K-058

GPU を用いた仮想三次元物体の実時間自由変形
Realtime Free-Form Deformation of Virtual 3D Object Using GPU

伊津　惇 † 　高橋　章 † 　若月大輔 ‡ 　駒形英樹 § 　石井郁夫 §
Atsushi Izu　Akira Takahashi　Daisuke Wakatsuki　Hideki Komagata　Ikuo Ishii

1. まえがき

近年，CT や MRI などの画像診断技術で Computer Graphics（CG）が応用されている．CT や MRI で撮影した画像はボクセルと呼ばれる小立方体で物体を表現しているが，データサイズが膨大となるため，サーフェイスデータに変換することが多い．最近のグラフィックスハードウェアはサーフェイスデータの描画に特化した機能が実装されており，高速かつ高品質な描画を行うことができる．しかし，人体や臓器のような非剛体の自由変形処理は実現が難しい．

非剛体を取り扱うには，サーフェイスデータを構成するポリゴンの集合を，仮想空間中で自由に変形する必要がある．自由変形を行う標準的な方法として Free-Form Deformation（FFD，2.1 節）がある [1]．しかし，FFD では仮想三次元物体を構成するポリゴンのすべての頂点に対して膨大な計算を行うため，実時間処理が困難であった．

コンピュータの性能向上手段として，CPU の演算ユニットの並列化（マルチコア化）が進んでいる．CPU で並列処理を行う方法として OpenMP[2] がある．また，グラフィックスデバイスである GPU はプロセッサの並列化が進んでおり，メモリバンド幅も太いため，汎用計算のために利用するための環境整備が加速している．GPU 上で汎用計算を行うために NVIDIA 社は CUDA[3] を開発・公開している（3.2 節）．Modat ら [4] は，CT や MRI 画像の撮影時における非剛体の位置合わせを行うために，CUDA で FFD を行う方法を検討し，GPU 上の計算用に最適化した式を提案した．しかし，この方法では GPU のハードウェア構成を十分活用する高速化設定や，変形結果の描画処理までは検討されていない．CUDA では，並列処理のための分割数をユーザが定義する必要があるが，分割数と処理時間の関係や適切な分割数を設定するための指標は仕様書 [1] には示されていない．また，FFD による変形結果を描画するには，演算結果として GPU から CPU へ転送された頂点や法線などのデータを，描画データとして CPU から GPU へ再転送する必要があり，効率が悪い．

そこで本研究では，GPU 上で FFD による仮想三次元物体の自由変形と，変形結果の描画を，効率よく切り替えることで高速処理の実現を検討する．処理時間の比較対象として，OpenMP によるマルチコア CPU 上での並列処理と，CUDA による汎用計算として FFD を実装し，結果の描画処理までの最適化を行わない方法を用いる．そして，描画処理を考慮して最適化を行う提案法の有効性を示す．さらに，CUDA での処理の分割数と処理時間の関係についても調査を行い，最適な分割数の設定について検討する．

† 長岡工業高等専門学校
‡ 筑波技術大学
§ 埼玉医科大学

2. 3DCG における描画・変形処理

2.1 Free-Form Deformation

FFD[1] では，多数の頂点，ポリゴンから構成された物体に対し，少数の制御点を配置する（図 1）．制御点の移動に追従させて物体の自由変形を行う．

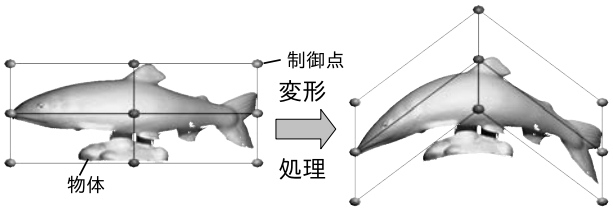

図 1　FFD による自由変形例

FFD による自由変形を行うにはまず，変形領域の座標系に仮想三次元物体を定義する．変形領域の原点を $\boldsymbol{X}_0$，基底ベクトルを $\boldsymbol{S}$，$\boldsymbol{T}$，$\boldsymbol{U}$ としたとき，N 個の頂点 $\boldsymbol{X}_a$，$a = 1 \sim N$ からなる形状データは次式で表される．

$$\boldsymbol{X}_a = \boldsymbol{X}_0 + s_a\boldsymbol{S} + t_a\boldsymbol{T} + u_a\boldsymbol{U} \tag{1}$$

ここで，s, t, u は媒介変数である．次に，変形領域を格子状に分割し，格子点を制御点とする．$(l+1)\times(m+1)\times(n+1)$ 個の制御点を考えたとき，制御点の位置ベクトル $\boldsymbol{P}_{ijk}$ は次式で表される．

$$\boldsymbol{P}_{ijk} = \boldsymbol{X}_0 + \frac{i}{l}\boldsymbol{S} + \frac{j}{m}\boldsymbol{T} + \frac{k}{n}\boldsymbol{U} \tag{2}$$

FFD では，制御点を任意の位置 ($\boldsymbol{P}'_{ijk}$) に移動させたときの格子の移動に追従させ，頂点を移動する．頂点が N 個ある形状データについて FFD による自由変形を行ったあとの頂点の位置ベクトル $\boldsymbol{X}'_a$ は次式で表される．

$$\boldsymbol{X}'_a = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n}\binom{l}{i}\binom{m}{j}\binom{n}{k} s_a^i(1-s_a)^{l-i}t_a^j(1-t_a)^{m-j}u_a^k(1-u_a)^{n-k}\boldsymbol{P}'_{ijk} \tag{3}$$

各頂点の位置ベクトルは，制御点だけに依存するので頂点ごとに並列処理を行うことができる．

2.2 シェーディング処理

仮想三次元物体で高品質描画を行うためのシェーディング処理では，光源から照射された光の物体表面における反射をシミュレートする．本研究では，反射を三つの成分の合成により表現する Phong の反射モデルを使用する [5]．Phong の反射モデルでは，反射を計算するため，単位法線ベクトルが必要となる．本研究では，頂点に法線ベクトルを指定するスムーズシェーディングを使用する．スムーズシェーディングでは，頂点に接続する各ポリゴンの法線ベクトル $\boldsymbol{N}_i$ の平均を正規化する頂点法線ベクトル $\boldsymbol{N}$ を求めることで面の接続を滑らかにする：

$$\boldsymbol{N} = \frac{\sum \boldsymbol{N}_i}{|\sum \boldsymbol{N}_i|} \tag{4}$$

自由変形により変形処理を行うたびに法線ベクトルの再計算が必要となるが，頂点を共有するすべてのポリゴン法線ベクトルを加算し，正規化を行うため，並列処理前に同期が必要である．

3. 並列処理

3.1 OpenMP

OpneMP はマルチコア CPU 上で処理を並列化するためのプログラミング支援環境である [2]．OpenMP では，繰り返し処理などを分割して並列化する場合，そのコードの前にコンパイラ命令文を挿入する．単純に同じ処理を複数回行う場合には，`#pragma omp parallel` という命令文を追加し，for 文を並列処理する場合には，`#pragma omp for` という命令文を追加する．このように OpenMP では，`#pragma omp` のあとに指示文を追加することで，同期に関する指定や排他的な実行を実現する．また，処理はスレッドという単位に分割され，複数のコアに割り振って計算を行うことができる．

3.2 CUDA での処理手順

CUDA は，科学技術用の汎用計算を行う目的で開発されたので，GPU のハードウェアやグラフィックスに関する知識がなくても並列処理プログラミングを行うことができる [3]．CUDA による汎用計算は，次の三つの手順で実行される．

1. **入力**　CPU 上のメモリから GPU 上のメモリに入力データをコピー
2. **演算**　GPU のプロセッサで計算
3. **出力**　GPU 側から CPU 側に出力データをコピー

この三つの手順で FFD と法線ベクトルの計算を行う方法を GPU1 とする（図 2 左）．

本研究では，FFD や法線ベクトルの計算を行った後に描画を行うことを考慮する．GPU1 では，FFD や法線ベクトルの計算後に出力データを CPU 側へコピーしているが，描画処理時にはそれらのデータを，再び GPU 側にコピーしなければならない．CPU 側の処理が不要なデータは，GPU 上のメモリに保持しておき，CPU に演算終了の通知をするだけにすれば，データ転送時間が省けると考えられる．そこで，データの受け渡しを効率化する方法を GPU2（図 2 右）とする．

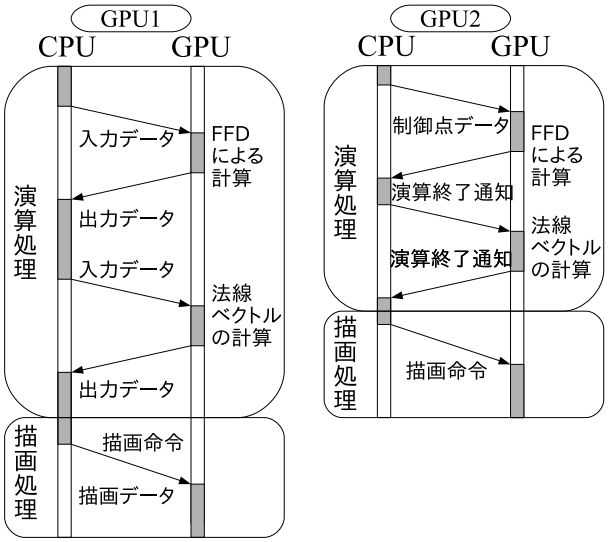

図 2　CUDA での処理手順

GPU2 のように，データを GPU 上のメモリに格納しておき，OpenGL で描画する際に使用するためには，OpenGL 2.0 で追加された Vertex Buffer Object（VBO）を用いる必要がある．本研究の開発環境では，OpenGL 1.1 までしか使用できず，2.0 以上の機能は拡張機能扱いとなる．そこで，The OpenGL Extension Wrangler Library（GLEW[6]）の導入を行った．VBO では，CUDA との相互運用を行うことができる．CUDA と OpenGL の相互運用を行い，VBO 内のデータを CUDA から値を書き換える場合には，以下の三つの手順が必要となる．

1. VBO 変数を定義し，GPU 側にデータをコピー
2. CUDA で使用できるように VBO の変数を登録
3. CUDA の関数内でアドレスを取得

3.3 CUDA での処理分割について

CUDA を用いて汎用計算を行う場合には処理をグリッド，ブロック，スレッドの三つの階層で分割する．また，GPU ではベクトル演算が得意なハードウェア構成であり，グリッドが GPU，ブロックが並列処理ユニット（Streaming Multi-Processor），スレッドがスカラプロセッサに対応する（図 3）．

並列処理ユニット内には，図 4 に示すように VRAM に比べてアクセスが高速なメモリやレジスタが搭載されている．これらを活用することで，より高速な処理が実現できる．

CUDA では，1 ブロックあたりのレジスタ数によって使用できるスレッド数が決まり，レジスタの使用量を削減しないと，並列処理による効果が得られない [3]．本

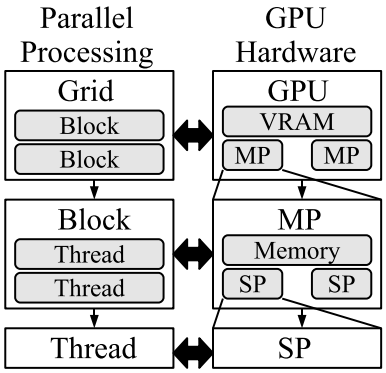

図 3　GPU での処理の分割

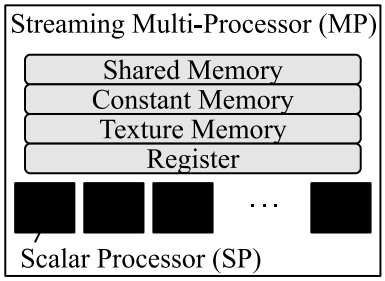

図 4　並列処理ユニットの内部構造

研究において FFD で計算を行う関数では，1 スレッドあたり使用するレジスタ数が 28 個であるため，最大 256 スレッドまで処理を分割することができる．法線ベクトルの計算を行う関数では，1 スレッドあたり使用するレジスタ数が 13 個であるため，最大 512 スレッドまで処理を分割することができる．

4. 処理時間の比較実験

4.1 実験内容

OpenMP による CPU 上での並列処理と，CUDA による GPU 上での並列処理の処理時間を比較するための実験を行った．KONICA MINOLTA 製の三次元スキャナである VIVID910 を用いて計測した表 1，図 5 に示す魚の模型について処理時間の比較を行った．

表 1　魚の模型の基本データ

頂点数	90633 個
ポリゴン数	179801 枚
制御点	$3 \times 3 \times 3$

実験環境を表 2 に，GPU のスペックを表 3 に示す．まず，CUDA による並列処理について，FFD 処理，法線ベクトル計算のそれぞれについて最適なスレッド数を求める実験を行った（4.2 節）．次に，GPU を変更した場合の処理時間の変化を調べた（4.3 節）．最後に，CPU 上でのシングルコア処理，OpenMP を用いた 4 コア処理，

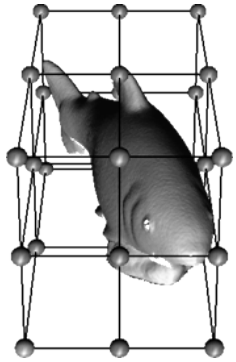

図 5　魚の模型と FFD の制御点

表 2　実験環境

OS	Windows XP SP3(32bit)
CPU	AMD PhenomII X4 2.4GHz (4 Core)
メモリ	3GB (800MHz)
開発環境	Visual Studio 2005 Pro (OpenMP 2.0)
CUDA	CUDAToolKit 3.0

表 3　NVIDIA Geforce 9600GT のスペック

Version	1.1	
メモリ	512MB	
プロセッサ数	64	
動作周波数	1.62GHz	
メモリバンド幅	CPU から GPU	1.69GB/s
	GPU から CPU	1.44GB/s
	GPU から GPU	37.18GB/s

CUDA による二つの提案方法の比較を行った（4.4 節）．

4.2 最適スレッド数の調査

CUDA で並列処理を行うためには，スレッド数とブロック数を指定する必要がある．FFD のスレッド数を α，ブロック数を $\lceil$(頂点数)/$\alpha\rceil$，法線ベクトルの計算のスレッド数を β，ブロック数を $\lceil$(ポリゴン数)/$\beta\rceil$ とする．ここで，$\lceil\ \rceil$ は整数への切り上げ関数とする．スレッド数，ブロック数をどのように指定すると処理が一番高速になるかは，処理内容や GPU によって異なるため，公式の仕様書には示されていない．そこで，最速な α, β を求めるため，FFD，法線ベクトル計算でのスレッド数と処理時間の関係を調べた．α を 2〜256，β を 3〜512 として，処理時間を計測した．FFD では，頂点数によりブロック数を変化させているが，今回の物体モデルでは $\alpha = 1$ のときブロック数が 90633 となり，使用できるブロック数の上限（65535）を超えてしまう．ブロックは 2 次元で管理されており，1 次元ずつ $\lceil$(ポリゴン数)/2$\rceil$ と指定すれば，$\alpha = 1$ でも処理が行える．しかし，2 次元で処理を行った場合と，1 次元で処理を行った場合の内

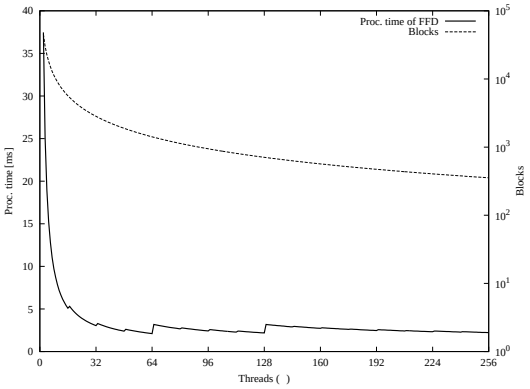

図 6　スレッド数に対する FFD の処理時間とブロック数

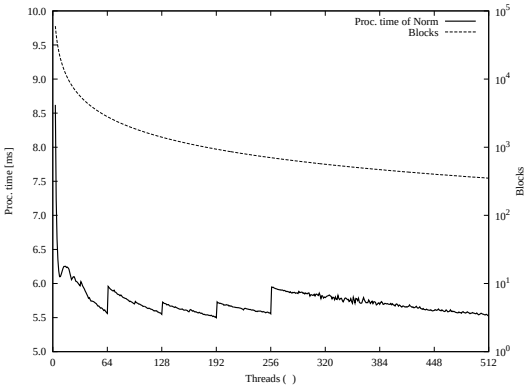

図 7　スレッド数に対する法線ベクトル計算の処理時間とブロック数

部処理の違いは，公式の仕様書には記載されておらず，直接比較ができない．そこで本研究では，$\alpha \geq 2$ として 1 次元のみ使用した．法線ベクトル計算も同様の理由で $\beta \geq 3$ とした．

FFD，法線ベクトル計算の処理時間の変化をそれぞれ図 6，図 7 に示す．また，いくつかのスレッド数に対する具体的な処理時間を表 4 に示す．なお，処理時間は 10000 回の平均として算出した．

図 6，表 4 より，FFD の処理時間はスレッド数 64，128，256 のときに短縮している．その中でもスレッド数 64 のときが一番処理時間が短かったことがわかる．CUDA では，一つのブロックが Warp と呼ばれる単位で実行される．1Warp= 32 スレッドであるため，32 の倍数のスレッド数で処理時間が短縮される仕様である．今回の実験においても 32 の倍数で処理時間が短縮している．

図 7，表 4 より，法線ベクトル計算の処理時間は 64，128，192，256，512 のときに短縮しており，FFD と同様に 32 の倍数で処理時間が短縮している．特にスレッド数が 192 のときが一番処理時間が短かったことがわかる．

並列処理ユニットとブロックは 1 対 1 に対応し，今

表 4　スレッド数に対する処理時間（9600GT，単位 [ms]）

スレッド数	FFD	法線ベクトル計算
63	2.135	5.576
64	2.108	5.562
65	3.183	5.958
127	2.196	5.562
128	2.186	5.548
129	3.181	5.727
191	2.475	5.511
192	2.469	5.498
193	2.571	5.731
255	2.232	5.570
256	2.226	5.556
257	-	5.948
511	-	5.529
512	-	5.522

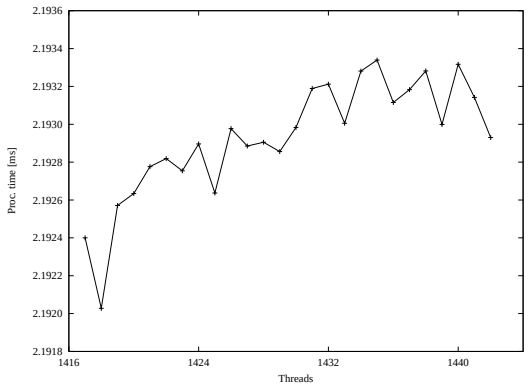

図 8　ブロック数に対する FFD の処理時間

回使用した GPU では並列処理ユニットは 8 個である．FFD でスレッド数を 64 とすると，ブロック数は 1417 となり 8 の倍数とはならない．そこでブロック数が 8 の倍数になれば処理時間がさらに短縮されるかを調べるため，データパディングを用いてブロック数を 1417〜1442 まで変化させて，処理時間の変化を調べた．その結果を図 8 に示す．図 8 より，8 の倍数（1424，1432，1440）前後で，ある程度処理時間が短縮されているが，その効果はごくわずかであることから，ブロック数の調整は行わないこととした．

以上より，処理時間の比較実験では $\alpha = 64$，$\beta = 192$ とする．

4.3　GPU 変更時の処理時間の変化

GPU のハードウェアによって，処理時間がどのように変化するかの比較を行った．比較として使用した GPU は NVIDIA Geforce 285GTX（285GTX）であり，スペックを表 5 に示す．9600GT に比べプロセッサ数は約 4 倍，メモリバンド幅は約 3 倍となっている．使用した

表 5　NVIDIA Geforce 285GTX のスペック

Version	1.3	
メモリ	1GB	
プロセッサ数	240	
動作周波数	1.48GHz	
メモリバンド幅	CPU から GPU	2.41GB/s
	GPU から CPU	2.11GB/s
	GPU から GPU	124.64GB/s

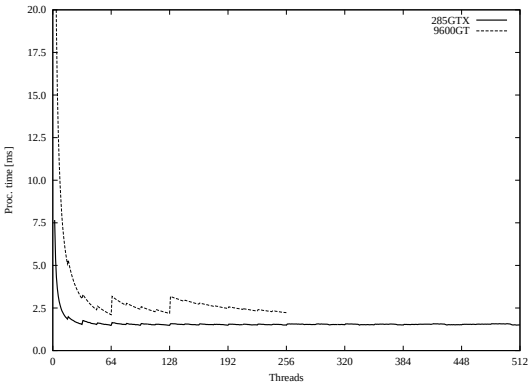

図 9　スレッド数に対する FFD の処理時間（GPU の比較）

OS，CUDA Toolkit のバージョンは同一である．

285GTX において GPU2 でのスレッド数に対する FFD と法線ベクトル計算の処理時間は図 9，図 10 となった．比較のため 9600GT の処理時間も再掲している．また，285GTX ではレジスタ数も増加しているため，FFD において 512 スレッドまで実装することが可能であった．

一部のスレッド数での具体的な値は表 6 のようになり，FFD ではスレッド数が 64，法線ベクトルの計算ではスレッド数が 448 のときが処理時間が最短であった．285GTX においても，Warp の倍数である 32 スレッドごとに処理時間が短縮されている．

9600GT（α = 64, β = 192）と 285GTX（α = 64, β = 448）の最適条件における処理時間の比較を表 7 に示す．

表 6　スレッド数に対する処理時間（285GTX，単位 [ms]）

スレッド数	FFD	法線ベクトルの計算
63	1.484	1.855
64	1.481	1.825
65	1.644	1.872
447	1.521	1.819
448	1.516	1.783
449	1.540	1.823

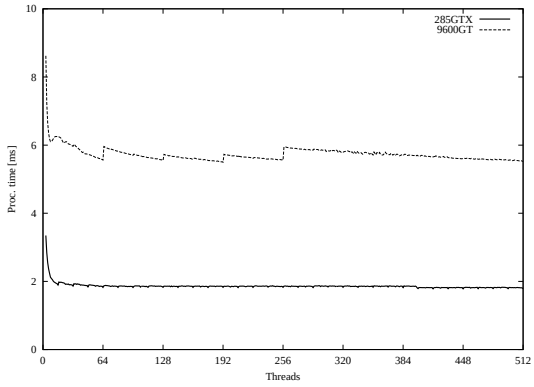

図 10　スレッド数に対する法線ベクトル計算の処理時間（GPU の比較）

表 7　GPU による処理時間の比較（単位 [ms]）

GPU	FFD	法線ベクトル計算	合計
9600GT	2.108	5.498	7.606
285GTX	1.481	1.783	3.264

表 8　FFD，法線ベクトル計算の処理時間（単位 [ms]）

計算対象	CPU1	CPU4	GPU1	GPU2
FFD	1264.570	320.300	3.873	2.108
法線ベクトル	58.430	14.680	8.577	5.498
合計	1323.000	334.980	12.450	7.606
fps	0.756	2.985	80.321	131.475

4.4　処理時間の比較

並列化していないシングルコア処理による CPU1，OpenMP を用いて並列化した 4 コア処理による CPU4，GPU1，GPU2 の処理時間の比較を表 8 に示す．使用した GPU は 9600GT であり，GPU1，GPU2 での FFD でのスレッド数は 64，法線ベクトルの計算でのスレッド数は 192 である．

4.5　考察

一般に，10[fps] 以上（処理時間 100[ms] 以下）で描画が行えると，人間は動画と認識できるのでリアルタイム処理が実現可能となる．

表 8 より，CPU4 では CPU1 と比べ，FFD，法線ベクトル計算のいずれも処理時間が 1/4 程度となり，CPU 上の四つのコアに均等に処理が割り当てられて効率よく並列化が行えたことがわかる．しかし，処理レートは 3fps で実時間処理は達成できていない．GPU1 では CPU4 と比べ，FFD で処理時間が 1% 程度，法線ベクトル計算で 58% 程度となり，処理レートは 80fps と実時間処理が可能となった．FFD では，頂点ごとに処理が独立しているため，並列化の効果が非常に大きく，GPU を用いることで大幅に処理時間が短縮できたと考えられる．法線ベク

トル計算では並列処理前に同期が必要なため，FFD に比べて並列化の効果は小さいが，GPU 間の高速なメモリアクセスや，処理の効率化により高速化が実現できたと考えられる．

CPU と GPU 間のデータ転送を効率化した GPU2 の処理時間は，GPU1 と比べて FFD で 54% 程度，法線ベクトルの計算で 64% 程度となり，処理レートも 131fps に向上した．今回使用した形状データ（図 5，表 1）の場合，FFD に関しては，頂点データの転送を省くことができた．そのデータ量は次式で見積もることができる：

$$\begin{aligned}&(\text{頂点数}) \times (XYZ \text{ 軸データ}) \times (float \text{ 型})\\ &= (\text{頂点データ})\\ &90633 \times 3 \times 4 \approx 1[MB] \end{aligned} \tag{5}$$

また，法線ベクトル計算では，頂点データと面データの転送を省くことができた．そのデータ量は次式で見積もることができる：

$$\begin{aligned}&(\text{頂点データ}) + (\text{ポリゴン数}) \times (\text{面データ}) \times (int \text{ 型})\\ &= (\text{頂点} + \text{面データ})\\ &1087596 + 179801 \times 3 \times 4 \approx 3[MB] \end{aligned} \tag{6}$$

表 7 より，GPU を変更した場合，FFD では処理速度が 1.4 倍，法線ベクトルの計算では処理速度が 3.1 倍となった．285GTX は 9600GT に対し，プロセッサ数が約 4 倍，GPU 間のメモリ転送速度が約 3 倍に向上しているが，動作周波数は 0.9 倍に低下している（表 3，表 5 参照）．FFD の計算では，式 (3) で示すように，一つのプロセッサあたりの計算量が多いため，プロセッサ数増加の効果と，動作周波数低下の効果が相殺されて，あまり処理速度が変わらなかったと考えられる．法線ベクトル計算では，GPU 間のデータ転送速度向上の効果が，処理時間向上につながったと考えられる．

図 9，10 において，285GTX はスレッド数に対する処理時間の変動が少ない．これは，CUDA では Warp ごとに処理をしており，Streaming Multi-Processor がメモリ領域にデータの読み込みが完了した Warp から処理を行っているため，GPU 間のメモリ転送速度が高速である 285GTX では処理時間の変動が少なかったと考えられる．

5. まとめ

FFD による物体の自由変形について，変形の効果をリアルタイムで描画するための高速処理について検討した．CUDA を用いて GPU 上で FFD による仮想三次元物体の自由変形と，変形結果の描画を，効率よく切り替えることで処理の高速化を行った．演算処理をする際，GPU のハードウェアの構成を活用するための高速化設定について検討し，分割数の最適化としてスレッド数，ブロック数の関係を実験的に求めた．演算処理と描画処理を CPU を介さず GPU 上で行う提案法（GPU2）では，頂点数 9 万程度，ポリゴン数 18 万程度の高精細な形状データの自由変形を，131[fps] 程度の更新レートで，リアルタイム処理することができた．今後は，自由変形における操作性について検討することが課題である．

謝辞

本研究の一部はエヌ・エス知覚科学振興会の研究開発助成の支援を受けて実施された．

参考文献

[1] T.W. Sederberg, S. R. Parry, “Free-Form Deformation of Solid Geometric Models”, SIGGRAPH 86, Vol.20, No.4，pp.151-160，1986.

[2] 北山洋幸，“OpenMP 入門マルチコア CPU 時代の並列プログラミング”，秀和システム，2009.

[3] “NVIDIA CUDA C Programming Guide 3.2”, http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf.

[4] M.Modat，G.R.Ridgway，Z.A.Taylor，M.Lehmann，J.Barnes，D.J.Hawkes，N.C.Fox，S.Ourselin，“Fast free-form deformation using graphics processing units”, Computer Methods and Programs in Biomedicine，2009.

[5] 野村，高橋，若月，駒形，石井，“アクティブ照光画像からの物体表面反射パラメータ推定”，電子情報通信学会信越支部大会講演論文集，p.86，2009.10.

[6] “The OpenGL Extension Wrangler Library”, http://glew.sourceforge.net/.