A voxel-based electrostatic field analysis for the virtual-human model Duke using the indirect boundary element method with a GPU-accelerated fast multipole method

S. Hamada
Department of Electrical Engineering, Kyoto University, Japan

Abstract

The voxel-based indirect boundary element method (IBEM) using the Laplace-kernel fast multipole method (FMM) is capable of analysing relatively large-scale problems. Furthermore, the voxel-based IBEM is suitable for acceleration using graphics processing units (GPUs). A typical application of the IBEM is the analysis of electrostatic fields for virtual-human anatomical voxel models such as the model called Duke provided by the IT'IS Foundation. An important property of voxel-version Duke models is that they have different voxel sizes but the same structural feature. This property is useful for examining the O(N) and O(D^2) dependencies of the calculation times and the amount of memory required by the FMM-IBEM, where N and D are the number of boundary elements and the reciprocal of the voxel side length, respectively. In this study, the O(N) and O(D^2) dependencies of the voxel-based GPU-accelerated FMM-IBEM were confirmed by analysing Duke models with voxel side lengths of 5.0, 2.0, 1.0, and 0.5 mm. The finest model comprised 2.2 billion voxels with 61 million square boundary elements, and a linear equation solver on a personal computer with four GPUs required 1,276 s to obtain a solution. In addition, a technique is proposed to improve the convergence performance of the linear equation solver by considering the non-uniqueness of the electric potential, and its effectiveness is demonstrated.

Keywords: voxel-based analysis, electrostatic field, indirect boundary element method, virtual-human model, fast multipole method, graphics processing unit, non-uniqueness of solutions.

WIT Transactions on Modelling and Simulation, Vol 57, © 2014 WIT Press
www.witpress.com, ISSN 1743-355X (on-line)
doi:10.2495/BE370121

1 Introduction

Various types of electromagnetic field analyses based on voxel models have been employed widely with the finite difference method, finite element method, etc. (e.g. Dawson *et al.* [1, 2], Hatada *et al.* [3]). The main advantages of voxel-based analysis include the facile production of realistic models from three-dimensional image data and the simple data structure, which is suitable for storage, handling, and visualisation. These advantages facilitate the analysis of relatively large-scale realistic models, particularly anatomical human models produced from magnetic resonance images.

Voxel-based boundary element methods (BEMs) for analysing relatively small-scale problems have been reported frequently. However, a voxel-based indirect BEM (IBEM) was applied to relatively large-scale problems by Hamada and Kobayashi [4] using the Laplace-kernel fast multipole method (FMM) (Greengard and Rokhlin [5]). Furthermore, this IBEM was accelerated using graphics processing units (GPUs) (Hamada [6, 7]) with NVIDIA's compute unified device architecture (CUDA) (NVIDIA [8]), thereby obtaining high-performance computation, even with a personal computer. This IBEM was used to analyse the electrostatic fields in human voxel models that describe the conductivities of biological tissues.

The IT'IS Foundation provides an adult male model called Duke (Christ *et al.* [9]), which was originally a surface-based model, but voxel-version models are available for voxel-based analyses. An important property of voxel-version Duke models is that they have different voxel sizes but the same structural feature. This property is useful for examining the $O(N)$ and $O(D^2)$ dependencies of the calculation times and the amount of memory required by the FMM-IBEM, where N and D are the number of boundary elements and the reciprocal of the voxel-side length, respectively. Note that the $O(D^2)$ dependency appears when the full size of the model under analysis is fixed.

In this study, the $O(N)$ and $O(D^2)$ dependencies of the voxel-based GPU-accelerated FMM-IBEM were confirmed by measuring the calculation times and the amount of memory required to analyse Duke models with voxel-side lengths of 5.0, 2.0, 1.0, and 0.5 mm. A technique is also proposed for improving the convergence performance of the linear equation solver for the IBEM, and its effectiveness is demonstrated. The proposed technique considers both the non-uniqueness of the electric potential and the existence of isolated voxel clusters.

2 Voxel-based indirect BEM with GPU-accelerated FMM

2.1 Magnetically induced electrostatic field analysis for biological samples

The basic equations of magnetically induced low-frequency faint currents in biological tissue were reported previously, e.g. by Dawson *et al.* [1], where it is assumed that the displacement current and secondary magnetic field are negligibly small. After applying an external magnetic flux density $\boldsymbol{B}_0$ and a vector potential

$\boldsymbol{A}_0$ which satisfy $\boldsymbol{B}_0 = \nabla \times \boldsymbol{A}_0$, the induced electrostatic field $\boldsymbol{E}$, the current density $\boldsymbol{J}$, and the scalar potential ϕ satisfy the following equations:

$$\boldsymbol{E} = -\mathrm{j}\omega\boldsymbol{A}_0 - \nabla\phi\,,\ \boldsymbol{J} = \sigma\boldsymbol{E}\,,\ \nabla^2\phi = 0\,, \tag{1}$$

where j, ω, and σ are the imaginary unit, angular frequency, and conductivity, respectively. The electric fields on both sides of a boundary surface with a unit normal vector $\boldsymbol{n}$ satisfy the following boundary equation:

$$\sigma_+\boldsymbol{E}_+ \bullet \boldsymbol{n} = \sigma_-\boldsymbol{E}_- \bullet \boldsymbol{n}\,. \tag{2}$$

The subscripts $\pm$ indicate the plus or minus side of the boundary with respect to $\boldsymbol{n}$. An indirect BEM is used to analyse the Laplace equation in eqn (1) by solving the simultaneous linear equations expressed in eqn (2).

2.2 Voxel-based IBEM

The voxel-based IBEM starts by converting a cubic-voxel model into an equivalent boundary element model by considering a square boundary sandwiched between two voxels with different conductivities as a square boundary element. The N pieces of the boundary elements are assumed to have uniform surface charge densities x_i (i = 1 to N), which numerically simulate $\nabla\phi$ in eqn (1). The surface integral of $\boldsymbol{E}_\pm \bullet \boldsymbol{n}$ on the jth element is as follows:

$$\int_{S_j} \boldsymbol{E}_\pm \bullet \boldsymbol{n}\mathrm{d}S = -\int_{S_j} \mathrm{j}\omega\boldsymbol{A}_0 \bullet \boldsymbol{n}\mathrm{d}S + \sum_{i=1,i\neq j}^{N} \int_{S_j} \left\{ \int_{S_i} \frac{x_i(\boldsymbol{r}_j - \boldsymbol{r}_i)}{4\pi\varepsilon_0 \left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|^3} \mathrm{d}S \right\} \bullet \boldsymbol{n}\mathrm{d}S \pm \frac{x_j}{2\varepsilon_0} S_j\,, \tag{3}$$

where $\boldsymbol{r}_i$ is the position vector on the ith element. Each boundary equation is formulated as a surface integral of eqn (2) on each square boundary element:

$$\sigma_+ \int_{S_j} \boldsymbol{E}_+ \bullet \boldsymbol{n}\mathrm{d}S = \sigma_- \int_{S_j} \boldsymbol{E}_- \bullet \boldsymbol{n}\mathrm{d}S\,. \tag{4}$$

The simultaneous linear equations $C\boldsymbol{x} = \boldsymbol{b}$ are composed of eqn (4) into which eqn (3) is substituted, where C is an $N \times N$ coefficient matrix, $\boldsymbol{x}$ is an $N \times 1$ unknown vector, and $\boldsymbol{b}$ is an $N \times 1$ constant vector. The surface charge densities $\boldsymbol{x}$ are determined by numerically solving $C\boldsymbol{x} = \boldsymbol{b}$. The fields of $\boldsymbol{E}$ and $\boldsymbol{J}$ are calculated for all the voxel centres on the basis of the value of $\boldsymbol{x}$ determined using an integral equation which is similar to eqn (3) in the post-processing stage.

2.3 Technique for obtaining a unique solution without numerical deviations

Eqn (4) corresponds to the Neumann boundary condition; thus, the unknown charge densities $\boldsymbol{x}$ have non-uniqueness due to the non-uniqueness of the electric potential. Some techniques are required to obtain a unique solution. First, the ith rows of C and $\boldsymbol{b}$ are divided by the diagonal component c_{ii} of C to scale c_{ii} unity. Second, $C\boldsymbol{x} = \boldsymbol{b}$ is separated into two parts, i.e. an arbitrary kth row and the other rows. The kth row can be regenerated by the others with elementary row operations because of the linear dependence due to the non-uniqueness. Third, the kth row is replaced by the following equation, which states that the total polarisation charge is physically zero:

$$(1 \cdots 1)\boldsymbol{x} = 0\,, \tag{5}$$

where $(1 \ldots 1)$ is a $1 \times N$ constant vector. The simultaneous equations obtained yield a unique mathematical solution (Hamada and Kobayashi [4]). However, the numerical solution of the kth unknown, x_k, tends to exhibit a measurable deviation because of the diagonally non-dominant nature of eqn (5). Fourth, we transform the simultaneous equations continuously into the following form: $C\boldsymbol{c} + \beta U_x/\mathrm{N} = \mathbf{b}$, where β is an arbitrary scalar constant which is set to be approximately one, and U is an $N\times$ constant matrix, where all entries are one. Every row of $\beta U_x/N$ represents β times the average of $\boldsymbol{x}$. Finally, by rescaling the diagonal components to unity, the following simultaneous equations are obtained:

$$\frac{N}{N+\beta} C\boldsymbol{x} + \frac{\beta}{N+\beta} U\boldsymbol{x} = \frac{N}{N+\beta}\boldsymbol{b}\,. \tag{6}$$

Eqn (6) provides a unique numerical solution without the deviation of x_k, and it can be implemented easily in iterative solvers for simultaneous linear systems. However, a voxel model sometimes comprises a number of clusters of voxels which are isolated by the air region, which has a conductivity of zero. In these cases, eqn (6) is modified on the basis of the fact that the total charge in each isolated cluster is zero. If two clusters consisting of N_1 and N_2 surface elements are considered, eqn (6) is modified to the following form:

$$\begin{pmatrix} C_{11}\dfrac{N_1}{N_1+\beta} & C_{12}\dfrac{N_1}{N_1+\beta} \\ C_{21}\dfrac{N_2}{N_2+\beta} & C_{22}\dfrac{N_2}{N_2+\beta} \end{pmatrix}\boldsymbol{x} + \begin{pmatrix} U_{11}\dfrac{\beta}{N_1+\beta} & 0 \\ 0 & U_{22}\dfrac{\beta}{N_2+\beta} \end{pmatrix}\boldsymbol{x} = \begin{pmatrix} \boldsymbol{b}_1\dfrac{N_1}{N_1+\beta} \\ \boldsymbol{b}_2\dfrac{N_2}{N_2+\beta} \end{pmatrix}, \tag{7}$$

where the sizes of C_{ij} and $\boldsymbol{b}_i$ are $N_i \times N_j$ and $N_i \times 1$, respectively. Note that if only closed-region problems are analysed, the field in each cluster can also be solved with eqn (6) simply by considering the voxels in the cluster. However, eqn (7) is available without any such restriction.

2.4 Voxel-based IBEM with GPU-accelerated FMM

The FMM (Greengard and Rokhlin [5]) is one of the best-known algorithms used to accelerate iterative solvers for simultaneous linear equations during BEM analysis. The Laplace-kernel FMM calculates the Coulombic interactions among the elements by separating the calculation process into two parts: (i) the direct interaction calculation of nearby elements and (ii) the non-straightforward interaction calculation with multipole- and local-expansion coefficients, which are referred to as the 'near-part' and 'far-part' calculations, respectively, in the present study. The translation algorithm used to convert the multipole-expansion coefficients into local-expansion coefficients in this study was a diagonal-form translation algorithm (Greengard and Rokhlin [5]). It is easy to apply FMM to the voxel-based BEM by considering a cubic-shaped cluster of voxels, e.g. $6 \times 6 \times 6 = 6^3$ cubic voxels, as a leaf cell defined in the FMM algorithm.

The GPU-accelerated FMM algorithm has been employed in many previous studies (e.g. Gumerov and Duraiswami [10] and Yokota *et al.* [11]). Details of GPU acceleration for the voxel-based FMM-IBEM were reported by Hamada [6, 7]. The basic points used to estimate the performance of the GPU-accelerated FMM with CUDA (NVIDIA [8]) are summarised as follows.

The GPU operations are classified into the following three categories: (i) using the GPU register and GPU shared memory, (ii) using the memories in (i) plus the GPU device memory, and (iii) using the memories in (ii) plus the CPU memory. The times required for these operations satisfy the following relationship: (i) << (ii) << (iii).

In the present study, CUDA kernels execute the following processes, almost in series: (a) transferring source data from the device memory to the register or shared memory using operation (ii), (b) collecting source data and producing target data with operation (i), and (c) transferring target data from the register or shared memory to the device memory using operation (ii).

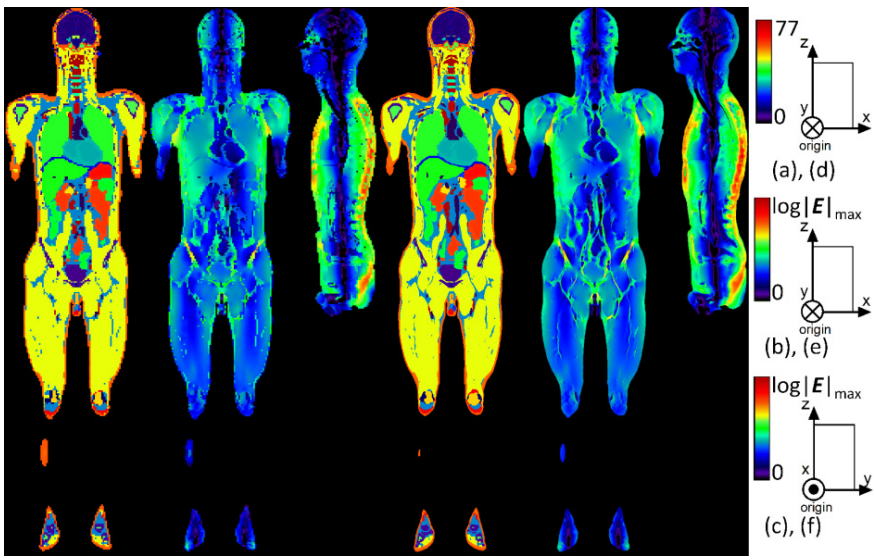
The data for the elements and expansion coefficients are accessed via the cell number. Thus, the times required for processes (a), (b), and (c) are affected by the number of cells (or leaf cells) and by the amount of data in a cell, which vary according to the parameter settings. The time required for (b) is affected by the number of numerical operations, which depends on the contents of the collection and production processes, and the performance of the multiprocessor.

2.5 Complexity of a voxel-based IBEM with a GPU-accelerated FMM

The number of numerical operations and the amount of memory required by the Laplace-kernel FMM-IBEM are expected to have $O(N)$ dependencies. During voxel-based analyses, N roughly exhibits an $O(D^2)$ dependency, where D is the reciprocal of the voxel side length when the full size of the model being analysed is fixed. The $O(D^2)$ dependency of the boundary elements contrasts with the $O(D^3)$ dependency of the number of volume elements, e.g. voxels. The CPU calculation time is almost proportional to the number of numerical operations. The GPU calculation time is roughly proportional to the number of numerical operations because it is affected by the time required for operations (i) and (ii).

3 Virtual-human model Duke

The IT'IS Foundation (www.itis.ethz.ch/vip) provides a series of anatomical virtual human models including an adult male model called Duke (Christ *et al.* [9]) which measures 1.77 m in height and consists of 77 tissue types. It was originally defined as a surface model, and we are contractually permitted to use its derivational voxel-version models in general program codes. Ready-made voxel-version models are provided in a distribution DVD with voxel-side lengths of 5.0, 2.0, 1.0, and 0.5 mm. The specifications of the models are summarised in Table 1. The number of voxels shows an O(D^3) dependency and the maximum number of total voxels exceeds 2 billion. The number of boundary elements, N, roughly exhibits an O(D^2) dependency, which is approximately O($D^{2.1}$), and the maximum value of N reaches 61 million. The number of fragment clusters fluctuates from zero to nine. The isotropic tissue conductivities are set mainly by referring to Hirata *et al.* [12]. Figure 1 shows the tissue numbers and calculated values of $|\boldsymbol{E}|$ for models with 5.0 and 0.5 mm voxel side lengths. In the present study, four voxel-version Duke models were used as a series of models with the same structural feature to examine the O(N) and O(D^2) dependencies of the calculation time and the amount of memory required by the voxel-based FMM-IBEM.



(a) 5 mm, tissue no. (b) 5 mm, log|$\boldsymbol{E}$|. (c) 5 mm, log|$\boldsymbol{E}$|. (d) 0.5 mm, tissue no. (e) 0.5 mm, log|$\boldsymbol{E}$|. (f) 0.5 mm, log|$\boldsymbol{E}$|.

Figure 1: Tissue number and calculated values of |E| in the Duke models.

Table 1: The numbers of voxels, boundary elements and isolated clusters.

Voxel-side length	5.0 mm	2.0 mm	1.0 mm	0.5 mm
Voxels in the x direction	112	272	544	1084
Voxels in the y direction	58	143	285	566
Voxels in the z direction	362	904	1806	3610
Total voxels	2,351,552	35,161,984	280,002,240	2,214,893,840
Tissue voxels	548,164	8,567,668	68,549,358	548,386,439
Boundary elements, N	472,284	3,617,128	15,070,962	61,492,477
Clusters: main + fragments	1 + 3	1 + 0	1 + 1	1 + 9

4 Computing environment and settings used for calculations

A personal computer running 64-bit Microsoft Windows 7 was used to perform the calculations. The computer had an Intel Core i7-4960X CPU (six CPU cores, 3.6 GHz), 64 GiB of RAM, and four NVIDIA GTX TITAN GPUs, where a GPU has 6 GiB of GDDR5 device memory, 14 multiprocessors (2,688 CUDA cores), and 48 kiB of shared memory per multiprocessor. Each GPU was used in the DP mode on a PCI Express 3.0 bus. The developed programs were compiled by the CUDA 5.5 NVCC compiler and Intel Visual FORTRAN Composer XE 2013 in Microsoft Visual Studio 2012. The number of OpenMP threads was set up to 12. The applied 50-Hz AC homogeneous magnetic field $\boldsymbol{B}_0 = B_0\boldsymbol{k}$ had strength of 1 μT. The vector potential was defined as $\boldsymbol{A}_0 = 0.5B_0(-y\boldsymbol{i} + x\boldsymbol{j})$, where $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ are the unit vectors parallel to the x, y, and z axes, respectively.

The order of the multipole and local expansions was set to 10 in the FMM algorithm. The diagonal-form translation algorithm (Greengard and Rokhlin [5]) was adopted. The voxel-cluster sizes used to define the leaf-cell size were 5^3, 6^3, 7^3, 8^3, or 9^3. Bi_IDR(s) (Onoue *et al.* [13]) was used as the iterative solver with s = 3. This solver requires one matrix–vector-product calculation per iteration step. In eqns (6) and (7), β = 1. Convergence occurred when the relative residual Euclidean norm was less than 10^{-6}, but this criterion value was set to 10^{-12} in subsection 5.1.

5 Results

5.1 Improving the convergence performance of the linear equation solver

The convergence performance of the linear equation solver was investigated to confirm the validity of eqns (6) and (7). The voxel side length, the number of fragment clusters, and the leaf-cell size were 5.0 mm, 3, and 7^3 voxels, respectively. Figures 2 (a), 2 (b), and 2 (c) show the results obtained using no measures, eqn (6), and eqn (7), respectively. The horizontal axis shows the number of iteration steps. The vertical axis shows the relative residual norm and the sum of x_i ($\sum_{i=1}^{N} x_i$) over the standard deviation of x_i, which is an index of the deviation of the total charges from zero. In the first case in Figure 2 (a), the index of deviation increased rapidly around the 140th step and remained almost constant at approximately

10-1. The relative residual decreased gradually with several stagnant periods. The increase in the index tended to decelerate the convergence as well as degrade the quality of the solution. Finally, the calculation was terminated at the 496th step owing to an execution error. In the second case in Figure 2 (b), the relative residual stagnated at approximately 10^{-8}, although the index of deviation remained at small values of approximately 2×10^{-8}. This stagnation was due to the lack of constraints on the zero total charge in an individual cluster. Finally, the calculation was terminated at the 490th step. In the third case in Figure 2 (c), the relative residual decreased almost uniformly to less than 10^{-12}, and the index of deviation remained at small values of approximately 2×10^{-8}. These results demonstrated that eqn (7) is effective in improving the convergence performance when a number of isolated voxel clusters exist. This implies that the use of eqn (7) ensures the quality of the solution even with a relatively high value for the convergence criterion.

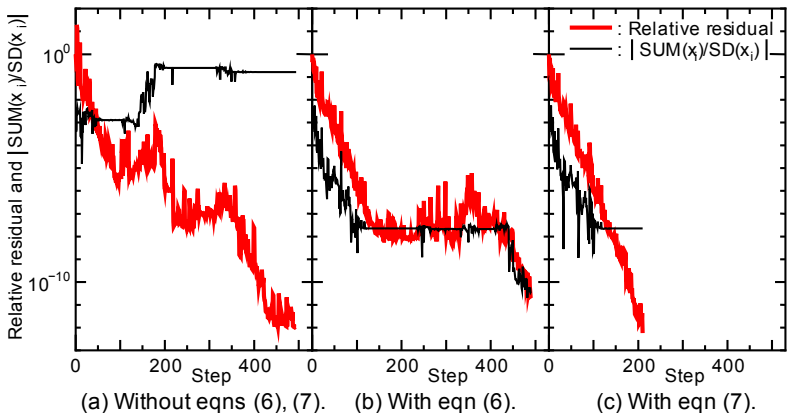
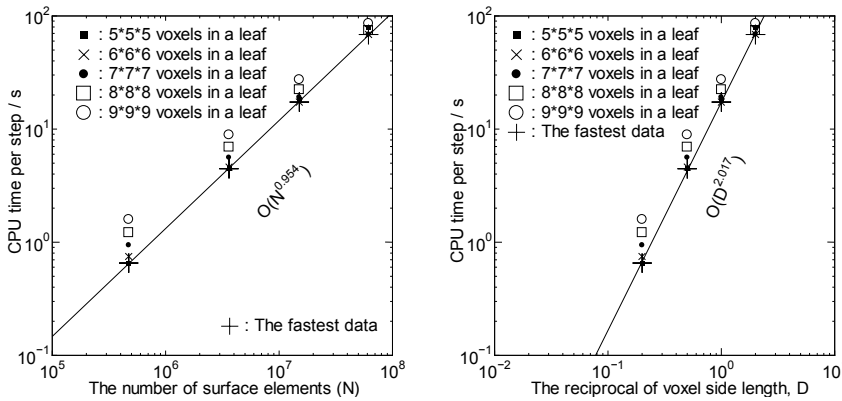


Figure 2: Convergence performance with eqns (6) and (7).

5.2 Calculation times with CPU code

The CPU calculation times required by the linear equation solver to analyse the four Duke models were measured, as shown in Figure 3. Table 2 lists the calculation times and iteration steps required when the fastest one-step time or the fastest total time was observed for each model, which are highlighted in boldface type. The number of iteration steps required was almost constant and ranged from 95 to 118. Figures 3 (a) and 3 (b) show the dependencies of the one-step calculation time on N and D, respectively. Figure 3 (a) shows the least-squares line of the fastest data for each N. The slope shows that the dependency is $O(N^{0.954})$. In a similar manner, Figure 3 (b) shows that the dependency is $O(D^{2.017})$. In addition, the dependencies of the near- and far-part calculation times are $O(N^{0.970})$ and $O(N^{0.935})$, respectively, based on the fastest parameter settings in Figure 3. This confirms that the voxel-based IBEM approximately exhibits $O(N)$ and $O(D^2)$ dependencies on the CPU calculation time, as expected.



(a) One-step time versus N. (b) One-step time versus D.

Figure 3: Dependencies of the one-step CPU calculation time on N and D.

Table 2: Calculation times with CPU code.

Voxel-side length	Voxels in a leaf	Time for one step / s	Time for near-part / s	Time for far-part / s	Iteration steps	Total time for solver / s
5.0 mm	5^3	**0.6546**	0.3518	0.2945	103	**67.42**
2.0 mm	5^3	**4.454**	2.126	2.266	105	467.6
2.0 mm	6^3	4.614	3.060	1.496	99	**456.8**
1.0 mm	6^3	**17.34**	9.465	7.627	110	**1,906.9**
0.5 mm	7^3	**68.52**	38.80	28.78	118	8,084.7
0.5 mm	8^3	74.53	51.45	22.17	95	**7,080.0**

5.3 Calculation times with GPU code

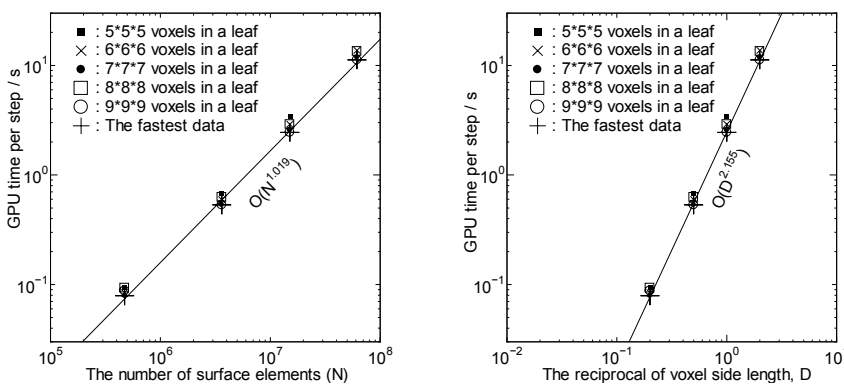
The GPU calculation times required by the linear equation solver were measured, as presented in Figure 4 and Tables 3 and 4. Figures 4 (a) and 4 (b) show that the dependencies of the one-step calculation time on N and D are $O(N^{1.019})$ and $O(D^{2.155})$, respectively. This confirms that the GPU-accelerated voxel-based IBEM approximately exhibits $O(N)$ and $O(D^2)$ dependencies. However, the dependencies were degraded slightly compared with those using CPU code. Table 3 summarises the calculation times and speed-up ratios compared with the CPU times. The dependencies of the near- and far-part calculation times are $O(N^{1.114})$ and $O(N^{0.904})$, respectively, with the fastest settings shown in Figure 4.

The far-part speed-up ratios in Table 3 were all low, ranging from 3.22 to 4.90. This was because the number of far-part calculations (process (b) with operation (i), see subsection 2.4) was of the same order as the number of device-memory read/write (process (a) and (c) with operation (ii)) operations due to the diagonal-

form translation algorithm. Thus, the calculation speed was strongly restricted by the latter. In contrast, the near-part speed-up ratios in Table 3 range widely from 6.45 to 28.6. This was because the number of near-part calculations (process (b) with operation (i)) was much higher than the number of device-memory read/write operations due to the repeated use of the shared memory data by the interaction calculation. The number of near-part calculations depends strongly on the number of elements in a leaf cell, which can be estimated roughly by N/L, where L is the number of leaf cells in Table 4. Figure 5 shows the dependency of the near-part speed-up ratios on N/L.

The one-step time speed-up ratios were approximately 11 for the 5.0, 2.0, and 1.0 mm models, whereas it was 7.64 for the 0.5 mm model. Table 3 suggests that a better speed-up ratio of approximately 11 might be obtained if a leaf cell is defined by 10^3 voxels. However, this condition could not be executed because the shared memory usage exceeded 48 kiB, which was inferred on the basis of the shared memory usage in Table 4. On the basis of a practical definition of the speed-up ratio as the fastest CPU time to the fastest GPU time for each model, the one-step time speed-up ratios for the 5.0, 2.0, 1.0, and 0.5 mm models were $\times 8.28$, $\times 8.33$, $\times 7.07$, and $\times 6.09$, respectively.

Table 4 also lists the number of iteration steps required and the total calculation time. The former was almost constant, as is the case for the CPU calculation. The solver required 1,276 s for the finest model analysis.



(a) One-step time versus N. (b) One-step time versus D.

Figure 4: Dependencies of the one-step GPU calculation time on N and D.

Table 3: Calculation times and speed-up ratios with the GPU code.

Voxel-side length	Voxels in a leaf	One step		Near-part		Far-part	
		Time / s	Speed-up	Time / s	Speed-up	Time / s	Speed-up
5.0 mm	5^3	0.09418	× 6.95	0.01768	× 19.9	0.07004	× 4.21
	6^3	0.08424	× 8.87	0.02768	× 19.6	0.05055	× 3.92
	7^3	**0.07906**	× **11.9**	0.03194	× 24.7	0.04080	× 3.49
	8^3	0.09308	× 13.1	0.05359	× 20.5	0.03350	× 3.29
	9^3	0.08791	× 18.1	0.05195	× 28.6	0.02968	× 3.22
2.0 mm	5^3	0.6842	× 6.51	0.1439	× 14.8	0.4906	× 4.62
	6^3	0.5958	× 7.74	0.2125	× 14.4	0.3339	× 4.48
	7^3	**0.5346**	× **10.5**	0.2417	× 18.4	0.2434	× 4.57
	8^3	0.6266	× 11.1	0.3877	× 15.5	0.1899	× 4.53
	9^3	0.5358	× 16.6	0.3365	× 24.3	0.1492	× 4.56
1.0 mm	5^3	3.414	× 5.41	0.6103	× 11.3	2.555	× 4.43
	6^3	2.898	× 5.98	0.9768	× 9.69	1.717	× 4.44
	7^3	2.565	× 7.46	1.140	× 11.6	1.221	× 4.64
	8^3	2.893	× 7.72	1.784	× 9.97	0.906	× 4.76
	9^3	**2.454**	× **11.2**	1.535	× 15.5	0.716	× 4.80
0.5 mm	6^3	13.70	× 5.01	4.156	× 6.87	8.723	× 4.48
	7^3	11.84	× 5.68	4.867	× 7.97	6.162	× 4.67
	8^3	13.43	× 5.55	7.977	× 6.45	4.637	× 4.57
	9^3	**11.25**	× **7.64**	6.791	× 9.89	3.643	× 4.90

Table 4: Calculation times and GPU memory usage with the GPU code.

Voxel-side length	Voxels in a leaf	Iteration steps	Total time for solver / s	The number of leaves, L	Shared memory for near-part calculation / bytes	Device memory per card / 10^6 bytes
5.0 mm	5^3	103	9.701	6,378	7,210	55.5
	6^3	97	8.171	3,985	12,936	44.5
	7^3	99	**7.827**	2,680	21,134	39.1
	8^3	112	10.43	1,891	32,248	41.0
	9^3	98	8.615	1,435	46,722	44.9
2.0 mm	5^3	105	71.838	67,335	7,210	393.7
	6^3	99	58.988	42,591	12,936	281.0
	7^3	116	62.008	28,884	21,134	215.5
	8^3	102	63.916	20,515	32,248	178.8
	9^3	102	**54.654**	15,119	46,722	154.8
1.0 mm	5^3	104	355.02	379,477	7,210	1,843.3
	6^3	109	315.85	243,518	12,936	1,298.9
	7^3	107	274.44	167,226	21,134	960.6
	8^3	104	300.88	119,952	32,248	760.0
	9^3	109	**267.48**	89,468	46,722	617.2
0.5 mm	6^3	110	1,506.64	1,306,251	12,936	6,055.9
	7^3	119	1,408.59	910,560	21,134	4,443.8
	8^3	95	**1,276.10**	662,703	32,248	3,476.6
	9^3	117	1,315.69	499,789	46,722	2,769.5

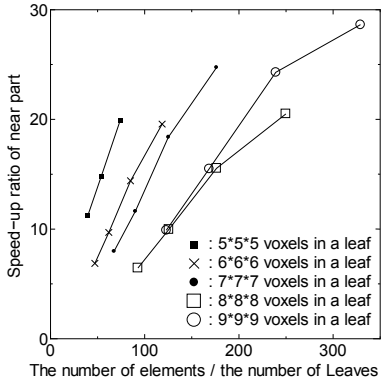


Figure 5: Near-part speed-up vs. N/L.

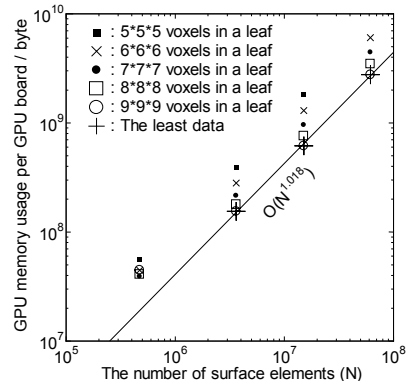


Figure 6: Device memory usage vs. N.

5.4 Device memory usage with the GPU code

The GPU device memory usage per GPU card was measured and is presented in Table 4 and Figure 6. The finest model analysis required approximately 3 GiB with the fastest parameter settings. Figure 6 shows the least-squares line of the data that required the least memory for each model. The data obtained from the 5-mm voxel model were excluded from the least-squares because they included measurable constant components, which would have made the slope gentler. The slope exhibits an $O(N^{1.018})$ dependency. This confirms that the IBEM approximately exhibits an $O(N)$ dependency on the device memory, as expected.

6 Summary

A voxel-based GPU-accelerated FMM-IBEM was used to analyse magnetically induced electrostatic fields in a virtual-human model called Duke. An important property of the voxel-version Duke models is that they have different voxel sizes but the same structural feature. By exploiting this property, the $O(N)$ and $O(D^2)$ dependencies of the calculation times and the amount of device memory required for the IBEM were confirmed successfully. The linear equation solver required 1,276 s to analyse the finest model with 61 million boundary elements and 2.2 billion voxels using a personal computer with four GPUs. In addition, a technique was proposed to improve the convergence performance of the linear equation solver for the IBEM, and its effectiveness was demonstrated successfully.
This research was supported by JSPS KAKENHI 25390153.

References

[1] Dawson, T.W., Caputa, K. & Stuchly, M.A., Influence of human model resolution on computed currents induced in organs by 60-Hz magnetic fields. *Bioelectromagnetics*, **18(7)**, pp. 478-490, 1997.

[2] Dawson, T.W., Caputa, K. & Stuchly, M.A., High-resolution organ dosimetry for human exposure to low-frequency electric fields. *IEEE Transactions on Power Delivery*, **13(2)**, pp. 366-373, 1998.
[3] Hatada, T., Sekino, M. & Ueno, S., Finite element method-based calculation of the theoretical limit of sensitivity for detecting weak magnetic fields in the human brain using magnetic-resonance imaging. *Journal of Applied Physics*, **97**, 2005.
[4] Hamada, S. & Kobayashi, T., Analysis of electric field induced by ELF magnetic field utilizing fast-multipole surface-charge simulation method for voxel data. *Electrical Engineering in Japan*, **165**, pp. 1-10, 2008.
[5] Greengard, L. & Rokhlin, V., A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, **6**, pp. 229-269, 1997.
[6] Hamada, S., GPU-accelerated indirect boundary element method for voxel model analyses with fast multipole method. *Computer Physics Communications*, **182**, pp. 1162-1168, 2011.
[7] Hamada, S., Performance comparison of three types of GPU-accelerated indirect boundary element method for voxel model analysis. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, **26**, pp. 337-354, 2013.
[8] NVIDIA, http://www.nvidia.com/page/home.html
[9] Christ, A. *et al*., The virtual family – Development of surface-based anatomical models of two adults and two children for dosimetric simulations. *Physics in Medicine and Biology*, **55 (2)**, pp. N23-N38, 2010.
[10] Gumerov, N.A. & Duraiswami, R., Fast multipole method on graphics processors, *Journal of Computational Physics*, **227**, pp. 8290-8313, 2008.
[11] Yokota, R., Bardhan, J.P., Knepley, M.G., Barba, L.A. & Hamada, T., Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. *Computer Physics Communications*, **182(6)**, pp. 1272-1283, 2011.
[12] Hirata, A. *et al*., Intercomparison of induced fields in Japanese male model for ELF magnetic field exposures: effect of different computational methods and codes. *Radiation Protection Dosimetry*, **138(3)**, pp. 237-244, 2010.
[13] Onoue, Y., Fujino, S. & Nakashima, N., An overview of a family of new iterative methods based on IDR theorem and its estimation. *Proc. Int. Multi-conf. of Engineers & Computer Scientists*, Hong Kong, pp. 2129-2134, 2009.