Vol.2009-CG-137 No.5
2009/11/5

流体ソリッドテクスチャリングのための ユーザーインターフェース

Philipp Holzer[†] 大和田 茂[††] 原田 隆宏[†††] Andreas Butz[†]

我々は,流体シミュレーションデータにソリッドテクスチャを適用するためのインタラクティブなユーザーインターフェースを提案する.システムの入力となるのはシミュレーションデータ,および,あらかじめ用意されたソリッドテクスチャの集合であり,テクスチャの混合比率はシミュレーションデータから抽出された局所特徴量によってコントロールされる.ユーザーは局所特徴量と混合比率の対応関係をインタラクティブに設定可能である.また,この設定に基いて,リアルタイムに計算されるシミュレーションデータに,同時にテクスチャを適用することが可能である.また,テクスチャは流体データに沿って流れるため,自然なアニメーションを生成できる.

A User Interface for Solid Fluid Texturing

Philipp Holzer[†] Shigeru Owada[††] Takahiro Harada[†††] Andreas Butz[†]

We propose a real-time solid texturing method on simulated data where feature-texture correspondence is interactively specified by the user. Once the correspondence is given as a preprocess, the feature vectors and the advections are automatically computed from the real-time simulated data, which is then used to synthesize textures. The contributions of our work are as follows: 1) A user interface for texture assignment and 2) Real-time rendering of a textured fluid simulation. Since our system has real-time performance, the user can interactively design and examine the texture assignment.

1. 背景と提案手法の概観

流体にテクスチャを設定することは,シミュレーションデータを美しく見せるために重要なテクニックである.そのため,CG デザイナーは手作業で流体の特徴を強調・抑制するようなテクスチャを設定しながら,より迫力のあるシーンを作成している[1].

一方このプロセスを自動化することは困難である.なぜなら,1. 流体の表面にはテクスチャを適用するための自明なパラメター(uv 座標)が存在しない.また,2. 流体表面は時間の経過に伴って表面形状やトポロジーが大幅に変化するため,時間的コヒーレンスを保つことが難しい.これらの問題点により,流体にテクスチャを適用することは一般に難しい問題であり,リアルタイム処理は今もって実現していない.

そこで我々は,流体の特徴や流れを考慮した上でソリッドテクスチャを貼ることにより,これらの問題を解決することを試みた.ソリッドテクスチャを用いる利点は以下の通りである.

1. 基本的には空間的な位置とテクスチャ座標値が一致するので,パラメター化の問題が存在しない.
2. 既存の二次元ベースのテクスチャ合成を拡張した手法[2][3]と比べて,コヒーレンスを考慮する必要がないので高速実行が可能である.

さらに,高速であるがゆえに

3. ユーザーがリアルタイムにフィードバックを得ながらデザイン作業を進めることが可能である.また,同時にシミュレーションを行うことも可能であるため,シミュレーションのデザインとテクスチャのデザインを並行して行なうことができる.また,ゲーム内でのリアルタイムシミュレーションなど,あらかじめテクスチャを計算しておくことができない用途への応用も可能となる.

2. 既存手法

流体へのテクスチャ適用については,オフラインであれば高品質な結果を出力できるシステムがすでにいくつか提案されている.

[†]Universtity of Munich
[††] Sony CSL
[†††] Havok

Bargteil et al. [2] は流体表面の複雑なパラメター化と二次元テクスチャ合成手法の利用によって,流体表面に時間的コヒーレンスのあるテクスチャの適用を可能にした.Kwatra et al. [3]は最適化アルゴリズムを用い,流体の流れに沿って動くテクスチャの合成を行った.Narain et al. [4]はシミュレーションデータから局所特徴量を計算し,それをガイドとしてテクスチャを合成する手法を提案した.これらの手法は全て二次元のテクスチャ合成手法を流体表面に適用したものであり,リアルタイム実行は困難である.我々の手法はソリッドテクスチャリングを用いているので根本的に異なっている.

我々の手法では,サンプルのソリッドテクスチャを入力としている.このソリッドテクスチャを合成する古典的手法としては,Heeger et al. [5]がアルゴリズム的にエレガントで,実装も簡単なので現在でも様々なところに用いられているが,クオリティはそれほど高くない.近年二次元の Markov Random Field に基づくテクスチャ合成を三次元に拡張したもので高品質かつ高速なものが発表されてきている Kopf et al. [6] Dong et al. [7].これらの手法を用いれば,サンプルのソリッドテクスチャを作成することはそれほど困難なことではない.

我々のシステムのもう一つの入力は流体シミュレーションデータであるが,こちらは数多くの研究があり,見取り図を描くことすら容易なことではないが,大きく分けてグリッドの上で Navier-Stokes 方程式を解く Lagrange アプローチと,数多くの粒子で流体を表現する Eulerian アプローチがある.どちらも近年高速化が顕著であり,また,どちらも原理的には我々のシステムで採用することが可能であるが,Eulerian アプローチを用いた場合細かなパーティクルが飛び散ることがあり,これにテクスチャを適用することはこれまで特に困難であったため,我々は敢えて Eulerian ベースの手法を用いることとした.Harada et al. [8]では,極めて多数のパーティクルをリアルタイムに動かすことが可能である.

3. 提案手法

我々のシステムでは,サンプルとなるソリッドテクスチャの集合と,外部で計算されたシミュレーションデータを入力とする. まずシステムはシミュレーションデータから局所特徴量を計算する.次にユーザーは局所特徴量とテクスチャの対応を指定する.また,特徴がない部分に直接テクスチャを貼りつけ,流れに沿って動くようにすることもできる.本章では,まずユーザーの側からみたシステムを説明し,次に内部で採用しているアルゴリズムの説明を行なう.

3.1 ユーザーインターフェース

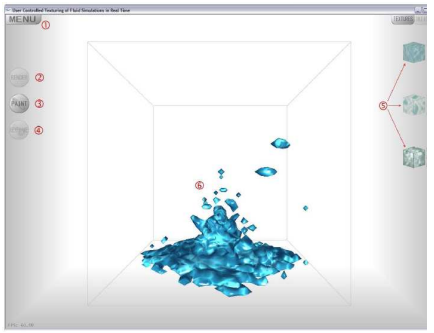


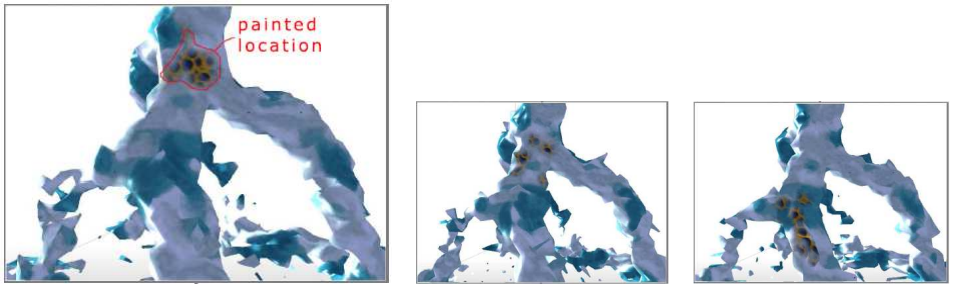
図 1：我々のシステムの外観

我々のシステムの外観を図 1 に示した.丸に数字は説明のために書き入れた番号である.画面左側,①～④まではシステムのモードをコントロールするもので,特に重要なのは,システムはローカルペイントモードと特徴ペイントモードを持っており,この二つをこのメニューで切り換えることができる点である.右側にある⑤は,サンプルとなるソリッドテクスチャである.この例では三種類のサンプルが入力されている.そして,中心に表示されているのがシミュレーションデータである.シミュレータ[8]からの出力はパーティクル集合だが,我々のシステムではこれをメタボール化し,さらに MC 法[9]で表面抽出してから用いている.この処理もリアルタイムに行なうことができる.

ペイント操作

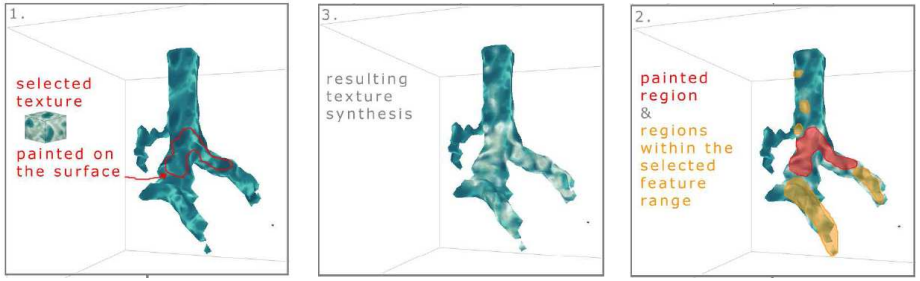
前述のように,我々のシステムにはローカルペイントモードと特徴ペイントモードの二つがある.どちらもサンプルのテクスチャをパレットのように用いてシミュレーションデータ上でブラシのように塗りつけるという点は共通だが,塗りつけたあとの挙動が異なる.

まず,ローカルモードの例を図 2 に示す.(a)のように,ユーザーがペイント操作を行った場所にテクスチャが貼られている.このテクスチャは,シミュレーションを再生し,流れが発生すると,それに伴って移動していく(b,c).



(a) (b) (c)

図 2：ローカルペイントモードでの挙動



(a) (b) (c)

図 3：特徴ペイントモードでの挙動

図 3 は,特徴ペイントモードでのユーザーの操作とその結果の例である.ユーザーがペイントした領域は(a)において線で囲まれた領域であるが,その結果は(b)のようになる.ペイントしていない領域のテクスチャも変更されている理由は,このモードでは,ユーザーがペイントした領域だけでなく,ペイントした領域と似た特徴を持った場所全てにテクスチャを設定するからである.ユーザーが指定した領域,および,その領域と似た特徴を持った領域を(c)に示した.

テクスチャ選択時の付加情報として,テクスチャの拡大率や,用いる局所特徴量の種類,そして,ローカルペイントモードではテクスチャの持続時間を設定できる(テクスチャは設定したキーフレームの後,永久に表示されているわけではなく,ある時間の後に消失させられる).

3.2 実装

我々が流体の局所特徴量として計算しているのは,粒子の密度(Lagrange ベースのシミュレーションデータでは圧力に対応する),粒子の密度を時間で微分したもの,流れの方向ベクトルと表面の法線方向のなす角,それに動きベクトルを時間で微分したものの 4 種類である.この特徴量だけを用いていることに深い意味はなく,[3]のように表面の曲率や速度場のダイバージェンスなどを特徴量として追加してもよい.

我々のシステムに入力されるサンプルテクスチャは,対向面がシームレスにつながるような,タイリング可能なものでなければならない.既存ソリッドテクスチャ合成手法はこのようなテクスチャを簡単に出力できる[5][6].そしてこれらは,ユーザーが指定した拡大率によってスケーリングされたのち,空間全体を埋めつくすように配列される.このようにしておいて,流体の表面を抽出した結果のポリゴンに,サンプルテクスチャを混合した色を塗ることにより流体のテクスチャリングを行なう.混ぜ合わせ比率は位置によって異なるので,空間的遷移を滑らかにするために,入力テクスチャ相互の見た目の特徴がおよそ合っていることが望ましい.

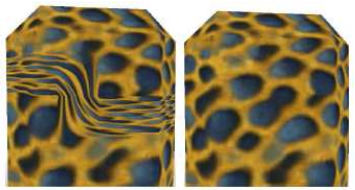
テクスチャの貼りつけと流れを制御するために,四種類の補助的なボリュームデータを用いる.テクスチャ座標ボリューム,ブレンド係数ボリューム,モーションボリューム,および反復コントロールフラグボリュームである.これらを便宜上 Vc, Vb, Vm, Vr と呼ぶ.これら四種のボリュームデータは,シミュレーションに用いるバウンダリボックスにちょうど一致するように配置されており,その解像度は 32^3 から 128^3 の程度である.レンダリング時には三次元テクスチャデータとして GPU に送られる.以下にそれぞれの機能を説明する.

テクスチャ座標ボリューム(Vc)は単純に,各要素が三次元ベクトルであり,ソリッドテクスチャの uvw 座標を保持しているボリュームデータである.この座標は,シミュレーションの結果の流体の動き(以下に説明する Vm)によって動かされる.つまり,シミュレーション結果に応じて流れるのは,uvw 座標である.

ブレンド係数ボリューム(Vb) はサンプルテクスチャの数の次元のベクトルを各要素に持つボリュームデータであり,その位置でのテクスチャのブレンド係数を保持している.GPU がレンダリングする際にこの値を参照し,重みつき和を計算してレンダリング結果の色として出力する.

モーションボリューム(Vm) は,各要素が三次元の流れベクトルを保持しているものであり,このデータはシミュレーションデータから直接にセットされるが,速すぎる流れはクランプされる.

反復コントロールフラグボリューム(Vr) は,テクスチャが流れにそって動かされていき,テクスチャ座標が乱れた時に補正をするためのバイナリ情報を保持するボリュームデータである.例えば Vc のあるボクセルの,一つの頂点がテクスチャ座標(0.1,0,0)を持っていたとし,その隣の頂点が(0.8,0,0)を保持していたとする.すると,この二点間 u 座標の補間は,0.1…0.2…0.6…0.8 となるべきか,それとも 0.1…0.0, 1.0…0.9...0.8 となるべきか,二通りの可能性がある.この問題を考慮しないと,図 4(a)のように,不連続ではないものの,テクスチャの一部が歪んだ外観になってしまうことがある.どちらの補間を用いるかを保持するのが Vr である.



(a) (b)

図 4：Vr の効果

Vr は Vc に対して双対,すなわち,Vc がボクセルをグリッドととらえ,その頂点で値を保持するのに対し,Vr ではボクセルの中心位置で値を保持している.そして,Vc を補間するさいに,Vr の値が 0 であれば,テクスチャ座標が 0 をまたがない通常の補間,1 であれば 0 をまたぐ補間を行なうこととする.これにより,図 4(b)のように,歪みのない外観を保持することができる.なお,Vr は各軸方向に独立したデータを保持する三次元ベクトルのボリュームデータである。

流れの実現 テクスチャ座標は Vm によって更新され,それによってテクスチャの流れが実現される.Vm の値の分だけ,Vc,Vb,Vr をオフセットすればよい.

ユーザーインターフェースとの関係 ローカルペイントモードの場合は,ユーザーは Vb をペイントにより変更する.変更した箇所は,Vm により自動的に流れに沿って動いていく.特徴ペイントモードの場合は,システム内に特徴量からブレンド係数へのテーブルが存在し,これを変更することになる.ユーザーがセレクトした領域の各特長量の最大値と最小値のレンジの中に入る特徴量を持つ領域の Vb が間接的に影響を受けることになる.

4. 結果

図 5 は我々のシステムの生成した結果画像である.これらの全ての結果画像において,Vc～Vr は 32^3 である.図 5 の(c,d)は NPR スタイルでのレンダリングを試みている.これまでの流体テクスチャリング手法は空間的コヒーレンスを考慮しながらシンセシスしていたために計算コストが非常に高かったが,我々のシステムではほとんどの場合に秒間 40 フレーム以上のテクスチャ適用が可能であるため,シミュレーション計算をもリアルタイムで同時に実行することが可能である.図 5 に示したものは,全てそういった例である.

表 1 に実行時間のデータを示した.ここでは,テクスチャリングの時間に併せてシミュレーションに要した時間,および,表面抽出等も含めたトータルのフレームレートも示している.シミュレーションに用いたパーティクルの最大数は 10,000 点であり,Vc～Vr のサイズは 32^3 である.実験に用いた PC は Intel Core 2 Duo 2.2 GHz, GPU は GeForce 8600M GT である.ここからわかるように,我々のシステムは全てを総合して 20FPS 以上のパフォーマンスを出すことが可能である.

5. 結論と今後の課題

本稿では,流体シミュレーションデータにテクスチャを適用するためのインタラクティブなユーザーインターフェースを提案した.我々のシステムではシミュレーションの同時実行が可能なパフォーマンスを持っているため,流れのデザインとテクスチャのデザインを同時に行うことができる.これにより,シミュレーションデザイナーの負荷を減らすだけでなく,ゲームなどでリアルタイムにシミュレーションを行なう用途にも利用可能であると考えられる.

課題としては,まず解像度の問題がある.流体表面の凹凸が激しいのは,表面抽出アルゴリズム(MC 法)の解像度の問題であり,また,現在では Vc～Vr の解像度も低いため,細かな設定ができない.これらの解像度を高めると,品質は向上するが実行速度が大幅に低下する.

また,流れが広がるような場所でテクスチャの歪みが強くなるという問題もある.これは,我々の手法がテクスチャを混ぜているだけであり,合成を行っていないというところに起因しているが,これはスピードとトレードオフである.

参考文献

1) Mahash Ramasubramanian: Animating Waves on the Beaches of Madagascar. Siggraph 2005 Sketch, 2005.
2) Vivek Kwatra, David Adalsteinsson, Theodore Kim, Nipun Kwatra, Mark Carlson, Ming Lin: Texturing fluids. IEEE Trans. Visualization and Computer Graphics (TVCG) , 13(5), pp.939-952, 2007
3) Rahul Narain, Vivek Kwatra, Huai-Ping Lee, Theodore Kim, Mark Carlson, Ming Lin: Feature-Guided Dynamic Texture Synthesis on Continuous Flows. Proc. EGSR 2007, pp.361-370, 2007.
4) Heeger D. J., Bergen J. R.: Pyramid-based Texture Analysis/Synthesis. Proc. Siggraph 1995, pp.229-238, 1995.
5) Yue Dong, Sylvain Lefebvre, Xin Tong, George Drettakis: Lazy Solid Texture Synthesis. Proc. EGSR 2008.
6) Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, Tien-Tsin Wong: Solid Texture Synthesis from 2D Exemplars. ACM Transactions on Graphics (Proc. SIGGRAPH 2007), 26(3), 2007
7) Takahiro Harada, Seiichi Koshizuka, Yoishiro Kawaguchi: Smoothed Particle Hydrodynamics on GPUs. Proc. CGI (2007), pp. 63-70, 2007.
8) Wiliam E. Lorenson, Harvey E. Cline: Marching Cubes: A high resolution 3D surface construction algorithm. Proc. Siggraph (1987), pp.163-169, 1987.

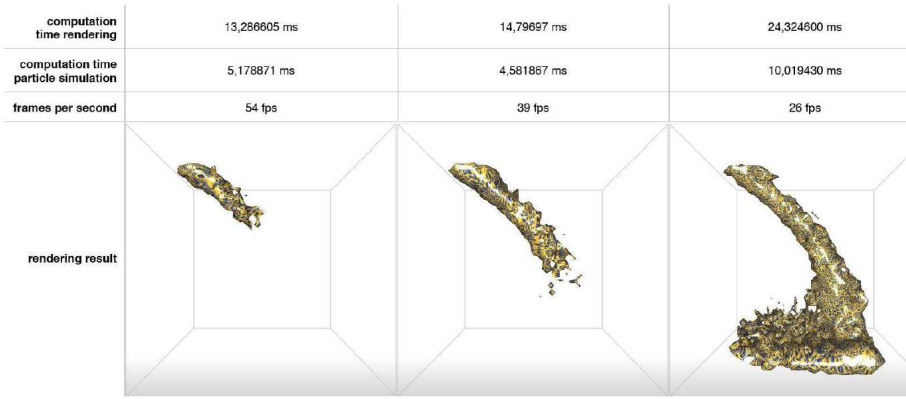
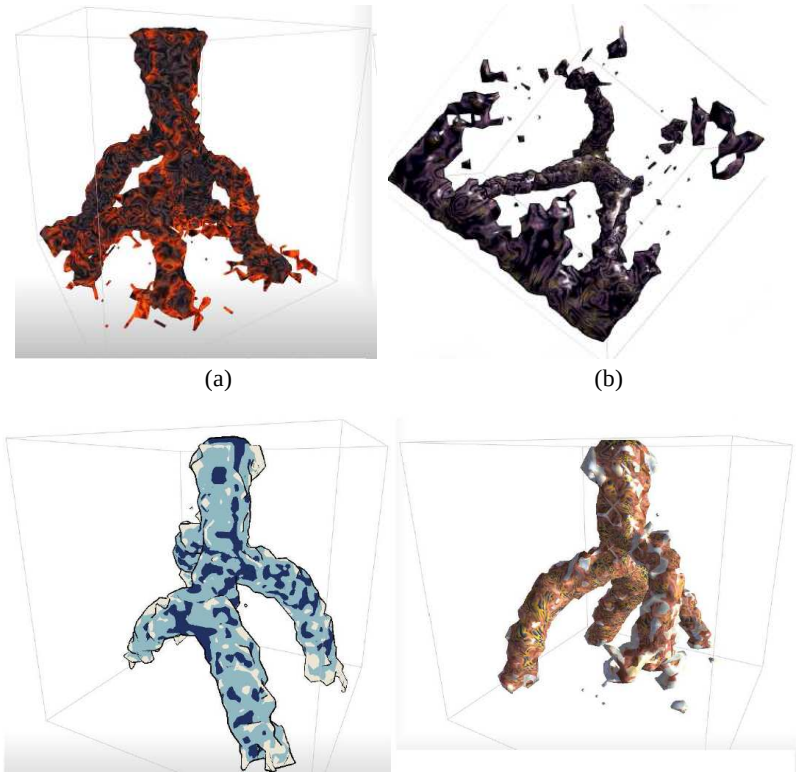


表 1：実行時間

Vol.2009-CG-137 No.5
2009/11/5



(a) (b)

(c) (d)

図 5 結果画像