Vol.2013-CG-151 No.5
2013/6/24

ベクトル量子化を施した 3 次元テクスチャの合成

谷 翼 [1]　土橋 宜典 [1,2]　佐藤 周平 [1]　山本 強 [1]

コンピュータグラフィックス(CG)の進歩により，映画やゲーム，アニメなど様々な場面において， CG が多く用いられるようになっている．これら CG における映像生成の手段の一つとして，流体シミュレーションにより物理現象を再現する研究が広く行われている．しかし，写実的な映像を生成するためには，高精度なシミュレーションが必要となり，計算コストが膨大なものになってしまう．そこで本研究では，流体シミュレーションにより得られた結果（3 次元テクスチャ）を合成することで，有限時間の映像を任意時間再生可能とする手法を提案する．3 次元テクスチャを複数の小領域に分割し，一部の領域のみ終端データから始端への遷移を行うことで，違和感のない映像を生成する．しかし，流体のシミュレーションデータは容量が大きいため，多くのデータを扱う場合，使用するメモリ量が膨大になってしまう．そのため，データ圧縮手法の一つであるベクトル量子化を用い，シミュレーションデータを圧縮したうえでメモリへ格納する．

Synthesis of Three-dimensional Textures With Vector Quantization

TSUBASA TANI[1]　YOSHINORI DOBASHI[1,2]
SYUHEI SATO[1]　TSUYOSHI YAMAMOTO[1]

1. はじめに

近年，映画やゲームなどにおいて，流体を含む様々な映像が CG で表現されるようになった．特に，流体などの自然現象は，物理シミュレーションを用いることにより，写実的な映像を生成することができる．しかし，物理シミュレーションは，一般に映像のリアリティや複雑さを追求するほど大きな計算コストが必要となる．また，シミュレーションにより得られた結果が理想と異なる場合，理想通りになるようパラメータを調整し，再度シミュレーションをやり直す必要がある．これには，さらに膨大な時間がかかる．

そこで本研究では，シミュレーション結果から得られたボリュームデータを合成することで，任意時間の映像を生成するための手法を提案する．提案法では，流体シミュレーションにより生成した密度場などのデータ（3 次元テクスチャ）を小さな領域に分け，各領域で時間をずらして段階的にループ処理を行う．これにより，スムーズなループが可能であり，有限時間長のデータから任意時間の映像を合成できる．しかし，流体のシミュレーションデータは容量が大きいため，多くのデータを扱う場合，使用するメモリ量が膨大になってしまう．そこで，データをベクトル量子化により圧縮してからメモリに転送し，結果を生成する際に復元することで，メモリ容量を削減する．提案手法により，有限時間のデータから，任意時間の映像を得ることができる．

1　北海道大学
Hokkaido University
2　独立行政法人科学技術振興機構 CREST
JST CREST

以降，本論文の構成は，2 節で従来研究について述べ，その中でも本手法に応用する Kiran らによる手法について 3 節で解説する．そして，4 節で提案手法について述べた後，5 節で提案手法を用いた実験結果とその考察について述べ，最後に 6 節でまとめとする．

2. 従来研究

ループ映像を作成する手法はいくつかあり，代表例として Arno らによる Video Textures[1]が挙げられる．しかし，この手法は動画像が対象であり，3 次元ボリュームデータには適用していない．Kiran らは滝の流れや煙などの流体を対象とした動画像の合成および編集処理手法を提案した[2]．本手法では，この方法を 3 次元のボリュームデータへ拡張する．このとき，２次元の動画像と比較して，３次元のボリュームデータは記憶容量が増大するため，圧縮処理を施す．ボリュームデータを圧縮する手法はいくつか提案されている．Yeo らは離散コサイン変換を用いて圧縮したボリュームデータをレンダリングする手法を提案している[3] ．また，村木は，ボリュームデータをウェーブレット変換することでレンダリングする手法を提案している[4]．しかし，これらの手法では，圧縮したデータを解凍する必要があるため，計算コストが高い．村木は，ウェーブレット変換後のデータを直接レンダリングする手法も開発しているが，X 線画像のような結果しか得ることができない．Ning らはベクトル量子化により圧縮されたボリュームデータを表示する手法を提案している[5]．この方法では，ボリュームデータを小さな nxnxn のブロックの集合と考え，これをベクトル量子化により圧縮する．これにより高い圧縮率を実現できる．また，ベクトル量子化は，後述する性

質から圧縮後のデータを展開するための計算コストも非常に小さい．本手法においてもこの圧縮データの復元の早さに着目し，ベクトル量子化によるデータ圧縮を行う．

3. Kiran らによる手法

本節では，提案手法の基礎となっている Kiran らの手法[2]について説明する．この手法は，滝や川などのような，全体の形状が大きく変化しない定常的な動作をする流体の動画に対し，部分的な合成処理および編集処理を行う手法である．これらの処理により，入力となる動画をスムーズにループさせることや，ユーザが指定した曲線に沿った変形が可能となる．この手法では，動画中の流体部分を小さな領域に分割し，その領域単位で合成および編集を行う．また，領域ごとに異なるタイミングでループさせることにより，スムーズに動画をループさせている．

この手法を用いて動画をループさせる大まかな流れを説明する．まず，編集対象となる入力動画中において，流体の流れの中心線をユーザが曲線により指定する（図 1 参照）．この経路を以降は flow curve と称する．この時，flow curve 周辺以外の領域, すなわち動画中の流体以外の範囲については時間変化しないものとし，入力動画の任意のフレームの画像（本稿では 1 フレーム目）を使用する．以下で，処理の詳細を説明する．

まず，flow curve 上に n 個の点を均等に発生させる．ただし，n はユーザにより指定する．そして，これらの点を中心とする正方形のブロック領域 d_i（i=1, 2, …n）を定義する（図 1 参照）．この手法では，これらのブロック領域単位で処理を行う．以降，各ブロック領域が取得する入力動画の画像をブロックテクスチャと称する．

次に，出力動画の各フレームの画像を生成する．入力動画中の領域 d_i から取得した各テクスチャを，出力動画の対応する領域に描画することで合成画像を作成する. この時, 複数のブロックテクスチャが重なっている範囲は，それらの平均値を用いる．また，この合成画像におけるブロック領域外には, 入力動画の 1 フレーム目をそのまま描画する．

上述の方法を用いて通常の動画再生を表現した場合，各領域に描画するブロックテクスチャは図 2 のように表される．この図は，縦が flow curve 上の領域 d_i，横が時間経過を表す．また表中の数字は，時間 t に取得する入力動画のフレーム番号を表している．例えば出力動画の 1 フレーム目においては，各領域 d_i に対して，入力動画の 1 フレーム目から取得したテクスチャを描画する．この場合，合成画像の流体部分については，入力動画の 1 フレーム目と同等になる．出力動画の 2 フレーム目，3 フレーム目を生成する場合も同様に，入力動画の 2 フレーム目，3 フレーム目から取得したテクスチャを d_i に描画する．通常の再生方法において動画がループする場合，各領域 d_i に描画されるテクスチャの全てが同時に，1 フレーム目の画像へジャンプ

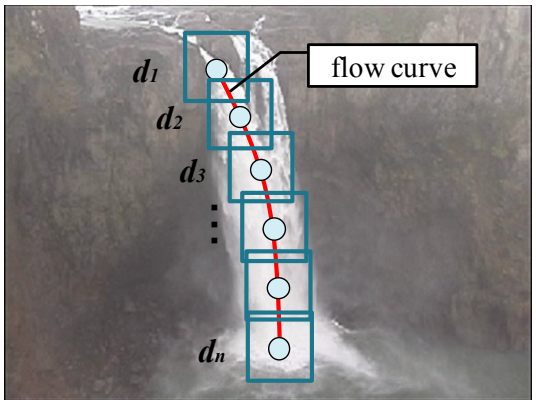


図 1　flow curve の分割とブロック領域

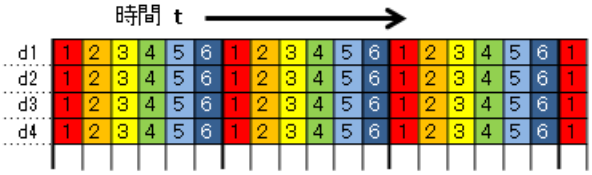


図 2 通常のループ

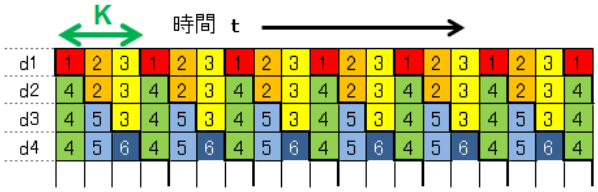


図 3 文献[2]のループ方法

するため，動画の不連続性が目立ってしまう．

この問題を解決するため，文献[2]の手法では，図 3 のように，各領域が互いに異なるタイミングでテクスチャをループさせる. これにより, 段階的に動画全体をループさせ，通常ループ時に生じる違和感を低減することができる．具体的な処理については，まずユーザがループ幅 K を指定する．そして，ある領域がループするタイミングでは，他のテクスチャは通常の動画再生を続けるようにループのタイミングを階段状にずらす（図 3 参照）．すなわち，出力動画の各ブロック領域 d_i において，入力動画のフレーム i を始端とする K フレームのループ動画を取得する．これを出力動画に用いるテクスチャとして扱う．例を図 3 に従って述べると，出力動画の 1 フレーム目において，領域 d_1 には入力動画の 1 フレーム目のテクスチャを描画し，領域 d_2 以降には入力動画の 4 フレーム目を描画する．2 フレーム目以降についても，同様に図に従って各ブロック領域にテクスチャを描画していく． このように，flow curve 上の各領域で時間をずらして，段階的にループをさせることにより，ループ時に流体全体が一斉に変化することが無く，違和感を生じないスムーズなループが実現される．

4. 提案手法

提案手法では，3 節で説明した 2 次元の動画のループ手法を，3 次元のボリュームデータに対して適用することにより，3 次元流体シミュレーションにより生成されたボリュームデータの合成を可能とする．文献[2]では 2 次元のブロック単位で行われていた処理を 3 次元のブロック単位で同様に行う．つまり本手法では，入力された 3 次元動画の各フレームから，ボリュームデータの持つ密度値をブロック単位で取得し，それらを合成することで出力ボリュームデータを生成する．

4.1 ループ手法の 3 次元への拡張

最初に，入力となるシミュレーションデータと同じ大きさを持つ表示用の空間を定義する．以降，この表示用の空間を表示用空間と称し，この空間において合成を行う．次に，2 次元の場合と同様に，元データの流体が流れる経路をユーザが指定し，flow curve を分割する点と，それを中心とするブロック領域 d_i（i=1, 2, …n）を定義する（図 4 参照）．これらブロック領域単位で，ボリュームデータを入力データの対応する位置から表示用空間へコピーすることで，流体全体を合成する．

それぞれの領域に対して，入力データのどのフレームからブロックをコピーするのかは 2 次元におけるループ手法に準ずる（図 3 参照）．つまり，合成結果の 1 フレーム目においては，d_1 には入力データの 1 フレーム目，d_2 以降には入力データの 4 フレーム目のボリュームデータを割り当てる．これを描画空間中の対応するブロック領域にコピーすることによって，流体全体を描画する．合成結果の 2 フレーム目，及びそれ以降のフレームについても同様に，それぞれ対応する位置及びフレームからボリュームデータを取得し，描画空間中の各ブロックにコピーする．複数のブロックが重なっている部分についても文献[2]と同様，ボクセル単位で各ブロックの密度値を加算していき，最後にボクセルごとにブロックの値が加算された回数で割ることで，密度値を決定する．

4.2 ベクトル量子化

流体のシミュレーションデータは容量が大きく，また様々な映像を生成するためには，十分な数のデータが必要となるため，そのデータ量は膨大なものとなってしまう．そこで本稿では，前処理として入力データに対して，データ圧縮手法の一つであるベクトル量子化を適用することで，ボリュームデータの圧縮を行う．これにより，データの格納に必要となるメモリ容量を削減することができる．以下で，ベクトル量子化の詳細について説明する．

ベクトル量子化とは，図 5 のように，あるベクトルの集合 V（黒矢印）に対して，より個数の少ないベクトル集合 V_c（青矢印）により近似して置き換える圧縮処理である．このとき，V_c および V_c 内の各ベクトルは，それぞれ，コードブックおよび代表ベクトルと呼ばれる．圧縮後のデータにおいては，V_c 内の各代表ベクトルの成分と，V の各ベクトルがコードブックの何番目の代表ベクトルで置き換えられたのかを記憶した配列（これをインデックス配列と呼ぶ）があれば，データの復元が可能となる．これにより，全てのベクトルについてその成分を記憶している元データと比較して，データ容量を削減することができる．

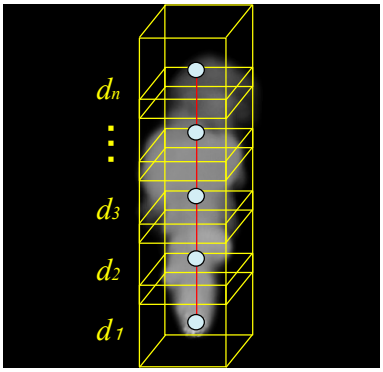


図 4　3 次元ボリュームデータへの拡張

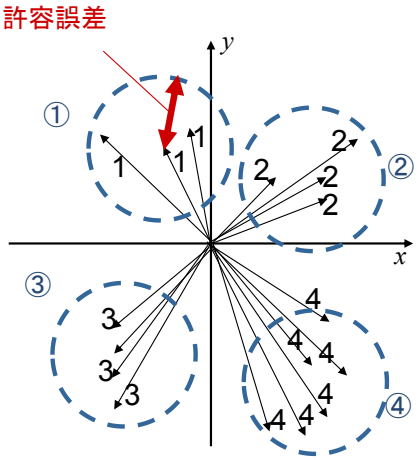


図 5　2 次元でのベクトル量子化の例

本手法においては，このベクトル量子化を以下の手順により 3 次元のボリュームデータに対して適用する．まず，図 6 に示すように，圧縮対象となるボリュームデータを nxnxn のボクセルからなる小ブロックに分割する．このとき，総ブロック数 N_b は(N_x/n) x (N_y/n) x (N_z/n)となる．ただし，N_x, N_y, N_z は n の整数倍とする．各ブロックが持つ n^3 個分のボクセルの密度値を n^3 次元ベクトルとみなせば，もとのボリュームデータは N_b 個の n^3 次元ベクトルの集合 V と考えられる．このベクトル集合 V に対してベクトル量子化を施す．本稿においては n=2 とし，最初に分割した各ブロックの中から，予め指定した数の代表ベクトルを k-means 法により選出した．また，データ容量のさらなる圧縮のために，1 フレーム目のコードブックを用いて全フレームのボリュームデータに対しベクトル量子化を行う．本稿では大きく流れの変わらない，定常的な流体のデータ

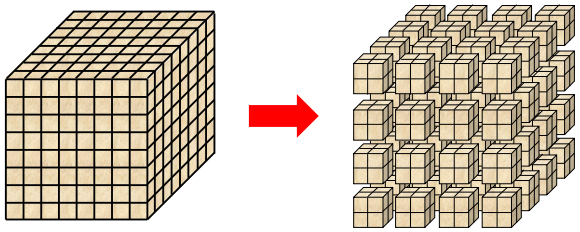


図 6 ボリュームデータの小ブロックへの分割

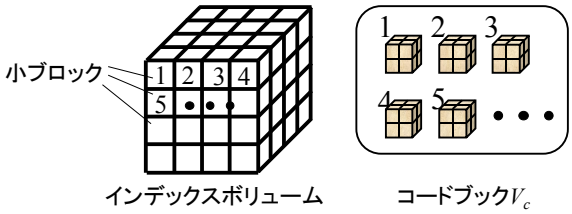


図 7 ベクトル量子化されたボリュームデータ

を対象としている．そのため，全てのフレームに対し，1 フレーム目のコードブックを用いても，十分な品質を保つことができる．また，圧縮したボリュームデータは，インデックス配列に従ってコードブックの各ボクセルが持つ密度値を順に参照することで容易に復元することができる（図 7 参照）．

5. 実験結果・考察

図 8 に，3 次元の煙のボリュームデータに対して提案法を適用した例を示す．実験環境は，CPU: Intel Core i7-3770 CPU，メモリ：3.5GB，グラフィックボード：NVIDIA GeForce GTX 660 である．入力のボリュームデータのサイズは 50x50x100 であり，各ボクセルの密度値を符号なし 8 ビット（1 バイト）で記録した煙のボリュームデータ 200 フレームを用いた．入力データの 1 フレーム目と 11 フレーム目をボリュームレンダリングした画像を図 8 に示す．このとき，入力データ 1 フレームあたりのデータ容量は 250K バイトとなる．ループ幅 K を 100 フレームとし，また，flow curve の分割数を変えた場合のループ処理の比較についても示す．

5.1 ループ処理の適用例

図 9(a)および(b)は，それぞれ，flow curve の分割数を 20 および 100 に設定し，3.1 節で述べたループ処理を適用した結果である．この図において，右図は左図から 10 フレーム経過した場合の形状を示している．分割数 20 の場合（図 9(a)）では，10 フレーム経過する間に，ループ処理により煙の形状が急速に変形してしまう．これに対し，分割数 100 の場合（図 9(b)）は，ループ処理による煙の変形が緩やかであり，自然な動きを保っている．つまり，flow curve の分割数が少ないほど，ループ処理による 1 フレームあたり変形が大きくなり，動画で見た場合の不自然さが増してしまう．これは，隣り合った位置が 1 フレームずつずれて順次ループする本手法の性質上，あるボクセルにおいて重複するブロック領域の数が多いほど，平均値を取った場合の周囲との差異が少なくなるためであると考えられる．しかし，分割数を増やすほど，入力データとしてより多くのフレーム数が必要になってしまう．この問題については元データから描画空間にブロックをコピーする際に，ブロック中央からの距離に応じて，ブロックの密度値に重み付けしてから合成することで改善が期待できる．

5.2 ベクトル量子化の適用例

次に，元データにベクトル量子化を施し，圧縮したボリュームデータに対して，ループ処理を適用した結果を図 10 に示す．ここでは圧縮についてのみ評価するため，flow curve の分割数は 100 で固定する．このとき，元データは 200 フレームを入力し，その容量は 50M バイトであると考える．代表ベクトルのサイズは 2^3 としたため，代表ベクトル 1 個あたりのデータ容量は 8 バイト，インデックスボリュームのサイズは 1 フレームあたり 25x25x50 となる．

図 10(b)は，代表ベクトル数 256 個の結果であり，このときインデックスボリュームの数値は 1 バイトで表される．圧縮後のインデックスボリュームのデータ容量は，1 フレームあたり，1x25x25x50=31,250 バイトであり，元データの 1/8 の容量になる．全体のデータ容量については，コードブックの容量 8x256=2048 バイトが上乗せされるものの，これは元データ 1 フレームの 1/1000 以下のデータ容量である．従って，入力データ全体に対する圧縮率もほぼ 1/8 となる．

図 10(b)は代表ベクトル数 31,250 個の結果である．これは，1 フレーム目のボリュームデータを小ブロックに分割し，その全てをコードブックとして保存した場合であり，このときインデックスボリュームの数値は 2 バイトで表される．このときのインデックスボリュームのデータ量は，1 フレームあたり，2x25x25x50=62,500 バイトであり，元データの 1/4 の容量になる．この場合の全体のデータ容量については，コードブックの容量が元データ 1 フレーム分あるため，入力データ全体に対する圧縮率は 51/200 となる．

それぞれの圧縮結果を見比べた場合，図 10(b)では圧縮前と殆ど画質が変化していないのに対し，図 10(a)では圧縮により強いノイズが発生している．ここで，以下の式で表されるピーク信号対雑音比（PSNR）を用いて，圧縮前のデータに対する劣化の比較を行った（表 1 参照）．

$$PSNR = 20 \cdot \log_{10} \frac{MAX_d}{\sqrt{MSE}}$$

この式において，MAX_d は取りうる密度の最大値（ここでは 1.0），MSE は圧縮していない場合との平均二乗誤差を意味する．ただし，表中の値は編集結果の K フレームの平均値をとっている．表の値からも，図 10(b)は図 10(c)に比べて大きく劣化していることがわかる．これは代表ベクトルの数が多いほど，各代表ベクトルとデータとの差分が小さ

表 2　代表ベクトル数に対する PSNR の比較

	代表ベクトル数	
	256	31250
PSNR[dB]	47.47703336	51.64990185

くなるため，元データにより近い結果が生成されるためである．

ノイズの見え方について，軸方向正面から見た場合に，特に圧縮によるノイズが目立った．これは，元データを小ブロックに分割した境界が，正面から見た場合に視点から一直線上に重なることによるものと考えられる.そのため，小ブロックに分割する際,ブロックを少しずつずらすなど，分割の仕方を工夫することによりある程度の改善が期待できる．

6. まとめ

本論文では，シミュレーションにより得られた結果を合成する手法について提案した．提案手法により，有限のフレーム数のボリュームデータをスムーズにループさせることが可能となった．また，ベクトル量子化によりデータ量の削減が可能となった.

今後の課題としては，主に以下の 2 点が考えられる．1 つ目は，煙以外のシミュレーション結果に対しても本手法を適用することである．本稿では，煙のボリュームデータにのみ提案法を適用したが，全体の形状が大きく変化しない定常的な流体であれば，炎や水などの流体に対しても有効であると考えられる．2 つ目は，流体の変形処理の実装である．ユーザが flow curve を任意に変形し，その変形した flow curve に沿うようにブロック領域を変形する．これにより，1 つのデータから，様々な種類の結果の作成ができると考えられる．

参考文献

[1]Arno Schödl and Richard Szeliski and David H. Salesin and Irfan Essa. Video textures. In *Proc. SIGGRAPH '00,* pp. 489-498,2000.
[2]Kiran S. Bhat and Steven M. Seitz and Jessica K. Hodgins and Pradeep K. Khosla. Flow-based Video Synthesis and Editing. In *Proc. SIGGRAPH 04*, pp. 360—363, 2004.
[3] B. Yeo, B. Liu, “Volume Rendering of DCT-Based Compressed 3D Scalar Data,” *IEEE Trans. on Visualization and Computer Graphics*, Vol. 1, No. 1, pp. 29-43 (1995).
[4] S. Muraki, “Multiscale Volume Representation by a DoG Wavelet,” *IEEE Trans. on Visualization and Computer Graphics*, Vol. 1, No. 2, pp. 109-116 (1995).
[5] P. Ning, L. Hesselink, “Fast Volume Rendering of Compressed Data,” *Proc. IEEE Visualization’93*, pp. 11-18 (1993)

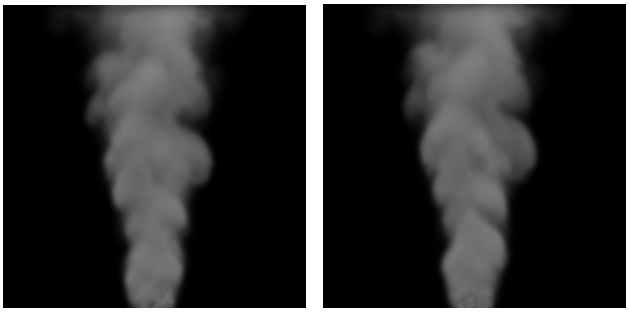
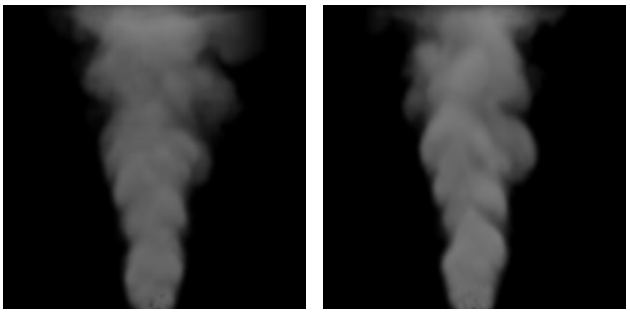
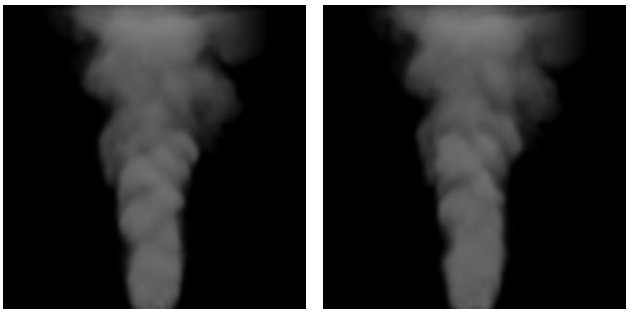


図 8 入力データ

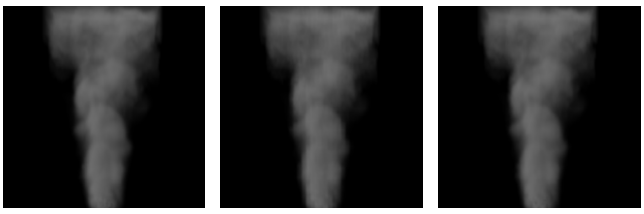


(a) flow curve の分割数 20



(b) flow curve の分割数 100

図 9 ループ手法を適用した結果．右図は，左図から 10 フレーム後の形状である．



(a) 未圧縮の場合　(b) 256　(c) 31250

図 10　ベクトル量子化による圧縮を施した結果