Stream Processors Texture Generation Model for 3D Virtual Worlds

Learning Tools in vAcademia

Andrey Smorkalov and Mikhail Morozov
Multimedia Systems Laboratory
Volga State University of Technology
Yoshkar-Ola, Russia
{smorkalovay, morozovmn}@volgatech.net

Mikhail Fominykh
Program for Learning with ICT
Norwegian University of Science and Technology
Trondheim, Norway
mikhail.fominykh@ntnu.no

Abstract—**In this paper, we address the challenges of applying three-dimensional virtual worlds for learning. Despite the numerous positive conclusions, this technology is far from becoming mainstream in education. The most common problems with applying it in everyday teaching and learning are steep learning curve and demand for computational and network resources. In order to address these problems, we developed a 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. It allows designing convenient and sophisticated tools for collaborative work with graphics inside a 3D environment. Such tools simplify the use of a 3D virtual environment, and therefore, improve the negative learning curve effect. We present the methods of generating images based on the suggested model, the design and implementation of a set of tools for collaborative work with 2D graphical content in vAcademia virtual world. In addition, we provide the evaluation of the suggested model based on a series of tests which we applied to the whole system and specific algorithms. We also present the initial result of user evaluation.**

Keywords-3D virtual worlds; image processing; stream processors; educational content; vAcademia.

I. Introduction

Three-dimensional Virtual Environments and Social Virtual Worlds (3D VWs) provide both opportunities and challenges for education, and many topics in this area need further research [1, 2]. Despite the repeated positive conclusions, 3D VWs have not become widely used, and researchers often report that their studies have experimental nature. The most common problems with applying 3D VWs in the everyday teaching and learning are steep learning curve and demand for computational and network resources [3, 4]. And while the computers and networks are constantly improving, the 3D VWs also require significant improvement to make them more convenient for educators and to deal with the steep learning curve.

The work presented in this paper is devoted to designing and implementing a method for texture generation of educational content using stream processors. Although image processing and texture generation on stream processors 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 propose a generalized mathematical model and a programming model for the method. In terms of scientific contribution, this work can be interpreted as solving a new problem using a well-known method.

We address one of the most serious challenges for applying 3D VWs for learning – enabling collaborative learning scenarios which require large amounts of 2D graphical content displayed. We present the design of a collaborative graphical workspace of vAcademia 3D VW, which is implemented as a set of tools for collaborative work on 2D graphical content. We argue that such a workspace should be specially designed for learning and integrated into the 3D environment. We propose that the integration of such a workspace is of high value, as they ease collaborative process and reduce the necessity for using additional software. Implementing tools that are familiar for educators can facilitate the process of adapting the system into practice.

The main challenge for the implementation of tools for collaborative work on graphical content in 3D VWs is the fact that 3D VWs are resource demanding. The CPU is constantly loaded with tasks supporting the virtual space. Therefore, sustaining additional resource-demanding processes with CPU, such as processing large amounts of images, often results in unsatisfactory performance.

We suggest a stream processors texture generation model for 3D VWs. The model allows passing the calculations of collaborative work on graphical content on to the stream processors to reduce the load on CPU. Stream processors (SPs) are highly-parallel calculation units that are mostly used for 3D graphics rendering [5]. However, they can be adapted for image processing within a 3D VW.

The model was used for implementing a set of tools for collaborative work on media content within vAcademia, a new 3D VW. These tools enable collaborative and convenient work with various types of dynamic and static graphical content inside the 3D virtual environment. The tools and the underlying mechanism were tested, first, comparing the performance when using CPU and SPs, and second, exploring general performance degradation of the system when increasing the load of SPs by image processing tasks. The results obtained allow to confirm that the suggested model can be applied in any 3D virtual environment for similar tasks.

II. Background and Related Work

A. *Learning Tools in 3D Virtual Worlds*

3D VWs provide a unique set of features that can be used for learning, such as low cost and high safety, 3D representation of learners and objects, interaction in simulated contexts with high immersion [6, 7].

One of the most serious challenges in adapting 3D VWs for learning is the lack of features that educators use in everyday teaching: "Most virtual worlds were not created for educational purposes. Second Life, nonetheless, is being adapted by educators […]. Many of the features educators take for granted in Learning Management Systems do not exist in Second Life" [2]. There is a belief among educators that the 3D VWs should be better used for simulating situations that are difficult or impossible to implement in reality, and not replicating the real-world educational structures [8]. However, the absence (or inaccessibility) of familiar and convenient learning tools in the 3D VWs is also contributing to the general attitude towards the technology.

Processing large amounts of images in 3D VW is mostly required when working on serious tasks, such as collaborative work and learning. In other tasks, displaying images, video or flash is also often 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.

B. *Stream Processors*

Stream Processors (SPs) are specialized processors characterized by a very high data parallelism [9]. 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 [10, 11] and their cores have relatively simple architecture [12]. Therefore, most of the classical algorithms that can be executed on CPU cannot be executed on SPs without modification.

III. Stream Processors Texture Generation Model

A. *Motivation*

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 many 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 main and device memories is done through the data bus, which has a limited capacity. The data-communication rate limits the performance of the approach significantly.

The described types of tasks 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.

B. *Mathematical Model*

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.

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

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):

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

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.

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:

$$T = T_C + T_{TR} + T_1 * W * H \quad (8), \text{ where}$$

T – 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),

W and H – width and height of the processed image.

C. Programming Model

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 [13]) 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, 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.

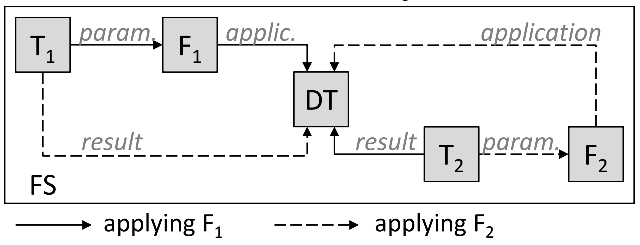

Figure 1. The relation of the objects of the model, Filter Sequence example of ping-pong technique

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 (8) needs an additional component – translation time T_T:

$$T = T_T + T_C + T_{TR} + T_1 * W * H \quad (9)$$

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 [14] 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 (9) is replaced by the time of loading a pre-compiled shader program T_{CL}:

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

where $T_{CL} << T_T + T_C$.

D. *Modification of the DWT Algorithm for SPs*

In this section, we describe an original modification of the Discrete Wavelet Transformation (DWT) algorithm to run on SPs based on the models described above. We apply it for Sharing changing blocks method (see section IV.A.1). 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 (11).

$$C_Q / D = 0 \qquad (11),\text{ where}$$

C_Q – value of chrominance component,

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 I is satisfying (12) or (13). The choice between (12) and (13) is made based on minimizing I. If I has the same value, (12) 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 images are different.

$$\begin{cases} \left| C(x,y) - (C(x-2^{I+1},y) + C(x+2^{I+1},y)) \right| < E \\ \text{x mod } 2^{I+1} = 2^I \end{cases} \qquad (12),$$

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

where C(x,y) – value of the processed color component at the point (x, y),

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.

$$C_Q = C_Q / D \qquad (14)$$

For each pixel, the algorithm is finding the minimal cascade step I that satisfies (12) or (13) 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 T_{TR} 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

$$\text{x mod } 2I + 1 = 2I \qquad (15),$$

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

$$C(x,y) = \begin{cases} \text{C(x, y), if C(x, y)} > 0 \\ (C(x-2^{I+1},y) + C(x+2^{I+1},y))/2, \mathit{if}\ C(x,y) = 0 \end{cases} \qquad (16)$$

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

$$T = T_{TR} * (K + 1) + T_1 * W * H \qquad (17)$$

and for the filter cascade scheme:

$$T = T_{TR} + (T_1 + T_2) * W * H \qquad (18),\text{ where}$$

T_2 – average wall time of minimal I that satisfies (12) or (13) for a given pixel. As the resource intensiveness of T_2 is small, T_2 * W * H is significantly smaller than T_{TR} * K.

E. *Rasterising Attributed Vector Primitives on SPs*

In this section, we describe an original method for rasterizing attributed vector primitives on SPs based on figure triangulation. We apply it for Sharing attributed vector figures method (see section VI.A.2).

Sharing attributed vector figures method is used for rasterizing complex vector drawings. 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.

Based on this rational, geometrical figures must be triangulated that is present them in the form of formula (5). We use a specific optimized method for triangulating each figure, as triangulation in general is over-complicated.

Vector primitives are both geometrical figures and text. Each letter of any font is a complex figure. Therefore, a high quality triangulation of it would require representing it with hundreds or thousands of triangles. In this case, triangulation does not provide any increase in performance. Therefore, we form font textures that store pre-rasterized letters. A texture is generated at the first time a new font of a specific size is used. As soon as the font is not used, the texture is deleted. The rasterization is implemented using the FreeType library. We rasterize only those symbols that are necessary for rendering a specific text. Each letter is rasterized in grayscale and stored in a single-channel texture minimizing video memory used. The color is applied to the text.

IV. vAcademia Tools for Collaborative Work on Graphical Content

vAcademia is an educational 3D VW developed by Virtual Spaces LLC in cooperation with the Multimedia System Laboratory at the Volga State University of Technology, Russia. Interactive virtual whiteboard (VWB) is the main tool (or a container of tools) for collaborative work on 2D graphical content in vAcademia. Multiple VWBs of different sizes can be set up in any location of the VW (Fig. 2). In addition, every participant can set up an extra VWB during the class and use 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 switching easily between the displayed data for better overview.

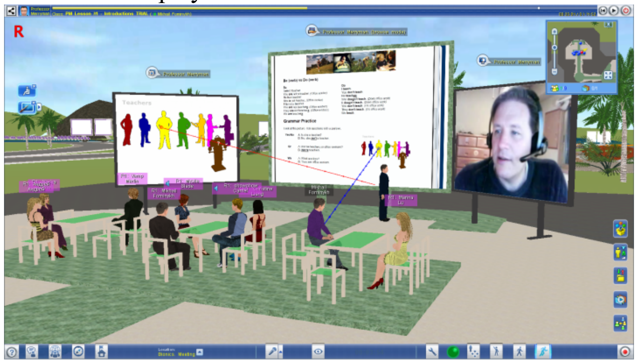

Figure 2. vAcademia environment

The VWB had the following major requirements that are based on the needs of a common teaching practice.

- The system should be able to display images from two sources simultaneously and independently.
- The system should support up to 50 VWBs within the visibility scope of the user.

The design of VWB is based on using a dynamic texture with two independent layers which are combined in one static texture when rendering. Generally, the lower level is used for permanent content, while the upper level – for temporary content with the possibility to change/erase it. This allows having a temporary dynamic figure above several different lower-lever contents (e.g., slides). In other cases, the lower lever may contain a dynamic image, while the upper lever may remain unchanged (e.g., for commenting video when the comments must be visible on all frames).

The number of 50 VWBs is based on the assumption that 2-3 classes (with 15-20 avatars) may be held in the visibility area, and each of the participants actively uses a VWB.

The tools can be classified into three groups by the method of generating the resultant image. In the following, we present the three methods and associated groups of vAcademia tools, including technical requirements, design, and implementation. The requirements for the methods are based on the results of preliminary testing.

A. *Sharing Changing Blocks*

Requirements: The system should be able to support up to three simultaneously working image-sharing processes in any location, with up to 1024 to 1024 pixels resolution each, and with a frame rate at least 5 FPS. The frame rate is considered as a requirement for the performance of the system only. However, it also depends on the Internet connection speed, and therefore meeting the requirements does not guarantee the desired frame rate (in case of low connection speed).

Design: The Sharing changing blocks method of generating the resultant image is based on an algorithm for DWT for image compression with quality reduction, which is adapted for SPs. It identifies the changing rectangular parts of the image by using occlusion query [15]. The algorithm is based on the filter cascade scheme that allows implementing the forward DWT in one pass. The method uses the lower layer of the VWB and processes the dynamic image. Quality may be adjusted by DWT coefficients.

Implementation 1: Sharing an application window allows to share the content of any window. The window is translated even if minimized or overlaid. The control over the window can be given to any user in the location.

Implementation 2: Sharing screen area allows sharing any square area on the desktop, which may contain any working applications.

Implementation 3: Sharing web-camera image allows sharing the web-camera image and adjusting its size.

B. *Sharing Attributed Vector Figures*

Requirements: The system should support simultaneous drawing or typing on up to 25 VWBs with the average performance degradation less than 15% and peaking performance degradation less than 25%.

Design: The Sharing attributed vector figures method of generating the resultant image is based on triangulation of vector primitives with attributes for one- or two-way rasterization. Displaying (both typed and inserted) text is implemented by using font textures, rasterizing each symbol on demand. The method uses the upper layer of the VWB and processes the dynamic image.

Implementation 1: Drawing figures and typing text allows to draw, erase, copy, and paste several types of geometric figures and text on the VWB. Drawing and typing actions from the time of cleaning the VWB can be undone. The undo depth is unlimited, which would be a performance prohibitive operation if using CPU only.

Implementation 2: Inserting text allows inserting text from the clipboard to the VWB as long as it fits its size.

C. *Processing Static Images*

Requirements: The system should support simultaneously changing static images on five VWBs within the visibility area. The average performance degradation should be less than 10% and peaking performance degradation less than 15%.

Design 1: The Processing static images method of generating the resultant image is based on resizing the source image, applying Filters, asynchronous unpacking, and uploading it. The method uses the lower layer of the VWB and processes the static image.

Implementation 1: Slideshow allows opening a PDF or PPT presentation on a VWB and navigating the slides.

Implementation 2: Area print screen allows displaying any square area on the desktop and adjust the image quality.

Implementation 3: Image insert allows inserting from the clipboard, from Resource Collection, and from applications (drag-and-drop). Image quality can be adjusted.

Design 2: Processing static images from 2D scene is based on a FS of a high number of alpha-blending Filters with different source images, blending settings, and hardware scissors. A 2D image or a rasterized image of a letter is taken as an input parameter.

Implementation 4: Backchannel allows displaying text-chat messages on a VWB.

V. Results and Discussion

In this section, we present the results of testing the tools for collaborative work on 2D graphical content and the underlying mechanisms. First, we compared the performance of the algorithms using SPs and CPU. Second, we explored 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.

A. *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 (Sharing changing blocks method) on CPU and on SPs for different image sizes is shown below (Fig. 3).

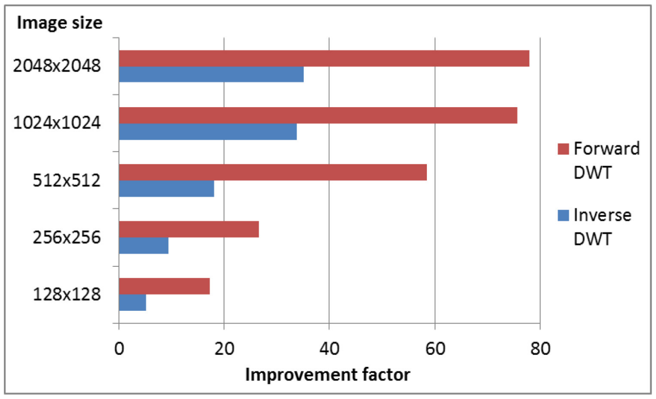

Figure 3. Discrete wavelet transformations on CPU and on SPs

A similar relation for rasterisation of attributed vector figures (Sharing attributed vector figures method) is presented below (Fig. 4).

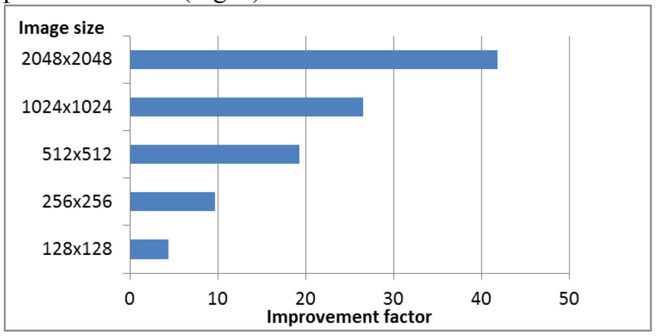

Figure 4. Rasterization of attributed vector figures on CPU and on SPs

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}. 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}.

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.

B. *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, including their satisfying of the requirements (section IV.A).

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. 5).

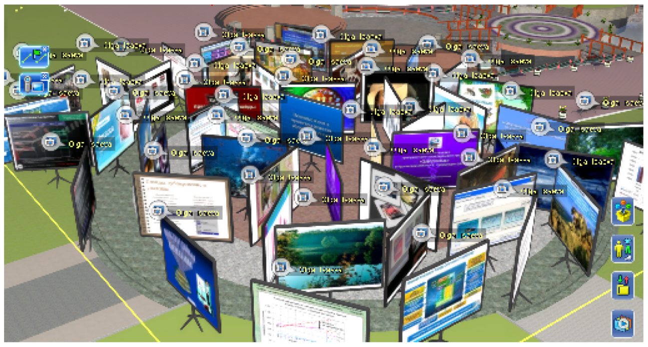

Figure 5. The process of testing performance degradation as a function of the number of VWBs

We present the results by demonstrating the ratio of the average and peaking performance degradation to the number of simultaneously working VWBs (Fig. 6). 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. 6).

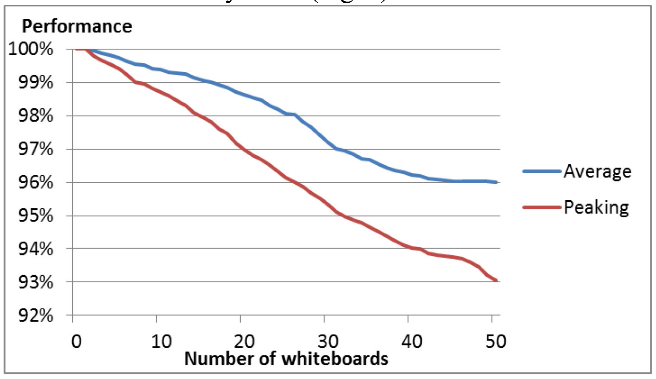

Figure 6. Average and peaking performance degradation as a function of the number of VWBs

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. 7). 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. 7).

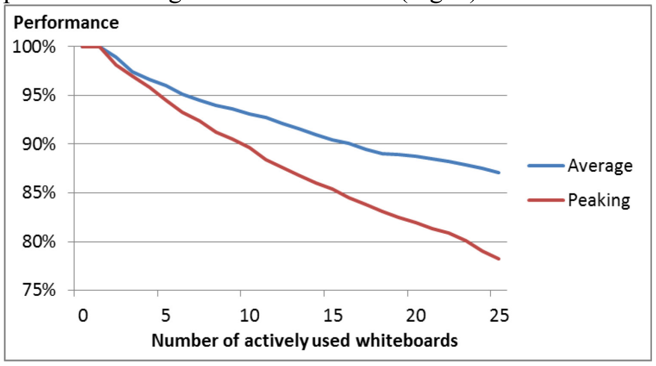

Figure 7. Average and peaking performance degradation as a function of the number of actively used VWBs

In the following case, we tested how simultaneously changing images (processing static images method) reduces the system's performance. This task is chosen as one of the most intensive, and it was essential to provide a smooth functioning of this method. We present the average and peaking performance degradation as a function of the number of simultaneous changes of images on VWBs (Fig. 8). The analysis of the data demonstrates that five simultaneous changes reduce the performance of the system by 14% in the worst case and by 8% on average.

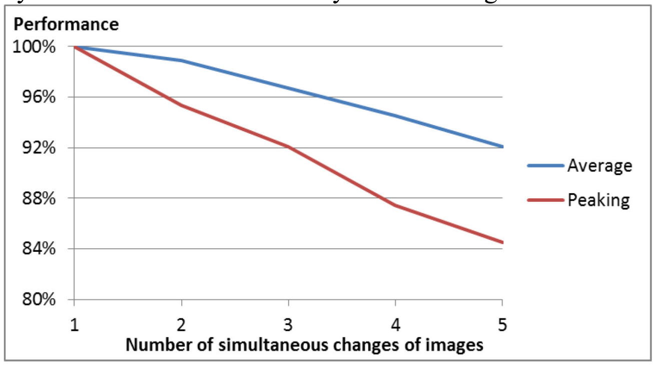

Figure 8. Performance degradation as a function of the number of simultaneous changes of images

The results acquired during testing indicate that all the technical requirements for the tools for collaborative work on 2D graphical content were satisfied. The implementation of image processing algorithms on SPs excels the performance using only CPU computational power tenfold and more.

C. *User Evaluation*

We present a small-scale user evaluation of the tools for collaborative work on 2D graphical content in vAcademia and their pedagogical value. We conducted a collaborative working session with students from Computer and Information Science department, giving them a task in which would require the system to process large amounts of graphical content. The data were collected from the system logs, a questionnaire suggested to the students after the session, and an interview with the teacher.

A group of 23 second-year computer science students participated in the evaluation. All of them had experience playing computer 3D video games. In addition, few students used vAcademia within other courses previously. Therefore, no tutorials on vAcademia were given during the session, but also no questions about the platform were asked.

We conducted the evaluation within Advanced Software Development course. The students were given a task to design a class diagram based on provided templates. In particular, they had to design a tool for collecting memory statistics. Practically, the group gathered in one location in vAcademia, and the teacher provided a slide presentation with images that contained the templates for the diagram. Each student uploaded the presentation to a personal VWB, selected a template, and drew a diagram on top it (Fig. 9). Technically, the provided presentation uploaded into the lower layers of the VWBs, and the students were drawing in the upper layers using the Sharing attributed vector figures method. The session concluded with voluntary presentations of the resultant diagrams, their analysis and a discussion.

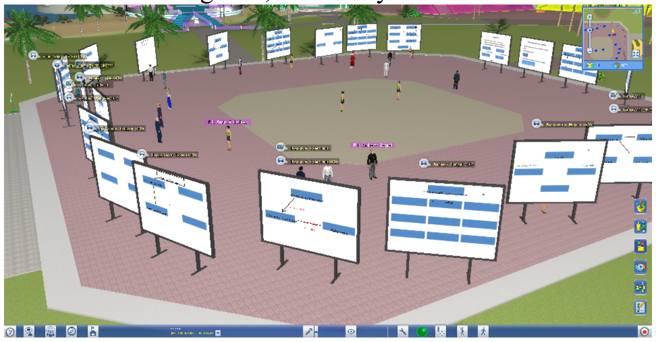

Figure 9. User evaluation session

After the working session, we offered a questionnaire with eight Likert scale questions requesting students to reflect on their experience of using the VWB. The students selected ‘strongly agree’ (SA), ‘agree’ (A), and ‘neutral’ (N) options for all questions (Table I). Options ‘disagree’ and ‘strongly disagree’ were not selected.

TABLE I. QUESTIONNAIRE RESULTS

Question	SA	A	N
It was clear what functions the VWB has and how to access them.	16	7	
It was comfortable "to look" at VWBs (to change the view angle).	15	8	
VWBs displayed the contents crispy and precisely enough to understand it.	14	9	
VWBs displayed the contents quickly enough, and delays did not influence the process of working on the task (one student did not respond).	14	8	
Increasing the # of VWBs in the virtual auditorium during the class did not lead to visible delays.	13	10	
VWB is convenient (handy) enough tool for working on similar tasks.	13	8	2
Working with vAcademia tools is more comfortable than with traditional tools, for similar tasks.	15	8	
It was clear how to work in vAcademia.	19	4	

The results demonstrate that the VWB successfully performs its functions: the features can be easily accessed, the content is displayed clearly, and no visible delays occur when 23 VWB are actively used in one location.

It has pedagogical value, as most of the students consider that it was convenient to use it and the vAcademia’s working environment is more comfortable than traditional tools. At the same time, the user evaluation we conducted is rather simple and has several limitations. We had only one group of rather specific participants, who performed only one task. A more extended evaluation is required e.g., involving less experienced students and comparing tools in other 3D VWs.

VI. Conclusion

In this paper, we suggest a technical solution that helps to overcome the most common problems with applying 3D VWs in everyday teaching and learning – steep learning curve and demand for computational and network resources. We present a SPs-based texture generation model that allows designing convenient and sophisticated tools for collaborative work with graphics inside a 3D VW. We demonstrate our design of a set of such tools, their implementation in vAcademia, and evaluation. The developed system allows solving the tasks of raster image processing, such as the DWT, texture generation, image filtering, and attributed vector primitives rasterization 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.

References

[1] M. Burkle and Kinshuk, "Learning in Virtual Worlds: The Challenges and Opportunities," in *8th International Conference on CyberWorlds (CW)*, Bradford, UK, 2009, pp. 320–327.

[2] S. Kluge and E. Riley, "Teaching in Virtual Worlds: Opportunities and Challenges," *The Journal of Issues in Informing Science and Information Technology,* vol. 5(1), 2008, pp. 127–135.

[3] J. Helmer, "Second Life and Virtual Worlds," Learning Light Limited, Sheffield, UK, 2007.

[4] S. Kumar, J. Chhugani, C. Kim, D. Kim, A. Nguyen, P. Dubey, C. Bienia, and Y. Kim, "Second Life and the New Generation of Virtual Worlds," *Computer,* vol. 41(9), 2008, pp. 46–53.

[5] Joao Afonso, Luis Pedro, Pedro Almeida, Fernando Ramos, and A. Santos, "Exploring Second Life for online role-playing training," in *Research conference in the Second Life® world*, 2009.

[6] C. Dede, "Immersive Interfaces for Engagement and Learning," *Science,* vol. 323(5910), 2009, pp. 66–69.

[7] 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.

[8] P. Twining, "Exploring the Educational Potential of Virtual Worlds – Some Reflections from the SPP," *British Journal of Educational Technology,* vol. 40(3), 2009, pp. 496–514.

[9] R. Marroquim and A. Maximo, "Introduction to GPU Programming with GLSL," in *Proceedings of the 2009 Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing*, 2009, pp. 3-16.

[10] K. Fatahalian and M. Houston, "A closer look at GPUs," *Communications of the ACM,* vol. 51(10), October 2008 2008, pp. 50–57.

[11] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach.* New York, USA: Morgan Kaufmann, 2012.

[12] 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.

[13] 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.

[14] 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*, http://www.opengl.org/registry/specs/ARB/get_program_binary.txt.

[15] M. Wimmer and J. Bittner, "Hardware Occlusion Queries Made Useful," in *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, M. Pharr, Ed. Boston, Ma, USA: Addison-Wesley Professional, 2005, pp. 91–108.