CUDA と OpenGL を用いたマルチ GPU による電磁界数値シミュレーションの高速可視化

河田　直樹[†a)]（学生員）　大久保　寛[†b)]（正員）
田川　憲男[†]（正員）　土屋　隆生[††]（正員）

Multi-GPU Numerical Simulation of Electromagnetic Field with High-Speed Visualization Using CUDA and OpenGL

Naoki KAWADA[†a)], *Student Member*, Kan OKUBO[†b)], Norio TAGAWA[†], and Takao TSUCHIYA[††], *Members*

[†] 首都大学東京システムデザイン学部，日野市
Faculty of System Design, Tokyo Metropolitan University, 6–6 Asahigaoka, Hino-shi, 191–0065 Japan
[††] 同志社大学理工学部，京田辺市
Doshisha University, 1–3 Tatara Miyakodani, Kyotanabe-shi, 612–0321 Japan
a) E-mail: kawada-naoki@sd.tmu.ac.jp
b) E-mail: kanne@sd.tmu.ac.jp

あらまし　本研究では，個人ユース型の計算機でマルチ GPU を用いた三次元電磁界数値解析のリアルタイム高速可視化について実現可能性を検討し，CUDA と OpenGL を利用することで，GPU による高速な計算と可視化が可能であることを示した．

キーワード　高速可視化，FDTD 電磁界数値解析，GPU，OpenGL，CUDA

1. まえがき

従来，計算の高速化という点では，スーパコンピュータやクラスタなどの大型の計算機が利用されているが，最近では GPU（Graphics Processing Unit）を用いて汎用的な数値計算を行おうとする GPGPU（General Purpose computation on GPUs）が様々な分野で注目され，盛んになってきている [1]．これまで，GPGPU の試みとして，流体計算や津波シミュレーションを先駆けとして [2]，電磁界や音場という時間領域の波動数値シミュレーションの GPU への実装が行われている [3]～[8]．

GPU の演算性能は数年前より飛躍的に向上しており，最新の GPU は少し前のスーパコンピュータ並の性能を秘めているとも考えられる．したがって，大型の計算機を使用しなくとも，パーソナルコンピュータ（パソコン）に GPU を接続することで高速計算を行うことができるのである [3]．

また従来，個人ユース型の計算機で可視化を含めた電磁界解析を行う場合，計算環境への負荷を考え，対称性を仮定した二次元モデルや一次元モデルを用いた解析が行われてきていた．特に，パソコンで解析する場合には，三次元空間をモデル化するためのメモリ容量の問題や，メモリが確保できたとしてもその計算を行うための計算時間を考慮すると，現実問題として三次元解析は困難な場合が多かった．更に，シミュレーションをしながら，同時に計算結果を可視化する，いわゆるリアルタイム可視化を行う場合は，十分な描画スピードを維持するためには，三次元解析はパソコンレベルではほぼ不可能であった．

しかし，近年の計算機技術の発展――特に，GPU による高速並列計算は，描画スピードの問題を一気に解決し，三次元の電磁界シミュレーションとそのリアルタイム可視化を現実のものとしようとしている [3], [5]．すなわち，個人ユース型の計算機でも高速な演算が可能となっており，GPU を用いたパーソナルスーパコンピュータが同時可視化シミュレーションを大きく後押しすることになる．

一方，GPU はもともとグラフィックス用のハードウェアであるため，数値計算結果の高速可視化という面でも，大きなアドバンテージを有していると考えられる．すなわち，数値解析中に GPU で画像出力を超高速で行うことで，高速リアルタイム可視化を可能とする．これは，個人使用においてタイムシェアリングが基本である大型スーパコンピュータでは，実現はかなり難しい．まさに，パーソナルスパコンを実現する GPU 計算環境に適しているといえるのではないだろうか．特に最近では，Fermi アーキテクチャが発表され，1GPU 当りのコア数が増え，キャッシュなども大幅に改善されたため，この分野への GPGPU 応用が広がると考える．

電磁界シミュレーションにおいて，この高速可視化をする上で，相性の良い手法としては FDTD 法 [9], [10] や GCIP（Generalized CIP）法 [5], [11]～[13] などの時間領域手法である．なぜならば，これらの手法は，離散化した電磁界空間を，時間領域で逐次更新していくため，更新結果を高速に表示することができれば，電磁界伝搬の様子を計算と同時にそのまま瞬時に把握することができるのである．更に，GPU 計算における高速化の効果は並列化効率に依存するため，並列化の効果が大きいこれらの手法は GPU 電磁界解析のリアルタイム可視化として実用に最も近い．

そこで本論文では，最近の GPU アーキテクチャ Fermi を搭載した GeForce GTX シリーズを用いて，これまで筆者らが報告してきている CUDA を用いた GPU 電磁界計算に描画用の API として OpenGL を組み合わせることで，GPU による電磁界数値解析と

リアルタイム可視化の可能性を検討する．また，複数の GPU を用いて GPGPU リアルタイム可視化のマルチ GPU 化を実装・評価する．更に，CUDA4.0 よりリリースされた GPU Direct 2.0 による GPU 間メモリ転送及び境界領域データの隠ぺい通信についても検討を行った．これにより，リアルタイム可視化を行う場合に，マルチ GPU 計算と OpenGL と CUDA の組み合わせにより，更に高速なリアルタイムシミュレーションが可能となることを示す．

2. GPU による時間領域電磁界シミュレーション

2.1 FDTD 法

本論文では，リアルタイムの高速可視化が目的であるため，比較的定式化が簡単で計算負荷の少ない Finite-Difference Time-Domain（FDTD）法を用いる．FDTD 法は Staggered grid を用いて支配方程式を直接差分化する手法である．離散化された空間では電界と磁界は 1/2 グリッド離れた点に配置される．最も基本的なスキームは，時間・空間二次精度の中心差分近似によって解く手法（Yee-FDTD 法）であり [9], [10]，リープフロッグアルゴリズムで各グリッドの電界と磁界を交互に 1/2 タイムステップごとに更新していく．

本手法は時間領域で解く手法であり，各グリッド上の値の更新は順序性がなく，場の値はそれぞれ独立して求めることができるため GPU 並列化に適した手法といえる．

2.2 CUDA による GPU プログラミングと OpenGL による GPGPU 可視化

近年，GPGPU 向けのプログラミング言語として CUDA（Compute Unified Device Architecture）などの C 言語に近い言語が開発されたことにより，C 言語の知識だけで比較的手軽にプログラミング可能となった [1], [16]．本研究で用いた GeForce GTX 580（GeForce GTX シリーズに属している）は 512 基のストリーミングプロセッサ（SP，CUDA Core とも呼ばれる）で構成され，高速アクセス可能な共有メモリ（SM）をもっている．

CUDA で GPU プログラミングを行う場合，高速化の重要なポイントはブロック内のスレッドモデルをどのように構成するかと，レジスタや共有メモリ（SM）を活用することである．すなわち，複数回グローバルメモリ（GM）にアクセスする必要がある場合には，参照するデータをいったん SM やレジスタへ転送し，メモリアクセスを極力低減させる必要がある．

続いて計算結果の可視化について述べる．従来の CPU 計算における可視化の方法としては，一般には，数値計算結果をディスプレイ上にリアルタイムで可視化しようとする場合，計算結果を色情報へ変換したのちにビデオカード上の描画用のビデオメモリ（VRAM）領域に転送する（図 1 (a) 参照）．このプロセスは，CPU を用いて電磁界数値計算をした場合，計算結果が格納されているメインメモリ（RAM）から VRAM への転送が必要であるため，表示させる結果によっては PCI インタフェースの転送が問題となる場合がある．

他方，CUDA で GPU 計算を実装する場合，OpenGL の関数を CUDA から直接呼び出すことができ，また，計算領域がビデオメモリ（VRAM）内に確保されているため，PCI インタフェースの転送をバイパスしてビデオカード上の描画用の VRAM 領域へ表示情報を書き込むことができる（図 1 (b) 参照）．これはリアルタイム可視化を目指す上で非常に大きな利点となり得る．CUDA による GPU 計算によって，計算速度自体が高速化させられるが，CUDA と OpenGL の連携によって更に描画におけるアドバンテージも受けることができるのである．

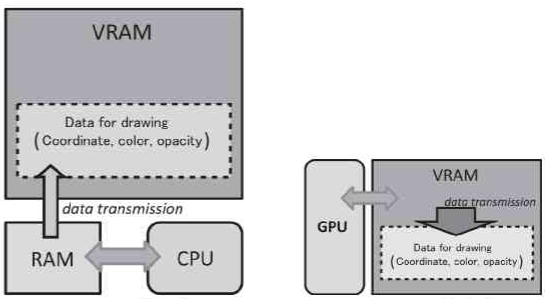
2.3 可視化方法（PMCC）

本節では，本研究で用いる可視化方法として PMCC（Permeable Multi Cross-section Contours）[14]（図 4 参照）の概要を示す．

この PMCC は一言でいえば，複数の表示断面に対して，表示する強度に合わせて不透過度を設定する，方法である．

描画の手順をまとめると，以下のようになる．

（1） 三次元空間のある断面に対して，電界値など



(a) CPU 使用時　(b) GPU 使用時

図 1 CPU/GPU 使用時の可視化におけるデータ転送概略図

Fig. 1 Schematic diagram of data transmission (a) CPU (b) GPU.

をカラー表示する．

（2） 更に，電界値の強度に合わせて不透過度（α 値）を設定する．（電界値が大きい場合に不透過度は 1 に近づく）

（3） 不透過度（α 値）の与え方は計算対象によって変動させることができるので，表示させる強度の下限や強度と α 値の関係式などを自由に設定できる．

（4） 同様の手順で表示させたい断面を複数描画する．

特徴としては，

- 複数の断面を表示しても，電界値が小さい点は不透過度が小さくなるため，オクルージョンが発生しにくく，ある断面の裏に隠れてしまっていた情報も見ることができる．
- したがって，平行な複数の断面も同時に表示可能である（図 4）．
- 断面の回転及び移動も可能である．しかも，GPU 実装では高速に動作し，待ち時間なく回転・移動することを確認している．
- 断面のみの不透過度を利用しているため，ボリュームレンダリングに比べて，描画のため転送データや計算負荷が極端に少ない．
- 断面表示をもとにしているため，複雑な波動伝搬現象も把握しやすい．

ということが挙げられる．アニメーションにより視点を変更することで，より三次元空間を把握することが容易になる [15]．

2.4 マルチ GPU を使用した GPGPU 可視化

図 1 (b) を見ると，シングル GPU における OpenGL を用いた可視化では単に GPU 上の計算領域から描画領域に描画用のデータを書き込むことでイメージングが可能であった．しかし，PCI バスで接続された二つ以上の GPU を用いて計算を行う場合（いわゆる単一ノード上のマルチ GPU 計算），計算したデータを描画するには，描画を担当する GPU に PCI バスを通して，描画用データを転送する必要がある．これは，マルチ GPU 計算での GPU 間の境界データの転送に加えて，描画のために必要な GPU 間のデータ転送作業となる（図 2）．

このマルチ GPU を用いた可視化の流れを簡単に説明すると，(1) 複数の GPU で領域を分割・担当し計算，(2) GPU 間の境界領域のデータを交換，(3) 描画用のデータを PCI バスを通して RAM に転送（ただし，描画担当の GPU は除く），(4) 描画用のデータを

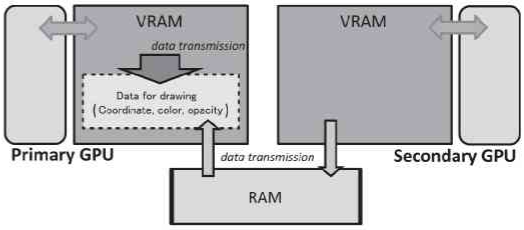


図 2 マルチ GPU 使用時の可視化
Fig. 2 Schematic diagram of data transmission on multi-GPU visualization.

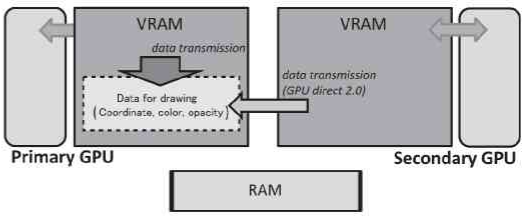


図 3 GPU Direct 2.0 を使用したマルチ GPU 使用時の可視化
Fig. 3 Schematic diagram of data transmission on multi-GPU visualization with GPU Direct 2.0.

描画担当の GPU の描画データ領域に転送（ただし，描画担当の GPU は PCI バスを介さずに直接 GPU 内で転送），(5) 再び複数の GPU で領域を分割・担当し計算，を繰り返す．このようにすることで，プライマリ GPU 以外からの PCI 転送のコストは必要とはなるが，マルチ GPU を用いたリアルタイム可視化が可能となる．

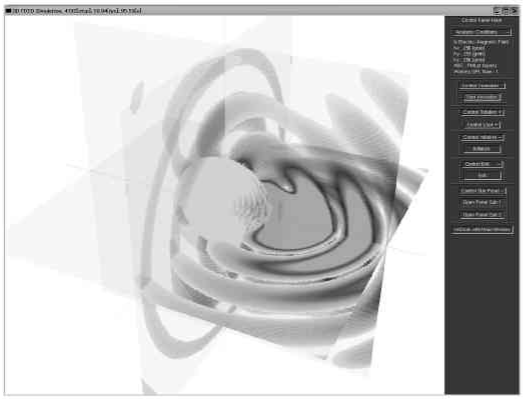
GPU 間で転送される描画用データは，各 GPU で電磁界成分から描画のための座標情報（XYZW）と色情報（RGBα）に変換したのち，そのデータを転送している．PMCC によって表示する断面の大きさや数にもよるが，数 MByte 程度のデータ量となる．

また，境界領域のデータの GPU 間転送については，非同期通信を利用し境界領域以外の計算中に転送を行うことで通信隠ぺいを行うことができる．一方，CUDA4.0 より GPU Direct 2.0 による GPU 間メモリ転送が新たにリリースされた．従来の GPU 間転送では，メインメモリ（RAM）を介する必要があったが，GPU Direct 2.0 では直接 GPU 間を転送するため，GPU 間のデータ転送時間を短くすることができる（図 3 参照）．

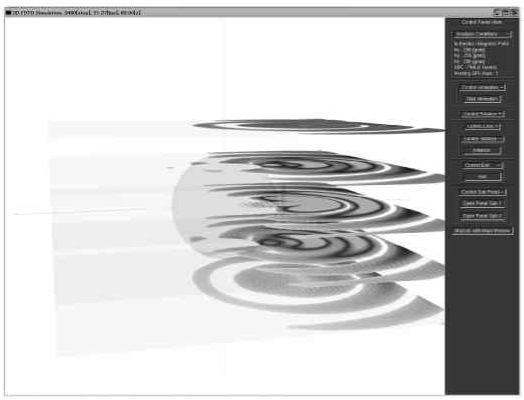
3. 数値計算結果

3.1 計算時間の比較評価

本節では，計算時間の比較評価を行う．電磁界計



(a) Right-angled contours with a dielectric sphere



(b) Inc. Parallel contours with a dielectric sphere

図 4 PMCC（Permeable Multi Cross-section Contours）による可視化
Fig. 4 GPU visualization using permeable multi cross-section contours. (PMCC)

算手法としては Yee-FDTD 法を用いている．使用した計算環境は OS：Windows 7 Pro x64 edition, CPU：Intel Core i7 930 2.80 GHz，メモリ：DDR3 6 GByte，コンパイラ：マイクロソフト C/C++ コンパイラ Ver.15.00 for x64，OpenMP：OpenMP 2.0 である．以下では単精度型を用いて計算を行っている．本計算では GPU として GeForce GTX 580 を使用している．表 1 に GPU の主要スペックを示す．

また，性能評価の指標として FLOPS（Floating point number Operations Per Second）と FUPS（Fields Update Per Second）を用いる．FLOPS は 1 秒間に浮動小数点演算が何回行われたか，FUPS は 1 秒間に計算領域内の何点の場の値が更新されたかを表す．電磁界解析の場合，計算速度としては FLOPS よりも FUPS を用いた方が適正な指標といえる．

表 1 GeForce GTX 580 のスペック一覧
Table 1 GeForce GTX 580 specification.

主要スペック	GTX 580
CUDA Core 数	512
プロセッサクロック	1544 MHz
メモリクロック	2004 MIIz
メモリインタフェース	384 bit
メモリサイズ	1536 MByte
メモリタイプ	GDDR5

表 2 に三次元計算領域のグリッド数を $256 \times 256 \times 256$，計算回数を 1024 と固定したときの FDTD 解析について，実行した場合の計算時間をそれぞれ示す．ただし計算には単精度型を用いている．また，CPU の結果も併せて示している．計算領域は立方体の領域として，吸収境界条件は除いた領域のみの評価を行っている．ここでは，1 thread i7 (CPU)，8 thread i7 (CPU)，GTX 580 (1GPU) の結果を示し，更に，GTX 580 を 2 個利用したマルチ GPU 計算の結果も示している．マルチ GPU 計算では，境界領域のデータの GPU 間転送について，非同期通信及び GPU Direct 2.0 を用いた場合と用いない場合の結果を表示している．ただし，どちらも用いない場合は GPU 間転送の同期通信コマンドを利用している．

同表より，以下のことが分かる．Yee-FDTD 法ではシングル GPU（GTX 580）で CPU（i7 8 スレッド並列）の約 17.8 倍の高速化が実現できている．一方，二つの GPU を利用した場合では約 30.4 倍の高速化が実現できている．三次元解析において，2.777 GFUPS はパソコンレベルでは非常に高速といえ，スーパコンピュータに迫る環境がマルチ GPU 計算によって実現できることが分かる．

3.2 描画スピードの比較評価

次に CUDA と OpenGL を組み合わせることで可視化について描画スピードの比較評価を行う．まず，描画スピード及び描画効率に関して，非同期通信（async.）と GPU Direct 2.0 の効果を評価する．

表 3 に async. と GPU Direct 2.0 を利用した場合の効果を描画速度（fps: frames per second）で示す．ただし計算領域を 256^3 グリッドとし，描画用データについて隠ぺいは行っていない．また，描画速度は 1024 ステップまでの描画にかかった時間より計算している．同表より両方を同時に用いることで描画効率が 20%程度上昇することが分かる．

次に計算領域（グリッド数）に対する描画速度の結果を図 5 に示す．図中は可視化法として PMCC [14]

表 2　Yee-FDTD 法の計算時間（単精度型，計算領域 256 × 256 × 256 cells，計算回数 1024 回）
Table 2　Calculation time of FDTD analysis. (single precision)

計算環境	計算時間	GFLOPS	GFUPS	Calc. efficiency
1 thread i7 (CPU)	660.5 s	1.014	0.0260	
8 thread i7 (CPU)	199.9 s	3.352	0.0859	
GTX 580 (1GPU)	9.688 s	69.15	1.773	
GTX 580 (2GPU) w/o async. and GPU Direct 2.0	5.613 s	119.3	3.060	72.60%
GTX 580 (2GPU) w/ async.	5.133 s	130.5	3.346	88.70%
GTX 580 (2GPU) w/ async. and GPU Direct 2.0	5.105 s	131.2	3.365	89.70%

表 3　Yee-FDTD 法の描画速度（fps: frames per second）（単精度型，計算領域 256 × 256 × 256 cells）
Table 3　Drawing speed of visualization of FDTD EM analysis. (single precision)

計算環境	描画速度（fps）	計算時間	Drawing efficiency
1 thread i7 (CPU)	0.53 fps	1922.31	
8 thread i7 (CPU)	2.08 fps	490.9	
GTX 580 (1GPU)	62.5 fps	16.36	
GTX 580 (2GPU) w/o async. and GPU Direct 2.0	100 fps	10.24	59.80%
GTX 580 (2GPU) w/ async.	104 fps	9.82	66.6%
GTX 580 (2GPU) w/ asyncánd GPU Direct 2.0	113 fps	9.04	81.0%

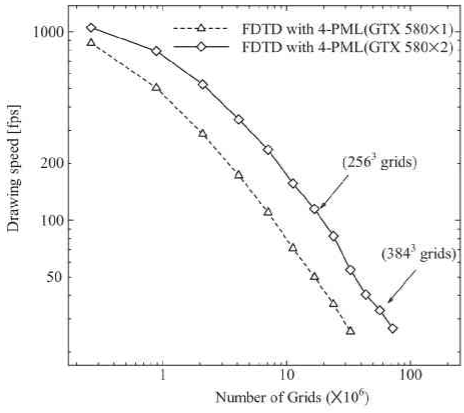


図 5　グリッド数に対する描画速度（FDTD，三次元電磁界解析）
Fig. 5　Drawing speed v.s. number of grids. (FDTD, three-dimensional electromagnetic field analysis)

を用いた場合の三次元シミュレーションの描画スピードを示している．解析モデルは図 4 で示したように計算領域の中心にダイポールアンテナ及び誘電体を配置したものである．同図では横軸がグリッド（セル）数，縦軸が描画速度（fps）としている．また吸収境界として，4 層 PML を適用した場合の結果を表示している．計算パラメータとしては，給電周波数 1.9 G[Hz]，誘電体球は比誘電率 43.37，導電率 1.204 S/m を想定している．

シングル GPU（GTX 580×1），デュアル GPU（GTX 580×2）を比較する．ただし解析にはマルチ GPU を用いるが，グラフィック出力はプライマリ GPU からのみ行われている（図 2 参照）．図 5 より計算領域が大きくなると，マルチ GPU の効果がより顕著になることが分かる．GPU をシングルからデュアルにすることで，約 1.8〜1.9 倍程度の高速化が可能となっており，可視化を行う場合も GPU のマルチ化が有効な手法であることが分かる．

この理由としては以下のように考えられる．本研究で用いた PMCC は透過性断面を複数表示することにより，擬似的 3D 結果を表現する手法であり，GPU 間の転送描画データは各断面数の分だけとなる．したがって，計算領域が大きくなったとしても，計算時間は一辺当りのグリッド分割数の 3 乗で増加するが，転送描画データは 2 乗で増加する．その結果，描画の相対的な負荷が低減した．

更に，図 5 を見ると，256^3 グリッドの場合，デュアル GPU（GTX 580×2）で GPU Direct 2.0 を用いることで 113 fps である．一方，本研究で用いた CPU：Intel Core i7 930 2.80 GHz にて OpenMP を使用し 8 スレッド並列化をした場合，同様の計算条件で PMCC で描画すると 2 fps 程度であり，GPU と CPU の可視化の性能差は歴然である．以上より，CUDA と

OpenGL によるマルチ GPU 計算を用いた高速可視化によって，従来の個人ユース型のアーキテクチャでは考えられないような性能が得られることが明らかとなった．

4. むすび

本研究では，Fermi シリーズ（GTX 580）のマルチ GPU を用いて三次元電磁界数値解析のリアルタイムの可視化（計算と同時に可視化する電磁界シミュレーション）について検討を行った．三次元電磁界の可視化法として，PMCC を用いて，その評価を行った．PMCC は従来の三次元空間中の複数断面を表示する（Multi Cross-section Contours）方法に不透過度を組み合わせたシンプルかつ高速な可視化方法である．

本研究の結果より，三次元電磁界数値解析の高速可視化において，マルチ GPU 計算と時間領域手法，そして OpenGL と PMCC を組み合わせ，更には GPU Direct 2.0 を利用することは，非常に有用な方法であることが明らかとなり，その実現可能性が示せたといえる．今後はこの高速可視化を用いた実用化を検討する予定である．

文　献

[1] http://www.nvidia.co.jp/object/cuda_home_jp.html

[2] 青木尊之，“フル GPU による CFD アプリケーション，” 情報処理，vol.50, no.2, pp.107–115, 2009.

[3] 河田直樹，大久保寛，田川憲男，土屋隆生，“音響数値計算のための GPU によるパーソナルスーパーコンピュータの実現へ向けて，” 音響春季講論集，2-2-6, March 2010.

[4] 河田直樹，大久保寛，土屋隆生，信学技報，AP2009-141, Nov. 2009.

[5] 河田直樹，大久保寛，土屋隆生，“GPU 計算を用いた時間領域電磁界数値解析の高速化性能に関する比較，” 信学論（B），vol.J94-B, no.3, pp.480–483, March 2011.

[6] 河田直樹，大久保寛，田川憲男，土屋隆生，“GCIP(l, m) 法を用いた音波伝搬シミュレーションと GPU 計算による高速化の検討，” 信学技報，AI2009-21, 2009.

[7] T. Tsuchiya and H. Sekoguchi, “An Evaluation Method of Calculation Performance for Acoustic Field Simulation Using a Graphic Processing Unit (GPU),” J. Japan Society for Simulation Technology, vol.27, no.4, pp.245–254, 2008.

[8] 土屋隆生，世古口久史，GPU による時間領域音場シミュレーションの高速化～DHM と FDTD の比較～，信学技報，US2007-89, Jan. 2008.

[9] K.S. Yee, “Numerical solution of initial boundary value problems involving Maxwell’s equations in isotropic media,” IEEE Trans. Antennas Propag., vol.AP-14, no.4, pp.302–307, May 1966.

[10] A. Taflove and S.C. Hagness, Computational Electromagnetics, Artech House London, 2005.

[11] K. Okubo and N. Takeuchi, “Analysis of an electromagnetic field created by line current using the CIP method,” IEEE Trans. Antennas Propag., vol.55, no.1, pp.111–119, Jan. 2007.

[12] T. Yabe, F. Xiao, and T. Utsumi, “The constrained interpolation profile method for multiphase analysis,” J. Comput. Phys., vol.169, pp.556–593, 2001.

[13] 大久保寛，竹内伸直，“CIP 法による線電流源から発生する電磁界の数値解析，” 信学技報，AP2004–336, March 2005.

[14] 河田直樹，大久保寛，田川憲男，土屋隆生，石塚　崇，“CUDA と OpenGL を用いた音響数値解析の高速可視化に関する検討～PMCC（Permeable Multi Cross-section Contours）の提案～，” AI2010-5-03, Feb. 2011.

[15] http://www.sd.tmu.ac.jp/klolab/gallery/3DV/PMCC/index.html

[16] 青木尊之，籔田　彰，はじめての CUDA プログラミング，工学社，2009

（平成 23 年 5 月 27 日受付，9 月 21 日再受付）