Cellular GPU Model for Structured Mesh Generation and its Application to the Stereo-Matching Disparity Map

Naiyu ZHANG, Hongjian WANG, Jean-charles CREPUT, Julien MOREAU and Yassine RUICHEK
Laboratoire IRTES-SET, UTBM, 90010 Belfort cedex, FRANCE
Email: {naiyu.zhang, hongjian.wang, jean-charles.creput, julien.moreau, yassine.ruichek}@utbm.fr

***Abstract*—This paper presents a cellular GPU model for structured mesh generation according to an input stereo-matching disparity map. Here, the disparity map stands for a density distribution that reflects the proximity of objects to the camera in 3D space. The meshing process consists in covering such data density distribution with a topological structured hexagonal grid that adapts itself and deforms according to the density values. The goal is to generate a compressed mesh where the nearest objects are provided with more details than objects which are far from the camera. The solution we propose is based on the Kohonens Self-Organizing Map learning algorithm for the benefit of its ability to generate a topological map according to a probability distribution and its ability to be a natural massive parallel algorithm. We propose a GPU parallel model and its implantation of the SOM standard algorithm, and present experiments on a set of standard stereo-matching disparity map benchmarks.**

***Index Terms*—Parallel Cellular Model; Mesh Generation; Stereo Disparity Map; Self-Organizing Map.**

I. Introduction

In the field of artificial vision, robot navigation and 3D surface reconstruction, a lot of work has been done to develop effective stereo-matching algorithms producing high quality disparity maps. Such a disparity map is obtained by a matching process of the two left and right images obtained by a stereo camera. The disparity map is a 2D image, which represents the 3D surface seen by a camera. But few approaches of stereo-matching and surface reconstruction currently match the requirements of real-time execution. Algorithms on Graphics Processing Units (GPU) are now developed to respond to this problematic. Furthermore, the volume of data stored and manipulated in the disparity map is often very important and may constitute an obstacle for real-time algorithms. Here, we address the problem of building, in a massively parallel way and by using GPU, a compressed and adapted structured mesh representing a given disparity map. The goal is to represent the 3D scene in an efficient way, with more details for objects closed to the camera than far from it. The compressed structured mesh obtained in 2D space as it is the disparity map, should then allows for fast treatment or visualization in 3D space.

To achieve such a goal of structured compressed mesh generation, we choose to implement the Kohonens [1], [2] self-organizing map (SOM) algorithm in a massively parallel way on GPU. The disparity map is the input. It is used as a 2D density distribution, on which the algorithm operates in a massive parallel fashion by deformation of a 2D hexagonal grid, called the neural grid. The grid deformation must respect the density distribution and topology. This means that high disparity values will be represented by higher densities of the corresponding neural grid points and that the neural structured grid will reflect the spatial topology or distances in 2D and 3D space. The SOM can also be seen as a center based clustering algorithm with topologic relationships between cluster centers. The neural grid is a visual pattern that adapts and modifies its shape according to some underlying distribution. While mesh generation is a large area of research [3]–[6], few approaches exist on structured mesh generation according to a density distribution. For example, [7], [8] deal with this subject but the implementation were sequential and not competitive. Approaches in [9], [10] have proposed fast generation methods for meshes but since their methods are too memory consuming, they can not handle large size problems. In [11]–[13] the self-organizing map is combined with mesh generation and surface reconstruction but the method never refer to the cellular space for further studies. In [14], it is discussed combination of self-organizing map and cellular automata in meshing processing, but the paper does not explore deeper neither in the properties of this combination nor the possibility of execution on parallel architectures. From our knowledge, we did not found GPU parallel implantation of the SOM algorithm when applied in 2D or 3D space.

One important point of the parallel model presented in this paper, is that it proceeds by a cellular decomposition of the data, i.e. the disparity map, in 2D space, such that each processing unit represent a constant and small part of the data. Hence, according to the increase of parallel processing units in the future, the approach should be more and more competitive, while at the same time being able to deal with very large size disparity images. This property holds because of the linear memory needed according to the image size. Hence, large size disparity maps can be compressed in an efficient and near real-time way. We will illustrate that point in experiments on large size input disparity maps.

This paper includes the following key characters:

- Propose a cell partition model for the parallel self-organizing map. This model is advantageous for large scale applications.

978-0-7695-5140-1/13 $26.00 © 2013 IEEE
DOI 10.1109/ISM.2013.18

- Verify the strategy by GPU-based implementation. Carry a series of elaborate experiments. Compare the results, with regard to both result quality and running time, to its corresponding serial implementation. Offer an in-depth analysis.
- Apply the model to stereo disparity maps and reconstruct the 3D visualization from the structured mesh obtained.
- Efficient system implementation on GPU with CUDA.

In the following parts of the paper, we present a description of the self-organizing algorithm, problem attempts and cellular parallel model in section II, we detail the GPU implementation of the model in section III, and then we present the experimental results in section IV. A conclusion terminates the paper.

II. Problem statement and parallel algorithm

In this section, we first present the standard online and sequential self-organizing map algorithm as stated originally by Kohonen [1], [2]. We present the problem of structured mesh generation as a balanced clustering problem in the plane dealing with an hexagonal topological grid (the neural network grid or mesh) that must adapt to an underlying distribution in the plane (the disparity map). Then, we detail the parallel model proposed for the SOM implantation on GPU.

A. The Kohonens self-organizing map

The standard self-organizing map [1], [2] is a non directed graph $G = (V, E)$, called the network, or topological grid, or structured mesh, where each vertex $v \in V$ is a neuron having a synaptic weight vector $w_v = (x, y) \in \Re^2$, where $\Re^2$ is the two-dimensional Euclidean space. Synaptic weight vector corresponds to the vertex location in the plane. The set of neurons *V* is provided with the d_G induced canonical metric $d_G(v, v') = 1$ if and only if $(v, v') \in E$, and with the usual Euclidean distance $d(v, v')$.

The training procedure structure when applied to a structured mesh with hexagonal topology is given in Algorithm 1. A fixed amount of t_{max} iterations are applied to a mesh, the vertex coordinates of which being initialized to a regular grid (each vertex has 6 neighbors). Here, the data set is the set of pixels of the disparity map. Note that the disparity map stands for a density map, where each pixel value represent some density value. Each iteration follows three basic steps. At each iteration *t*, a point $p(t) \in \Re^2$ is randomly extracted from the data set (extraction step) according to a roulette wheel mechanism depending on the disparity values. Then, a competition between neurons against the input point *p(t)* is performed to select the winner neuron n^* (competition step). Usually, it is the nearest neuron to *p(t)*. Finally, the learning law (triggering step) presented in equation 1 is applied to n^* and to the neurons within a finite neighborhood of n^* of radius σ_t, in the sense of the topological distance d_G, using learning rate $\alpha(t)$ and function profile h_t. The function profile is given by the Gaussian in equation 2. Here, learning rate $\alpha(t)$ and radius σ_t are geometric decreasing functions of time. To perform a decreasing run within t_{max} iterations, at each iteration *t*, coefficients $\alpha(t)$ and σ_t are multiplied by $exp(ln(\chi_{final}/\chi_{init})/t_{max})$ with respectively $\chi = \alpha$ and $\chi = \sigma$, χ_{init} and χ_{final} being respectively the values at starting and final iteration.

$$w_n(t+1) = w_n(t) + \alpha(t) \times h_t(n^*, n) \times (p(t) - w_n(t)) \quad (1)$$

$$h_t(n^*, n) = exp(-d_G(n^*, n)^2/\sigma_t^2) \quad (2)$$

Examples of a basic iteration with different learning rates and neighborhood sizes are shown in Fig. 1. Application of SOM to the stereo disparity map consists of applying the training procedure to a structured mesh (the network) according to a density map that is a simple transformation of a disparity map. This simple transformation consists, before the training starts, in removing background values (very low disparity values) and increasing contrast of disparity values. This is done in order to increase the data point density for closest objects in the image. Here, density values are set to the square of the disparity values. Note that a SOM simulation is characterized by the 4 running parameters $(\alpha_{init}, \alpha_{final}, \sigma_{init}, \sigma_{final}, t_{max})$.

B. The structured mesh generation optimization problem

In this paper, the application to structured mesh generation is presented as the solution of a balanced clustering optimization problem in the plane. The goal is to homogeneously divide a density map, the disparity map in our application, between the many triangles of the structured hexagonal mesh.We now present our notation and formalization of these aspects.

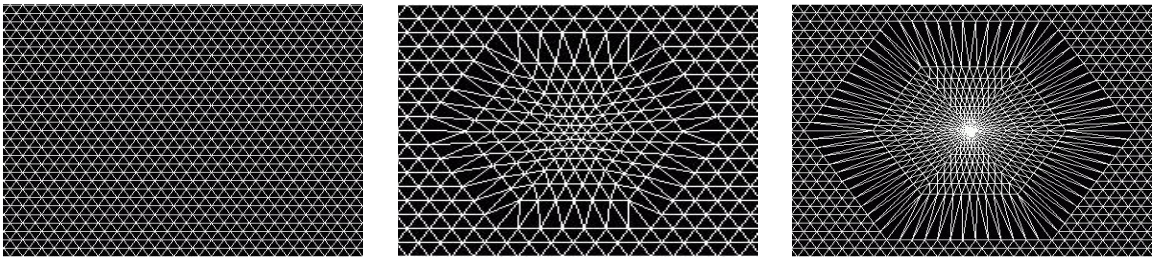
The definition of the balanced structured mesh optimization problem needs to consider both the hexagonal grid (the mesh) and the underlying density distribution (the disparity map). The Fig. 2 illustrates the main elements. The target adaptive hexagonal mesh is illustrated in the left part of the figure. It is defined as a set of hexagonal cells, each one containing six subdivided triangles. These basic honeycomb cells are the units used to evaluate the amount of the underlying pixels they cover. The right part of the figure shows such an hexagonal cell and its covering pixels from the density map. In the example of the figure, the total value covered, called the weight of the honeycomb cell, is the summation of the underlying pixel values.

Let M_k be the set of honeycomb cells of the mesh. Let W_k the weight of a single honeycomb cell, defined as the sum of the underlying pixel values from the disparity map. Note, that this weight can be computed by using a standard pixel coloring algorithm.

Let W be the average weight of the K honeycomb grid cells defined by equation 3. We define the optimization problem as the minimization of the average percentage deviation of each individual honeycomb cell weight to the average honeycomb cell weight as defined in equation 4. Hence, the structured mesh generation problem consists in minimizing this criteria while preserving the regularity of hexagonal topology, such geometrical constraints being only visually verified in this paper.

Algorithm 1 Self-organizing map training procedure.

1: Randomly generate a regular grid of neurons.
2: **for** $iter \leftarrow 0$ To t_{max} **do**
3: Randomly extract a point p from the data set.
4: Perform competition to select the winner neuron n^* according to p.
5: Apply learning law to move the neurons of a neighborhood of n^*.
6: Slightly decrease learning rate α and radius σ of neighborhood.
7: **end for**



(a) (b) (c)

Fig. 1: A single SOM iteration with learning rate α and radius σ. (a) Initial configuration. (b) $\alpha = 0.5, \sigma = 6$. (c) $\alpha = 1, \sigma = 12$.

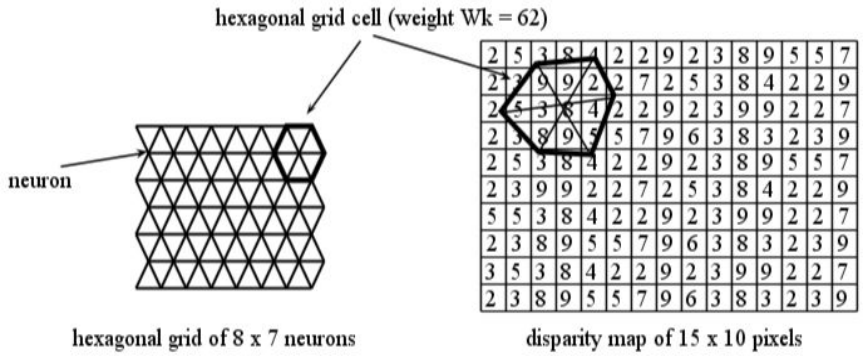


Fig. 2: Hexagonal structured mesh with honeycomb cells and triangular subdivision (left), density map covering with of a single honeycomb cell with weight $W_k = 62$.

$$W = \frac{\sum_k W_k}{K} \tag{3}$$

$$Cost = 100 \times \frac{\sum_k |W_k - W|}{K \times W} \tag{4}$$

C. Parallel Cellular Model

It is intuitive that the disparity map and the structured mesh built by the SOM algorithm are connected by sharing the same Euclidean space. The space is defined by the disparity map dimensions of size $W \times H$, and each grid point can move on the density map.Given an input data density map of size $W \times H$, a two-dimensional topological grid will be created, of a given size $W_g \times H_g$. Note that each grid node is indexed in its grid, and that the size of the grid is lower and in relation to the size of the density map in such a way that the grid constitutes a compressed representation of it. Note also that each grid node has coordinates (neuron weights) in the Euclidean plane, and that these coordinates are defined by the dimensions of the density map.

In order to implement the parallel level, at which parallel execution will takes place, we now introduce a supplementary level of decomposition of the plane and input data. Between the topological grid and the density map, we now introduce a two-dimensional cellular matrix of size $W/co \times H/co$, where co is a constant factor. The three main data structures of the parallel model are illustrated in Fig. 3. This intermediate cellular matrix is in linear relationship to the input size. Its role will be to memorize the grid nodes in a distributed fashion and authorize many parallel closest point searches in the plane by a spiral search algorithm [15].

Each cell in the cellular matrix is a basic training unit and

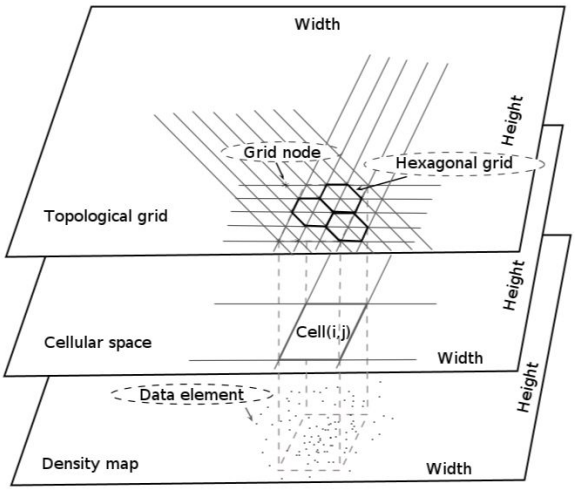


Fig. 3: Parallel cellular model: the input density map, the cellular matrix and the topological grid. To a given cell of the cellular matrix corresponds a part of the input data as well as a part of the structured hexagonal mesh.

will be handled by one GPU thread during the iterations of the SOM processing. This is at this level that massive parallelism takes place. Since the cellular matrix division is proportional to the input density map, and the GPU threads correspond one-to-one with the cells respectively, then, the GPU threads are also in linear correspondence to the input density map size, in $O(W \times H)$.

Now, we just have to define the many SOM processes that are implemented by the many cells/threads corresponding to the cellular matrix decomposition. The goal is to implement the four steps of the SOM algorithm in a distributed way. These steps are the input data point extraction step according to the density map, the closest point, or winner neuron, finding step, the application of learning rule step, and the parameter decrease step. Then, this process performed independently in parallel by the many cells/threads will be repeated a given number of times as stated by the parameter t_{max} of the original SOM algorithm. Note that t_{max} is now the number of parallel executions, rather than the number of sequential iterations. We now give more details of each SOM main steps. The Algorithm 2 resumes the parallelized SOM algorithm with cell partition.

$$p_i = (\frac{q_i}{\max\{q_1, q_2, \ldots, q_{num}\}}) \times \delta \tag{5}$$

First is the problem of random data point extraction by a roulette wheel mechanism performed in parallel and according to the density distribution. As a solution to this problem, we propose a particular cell activation formulae in equation 5 to determine if a cell/thread will be activated at each parallel iteration. Here, the p_i is the probability that the cell *i* will be activated, q_i is the sum of the pixel values of the density map corresponding to the cell *i*, i.e. the density of the cell, and *num* is the total number of cells. Then, a cell is activated proportionally to its density value according to the whole density map. Hence, the cell with higher density will be activated with probability one. The empirical preset parameter δ is used to adjust the level of activity of the threads in order to avoid too many memory access conflicts. In this paper, parameter δ will be set to 1.

The activation probability of each cell is pre-computed on GPU before the parallel SOM operations begin.

$$prob(i) = \frac{d_i}{\sum_{j=0}^{cellNum} d_j} \tag{6}$$

To complete the extraction step, each activated cell/thread simply performs a local roulette wheel mechanism, into the cell itself, in order to produce the extracted pixel. The probability of a pixel choice local to a cell is defined by equation 6, where *cellNum* is the number of pixels in the cell, and d_i the density value of a pixel in the cell.

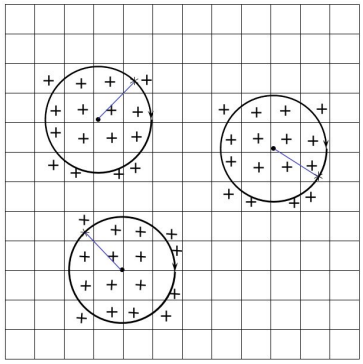


Fig. 4: Spiral searches performed in parallel.

After the extraction step, each activated GPU thread has gotten its input data point for further operations. The next job is to find out the nearest topological grid node corresponding to the extracted point. Each GPU thread, with its input extracted point, performs a spiral search [15] on the cellular matrix that contains the topological grid nodes as illustrated in Fig. 4. The spiral search proceeds from the cell that contains the extracted point, as in the middle of the circle of the figure and progressively extends the search to the cells surrounding until a grid node is found. Once there is one grid node that is found, it is guaranteed that it is not necessary to search any cell that does not intersect the circle of radius equal to the distance to the first grid node found and centered at the extracted point. It is worth noting that a single spiral search process takes *O(1)* computation time on average for a bounded distribution according to the instance size. Then, one of the main interests of the proposed approach is to allow the execution of approximately *N* spiral searches in parallel, where $N = W \times H$ is the problem size, thus transforming a $O(N)$ sequential algorithm search into a constant time *O(1)* theoretical parallel algorithm in the average case for bounded distributions. This is what we call "massive parallelism", the theoretical possibility to reduce computation time by factor *N*, when solving a Euclidean NP-hard optimization problem.

Algorithm 2 Parallel self-organizing map on-line algorithm.

1: Generate a regular grid of neurons;
2: Calculate the activated probability for each cell according to the cell activation formulae;
3: **for** $iter \leftarrow 0$ To t_{max} **do**
4: Check in parallel whether $cell_i (i = 1, 2, \ldots, num)$ is activated or not;
5: **if** $cell_i$ is activated **then**
6: Randomly extract a point (element on density map) *p* from the $cell_i$;
7: Perform competition to select the winner neuron n^* according to *p*;
8: Apply learning rule to move the neurons of a neighborhood of n^*;
9: Wait until all the other cells finish their works;
10: Slightly decrease learning rate α and radius σ of neighborhood
11: **end if**
12: **end for**

In the learning step, each GPU thread performs the displacements of the grid nodes starting from the winner closest grid point in the plane according to the learning rule. The displacement involves not only the grid node found (winner neuron), but also its closest neighbors in the sense of the topological distance into a neighborhood of radius σ_t. (shown in Fig.1 in section II-A). The modifications of the grid node locations is done according to equation 1 and equation 2 of section II-A.

The last step in SOM processing is to decrease the learning parameters. The main operation is to modify the values of the learning intensity and the radius of neighborhood. This specific work is carried out on CPU for convenience in order to avoid the load and duplication of the parameters into the GPU memory space.

III. Platform and CUDA implementation

A. Platform background

Our experiments are carried out both on CPU and GPU. For the CPU, it is a Intel(R) Core(TM)2 Duo CPU E8400, 3.00GHz, each of the two cores with a cache of 6144KB. The GPU used in the experiments is a GPU GeForce GTX 570 of NVIDIA. It has 15 streaming multiprocessors of thirty two cores each, GPU clock speed 1.54GHz, Memory clock rate 2000MHz, Memory Bus width 320 bits while the total amount of global memory is 1280 Mb (constant memory 65536 Kb, shared memory per block 49152 Kb). The system on which the experiment is performed is Ubuntu 11.04, 32 bits.

The programming interface we used for parallel computation on GPU is the Compute Unified Device Architecture (CUDA). The parallel computation work is realized by a *kernel* function, which is executed concurrently by multiple threads on data elements. All these threads are organized into a two level concepts: CUDA *grid* and *block*. A *kernel* has one CUDA *grid*, which contains multiple *blocks*. Every *block* is formed of multiple threads. The dimension of CUDA *grid* and of *blocks* can be one-dimension, two-dimension or three-dimension. The performance of GPU with CUDA is closely related to thread organization and memory accesses, which should attract much attention according to various computation works and GPU platform.

B. CUDA implementation

Based on our experimental platform, given an image of size *W*×*H*, we first determine the size of the cellular matrix that defines the total number of threads used. To each cell of the cellular matrix is assigned a single thread. Since we want a linear relationship between the thread number and the image size, we divide each dimension of the image by a constant factor that will be the same for all experiments. Based on such image decomposition between threads, we define the size of each CUDA *block*, and deduce the number of *blocks* necessary, hence the size of the CUDA *grid*. It appears in experiments, because of thread divergence within *blocks*, that the best performance was reached with *blocks* having each a single thread. Hence, only parallelism from independent streaming multiprocessors appears to have some important benefit in this implementation. Threads within warps inside a same block are thus subject to warp divergence that introduces serialization on the computation.

The CUDA code is composed of an initialization phase, followed by the iteration of the main loop of the algorithm. At the initialization, two simple *kernels* are created. These two *kernels* are used to compute in a distributed way the random numbers and activation probabilities respectively, that are both assigned to each cell. The data generated by these two *kernels* are kept on the global memory space for following steps.

For the main loop iteration, we employ two *kernels* to carry out the SOM processing. The first *kernel* is responsible to refresh the cellular matrix. Each cell has data structures where to deposit information of the number and indexes, in the neural network grid, of the neurons it contains. This information may change at each iteration, but it appears that it can be sufficient to make the refreshing based on a lower rate defined by a coefficient rate parameter. The second *kernel* handles the main process of SOM: the extraction of pixel, finding the closest topological grid node by spiral search method, and learning step. Note that these two *kernels* are executed t_{max} times and that the number of parallel training procedures executed at each iteration depends on the number of simultaneous activated cells.

TABLE I: EXPERIMENT PARAMETERS.

$MNPC^1$	α_{init}	α_{final}	σ_{init}	σ_{final}	CPU_CRR^2	GPU_CRR^3
200	1	0.01	24	1	4 000	50

[1] Maximum Nodes Per Cell. [2] CPU_cell refresh rate. [3] GPU_cell refresh rate

TABLE II: EXPERIMENT PARAMETERS FOR COMPARATIVE EVALUATIONS ON CPU AND GPU

Image	Image size	grid size	cellular size	iteration_CPU	iteration_GPU
Tsukuba	384×288	64×48	20×15	1 000 000	1 500
Venus	434×383	110×96	22×20	1 000 000	1 500
Teddy	450×375	150×125	23×19	1 000 000	1 500
Cones	450×375	150×125	23×19	1 000 000	1 500

TABLE III: EXPERIMENT PARAMETERS FOR LARGE SIZE IMAGES 'CONES' .

Image size	grid size	cellular size	iteration_CPU	iteration_GPU
450×375	150×125	23×19	1 000 000	1 500
900×750	300×250	45×38	1 000 000	1 500
1800×1500	300×250	90×75	2 000 000	1 500

IV. EXPERIMENTS RESULTS

In this section, the parallel cellular model is mainly experimented with two set of tests: the first set of tests consists of a comparative evaluation between the GPU and the CPU applied on four disparity images of small sizes, the second set of tests concerns the application to larger size disparity maps in order to compare GPU and CPU as the image size increases. All the images used are from the stereo dataset of Middlebury [16]. The parameters which are kept constant for all tests on CPU and GPU are given in Table I. For each of the two set of tests, the supplemental parameters are presented in Table II and III respectively.

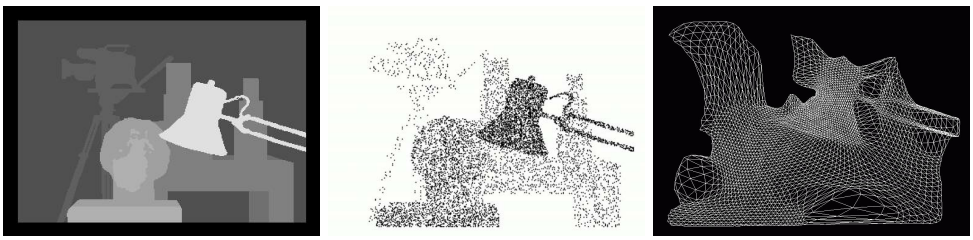
In Table II, are given the parameters for the first set of experiments. The neural network grid size, called mesh size, is obtained by $W/co \times H/co$. Coefficients co are set to 6, 4, 3, 3 for the four disparity images respectively. Because of its size, the mesh can be considered as a compressed representation of the 3D surface defined by the disparity map. The cellular matrix sizes are obtained by $W/20 \times H/20$ for all images. It should be noted that the number of iterations on CPU corresponds to serial operations, and that the number of iterations on GPU corresponds to parallel operations. In Table III, are given the parameters for the second set of experiments. The coefficient co for the three different image sizes are 3, 3, 6 respectively. Considering parameters from Table I, only the initial radius of neighborhood σ_{init} is set to 48 for the two largest images and for the CPU version only. It was the only way to improve the CPU behavior within moderate computation time as the size grows. The cellular matrix sizes are still obtained by $W/20 \times H/20$.

In the first set of experiments on CPU and GPU, four input disparity images are considered with varying characteristics of sizes and textures. The comparative results are presented in Table IV and two examples of visual results are presented on Fig. 5 and Fig. 6. The result values in the table are average values over five runs for each disparity image. The parallel cellular model on GPU outperforms the serial model on both computation time and cost minimization. Computation time takes about from 0.7 seconds to 1.2 seconds on GPU and from 3.2 seconds to 4.2 seconds on CPU. This leads to an acceleration factor of about 4 or 5. Besides the speedup, the parallel cellular model may increase the quality of results of about 1 or 2 percents for three of the images.

The Fig. 5(a-c) illustrates a result on the Tsukuba image. In (a), is shown the disparity map. In (b), is shown a sampling of the disparity map obtained by extraction of 10 000 points with a roulette wheel mechanism. This sample has been obtained after having removing the background small density values from the disparity map, and after augmenting the contrast in it. Then, objects that are closed to the camera have higher density values. In (c), is shown the adapted 2D mesh obtained by the GPU SOM algorithm applied to the modified disparity map. In Fig. 5(a), whiter regions are nearer to the camera view-point. In (b), such nearest regions present higher density of extracted points. In (c), the adapted grid presents higher density of neural network nodes on such regions, with respect of the topology of the scene. That is, proximity of grid points reflects proximity in Euclidean space.

A second illustration of the meshing process is given in Fig. 6(a-d) on the Teddy example. The image in (a) represents the left color stereo-image. A sampling of the disparity map, after removing background and augmenting contrast, is presented in (b), and the 2D adapted mesh in (c). The image in (d) presents the surface reconstruction in 3D space obtained by using the adapted mesh. Note that this mesh can be seen as a compressed representation of the 3D surface, such that objects closest to the camera have higher resolution and their details more finely represented.

In the second set of experiments dealing with larger size disparity images, we use the Cones image to perform the com-

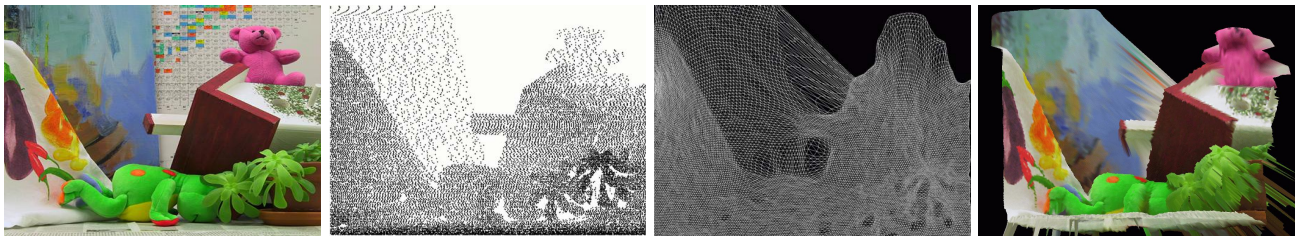


(a) (b) (c)

Fig. 5: Meshing of the 'Teddy' image: (a) Original input density map. (b) Density sampling. (c) Meshing for Tsukuba image.

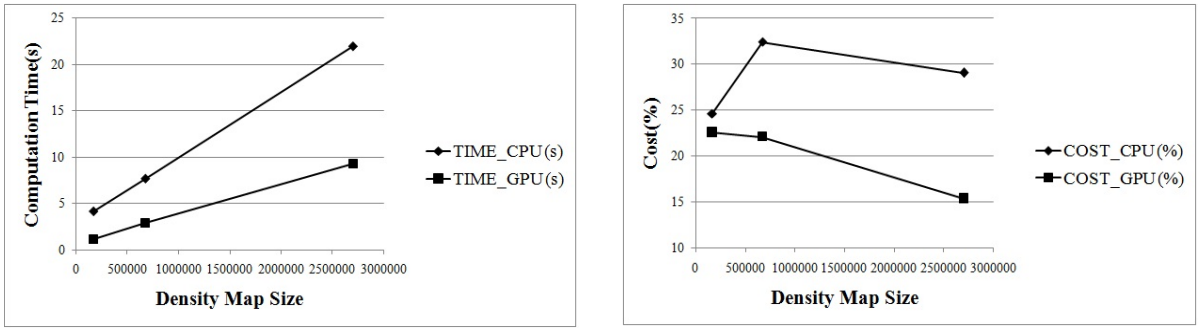
TABLE IV: MESHING EVALUATION RESULTS ON THE FOUR MIDDLEBURY DATA SETS OF SMALL SIZE.

		CPU		GPU	
Image	Size	Cost(%)	CT (s)	Cost(%)	CT (s)
Tsukuba	384×288	24.34	3.2	25.70	0.768
Venus	434×383	26.12	3.476	24.92	0.896
Teddy	450×375	25.08	4.1	23.14	1.2
Cones	450×375	24.66	4.16	22.6	1.18



(a) (b) (c) (d)

Fig. 6: Meshing of the 'Teddy' image: (a) Original color image. (b) Density sampling. (c) Meshing for Teddy image and (d) visualization on 3D space.



(a) (b)

Fig. 7: Comparative between CPU and GPU for the increasing size 'Cones' images: (a) Comparison of relationship between computation time and image size in pixels. (b) Comparison of relationship between cost and image size in pixels.

TABLE V: MESHING EVALUATION RESULTS ON LARGE SIZE 'CONES' DISPARITY IMAGES.

		CPU		GPU	
Image	Size	Cost(%)	CT (s)	Cost(%)	CT (s)
ConesT	450×375	24.66	4.16	22.6	1.18
ConesH	900×750	32.42	7.7	22.1	2.9
ConesF	1800×1500	29.06	22	15.38	9.3

parative evaluation. The numerical results are given in Table V and resumed in Fig. 7. Here again, the GPU parallel cellular model outperforms its CPU counterpart both in computation time and result quality. With the increase of image size, the performance in cost minimization of the GPU model augments substantially, whereas computation time slightly increases. On the CPU side, however, the performance highly degrades both in computation time and cost minimization. This indicates that it should be difficult to reach the same quality of results, except by drastically augmenting the CPU computation time.

V. Conclusion

We have presented a GPU cellular parallel model for the self-organizing map and its application to structured mesh generation of stereo-matching disparity maps. The structured mesh generation problem is presented as an NP-hard balanced clustering problem in the plane, and the SOM process is used as an heuristic to deal with it. The disparity map is partitioned into an appropriate number of cell units, that are each associated to a computation processor and responsible of a certain area of the plane. The advantage of the parallel algorithm is that it is decentralized and based on data decomposition, rather than based on data duplication, or mixed sequential/parallel computation, as often with GPU implementation of optimization metaheuristics. The processing units and the required memory are with linearly increasing relationship to the problem size (image size), which makes the model able to deal with very large scale problems in a massively parallel way. Future work should deal with verification of effectiveness as the number of cores augment, and with a more thorough examination of the memory management in order to achieve faster real-time computation on massive parallel platforms.

References

[1] T. Kohonen, "Self-Organization Maps and associative memory," *Spring Verlag*, 2001.

[2] ——, "Clustering, Taxonomy, and Topological Maps of Patterns," *Proceedings of the 6th International Conference on Pattern Recognition*, October 1982.

[3] G. F. Carey, "Computational Grids: Generation, Adaptation and Solution Strategies," *Taylor & Francis*, June 1997.

[4] P. George, "Génération automatique de maillages. ," *Applications aux méthodes d'éléments finis,Masson, RMA 16*, 1991.

[5] F. Tombari, S. Mattoccia, and L. D. Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," *Proc PSIVT*, pp. 427–438, 2007.

[6] J. P. Pons, F. Segonne, J. D. Boissonat, L. Rineau, M. Yvinec, and R. Keriven, "High-Quality Consistent Meshing of Multi-Label Datasets," *;Information Processing in Medical Imaging*, pp. 198–210, 2007.

[7] E. Bonomi, "Generation of structured adaptative grids based upon molecular dynamics," *Comptes rendus des journées d'électronique, Presses Polytechniques Romandes, Lausanne*, 1989.

[8] O. Sarzeaud, Y. Stephan, and C. Touzet, "Finite Element Meshing using Kohonen's Self-Organizing Maps," *International Conference on Artificial Neural Network*, juin 1991.

[9] R. Schneiders, "Algorithms for Quadrilateral and Hexahedral Mesh Generation," *Proceedings of the VKI Lecture series on Computational Fluid Dynamic*, pp. 2000–2004, 2000.

[10] Y. Zhang and C. Bajaj, "Adaptive and quality quadrilateral/hexahedral meshing from volumetric data," *Comput. Methods Appl. Mech. Eng.,*, pp. 942–960, 2006.

[11] L. Hongqiang, W. Yizhao, and C. Songcan, "A new method based on SOM network to generate coarse meshes for overlapping unstructured multigrid algorithm," *Applied Mathematics and Computation*, pp. 353–360, 2003.

[12] O. Nechaeva, "Using Self Organizing Maps for 3D surface and volume adaptive mesh generation," *http://www.intechopen.com/books/self-organizing-maps/using-self-organizing-maps-for-3d-surface-and-volume-adaptive-mesh-generation*, 2010.

[13] R. L. M. E. do Rego, A. F. R. Araujo, and F. B. d. L. Neto, "Growing Self-Organizing Maps for Surface Reconstruc tion from Unstructured Point Clouds ," *Int. Joint Conf. Neural Netw.*, pp. 1900–1905, 2007.

[14] A. C. Castilla and N. G. Blas, "Self–organizing map and Cellular automata combined technique for advanced mesh generation in urban and architectural design," *International Journal "Information Technologies and knowledge"*, vol. 2, 2008.

[15] J. L. Bentley, B. W. Weide, and A. C.-c. Yao, "Optimal expected-time algorithms for closest-point problems," *Computer Science Department*, p. 2451, 1979.

[16] S. A. N. Brad Hiebert-Treuer and D. Scharstei, "2006 Stereo Datasets," *http://vision.middlebury.edu/stereo/data/scenes2006/*, Nov 2006.