I-033

ボリュームデータのインタラクティブなペイントシステム

Interactive Paint System for Volume Data

岩崎 航† 土橋 宜典† 山本 強†
Wataru Iwasaki Yoshinori Dobashi Tsuyoshi Yamamoto

1. まえがき

コンピュータグラフィックスにおいて，ボリュームデータは雲や煙などの表現や三次元テクスチャとして多用されている．このようなボリュームデータを生成する手法は数多く提案されているが，生成されたボリュームデータの色は必ずしもユーザの希望通りであるとは限らない．本稿では，二次元のグレースケール画像をカラー化する手法[1]を応用することでボリュームデータの色付けを行うシステムを提案する．提案法により，ボリュームデータに対してインタラクティブに色付けを行うことが可能である．

2．従来研究

二次元のグレースケール画像をカラー化する手法として文献[1][2]が挙げられる．文献[1]では，ユーザがグレースケール画像上に直接指定した色情報をもとに，最適化によって画像のカラー化を行う．文献[2]では，ユーザが所望する色情報を持ったカラー画像を見本データとして用い，その色情報をもとに任意のグレースケール画像をカラー化する．この手法では，希望に合った見本データをユーザがあらかじめ所持していることが前提となる．そのため，一度色付けを行った結果に対して再度色付けを行う場合，ユーザは要求に合った見本データを新たに用意する必要がある．本研究では，より直感的に色の編集を行うことができるシステムを目指すため，見本データを必要としない文献[1]の手法を用いる．

3. グレースケール画像のカラー化

本手法では，文献[1]を応用することでボリュームデータへの色付けを実現する．そのため，まず，文献[1]の手法について詳しく述べる．

この手法では，輝度が類似していれば色彩も類似しているという仮定に基づき，YUV 色空間において各ピクセルの色を決定していく．ここで，Y は輝度，U は輝度と青の色差，V は輝度と赤の色差を表す．入力画像はグレースケールであるため，YUV 色空間においては Y 成分しか持たない．ユーザが指定した色情報をもとに，各ピクセルに対して最適な U, V を求めることで，画像のカラー化を行うことができる．

各ピクセルの U は，式(1)に示す $J(U)$ を最小化するよう決定する．

$$J(U)=\sum_{\mathbf{r}}\left\{U(\mathbf{r})-\sum_{\mathbf{s}\in N(\mathbf{r})}w_{\mathbf{rs}}U(\mathbf{s})\right\}^2 \qquad (1)$$

$$w_{\mathbf{rs}}\propto\exp\left(-\frac{(Y(\mathbf{r})-Y(\mathbf{s}))^2}{2\sigma_{\mathbf{r}}^{\,2}}\right) \qquad (2)$$

ここで，r は注目ピクセル，s は注目ピクセルの 8 近傍ピクセル，σ_r は注目ピクセルおよび 8 近傍ピクセルにおける輝度の分散，w_{rs} は注目ピクセルに対する 8 近傍ピクセルの重みを表す．式 (1), (2) は，注目ピクセルの U と 8 近傍ピクセルの U の加重平均との差の二乗和を表す．J を最小化することによって U を算出した後，V についても同様にして求める．これにより，最適な U, V を算出することができる．

提案法では，上述した手法をボリュームデータのカラー化に応用する．上述の手法は 2 次元画像が対象であるため，式(1)で表される最適化問題を 3 次元のボリュームデータに拡張する必要がある．最も単純な方法は，ボリュームデータの各ボクセルの色とその 26 近傍のボクセルの色の重み付き平均との差の二乗和を最小化することである．しかし，この方法には以下のような問題がある．

第一に，記憶容量の問題がある．式(1)で表される最適化問題は，J の微分値を 0 と置いた連立方程式により解くことができる．しかし，そのためには，係数行列を記憶しなければならない．ボリュームデータの場合，係数行列を記憶するためには，元のボリュームデータの 27 倍のサイズという極めて大量のメモリ領域を確保しなければならず，現実的ではない．

第二の問題は計算時間である．一般に，上述した連立一次方程式は，繰り返し計算により数値的に解く．しかし，ボリュームデータの場合，未知変数の数はボクセル数であり，極めて大規模な問題となるため，計算コストが高い．

本研究では，上述の問題点を解決するため，式(1)を 3 次元に拡張するのではなく，2 次元の最適化問題を繰り返し解くことでボリュームデータをカラー化する手法を提案する．

4. 提案手法

提案法では，ボリュームデータが二次元画像の重ね合わせであるという点に着目する．ボリュームデータをスライスすることで生成される各断面図に対して文献[1]を繰り返し適用することで，ボリュームデータ全体の色付けを行う．これにより，一度の計算に必要な記憶容量を削減することができる．さらに，文献[1]の手法を GPU を用いて実装することで飛躍的な高速化を図る．

提案法では，ユーザはボリュームデータに対し，x, y, または z 軸に垂直な任意の断面を選択し，ペイントソフトの要領で所望のボクセルの色を指定する．ユーザが指定したボクセルの位置および色情報はマスク用ボリュームデータとして別途保存しておく．以下，ボリュームデータのカラー化と GPU 実装について詳しく説明する．

†北海道大学大学院情報科学研究科, IST

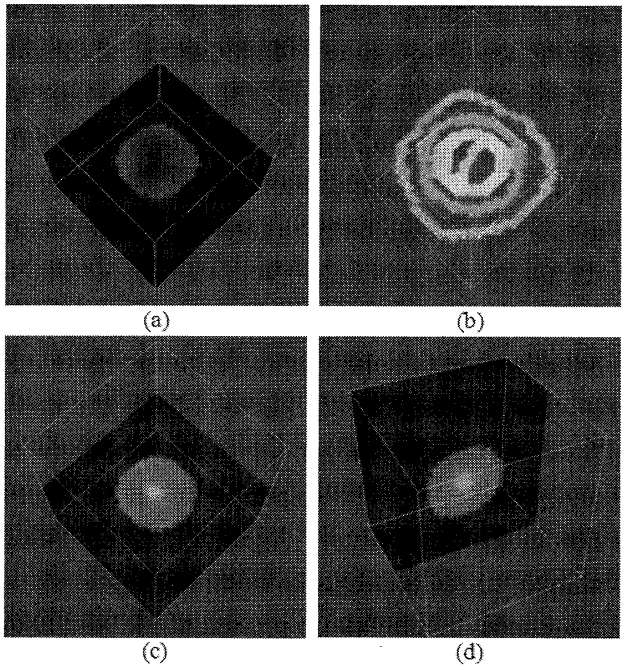

図 1. 実験結果

4.1 ボリュームデータのカラー化

提案法では，x, y および z 軸それぞれに垂直な断面図について，繰り返し 3 節で述べた手法を適用する．具体的には，まず，ボリュームデータの x 軸に垂直な各断面図を抽出し，二次元画像としてカラー化を行い，その結果を再びボリュームデータに格納する．同様の処理を，y および z 軸に垂直な断面について繰り返す．これにより，ボリュームデータ全体のカラー化を行うことができる．また，このようにすることで，3 次元での重み係数行列を保持する必要がなく，少ないメモリ効率でカラー化を行うことができる．

提案法では，ユーザがペイント操作を行っていない間は，上述の最適化処理が常に行われる．そのため，ユーザはインタラクティブにカラー化結果を確認しながらペイント操作を行うことができる．

4.2 GPU による高速計算

高速にカラー化処理を行うため，式(1)の最適化問題をマルチグリッド法[3]により解き，さらに GPU を利用する．具体的には，カラー化するボリュームデータのある断面図が与えられると以下の処理により最適化問題を解く．

まず，断面図が与えられると，この画像を GPU に転送する．そして，重み係数行列を計算して記憶する．次に，断面図の解像度を 1/2 にダウンサンプルし，同様に，重み係数行列を計算する．この処理を画像サイズが 2 になるまで繰り返す．これらの処理は全て GPU 上で行う．これにより，CPU と GPU とのデータ転送にかかる時間を低減する．

次に，元の解像度に対応する連立一次方程式を反復法により解く．本研究では，反復法として並列性の高い Jacobi 法を採用した．また，反復回数は，数回程度として，近似解を算出する．これにより，全画素の色の近似解が得られる．この近似解の残差を計算し，1/2 の解像度にダウンサンプリングする．そして，残差方程式を解いて，誤差を求める．以下，同様に残差のダウンサンプルと誤差の計算を画像サイズ 2 の場合まで全て繰り返す．その後，逐次，誤差をアップサンプルし，誤差を補正する．これら全ての処理を GPU により計算する．GPU での計算が終了した後，カラー化された画像を CPU に転送する．

以上の方法により，極めて高速に断面図のカラー化を行うことができる

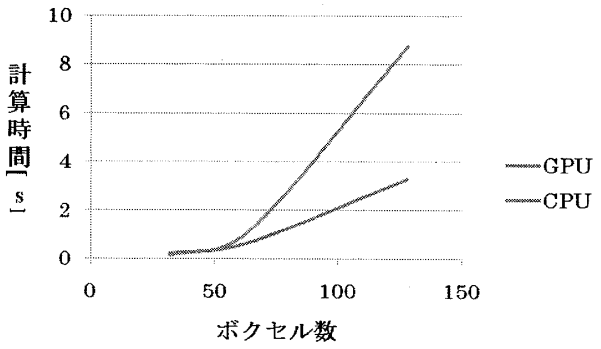

図 2. 計算時間の比較

5. 実験結果

提案法を適用した結果を示す．実験環境は，CPU が Intel Core 2 Quad Q9400，GPU が NVIDIA GeForce GTS 250，グラフィックス API が OpenGL である．

図 1(a)は編集を行っていない状態である．そして，(b)に示すマスク用ボリュームデータを用いてカラー化を行った結果が(c)である．また，(d)は(c)とは異なる断面図である．提案法により，ユーザの指定した色情報に従って滑らかに色付けが行われていることが確認できる．また，GPU によりボクセル数が大きくなるほど効果的に高速化を行うことができた（図 2）．

6. まとめと今後の課題

本稿では，二次元のグレースケール画像をカラー化する手法を応用することで，ユーザがインタラクティブにボリュームデータの色付けを行うことができる手法を提案した．提案法により，既存のボリュームデータに対し，ユーザが所望する色で色付けを行うことが可能となった．

今後の課題としては，ユーザインターフェースの改善が挙げられる．本稿のシステムにおいて，ユーザが色塗りを行うことができるのは x, y, または z 軸いずれかに垂直な断面図のみである．より直感的かつ効率的に色付けを行うために，座標軸に囚われない断面を生成可能であることが望ましい．

参考文献

[1] A. Levin, D. Lischinski and Y. Weiss, Colorization using optimization, *In Proceeding of ACM SIGGRAPH 2004*, pp. 689-694

[2] T. Welsh, M. Ashikhmin and K. Mueller, Transferring color to grayscale images, *In Proceeding of ACM SIGGRAPH 2002*, pp. 277-280

[3] William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, Numerical Recipes in C Second Edition, Cambridge University Press, 1992