

内部構造のある三次元物体のモデリング

Volumetric Illustration: Designing 3D Models with Internal Textures

大和田 茂†‡ フランク ニールセン† 岡部 誠‡‡ 五十嵐 健夫‡§
Shigeru Owada Frank Nielsen Makoto Okabe Takeo Igarashi

†東京大学

‡ソニーコンピュータサイエンス研究所

§PRESTO/JST

The University of Tokyo

Sony CS Laboratories Inc.

PRESTO/JST

論文要旨

物体の内部構造をモデリングするユーザーインターフェースおよび、新しいデータ構造を提案する。ユーザーはあらかじめ区分けされたポリゴンモデルと二次元画像を入力として直観的に内部構造を作成でき、作成したモデルは自由に切断することができる。この枠組みでは、システムは完全なボリュームデータを保持するのではなく、二次元画像と三次元のマッピング関数だけを持ち、ユーザーにより新しい切断面が指定されるごとにテクスチャシンセシス技術を用いて適当な切断面画像が生成される。そのため、ユーザーは切断しかできないものの、コンパクトなデータから高品質の切断面画像を生成することができる。

背景および目的

実世界にある全てのものは、例外なく三次元的な内部構造を持っている。しかしコンピュータグラフィックスにおいては、現在ではサーフェスモデルが最も広く使われている。このデータ構造は、静止画や映画などユーザーと物体との動的な対話性がない場合であれば破綻なく画像を生成できるが、より動的なコンテンツには適さない。例えば生物・医学教育やバーチャル博物館などにおいては、ユーザーが自由に物体を切断できて然るべきだが、サーフェス情報しか持たなければこれは難しい。

そのような内部情報を持つデータ構造としてはボクセルなどのボリュームデータ構造があるが、コンテンツを製作するためにはモデリングを行う適当なモデラーが必要となる。しかしながら、これを行う効率的なモデラーは、現実的には Perlin らによるシステム [Perlin85] をルーツとする関数ベースの手法以外には実用レベルに達しているものは存在しない。関数ベースの手法は直観性を欠き、生成できる構造も限られることから、我々は、通常のマウスを用いて直観的にボリュームモデリングを行うシステムを提案する。とりわけ、我々は物体の切断面に着目し、インターフェースおよびデータ構造の両面から、なるべく二次元的な処理を行うシステムを提案する。これにより、特殊な入力デバイスなどを必要とすることなく直観的にモデリングを行うことができ、データ量も少なくすることが可能である。

既存研究

体の内部構造をユーザーが手作業でモデリングするシステムとしては、Perlin らによる Solid Texturing と呼ばれる手法を拡張したものが最もよく使われている [Perlin85]。この手法では、ユーザーは空間座標を入力とし、色やその他の属性を出力する関数をプログラミングし、そこに適度にランダム性を加えることにより、リアルな構造をモデリングできる。この手法はある意味モデリングの詳細部分を乱数関数に任せることによって詳細でリアルな構造のモデリングを可能としているが、WYSIWYG 的でないという意味で直観性を著しく欠いている。期待する構造を作成しようと思っても、それを関数として表現できなければ作成できない。その欠点を克服するために、あらかじめほとんどモデリングされたライブラリを供給したり、サーフェスモデルから幾何情報をとりこむなどの手法が提案されているが [Cutler02]、根本的な問題は解決していない。

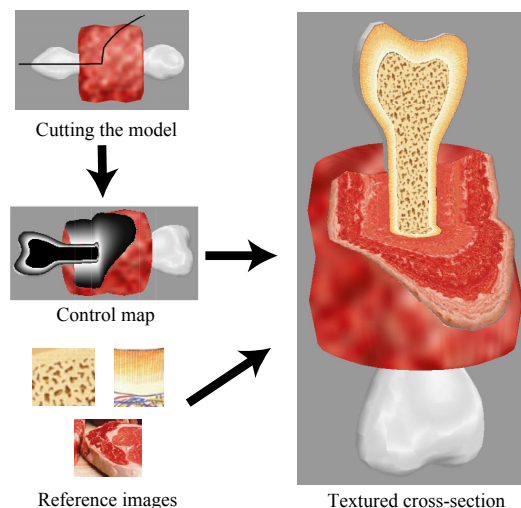


図 1 : システムの概要

提案手法

本稿ではあらかじめ形状情報をポリゴンデータとして取り込み、その切断面に二次元画像を貼りつけることによって内部構造をモデリングする手法を提案する。この枠組みでは完全なボリュームデータは存在せず、ユーザーが物体を切断するごとに逐次切断面が生成される。そのため、ユーザーは切断しかできないものの、少ないデータから高品質の切断面を生成することができる。また、通常図鑑などでは情報を効率的に伝えるために、実際にはあり得ないほど構造が明確にされた恣意的な断面が描かれることが多い。例えば図 2 の葉の断面イラストの例では、切断された全ての細胞に核が見えているが、実際の三次元物体を切断した場合にはそのように整然と構造が見えるような切断面は存在しない。我々のシステムではボリュームデータを明示的に持たないことによって、このような恣意的な断面を任意の入力に対して生成することが可能である。

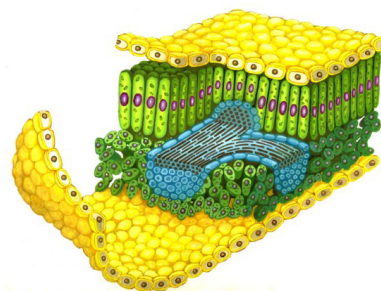
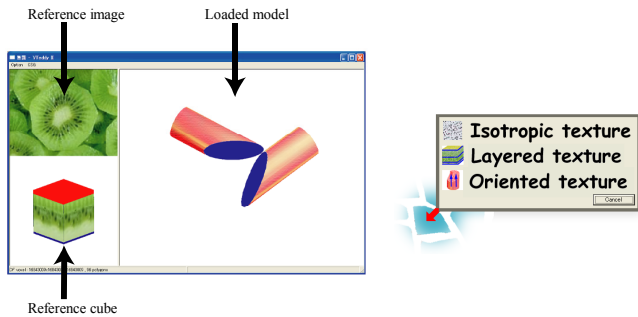


図 2 : 内部構造のイラストレーション(提供 : 佐藤佐恵子)

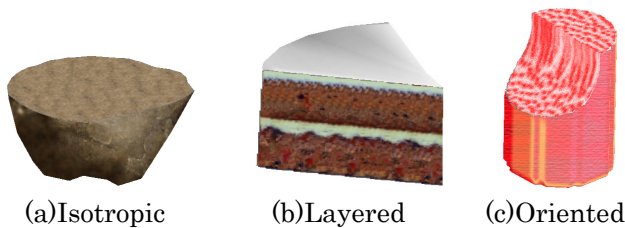
ユーザーインターフェース

ユーザーは、まずあらかじめ領域分けされたポリゴンデータを外部で作成する。次に物体を切断し、モデリングしたい領域を指定した上で二次元画像データを読みこむ。対応づけを行うシステムの外観は図 3a のようになっており、右側のペインに三次元物体、左上に読みこんだ二次元画像、左下には二次元画像と三次元空間の対応関係を視覚化するための Reference cube が表示される。



(a) エディット画面 (b) テクスチャ選択ダイアログ
図 3 : 我々のシステムの外観

画像と三次元領域の対応を指定する際には、三種類の貼りつけ方が選べる。方向性のない **Isotropic** テクスチャ、層状の構造を持った **Layered** テクスチャ、そして、繊維のように方向性を持った **Oriented** テクスチャである(図 4)。これらの指定は、テクスチャが未設定の領域をクリックすると選択ダイアログが出るので(図 3b)、ここで望みの種類を選択すればよい。



(a)Isotropic (b)Layered (c)Oriented
図 4 : テクスチャの種類

これら三種類で全ての物体が表現できるわけではないが、適切に領域分けを行えば、これだけで多くの物体を表現できる。貼りつけ方の種類を選んだら、次にそれぞれの種類に応じて必要とされる情報を与える。

Isotropic texture の割りあて

Isotropic texture は方向性を持たないので、読みこんだ画像中の領域をラバーバンドで選択すると、直接それが **Reference cube** と三次元物体の断面に割りあてられる。断面の解像度は固定(150×150~400×400)なので、断面画像の大きさを変えたい場合は、入力画像をスケーリングする必要がある。

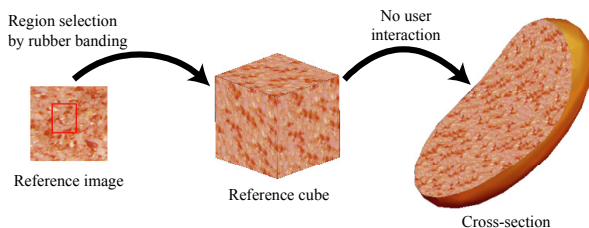


図 5 : Isotropic texture 設定の流れ

Layered texture の割りあて

Layered texture は層の深さに応じたテクスチャが貼られるので、深さ情報をユーザーが指定してやる必要がある。このため、ユーザーはまず読みこんだ二次元画像の上で二本のストロークを描くことで層の一番浅い場所と一番深い場所を指定し、使用する領域を切り出す(図 6 左)。すると、上層と下層をそれぞれ上面、下面とした **Reference cube** の側面が生成される(図 6 中)。次に、ユーザーは **Reference cube** の上面、底面が三次元物体の中でどこに位置するかを指定する。これは、**Reference cube** の面をクリックしたのちに、対応する場所を三次元物体中でクリックもしくはドラッグすることによって行う(図 6 右)。領域の境界をクリックすると、その境界全体(三次元的な境界。サーフェスとなる)が束縛条件として指定され(図 7a)、断面のドラッグは、曲線状の束縛条件となる(図 7b)。

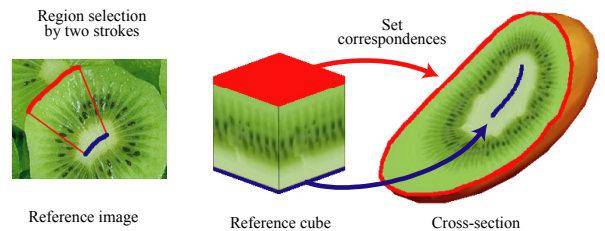


図 6 : Layered texture 設定の流れ

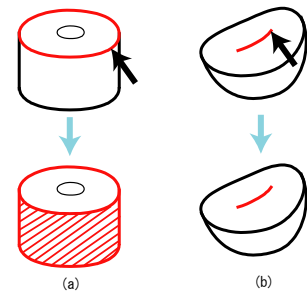


図 7 : Layered texture における束縛条件の指定

Oriented texture の割りあて

Oriented texture では、まず二次元画像中の領域をラバーバンドで選択すると、それが **Reference cube** の上面に設定され、それが自動的に高さ方向にスイープされる。三次元物体中では流れの方向(**Reference cube** では高さ方向)の指定が必要である。これは、断面上をドラッグして矢印を描くことにより行う(図 8)。

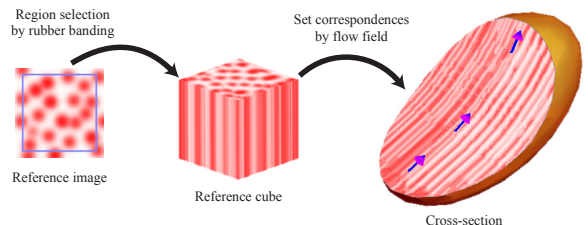


図 8 : Oriented texture の割りあて

ブラウザ

対応づけが終われば、ユーザーは自由な平面で物体を切断し、断面を見ることができる。物体を横切る自由曲線を描くと切断され [Igarashi99]、自動的に開かれる [Owada03]。

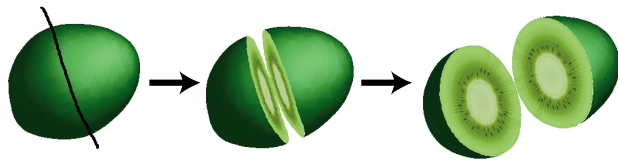


図 9 : 物体の切断

アルゴリズム

我々の手法のデータ構造は、二次元画像および三次元のマッピング関数からなる。マッピング関数は Layered および Oriented テクスチャで必要とされ、モデリング時のユーザー入力を補間することにより作成される。物体が切断されると、まずマッピング関数とその切断面上でサンプリングされ、二次元画像が作られる。この画像は Control Map と呼ばれ、入力二次元画像の貼りつけ方を制御する。Control Map は図 1 では濃淡画像として表現しているが、実際にはテクスチャの種類によって、各ピクセルが保持する値の意味は異なる。以下に各テクスチャのアルゴリズムの詳細を述べる。

Isotropic texture

Isotropic テクスチャでは、構造が方向に依存していないため、Control map にも特に値は保持しなくてよい。断面上で通常のテクスチャシンセシスを行えばよい [Efros99; Wei00]。

Layered texture

Layered の時には、まずユーザーが二次元画像中に描きこんだ二本の線から領域を切り出し、また、その線を補間して、二次元の「深さ場」を作る(図 10 左)。この補間には、テクスチャの一番浅い領域を表す線に 0 の値を、一番深い領域を表すもう一方の線には 1 の値を与え、Turk らによる Thin-plate interpolation を用いて補間する [Turk99]。さらに、三次元形状の方でも指定した深さ情報を束縛条件として三次元の「深さ場」を作る。こちらにも、Turk らによる三次元版の補間法により三次元的に補間する [Turk99]。ユーザーが物体を切断したら、この関数を断面上でサンプリングする。サンプリングされる補間関数はスカラー値を返す関数なので、Control Map の各ピクセルに対し層の深さを表す float 値一個が保持されることになる(図 10)。

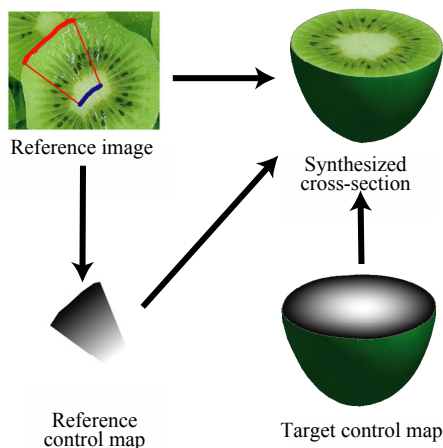


図 10 : Layered texture 生成の流れ

Reference image、二次元の深さ場 : 図 10 - Reference control map、三次元物体の二次元の断面:図 10 - Target control map)から断面画像を生成する手法は、Field distortion synthesis と呼ばれる既存の手法に類似しているが [Turk01; Zhang03]、以下の点で異なっている。

1. テクスチャの方向は、Target control map の gradient 方向として計算される。
2. Reference image 中の各ピクセルも、Reference control map の gradient として計算される方向を持っている
3. ピクセルの生成順は、Target control map において値の小さいものから順に行われる。
4. ピクセル生成では類似した近傍を Reference image の中から探す、その時の Reference image における中心ピクセルの深さ値は、現在生成中のピクセルの深さ値(Target control map の値)と同じ値でなければならない(実際には、float の値を 32 段階に離散化した上で比較される)。

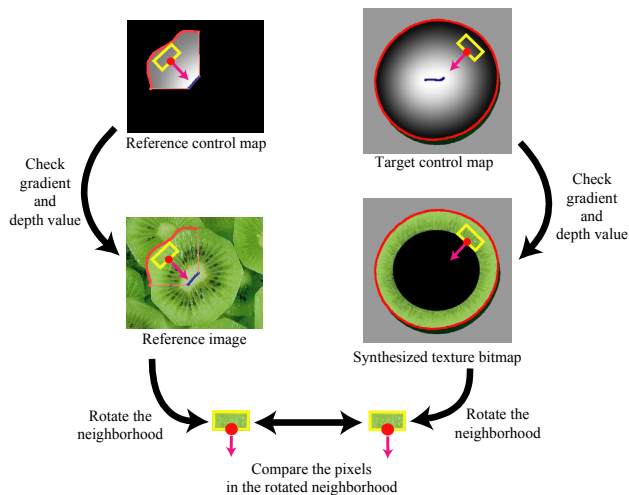


図 11 : Layered texture 生成の詳細

Oriented texture

Oriented の場合は、(二次元の画像でなく)三次元の参照用ボリュームと、繊維の方向を示す流れ場が必要である。すでに述べたように、参照用ボリュームは二次元的にシンセシスされた Reference cube の上面を縦方向にスイープすることによって作られる。この参照用ボリュームのサイズは、我々のシステムでは $64 \times 64 \times 64$ とした。ユーザーが入力した矢印から三次元の流れ場を計算するには、再び Turk らの三次元の補間法を、 x, y, z の各要素について適用し、それらをまとめ正規化することにより行う。ユーザーが物体を切断すると、断面に対する相対的な流れ場ベクトルの向きが二つの float 値として保持される。この値を用いて、参照ボリュームの中に二次元のサンプリング平面を設定し、二次元画像を作る(図 12)。この画像を通常のテクスチャシンセシスの参照用画像として用いる。従って、Oriented テクスチャの断面画像生成時には、各ピクセルごとに異った参照画像が作成されることになる。我々の実装では高速化のために断面画像のキャッシュを行っているので、画像生成にかかるオーバーヘッドはさほどでもない。

実際にこれらの三つの情報(入力二次元画像 : 図 10 -

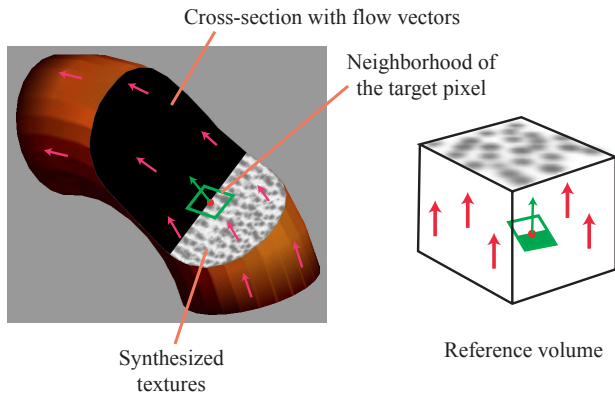


図 12 : Oriented texture における、参照用ボリュームから、断面生成に用いる二次元画像を作り出す例

さらに、各テクスチャは多重解像度を用いて生成を行っているため、最も低い解像度の画像が作成できた段階で表示をスタートさせ、より時間のかかる高い解像度の画像は別スレッドで計算して逐次更新することにすれば、ユーザーはストレスなくエディットを行うことができる。

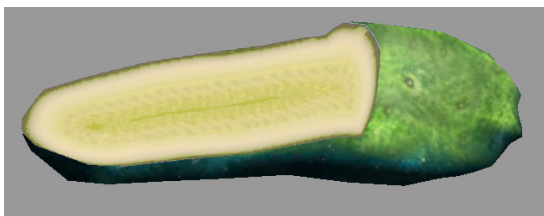
結果

本システムを用いて、図 13 のような内部構造を持ったモデルが作成できた。ボリューム構造を明示的に持たないので、データ量が小さくおさえられる(表 1)。断面画像の解像度はおよそ $150 \times 150 \sim 400 \times 400$ 程度に設定してあるので、同程度の解像度のデータをボクセルデータとして保持するのに比べ、そのサイズは数十～数百分の一程度であると言える。

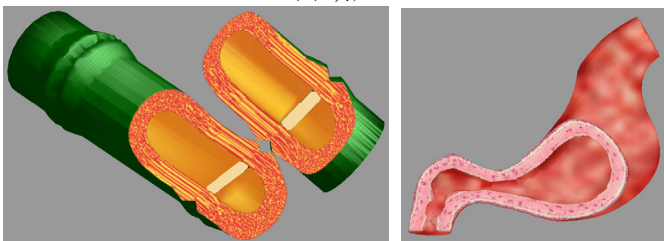
また、本システムでは領域ごとにテキストアノテーションを付けることもできる(図 14)。ユーザーは、どの角度から物体を切断しても、適当な位置に文字情報が表示される。これにより、コンテンツとしての利用価値が高まったと考えられる。

今後の展望

現時点での最も大きな問題点は、断面画像の品質が十分でないことである。とりわけ、Layered テクスチャの品質が悪い。これは、繰り返しリサンプリングが必要となるためだと考えられるので、アルゴリズムを見直し、品質のよい画像が生成されるようにする必要がある。



(a) 胡瓜



(b) 竹

(c) 胃



(d) 歯

図 13 : 本手法で内部構造をモデリングした例

名前	データサイズ	モデリング時間	断面画像生成時間
肉(図 1)	622 kb	90 sec	22 sec
胡瓜	53 kb	40 sec	4 sec
竹	291 kb	30 sec	14 sec
胃	402 kb	15 sec	18 sec
歯	307 kb	120 sec	32 sec

表 1 必要とされるリソース。「モデリング時間」には、最初のメッシュを作成するのに要した時間は含まれていない。また、「断面画像生成時間」は、最も高解像度の画像が完成するまでにかかった時間であり、実際にはプログレッシブ表示を用いているので、ユーザーがこの時間待たねばならないというわけではない。なお、測定に用いたマシンは、PentiumM 1.6GHz メモリ 1GB である。

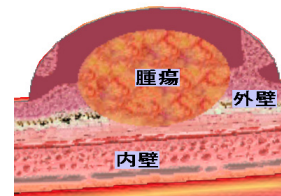


図 14 : テキストアノテーションの例。

References

[Perlin85] Perlin, K. 1985. An Image Synthesizer. proc. Siggraph 1985, 287-296

[Cutler02] Cutler, B., Dorsey, J., McMillan, L., Müller, M. and Jagnow, R. 2002. A Procedural Approach to Authoring Solid Models. proc. Siggraph 2002, 302-311

[Efros99] Efros, A. and Leung, T. 1999. Texture Synthesis by Non-parametric Sampling. proc. ICCV, (2), 1033-8, 1999

[Igarashi99] Igarashi, T., Matsuoka, S., and Tanaka, H. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In Proceedings of Siggraph 1999, 409-416.

[Owada03] Owada, S., Nielsen, F., Nakazawa, K., and Igarashi, T. 2003. A Sketching Interface for Modeling the Internal Structures of 3D Shapes. Lecture Notes in Computer Science (LNCS 2733; Smart Graphics 2003), Springer-Verlag, 49-57

[Turk99] Turk, G., and O'Brien, J. F. 1999. Variational Implicit Surfaces. Technical Report GIT-GVU-99-15, Graphics, Visualization, and Usability Center. Georgia Institute of Technology.

[Turk01] Turk, G. 2001. Texture Synthesis on Surfaces. proc. Siggraph 2001, 347-354.

[Wei00] Wei, L-Y and Levoy, M. 2000. Fast Texture Synthesis using Tree-structured Vector Quantization. proc. Siggraph 2000, 479-488

[Zhang03] Zhang, J., Zhou, K., Velho, L., Guo, B., and Shum, H-Y. 2003. Synthesis of Progressively-Variant Textures on Arbitrary Surfaces. ACM Transactions on Graphics (Proc. Siggraph 2003), 295-302.