data Engineering and Management (IJMDEM), vol. 5(2), 2014, pp. 18–40, doi:10.4018/ijmdem.2014040102.
12. M. Morozov, A. Gerasimov, M. Fominykh, and A. Smorkalov, "Asynchronous Immersive Classes in a 3D Virtual World: Extended Description of vAcademia," in Transactions on Computational Science XVIII. vol. 7848, M. Gavrilova, C. J. K. Tan, and A. Kuijper, Eds.: Springer Berlin Heidelberg, 2013, pp. 81–100, doi:10.1007/978-3-642-38803-3_5.