芸術科学会論文誌 Vol. 15, No. 2, pp. 75 – 85

3DCG における画像処理と形状処理の併用によるリアルタイム輪郭線描画

奥屋武志 [1)](正会員)　田中克明 [2)]　坂井滋和 [2)]

1) 早稲田大学大学院基幹理工学研究科表現工学専攻　2) 早稲田大学基幹理工学部表現工学科

Real-Time Line Drawing Using Image Processing and Deforming Process Together in 3DCG

Takeshi Okuya[1)]　**Katsuaki Tanaka**[2)]　**Shigekazu Sakai**[2)]

1) Department of Intermedia Art and Science, Graduate School of Fundamental Science and Engineering , Waseda University

2) Department of Intermedia Art and Science, Fundamental Science and Engineering , Waseda University

okuya.waseda @ gmail.com

概要

3DCG のトゥーンレンダリング等において輪郭線を生成する手法は画像処理を用いる手法とモデルの 3 次元形状を用いる手法に分類される．画像処理を用いる手法では輪郭線の検出処理に用いる G-Buffer を選択することにより制作者の意図を反映した高精度な検出が可能であるが，線の太さは均一であり手描きではない機械的な印象となる．モデルの 3 次元形状を用いた手法では線の太さは調整可能であるものの，線の検出は各描画アルゴリズムに依存し，制作者の意図した線を検出できない．本論文では輪郭線の検出と描画の工程を分離し，検出には画像処理の手法を描画にはモデルの 3 次元形状の手法を用いることによって高精度で輪郭線を検出しつつ線の太さの変化による誇張表現を行った．さらに，提案法を既存のグラフィックス API 上でリアルタイムに実行するためのパイプラインを構築・実装し，リアルタイムに実行できることを確認し，より高速化する手法についても検討を行った．

Abstract

In cartoon rendering of 3DCG, methods for generating a contour lines are classified into the method using the image processing or the method using the three-dimensional shape of the models. By selecting G-Buffers, the method using the image processing can accurately detect intended lines of users, but the lines are mechanical impression by the uniform thickness. The methods using the three-dimensional shape of the models can adjust thickness of the lines, but detection of the lines is dependent on the rendering algorithm and it can't detect the intended lines of users. In this paper, we separated contour lines process into detection and rendering. Our approach uses image processing for detection and model shapes for rendering. It can detecting intended line of users and draw exaggerated lines by changing the thickness of lines. In addition, we implemented a graphics API pipeline to run our method in real-time, and also examined a technique to accelerate processing speed.

この論文は「NICOGRAPH 2015」に投稿した論文 [1] を芸術科学会論文誌に投稿するものである．

1 はじめに

人間が手描きしたような画像を 3DCG で生成するトゥーンレンダリングは輪郭線を描画するインキングとモデル内部の色を塗るペインティングという二つの工程から構成される [2]．この手法は鉛筆による線画のセルへの転写と絵の具による塗りによって制作されていた手描きアニメーションの映像を 3DCG で再現できることが期待され，特に日本国内で制作が盛んな手描きアニメーション（以下『手描きアニメ』）制作の代替手段として導入が進んでいる．近年では手描きアニメ調の CG 制作手法が実用的な段階に達したことから，テレビアニメ，長編映画，ゲームなど様々な分野で全てが 3DCG で作られた作品が制作されるようになってきた [3]．

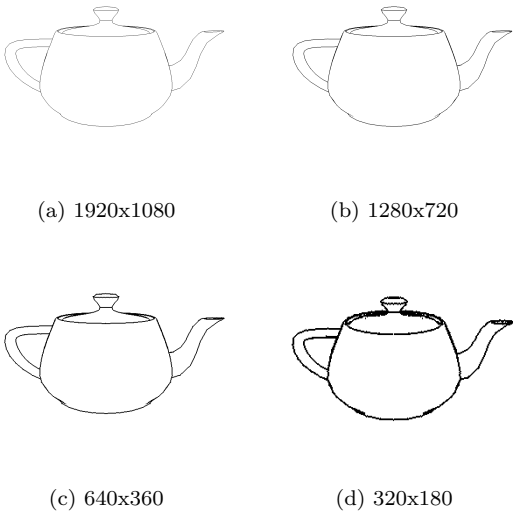
従来の手描きアニメではインキングに相当する工程は全て鉛筆で描かれていたため，線の太さは変化し，その強弱によって誇張表現もなされていた．3DCG における輪郭線の描画は画像処理を用いた手法とモデルの 3 次元形状を用いた手法にわけられるが，前者は線の検出精度が高いが太さは均一で調整できず，後者は太さの調整は可能だが検出される線が描画アルゴリズムに依存する．本研究では，高い検出精度のまま線の太さの強弱による誇張表現を実現するため，輪郭線を検出する工程と描画する工程を分離し，前者には画像処理的手法を後者にはモデルの 3 次元形状による手法を用いた．従来法との比較から提案法では検出精度と誇張表現の両立が成されていることが確認された．また，提案法をゲームなどのリアルタイムコンテンツで用いるため既存のグラフィックス API におけるグラフィックパイプライン上での実装を行い，リアルタイムレンダリングが可能であることを確認した．また，今後生成画像が高解像度化した場合にもリアルタイム性が維持できるように高速化に対する検討もあわせて行った。

2 関連研究

3DCG における輪郭線の検出と描画の手法はアプローチの違いから画像処理を用いた手法とモデルの 3 次元形状を用いた手法に分類される．

(1) 画像処理を用いた手法

Saito ら [4] は先にモデルをレンダリングし，ピクセル毎に 3 次元座標や法線を G-Buffer と呼ばれる画像に出力した後に，これに画像処理を行うことで輪郭線を生成した．Decaudin[5] はこれをトゥーンレンダリングに応用している．この G-Buffer は輪郭線描画のための利用だけに留まらず，後に Deering らのシェーディング手法 [6] と組み合わせてシェーディングにも用いることが提案され，現在ではグラフィックス API の Multiple Render Targets 機能を用いてリアルタイムにレンダリングする手法が確立されている [7]．手描きアニメ調 3DCG の制作用に輪郭線生成のプラグインも製品化されており [8]，輪郭線を検出する基準をオブジェクト境界や法線角度などから選択することで，制作者の意図を反映して必要な輪郭線のみを検出・描画することが可能である．このように，画像処理を用いた手法では制作者の意図を反映して高精度に輪郭線を検出できるものの，一方で線の太さは均一であり，手描きに似せた誇張表現は行われていない．また，1 パスで輪郭線検出フィルタをかけただけの場合，図 1 に示すように線の太さは解像度に依存してしまう。



(a) 1920x1080　(b) 1280x720

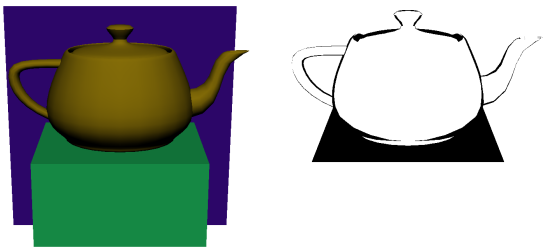
(c) 640x360　(d) 320x180

図 1　画像処理を用いた手法による解像度毎の線の太さの違い

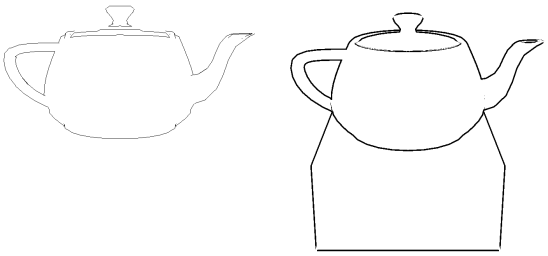
(2) モデルの 3 次元形状を用いた手法

モデルの 3 次元形状を用いた手法はさらに複数のアプローチに分類される [9]．サーフェイス角度による輪郭処理 [10] では視線ベクトルと法線の内積から閾値処理で輪郭線となるか判定し，マテリアル毎に閾値を調整することで線の太さは変更される．輪郭エッジ検出を用いた

手法 [2] では二つのポリゴンが共有するエッジに対して双方のポリゴンが視点から見て表裏となる場合に輪郭線として描画する．手続き型ジオメトリ輪郭処理（裏ポリゴン法）[9] では視点から見て裏面となるポリゴンの頂点を移動して拡大し，輪郭線として描画している．このとき頂点の移動量を調整することで太さの変化が可能となる．松尾ら [11] はモデル周辺に複数の制御点を配置して頂点の移動量を変化させ，裏ポリゴン法による線の太さを変化させた．さらに，モデルの曲率を移動量に反映させることで，変形するモデルに対して動的にリアルタイムな輪郭線の誇張表現を可能とした [12]．筆者らはこの曲率をより高速に計算する手法を開発している [13]．このように，モデルの 3 次元形状を用いた手法では線の太さの変化による誇張表現は確立しているものの，一方で線の検出が各描画アルゴリズムに依存しており，制作者の意図通りに輪郭線を検出できないという問題がある（図 2）．



(a) 輪郭線を描画するモデル　(b) サーフェイス角度



(c) 輪郭エッジ検出　(d) 裏ポリゴン法

図 2　モデルの 3 次元形状を用いた手法による輪郭線

3　輪郭線の検出と描画

3.1　画像処理を用いた手法とモデルの 3 次元形状を用いた手法の併用

前章より画像処理を用いた手法ではレンダリングする G-Buffer を選択することによって制作者の意図した通りに高い精度で輪郭線を検出できるが，線の太さは均一になり頂点に付加された情報を用いて線に変化を与えることは困難であることがわかった．一方，モデルの 3 次元形状を用いて輪郭線を描画する手法では頂点に付加された情報を用いて線に変化を与えることができるが，輪郭線の検出が描画アルゴリズムに依存してしまうため，制作者の意図通りの輪郭線を検出することが困難であることがわかった．これらの違いをまとめると表 1 のようになる．

表 1　手法による検出精度と線の太さの違い

	検出精度	線の太さ
画像処理を用いた手法	高い（G-Buffer の選択により制作者の意図を反映できる）	均一
モデルの 3 次元形状を用いた手法	低い（手法に依存し，制作者の意図は反映されない）	調整可能

輪郭線の検出では画像処理を用いた手法が優れ，太さの変化による輪郭線の表現ではモデルの 3 次元形状を用いた手法が優れており，それぞれ長所が異なる．したがって，本研究では輪郭線を検出する工程と描画する工程を分割して検出には画像処理的手法，描画には 3 次元形状を用いた手法を利用し，複数パスでそれぞれのアルゴリズムを用いて検出と描画を行うことによって，双方の手法の長所を活かして高い検出精度と変化のある線の描画の両立を行う．次節以降でそれぞれのアルゴリズムの詳細を示す．

3.2　輪郭線の検出

輪郭線の検出には画像処理を用いた手法を利用する．まず、Saito らの手法と同様にモデルを一度レンダリングし、輪郭線検出に必要な各種情報をピクセル毎に G-Buffer へ出力する．G-Buffer に出力する情報は制作者

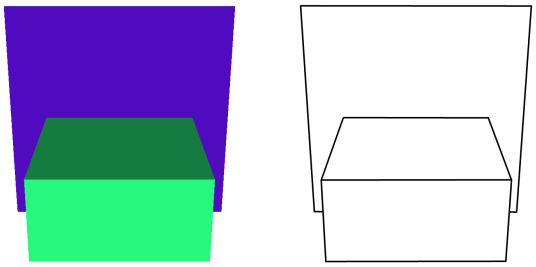
が描画したい輪郭線の内容に応じて適宜選択する．出力された G-Buffer それぞれに対して輪郭線検出の画像処理を行って各ピクセルが輪郭線となるか否か判定する．各 G-Buffer の結果を合成することにより，輪郭線が検出された画像が完成する．この輪郭線画像は線の太さが均一であり頂点の持つ輪郭線を変化させる情報は反映されていないが，制作者の意図通りに高い精度で輪郭線が検出されている．

輪郭線を検出する例として図 3 のようなモデルの輪郭線の検出を行う．このモデルでは手前に立方体があり，その奥に平面が存在する．図 3(a) は輪郭線を検出するモデルをシェーディングした画像であり，図 3(b) はこの例において図上で検出したい輪郭線である．この輪郭線には境界エッジと折り目エッジが含まれている．これらのエッジを検出するため G-Buffer には Z 座標と法線マップを出力する．それぞれの G-Buffer から検出したエッジを図 4 に示す．図 4(a) は Z 座標の G-Buffer であり，図 4(b) はそこから検出した輪郭線である．これを図 3(b) と比較すると境界エッジは全て検出されているが，立方体の手前側にある折り目エッジが検出されていない．図 4(c) は法線マップの G-Buffer であり，図 4(d) はそこから検出した輪郭線である．Z 座標から検出されたエッジと比較すると，立方体の手前側にある折り目エッジは検出されているが，立方体と平面の境界となっているピクセルでは境界エッジが検出されていない．Z 座標と法線マップそれぞれの検出結果を合成した結果を図 5 に示す．この合成結果で検出された輪郭線は図 3(b) で示した検出したい輪郭線と一致している．

3.3 輪郭線の描画

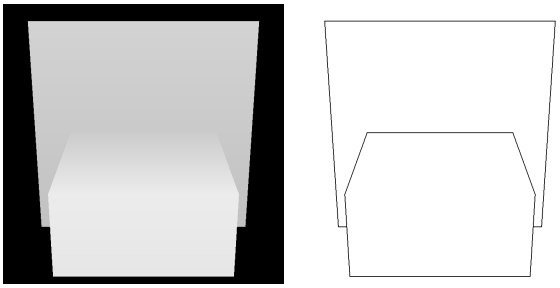
前節で作成した輪郭線の検出画像と各頂点に付加された情報から輪郭線の描画を行う．輪郭線の描画は以下の手順で実行される．

1. 各ポリゴンに対して視点から見たときに裏表どちらになるか判定し，裏面を除外
2. 残ったポリゴンの頂点を各エッジ毎に投影面上でエッジ方向と垂直になる方向に移動
3. 移動した頂点で四角形ポリゴンを生成
4. ポリゴンの描画においてピクセル毎に元のエッジの座標を用いて前節の輪郭線画像を参照し，輪郭線が検出されている場合のみ描画

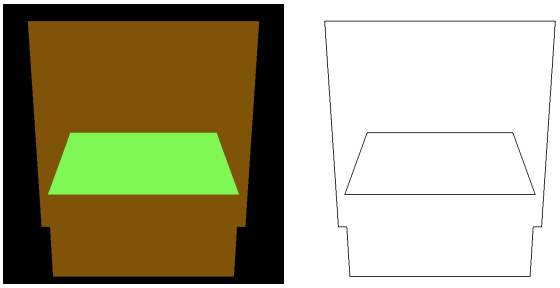


(a) シェーディング　(b) 検出したい輪郭線

図 3　輪郭線を検出するモデルと検出したい輪郭線



(a) G-Buffer（Z 座標）　(b) 検出された輪郭線（Z 座標）



(c) G-Buffer（法線マップ）　(d) 検出された輪郭線（法線マップ）

図 4　G-Buffer から検出された輪郭線

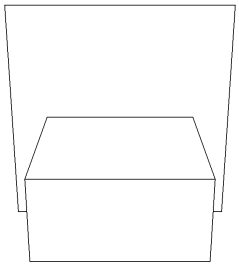
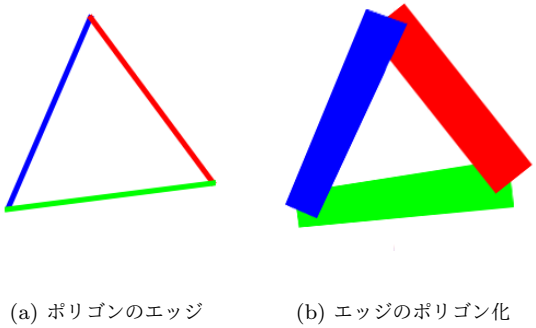
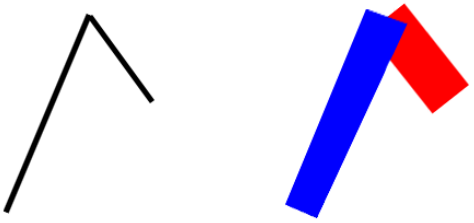


図 5　合成した検出結果



(a) ポリゴンのエッジ　(b) エッジのポリゴン化



(c) 検出された輪郭線　(d) 描画された輪郭線ポリゴン

図 6　エッジのポリゴン化と描画

手順 1 において、ポリゴンの裏表の判定はそのポリゴンを構成する頂点が投影面上で時計回りか反時計回りのどちらで結ばれている場合に表となるか予めモデリング時に決定しておき，それを基準に判定する．ここで裏面となったポリゴンは以後の工程には用いられず除外され，エッジは描画されない．したがって，表面の見えるポリゴンだけが残る．

手順 2 から 4 までを図 6 に示す．図 6(a) は 1 つの三角形ポリゴンを構成するエッジを表す．これらのエッジは手順 2,3 でポリゴン化され図 6(b) の様になる．本来のポリゴン色は輪郭線として設定された色であるが，図上では簡略化のため 3 色に塗り分けている．手順 4 ではポリゴンの描画時に元のエッジの座標（参照座標）を用いて，前節で作成した輪郭線検出画像である図 6(c) のピクセルが参照され，輪郭線が検出された参照座標を持つ部位のみ描画される．この描画されたポリゴンが輪郭線となる．

手順 2,3 において一本のエッジからポリゴンを生成する手法の詳細を図 7 に示す．エッジを構成する頂点 V_0, V_1 が投影面上の座標 P_0, P_1 に配置されている．P_0 から P_1 に向かうベクトル $\overrightarrow{V_0V_1}$ に垂直な単位ベクトル $\boldsymbol{u}$ を求める．V_0, V_1 を $\boldsymbol{u}$ の正負の方向にそれぞれ t_0, t_1 だけ移動した頂点 V_{00}, V_{01}, V_{10}, V_{11} を生成する．このときの移動量 t_0, t_1 は，予め V_0, V_1 に付加されている輪郭線の太さを決める値から求める．輪郭線の太さは V_0, V_1 でそれぞれ $2t_0$, $2t_1$ となり，頂点間では線形に変化する．新たに生成される頂点は自身の位置だけでなく，元のエッジの位置を記録する参照座標を持つ．この参照座標が手順 4 で輪郭線検出画像の参照に用いられる．これらの新たに生成された頂点によって作られる四角形ポリゴン $V_{00}V_{10}V_{11}V_{01}$ が手順 4 で輪郭線として描画される．ポリゴンからのピクセル描画時には参照座標はピクセル毎にそれぞれの頂点から線形補間される．

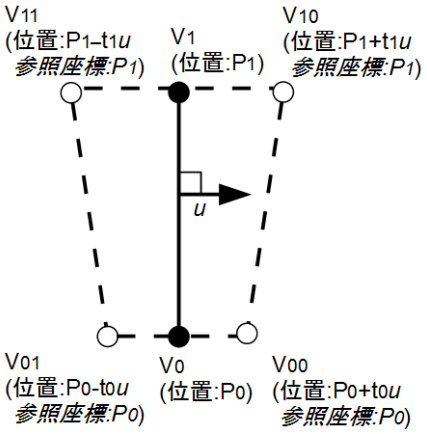


図 7　エッジからのポリゴン生成

これの手順により，前節で検出した輪郭線に対応するエッジのみ，頂点に付加された情報を用いて太さに変化を与えられた線が描画される．

4　グラフィックスパイプライン上での実装

前章のアルゴリズムによる輪郭線の検出と描画をリアルタイムに行うため，リアルタイムレンダリング用のグラフィックス API におけるグラフィックスパイプラインに適した形で前章のアルゴリズムの実装を行った．本論文では API に Direct3D 11 を利用し，シェーダーモデル 5.0 での実装を行った．実装したパイプラインの概要を図 8 に示す．図中の実線矢印はパイプラインの流れを表し，破線はデータの入出力を表す．

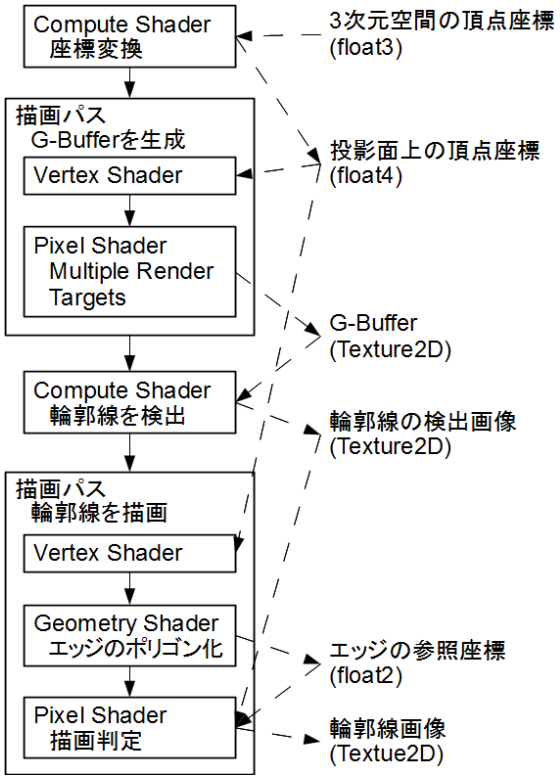


図 8　グラフィックスパイプライン

4.1　Compute Shader での座標変換

まず，Compute Shader を用いて座標変換を行い，各頂点の投影面上の座標を求める．一般的に座標変換は Vertex Shader で行われるが，本研究の手法では二度の描画パス両方で座標変換後の頂点座標を用いるため，Vertex Shader で座標変換を行うと計算内容が重複し，無駄な処理となる．そのため，事前に Compute Shader で変換後の座標を求めておくことにより，処理の効率化を行った．Pixel Shader が実行される前にラスタライズと Depth Buffer への書き込みが行われることも考慮して，投影面上の頂点座標は float4 型で出力する．ここで実行されるスレッド数は頂点数と等しい．View 座標の Z 値を G-Buffer として用いる場合には，View 座標への変換もここで行う．また，ボーン付きのモデル等でシェーダ上で変形を行う場合にはこの段階で同じく処理を行う．

4.2　G-Buffer の生成

前工程で座標変換は完了しているため，Vertex Shade では次のシェーダにデータを渡す処理のみを行う．ここでの Geometry Shader は不要であり，次に Pixel Shader が実行される．Pixel Shader では輪郭線検出に必要な情報が G-Buffer となるテクスチャ画像に出力される．G-Buffer に出力される情報が 4 チャンネル以上となる場合には Multiple Render Targets を利用して複数のテクスチャ画像に対して出力を行う．

4.3　輪郭線の検出

Compute Shader を用いて G-Buffer に対して近傍画素の差分による閾値処理を行い，各ピクセルが輪郭線となるか判定し，その結果を出力用のテクスチャに記録する．ここで実行されるスレッド数はテクスチャのピクセル数と等しい．

4.4　輪郭線の描画

Vertex Shader では前の描画パスと同様に次のシェーダにデータを渡す処理のみを行う．各頂点は Geometry Shader へ三角形ポリゴン毎に入力され，Geometry Shader からは TriangleStream へ新たな頂点が出力される．まず，ポリゴンの裏表について判定を行い，裏の場合はこのポリゴンの Geometry Shader は終了する．表の場合は三角形ポリゴンの各エッジに対してそれぞれ図 6 のポリゴン化する処理を行う．ここで生成される四角形ポリゴンは二つの三角形ポリゴンとして TriangleStream に出力される．Pixel Shader には元のエッジの座標が参照座標として入力される．この参照座標はラスタライザによって自動的に頂点間で線形補間されている．参照座標を用いて輪郭線の検出画像の輝度値を取得し，輪郭線となっている場合のみ輪郭線の描画色を return して輪郭線画像のピクセルへ描画を行い，輪郭線でない場合は処理を破棄する．

4.5　高速化

前節までの処理をより高速化する手法として，検出工程を低解像度で行い，描画工程でのレンダリングのみを高解像度で行う．検出工程の低解像度化により，G-Buffer の生成と輪郭線検出フィルタによる検出・結果の合成に要する処理時間が短縮される．この手法では G-Buffer に記録される形状が同一解像度のときと一致することから閾値の調整で同一像度の場合と同様の輪郭線を検出することができ，UV 座標上での輪郭線の位置は変化していない．描画工程において描画する画像と輪郭線検出画像の解像度が異なるが，UV 座標上の検出輪郭線は解像度が一致するときと等しいため，参照座標を UV 座標に変換することにより，対応する輪郭線を参照

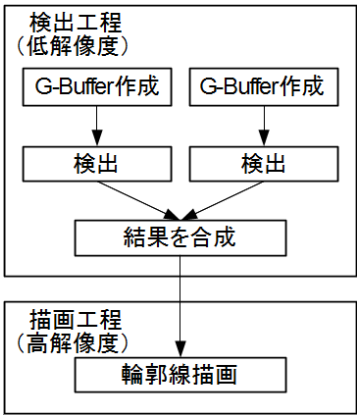


図 9　高速化処理の流れ

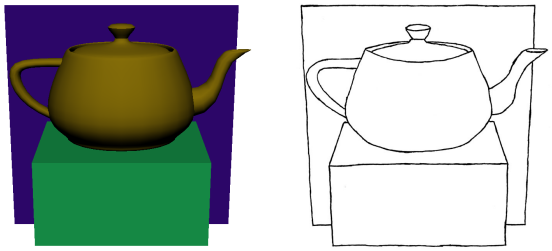
して描画判定を行うことが可能である．したがって，最終的なレンダリングでの画質は維持されつつ，検出工程での処理が高速化される．また，参照座標に UV 座標を用いることにより，描画工程のシェーダは検出画像の解像度に関係なく同一のコードで実装可能である．この処理の流れを図 9 に示す．

5　評価

5.1　輪郭線の検出と描画

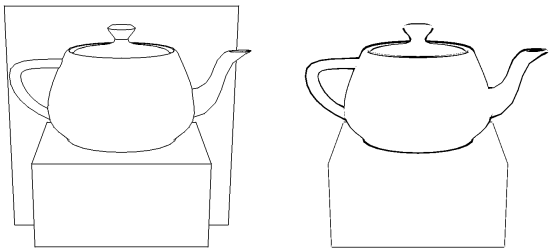
本手法による輪郭線の検出精度と太さの変化について従来法との比較から定性的な評価を行う．従来法として，画像処理を用いた手法には Saito らの手法 [4] を用い，3 次元形状を用いた手法には松尾らの曲率に応じて輪郭線の太さが変化する手法 [12] を用いる．

図 10(a) は輪郭線を描画するモデルに対してシェーディングを行った画像である．このモデルでは直方体の上にティーポットが乗っており，それらの奥に平面が存在する．これをトレースし、検出したい輪郭線を示した画像が図 10(b) である。各手法によって描画された輪郭線を図 11 に示す．図 11(a) は画像処理を用いた手法による輪郭線である．G-Buffer には法線マップと View 座標上の Z 座標を用いた．二つの G-Buffer の組み合わせにより，境界エッジと折り目エッジが検出されるため，直方体の面と面の交線やティーポットの本体と注ぎ口の境界も輪郭線として描画されている．線の太さは均一である．図 11(b) は 3 次元形状を用いた松尾らの手法による輪郭線である．この手法では裏ポリゴン法によって輪郭線を描画している．線の太さは曲率によって変化し，曲率の大きな箇所でより太い線となる．画像から太さは

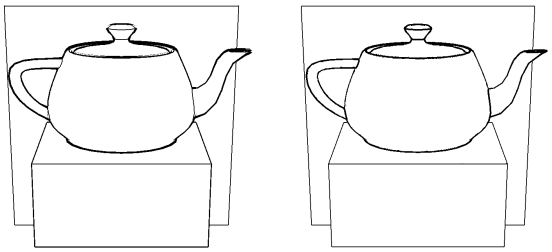


(a) シェーディング　　(b) 検出したい輪郭線

図 10　輪郭線を描画するモデル



(a) 画像処理を用いた手法　　(b) 3 次元形状を用いた手法



(c) (a),(b) を合成した輪郭線　　(d) 提案法

(e) 合成 (c) の拡大図　　(f) 提案法 (d) の拡大図

図 11　各手法の比較

変化していることがわかるが，画像処理を用いた手法と比較すると，ティーポットの本体と注ぎ口や取っ手との境界など輪郭線として検出されていない箇所がある．図 11(c) は図 11(a) と図 11(b) の輪郭線を合成した画像である．図 11(a) と同じく全ての輪郭線が検出され，大部分の輪郭線は図 11(b) と同様に太さが変化しているが，拡大図である図 11(e) を参照すると図 11(b) では検出されなかったティーポット本体と注ぎ口の境界の線などでは太さが均一で曲率が反映されておらず，細い線となっている．図 11(d) は提案法による輪郭線である．太さの変化には松尾らの手法との比較のため同じく曲率を用いた．図 11(a) と同じく全ての輪郭線が検出され，かつ図 11(b) と同様に線の太さも変化している．図 11(e) では細く均一な線となっていた箇所も提案法の拡大図である図 11(f) では曲率に応じて太さが変化している．したがって，提案法は画像処理を用いた手法の検出精度と 3 次元形状を用いた手法による線の太さ変化の表現という双方の長所を両立していることがわかった．

5.2 リアルタイムレンダリング

描画速度を計測することにより提案法を用いてリアルタイムレンダリングできることを評価する．評価に用いた環境を表 2 に示す．各ポリゴン数と解像度で輪郭線画像を描画し，フレームレートをそれぞれ計測した．この結果と逆数から描画時間を求めたグラフをそれぞれ表 3，図 12・13 に示す．

表 2　評価に用いた環境

OS	Windows 10 Pro
CPU	Intel Core i7 3370K(3.5GHz)
GPU	NVIDIA GeForce GTX 580(772MHz)
開発環境	Visual Studio 2015
API	Direct3D 11.0(Visual Studio 2015)

表 3　各ポリゴン数 (三角形) と解像度における描画速度 [fps]

		解像度 (横 x 縦)		
		640x360	1280x720	1920x1080
三角形	960	1699.3	649.7	331.7
	9566	1394.6	610.4	317.0
	66400	445.1	318.4	209.8

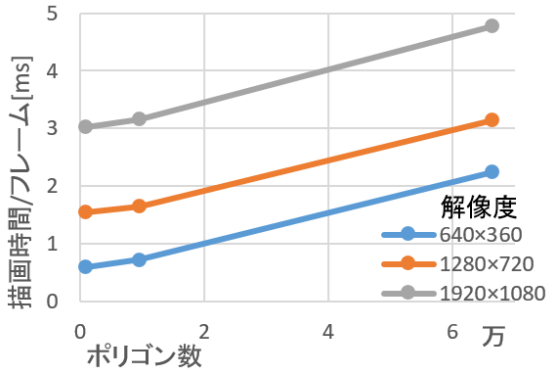


図 12　ポリゴン数と描画時間の関係

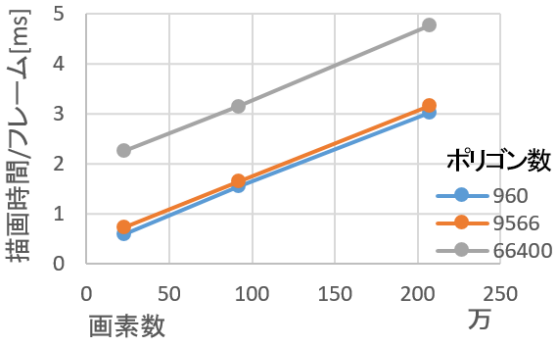


図 13　画素数と描画時間の関係

いずれの結果も 60fps を超えていることから，輪郭線の検出と描画のみではリアルタイム処理が可能である．最も処理に時間のかかるポリゴン数 66400 個のモデルについて fps の逆数から輪郭線の処理に必要な時間を求めると [1280x720] で 3.1ms，[1980x1080] で 4.7ms となった．これは 60fps, 30fps をそれぞれ満たすときに必要な 1 フレームあたりの処理時間である 16ms, 33ms の 2 割前後であり充分小さい．したがって，本手法の処理時間はレンダリングに必要な処理時間全体の僅かな割合を占めるのみであり，シェーディングやペインティングに必要な時間が十分確保されていることから，本手法はビデオゲーム等のリアルタイムコンテンツにおいても利用することが可能である．また，グラフからポリゴン数と画素数の増加に伴ってそれぞれ処理時間は増加するが，図 12 においてポリゴン数増加に伴う描画時間の増加量は解像度によって変わらず，図 13 において画素数の増加に伴う描画時間の増加量はポリゴン数によって変わらない．よって、画素数とポリゴン数の増加に伴う描画時間の増加量はそれぞれ独立していることがわかった．

5.3 高速化

4.5 節で述べた高速化を行った結果について，検出の解像度毎の描画速度と逆数から求めた描画時間を表 4・図 14 に，検出結果と最終的な輪郭線画像を図 15 に示す。なお，HLSL の Pixel Shader 上で UV 座標と Sample メソッドを用いて検出画像の輝度値を取得すると，検出画像の生成時と僅かに異なる値が返される現象が発生したため，UV 座標から検出画像のピクセル座標へ変換し，Load メソッドで取得している．

表 4　検出解像度による描画速度の違い（ポリゴン数 66440）

検出解像度	レンダリング解像度	速度 [fps]
1920x1080	1920x1080	209.8
1280x720		282.4
640x360		346.0
320x180		377.6

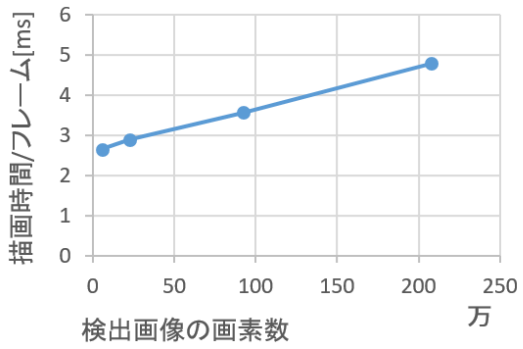
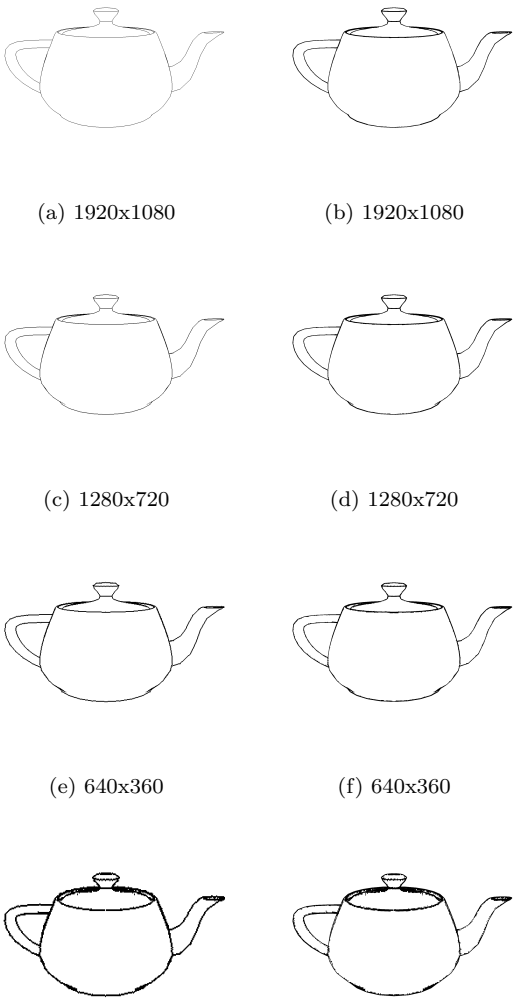


図 14　検出画素数と描画時間の関係



(a) 1920x1080　(b) 1920x1080

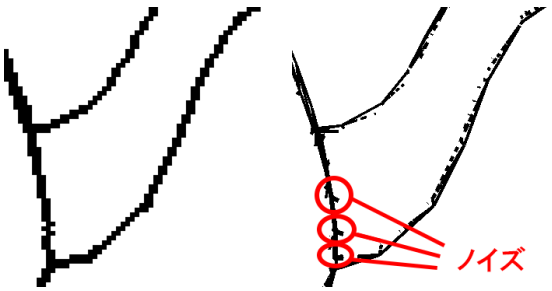
(c) 1280x720　(d) 1280x720

(e) 640x360　(f) 640x360

(g) 320x180　(h) 320x180

図 15　検出解像度毎の検出結果（左）とレンダリング結果 (右)[レンダリング解像度 1920x1080]

これらの結果から、検出解像度を低くすることにより、レンダリング結果の見た目をほぼ保ったまま高速化することが可能であることがわかった。検出解像度を下げるほど高速化の効果は大きい．ただし、[320x180] の画像を拡大した図 16 では輪郭線描画にノイズが現れている。検出結果にはノイズが含まれていないことから、描画工程で初めてノイズが発生していることがわかる。両者の比較から，ノイズの原因はレンダリング時に本来は輪郭線でない部位でも参照した検出画像の座標では輪郭線となっていることであると推定される。したがって、検出解像度を下げる場合には、検出結果の輪郭線の太さが最終的な描画の輪郭線の太さ以下であることが条件となる。



(a) 検出結果　(b) レンダリング結果

図 16　検出解像度 320x180 の拡大図

6 まとめ

本研究ではトゥーンレンダリングにおける輪郭線の検出と描画について画像処理を用いた手法とモデルの 3 次元形状に基づく手法の組み合わせにより，制作者の意図した通りに高い精度で輪郭線を描画しつつ，形状に応じて輪郭線を変化させることが可能となった．

本研究の成果は以下の通りである．

(1) 輪郭線の検出精度と誇張表現の両立

従来法では輪郭線の検出と描画が同時に行われていたため検出精度と誇張表現は各手法毎に一長一短であったのに対し，本研究では輪郭線を検出する工程と描画する工程を分割して，輪郭線の検出に画像処理を用いた手法を用い，線の誇張表現では 3 次元形状に基づく手法を用いることによって検出精度と誇張表現の両立を実現した．従来法と本研究の提案法を比較検証した結果から，提案法では画像処理を用いた手法と同等の輪郭線の検出精度を維持したまま 3 次元形状を用いた手法と同じく線の太さの変化による誇張表現が可能となることがわかった．検出する輪郭線は G-Buffer の種類を選択することにより制作者側で調整できる．従って，ノンフォトリアリスティックな CG 制作において提案法を用いることで，より手描きらしく制作者の意図が反映された画像を制作することが期待される．

(2) グラフィックスパイプライン上での実装とリアルタイムな描画

本研究では提案法のアルゴリズムを既存のグラフィックス API 上で実装することにより，グラフィックスパイプラインに適合した形でリアルタイムなレンダリングが可能となることを示した．一般的な描画パスの次に Compute Shader による画像処理を行って再度描画パスを実行することにより，画像処理とポリゴンによる輪郭線の描画の双方を行っている．検出工程のみ低解像度で処理することにより、レンダリング画質を維持したまま高速化ができることを確認した。グラフィックスパイプライン上での実装が可能であり評価によってリアルタイムな描画と高速化が可能であることを確認したことにより，本研究で開発した手法はノンフォトリアリスティックなゲームなどのリアルタイムコンテンツに取り入れることが可能である．また，既存の 3DCG 制作ソフトのプレビュー機能に組み込めば，手描きアニメ調の 3DCG 映像制作において輪郭線の状態を常に確認しながらモデリングやショット作業を行うことが可能となり，制作の効率化と品質の向上が期待できる．

今後の課題としては以下の 2 点が挙げられる．

(1) より手描きらしい線の誇張表現のためのパラメータの検討

本研究で描画される輪郭線の太さは制作者側で頂点の移動量 t を求めるパラメータと計算手法を設定することにより自由に設定可能であり，特定の誇張表現アルゴリズムに依存しない．本研究の検証では従来法との比較のため太さの変化に頂点の 3 次元での曲率を用いた．しかし，既往文献でも指摘されているように同一のモデルでも 3 次元形状の曲率と投影面上での曲率は異なる [12]．したがって，移動量 t について，より手描きらしい線の誇張表現を実現するためのパラメータと計算手法を検討してゆくことが必要である．

(2)GPU を用いた効率的かつ高品質な直線の描画手法の開発

本研究ではポリゴンのエッジから新たな四角形ポリゴンを生成し，これを輪郭線として描画している．しかし，直線同士の連結については特に考慮していないため，太さによっては線が曲がる部分において凹みが目立ち，線のぎざぎざになって品質が低下する可能性が考えられる．したがって，太さに変化のある線を凹みが発生することなく描画するアルゴリズムを開発する必要がある．開発するアルゴリズムはリアルタイムレンダリングに用いられるため GPU 上で実装可能であることが望ましい．

参考文献

[1] 奥屋武志，田中 克明，坂井 滋和: “3DCG における画像処理と形状処理の併用によるリアルタイム輪郭線描画”，NICOGRAPH 2015 論文集, 2015

[2] Carl S, Marshall: “トゥーン レンダリング：リアルタイム輪郭エッジ検出とレンダリング”,『Game Programming Gems 2』, pp436-443，ボーンデジタル，ISBN: 978-4939007330, 2002

[3] ワークスコーポレーション書籍編集部:『アニメ CG の現場 2015 —CGWORLD 特別編集版—』，ワーク

スコーポレーション，ISBN: 978-4862671745, 2014

[4] Takafumi Saito and Tokiichiro Takahashi:“Comprehensible Rendering of 3-D Shapes”, Computer Graphics(SIGGRAPH ’90 Proceedings), pp197-206, 1990

[5] Philippe Decaudin: “Cartoon-Looking Rendering of 3D-Scenes”, Research Report INRIA 2919, 1996

[6] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, Neil Hunt: “The triangle processor and normal vector shader: a VLSI system for high performance graphics”, ACM SIGGRAPH Computer Graphics Volume 22 Issue 4, pp 21-30, 1988

[7] Shawn Hargreaves, Mark Harris:“Deferred Shading” NVIDIA, http://http.download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_Deferred_Shading.pdf, 参照: 2015-7-7.

[8] “Pencil+ 3”：PSOFT, http://www.psoft.co.jp/jp/product/pencil/index.html, 参照: 2015-7-7.

[9] Tomas Akenine-Moller: 『リアルタイム レンダリング 第 2 版』，ボーンデジタル，ISBN: 978-4939007354, 2006

[10] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, Richard Riesenfeld: “Interactive Technical Illustration”, Proceedings of the 1999 symposium on Interactive 3D graphics, pp. 31-38, 1999

[11] 松尾隆志, 三上浩司, 渡辺大地, 近藤邦雄: “リアルタイム 3DCG における物体の形状を考慮した輪郭線の誇張表現手法の提案”，芸術科学会論文誌 Vol. 10, No. 4, pp. 251-262, 2011

[12] 松尾隆志，三上浩司，渡辺大地，近藤邦雄: “形状の特徴や動的な変形を考慮したリアルタイム 3DCG における輪郭線の誇張表現手法”，映像情報メディア学会誌 Vol67, No2, pp. J36-J44, 2013

[13] 奥屋武志，藪野健，大谷淳，高橋信之: “ポリゴンモデルにおける法線ベクトルを用いた曲率の高速計算”，電子情報通信学会総合大会講演論文集 2013 年情報・システム（2）, 20, 2013

奥屋 武志

2012 年早稲田大学基幹理工学部表現工学科卒業．2014 年早稲田大学基幹理工学研究科表現工学専攻修士課程修了．現在は同専攻博士課程に在籍．2015 年より早稲田大学助手．CG 制作の効率化，ノンフォトリアリスティックレンダリング，画像処理を用いた表現技法の開発に興味を持つ．芸術科学会会員．

田中 克明

早稲田大学基幹理工学部表現工学科に在学．リアルタイムレンダリング，高速ジオメトリ処理，メディアエルゴノミクスに興味を持つ．

坂井 滋和

1980 年東京工業大学卒業．1984 年からフリーランスの CG クリエイターとして活動，学習研究社や NHK において CG 映像を制作．1994 年より九州芸術工科大学，2001 年から早稲田大学にて教育研究に従事する．主な作品としては，NHK スペシャル『銀河宇宙オデッセイ』（1989），同『ナノ・スペース』（1992），同『生命 40 億年はるかな旅』（1994），ETV 宇宙デジタル図鑑（1998）などがある．現在の専門分野は CG，サイエンティフィック・ビジュアライゼーション．