**AUTOMATIC ERRORS CORRECTION METHOD BASED OF
THE LAYER-BY-LAYER PRODUCT REPRESENTATION
WHICH PARALLEL ALGORITHMS ARE DEVELOPED FOR
MULTIPROCESSOR COMPUTER HARDWARE**

A.V. Ripetskiy¹, S.V. Zelenov²,
Dean Vučinić³, L.N. Rabinskiy⁴, E.L. Kuznetsova⁵

^{1,2,3,4,5}Moscow Aviation Institute

National Research University

125993, Volokolamsk Highway 4, Moscow, RUSSIAN FEDERATION

Abstract: Presence of channels in the geometric model makes the new requirements to software in the frames of manufacturing technological preparation task. Hidden structure of the channels complicates their visual inspection and evaluation, and therefore the special program tools are required for their detection and analysis. Tracing mathematical tool allows effective solving the part of tasks associated with the internal volumes and channels search and analysis. Internal architecture of the considered program components construction allows effective solving given task using the parallel algorithms on the multiprocessor engineering.

AMS Subject Classification: 34C60, 37M05, 97M50

Key Words: parallel algorithms, computing engineering, automatic correction, graphics processor

**1. Urgency of the Parallel Computations Task Development Using
the GPU (Graphics Processing Unit)**

The possibility to run the computations on the graphics processor arose from

Received: October 15, 2016

Revised: November 25, 2016

Published: December 11, 2016

© 2016 Academic Publications, Ltd.

url: www.acadpubl.eu

their shaders and programmable graphics transformation pipeline. Shader is the special program to run on the graphics processor, which considerably speeds up the rendering of the final image. The shaders are distinguished by the graphics transformation to which they belong. The wide application the GPU for general computations began after the appearance of the commercial graphics cards NVidia GeForce Series 8 (models 8xxx) based on the G80 chip. The main feature of this video card is its unified shader model concept. The shaders are concerned with the different stages of the graphic data processing transformation pipeline which has the possibility to run in parallel on the unified computing units of the graphics processor. The shaders' capabilities were unified and got the common functionality, with small distinctions, which are depending on the graphics transformation phase. Approximately, at the same time, the first application programming interface (API) for parallel computations CUDA from NVidia has appeared. And some time latter on, the open cross-platform standard for parallel computations OpenCL similar to CUDA has appeared. The dedicated shader (Compute Shader) appeared in the graphics DirectX API from Microsoft, starting from the version 11, and in OpenGL, from version 4.3. By summarizing what has been mentioned above, it may be noted that today, many API for supporting parallel computations exist, and using, both the central processor(s) cores and GPU. CUDA and OpenCL are the most widely spread interfaces. The conventional API for DirectX, OpenGL and newly produced Vulkan have similar capabilities. The further description is based on the example based on the OpenGL Version 4.5, as the open cross-platform API, which is well supported by the hardware manufacturers.

The GPU development has expanded possibilities for the parallel programming solutions. The branch and cycle operations are enabling the creation of more complex shaders. The support of 32-bit computations, with floating point operations, promotes the active growth of GPGPU development direction. The OpenGL version 2.0 adds support for the high-level shader language GLSL. The GPU performance reaches the hundreds of gigaflops. In their 3rd generation the support for integer operations and operations with double accuracy is present, with dedicated functionality, which allows interacting directly with GPU without passing through the 3D-graphics programming interface level (CUDA NVIDIA, CTM ATI) [1, 2, 3, 4, 5, 6].

2. Parallel Computing Technology with Graphics Processor

The above-mentioned API assumes the use of the external libraries. It is necessary to indicate the existence of parallel computational support at the initial code level.

OpenMP (Open Multi-Processing, <http://openmp.org/>) is the open standard for parallel programming, which supports C, C++ and Fortran (<https://ru.wikipedia.org/wiki/OpenMP>). The usage of OpenMP is based on the special programming coding syntax (using the pre-processor directives `#pragma`, as example, in case of C++). In such way, the compiler can automatically generate the code, which execute the computing task in several flows. Thus, OpenMP allows to parallelise the computations, with minimal changes to the code, and the developer is relaxed from the need to create the flows control.

OpenACC (<http://www.openacc.org/>) (Open Accelerators) is the parallel programming standard developed by Cray, CAPS, Nvidia and PGI. The OpenACC technology idea is similar to OpenMP, coming with the difference that OpenACC allows paralleling the computations through GPU. In OpenMP the initial coding is based on the pre-processor directives `#pragma`, and in such way the compiler is instructed to generate the code to be executed on GPU. Today, this technology is developing, and some compilers already support it. The minimum cost for the code changes is an evident advantages for using OpenMP.

The GPU use for computation constrains the program algorithms and architecture. In order to successful implement the computing algorithm the development task is divided to equal independent sub-tasks, where each of them operates with its own data portion. The greater number of defined sub-task and the smaller dependence between them is the requirement, which, if fulfilled, gives higher chances that the algorithm implementation and its respective executions on GPU are successful, and also performing better.

If it is necessary to process a data array, and store the processed results into another array, under the condition that this data allows independent processing of every separately taken element (example: conversion of the 24-bit image to the black-and-white 8-bit image). For such image conversion, it is necessary to consider all the image pixels on the one-by-one base, in order to compute the pixel brightness and store the result into the appropriate pixel in the resulting image. The conventional approach consists to cycle over all the image pixels, and thus, execute the respective pixels operations for each pixel. In case of the

large image large the operation can take significant time. By using the multi-flow method this process is accelerated. If we divide the image in N blocks, where N is the number of processor cores, and by starting the N flows, each flow will process the cycle of its own block, which will be accelerated approximately by N times. The modern processors can parallelized the simultaneous in the 4-8 flows. These modern GPU-s support simultaneous executions of several hundred flows (shaders) that allows processing the data considerably faster. Let's consider how the large image conversion task can be executed using GPU and OpenGL 4.5.

The access to the adapter computing resources is done via the standardized graphics interface OpenGL. The use of OpenGL requires the window and OpenGL context. The creation of the window and OpenGL context is outside this paper scope, but it is important to note that the existence of several open libraries (GLEW, GLFW, etc.) enable the programmer to write only about twenty lines of code to create both, the window, and the context. We will assume hereinafter that the window and the OpenGL context have been created and the context is active.

3. Data Transfer Structure Analysis for Computation Using the Graphics Processors

The data transfer to the shader input is done through buffers, textures and images. The buffers are the most suitable for our purposes, because the shader storage buffer allows writing and reading. The result of writing is also transferred to the shader storage buffer. The OpenGL buffer represents the linear memory part located in the video driver memory or video card memory. To create the buffer, it is necessary to request from OpenGL the buffer identifier (descriptor) to address it. After receiving the handle, the buffer's size and its operations can be specify. Thus, the buffer memory location of the driver, can be specified, but not changed. Simultaneously, by specifying the buffer size, the data indicator to function is transferred, which is followed by the data copy to the buffer. In the same way, the buffer is set to receive the computational results.

After the buffer is prepared, the activation of the shader program with set connection to its prepared input buffers, enables that the computational execution command can be started. The results of this command execution may be copied from the output buffer to the client memory.

Figure 1: Computation architecture with CPU and GPU

The application code interaction with the graphics adapter occurs with the help of API (OpenGL in our case). The API implementation represents the graphics adapter driver. The driver interacts with the GPU hardware by the means of commands to which the driver converts the API calls. The driver's commands are placed to the graphics adapter commands queue. The graphics adapter takes the commands from the queue for the further execution. The figures show the adapter commands queue. The new received command is positioned on the top of its stack. The currently executed command is placed at the bottom of the stack. While the adapter takes the command from the queue, the command is displaced lower and lower in the stack till the bottom, for the execution. Such order of the graphics adapter commands execution leads to the asynchronous interaction of the user with it. Due to that, the result of some calls API execution will be available after the call completion. The API special calls are used for synchronization.

Simplified diagram of program code and graphics adapter interaction

Figure 2: Tracing the task execution in the graphics processor architecture

4. Using the Task Execution Tracing for Analyzing and Correcting the Geometrical Errors in the Model Inside Volumes Based on the Channel Example

To store and transfer of the channels models is done with the parametric presentation files and STL-files. From the most CAD systems the geometric model can be stored in the STL-format representation, which is in the form of multiple triangles.

In the CAD systems, the STL-representation is defined with the solid body geometric representation. During the triangles export, this transfer conserves the vertexes winding order, thus conserving the normal direction of the boundary from the body.

Figure 3: Normal direction in STL coincides with the normal direction to solid body

In STL the normal vector is not conserved. However, the normal direction is determined by triangle vertexes winding order.

Figure 4: Normal direction is determined by triangle vertexes winding order.

Normal to triangle is determined through the cross-product of vectors collinear to its sides.

$$n = [v_1 \times v_2] = (v_{1y}v_{2z} - v_{1z}v_{2y}, v_{1z}v_{2x} - v_{1x}v_{2z}, v_{1x}v_{2y} - v_{1y}v_{2x})$$

The correct triangle vertexes winding order is exported with the solid body geometric representation to STL allows applying the tracing algorithm for determination of the body internal volumes.

The analysis of the normal direction in every point of the intersection of the scanning ray with the triangle model allows determining the internal volumes, which are not occupied with the main model body, which includes the channels.

Figure 5: Normal direction for the tracing model with channels

The appearance of the triangle with the normal opposite to the tracing direction indicates that the scanning ray leaves the 3D-body boundaries. If 2 unidirectional normals appear as the result of tracing indicates that it is intersecting or embedding other models. And we can conclude that such model does not represent correctly the solid body, thus such case indicates the presence of an error.

Figure 6: Normal direction when tracing the model including embedded bodies

5. Tracing Method

Tracing method is the main algorithmic tool for searching the errors in the model geometry and presents the geometric model in the form of grid – as connected set of faces (Fig. 10.1.). Let's select the space point x_0 and consider the intersection points in order to obtain a straight line aligned with the model faces.

Let's consider the intersection points of the obtained straight line $L(\bar{p}, x_0)$ with the model faces. We assume that the faces $\bar{\nu}_i$, $i = [1..n]$ are intersected with the straight line $L(\bar{p}, x_0)$ at points x_i . As the face $\bar{\nu}_i$ is the ordered with the set of points, it is possible to calculate the face normal as the vectors' cross-product:

$$\bar{\nu}_i = [x_1, \dots, x_3] \quad (1)$$

$$\bar{p}_i = [\bar{p}_{i1}, \bar{p}_{i2}] = [x_2 - x_1, x_3 - x_1]$$

All faces with which the straight line $L(\bar{p}, x_0)$ intersects can be grouped as front and back faces. Face $\bar{\nu}_i$ will be the front face for the vector $\bar{p}$, if the angle between vectors $\bar{p}_i$ and $\bar{p}$ will be less than 90° . Let's determine it with the vectors' cross-product:

$$\nu_i = \begin{cases} front, & \text{if } (\bar{p}_i, \bar{p}) = |\bar{p}_i| |\bar{p}| \cos(\phi) > 0, \\ back, & \text{if } (\bar{p}_i, \bar{p}) = |\bar{p}_i| |\bar{p}| \cos(\phi) < 0. \end{cases} \quad (2)$$

Figure 7: Model representation.

Let's compose the set M from pairs $(x_i, \bar{p}_i, x_i)$ – point of the face $\bar{v}_i$ intersection with straight line L , $\bar{p}_i$ – vector of the face normal $\bar{v}_i$. Let's order the points x_i on the straight line L . The model is closed relative to straight line L if the first face is the front one and orientation of every next face differs from the previous face. Let's write it through the scalar product: set of pairs $M = (x_i, \bar{p}_i)$, $i = 1, \dots, n$ is correct if

$$(\bar{p}_0, \bar{p}) > 0, (x_0, x_i) < (x_0, x_{i+1}), (\bar{p}_i, \bar{p}) = -(\bar{p}_{i+1}, \bar{p}) \quad (3)$$

The correctness of the entire model, as the whole, can be formulated in the following way: model V is correct if there is no such vector $\bar{p}$ and point x_0 for which set of pairs $M = (x_i, \bar{p}_i)$, $i = 1, \dots, n$, is invalid. The model correctness check has the sense only from the direction from which its layer-by-layer calculation is executed, and for with such resolution is required for the manufacturing with the specific equipment.

6. Modeling the Channels in the STL-Model with the Tracing Method

For modeling the internal volumes, including the channels, the tracing method is used with the special algorithm based on the normals analysis.

Figure 8: Using the normal direction for determination of channels resulting from tracing.

The construction of the channels' geometric model requires the multiple application of the tracing method with the scanning ray different parameters.

The repeatability of the computations and the independence from the previous iteration results allows to apply effectively the parallel computations for solving the task of the channels analysis in the geometric models. By tracing the results through special processing allows unifying the results of multiple calls for constructing the internal volume geometric model and further numerical calculations and analysis.

Let's consider the tracing ray as the straight line in space:

$$r = r_0 + t \cdot a, \quad (4)$$

where r_0 is point at straight line, a is vector collinear to straight line, t is parameter.

Let $T(M, r)$ is tracing function the result of which is the ordered set of the model M faces intersection with the scanning ray r .

$$T(M, r) = [(p_1, n_1), \dots, (p_n, n_n)] \quad (5)$$

Let $V([(p_1, n_1), \dots, (p_n, n_n)])$ is function, which calculates the internal channels length based on the results of the tracing with scanning ray. Then

$l_{chan}(M, r) = V([(p_1, n_1), \dots, (p_n, n_n)]) = M(T(M, r))$ – channels length as the result of the model tracing with scanning ray.

For analyzing the channels in model with the tracing method, let's consider the set of the tracing collinear rays

$$R = r_1 + t \cdot a, \quad (6)$$

such that r_i lie in one plane, and this plane is orthogonal to tracing direction a .

If all the points on x_i, y_i, z_i represent on the plane the regular grid with pitch d then the channels volume (V_{chan}) can be calculated as the channel lengths sum for every tracing ray multiplied by d^2 :

$$V_{chan} = \sum_i V(T(M, r_i)) \cdot d^2 \quad (7)$$

7. Modeling the Channels for STL-Model with Tracing Method on Example of Geometric Model of Plate with Internal Channels

Model of plate with internal channels is considered in this example. Plate has the following geometric dimensions: $100 \times 40 \times 5$ mm.

Figure 9: Plate with internal channels

Tracing method was used to determine the channels and to construct their geometric model.

Figure 10: Plate and geometric model of internal channels

On base of the layer-by-layer plate and channels presentation there was obtained data of the volumes distribution per layers (Curve 1.) and conducted validation with the linear dimensions of initial model.

$$V_{model} = 17430.33mm^3$$

$$V_{chan} = 2569.669mm^3$$

Plate and the channels total volume obtained on base of the layer-by-layer representation equals 20 000 mm³ that corresponds to the plate volume according to its linear dimensions: $100 \times 40 \times 5 = 20000 \text{ mm}^3$.

Figure 11: Plate and channels volume distribution per layers

Conclusion

The presence of channels in the geometric model creates new requirements for the software which prepares the frames for the manufacturing of the technological preparation task. The hidden structure of the channels complicates their visual inspection and evaluation, and therefore the special program tools are required for their detection and analysis. The tracing mathematical tool allows to effectively solve the part of tasks associated with the internal volumes and channels search and analysis. The internal architecture of the considered program components construction allows an effective solving of the given task by using the parallel algorithms for the multiprocessing hardware.

The task of forming the model layer-by-layer representation by using the tracing method requires the significant computing resources. The algorithmic complexity grows linearly in respect to the geometric model data volume and quadratically in respect to the layer-by-layer representation accuracy improvement. However, the fact that the tracing function handles the unchangeable data field - model geometric representation, the tracing results do not depend on the other rays tracing the data, which allows an effective application of the method of paralleling the computing algorithms.

Acknowledgement

The work was performed by the Moscow Aviation Institute at financial support of the Federal goal-oriented program “Research and development of priority directions of Russia scientific-and-technological complex development on 2014-2020”, Agreement No. 14.586.21.0019 (unique identifier RFMEFI58615X0019).

References

- [1] J. Kessenich, D. Baldwin, R. Rost, *The OpenGL Shading Language*, The Khronos Group Inc. (2016).
- [2] D. Gu, *Laser additive manufacturing of high-performance materials*, Springer, Berlin (2015).
- [3] I. Gibson, D. Rosen, B. Stucker, *Additive manufacturing technologies*, Springer, Berlin (2010).
- [4] P.K. Vevuinod, W. Ma, *Rapid prototyping: Laser-based and other technologies*, Springer, Berlin (2004).
- [5] R. Anamova, A. Avtushenko, A. Ivanov, A. Ripetskiy, A. Osipov, *J Vestnik BGTU* **2** (2015), 8.
- [6] R.R. Anamova, S.V. Zelenov, M.U. Kuprikov, A.V. Ripetskiy, *Multiprocessing and Correction Algorithm of 3D-models for Additive Manufacturing*, IOP Conference, Series: Materials Science and Engineering **140** (2016), 012003. doi:10.1088/1757-899X/140/1/012003