IL NUOVO CIMENTO Vol. 37 C, N. 4 Luglio-Agosto 2014
DOI 10.1393/ncc/i2014-11819-x

Communications: SIF Congress 2013

Efficient implementation of a 3-dimensional lattice-free tumor model

S. Stella(*)

Dipartimento di Fisica, Università di Trieste - Via Valerio 2, I-34127 Trieste, Italy

ricevuto il 31 Gennaio 2014; approvato il 19 Marzo 2014

Summary. — The biophysical complexity of tumors can be attacked with numerical models and with the power of modern computers. A numerical model of avascular solid tumors has recently been developed in Trieste and Verona (Milotti E. and Chignola R., PLoS One, **5** (2010) e13942), it includes the basic biology of tumors and it has been successfully validated with experimental data. The model could be further extended and exploited as a tool for understanding tumors and this will require the introduction of new biological details with a consequent increase of the computational effort. Therefore the upgrade of the model requires a prior redesign step. In this paper alternative programming solutions are proposed both for diffusion-reaction part of the code and for the geometrical-topological description of the cell aggregate. These changes will make the code more modular and efficient, and they will make it portable on highly parallel machines.

PACS `87.17.-d` – Cell processes.
PACS `87.17.Aa` – Modeling, computer simulation of cell processes.

1. – Introduction

In the last decades different mathematical models that aim to simulate tumor growth have been developed [1]. The management of the biological complexity requires the definition of clear initial assumptions. They represent important constraints that influence both the computational effort and the model reliability for a specific experiment. Usually such models integrate processes on different temporal and spatial scales, such as gene expression, molecular pathways, cell-cell interactions and cell mobility.

All these chemical and physical processes can be modeled on a spatial domain where the cells either are free to move in every direction or are forced to move following a lattice

(*) E-mail: `sabrina.stella@ts.infn.it`

structure. This topological choice strongly influences the model both in its mathematical formulation and resolution and in its potential to reproduce the biological reality.

In this paper I concentrate on a lattice-free numerical model of tumor spheroids [2-5] where cells are represented by soft spheres that interact with one another and with the environment by exchanging nutrients, exerting cell-cell forces and moving freely in 3-dimensional space. In this disordered network of cells chemicals diffuse and react, determining the cells' life and death, influencing the cell cycle and the cellular events. Already with a midsize tumor spheroid the problem can be extremely challenging: the diffusion-reaction part alone requires the simultaneous solution of a few million non-linear differential equations, which change from step to step as cells move and proliferate and proximity relations change.

The numerical model [2] was experimentally validated and actually represents a valid tool to explore the biology of the tumor microenvironment. To perform further simulated/experimental data comparison the model must be extended to adjust to a different experimental conditions, and this could require the introduction of new biological details such as new molecular circuits, with a resulting increase in computer time.

For this reason it was decided to refurbish the code to improve its flexibility and speed-up. In this work the two most important changes in the code are explained. In particular the first part of the paper is related to the description of a different set of classes design to facilitate the introduction of new molecular pathways, without additional overhead. Whereas the second part concerns the description of a new algorithm for the adjacent-cell search to be performed on Graphic Processing Units (GPU).

2. – Chemical-based class scheme

The simulation program includes a biochemical model of the diffusion-reaction processes for a selected number of molecules. The corresponding PDEs are numerically solved discretizing on the disordered network of cells and, for each time step, the cellular events and the proximity relations are also evaluated. This model was implemented in C++ language by using an abstract model that consider the cells as basic objects. A class `Cell` was created in which the concentration of the molecules, the cell's state and the list of adjacent cells are the main data. The `Cell` objects are stored in turn in STL vectors and the solution of the reaction-diffusion equations requires the access to the internal data stored in each cell in the vector, and possibly a temporary storage in an auxiliary vector. This solution is not effective because of the large number of memory access operations and it leads to a long execution times. This inefficiency gets worse with the introduction of new molecular species.

These considerations suggest the use a "reverse logic" in which the single chemical is the main computational object and the cells represent small compartments where the diffusion and reaction take place.

Therefore a new abstract model was designed to manage the collective diffusion of a set of molecules and facilitate the introduction of new substances in the biochemical model.

A test program was used to design the classes and to verify the potential of the new scheme. It consists of an on-lattice 2-dimensional cell aggregate, in which two different molecular species, A and B, whose concentration is $\rho^{(A)}$ and $\rho^{(B)}$, can diffuse and react

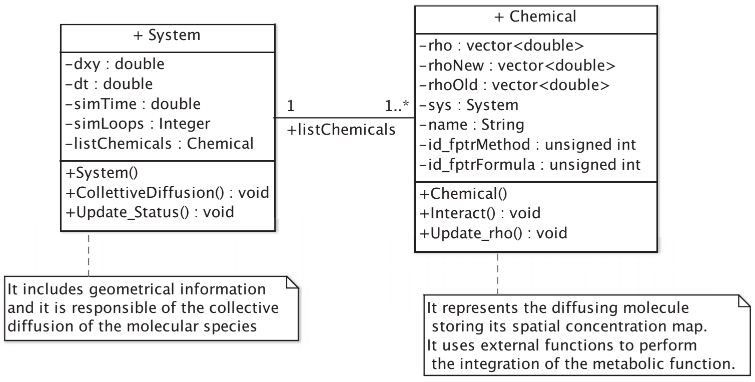
+ System
-dxy : double
-dt : double
-simTime : double
-simLoops : Integer
-listChemicals : Chemical
+System()
+CollettiveDiffusion() : void
+Update_Status() : void
1
1..*
+listChemicals
+ Chemical
-rho : vector<double>
-rhoNew : vector<double>
-rhoOld : vector<double>
-sys : System
-name : String
-id_fptrMethod : unsigned int
-id_fptrFormula : unsigned int
+Chemical()
+Interact() : void
+Update_rho() : void
It includes geometrical information and it is responsible of the collective diffusion of the molecular species
It represents the diffusing molecule storing its spatial concentration map. It uses external functions to perform the integration of the metabolic function.

Fig. 1. – UML class diagram representing the classes introduced to implement the simplified diffusion problem.

according to the following PDEs:

$$\frac{\partial \rho^{(A)}(x,y,t)}{\partial t} = D_A \nabla^2 \rho^{(A)}(x,y,t) - \frac{V^{(A)}_{max}\rho^{(A)}(x,y,t)}{K^{(A)} + \rho^{(A)}(x,y,t)}\,, \tag{1a}$$

$$\frac{\partial \rho^{(B)}(x,y,t)}{\partial t} = D_B \nabla^2 \rho^{(B)}(x,y,t) - \frac{V^{(B)}_{max}\rho^{(B)}(x,y,t)\rho^{(A)}(x,y,t)}{K^{(B)} + \rho^{(B)}(x,y,t)\rho^{(A)}(x,y,t)}\,, \tag{1b}$$

where D is the diffusion coefficient and the second term of the r.h.s comes for the Michaelis-Menten (MM) approach to enzyme-catalyzed reactions, where V_{max} is the maximum reaction rate and K is the MM constant. The MM approach is extensively used in biochemistry and it is particularly important for cellular biology since this is associated to a large number of cellular metabolic reactions.

The equations are solved numerically using a scheme similar to that used in the tumor growth model: diffusion is discretized on the cells' lattice and then the implicit Euler method is used to solve the resulting time-dependent differential system. This leads to system of coupled non-linear equations which are solved by using Newton-Raphson steps that are iterated until a given convergence criterion is achieved.

2·1. *Description of the new class scheme.* – The new abstract model, in its first implementation, consists mainly of a couple of classes named `Chemical` and `System` (see fig. 1). The first represents a single molecular species and integrates the corresponding diffusion-reaction equations for a single time step. The class `System` contains information about the environment, it manages the collective diffusion of the whole set of substances in the model, and it also deals with the printout of the results on appropriate output files.

The concentration map of a given chemical species is stored in the vector `Chemical.rho`: its size is not fixed *a priori* in order to be ready for the more general problem where cells proliferate and increase in number. Among the `Chemical`'s methods, there is `Chemical::Interact(double)` which integrates the proper reaction-diffusion term, which in this the toy is the r.h.s. of eqs. (1). Since different chemical species have different diffusion-reaction term, the function `Chemical::Interact(double)` cannot be univocally implemented, but its behavior needs to be differentiated according to the substance under consideration. For this purpose an appropriate *strategy pattern*, exploiting function pointers, was used [6]. Indeed, two integer variables were introduced in the class, id_fptrMethod and id_fptrFormula, which appear as constructor's arguments, therefore they needs to be initialized when a new Chemical object is created. These data represent the link to the correct functions to be called in order to perform the numerical calculation; in particular, the first value points to the numerical method to be used and the second variable addresses the correct diffusion-reaction term. These functions are declared and implemented in auxiliary files (for details see [7]).

Within a given cell aggregate different nutrients can diffuse simultaneously; the class System abstracts the concept of cell aggregate and its surrounding environment, it stores data associated to the cells' topology and to the set of substances. Indeed, an STL dynamic vector, ListChemical, is included in System and it is used to store a list of Chemical's pointers. The collective diffusion of the molecules can be performed by a single call to the function System::CollectiveDiffusion, hence an iterator is used to move over the ListChemical vector and to call serially the function Chemical::Interact, which in turn, addresses to the proper auxiliary functions. When the process ends the function System::Update_Status() is called and the map of concentration values rho for each chemical species is updated to the new time-step. The code below shows the function's body in its simplest version.

```
void System::CollectiveDiffusion(double eps){
    cout << "Set the simulation time (s): ";
    cin >> simTime;
    simLoops =  simTime/dt;

    while(n<simLoops){
       n++;
       for(Iter i = listChemicals.begin(); i!=listChemicals.end(); i++){
           (*i)->Interact(eps);
       }
       Update_Status();   //Update the vector rho.
    }
   return;
}
```

The CollectiveDiffusion function calculates the diffusion processes for a user-defined simulated time step.

2·2. *Toy model implementation and results.* – The toy model was implemented by using these two classes according to the following pseudocode:

```
    Define the cell lattice where the diffusion takes place:
System sys(100, 0.0005, 10); //Lattice size and lattice spacing, time step

    Define the biochemical system
Chemical A(name, sys, id_method, id_function)
Chemical B(name, sys, id_method, id_function)

    Add molecules to System:
sys.Add_ListChemicals(A);
sys.Add_ListChemicals(B);

    Start the diffusion process:
sys.CollectiveDiffusion(0.01);
```

When an object System is created its constructor automatically generates a regular square lattice of cells, according to the parameters *lattice size* and *lattice spacing* set in the constructor. In this case a lattice of 100×100 cells whose diameter is set to $5\,\mu$m, is considered. The initial concentrations for both A and B molecular species, were set to $1\,\text{g/cm}^3$ (conventional value) for the boundary cells and to a null value for the inner cells. The initial concentrations as well as the topology of the cell aggregate can be differentiated by implementing a new `System`'s constructor. With this class scheme any extension of the biochemical model is automatically managed by the class `System` once that the proper diffusion-reaction term is added in the auxiliary file.

Several test runs where performed with realistic model parameters. During these simulations the concentration map changes as expected: the nutrients move gradually from the border to the inside of the lattice, gradually increasing the concentration values. The consumption term corresponds to a negative contribution and sets a limit to the maximum concentration values.

A further implementation of the toy model was performed by using the cell-based class scheme with the aim to compare the two approaches in term of code flexibility, safety (incapsulation) and speed-up.

The comparison between the two programs shows that using the new class scheme improvements in terms of flexibility and code safety can be achieved without loss in performance. For more detail see [7].

2˙2.1. The code optimization. The numerical calculation of a diffusion-reaction equation can be efficiently parallelized by using GPU [8]. A GPU is a many-core processor with its own device hierarchical memory, initially developed for graphic purposes [9]. The Compute Unified Device Architecture (Nvidia - CUDA) [10] is a general purpose software architecture which provides, by using extensions of the C/C++ or Fortran language - a simple model of GPU programming for applications that range far beyond simple graphic processing. By using CUDA code the data-processing can be split into independently executed *threads*, grouped into *blocks*; the number of threads is hardware limited but their management has a low cost. Once the data to be processed are passed from CPU to GPU each thread can execute the same user-defined function, called *kernel*, but on different data.

The classes' structure was partially modified in order to compute the substances' diffusion processes on a NVidia GPU.

A preliminary implementation of the numerical solution for GPU using CUDA was performed and tested on a Quadro 4000 NVidia GPU. The code was tested using different lattice sizes for a simulation time equal to 3600 s. Similar problems were run on a quad core MacBook Pro using the compiler GCC 4.7 and the comparison of the execution times for CPU and GPU was performed, the results are shown in table I. The test shows that using a 1000×1000 set of cells a speed-up about 5 was obtained. An additional speed-up can be achieved by optimizing the use of GPU's memory levels.

3. – Proximity relations among cells

In a lattice-free mathematical model of tumor growth the cells can move in every direction and change their distance under the pushes and pulls of cell-cell forces, especially after a mitosis event. Figure 2 shows two snapshots of the same cell aggregate after 100 and 110 hours of simulated time, respectively. The snapshot on the left has 29 cells and the one on the right has 32 cells: the migration of cells and the consequent

Table I. – *Execution times* (s) *of the diffusion-reaction process implemented on GPU and CPU with or without optimization flag for different lattice size and simulation time* = 3600 s.

Lattice size	GPU	CPU (no opt)	Speed-up	CPU (-O3)	Speed-up
100×100	0.73	1.2	1.6	0.46	< 1
300×300	3.04	20.8	6.8	7.8	2.6
600×600	13.53	144.8	10.6	55.2	4.8
1000×1000	46.02	624.8	13.5	239.5	5.2

changes of proximity relations are clearly visible. The implementation of a proper nearest neighbor search algorithm that continuously updates the proximity relations among cells is therefore mandatory. This topological information is essential not only to perform the numerical calculations of the diffusion-reaction equations but also to compute the cell mechanics formulation.

For a given time step the number of adjacent cells is not fixed, it varies from cell to cell and depends upon the approximations introduced in the model to simulate cells and their interactions. For the model under study the cells are represented by soft spheres and they can experience elongation or contraction in order to consider the viscoelastic nature of the cells. These cell properties are modeled by using selected values of *Young's modulus* and *Poisson ratio* that parameterize respectively the elastic and contractile property of the cell. The cell-cell interaction is modeled by a formula that includes, not only these parameters, but also the cell adhesion force, which in turn comes from the number of cross links between cells, *i.e.* the adhesion molecules located on the cell membrane; the corresponding formula is reported and explained in detail in [2]. The direction and modulus of this interaction depends upon the distance between the cells' centers; it is repulsive for distances lower than the sum of the cells' radii, and attractive otherwise (see fig. 3).

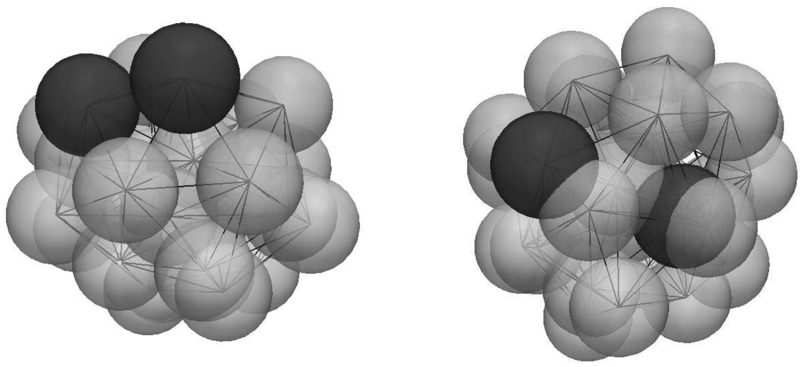

Fig. 2. – The same cell aggregate after 100 hours (left panel) and 110 hours (right panel) of simulated time. The snapshot on the left has 29 cells and the one on the right has 32 cells. Two cells are marked in dark gray to display their movement during the 10 hours of simulated time between the two snapshots: the migration of cells and the consequent changes of proximity relations are clearly visible.

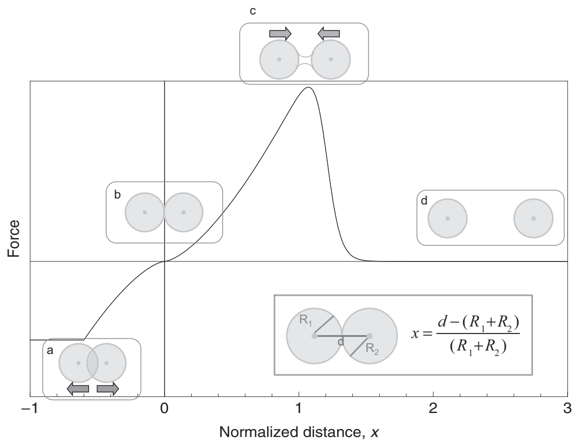

Fig. 3. – The figure illustrates the qualitative behavior of the attractive-repulsive force between cells (discussed in detail in [2]). When the distance d between cells is lower than the sum of the radii R_1 and R_2, as after a mitosis event, the cell-cell force is repulsive (inset a), so the two cells gradually separate until the force vanishes (inset b); when the distance increases the force becomes attractive due to the membrane cross-links (inset c). At larger separation the cross links break an the attractive force vanishes (inset d).

Two cells are in contact if the following condition in satisfied:

$$d < (R_1 + R_2)\ \texttt{ext\_coeff} \tag{2}$$

where R_1 and R_2 are the radii of the cell 1 and 2, respectively, and the extension coefficient `ext_coeff` is a parameter (greater than 1) that takes into account the softness of the cells. This last parameter is already part of the model and it is used to calculate the contact surface between cells and the corresponding extracellular volume.

The adjacent-cells search algorithm should also identify the boundary cells, which have a lower number of neighbors. This information is important for the diffusion computation because the border cells represent the interface with the nutrient supply.

In the present version of the simulation program these geometrical problems are partially solved by using a C++ library, CGAL [11], that includes algorithms and classes for the solution of several computational geometry problems. In particular the Delaunay triangulation is used to find the proximity relations among cells whereas the alpha-shape algorithm defines the cell's aggregate boundary. In the next future we plan to replace this library with a parallel code running on an NVidia GPU. This change will involve the implementation of a new algorithm to search the adjacent cells and to identify the boundary. The new code is expected to be much better in terms of reliability and speed.

3˙1. *Adjacent-cell search algorithm on GPU*. – The aim of the algorithm is to provide for each cell a list of adjacent cells and the corresponding cell-cell distance to compute the numerical solution of the diffusion-equation, the cell-cell force and the cell dynamics. This is a typical *intense data-parallel computations* problem in which the same flux of instructions can be executed for different data independently, therefore it is a problem suitable to be run on Graphic Processing Units (GPU).

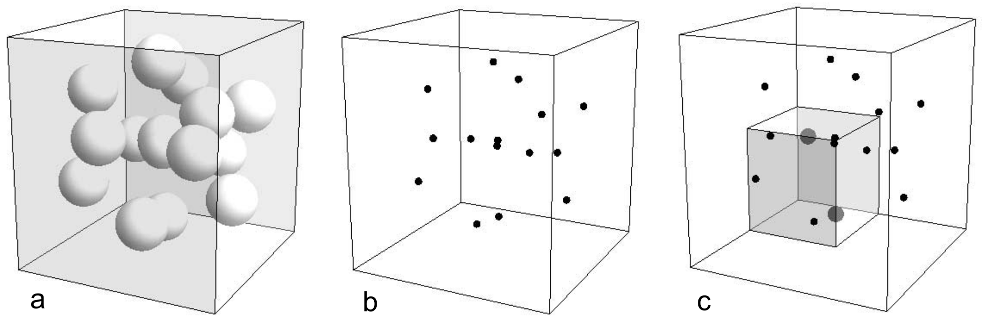

Fig. 4. – The figure illustrates the space partitioning operation. At first the extended bounding box enclosing the cells is found (a); the cells are identified by the coordinates of their centers (b) and the bounding box is successively subdivided into s^3 small boxes as the one shaded in panel c, where the bigger dots represent the cells enclosed in the shaded box.

The proposed adjacent-cell search algorithm begins with the partition of the computational space into different regions and with the assignment to each region of the list of cells that are enclosed into them. There are several different ways, and corresponding data structures, to perform the space-partitioning operation from the simplest, which is a regular cubic grid, to the most refined space partitionings that provide a further cell-grid-specific levels of subdivision, such as the k-tree and R-tree data-structures.

For a first implementation a regular cubic grid was used. Since the data set is dynamic — the number of cells grows during simulation — the computational volume cannot be fixed *a priori* but must be recalculated at each time step. This is performed by finding the *extended bounding box*, that is the smallest region containing the cells, eventually extended to have a cubic shape. The extended bounding box is further subdivided into s^3 smaller boxes, where s is the number of subdivisions of each side (see fig. 4).

A struct `point4d` was created containing the radius and the center coordinates (x, y, z) of a single cell, therefore a vector of this struct is used to store all the cells. Successively for each cell a *hash value* is assigned according to the following formula:

$$hash(\mathtt{p}) = \mathrm{lc}_{\mathrm{p.x}} \times s^2 + \mathrm{lc}_{\mathrm{p.y}} \times s + \mathrm{lc}_{\mathrm{p.z}} \tag{3}$$

where `p` is a `point4d` data type and `(p.x, p.y, p.z)` is the position of the single cell's center in the 3-dimensional space. The $(\mathrm{lc}_{\mathrm{p.x}}, \mathrm{lc}_{\mathrm{p.y}}, \mathrm{lc}_{\mathrm{p.z}})$ are the labels (integer indexes) that identify the small box enclosing that cell.

With this space partitioning the search of the adjacent cells becomes less complex, since the cell-cell distance calculation is performed on a small subset of cells, that is those having the same or adjacent hash value. The data, *i.e.* the position and radius of the cells, can be sorted in the memory buffer by increasing hash value and allowing a coalescing data access. This provides a further optimization of the search operation, since the required data are located in aligned memory positions, reducing the number of memory read operations.

3˙2. *Concurrent selection of adjacent cells*. – Mapping the cells on the regular grid by using a hash value is the first step of the algorithm. The prototype code was implemented to run on GPU by using different kernel functions according to a scheme similar to those suggested in [12]. The second step consists in the calculation of the list of adjacent cells: this operation can be performed independently for each cell, therefore it is assigned to a single thread.

To determine the list of neighbors of a given cell `p` and hash value α, the distance with the cells having the same hash value and with hash values corresponding to the 26 neighboring boxes needs to be performed. The number of boxes to be scanned depends on the space partitioning operation, and indeed the value s, the number of subdivision of a single side of the bounding box, sets the average box count, that is the number of cells that are enclosed in a box. By geometrical considerations, and by the previous simulation, the number of neighbors for a compact arrangement of cells in a 3-dimensional space is expected to be about 12. In the test program the value for s was chosen to yield an average box count of about 6. Therefore each thread will scan over 27 boxes to find about 12 adjacent cells. In case of threads acting on the boundary of the bounding box, the proper number of neighboring boxes is determined from time to time.

Once the scan volume is determined, before computing the cell-cell distance a further scan is performed over the volume, to check the possible empty boxes and eventually the hash value of the non-empty cells is recorded in a local vector `CL`; this operation becomes convenient for cells belonging to the boundary region. Then, for each cell enclosed in this new volume, the adjacency condition is checked 2 and, if verified, the cell's address is stored in a local vector `NLN`; at the same time the counts of the total number of adjacent cells is updated. When the loop over the hash index ends, the final list of adjacent cells is copied on a common vector `NL` so that the list of the adjacent cells of cell `j` is stored between `NL[j*14]` and `NL[(j+1)*14-1]` where 14 was set as the maximum number of the adjacent cells. This part of the algorithm can be summarized by the following pseudocode:

```
1: for all $p_2 \in$ CL
2:    $d = |\mathbf{p}_2 - \mathbf{p}_0|$
3:    $R_{ext} = (p_2.r + p_0.r)$ext_coeff
4:       if $d \le R_{ext}$
5:       append $p_2$'s address to NLN vector
6:       update the total number of neighbors
7:      end if
8: end for
9: Save the list of adjacent cell in NL
```
where $\mathbf{p}$ = `(p.x,p.y,p.z)` *is the position of the cell's center;* `p.r` *is the cell's radius*

Finally, a comparison between a total cell's surface and its contact surface with neighboring cells is used to determine whether it is in direct contact with the environment.

3˙2.1. Test. Preliminary tests of the algorithm were performed starting with relatively small-sized spheroids. As expected the efficiency of the algorithm increases with the spheroid size. The corresponding execution times are shown in table II

Table II. – *Execution time for the adjacent-cell search kernel for different spheroids size.*

Tot Particle	s (boxes)	Box's size	Exec time (s)
984	6 (216)	4.5	0.5
2759	8 (512)	5.3	0.68
3119	8 (512)	6.1	0.65
4900	9 (729)	4.6	0.67

4. – Conclusion

Although the structural changes described in sect. **2** have already been tested, they still await a practical implementation in the simulation program. The reason is that, at the moment, the new class structure does not include the geometrical-topological algorithm described in sect. **3** which is optimized for GPU's but works just as well with CPU's. In the next future the preliminary study of the algorithm described in sect. **3** will be completed and it will be implemented in the simulation program. This will allow the inclusion of additional substances in the simulator and correct some minor bugs that are carried over into the simulation program by the alpha-shape algorithm. The resulting improved program shall open up new opportunities for understanding the biophysics of solid tumors.

* * *

I wish to thank Prof. Edoardo Milotti and Dr. Roberto Chignola for useful discussions.

REFERENCES

[1] Rejniak K. A. and Anderson A. R.,, *WIREs Systems Biol. Med.*, **3** (2011) 115.
[2] Milotti E. and Chignola R., *PLoS One*, **5** (2010) e13942.
[3] Chignola R., Del Fabbro A., Farina M. and Milotti E.,, *J. Bioinform. Comput. Biol.*, **9** (2011) 559.
[4] Chignola R. and Milotti E., *AIP Adv.*, **2** (2012) 011204.
[5] Milotti E., Vyshemirsky V., Sega M., Stella S., Dogo F. and Chignola R., *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **10** (2013) 805.
[6] Meyers S., *Effective C++: 55 specific ways to improve your programs and designs* (Pearson Education) 2005.
[7] Stella S., Chignola R. and Milotti E., *Comput. Phys. Commun.*, **185** (2013) 826.
[8] Molnár Jr F., Izsák F., Mészáros R. and Lagzi I.,, *Chemom. Intell. Lab. Syst.*, **108** (2011) 76.
[9] Kirk D. B. and Wen-mei W. H., *Programming massively parallel processors: a hands-on approach* (Morgan Kaufmann) 2010.
[10] http://docs.nvidia.com/cuda/cuda-c-programming-guide/.
[11] http://www.cgal.org.
[12] Green S., *Particle Simulation Using CUDA*, http://docs.nvidia.com/cuda/samples/5_simulations/particles/doc/particles.pdf.