High-Performance Host-Device Scheduling and Data-Transfer Minimization Techniques for Visualization of 3D Agent-Based Wound Healing Applications

N. Seekhao\textsuperscript{1}, G. Yu\textsuperscript{2}, S. Yuen\textsuperscript{2}, J. JaJa\textsuperscript{1}, L. Mongeau\textsuperscript{3} and N. Y. K. Li-Jessen\textsuperscript{2}
\textsuperscript{1}Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA
\textsuperscript{2}School of Communication Sciences and Disorders, McGill University, Montreal, Québec, Canada
\textsuperscript{3}Department of Mechanical Engineering, McGill University, Montreal, Québec, Canada

Abstract—High-fidelity numerical simulations produce massive amounts of data. Analyzing these numerical data sets as they are being generated provides useful insights into the processes underlying the modeled phenomenon. However, developing real-time in-situ visualization techniques to process large amounts of data can be challenging since the data does not fit on the GPU, thus requiring expensive CPU-GPU data copies. In this work, we present a scheduling scheme that achieve real-time simulation and interactivity through GPU hyper-tasking. Furthermore, the CPU-GPU communications were minimized using an activity-aware technique to reduce redundant copies. Our simulation platform is capable of visualizing 1.7 billion protein data points in situ, with an average frame rate of 42.8 fps. This performance allows users to explore large data sets on remote server with real-time interactivity as they are performing their simulations.

Keywords: high-performance agent-based modeling, real-time in situ visualization, heterogeneous platform scheduling, wound healing, vocal fold model

1. Introduction

The attempt to understand real-world phenomena through modeling has been ubiquitous. The prevalence of computing power has given researchers and scientists tools to realize increasingly complex models. As models gain higher degree of fidelity, the complexity and amounts of data involved grow. To make sense of these models, the analysis and visualization of the output data as they are being generated become crucial. However, efficiently handling such amounts of data interactively requires a carefully designed mechanism to make the most effective use of the available resources. The resulting mechanism should exhibit two desirable characteristics: (1) real-time performance, (2) high level of interactivity, which enable computational steering and interactive data exploration.

Graphics processing units (GPUs) are typically used for rendering as they are specialized accelerators designed for optimized graphics operations. Since GPUs were not designed to serve as stand-alone devices, they need a host, namely central processing units (CPUs), to operate. This host-device setup is a common configuration in modern personal machines. This work, thus, focuses on a single-node with multi-core CPU and multiple GPUs platform configuration.

The bottleneck of applications using GPU accelerators is usually the communication between the host and the device, and the relatively limited amount of memory available on the GPUs. First, any data to be stored and used on the GPUs would have to be transferred from the CPU host through a peripheral component interconnect express (PCIe) bus which is very slow (bandwidth of approximately 2.0 to 8.0 GB/s) compared to the GPU memory bandwidth (typically hundreds of GB/s). Second, GPU global memory is usually much smaller compared to that of the CPU host. Thus, in large data applications, memory transfer overheads become formidable due to the on-demand data transfers from host to device.

While multiple simulation approaches apply well to biological applications, this work focuses on agent-based modeling (ABM) due to its bottom-up structure that makes cell modeling and incremental knowledge aggregation relatively simple. The ABM approach has been widely used to quantitatively simulate complex dynamical systems [1]. Many researchers have gained insights into their system of interest using ABMs [2], [3]. As model complexity grows, improvements through high-performance computing (HPC) techniques to ABM have been reported in a number of publications [4], [5], [6], [7], [8], [9]. Unfortunately, despite the significance of data visualization in output data interpretation, only a few of these works presented improvements on the visualization components. Our goal is to fill in this gap by presenting a combination of techniques that enable effective and interactive in situ visualization of the simulated data concurrently with the simulation.

Our previous work [10] introduced a scheduling scheme to fully utilize the power of both CPU and GPUs when performing simulation computations. Although the performance of the simulation computation outperformed other related work, the visualization interactivity was very low as the output data were distributed among the host and different devices. In this study, we addressed this shortcoming by reconstructing the scheduling scheme to accommodate the vi-
sualization process by hyper-tasking the GPUs and allowing all GPUs to participate in the rendering process through the compression of their local data. Furthermore, we developed a host-device data-transfer minimization technique to decrease the overhead of CPU-GPUs data copies of structural data.

2. Background

2.1 Software and Hardware Environment

Our base ABM framework was written in C++. The parallelization on multi-core CPUs was written with Open-Multiprocessing (OpenMP). OpenMP is a portable application programming interface (API) for multi-threading on shared memory platforms. The communication with GPUs was implemented using Compute Unified Device Architecture (CUDA) version 8.0.44. All graphics were implemented with legacy OpenGL (Open Graphics Library). The visualization was made in situ [11] using VirtualGL [12] and TurboVNC [13]. The pre-processing of data to predict best parameters for our host-device data minimization algorithm was developed in Python 3.7.

The program was compiled using the intel C++ compiler, icpc version 15.0.0, with release default optimization level. The source code of the vocal fold ABMs along with all optimizations described in this work can be found at https://github.com/HPC-ABM/vf-version_7_3.

All results reported in this work were obtained from program execution and benchmark on a single compute node consisting of a 44-core Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20 GHz host with 128 GB of main memory and two accelerators, NVIDIA Tesla M40. Each NVIDIA Tesla M40 graphics card consists of 3072 CUDA cores and 22.4 GB of global memory with compute capability 5.2.

2.2 Agent-Based Modeling (ABM)

Agent-based modeling is a widely-used approach to quantitatively simulate complex dynamical systems [1]. Each ABM is defined by the autonomous agents whose decisions and actions are governed by user-defined deterministic and stochastic rules. As opposed to equations-based approaches, ABM is decentralized. That is, the system behavior is determined by the collective property derived from individual agents in the system. Thus, ABM simulations aim to gain insight of complex systems by yielding emergent behaviors from a group of simple individuals. The basic components of ABM are:

- **Agent** — Autonomous objects that perform action and interact with other agents and their environment
- **Agent Rules** — Set of pre-defined deterministic/stochastic behaviors of agents
- **ABM World** — The environment that houses the agents. This space can be continuous or discrete

Agents can represent any individual from cities in economic models to inflammatory cells in systems biology models. These agents ‘think’ and make decisions based on the ABM rules. These decisions turn into actions that affect other agents and their environment (ABM world).

2.3 Case Study — Vocal Fold Tissue Inflammation and Repair

Inflammation and wound healing is a complex process that our body uses to restore damaged tissue. The process involves a number of interacting chemical, biological and mechanical phenomena [14]. The spatio-temporal scale of some of these phenomena vary substantially from others. Despite the tremendous efforts spent in various studies [14], understanding the precise underlying mechanisms of wound healing process still remains an open challenge. Computational models can serve as a tool to verify current understanding of the problem with the real phenomena. Interactive visualization concurrent to the execution of the simulation can help refine the agent-rules of the model.

Due to availability of experimental data [15], [16], the ABM rules, constants, and parameters used in the models were derived mainly from empirical vocal fold experiments and literature reviews [17], [18], [19], [20], [21], [22], [23], [24]. The main components of the vocal fold (VF) ABM are biological cells, chemical signals and extracellular matrix (ECM) proteins. The biological cells serve as the main actors of the model who communicate via chemical signals, and perform healing via secretion of ECM proteins. The roles of specific cell types and ECM proteins, along with their integration into the VF ABM is described in our previous work [10].

3. Related Work

3.1 HPC ABMs for Biological Applications

High-fidelity ABMs in biology (Bio-ABM) often involve with large amounts of data. Multiple high-performance computing (HPC) ABM tools have been developed to address the challenges in processing these large data sets. For example, FLAME [8] is an implementation of an ABM framework for parallel architectures based on stream X-machines. FLAME has been used to speed up the simulation of ecological systems in various fields including systems biology [25]. FLAME was further extended with a GPU support [26]. SugarScape on steroid [4] is another example of ABM acceleration on GPU platforms. These tools have demonstrated their applicability to biological system simulations such as tissue wound and disease modeling [25], [27], [28].

In the realm of distributed computing, Repast HPC [5] was developed as an MPI extension to its predecessors, Rapast and Repast Symphony [29]. Repast HPC was adopted to accelerate the simulation of bone tissue growth [6].

Due to the popularity of ABMs in biological applications, some HPC ABM tools have also been developed specifically for biological applications. An example includes AgentCell,
a Repast-based framework for single-cells and bacterial population [30]. The AgentCell framework provides support for running multiple non-interacting single-cell instances concurrently on massively parallel computers. Chaste [31] is an example of bio-ABM framework utilizing HPC platforms via various existing high-performance libraries like PETSc and (par)METIS for parallel linear algebra and mesh distribution. More examples include HPC ABM frameworks for multi-core CPUs such as CompuCell3D [7], CellSys [32], and Morpheus [33]. In these frameworks, parallelization was performed with OpenMP to speed up the performance on single-node multi-core CPUs. In addition, other techniques have been proposed to accelerate specific biological models on multi-core CPUs or GPUs. The inability to exploit both CPUs and GPUs simultaneously results in a sub-optimal resource utilization. Our previous work [36], [10] addressed this issue by assigning appropriate computational tasks to the CPU while it waits for the GPUs to complete their computations. However, these previous works only focused on the simulation computation. This leaves an opportunity in visualization optimization an open problem.

### 3.2 3D Visualization in Bio-ABMs

The challenge in developing HPC bio-ABM has been relatively well-explored. However, the same does not apply to the visualization aspect of ABM frameworks. Due to the scales of the data size being generated, the conventional method of post hoc visualization has become increasingly infeasible. The post hoc method refers to a visualization process where the visualization is performed after the numerical simulation has completed. The output data are written to disk during the simulation and retrieved later for visualization. This visualization method puts an enormous load on the disk and network. For this reason, another type of visualization, in situ visualization, has gained the interests of researchers [11]. In situ visualization allows the outputs to be analyzed on the same machine that produced them. The ability to perform on-site data analysis reduces the amount of data movements between the server and remote users. This property makes in situ visualization an ideal way to visualize simulations that produce large data sets such as our case. Paraview Catalyst [37] and work reported in [38] are examples of libraries developed to enable in situ processing of simulation output on popular existing visualization frameworks such as Paraview [39] and VisIt [40]. A bitmap-based and a quadtree-based ABM approach [41], [42] were proposed respectively to analyze the numerical output in situ and reduce non-essential simulation data.

### 4. Scheduling

Simulation tasks are either computational or visualization. In general, most simulation computational tasks can be further classified into two categories: coarse-grain or fine-grain. A task is labeled coarse-grain if it involves complex computations and access relatively small amounts of data. These coarse-grain tasks are suitable for CPUs. On the other hand, GPUs excel at executing tasks that involve simple computations that get applied large amounts of data. These tasks are then classified as fine-grain. The scheduling schemes described in our earlier work [36], [10] outperformed other related ABM works in terms of computational throughput. However, since the computational tasks were distributed among the host and different accelerators, visualization was performed at the end of each iteration after all data were copied back to the host and synchronized. The visualization framerate was, thus, bounded by the execution time of the computation and synchronization of the whole iteration.

In this work, the scheduling scheme proposed in [10] was restructured to mitigate the bound on user-simulation interactivity by (1) scheduling GPU tasks at a finer level to enable (2) GPU hyper-tasking (GHT). The benefits of GHT are two folds. First, the GPUs responsible mainly for fine-grain computations (GPU_compute) can hyper-task between fine-grain computations and ray-casting (visualization subtask). This allows these non-screen-attached GPUs to aid in visualizing their local data and minimize peer communications. Second, GHT allows the rendering GPU (GPU_vis) to lighten the computational loads of its peers by executing leftover fine-grain subtasks, while maintaining high level of visualization interactivity.
Fig. 1 demonstrates the workflow of the proposed scheduling scheme. This scheme was executed with \( p \) CPU threads, where \( p \) denotes the number of available CPU cores. A total of \( p - N_{GPU} \) threads get used for coarse-grain computations. This part remains unchanged from our previous work [36], [10]. In this work, the rest of the threads are not simply launching fine-grain tasks, but rather communicating with the rendering GPU (\( GPU_{vis} \)) and compute GPUs (\( GPU_{compute} \)) to orchestrate GHT.

The program uses \( N_{GPU} – 1 \) threads for communications with the GPUs responsible mainly for fine-grain computations (\( GPU_{compute} \)). Each time a \( GPU_{compute} \) completes its fine-grain task, the GPU compresses the output and buffer it for visualization in the next iteration. In addition, these GPUs participate in visualizing its local data from the previous iteration when they finish a task by performing ray-casting locally. Each \( GPU_{compute} \) then sends the results as GLtexture to the \( GPU_{vis} \) for rendering. This prevents the GPUs from having to send large 3D volumetric data to their peer, and only send ready-to-render 2D textures to \( GPU_{vis} \).

One thread is spared for communication with \( GPU_{vis} \). The main responsibility of \( GPU_{vis} \) is to perform visualization of all outputs including those residing on the CPU host and other GPUs (\( GPU_{compute} \)). However, between visualization frames, \( GPU_{vis} \) switches to compute mode to assist in fine-grain computations. Since, other compute GPUs spare some of their resources to assist with the visualization process to reduce amounts of data being sent off-device, it is necessary for \( GPU_{vis} \) to lighten the loads of other GPUs whenever it can for optimal performance.

Our tested configuration consisted of one multi-core CPU and two GPUs. However, this scheduling scheme can be extended to work with more than two GPUs by executing the scheduling logic of \( GPU_{compute} \) shown in Fig. 1 on additional GPUs. The added GPUs will be beneficial if the execution time of the fine-grain tasks on GPUs exceeds that of the CPU coarse-grain tasks. This scheduling scheme, thus, scales out with additional GPUs as the amount of fine-grain tasks increases.

5. Host-Device Data Transfer Minimization

So far, we have only discussed an efficient scheduling technique to compute and visualize the simulation outputs. However, the details of data movements between devices still need to be examined as off-device data copies are relatively time consuming. In this section, we will discuss the technique used to minimize the copies of redundant data.

5.1 Observations

For optimization purposes, we categorize a dynamic pattern of a data type by two properties. First, a pattern is defined by the distribution characteristics in a single iteration. For example, the distribution of a chemical concentration is typically smooth due to diffusion, whereas ECM protein and cell distributions exhibit some degree of discontinuity. Second, the dynamics are defined by how the distribution changes from one iteration to another, for example localized vs. distributed changes. Table 1 shows the categorization of dynamic patterns for the different types of output and which device they were computed on.

<table>
<thead>
<tr>
<th>Table 1: Data Dynamic Pattern Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Localized Distributed</td>
</tr>
<tr>
<td>Smooth</td>
</tr>
<tr>
<td>I—ECM Proteins</td>
</tr>
<tr>
<td>GPU</td>
</tr>
</tbody>
</table>

Notice that none of the patterns observed in wound healing ABMs fall into category 0. Data type that are discontinuous and distributed fall into category III. Examples include mobile agents such as neutrophils, macrophages and fibroblasts, or non-mobile agents such as platelets. In this work we focused our optimization efforts only on protein (signaling and ECM) data which fall into category I and II. The direction and challenges of developing optimization for category III will be discussed in Section 7.

Data category I are quantities that do not diffuse or spread, thus changes are typically local with discontinuous distributions. These are mostly structural quantities. For example, ECM proteins form an underlying structure of the tissue and provide building blocks of the overall structure similar to laying out bricks for a building. Once a brick is laid, it stays there until it is degraded or destroyed. In wound-healing models, these quantities are localized, and thus changes to these quantities happen mostly in highly-active areas such as wound sites.

In contrary, category II data diffuse or spread, thus their distributions are typically smooth. Once the mass gets deposited at a location, it undergoes immediate changes in concentrations and reaches other areas. This includes diffusible quantities such as chemicals, gas, and heat. The dynamics of these quantities are very distributed in nature. The changes usually keep happening and affect other areas until an equilibrium is reached. However, in wound healing models, inflammatory cells may not stop protein secretion until the wound is healed. Thus, an equilibrium may not be reached until the end of the simulation, resulting in constant changes all over the simulation area.

Since the computations of diffusible quantities involve relatively large amounts of data, they were performed on GPUs. These output data are, thus, local to the GPUs. As mentioned in Section 4, to lower the amounts off-device data copies, these data were compressed and store on the producer GPU using linear sampling so each GPU can assist in ray-casting. Through the CUDA texture filter mode, cudaFilterModeLinear, a sample fetched from a CUDA texture is automatically interpolated. An exper-
Fig. 2: An example sequence demonstrating the Host-Device Activity-Aware Data Copy (HADC) technique to minimize the copying of redundant category I data. First, the ABM world gets divided up into smaller sub-volumes. In each iteration, when a patch is modified, the program identifies the parent sub-volume of that patch. The algorithm then looks up the corresponding entry in the activity map and increments its activity index. Only the sub-volumes with significant changes get copied to the GPU for visualization.

The computations of indiffusible structural quantities (data category I) involve relatively small amounts of data so they were performed on CPU. Since these data are local to the CPU host, host-device data copies are inevitable during the visualization process.

5.2 Data Category I: Host-Device Activity-Aware Data Copy (HADC)

As discussed in the previous section, this type of data involves structural change and the outputs reside on the CPU host. These data, thus, need to be copied from CPU to GPUvis during visualization. Structural data change locally i.e. the change in one location does not directly impact other locations. More specifically, in wound healing models, most activities occur in the damaged areas. Given these observations, we developed a Host-Device Activity-Aware Data Copy (HADC) technique to reduce the copying of redundant data.

5.2.1 HADC Algorithm

Fig. 2 demonstrates the work flow of HADC during the simulation. First, the ABM world is divided into smaller sub-volumes. In each iteration, when a patch gets modified, the program identifies the parent sub-volume of that patch. The algorithm then looks up the corresponding entry in the activity map and increments its activity index. During the data buffering stage, only the data in the sub-volumes that had gone through significant changes get copied to the GPU for visualization. In this way, the amount of redundant data copies is minimized, resulting in a faster buffering process.

5.2.2 HADC Sub-Volume Size Recommendation Heuristic

The size of the sub-volumes is a very important factor in determining the performance of HADC. If the size is too large, a large amount of unnecessary data copies will be performed. However, if the size is too small, since each small sub-volume will be copied in a separate transaction, the latency of host-device copy operations will hurt the overall performance. To find an optimal partition, the program runs the model, and logs the locations of activities in each iteration. For a given $M$ points (representing activity locations) in a log from the most active iteration, the goal is to find a partition of size $w \times h \times d$, where the $M$ points appear in as few partitions as possible. In other words, the objective is to minimize the partition size (space) and number of communications (time) between host and device, where the two extremes are $< O(WHD) space, O(1) time >$ and $< O(1) space, O(M) time >$.

To determine a good partition size, a modified octree algorithm is used. This heuristic consists of two passes: top-down and bottom-up. At the end of each pass, each node (partition) will either be labeled white (insignificantly low density) or black (significantly high density). The first pass starts top-down from the whole simulation grid, and split on the density conditions given the parameters $\rho_{low}$ and $\rho_{high}$. Once the program is done splitting as all nodes satisfy one of the density conditions (node density $< \rho_{low}$ or $> \rho_{high}$), the program will return a list of black node dimensions. Here, the smallest volume dimensions will be picked to minimize the partition size. The program then execute the second pass in an attempt to combine consecutive partitions if and only if the density condition requirements still hold. This bottom-up pass is executed to further minimize the number of host-device communications.

Given a wound healing model $m = \langle wc, mc, ic, r \rangle$, where $wc$ denotes the wound configuration (dimensions, wound location), $mc$ denotes model configuration (dimensions), $ic$ denotes the initial conditions (patient’s cytokine
levels, treatment type etc.) and \( r \) denotes the model rules, if we know the optimal subvolume size of \( m \), then we can choose a scaled subvolume size for model \( m' = \langle wc', mc', ic', r \rangle \). For example, if the wound depth of \( m \) and \( m' \) is 1 mm and 0.5 mm respectively in the x-direction, then we can scale the optimal subvolume size of \( m \) in the x direction by \( \frac{1}{2} \) and reuse it for \( m' \). This way, once we determined an optimal subvolume size for model \( m \), we can apply it to any model \( m' \) defined earlier.

6. Results

6.1 Model Configurations

We have developed VF ABM models for two different mammals: rat and human. As discussed in Section 2.3, the rat VF ABM serves as a test model due to its size and the availability of empirical data. The model configurations were determined based on empirical data and vocal fold literature reviews [17], [18], [19], [20], [22], [23], [24]. Table 2 summarizes the configurations of both models.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Rat</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>World Size</td>
<td>3D Patches</td>
<td>153.9 M</td>
</tr>
<tr>
<td>ECM Data</td>
<td>Data Points</td>
<td>0.46 G</td>
</tr>
<tr>
<td>Chemical Data</td>
<td>Data Points</td>
<td>1.23 G</td>
</tr>
<tr>
<td>Platelets</td>
<td>Initial Number of Cells</td>
<td>78.8 K</td>
</tr>
<tr>
<td>Neutrophil</td>
<td>Initial Number of Cells</td>
<td>517</td>
</tr>
<tr>
<td>Macrophage</td>
<td>Initial Number of Cells</td>
<td>315</td>
</tr>
<tr>
<td>Fibroblast</td>
<td>Initial Number of Cells</td>
<td>3.5 K</td>
</tr>
</tbody>
</table>

6.2 Performance Evaluation

To evaluate the quality of the subvolume size recommendation made by the heuristic proposed in Section 5, an experiment was run with different HADC partition sizes for different model and wound configurations. As shown in Fig. 4, the 2-pass octree technique worked well for small and medium size wounds (sizes up to \( \frac{1}{2} \) of the VF size). However, it did not work well for large wound sizes as it recommended a sub-optimal partition. Thus, it is best for the modeler to use the heuristic to obtain the recommendation from smaller wound model, \( m \) and scale the partition size for \( m' \) with larger wounds.

The original frame rate of the human VF visualization was bounded by the compute time of approximately 7 s per iteration [36], [44]. This resulted in poor interactivity due to the 0.14 fps frame rate. With the scheduling technique described Section 4, the frame rate improved to 7.2 fps. By using the subvolume size recommended by the heuristic proposed in Section 5 for HADC optimization, the visualization performance improved significantly to 42.8 fps.

For performance evaluation purposes, the human VF ABM was compared against our previous and other similar ABM work (Table 3). The bacteria-macrophage-antibiotic ABM was implemented with FLAME GPU [27]. FLAME GPU is a widely used modern HPC ABM framework, and thus, serves as a good performance standard. Additionally, we included a well regarded high-performance visualization prototype, MegaMol for comparison [45]. Despite MegaMol being an atomic-level visualization prototype, this powerful visualization tool is particle-based. Since particle-based simulation engine offer frameworks that can be adapted to visualize cellular-level data, MegaMol is well-suited to serve as our visualization performance comparison base. Our VF ABM is able to process data orders of magnitude larger than FLAME GPU at a similar frame rate. Furthermore, we are
able to visualize 17x more data points than MegaMol at a 4.2x better frame rate. It is worth noting that MegaMol does perform more sophisticated rendering process than our framework. However, while our framework couples the visualization engine with grid-based real-time data generation, MagaMol did not support this feature [45].

### 7. Conclusion

We presented a scheduling technique for high-performance 3D ABM for wound healing application using GPU hyper tasking (GHT) to achieve high level of simulation interactivity. To achieve optimal concurrency in this scheduling scheme, different tasks were assigned to different devices. This task assignment resulted in output data being distributed across the CPU host and multiple GPUs. Thus, a Host-Device Activity-Aware Data Copy (HADC) technique was proposed to minimize host-device data copies. The resulting framework was used to implement a VF tissue repair response to injuries. The in situ visualization of the ECM and signaling proteins in healing VF was capable of processing and rendering 1.7 billion data points, at an average of 42.8 fps frame rate. Our ABM framework offers biomedical researchers a tool to efficiently explore the large amounts of output data with a high level of user-simulation interactivity.

We are currently exploring verification techniques to quantitatively compare model-generated ECM images to fluorescence microscopy images obtained from real tissue samples. Further, we plan to develop optimization techniques for visualization of inflammatory cells. As cell populations exhibit both structural and mobility properties, capturing their dynamics efficiently through visualization is a challenging task. The goal would be to couple cell visualization with the current framework, while maintaining a high level of interactivity.

## Acknowledgment

The work is supported by National Institute of Deafness and other Communication Disorder of the National Institutes of Health under Grant No R01DC005788, Natural Sciences and Engineering Research Council of Canada under Grant No. RGPIN-2018-03843 and the Canadian Institutes of Health Research under Grant No. 388583. The authors gratefully acknowledge the support provided by National Science Foundation under Grant No. CNS-1429404 MRI Project.

The authors would like to thank Sujaal Bista for guidance in developing the visualization component and UMIACS staff for assistance in VirtualGL configuration.

## References


