29th Congress of the International Council of the Aeronautical Sciences
St. Petersburg, Russia; September 7-12, 2014

VIRTUAL PROTOTYPING MODELING IN THE CAVE 3D ENVIRONMENT

Shabrov N.N.* , **Kuzin A.K.*** , **Orlov S.G.*** , **Chetverushkin B.N.**** , **Iakobovski M.V.****
***St. Petersburg State Polytechnical University (Russia),** ****Institute for Mathematical Modeling of RAS, (Russia)**

Keywords: *virtual prototyping, isosurface, mesh simplification*

Abstract

Currently, the technologies of digital engineering are further developed in the direction of virtual engineering. Virtual environment systems like CAVE 3D play the key role in the analysis and simulations of very large and multiscale models in the framework of high performance computing (HPC). This paper outlines various aspects of the results of our investigations in the framework of virtual prototyping:

- *integration of CAD/CAE/CFD/PDM/FSI/ HPC/3DiVR technologies;*
- *development of the interactive functionality of virtual reality systems with the help of Fingertracking devices in real time operation mode;*
- *development of new approaches for analysis of the results for large models of complex technical systems in real time operation mode.*

One of the most serious problems encountered in addressing contemporary problems of modeling is the current status of specialized commercial software. This software is characterized by extremely low efficiency of use with modern HPC systems. The reason for this is the growing day by day gap between software and hardware architectures. Software architecture is more conservative with respect to the architecture of hardware. As acknowledged by the world renowned expert in the field of HPC Jack Dongarra, in light of the "revolution" in the hardware architecture, all the applied software must undergo renovation in the coming years. To change the situation, the concepts of creating software should also radically change, basing on the receipt and application of new knowledge. This is a really global problem. The creation of qualitatively new software applications in the broad sense, including advanced interactive 3D interfaces human-computer interaction, will provide a qualitative effect in virtual prototyping of large scale models. The increasing scale and complexity of simulations, and the data they produce, will be a key driver of the research agenda in the area of data analysis and visualization. Interactive data exploration will also become increasingly important as dataset scale and complexity continue to grow. These will force new approaches to multidisciplinary analysis and visualization computations to the larger datasets. In particular, there is a significant problem to visualize the results of modeling on large scale (10^9 nodes) meshes in real time. We consider the problem of isosurface visualization on such a mesh. At first, the isosurface is generated on the original tetrahedral mesh; then it is reduced to be suitable for visualization. Implementations are proposed for parallel algorithms of isosurface generation, aimed on multicore and GPU hardware architectures. 3-Sided CAVE 3D (Computer Aided Virtual Reality) system and Display Walls are used to visualize isosurface of a field on large scale mesh in real time.

1 Virtual prototyping concepts

Among the major challenges of the 21st century in the field of breakthrough computer technologies the problems of software development for high-performance computing (HPC) systems and solving of the problems of predictive modeling of technical systems with complex geometrical structure should be specified. The tasks of computational support of the 21st century's breakthrough technologies have got their deserved place in the IESP Roadmap, the document prepared by international community of software developers and devoted to fundamental revision of strategy and development of the software for HPC systems for the period 2010–2019. The amount of computational recourses, which is necessary for the technologies support, requires the creation of computational systems with peta/exascale performance level. It is assumed that the volume of data generated by the peta/exaFLOPS computing reaches the level of petabytes and exabytes of a dataset.

Analysis and visualization of massive dataset streams of peta/exabytes level generated as a result of predictive modeling in engineering studies, requires the creation of both new software technologies for analysis and visualization and new hardware rendering facilities. It means that real-time analysis of results of peta/exaFLOPS simulation requires systems of virtual environment such as CAVE 3D (Computer Aided Virtual Environment), which are the one and only effective tools for comprehension of the huge amount of data and will play the key role in the near future. These environments are most in demand in high-tech industries such as aerospace and aviation (see Fig. 1).

Virtual environment systems such as X-sided CAVE 3D and DisplayWalls complete the processing chain CAD/MBS/CAE/CFD/PDM/FSI/HPC/3DiVR of virtual prototyping or virtual engineering technologies. Virtual prototyping or virtual engineering is the new stage of development of simulation and analysis technology; it represents the extension of digital engineering technologies (see Fig. 2). Virtual engineering environment is an open functional interactive environment where a set of virtual prototyping technologies are integrated into (Fig. 2). The set of virtual prototyping technologies includes geometry modeling technologies through CAD/CADG systems, MBS/CAE/CFD/FSI/HPC simulation tools supporting work distribution between the groups of researchers (Collaborative Work), simulation results analysis through 3DiVR hardware and software tools, and decision-making technologies.

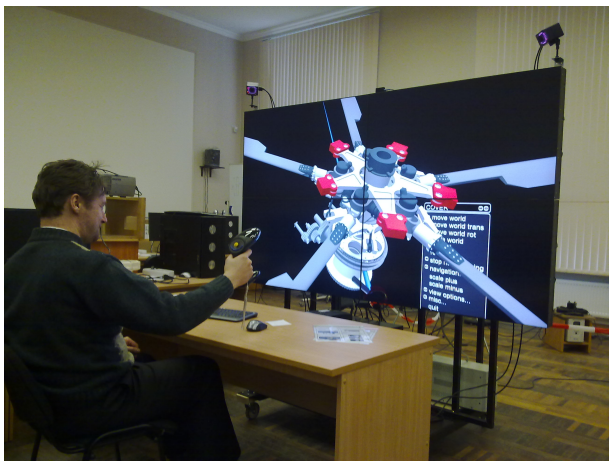

Fig. 1 Analysis of CAD model of main helicopter rotor using displaywall based on 4 LCD NEC 46" UN and optical tracking system TrackPack2

Currently there are many efforts around the world to enhance the software in order to improve the functionality of virtual environments. The global goal is to create, develop and integrate distributed co-simulation and interactive virtual reality environments that would provide real-time results analysis. The solution of this problem should allow the user to get basic functions necessary for collaborative work within the framework of granted session.

One of the main tasks related to the virtual environment functionality enhancement is the problem of visualization of simulation results on very large meshes. There are difficulties associated with rendering as well as with limited bandwidth of transferring datasets into the visualization system. Particularly, these problems arise when the results of dynamics simulations are to be visualized. Specialized computer simulation systems are currently able to process and generate enormous amounts of data; however, in practice the

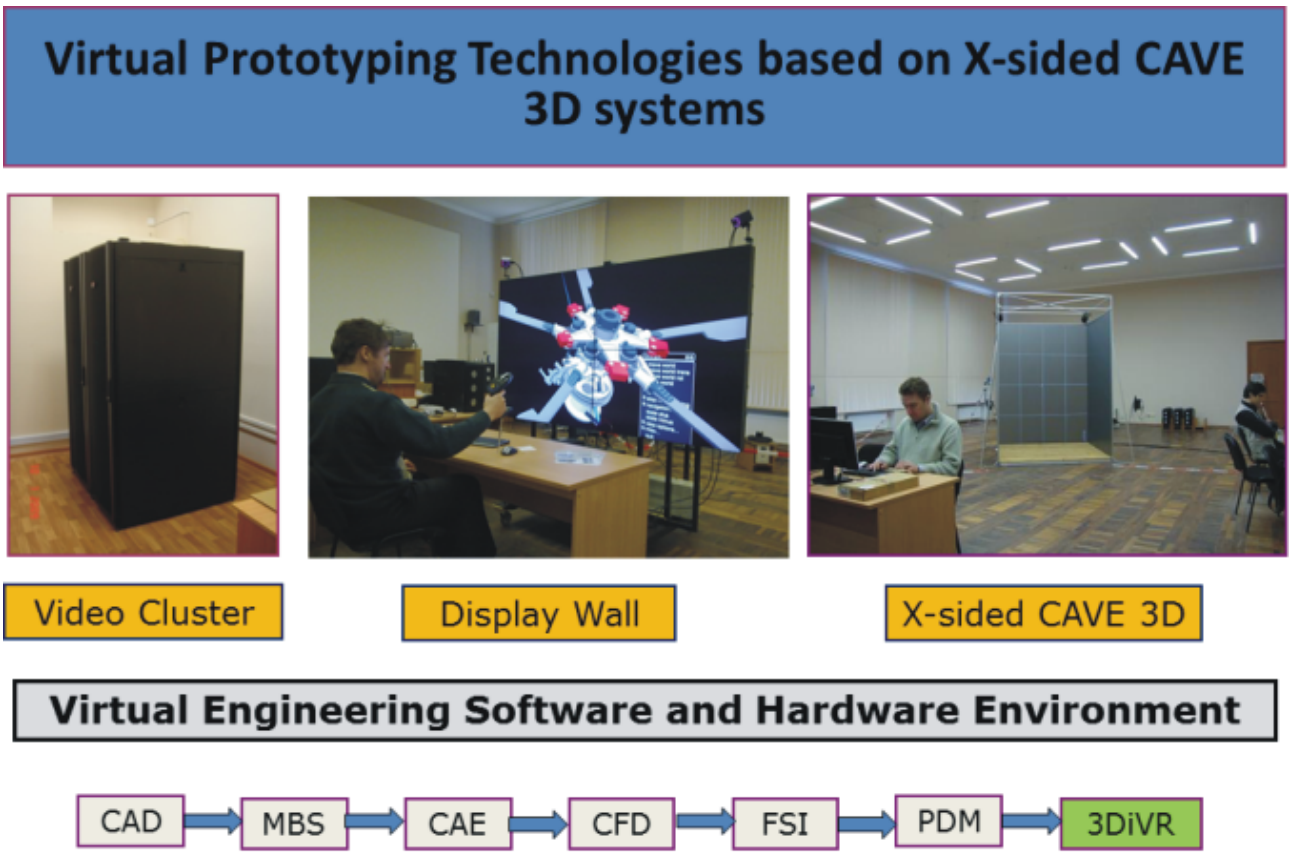

Fig. 2 Integration scheme of virtual prototyping technology of the technological systems

tasks of modeling are usually not performed in real time. At the same time visualization systems must process data with rate sufficient for the real time user work.

The most promising approaches for solving of this problem are:

1. Preprocessing of simulation results in order to reduce the amount of data. Algorithms should be developed to determine and extract a fraction of data necessary for visualization.

2. Compression of geometric data before transferring to the visualization system with fast decompression afterwards using processors installed on the graphics accelerators. This approach should solve the bandwidth bottleneck problem.

3. Execution of calculations directly on processors of visualization system (GPU). This approach is efficient when the visual representation of an object can be partially reconstructed from parameters of simulation results, the amount of which is substantially less than the amount of data to be restored. Recovery algorithm is partially being implemented on the GPU.

An important component of a hardware-software complex like CAVE virtual environment system is the optical tracking system that largely determines the quality of the analysis of simulation results. The tracking system gives user the necessary set of interactive functionality through the tools such as Flystick and Fingertracking. Fingertracking technology gives user the unique opportunity of interactive modification of the surface of the virtual object in real time and allows to make decisions directly in the virtual environment (see Fig. 3).

Fig. 3 Demonstration of interactive modification of the surface of a virtual object in the framework of 3-sided CAVE 3D in real time. Original shape (left) and the shape after modification (right)

2 Problem overview

The next two basic approaches to the problem of simplified isosurface generation may be considered. The classical approach is to apply one of the mesh simplification algorithms to the entire isosurface, which has to be extracted before. One of the main problems of such an approach is big memory consumption due to the necessity to store the whole uncompressed isosurface; therefore, so-called out-of-core mesh simplification techniques have to be used. On the other hand simplified mesh quality control looks straightforward because the whole initial mesh is known at the beginning of the simplification procedure.

The method of choice for the uncompressed isosurface extraction depends on the structure and the sizes of original 3D grid. In rather generic case of unstructured tetrahedral grid with linear interpolation the modifications of *marching tetrahedra* are widely used [1]. Also more advanced techniques, for example, *marching diamonds* method, introduced in [2], may be applied.

Finally the whole isosurface is the subject to be reduced by one of the mesh simplification techniques that are usually the combinations of vertex clustering [3, 4] and edge contraction [5] operations.

Another approach to the simplified isosurface generation is to simplify the mesh directly during the isosurface extraction. There are promising researches in this direction, such as so-called *tandem algorithm* introduced in [6]. An extended parallel version of this method can be found in [7]. It is worth mentioning that [1] also suggests a sort of simultaneous isosurface generation and simplification. In addition to the marching tetrahedra step, the vertex clustering takes place.

So let us consider original unstructured mesh, split into domains each containing about 10^6 nodes (thus, there are about 1000 domains); the correspondence between nodes at interface boundaries between domains is also provided. Each domain can be processed separately, independently from the others. The result of domain processing is a piece of reduced isosurface on it. Pieces of isosurface obtained after domain processing are seamed pairwise, and then further reduced.

A scheme of operation of the proposed software implementation of algorithms of isosurface generation and reduction is shown in Fig. 4.

In the beginning, a pool of MPI processes is created. One of them, the controlling process, executes job scheduler code, and others wait for tasks from the scheduler. The scheduler gives tasks to working processes as they accomplish previous tasks. In each task, one domain is processed. Thus, dynamic balancing of CPU load is achieved. The performance grows with the number of available processors, as soon as the number of domains exceeds the number of CPUs.

After all domains are processed, the seaming of isosurface part is performed, as well as further reduction of larger isosurface parts that result. The process continues until the only part remains. At this stage, a task for a working pro-

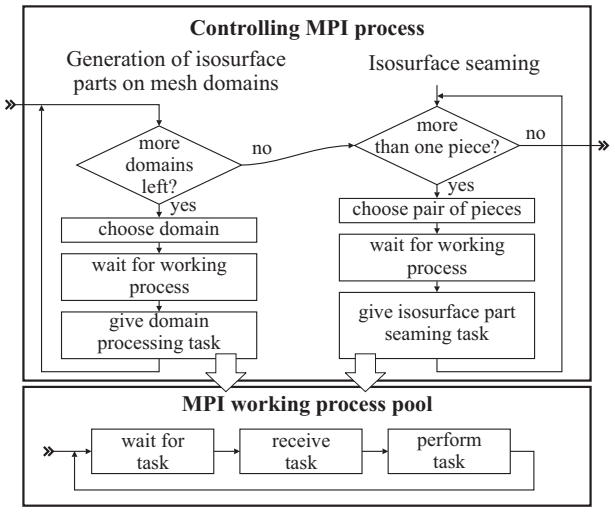

Fig. 4 Scheme of operation of software

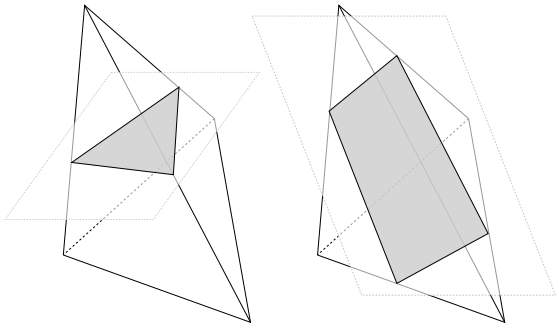

Fig. 5 Intersections of tetrahedron and plane

cesses is to seam a pair of isosurface parts and to reduce the resulting part. The choice of pairs is done by job scheduler process.

For domain sizes of about 10^6 nodes, it can be processed either on CPU or GPU. We can expect that the time of processing on GPU will be much less due to the parallelization within the domain. Specialized parallel implementations of isosurface generation and reduction algorithms have been developed in order to utilize GPU resources efficiently. Working MPI processes determine whether to use CPU or GPU, depending on whether a GPU waiting for new tasks is available. A resource manager has been developed for this, which keeps track on the utilization of computational resources.

Therefore, there are several distinct parts in the software, which are as follows: algorithm of isosurface generation at a separate domain (CPU and GPU implementations); isosurface reduction algorithm (CPU and GPU implementations); algorithm for seaming a pair of isosurface parts (CPU implementation); job scheduler; resource manager.

3 Isosurface generation algorithm

It's essential for the isosurface generation algorithm that the mesh consists of tetrahedra, and the scalar field is linear within each of them. Thus, each mesh edge intersects the isosurface at most at one point. The intersection of the isosurface and a tetrahedron is either a triangle of a rectangle, as soon as isosurface part within a tetrahedron is flat (Fig. 5).

Special attention has to be paid to the case when the value of the field at certain node is equal to field value at the isosurface. This case leads to an uncoarse situation that significantly increases the complexity of the entire algorithm. The developed algorithm versions avoid this problem by adding small terms to those nodal values of the field that are equal to the value at the isosurface.

The isosurface generation algorithm consists of the following steps.

1. Calculate the range $[f_{\min},\ f_{\max}]$ of nodal values of the field f.

2. Add small terms to nodal field values that coincide with the value of isosurface level f_0. The magnitude of the term is taken equal to εf_0, if $f_0 \neq 0$, and to $\varepsilon(f_{\max} - f_{\min})$, if $f_0 = 0$. The value of ε is assumed to be 10^{-7}, since the calculations are done in single-precision floating point numbers. The change of the field doesn't affect the visible isosurface geometry; at the same time, this makes the algorithm significantly simpler due to the absence of uncoarseness.

3. In a loop over all edges, check for the intersection of an edge with the isosurface. If the intersection is found, the correspondence between the edge and a new isosurface mesh node number is set. Besides, a

parameter in the range [0, 1] is calculated that determines the position of the node on the edge.

4. Generate isosurface triangles. At this stage, all tetrahedra intersecting the isosurface are processed. For each of them, one or two triangles are generated, that make up a part of the isosurface within the tetrahedron. The concordance of orientations of triangles can be set basing only on the knowledge of field values at each certain tetrahedron (notice, however, that the orientations of tetrahedra in the original mesh are expected to be concordant)

5. Generate isosurface nodes. At this stage, isosurface node coordinates are calculated; the nodes are in fact found at stage 3.

6. Set correspondence between the nodes at isosurface border and the edges of domain mesh at its boundary. This step is necessary for further seaming of isosurface parts belonging to neighboring domains. The mentioned correspondence makes it possible to find nodes corresponding to each other algebraically, without coordinate comparison.

Let us mention that the GPU implementation of this algorithm is nontrivial, though it can be deduced to the sequence of standard algorithms `for_each`, `transform`, `partition`, `sort`, `scan`, `gather`, `scatter`, `unique`, `remove_if`, `copy` [8]. The developed software employs the Thrust library [9] that provides the parallel GPU implementations of these algorithms.

4 Isosurface reduction and seaming

The proposed algorithm allows to significantly decrease the size of isosurface mesh, obtained at the previous stage. The basic operation performed on the isosurface mesh is *edge collapsing* (Fig. 6), such that the edge collapses, turning into a node, so this is the sort of edge contraction method in the terms of [5]. Besides,

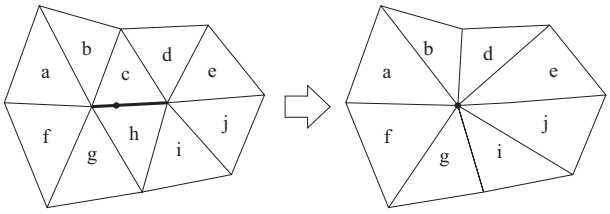

Fig. 6 Edge collapsing

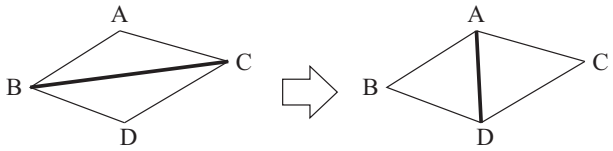

Fig. 7 Edge flipping

some pairs of neighboring triangles are replaced by other pairs, in which the common edge goes in a different way (Fig. 7); notice that each of such pairs is identified by certain edge of isosurface mesh. Let us call the second operation the *edge flipping*. The possibility of performing one of the mentioned two operations on an edge of the mesh is determined by the value of criterion Φ calculated on that edge (the criterion is discussed below). It's important that the collapse or flip operation on the edge affects the value of criterion at nearby edges. Keeping the parallel implementation (using GPU) in mind, we had to impose certain limitations on the order of operations. The algorithm performs a sequence of iterations; at each iteration, the following steps are taken.

1. Calculate the value of criterion Φ_i at i-th edge, for all edges of the isosurface mesh. This operation can be performed in parallel (in particular, GPU implementation uses one thread per edge). Edges at which Φ_i exceeds certain Φ^* (which in turn determines the quality of reduced isosurface) are potentially suitable for collapsing (or flipping, which is determined by a special per-edge flag). Denote as G^* the set of edges g_i at which $\Phi_i > \Phi^*$. Notice that in the case of edge collapsing, an additional parameter is also calculated that determines the position of node replacing the edge.

2. Choose subset G_1^* of edges that can be

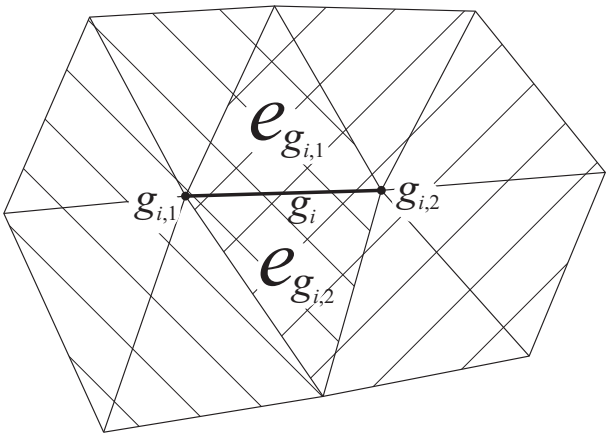

Fig. 8 Leaves centered at edge nodes; faces containing the edge

collapsed or flipped simultaneously. The possibility of simultaneous edge collapsing implies that the collapsing of each edge from G^* do not affect the value of criterion at any other edge also belonging to G_1^*. For the criterion that we have chosen this means that in the graph formed by the nodes and edges of the isosurface mesh, the length of the path between any two nodes belonging to any two different edges from G_1^* should be not less than two. Currently only sequential algorithm for choosing such a subset is implemented.

3. Collapse or flip each edge from G_1^*. This operation can be performed in parallel.

The iterations continue until G_1^* is empty, or until two consecutive iterations appear, at each of which no edges are collapsed.

To calculate the criterion Φ_i at edge g_i, the information is necessary about all isosurface faces that contain nodes of the edge. Let us call a *leaf* centered at k node, L_k, the set of all isosurface faces containing that node. The value Φ_i is determined by the union of leaves $L_{g_{i,1}} \cup L_{g_{i,2}}$, where $g_{i,1}$ and $g_{i,2}$ — are the numbers of nodes at the ends of edge g_i (Fig. 8). Let us also denote as $e_{g_{i,1}}$ and $e_{g_{i,2}}$ the numbers of faces containing the edge g_i. Notice that the isosurface generation algorithm guarantees that each edge belongs to either two faces or (for edges at isosurface mesh border) to one face.

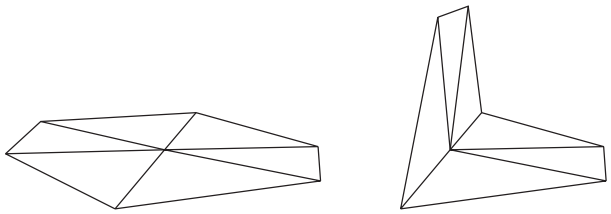

Fig. 9 Flat and "bended" leaves

The procedure of calculation of the criterion Φ_i at the edge g_i is rather branched. The most often the value is calculated basing on the estimation of local curvature of leaves $L_{g_{i,1}}$ and $L_{g_{i,2}}$. For example, consider an inner edge — it belongs to two faces. Denote unit normal vectors to these faces as $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$. Each of the leaves $L_{g_{i,j}}$ $(j = 1,2)$ is divided into two parts, $L_{g_{i,j},1}$ and $L_{g_{i,j},2}$, in the following way. The first face, $e_{g_{i,1}}$, falls into the first part. Further the neighboring faces of the leaf are traversed; the traversal direction is such that a face other than $e_{g_{i,2}}$ follows face $e_{g_{i,1}}$. All subsequent faces, whose normals are closer to $\boldsymbol{n}_1$, rather than to $\boldsymbol{n}_2$, also fall into the first part. The remaining faces fall into the second part. Denote the sets of unit normal vectors to faces from first and second parts as $\boldsymbol{n}_{g_{i,j},1}$, $\boldsymbol{n}_{g_{i,j},2}$. For each of two leaves centered at edge ends, now calculate

$$\Phi_{i,j} = \min_{s=1,2} \left\{ \min_{\boldsymbol{n} \in \boldsymbol{n}_{g_{i,j},s}} \{\boldsymbol{n}_s \cdot \boldsymbol{n}\} \right\}, \qquad j = 1,2$$

The closer the normal vectors of faces from $L_{g_{i,j},1}$ to $\boldsymbol{n}_1$, and the normal vectors from $L_{g_{i,j},2}$ — to $\boldsymbol{n}_2$, the more the value of $\Phi_{i,j}$ is. The maximum of 1 is achieved at flat and "bended" leaves (Fig. 9). Finally, the value of criterion Φ_i at the edge is calculated by the formula

$$\Phi_i = \Phi_{i,1} t_i + \Phi_{i,2}(1 - t_i), \qquad t_i = \frac{2}{\pi} \arctan \frac{\Phi_{i,2}}{\Phi_{i,1}}.$$

The parameter t_i determines the position of node that replaces the edge as it collapses.

The use of the above formulas for the calculation of Φ_i allows reducing isosurfaces with corners by collapsing the edges lying at the corners (not just at "flat" parts of the mesh).

The calculation of criterion value at edger sometimes involves other branches, allowing

to collapse very short edges; specifically consider edges with leaves consisting of three faces; specifically consider edges lying at the isosurface mesh border; flip long edges between narrow triangles. We cannot describe all these branches here, but their presence in the algorithm allows to significantly improve the quality and decrease number of faces of reduced isosurface. Besides, a number of tests are performed that cull edges whose collapse would cause topology faults or appearance of nearly degenerate faces. Due to these tests, the homeomorphism of original and reduced isosurface meshes can be guaranteed.

Let us also notice that the isosurface reduction algorithm allows do keep a subset of nodes at mesh border unchanged. This feature is necessary for providing the possibility of seaming of isosurface parts belonging to neighboring domains.

The seaming of a pair of isosurface parts consists in the generation of common numbering of nodes and faces, taking into account that nodes belonging to the interface between domains is common at both parts. Besides, the seaming is followed by another reduction pass. This is done in order to reduce the mesh hear the seam line.

5 Load balancing and job scheduling

The development of an application for a multiprocessor system unavoidably arises the problem of load balancing. In our particular case, this is done using a simple and at the same time efficient mechanism of dynamic distribution of workload between MPI processes. One of MPI processes is the controlling one. Its objective is to distribute the work among other processes. A working process informs the controlling one when it is ready to perform a new task, and the task is provided by the controlling process in the reply message. As soon as the task is complete, the working process sends the result to the controlling process and again starts waiting for either a new task or the termination command. The controlling process gives tasks from the queue. Due to specific features of our case, this approach is rather efficient and allows to reduce CPU downtime to zero.

However, the use of heterogeneous computational hardware, such as a cluster with GPUs installed on its nodes, creates a new problem of distribution of tasks among processors in such a way that maximum performance is achieved. To accomplish this task, a resource manager has been developed that helps a working process to take the decision if it should use GPU or CPU for the minimization of task running time.

The manager has the ability to predict the job running time, basing on the statistics accumulated on previously completed tasks.

Besides, the manager keeps track the loading of hardware devices and distributes the tasks such that the loading level does not exceed certain device-specific level.

The manager sets the correspondence between processors of the node and a set of n counters N_i, $(i = 1 \dots n)$. Counters having numbers $i = 1 \dots n-1$ correspond to GPUs installed on the node, and the last counter corresponds to CPUs. The possibility of allocation of a separate counter for each CPU of a node is easily implemented but seems useless in our case.

When the manager is requested for a processor, the client process specifies some quantity of resource units ΔN_i, $i = 1 \dots n$ that is to be used for processor i (CPU or GPU) in the case of allocation of that processor. The manages uses the specified ΔN_i and determines the time t_i^b when the value of the counter N_i becomes greater than ΔN_i. As mentioned above, the manager is able to predict the duration of an operation, τ_i, basing on the statistics of previous calculations. Firstly, this is necessary to find t_i^b and, secondly, for the prediction of time of completion of operation $t_i^e = t_i^b + \tau_i$ at i-th processor. Finally, the processor is being chosen for the task, at which predicted time of task completion t_i^e is minimal.

Let us notice that the client process is put into waiting state until time instant t_i^b. After that, the client obtains the number of processor i, and the value of resource counter N_i is decreased by ΔN_i units.

As soon as there is one resource manager per node, it serves several client working processes running on the node, and client requests for processor resources form, in general, a queue of requests.

The meaning of units used in counters N_i remains an open question. Obviously, the value ΔN_i characterizes the hardness of the task: the more it is, the less number of tasks can share the usage of the processor.

For counter N_n (CPU) we assume for all tasks that $\Delta N_n = 1$, i. e., the reason of current value of the counter is the number of processes that can be currently run on CPU. For GPU (counters with $i < n$) the value ΔN_i is currently proportional to the required memory size. Of course, such a parameter can be introduced in infinitely many ways. In particular, it appears reasonable to set for GPU counter the same reason as for CPU counters: the number of processes that can be simultaneously run on the device.

6 Testing software implementation

As a test problem, a cube-shaped volume has been considered. The cube has sizes $5 \times 5 \times 5$. The mesh of tetrahedra having $500 \times 500 \times 500$ nodes is specified, and the nodal field values are calculated by the following formula:

$$f(x,y,z) = 2\cos(10x) + 2\sin(10y) + \cos(10z);$$

the isosurface was generated for the level of $f = 0.5$; its fragment is shown in Fig. 10. The software implementation of algorithms of isosurface generation and reduction has been tested on a node having 12 Gb operating memory, 16 CPU cores and two Tesla GPUs having 4 Gb memory each.

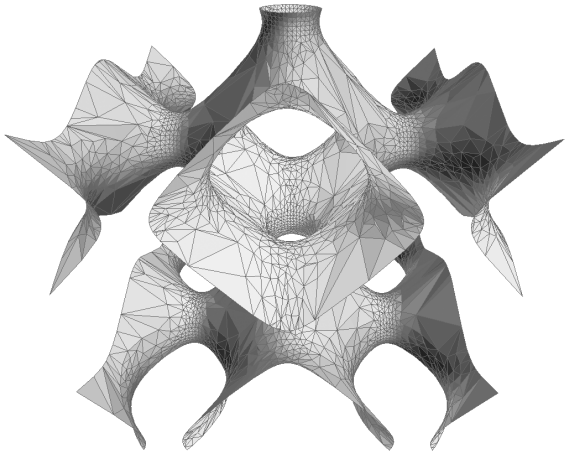

Fig. 10 Fragment of test isosurface

Three test series have been carried out, in which domain sizes changed as follows:

- 250 domains having $5 \cdot 10^5$ nodes each;
- 125 domains having 10^6 nodes each;
- 63 domains having $2 \cdot 10^6$ nodes each.

Let us notice that in each of these cases, approximately equal-sized isosurface part bbelongs to each domain.

The dependencies of calculation time on the number of working MPI processes and on the domain size have been considered. The plots are presented in Fig. 11. Notice that in this test no GPUs have been used.

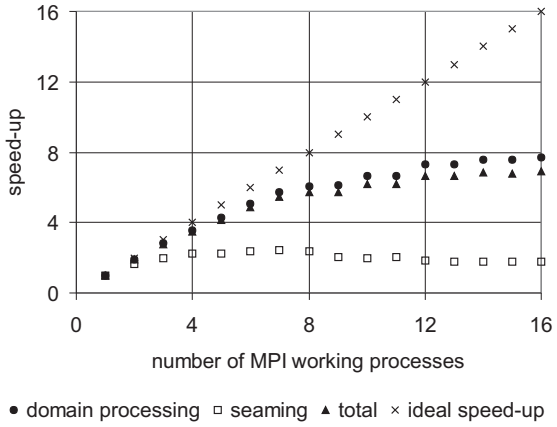

Fig. 11 Speed up of isosurface calculation with growth of number of MPI work processes used

7 Conclusions

The testing of the developed software implementation of algorithms of isosurface generation and reduction has shown its workability and scalability in case that number of processors used is significantly less than the number of domains in the original mesh.

The analysis of tests has shown that the bottleneck for the decrease of speed-up with the growth of CPU count is most likely the memory bandwidth. The more processes are run, the more the load on memory subsystem is.

The attempt to use GPUs installed on the system has not led to any growth of performance, thow isosurface reduction algorithm runs on a GPU much faster than on a CPU. The research in this area is currently in progress.

References

[1] Treece G. M., Prager R. W. and Gee A. H. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics*, Vol. 23, No. 4, pp 583–598, 1999.

[2] Anderson J.C., Bennett J. and Joy K.I. Marching Diamonds for Unstructured Meshes. *IEEE Visualization 2005*, pp 423–429, 2005.

[3] Rossignac J. and Borel P. Multi-resolution 3d approximations for rendering complex scenes. *Modeling in Computer Graphics: Methods and Applications*. Falcidieno B., Kunii T., (Eds.), Springer Verlag, pp 455–465, 1993.

[4] Lindstrom P. Out-of-core simplification of large polygonal models. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, pp 259–262, 2000.

[5] Garland M. and Heckbert P. S. Surface simplification using quadric error metrics. *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, pp 209–216, 1997.

[6] Attali D., Cohen-Steiner D. and Edelsbrunner H. Extraction and simplification of iso-surfaces in tandem. *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, pp 139–148, 2005.

[7] Dupuy G., Jobard B., Guillon S., Keskes N. and Komatitsch D. Parallel extraction and simplification of large isosurfaces using an extended tandem algorithm. *Computed Aided Design*, Vol. 42, No. 2, pp 129–138, 2010.

[8] S. Gorlatch, C. Lengauer. (De)Composition for Parallel Scan and Reduction. *Proc. 3rd Working Conf. on Massively Parallel Programming Models (MPPM'97)*, pp 23–32, 1997.

[9] Thrust — a CUDA library of parallel algorithms. http://code.google.com/p/thrust/

8 Contact Author Email Address

mailto:kuzin_aleksei@mail.ru

Copyright Statement