The Visual Computer manuscript No.
(will be inserted by the editor)

A GPU framework for parallel segmentation of volumetric images using discrete deformable models

Jérôme Schmid[1] · **José A. Iglesias Guitián**[2] · **Enrico Gobbetti**[2] · **Nadia Magnenat-Thalmann**[1]

Received: date / Accepted: date

Abstract Despite the ability of current GPU processors to treat heavy parallel computation tasks, its use for solving medical image segmentation problems is still not fully exploited and remains challenging. A lot of difficulties may arise related to, for example, the different image modalities, noise and artifacts of source images, or the shape and appearance variability of the structures to segment. Motivated by practical problems of image segmentation in the medical field, we present in this paper a GPU framework based on explicit discrete deformable models, implemented over the NVidia CUDA architecture, aimed for the segmentation of volumetric images. The framework supports the segmentation in parallel of different volumetric structures as well as interaction during the segmentation process and real-time visualization of the intermediate results. Promising results in terms of accuracy and speed on a real segmentation experiment have demonstrated the usability of the system.

Keywords Simulation and Modeling, GPU Programming, Segmentation

1 Introduction

Medical image segmentation is nowadays at the core of medical image analysis and supports e.g. computer-aided diagnosis, surgical planning, intra-operative guidance or post-operative assessment. Segmentation is also present in computer vision applications such as tracking and recognition. Furthermore, segmentation attracts the interest of the Computer Graphics community, by supporting e.g. visualization of medical datasets. Although research has been very active these last decades, segmentation is still a very challenging problem. The large variety of image modalities with associated artifacts, the variability of the structures to segment and the strong demanded requirements (e.g., high accuracy, automation) seriously hinder the design of efficient segmentation methods. In this context, the use of interactive and fast segmentation approaches can expedite tedious parameter tuning and reduce the limitations of segmentation methods since interactive control is available [22].

The rapid development of Graphics Processing Units (GPU) was followed by the porting and adaptation of segmentation approaches to the GPU architectures. These methods strongly contributed to the fostering of fast interactive segmentation. Initially, GPU approaches stemmed from the idea to speed-up time consuming CPU segmentation approaches and to provide an interactive visualization of the segmentation evolution. GPU programming became then easier and more efficient yielding the implementation of more advanced segmentation approaches.

We present in this paper a GPU framework, implemented using the NVidia's Compute Unified Device Architecture (CUDA) [21], aimed for the segmentation of *volumetric* images based on *discrete physically-based deformable models*. The framework exploits parallelism and performs completely in the GPU being capable of managing real-time interactive segmentation of multiple structures. To our best knowledge, such a GPU-based 3D segmentation framework based on discrete deformable models has not been reported in the previous literature.

The paper is structured as follows. Related work on GPU-based segmentation is reviewed in Sec. 2, while Sec. 3 describes our segmentation algorithm. In Sec. 4 we describe the most important features of our GPU framework implementation. We present our results in Sec. 5 which are analyzed and discussed in Sec. 6. Section 7 concludes this paper with possible future improvements of our GPU-based implementation.

[1]MIRALab, University of Geneva, Battelle, 1227 Carouge, Switzerland
[2]CRS4 Visual Computing Group, POLARIS Ed. 1, 09010 Pula, Italy

2 Related work on GPU-based segmentation

In general, segmentation approaches can be classified as low- and high-level approaches. Low-level approaches directly work on voxel information and are usually burdened by the intensive manipulation of volumetric image data. As a result, these methods naturally appeared at first as good candidates for GPU implementations. Indeed, images are regular lattices on which read access can be very efficient due to the optimized GPU texture management (e.g., caching). We can find in the literature GPU-based implementations of the watershed [34] and region growing methods [32], along with Markov random fields [43] and graph cuts approaches [24, 17]. Image registration also turned out to strongly benefit from parallel computing, giving birth to a variety of implementations based on mutual information [37], sum of squared differences [14], demons [31, 20], viscous-fluid regularization [23] or regularized gradient flow [35].

Higher-level approaches, such as deformable models, were also ported to GPU architectures by considering implicit deformable models as image lattices (e.g., a 2D curve is implicitly represented as the iso-value of a field encoded as a 2D image). Level-sets approaches [16, 2] became particularly popular in the GPU-segmentation community as significant speed-ups and interactive rendering were made available. Geodesic active contours [1], which are a combination of traditional active contours (snakes) [12] and level-sets evolution, were efficiently implemented in GPU [40, 26] by using the total variation formulation, mostly to quickly segment structures by foreground and background separation in 2D images.

Nevertheless, little work has been made in implementing explicit discrete deformable models in GPU for segmentation purposes. That is unfortunate since discrete deformable models offer several advantages. Indeed, they provide an intuitive and more appropriate control of the shape deformations compared to implicit models. Furthermore, they are much more robust against image artifacts than most low-level approaches thanks to the use of shape regularization. In GPU, methods for implementing active contours based on gradient flow have been proposed [13, 10], but they were limited to the case of 2D images. On the other hand, many works exploited physically-based surface or volumetric deformable models in GPU in other application domains, such as spring mass systems [19, 6], cloth simulation [25], volumetric mesh deformation [36] or Finite Element Modeling [9].

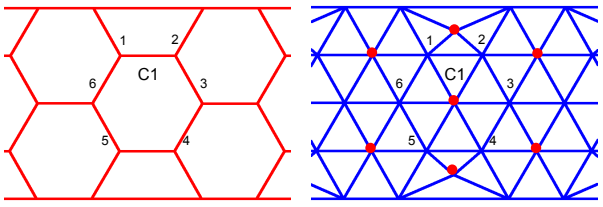
In this paper, we propose hence a flexible GPU framework for fully interactive parallel segmentation of multiple volumetric objects based on discrete deformable models. We demonstrate the interactivity of our framework implementing several image-based and shape preserving forces complemented with local control-point forces which demonstrate the interaction capabilities of the system.

3 Segmentation algorithm

Our segmentation approach is based on our previous work on dynamic deformable models [27, 7]. The general principle is to consider mesh vertices as a set of lumped mass particles with a state (position and velocity) subjected to internal and external forces. The concepts are hence similar to any deformable model-based simulation with the specificity here that images are used to drive model deformation for segmentation purpose. As a result, we might hereon refer the segmentation to as the simulation and vice versa.

3.1 Mesh representation

A mesh j is composed of M_j vertices and represented as a 2-simplex mesh [5]. A 2-simplex mesh is characterized by the property that a vertex has exactly three neighbors (See Fig. 1 for an example of 2-simplex mesh with a possible triangular tessellation). This representation is popular in image segmentation as local descriptors can be easily computed, such as the curvature. Furthermore, three local parameters uniquely define vertex positions from their three neighbors. These three parameters are independent, invariant under similarity transform and are denoted as simplex parameters. We also define for any point an elevation which is the signed distance between the point and its projection on the triangle formed by the 3 neighbors.



(a) 2-Simplex mesh (b) tessellation 2-Simplex mesh

Fig. 1 (a) 2-Simplex mesh representation: this example shows a 2-simplex mesh where the cell (i.e., a face) $C1$ is composed of 6 points with indices $\{1,2,3,4,5,6\}$; (b) depicts a possible triangular tessellation of the mesh which is built by connecting the points to the center of the corresponding cells (red discs).

3.2 Internal and external forces

Internal forces are necessary to regulate the segmentation while external forces effectively drive it towards the correct result, as detailed in the following.

3.2.1 Internal smoothing and shape prior forces

Internal forces ensure that the model evolution is perturbed as little as possible by image artifacts or possible numerical instabilities. Assumptions are thus made on the model smoothness and shape. A smoothness force is expressed by a weighted Laplacian smoothing coupled with an additional term accounting for the average local elevation of neighboring vertices. For each point, the Laplacian smoothing attracts vertices towards the barycenter of its neighbors. In case of weighted barycenter, weights are proportional to the area "covered" by a vertex, which is defined as the sum of the surfaces of all triangles sharing this point according to the presented tessellation (Fig. 1(b)). The local elevation term minimizes the shrinking effect recurrent with Laplacian smoothing.

Shape constraints are enforced by creating a force which aims to move the vertices to have a local description identical to a predefined one. This is simply done by using the simplex mesh parameters of a reference "average" shape [5]. This reference shape is constructed once with interactive approaches [28] that yield models with appropriate characteristics, such as smoothness and quasi-regularity of mesh faces.

The manual creation of the reference shape may be seen as a drawback with respect to methods using a small "seed" primitive (e.g., tethraedron in [33]) which is e.g. placed in the structure interior and is progressively inflated until the structure boundaries are reached. These approaches avoid indeed the creation of the reference model. However, as we will see in experimental Sec. 5, shape priors expressed by the simplex parameters of the reference model are essential to regulate the segmentation in presence of image artifacts. In fact, these artifacts mislead the evolution of approaches which ignore shape priors and are exclusively based on smoothness constraints (e.g., active contours [12], geodesic active contours [1], discrete deformable models [33]). Furthermore, it is important to understand that simplex parameters are invariant under similarity transform and hence some flexibility is given to the shape deformations. This implies that the structure to segment does not necessarily need to be very similar to the reference mesh. Figure 2 illustrates the use of these internal forces to denoise two bone shapes and recover their original aspect.

3.2.2 External image forces

Image forces are based on the minimization of image-based energies. Along the normal $\mathbf{n}$ of each mesh point $\mathbf{x}$, an image energy E is computed at various positions $\{\mathbf{y_1},\ldots,\mathbf{y_m}\}$ regularly sampled. A force is then built to attract the vertex towards the optimal target position $\mathbf{y}^*$ with the lowest image energy. We use two types of image energies. A first energy aims at aligning the gradient $\nabla I(\mathbf{x})$ with the normal $\mathbf{n}$: $E^g(\mathbf{x}) = \varepsilon \nabla I(\mathbf{x}).\mathbf{n}$, where $\varepsilon = 1$ when the expected gradient direction should be in the opposite direction of the normal and $\varepsilon = -1$ otherwise.

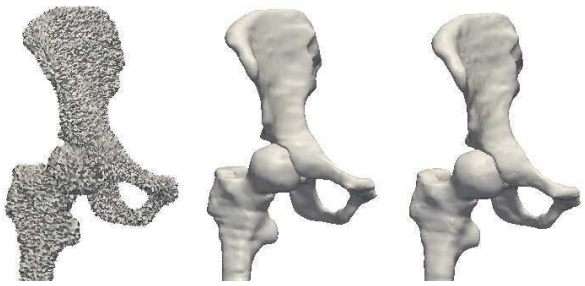


Fig. 2 Shape denoising example. From left to right: perturbed models, result with shape prior, and with smoothing.

Another energy E^{ip} can be computed by intensity profile (IP) similarity maximization [5, 7]. An IP is a vector of intensity values collected in a neighborhood swept along the normal direction during the search. At each neighborhood position, the similarity between the current neighborhood and a predefined reference IP is computed. The position which returns the highest similarity is used as the target position $\mathbf{y}^*$. The computation of the reference IP is computed once from a training image whose characteristics are alike but not necessarily identical to those of the image to be segmented (i.e., similar imaging protocol). In fact, we used the Normalized Cross Correlation (NCC) [11] as the similarity measure between IPs. NCC is quite robust to linear image intensity variations, which confers flexibility in choosing the imaging protocol. In conclusion, this IP-based force exploits the appearance of the structure of interest and can thus assist the segmentation of structures with inhomogeneous intensities as demonstrated in the experimental Sec. 5.

3.3 Deformable-model evolution

The evolution of the model is based on the resolution of a discrete differential equations system, which is the result of the Newtonian law of motion applied to the particle system. Given the forces and the particle state, the numerical integration yields a new state of the particle. Various approaches are available for integration (e.g, Explicit/Implicit Euler) depending on stability, accuracy and technical implementation constraints. Choices relative to our implementation of the numerical integration in GPU will be discussed in Sec. 4.3.

4 GPU Framework architecture

Our GPU segmentation framework is implemented on top of NVidia CUDA. We suggest interested readers non-familiar with CUDA architecture to refer to the corresponding programming guide [21]. The framework delivers interactive performance, real-time rendering, while being flexible enough to support new forces or even new segmentation paradigms. The framework architecture is divided according to the major stages involved in a simulation step (see Fig. 3). In the following, each stage is described along with the designed data access layer for storage and retrieval purposes of all the basic simplex-mesh data.

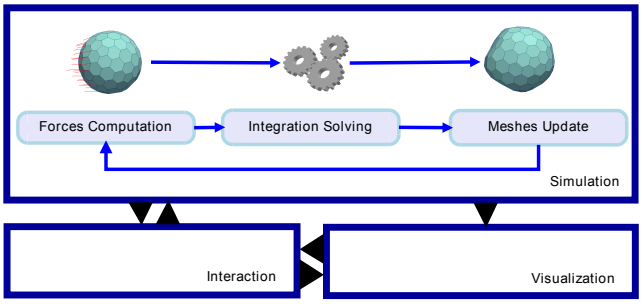


Fig. 3 GPU segmentation: In a simulation step, forces are computed for each particle (Sec. 4.2), whose state is updated in the numerical integration (Sec. 4.3), before updating the parameters of the meshes (Sec. 4.1). In parallel and asynchronously, meshes and image data are rendered (Sec. 4.4) while user can interact with the segmentation (Sec. 4.5).

4.1 Simplex mesh data access layer

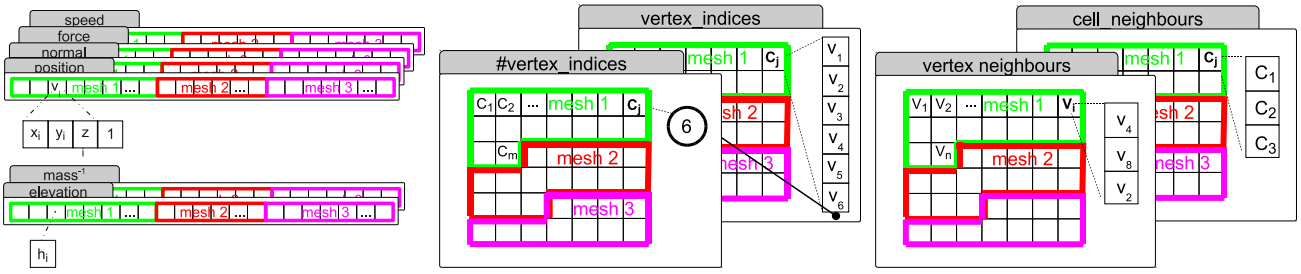
We have designed a simple data access layer for storage and retrieval of the simplex mesh data (see Fig. 4). In our GPU segmentation framework, meshes are encoded as vertices within one unique array. This encoding strategy has the advantage of being very compact in terms of space requirements and supports a straightforward distribution of the data for parallel processing within the GPU. We store position and normal for each vertex using a CUDA one-dimensional texture sampler and bound to linear memory as we need to use this attributes in both, read and write modes. The same strategy is used for other local parameters like the mass or elevation of a particle with respect to its neighbors. Besides, we store other *(mostly) invariant* per-vertex attributes using 2D samplers mapped to `cudaArrays`, because in that way (i) bigger amounts of memory in the GPU can be allocated and (ii) fast cached reads are available. In order to access the desired data we just need to compute the offset in each dimension of a 2D texture. Examples of per vertex attributes stored using 2D textures are the neighbor indices and the indices of the 3 neighbor cells sharing the same vertex. Finally, we store some per-cell information like the indices of the vertices of each cell and the number of vertices forming a cell, which is not necessarily the same for all the cells.

The volumetric image information is stored in a raw uncompressed format by using 3D textures. This is to take advantage of spatial locality to provide hence a fast access for read operations, which are the most intensive in terms of access time. Moreover, 3D textures provide a cheap trilinear interpolation, which is intensively used by image-related operations. In this first implementation of the framework, moderate-size volume data sets are handled, so no special compression or multi-resolution strategies are applied but they should be easily integrated. We refer interested readers to recent publications [8] and [4] which tackled the problem of massive volume management by organizing the volume dataset into a hierarchical octree based data structure. After each numerical integration, all meshes are processed in parallel at once by calling a CUDA kernel responsible of the update for each point, normal, area and elevation. Cell centers are also updated if necessary since some forces (e.g., smoothing and shape prior forces) or the visualization may need them. Writing operations are then performed using data structures in global memory space, as `cudaArrays` and texture memory writing operations are not yet supported by current versions of CUDA.

4.2 Forces computation

Once the information of all meshes has been updated, the computation of the available set of forces is ready to start. The computation of the forces in our framework is distributed in parallel assigning each particle to a GPU thread. Forces can be activated and disactivated for each mesh registered in the framework. For this purpose we maintain in the GPU a simplex-mesh activation register storing a collection of values α_i^f, being $0 \leq \alpha_i^f \leq 1$ the contribution of the force f to the i-th mesh. Our framework supports changes on this state at any moment during the runtime of the segmentation algorithm. The different forces are computed in a sequential order to avoid asynchronous updates of the resultant force. This behavior is directly related to the availability of atomic operations, only supported by NVIDIA graphics cards with CUDA compute capability above `1.0`. In any case, forces computation time is quite heterogeneous and so variable. In particular when one force is much more expensive than the others, we can assume that the total time of computation in parallel for all the forces is almost equivalent to the time of the most expensive force in terms of computation time (See forces computation percentages in Sec. 6).

Most of the forces were implemented without any special GPU optimizations as the chosen data access layer (e.g.,



(a) per vertex GPU arrays (b) per cell GPU arrays (c) vertex/cell neighbours GPU arrays

Fig. 4 GPU arrays. (a) Per vertex gpu arrays which are shared by all meshes in the system; (b) Cell description held in GPU: e.g. cell C_j is composed of vertices $\{V_1,V_2,V_3,V_4,V_5,V_6\}$ and the number of vertices (6) is stored in a separated GPU array; (c) Neighbor information per vertex on the GPU: e.g. vertex V_i has 3 neighbors $\{V_4,V_8,V_2\}$. Similarly, cell C_j has neighbor cells $\{C1,C2,C3\}$.

texture samplers) and parallelism strategies (e.g. simultaneous processing of all vertices with one kernel) were already carefully chosen. However, 'interactive' forces required more attention and are detailed in Sec. 4.5.

4.3 Numerical integration

Once the resultant force per particle is computed, we need to update the new particles state by solving a set of discrete differential equations. An appropriate integration technique is selected to reach a compromise between two conflicting criteria:

- Simulation criterion: since force evaluations are time-consuming, the integration technique should minimize extra simulation timesteps, requiring the minimum possible force evaluations per step.
- Interaction criterion: since the selected solution will be used within an interactive framework, the time we have for performing the simulation steps as well as for other GPU computations is severely limited.

While implicit based techniques are generally more stable and permit bigger stepsizes, the complexity of the technique in terms of computation time and programming complexity is quite high. The solution generally requires to invert a considerable size matrix, which dimensions depend directly on the number of forces and on the total number of particles involved in the system. Although some approaches, e.g. based on the conjugate gradient, avoid the effective inversion of the system, they rely on iterative procedures requiring a considerable effort to be successfully implemented in the GPU.

Then, since solving the integration in the CPU with an explicit method was shown to require a small stepsize in order to converge, we chose to move the calculations to the GPU in order to decrease the stepsize and reach better interactive frame rates. Among the explicit techniques implemented in the GPU, we have obtained the best results with a Verlet-based approach [42]. Our method is derived by writing two Taylor expansions of the position vector $\mathbf{x}(t)$ in different time directions. Adding these two expansions we obtain

$$\mathbf{x}(t+\Delta t) = 2\mathbf{x}(t) - \mathbf{x}(t-\Delta t) + \mathbf{a}(t)\Delta t^2 + \mathscr{O}(\Delta t^4)$$

where substituting $\mathbf{v}(t) = \frac{\mathbf{x}(t)-\mathbf{x}(t-\Delta t)}{\Delta t}$ gives us a position vector equation depending just on the previous position, the velocity and the timestep

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \mathbf{a}(t)\Delta t^2$$

Finally, the Verlet equations can also be modified to create a very simple damping effect, consisting on a value γ, being $0 \leq \gamma \leq 1$, and representing the fraction of the velocity per update that is lost to friction. The final resulting equation used in our framework is:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + (1-\gamma)\mathbf{v}(t)\Delta t + \mathbf{a}(t)\Delta t^2 \quad (1)$$

This equation is implemented in a CUDA kernel and is computed in parallel for each particle of the simplex meshes. The final result is written in the linear memory position array and then rebound to the texture sampler.

4.4 Visualization

To ensure real-time interaction, we established a balance between simulation and visualization tasks by fixing the maximum amount of time that the simulation can spend. As long as we are within the bounds of this time interval, one or more segmentation iterations are performed. Straight afterwards the simplex meshes are rendered employing the Vertex Buffer Object (VBO) OpenGL extension, in order to minimize the data transfers between GPU and CPU.

The sequence of operations needed for updating and displaying the simplex-mesh data in the VBO are shown in Fig. 5. First, all the VBO attributes array are mapped in the

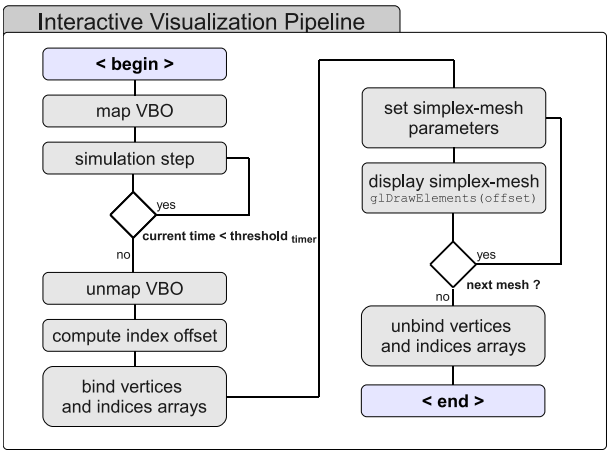


Fig. 5 Interactive Visualization Pipeline. In order to synchronize interaction and visualization while performing an interactive segmentation, we employ a timer to decide how many simulation steps we would like to perform each new frame. For the rendering itself, we employ the OpenGL VBO extension and the `glDrawElements` call.

GPU linear memory. Next, we perform a series of simulation iterations and write the partial result in the memory used by the VBO. Meshes are then updated in parallel before the VBO is unmapped. All meshes are then rendered by (i) calculating the offset in the index array, (ii) binding all VBO attribute arrays in OpenGL, (iii) setting all particular simplex mesh parameters (e.g. color or clipping planes) and finally calling the `glDrawElements` OpenGL command with the current index offset and bounded arrays as parameters. Finally, we restore the default OpenGL state by unbinding the vertex and index arrays. In addition, we display other useful information such as standard axial, sagittal or coronal slices to explore the volumetric image. To render these slices, the Pixel Buffer Object (PBO) OpenGL extension was used as it supports interoperability with the CUDA architecture. More specifically, a PBO buffer is declared and the 2D slice of our interest is written by using a kernel. Finally, the PBO is used to define a 2D texture which is rendered on a simple quad.

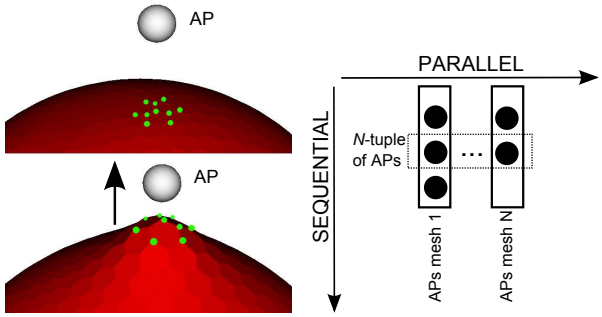
4.5 Interaction

All the parameters affecting the segmentation (e.g., force contribution α_i, timestep) can be dynamically changed, providing a first level of interactivity. Such actions are however insufficient to accurately interact with the segmentation, and "pictorial-input" is rather preferred [38, 22]. We implemented an interaction based on attraction points (AP). For each mesh j, different attraction points can be positioned in the 3D space by clicking on rendered slices. Given $\mathbf{C_j^k}$ the k-th AP associated with mesh j, we find the closest mesh point $\mathbf{P_j^0}$ to $\mathbf{C_j^k}$ and its p-order neighbor points $\mathbf{P_j^i}, \forall i \in [1, N_p]$ (e.g., a 2-order neighborhood is composed of the neighbors of neighbors of $\mathbf{P_j^0}$ as exemplified in Fig. 6(a)), and compute the following weighted attraction forces $\mathbf{f^a}$ applied on each vertex $\mathbf{P_j^i}$:

$$\forall i \in [0, N_p], \mathbf{f}^{\mathbf{a}}_{\mathbf{P_j^i}} = \frac{w_i(\mathbf{C_j^k} - \mathbf{P_j^0})}{\sum_{k \in [0,N_p]} w_k} \tag{2}$$

$$w_i = \alpha_j^a / \left\| \mathbf{C_j^k} - \mathbf{P_j^i} \right\| \tag{3}$$

In practice, various N-tuples of APs $\{\mathbf{C_1^k}, \ldots, \mathbf{C_N^k}\}$ are sequentially processed (Fig. 6(b)), where N denotes the number of meshes. Note that this N-tuple can actually have less than N elements as not all meshes have the same number of APs. Given a N-tuple of APs, the closest point $\mathbf{P_j^0}$ is found on each mesh j by (i) running in parallel a kernel on each mesh point $\mathbf{x_j^i}$ that computes the squared Euclidean distance between $\mathbf{C_j^k}$ and $\mathbf{x_j^i}$, and by (ii) applying a parallel reduction on the array of squared distances to get for each mesh j the index of the closest mesh point $\mathbf{P_j^0}$. Since for each mesh j of M_j points, M_j CUDA threads will access the same memory location containing the coordinates of the AP C_j^k, this information is initially loaded into *shared* memory to speed-up read-accesses. Furthermore, the possibility to interactively select, move or delete each AP is provided to users by intuitive point and grab actions, leading to a real-time control of mesh deformation.



(a) Interactive mesh deformation by APs (b) APs processing

Fig. 6 Interaction by attraction points: (a) a point and its 2-order neighborhood (small discs) of a spherical mesh are attracted by an attraction point (AP) yielding the mesh deformation. (b) In a sequential manner, a N-tuple of APs is processed in parallel to find the points on each mesh to be attracted.

5 Experimental results

5.1 MRI bone segmentation

Our GPU segmentation framework was evaluated under real conditions for the segmentation of the hip joint bones, i.e. femur and hip bone, from Magnetic Resonance Imaging (MRI) (Figs. 8(a)-8(d)). MRI bone segmentation is generally very challenging due to (Fig. 7(a)):

1. Inhomogeneous imaged bone intensity caused by different cortical and trabecular bone tissues [18, 30],
2. A strong proximity of bones in the joint area with unclear and diffused boundaries [33, 30]
3. A possible low image resolution (as in our test images) causing a large partial volume effect.

The first issue affects segmentation approaches such as thresholding, edge detection, level-sets and region growing as they are usually unable to segment a structure with a very inhomogeneous intensity. This issue is addressed by exploiting a prior knowledge of the appearance of the structures to segment, which we consider with our IP-based image force. Similarly, the second and third pitfalls of MRI images also hinder these segmentation approaches as they might diffuse the segmentation evolution across the close and fuzzy bone boundaries (i.e. "leaking" phenomenon). Once again, we efficiently tackle this problem by using a shape prior-based force which regulates the segmentation evolution and reduces its sensitivity to image artifacts.

5.2 Setup

We acquired 28 MRI images of female subjects in the supine position with a 1.5T Philips Medical Systems MRI. The acquisition protocol was: Axial 3D T1, TR/TE= 4.15/1.69ms, FOV/Matrix= 35cm,256×256, resolution= $1.367 \times 1.367 \times 5$ mm^3. The field of view fully covered both right and left hip bones and femurs.

For each bone side (right/left) and type (femur/hip bone), a low resolution simplex mesh was created and initialized in the image by using an interpolation technique [7]. This initialization technique only required the placement of 7 landmarks per bone in the image, which was performed in less than 5 min. A coarse-to-fine strategy was then adopted. All meshes at a given resolution level were simultaneously deformed for a fixed number of iterations. Afterward, the resolution of the meshes was increased and a new simulation was carried out again. The process was repeated until the finest resolution was reached.

We used three different resolutions per bone and adopted, based on empirical tests, an iteration schedule consisting on 300 iterations for the lowest resolution, 100 for the medium resolution and 20 for the highest resolution. Obviously this schedule must be tuned in order according to the number of levels-of-detail (LOD) used in the multi-resolution scheme. In practice, each particle was attributed a mass proportional to its coverage area (Sec. 3.2) multiplied by an arbitrary density. Due to the construction process of the reference mesh, faces were quasi regular and vertices were uniformly distributed over the model surface. As a result, particle mass was almost the same for all particles. Profile size, search depth and weights for image forces were chosen based on our previous works [27, 7]. In particular, we use intensity profiles with 25 and 5 samples spaced by 0.5 mm in the mesh interior and exterior, respectively. Segmentation parameters were kept identical throughout all trials. Furthermore, gold-standard segmentations were produced based on supervised segmentations performed by an experienced researcher under the guidance of a radiologist. The error metric to assess the segmentation accuracy was the average symmetric surface distance (ASSD) [41] measured in mm.

5.3 Results

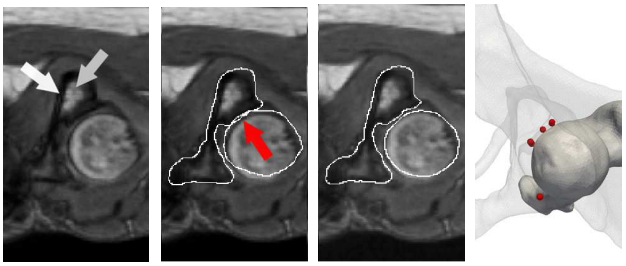
	coarse	medium	fine
$ASSD_{GPU}$ (mm)	1.93±0.50	1.66±0.43	1.62±0.44
$ASSD_{CPU}$ (mm)	2.07±0.70	1.60±0.63	1.58±0.63
$time_{GPU}$ (s)	1.85 (0.006)	1.27 (0.012)	0.42 (0.021)
$time_{CPU}$ (s)	44.9 (0.15)	57.2 (0.57)	29.9 (1.49)
#iterations	300	100	20
#vertices	2656	10624	42496

Table 1 MRI Hip joint bone segmentation results for each mesh resolution level: accuracy error (*ASSD*) for GPU and CPU; times for GPU and CPU with time/iteration in parenthesis; #iterations is the number of iterations and #vertices the total number of vertices for all meshes

We have compared the accuracy and speed of our GPU-framework against our CPU-based implementation of discrete deformable models [27, 7], without collision detection and any advanced shape priors (e.g., statistical shape model) forces in order to make comparison more fair for the GPU version (See Table 1). Experiments were run on a single-core 3.40 GHz PC equipped with an NVidia GTX 8800 graphics board with 768 Mb TRAM. Computation times listed in Table 1 do not account for loading, saving and rendering of the meshes.

The GPU approach was consistently about $25-70\times$ faster than the CPU version to execute a single time step. As expected, this difference became more significant when the number of vertices increased, as e.g. the CPU needed 1.49 s to process 42K vertices while 21 ms were only necessary for the GPU implementation. These figures also show that the parallelization is not fully performed at the vertex level as an increase of the vertices yielded a sensitive increase of the time spent to perform an iteration. Nevertheless, this time did not scale linearly with the number of vertices (it seems to double when the number of vertices is quadrupled) which highlights the presence and good performance of the underlying parallelization. Most importantly, the update frequencies of the GPU approach were consistently above the minimum 10Hz of refresh rates required for interactivity.

In terms of accuracy, both CPU and GPU approaches returned similar results. Indeed, the final ASSD error of the CPU segmentation was 1.58 ± 0.63 against 1.62 ± 0.44 for



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

Fig. 7 Correction of bone segmentation by attraction points (APs). a) The hip joint area is challenging to segment due to a strong difference in intensity between cortical (bone exterior, white arrow) and trabecular (bone interior, gray arrow) bone. Furthermore, hip bone and femur are very close and may present fuzzy boundaries. b) Result without APs where the red arrow points at mesh inter-penetrations. c) Corrected result with APs, which are displayed as d) small spheres in a 3d view with a transparent hip bone.

the GPU implementation. Similarly, errors between both approaches were very close at the end of each iteration sequence which exploited a given mesh resolution. These similar accuracy results highlight the correct implementation of the segmentation into the GPU formalism.

A visual inspection (Fig. 8) confirmed a satisfactory segmentation in most bony regions. Nevertheless, the current GPU approach suffered from mesh inter-penetrations observed in the joint area (Fig. 7(b)). However, thanks to the new possibilities of higher update frequencies of the GPU approach, attraction points could be easily added in real-time to correct bad segmentation results as shown in Fig. 7(c), where inter-penetrations were avoided improving the ASSD about 9% (1.52 to 1.40 mm) for this particular subject.

6 Discussion

In this Section, we discuss our GPU implementation with respect to key aspects of image segmentation. Comments on possible improvements of the framework through extensions are also provided.

6.1 Accuracy and robustness

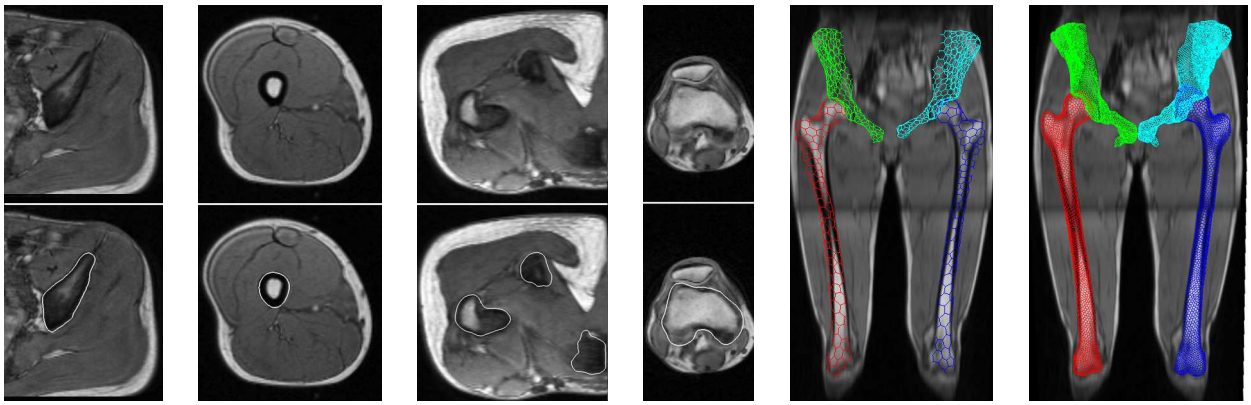
In terms of accuracy, both CPU and GPU approaches returned similar results. Small discrepancies between both approaches can be explained by the use of single precision floating-point in the GPU approach, compared to double precision for the CPU segmentation. Furthermore, the image interpolation performed by the texture sampler may slightly differ from a CPU calculation. These minor differences can yield after several iterations to minor variations observed in the results.

Regardless the chosen approach, the segmentation achieved satisfactory results given the relatively low image resolution of these clinical images ($1.367 \times 1.367 \times 5$ mm^3). The sub-voxel standard deviation error also revealed the consistency of the segmentation over the trials with the 28 subjects. Visual inspection of the segmented images (See Figs. 8(a) to 8(d)) confirmed the good quality of the segmentation in most of the bony regions. However, the articular area sometimes presented segmentation errors mainly due to the pitfalls listed in Sec. 5.1, namely the bone proximity and the presence of fuzzy boundaries.

We discussed the possibility to tackle these errors in the articular region by using attraction points (Fig. 7). While this interactive action is efficient in most of the cases, it is preferred to exploit additional corrective techniques to minimize or simply remove the user interaction. One of them is the use of more appropriate internal regularization forces. Typically, the exploitation of more sophisticated shape priors such as statistical shape models (SSM) has proven to be quite effective to segment these challenging MRI images [27, 29]. Moreover, collision response and detection techniques could prevent mesh inter-penetrations and hence bring additional robustness. The implementation of SSM-based forces in GPU may demand quite complex numerical tools. In particular, Singular Value Decomposition (SVD) is required to achieve the alignment of point-sets [39] in order to apply the SSM iterative regularization process [3]. Many of these tools are common but demand effort to be efficiently implemented into GPU formalism. Thankfully, they are progressively ported to CUDA, like the SVD implementation of Lahabar and Narayanan [15]. However, their integration into an existent framework is not always straightforward as special data structures are often required. These structure do not necessarily satisfy the requirements of our simulation, and as a result this may demand a careful (re-)design of the framework.

6.2 Speed and interactivity

The GPU approach was consistently about $25-70\times$ faster than the CPU version to execute a single time step. This was predicable since the GPU implementation benefited from parallel processing while the CPU-based approach operated in a sequential manner. In all cases, the time taken by a single GPU iteration is perfectly matching with interactivity constraints. Indeed, update frequencies for the GPU were about $47-162$Hz, thus easily supporting the minimum 10Hz of refresh rates required for interactivity. On the other hand, the CPU version was unable to fully support interactivity, since update frequencies were about $0.6-6.7$Hz. Having full support to interactivity paves the way to novel segmentation approaches, in which the user is fully able to interact with the segmentation loop.



(a) (b) (c) (d) (e) (f)

Fig. 8 GPU-based MRI hip joint bones segmentation examples: from a) to d) axial slices are shown without and with white mesh overlays respectively on top and bottom of the subfigure. In e), a coarse mesh is initialized at the beginning of the segmentation, and in f) the final result is shown with meshes at their higher resolution

Different steps in the simulation (i.e. mesh update, forces computation and numerical integration) are performed in a sequential order for both GPU and CPU implementations. For each step, we look for a parallelization at vertex level. The speed-up found between both GPU and CPU versions is thus essentially dictated by the number of particles involved in the simulation.

We have measured that, in the GPU implementation, force computation accounts for approximatively 99% of total time per iteration, mesh update and numerical integration contributions being below 1%. Hence, an additional level of parallelization could be achieved by computing all forces in parallel. This force parallelism could be achieved with atomic operations which would sum up the force contributions in parallel in the force accumulation array. Since atomic operators availability depends on the CUDA compute capability of the GPU, an alternative approach consists in using a proper array for each force to avoid memory writing conflicts and thus the use of atomic operations. However, such an implementation is obtained at the expense of a larger memory consumption.

In our segmentation context, effort should be instead spent in speeding up image-based forces, and especially those using intensity profile. In fact, Fig. 9 reports the time taken by each type of force in the GPU implementation. IP-based force clearly dominates with its 93% of the total force computation time. Despite we used an efficient computation of the NCC by caching invariant quantities only dependent of the reference profiles and by using an iterative summation of denominator terms, this force remains very expensive. The main reason is that during the search for the optimal target position $\mathbf{y}^*$ with the lowest image energy, we compute the IP-based energy m times in a sequential manner. Hence, once again we could parallelize the computation of the energy for each position y_i, $i \in [1, m]$. Before considering such parallelization, special attention should be paid in preserving a good tradeoff between memory usage, speed and implementation complexity.

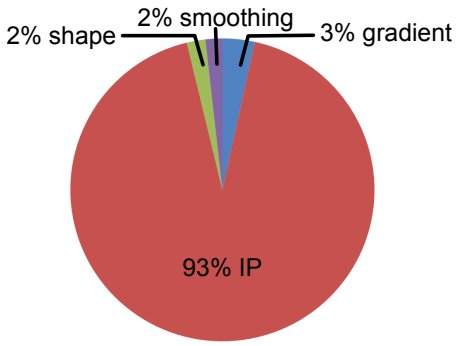


Fig. 9 Average time distribution for forces in GPU implementation.

Finally, it is sure that a multi-core CPU hardware architecture exploited with an efficient parallel implementation, will notably perform faster. Nevertheless, the speed-up obtained is expected to be not as significant as the one obtained with our GPU-based approach. However, CPU architectures have some non-negligible advantages compared to GPU's, such as a greater flexibility in programming or a universal support for double precision arithmetics.

7 Conclusions and future work

The goal of this work was to design a simple yet efficient and extensible framework to allow the simulation, manipulation and rendering of deformable models for segmentation purpose. Since, to date, any segmentation approach is prone to errors due to the extreme variety of segmentation conditions, the focus of this work was to devise a fast approach

which provided an interactive control on the segmentation evolution. Its performances were illustrated in a challenging segmentation scenario in which possible segmentation errors were interactively corrected. The GPU hardware requirements were not very demanding, as a slightly outdated CUDA compatible graphics board already demonstrated an excellent interactivity.

Still, potential limitations of the GPU framework will be overcome through extension mechanisms, we particularly target better segmentation accuracy and robustness by considering additional segmentation strategies such as internal forces based on statistical shapes models and collision detection techniques to prevent mesh inter-penetrations. Although this is not an easy task as many existing approaches do not easily translate into GPU parallel formalism. Furthermore, effort will be spent in finely tuning the CUDA implementation to better address technical aspects such as memory bank conflicts or multiprocessor occupancy. Finally, richer interactive visualization approaches will be also explored, such as advanced volume rendering built upon efficient implementations able to handle larger volumetric images.

Acknowledgements This work is partially supported by the EU Marie Curie Program under the 3D Anatomical Human project (MRTN-CT-2006-035763). We thank all the volunteers who took part to this study as well our medical partner the University Hospital of Geneva.

References

1. Caselles V, Kimmel R, Sapiro G (1997) Geodesic active contours. Int J Comput Vis 22(1):61–79
2. Cates JE, Lefohn AE, Whitaker RT (2004) Gist: an interactive, gpu-based level set segmentation tool for 3d medical images. Med Image Anal 8(3):217–231
3. Cootes TF, Hill A, Taylor CJ, Haslam J (1993) The Use of Active Shape Models for Locating Structures in Medical Images. In: Proc. IPMI, Springer-Verlag, vol LNCS 687, pp 33–47
4. Crassin C, Neyret F, Lefebvre S, Eisemann E (2009) Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In: Proc. SIGGRAPH I3D, ACM Press, pp 15–22
5. Delingette H (1999) General Object Reconstruction Based on Simplex Meshes. Int J Comput Vis 32(2):111–146
6. Georgii J, Westermann R (2005) Mass-Spring Systems on the GPU. Simulat Model Pract Theor 13:693–702
7. Gilles B, Magnenat-Thalmann N (2010) Musculoskeletal MRI segmentation using multi-resolution simplex meshes with medial representations. Med Image Anal 14(3):291–302
8. Gobbetti E, Marton F, Iglesias Guitián J (2008) A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. Vis Comput 24(7-9):797–806, proc. CGI 2008
9. Göddeke D, Strzodka R, Mohd-Yusof J, McCormick P, Buijssen S, Grajewski M, Turek S (2007) Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. Parallel Computing 33(10-11):685–699
10. He Z, Kuester F (2006) GPU-Based Active Contour Segmentation Using Gradient Vector Flow. Advances in Visual Computing 4291:191–201
11. Holden M, Hill D, Denton E, Jarosz J, Cox T, Hawkes D (1999) Voxel similarity measures for 3D serial MR brain image registration. In: Proc. IPMI, Springer, Springer, vol LNCS 1613, pp 472–477
12. Kass M, Witkin A, Terzopoulos D (1988) Snakes: Active contour models. Int J Comput Vis 1(4):321–331
13. Kienel E, Brunnett G (2009) Tile-based Image Forces for Active Contours on GPU. In: Proc. Eurographics, Eurographics Association, pp 89–92
14. Köhn A, Drexl J, Ritter F, König M, Peitgen H (2006) GPU accelerated image registration in two and three dimensions. In: Proc. BVM, Springer, pp 261–265
15. Lahabar S, Narayanan PJ (2009) Singular value decomposition on gpu using cuda. In: Proc. IPDPS, IEEE Computer Society, pp 1–10
16. Lefohn AE, Cates JE, Whitaker RT (2003) Interactive, GPU-Based Level Sets for 3D Segmentation. In: Proc. MICCAI, Springer, vol LNCS 2878, pp 564–572
17. Liu J, Sun J, Shum H (2009) Paint selection. ACM Trans Graph 28(3):1–7
18. Lorigo LM, Faugeras OD, Grimson WEL, Keriven R, Kikinis R (1998) Segmentation of Bone in Clinical Knee MRI Using Texture-Based Geodesic Active Contours. In: Proc. MICCAI, Springer-Verlag, vol LNCS 1496, pp 1195–1204
19. Mosegaard J, Herborg P, Sorensen T (2005) A GPU accelerated spring mass system for surgical simulation. Stud Health Tech Informat 111:342–348
20. Muyan-Ozcelik P, Owens J, Xia J, Samant S (2008) Fast deformable registration on the GPU: A CUDA implementation of demons. In: Proc. ICCSA, IEEE Computer Society, pp 223–233
21. NVIDIA Corporation (2010) NVIDIA CUDA Programming Guide, ver. 3.1
22. Olabarriaga S, Smeulders A (2001) Interaction in the segmentation of medical images: A survey. Med Image Anal 5(2):127–142
23. Østergaard Noe K, De Senneville B, Elstrøm U, Tanderup K, Sørensen T (2008) Acceleration and validation of optical flow based deformable registration for image-guided radiotherapy. Acta Oncol 47(7):1286–1293

24. Pan W (2009) Improving Interactive Image Segmentation via Appearance Propagation. In: Alliez P, Magnor M (eds) Proc. Eurographics, Eurographics Association, pp 93–96
25. Rodriguez-Navarro J, Susin A (2006) Non structured meshes for Cloth GPU simulation using FEM. In: Mendoza C, Navazo I (eds) Proc. VRIPHYS, Eurographics Association, pp 1–7
26. Santner J, Unger M, Pock T, Leistner C, Saffari A, Bischof H (2009) Interactive Texture Segmentation using Random Forests and Total Variation. In: Proc. BMVC, BMVA Press, London, UK, to appear
27. Schmid J, Magnenat-Thalmann N (2008) MRI Bone Segmentation using Deformable Models and Shape Priors. In: Proc. MICCAI, Springer-Verlag Berlin Heidelberg, vol LNCS 5241, pp 119–126
28. Schmid J, Nijdam N, Han A, Kim J, Magnenat-Thalmann N (2009) Interactive Segmentation of Volumetric Medical Images for Collaborative Telemedicine. In: Modelling the Physiological Human, Proc. 3DPH, Springer, vol LNCS 5903, pp 13–24
29. Schmid J, Kim J, Magnenat-Thalmann N (2010) Extreme leg motion analysis of professional ballet dancers via MRI segmentation of multiple leg postures. Int J Comput Assist Radiol Surg In Press
30. Sebastian T, Tek H, Crisco J, Kimia B (2003) Segmentation of carpal bones from CT images using skeletally coupled deformable models. Med Image Anal 7(1):21–45
31. Sharp G, Kandasamy N, Singh H, Folkert M (2007) GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration. Phys Med Biol 52(19):5771–5784
32. Sherbondy A, Houston M, Napel S (2003) Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In: Proc. VIS, IEEE Computer Society, Washington, DC, USA, p 23
33. Snel J, Venema H, Grimbergen C (2002) Deformable triangular surfaces using fast 1-D radial Lagrangian dynamics-segmentation of 3-D MR and CT images of the wrist. IEEE Trans Med Imag 21(8):888–903
34. Stoev S, Straβer W (2000) Extracting regions of interest applying a local watershed transformation. In: Proc. Conf. Visualization, IEEE Computer Society, pp 21–28
35. Strzodka R, Droske M, Rumpf M (2004) Image registration by a regularized gradient flow. a streaming implementation in dx9 graphics hardware. Computing 73(4):373–389
36. Tejada E, Ertl T (2005) Large steps in GPU-based deformable bodies simulation. Simulat Model Pract Theor 13(8):703–715
37. Teßmann M, Eisenacher C, Enders F, Stamminger M, Hastreiter P (2008) GPU Accelerated Normalized Mutual Information and B-Spline Transformation. In: Botha C, Kindlmann G, Niessen W, Preim B (eds) Proc. Eurographics Workshop VCBM, Eurographics Association, Delft, The Netherlands, pp 117–124
38. Udupa JK (1982) Interactive segmentation and boundary surface formation for 3-d digital images. Computer Graphics and Image Processing 18(3):213 – 235
39. Umeyama S (1991) Least-squares estimation of transformation parameters between two point patterns. IEEE Trans Pattern Anal Mach Intell 13(4):376–380
40. Unger M, Pock T, Trobin W, Cremers D, Bischof H (2008) Tvseg - interactive total variation based image segmentation. In: Proc. BMVC, BMVA Press
41. Van Ginneken B, Heimann T, Styner M (2007) 3D Segmentation in the Clinic: A Grand Challenge. In: Heimann T, Styner M, Van Ginneken B (eds) Proc. MICCAI Workshop 3D Segmentation in the Clinic: A Grand Challenge, Srpinger, pp 7–15
42. Verlet L (1967) Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. Physical Review 159(1):98–103
43. Walters J, Balu V, Kompalli S, Chaudhary V, Buffalo N (2009) Evaluating the use of GPUs in liver image segmentation and HMMER database searches. In: Proc. IPDPS, IEEE Computer Society, pp 1–12