Bull. Nov. Comp. Center, Num. Anal., 15 (2011), 85–90

A method for solving the mass problem of belonging a set of points to arbitrary coverings on GPU

D.Yu. Tribis

Abstract. This paper describes a numerical algorithm and results of numerical calculations of the mass problem solution of determination of belonging of a set of points to a set of arbitrary figures covering an area. Such figures could be irrelative crossed or not. The problem is solved by the earlier described methods of Geometrical Informatics (GI) which is a new approach to the organization of calculations on the GPU (Graphics Processor Unit). In this paper, we present results of comparison between the time of execution of the fastest classical algorithms on the CPU and that of the algorithm GI on the GPU. We show that with some problems, the accelerations obtained can reach from 6 up to 700 times.

Though as a modeling problem we take a geophysical model of the Earth and the construction of a grid for subsequent numerical geophysical calculations is chosen, such problems are typical of many areas of constructing grids for numerical experiments, geometrical modeling, everywhere where coverings are used, triangulation problem or problems of rendering.

Introduction

A rapid growth of the efficiency of the GPU and an increase in the difference between the performance of the GPU and the CPU in the last few years [1] has generated interest in calculations on graphics processors. Though such decisions as CUDA [2], provide the programmer with possibility of organizing of parallel calculations, they demand the fundamental knowledge of parallel programming methods, as well as of the GPU architectures. The new algorithms are necessary for devices with mass parallelism, therefore the programmer is obliged to completely rewrite an old algorithm in rapidly changing hardware environment. As a result, programs are often incompatible with different graphics accelerators.

In the GI approach [3, 4], the problem is reduced to a sequence of two- or three-dimensional images (stages). From the standpoint of parallel execution, the algorithms GI offer advantages over the classical way because they always have several levels of parallelism. It is obvious that not always programs realized in the GI paradigm, will be more effective, rather than classical, but with the advent of super-power graphics accelerators, this direction becomes interesting from the practical point of view. The GI does not demand a high professional skill from the programmer as well as un-

derstanding of details of the GPU functioning and even a deep knowledge of parallel programming. Actually, it starts with the fact, that the GPU's are first of all developed for graphic operations of drawing under which their hardware architecture is optimized, therefore the maximal productivity should be expected on operations of drawing.

As for intuitive clearness of the programming process the GI uses a number of simple methods, such as the Painted Prioritized Projections [3], where for a solving task one uses colors, operations with them and sequences of drawing different sets of figures. As will be shown below on some problems and, in particular, on problems of the belonging of points to coverings, which are used for constructing grids, these methods offer a significant acceleration in comparison with classical ones. Currently, in terms of the GI, there are formulations for many classical computing problems, such as calculations of sections of arbitrary figures with any curves, mutual crossings, additions of any figures, the search for points being in the surroundings, triangulation, construction of a convex hull of a finite set of points, numerical integration, a set of problems from the area of descriptive geometry, operations with sets, proofs of calculations of predicates.

This paper is devoted to a detailed comparison, based on the numerical experiment of productivity of classical algorithm and GI algorithm on mass problems of the belonging a set of points to a set of arbitrary figures. This is widely used, in particular, when constructing grids for numerical methods as part of numerical calculations in various physical and mathematical problems.

1. Numerical testing task

As a testing task we choose the task of building a grid above representation of a real model of the ocean bottom in the area of Brazil (Figure 1). Two models were created based on this real model: a rough representation with 14 layers (Figure 2) and a more exact representation with 1400 layers (Figure 3).

Each layer represents a polygon whose number of nodes varies from 7 up to 40 with prevalence of polygons with the number of nodes nearly 30, this size will be accepted as a characteristic number of nodes. We accept that in a two-dimensional case, at the preparation stage of data for further numerical modeling, 1000×1000 grid for rough calculations and 5000×5000 grid for exact cases should be constructed. Thus, for a rough model, in most "easy case" with a minimal grid, we need to solve $1000 \times 1000 = 1{,}000{,}000$ tasks of belonging a point to 14 triacontahedrons, and in an exact case, $5000 \times 5000 = 25{,}000{,}000$ for 1400 triacontahedrons.

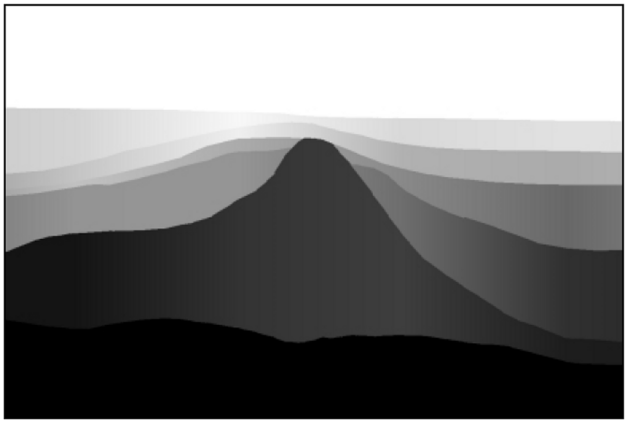

Figure 1. Original model of the ocean below bottom

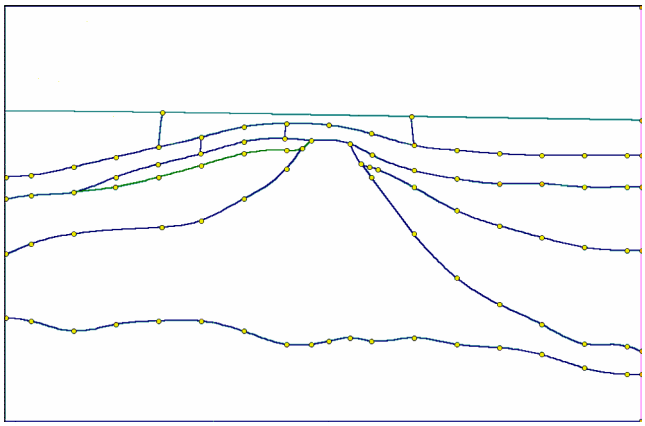

Figure 2. Rough representation with 14 layers

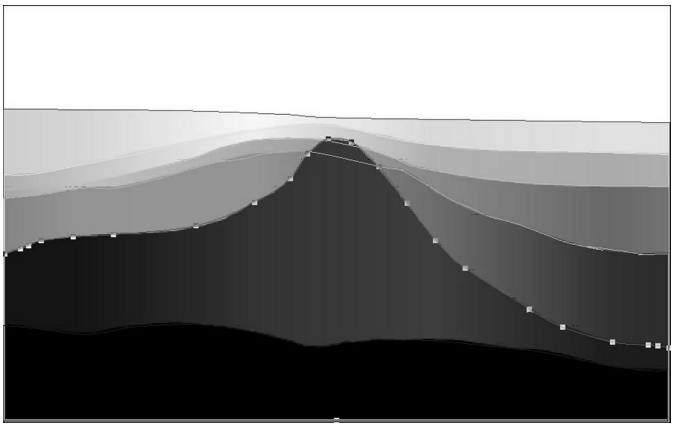

Figure 3. The detailed representation with 1400 layers

2. The numerical experiment

We especially choose as much as possible adverse conditions for the GI hardware environment, namely: old and slow graphic processor ATi Radeon Graphics Processor X700, DAC 40 Mhz, 128 Mb against the fast central processor with clock frequency 3 Mhz and memory 1Gb under control of Windows XT to show advantages of the GI approach, rather than power of modern graphic processors. All operations of drawing are synchronous, i.e., before drawing a next polygon, we expect that drawing of the previous one is completely finished. Let us show how even for such bad conditions when the GPU on a single operation looses CPU by the factor ten and more, the resulting time for the GPU for the full modeling task is better on more than factor ten.

Programs have been written in C++ language and compiled with Microsoft Visual Studio 2008. The program of belonging a point to a polygon has been taken from "Wikipedia" [5] as, in their opinion, the fastest existing algorithm. It includes just a few operations of comparison.

Figure 4 shows results for execution a single task of belonging a point to a polygon with a different number of nodes. Performance of the single task depending on the number of nodes is shown in the table.

As we can see from Figure 4, all the dependencies of execution time on the number of tasks are linear. The productivity rapidly falls with increasing the number of nodes of polygons. We should notice that the CPU on this task shows a good productivity, carrying out one million tasks per 0.656 second. The GPU on a single task loses, approximately by the factor of 10–12. Even the operation of obtaining a pixel (GetPixel) is almost twice as

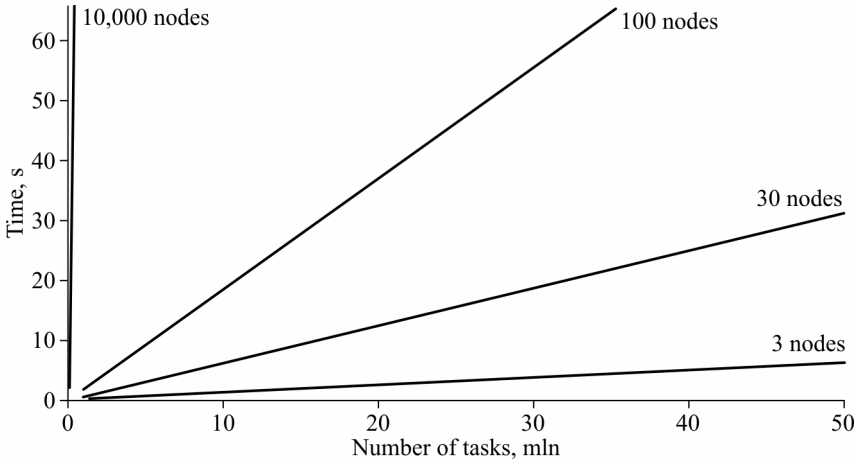

Figure 4. Dependencies of execution time on the number of nodes and the number of solved tasks

CPU/GPU productivity on a single task

Nodes	Classic, mln/s	GI, mln/s	Cl./GI
3	8 (0.125 s)	0.641 (1.56 s)	12.480
30	1.524 (0.656 s)	0.152 (6.75 s)	10.026
100	0.528 (1.891 s)	0.046 (17.19 s)	9.09
10000	$5.614 \cdot 10^{-4}$ (30 min)	$7.874 \cdot 10^{-4}$ (21 min)	

long (one million per 1.26 second). Though, with an increase of the number of nodes, the situation is slowly changing. However, extremely slowly, the productivity becomes comparable for several thousands of nodes that has no practical value in our case.

3. Performance comparison

CPU performance:

Rough case: 1000×1000 tasks for 14 triacontahedrons or $1.524 \times 14 = 21.336$ s.

Exact case: 5000×5000 tasks for 1400 triacontahedron or $25 \times 1.523 \times 1400 =$ 53,340 s or about 15 hours.

There is one characteristic feature of solving such tasks on the GPU with using the GI methods, namely: in each case, even in solving a single task, the algorithm turns out to be not only for one point but for the whole area. Thus, in the case when it is necessary to solve many tasks of belonging to different points but to the same figure or even to the same figures, there is no necessity to draw them each time. In this case, the whole task is reduced to rather a fast obtaining of a pixel, which, as was shown above, is still twice slower than one task on the CPU. However, in the case of CPU, such tasks should be solved for each figure, i.e., in a rough case, for 14 and, in the exact one, for 1400 figures, and we have a significant gain in complete time for the GPU as a result. In a rough case, we will have about 7 times, and in the exact case — more than 700 times acceleration.

GI algorithm performance on GPU: in the rough case, one task of drawing 14 triacontahedrons (this time can be ignored), 1000×1000 tasks of setting a pixel (SetPixel) and obtaining a pixel (GetPixel), and we have:

Rough case: $1.26 \times 2 = 2.52$ s for one million tasks.

Exact case: $1.26 \times 2 \times 25 = 63$ s for 25 million tasks.

Conclusion

This paper proposes a new approach, the Geometric Informatics, for solving the mass problem of determination of belonging a set of points to a set of arbitrary figures making the covering of an area. Though, in the example presented, the GPU was pretty limited from the point of view of hardware, we still achieve a significant acceleration in comparison with the execution on CPU. This allows us to expect even greater accelerations on similar problems for more powerful GPUs.

References

[1] iXBT.com. — http://www.ixbt.com.

[2] NVidea, Cuda Zone. — http://www.nvidia.ru/object/cuda_home_new_ru.html.

[3] Il'in V.P., Tribis D.Yu. Geometrical informatics of models of complete media // Vycislitelnye Metody i Programirovanie. — 2009. — Vol. 10. — P. 306–313.

[4] Il'in V.P., Tribis D.Yu. Geometrical informatics of models of complete media // Proc. the 18-th Intern. Conf. on Computer Graphics and Vision: GraphiCon, 2008. — Moscow: Moscow State University, 2008.

[5] Wikipedia, Algorithm of a point in a polygon. A very rapid algorithm. — http://ru.wikipedia.org/.