2004－CG－116　(4)
2004／8／18

画像の点集合表現を用いた、歪みのあるテクスチャの高品質な合成手法

大和田 茂†‡　Frank Nielsen†　五十嵐 健夫‡

†ソニー コンピュータサイエンス研究所　‡東京大学

歪んだテクスチャの合成において高品質な結果が得られるアルゴリズムを提案する。品質劣化の大きな原因はリサンプリングであるが、本手法では、元画像のサンプリング点を保存した非一様なピクセル配置をもつ画像を導入することにより、品質劣化なくパッチを貼りつけていくことを可能にした。また、パッチの連結は、Graphcut 法を非一様なピクセル配置画像向けに拡張して用いた。

High Quality Distorted Texture Synthesis using Point Representation of Images

Shigeru Owada†‡　Frank Nielsen†　Takeo Igarashi‡

† Sony Computer Science Laboratories, Inc.　‡The University of Tokyo

We propose a new distorted texture synthesis algorithm that produces high quality result. The main cause of degraded output is fixed pixel layout of the intermediate and final images because those representations require resampling of an original source image. Throughout the process, we represent images as a set of individual points that maintain original sample points. Each pixel is transferred by the predefined distortion field and then combined by Graphcut algorithm, extended to support irregularly sampled images.

1. 背景

テクスチャ合成(Texture Synthesis)アルゴリズムは、サンプル画像を入力として、その特徴をとらえつつ、継ぎ目なくサイズを自由に変更することを可能にする技術である。近年三次元物体上で合成する[Wei01]、複数の画像を混合する[Zhang03]、あるいは、ボリューム構造を近似する[Owada04]などの様々な応用が提案されているが、その際に入力画像を大きく変形して生成を行うことがある。既存手法ではターゲットのピクセルの配置(解像度)を一定としているものが多いため、変形が大きい場合にはリサンプリングにより画質が劣化し、近傍探索や最終的な画像生成に悪影響を及ぼす。そこで、我々はリサンプリングを行わず、元画像のサンプリング点をそのまま保持することにより、画質の劣化がないテクスチャ合成を可能にする手法を提案する。

2. 既存手法

テクスチャ合成アルゴリズムは、古くは統計的パラメータを抽出したり、周波数空間で演算を行うものなどが存在したが、現在ではソース画像から直接画像の一部をコピーしてターゲット画像を生成する手法が一般的である。それらは大きく分けて、1 ピクセルずつ貼りつけていくピクセルベースの手法と[Efros99]、領域を一気に貼りつけるパッチベースの手法がある[Xu00;Efros01]。それぞれ特徴があるが、画像を大きく変形させながら生成を行う場合にはピクセルベースの手法が使われる事が多い[Wei01;Zhang03;Owada04]。それは、ピクセルベースの手法の方が局所的な変形をアルゴリズムに自然に組みこめるからだと思われる。しかしながら、ピクセルベースの手法は構造がぼやけやすいという欠点がある。さらに、変形を伴う場合には繰り返しリサンプリングを行うため、さらに画質が低下する。パッチベースの手法は、空間的コヒーレンスが保持されるため一般的に言ってピクセルベース手法よりよい結果を出力することが多い。パッチ境界で構造が不連続になりやすいという問題も解決方法が提案されてきている[Wo04]。そこで我々は、パッチベースの手法を歪んだテクスチャ合成に適用し、さらに、既存手法では避けられなかったリサンプリング[Fang04]を防ぐことにより、元画像の画質を保つ手法を提案する。

3. アルゴリズム

本手法では、歪み場を入力として、パッチを変形しながら一枚づつ貼っていく(図 1)。歪み場は、本稿では各点に拡大率と方向が定義されたものを考えることとした。パッチの変形は、中心に近いピクセルから順に位置を決めていくことにより行う。変形されても各ピクセル値はリサンプリングされることなく、ピクセル(サンプリング点)の位置だけが歪み場によって変更され、新たな位置が浮動小数値として格納される。この非一様なピクセル配置を持つ画像を、以下 ISI(Irregularly Sampled Image)と呼ぶことにする。そして、最後にそれらを、拡張した Graphcut 法[Kwatra03]を用いて結合する。

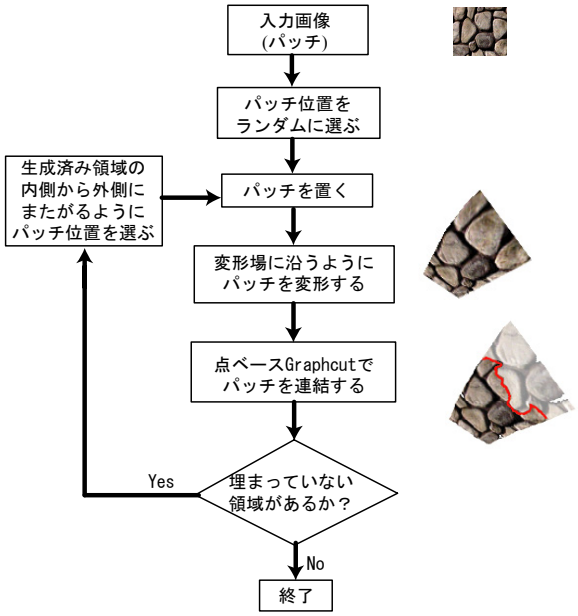


図 1 アルゴリズム全体の流れ

以下に、各ステップを細かく説明するが、その前に離散的な点集合と連続的な領域とを対応づける α-shape について述べる。

3.1 α-shape

α-shapeは点集合と定数 α とを入力として計算される閉じた領域で、点集合の大まかな形状を表すものである。直観的には、直径 α の円を使って、その円の内部に入力点が入らないように領域を削っていった結果の形状である[1] (ただし、最外側の点は折れ線で繋ぐ)。我々は、点集合の Delaunay三角形分割から、α より長い枝を除き、さらに、残った枝のうち、左右の領域がどちらとも三角形をなさないようなものも除いたものを用いる。これは、α-shapeを三角形分割したものとも言えるが、以下の文脈では、この三角形メッシュ自体を α-shapeと呼ぶことにする。このメッシュは、点集合を画面に表示する際にも有効である。点集合をそのまま点プリミティブで表示すると各点のサイズや形状が問題となるが、三角形ポリゴンとして表示すれば点の間の空間はラスタライザにより自動的に埋められ、色の補間もハードウェアが行うからである。

3.2 パッチ中心位置の決定

本稿では、パッチを貼りつける位置の決定は(生成済み領域の内側と外側をまたぐ、という制限はあるものの)ランダムに行っている。従って、生成済み領域との連続性が問題になる場合がある。これを防ぐには、変形後のパッチと、生成済み領域との類似度を測定し、最適な位置を選ぶ必要があるが、これはあまりに計算コストがかかるため行っていない。ただし、変形に制限を設けるなどによって FFT などの高速化手法が適用できるようになる可能性はある。これは今後の課題とする。

3.3 パッチの変形

パッチ中心位置を決定したら、次に変形場に沿うようにパッチを変形する。これには、中心に近いピクセルから順に変形後の位置を決めていく。具体的なアルゴリズムを擬似プログラムで示すと次のようになる。

```
function DistortPatch( Patch patch,DistortionField distortField)
  currentPoint = CenterOfPatch(patch)
  currentPoint.applyDistortion
    = distortField.Eval( currentPoint.position )
  priorityQueue.push( currentPoint )

  while( priorityQueue is not empty )
    currentPoint = priorityQueue.GetTopAndErase()
    if( currentPoint.visited ) continue

    initialPosition
      = PredictPosition( VisitedNeighbors(currentPoint))
    currentPoint.position
      = LocalOptimization( initialPosition
                          ,VisitedNeighbors(currentPoint)
                          ,distortField )
    currentPoint.visited = true
    priorityQueue.push( UnivisitedNeighbors(currentPoint))
```

PredictPosition() 関数は、周囲の生成済みの点に適用された変形結果をもとに、現在の点の位置を予測する。点の位置は最終的には LocalOptimization()関数による最適化の結果決められるため、実際に適用された変形は、必ずしも同位置での変形場の値とは一致しない。LocalOptimization()関数は、実際に適用される変形と、変形場の値の差を最小にするように

[1] α-shapeの定義は、実際には複数存在するが[Edelsbrunner95]、我々はシンプルな上記の定義を用いることとした。

currentPoint の位置を決定する。このために、*currentPoint* の x,y 座標を変数とするエネルギー関数を定義する。このエネルギー関数が最も減少する向きに x,y を動かす。この計算は解の変化が小さくなるまで反復しなければならないが、実際には 3 回から 6 回の反復で十分である。
このエネルギー関数は、次の形をしている。

$$E = (\text{拡大率の誤差}) + \gamma\,(\text{回転角度の誤差})$$

ここでγはユーザーが指定する重みである。右辺の各項は次のように計算される。

拡大率の誤差 *currentPoint* における拡大率は、α-shape において *currentPoint* に連結している点への距離を平均したものを用いて計算する。この距離は、元のパッチ内での距離と、現在変形中のパッチ内での距離があり、この比が現在の拡大率である。この比と、変形場から与えられる拡大率の二乗誤差が拡大率の誤差であると定義する。

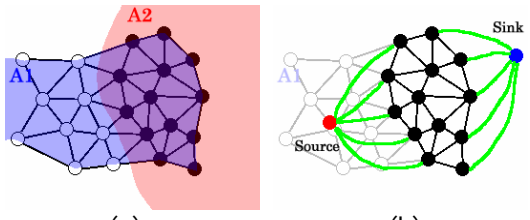
回転角度の誤差 これも同じように、α-shape において連結している各点の方向の差(元のパッチ内での方向と、現在変形中のパッチ内での方向)と、変形場から与えられる方向の二乗誤差が回転角度の誤差であると定義する。

なお、このプロセスは非常にコストが大きく、計算時間の大半を占める。将来可能な高速化手法としては、貼りつけるパッチサイズを小さくし、中心位置での変形のみを用いて全体に一度に変形するという方法が考えられる。こうすることで、3.2 節の最後で述べたパッチのサーチも組み込むことができると考えられる。

3.4 パッチの連結

パッチを変形したら、実際にそのパッチを、すでに生成された領域に貼りつける。この際の手法としては、重なった領域をブレンドする(Feathering)、または、最適な混合ラインを見つけて組み合わせる(Seam optimization)、という、主に二通りの方法が存在する。ブレンドする方法では境界がぼやけたり、構造がだぶったりする欠点があるので、我々は[Kwatra03]で提案された Graphcut 法を、ISI に拡張して用いることとした。入力は二枚の ISI であり、出力は混合後の一枚の ISI である。この手法の概要は次のようになる。

1. まず、図 2(a)において、A1 がこれから貼りつけたいパッチ、A2 がすでに生成が終わった領域とする。これらはα-shape がすでに求まっているとする。従って、それぞれ三角形メッシュである。便宜上 A1 の方のみ、そのメッシュ構造を描画している。
2. A1 に含まれる点のうち、A2 上にもある点を選ぶ。これを Vm と呼ぶ(図 2 における黒丸)。
3. Vm の点にコストを設定する。コストはその点の色と、A2 内にある一番近い点の色との差である。本研究では、RGB 空間内での L2 ノルムを用いることとした。
4. A1 のメッシュ構造をグラフのネットワークととらえ、両端点とも Vm に含まれる枝のみを取り出してサブグラフ G12 を作成する。この各枝にもコストを設定する。このコストは両端点のコストの和である。
5. G12 に、source ノードと sink ノードを加える(図 2b)。
6. A1 に含まれる枝のうち、片方の端点のみが Vm に含まれる枝を G12 に加える。Vm に含まれない方の端点は、そのまま source ノードに接続する。この接続する枝のコストは∞とする。この時この新たな枝に接続された Vm 内の点にはマークをしておく。
7. G12 の点のうち、A1 内で最外側の点であり、かつ、前ステップでマークされなかった点を sink ノードに接続する。この枝のコストは∞とする。
8. source から sink に至るフローを考え、グラフカット法を適用する。この結果は、ノード Vm を source 集合 Vs1、と、シンク集合 Vs2 に分類する。Vs1 を結果集合 Vr に加える。
9. Vs1 のα-shape を求める。A2 内の点のうち、このメッシュ上に乗っていない部分を Vr に加える。
10. A1 内の点のうち、A2 上にない点を Vr に加え、Vr を出力する。



(a) (b)

図 2 ポイントベース Graphcut の説明

このポイントベース Graphcut を行った例が図 3 である。パッチの局所変形は行っていないが、それぞれ適当に回転させてから貼りつけている。拡大すると(図 3c)、領域ごとにピクセルの配置が違うパッチが連結されていることがわかる。

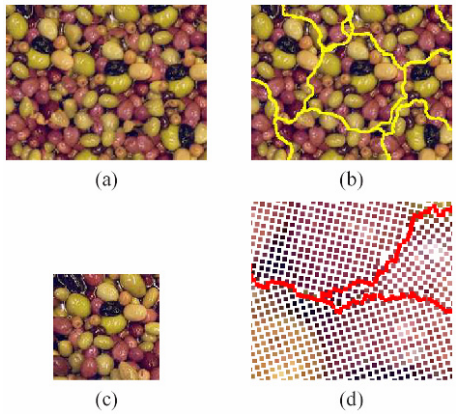


図 3 ポイントベース Graphcut の例。(c)が入力画像であり、(a)(b)は結果である。(d)は(b)を拡大したものである。

結果

我々は変形のための場関数を定義するユーザーインターフェースを作成した(図 4)。このツールは、ユーザーが二本の線を描くと(図 4a)それぞれ 0 と 1 の値が割りあてられ、空間内で線型補間してグレイスケールの場が作られる(図 4c)。各点のグラディエントの大きさの逆数が拡大率を定義し、グラディエントの向きが回転方向を定義する。

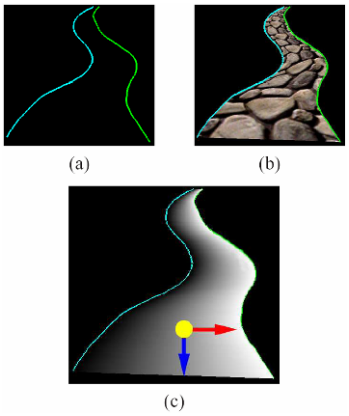


図 4 場関数定義ツール

このツールを使ってテクスチャ合成を行った例が図 5 と図 6 である。

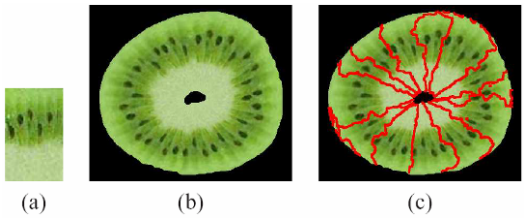


図 5 キウイの例。

図 6 フォントデザインに応用した例

考察と今後の予定

本稿では、歪みのあるパッチベースのテクスチャ合成を、リサンプリングを行わずに行う手法を提案した。これによって画質を犠牲にすることなく画像を歪ませることが可能になった。
今後は変形パターンの制限による高速化、およびパッチ位置のサーチを組みこみたい。また、点集合の生成は、画像以外にも点の分布(植生図など)の生成等、様々な応用が考えられる。また、ここで提案した特殊な Graphcut は誤差の少ない画像連結手法として、パノラマ画像の生成にも用いることができると考えられる。このような応用についても検討したい。既存の手法との比較も今後の課題の一つである。

References

[Edelsbrunner95] Edelsbrunner, H. “The Union of Balls and its Dual Shape”, Descrete Computer Geometry 13(1995), 415-440
[Efros99] Efros, A.A. and Leung, T.K. “Texture Synthesis by Non-parametric Sampling”, proc. ICCV 1999, 1033-1038
[Efros01] Efros, A.A. and Freeman, W.T. “Image Quilting for Texture Synthesis and Transfer”, Proc. Siggraph 2001, 341-346
[Fang04] Fang, H. and Hart, J.C. “Textureshop: Texture Synthesis as a Photograph Editing Tool”, proc. Siggraph 2004
[Kwatra03] Kwatra, V., Schödl, A., Essa, I., Turk, G. and Bobick, A. “Graphcut Textures: Image and Video Synthesis using Graph Cuts”, proc. Siggraph 2003, 277-286
[Owada04] Owada, S., Nielsen, F., Okabe, M. and Igarashi, T. “Volumetric Illustration: Designing 3D Models with Internal Textures”, proc. Siggraph 2004
[Wei01] Wei, L-Y. and Levoy, M. “Texture Synthesis over Arbitrary Manifold Surfaces”, proc.Siggraph 2001, 355-360
[Wo04] Wu, Q. and Yu, Y. “Feature Matching and Deformation for Texture Synthesis”, proc. Siggraph 2004
[Xu00] Xu, Y-Q, Guo, B. and Shum, H-Y “Chaos Mosaic: Fast and Memory Efficient Texture Synthesis”, Technical MSR-TR-2000-32, Microsoft Research, 2000
[Zhang03] Zhang, J., Zhou, K., Velho, L., Guo, B. and Shum, H-Y. “Synthesis of Progressively-Variant Textures on Arbitrary Surfaces”, proc.Siggraph 2003, 295-302