社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－CG－131 (7)
2008／7／7

GPUを用いた曲面上へのベクトルデータのレンダリング

中村 徳裕　　爰島 快行　　山内 康晋
（株）東芝 研究開発センター

本稿では，GPU(Graphics Processing Unit)を用いて曲面上にベクトルデータをレンダリングする手法を提案する．本稿で提案するのは2次元で定義されたベクトルデータをガイドとなる曲面上にレンダリングする方法である．提案手法ではベクトルデータをテクスチャ空間に定義し，曲面を描画する際にテクスチャ空間でピクセルを描画するべきかどうかを内外判定により決定する．そのため提案手法では，従来ベクトルデータを曲面上に描画する場合に必要であったベクトルデータの幾何や位相を考慮した三角形分割を必要としない．本稿では幾つかのベクトルデータをレンダリングした場合の結果を示す．

Resolution Independent Rendering of Deformable Vector Objects on Curved Surface Using Graphics Hardware

Norihiro Nakamura　Yoshiyuki Kokojima　Yasunobu Yamauchi
Toshiba Corp. Research & Development Center

In this paper, we suggest a rendering method for deformable vector objects on curved surface using GPU (Graphics Processing Unit). Our method renders 2D vector objects on a deformable curved surface. Our method decides whether each pixel is inside or outside of the vector objects which are defined on texture coordinate space. Thus, there is no need to subdivide vector objects considering its geometrical and the topological condition. Finally, this paper shows some rendering results of some vector object.

1. はじめに

Webで頻繁に目にするFlashコンテンツや出版物に用いられる文字などは，直線と曲線の組み合わせで定義されている．この様な図形の表現形式はベクトルグラフィックスと呼ばれ，解像度に依存せずに一定の表示品質を保てるなどの利点から，近年様々な分野で用いられている．しかしながらベクトルグラフィックスはラスタデータを用いる場合に比べてレンダリング時の処理負荷が高いという問題があった．

そこで，性能向上が著しいGPUを用いてベクトルグラフィックスのレンダリングを高速化する研究が行われている．もっとも単純な方法はCPUでベクトルデータの内部領域を三角形群に分割し，GPUでその三角形群を塗りつぶすことで描画する方法である．この時，ベクトルデータの曲線部分は微細な三角形群で近似される．しかしながらこの方法には以下の2つの問題点があった．

1. 曲線部分を近似する際の分割数が表示解像度に依存する
2. ベクトルデータが変形した場合に三角形群を再生成する必要があるため，動的な変形が行われる場合に実時間で描画するこ

とが難しい

Loop らは曲線部分を三角形群で近似するのではなく，GPU を用いてピクセル単位で描画すべきかを判定することで 1 つめの問題を解決した[1]．この方法は曲線部分を微細な三角形で近似しないため少ない三角形でベクトルデータを描画可能であり，表示解像度に依存せず常に滑らかな曲線を描画可能である．一方，Kokojima らは Loop らの手法を拡張し，ベクトルデータの凹凸や位相構造に影響されず，曲線と直線のつながりが変化しない限り三角形群を再生成する必要がない方法を提案することで 2 つめの問題点を解決した[2]．

ところで，近年の計算機の性能向上に伴い，近年の OS に代表されるような一般ユーザ向けのアプリケーションでも 2 次元的な表現方法だけでなく 3 次元的な表現方法が用いられるようになってきており，今後ベクトルグラフィックスにおいても任意のオブジェクトに合わせて変形させるなどの 3 次元的な表現方法が用いられるようになると考えられる．

しかしながらベクトルデータを曲面などの 3 次元オブジェクト上に描画する場合，従来法ではそのまま描画する事はできず，三角形群を曲面に合わせて変形することが可能な細かさまで分割する必要がある．また，分割後の三角形がある程度規則的でかつ均一に近い大きさになるように分割する必要があるため，実時間で処理することが難しい．さらに，分割によって位相構造が変化しない限り三角形群を再生成する必要がないという Kokojima らの手法の利点が損なわれてしまう．

そこで本稿では，Kokojima らの手法を拡張し，ベクトルデータを曲面上に描画する場合でもその利点を損なうことなく描画可能な手法を提案する．提案手法では従来手法のようにベクトルデータをモデル空間に定義するのではなくテクスチャ空間に定義し，曲面を描画する際にそのベクトルデータを参照してピクセルを塗り分けることで曲面上にベクトルデータをレンダリングする．これにより，ベクトルデータを曲面に変形可能な細かさまで分割することなく曲面上にレンダリングする．

2. 従来法

2.1. 従来法の概要

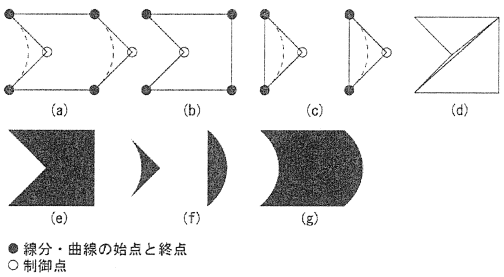

図 1 (a) ベクトルデータ，(b) 多角形セグメント，(c) 曲線セグメント，(d) 直線セグメント，(e) 直線セグメント描画結果，(f) 曲線セグメント描画結果，(g) 描画結果

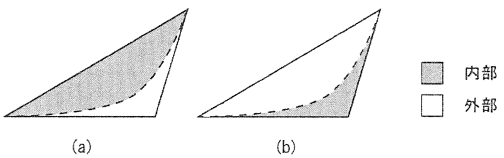

図 2 (a) 凸ベジェ曲線，(b) 凹ベジェ曲線

Loop らはベクトルデータを CPU で図 1(c)(d)に示すような直線セグメントと曲線セグメントに分割し，曲線セグメントを描画する際に各ピクセルが図形の内側に属するかをピクセルシェーダで判定して内部の場合のみ描画することで曲線部分を解像度に非依存で滑らかに描画する手法を提案した．なお，曲線セグメントとはベジェ曲線の始点・制御点・終点で構成される三角形群である．また直線セグメントとは，ベクトルデータに含まれる線分の始点・終点，図 2(b)に示すような凹ベジエ曲線の始点・制御点・終点，および同図(a)に示すよう

な凸ベジエ曲線の始点・終点を結ぶことで得られる多角形セグメントを三角形分割して生成される三角形群である.

Loop らの手法は高品質な曲線を描画することができるが, ベクトルデータの凹凸や曲線セグメントの重複を考慮して三角形分割を行わなければならない. そのため, ベクトルデータを変形して凹凸や曲線セグメントの重複状態が変化した場合には一連の処理を再度やり直す必要がある.

一方, Kokojima らはベクトルデータがリアルタイムに変化する場合や逐次入力される場合など, 前処理でのベクトルデータの分割が困難な場合に効果的な手法を提案している. Loop らの手法と異なる点は, 直線セグメントの生成方法とステンシルバッファを用いて描画する点である. 図 3(b)(c)は同図 (a)のベクトルデータから Kokojima らの方法で直線セグメントと曲線セグメントを生成した結果である. Kokojima らの手法では閉路ごとに任意の点をピボット点とし, そのピボット点と線分で構成される三角形を直線セグメントとしており, ベクトルデータの凹凸を考慮する必要はない. 曲線セグメントの生成方法は基本的には Loop らの方法と同様だが, 曲線セグメントの重複状態は分割に影響しない. これらの手順で生成された直線セグメントと曲線セグメントの凸領域部分をステンシルバッファに描画する. この時のステンシルオペレータをビット反転に設定する事により, 奇数回描画されたピクセルのステンシル値のみが非ゼロとなる. 図 3(e)の網掛け部は直線セグメントの描画によってステンシル値が非ゼロとなり, 曲線セグメントの描画によってゼロに戻る部分である. 同図(f)に示すように, 最終的にはベクトルデータの内部領域を表すピクセルのステンシル値のみが非ゼロとなる. この状態でベクトルデータ全体を覆うポリゴンをフレームバッファに描画する. この時にステンシル値が非ゼロのピクセルにのみ書き込むことで最終的な描画結果が得られる. Kokojima らの手法は曲線セグメントが重複する場合でも分割する必要がないため, Loop らの手法に比べて各セグメントを高速に生成する事ができ, またセグメント数も少なくなる. また, ベクトルデータの凹凸を考慮した三角形分割が不要であるため, ベクトルデータが変形しても各セグメントの頂点の位置座標を変更するだけで対応できる.

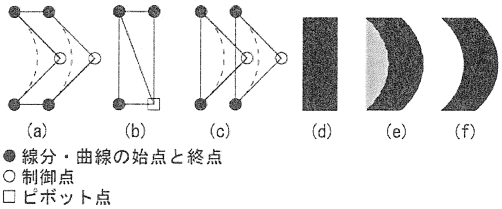

図 3 (a) ベクトルデータ, (b) 直線セグメント, (c) 曲線セグメント, (d) (b)描画後のステンシルバッファ, (e) (c)描画後のステンシルバッファ, (f) 描画結果

2.2. 曲面上に描画する際の問題点

前述した 2 つの従来法は平面上にベクトルデータを描画する場合には有効な手法であるが, 曲面や平面で構成される 3 次元オブジェクトに沿ってベクトルデータ変形させて描画したい場合, 各セグメントをそのまま用いることができない.

どちらのアルゴリズムでも各セグメントを曲面に合わせて変形することが可能な細かさまで分割すれば描画することは可能であるが, その場合分割後の三角形がある程度規則的でかつ均一に近い大きさになるように分割しなければならず, そのような分割を実時間で処理することは容易ではない.

Kokojima らの手法の場合三角形同士が重複しており, その重複回数を調べる事で描画結果を得ている. そのため, 分割前に一部分が重な

っていた三角形は，分割後の三角形においても同様に重なっている必要がある．しかし三角形ごとに独立して分割すると重複している三角形間で同じ位置に分割線が生成されるとは限らないため，視点を変更した場合に正しい描画結果が得られない．Loop らの手法ではその問題は起きないが，曲線セグメントを分割する場合には曲線分割のルールに拘束を受けるため分割後の各セグメントが不規則で不均一な状態になりやすく，曲面や任意のオブジェクトの形に変形させることは困難である．

3. 提案手法

3.1. 提案手法の概要

提案手法の概念を図 4 に示す．提案手法は Kokojima らの手法を曲面上への描画に対応可能なように拡張したものである．提案手法では同図に示すようにテクスチャ空間にベクトルデータを，モデル空間に描画したい曲面データを定義する．この時，ベクトルデータのレンダリング位置はテクスチャパラメータとして曲面データに与えられているものとする．次に曲面モデルを描画する．この時にテクスチャ座標を利用してテクスチャ空間に定義されているベクトルデータを同図の矢印で示すように参照し，最終的に出力するピクセルを塗り分ける事で曲面上にベクトルデータを描画する．提案手法の処理の流れを図 5 に示す．提案手法は同図に示すようなベクトルデータ入力ステップと描画ステップの 2 つに分けられる．まずベクトル入力ステップにおいて直線セグメントと曲線セグメントに変換されたベクトルデータが GPU に入力される．その後，描画ステップにおいてベクトルをレンダリングしたい曲面データとベクトルデータのレンダリング位置が GPU に入力され，各セグメントを参照しながら描画判定を行った結果がフレームバッファに出力される．

従来法と異なる点は二点ある．一点目は従来法がベクトルデータから生成した各セグメントを描画するのに対し，提案手法の各セグメントはテクスチャ空間に定義されることである．テクスチャ空間は GPU に描画するためのデータとして渡されるモデルデータが定義されている空間とは独立した空間であるため，モデルが変形してもそれに応じて変形を行う必要がない．二点目は従来法がステンシルバッファを利用して三角形の重複回数が奇数かどうかを判定するのに対し，提案手法では GPU のピクセルシェーダで幾何的に内外判定を行うことで奇数かどうかを判定する点である．

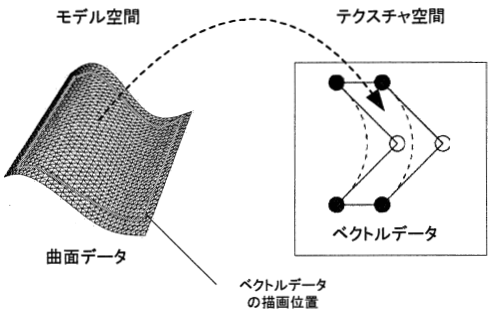

図 4 提案手法のイメージ

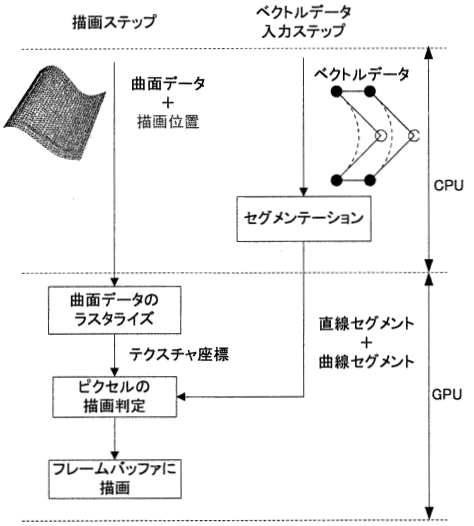

図 5 提案手法の処理の流れ

3.2. ベクトルデータ入力ステップ

まず，図 5 に示すようにベクトルデータから直線セグメントと曲線セグメントを生成する．生成手順は前述した Kokojima らの手法と同様である．このとき，描画したいベクトルデータをテクスチャ空間の座標系に収まるようにしておくことに注意する．例えば，テクスチャ空間の U, V 座標がそれぞれ 0.0~1.0 の値を取る場合，ベクトルデータもその範囲に収まるようにしておく．生成された各セグメントは描画パスを経由せず，直接 GPU に渡される．これにより，各セグメントはテクスチャ空間に定義されたものと考えることができる．なお，各セグメントはテクスチャメモリにラスタイメージの形で定義されているわけではなく幾何データとして定義されており，テクスチャ空間の考え方のみを利用している．そのため，提案手法の描画結果はテクセルの解像度に依存しないことに注意する．また，この時のセグメント数はアルゴリズムの性能に影響するため，Kokojima らのセグメント生成手法を利用することは性能面からも有効である．

3.3. 描画ステップ

3.3.1. CPU による処理

次に，ベクトルデータを変形する際のガイドとなる曲面データを入力する．本稿では曲面を一定のルールで微細な三角形に分割し，曲面に沿って頂点座標を変更したものを曲面データとする．ただし曲面データは必ずしも三角形の集合である必要はなく，同図の「ピクセルの描画判定」処理までにテクスチャ座標が算出できる方法であればよい．また提案手法では曲面データはベクトルデータの形状に依存しないため，最新の GPU で採用されているジオメトリシェーダを用いた三角形分割が可能である．

3.3.2. GPU による処理

GPU では，曲面データがラスタライズされた際に算出されるテクスチャ座標が各セグメントの内部と判定された回数を調べる事で画面に描画するかどうかを決定する．直線セグメントの場合にはテクスチャ座標がそれぞれの三角形の内部に入った回数を，曲線セグメントの場合には定義されている凸領域に入った回数を調べる．この判定はピクセルシェーダで行われ，結果として得られた回数が奇数であった場合のみ，フレームバッファにピクセルを出力する．

3.4. アンチエイリアシング

提案手法ではベクトルデータは GPU でラスタライズされないため，GPU のアンチエイリアシング機能を利用することができない．そこで提案手法では，Kokojima らと同様に alpha-to-coverage [3,4]を用いることでアンチエイリアシング機能を実現する．

提案手法では，描画判定を行う際に同時にベクトルデータまでの最短距離を求め，その距離を基準としてアルファ値を決定する．その後，alpha-to-coverage を利用して描画することでアンチエイリアシングの効果を得る．

4. 実験と考察

提案手法を用いて，正弦関数に沿って変形させた曲面上に図 6 に示すベクトルデータを描画した結果を示す．同図の図形の下の数値は各セグメントの合計であり，括弧内は内訳(直線セグメント数，曲線セグメント数)である．描画には CPU に Intel 社の Core2Quad Q6600 2.4GHz，GPU に NVIDIA 社の GeForce8800GTX を搭載した PC を用いた．描画結果を図 7 に，描画時のフレームレートを図 8 に示す．フレームレートはそれぞれを 100 回描画した平均値から求めており，同図中の AA はアンチエイリアシング処理の有無を示している．同図の数値は各セグメントの生成

と曲面データの生成をそれぞれ初回の 1 度のみ行う処理を含んでおり，曲面データの生成にはジオメトリシェーダを用いている．なお，提案手法では一度各セグメント，および曲面データを生成すればベクトルデータの幾何，および位相構造が変化しない限り再生成する必要がないため，今回それぞれの処理は 1 度のみとした．

84(40,44)

(a)

162(81,84)

(b)

図 6　実験用ベクトルデータ

(a)

(b)

図 7　描画結果

		200x150 pixel	400x300 pixel	800x600 pixel
(a)	AA なし	252.2	160.3	70.9
	AA あり	210.9	125.7	55.5
(b)	AA なし	164.9	93.0	39.0
	AA あり	129.3	67.1	29.0

単位：fps

図 8　フレームレートの比較

図 7 より，それぞれのベクトルデータが曲面上に描画できていることがわかる．また図 8 より，提案手法の性能は画面解像度とセグメント数に依存すること，アンチエイリアシングにより性能が約 16%～28%低下することがわかる．

5.　おわりに

本稿では GPU を用いて曲面上にベクトルデータをレンダリングする手法を提案した．提案手法ではベクトルデータをテクスチャ空間に定義してテクスチャ座標で描画の可否を判定することで，曲面の複雑さに合わせてベクトルデータを微細な三角形群に再分割することなく描画を可能とした．また曲面上に描画した場合でも，ベクトルデータの入力から表示までのレスポンスが高速である，ベクトルデータを動的に変形しても CPU の処理負荷が低いなどの Kokojima らの手法の持つ利点を維持している．加えて曲面データを生成する際にベクトルデータの幾何や位相に拘束されないため，GPU の新機能であるジオメトリシェーダと組み合わせて利用しやすい．

提案手法ではピクセルシェーダで幾何演算を行っており，計算精度の影響で描画結果に 1～2 ピクセル程度のノイズが発生する場合があるため，この問題解決が今後の課題としてあげられる．また，一般的な CG 処理との組み合わせの整合性の検討に取り組んでいきたい．

参考文献

[1] C. Loop and J. Blinn. Resolution Independent Curve Rendering using Programmable Graphics Hardware. In Proceedings of ACM SIGGRAPH 2005, pages 1000-1010.

[2] Y. Kokojima, K. Sugita, T. Saito and T. Takemoto. Resolution Independent Rendering of Deformable Vector Objects using Graphics Hardware. In ACM SIGGRAPH 2006 Sketches.

[3] Antialiasing with Transparency. nVIDIA technical report.

[4] Alpha to coverage. ATI white paper, Radeon SDK Oct. 2005.