GPU による細分割曲面の意匠形状評価*

金井　崇　　安井 悠介
慶應義塾大学 環境情報学部　先端力学シミュレーション研究所
kanai@sfc.keio.ac.jp　　yasui@astom.co.jp

概要．本論文では，細分割曲面の反射線による意匠形状評価をすべて GPU 上で行うための手法について提案する．反射線は，その形状が曲面形状のわずかな変化に敏感であることから，曲面評価に適したツールである．このことは暗に，反射線は正確に計算する必要があることを示している．我々は， GPU 上のフラグメントプログラムの中で，平面光源テクスチャを用いた直感的で頑健な反射線の計算を行うことができる．また， GPU 上の各フラグメントにおいて，細分割曲面上の正確な位置と法線を求めるための手法についても述べる．我々の提案する反射線計算のための枠組みは，細分割レベルに依存せず，粗いレベルの細分割ポリゴンにおいても正確な計算が可能であることを示す．

1　はじめに

意匠形状の曲面品質評価は，工業用設計・製造分野において，三次元 CAD/CAM システムの普及とともにますます重要な課題となっている．特に，自動車の外板ボディの設計において，デザイナは平行な蛍光灯をクレイモデルに照射し，その反射像を観察することで曲面の品質をチェックしている．彼らは，反射光により外観をチェックし，像の歪みを検出し，また該当する領域をどのように修正すべきかを精査する．

このような光学的現象をコンピュータ上で人工的にシミュレートするために，我々はしばしば，曲面の反射像の曲線を，曲面の幾何，視点，および光源の位置から数値的に計算する．このような曲線は反射線 (*reflection line*) [6, 5] (図 1) と呼ばれる．初期の CAD システムでは，計算機環境の制限から，より簡単なシミュレーションモデルである擬似ハイライト線 (*pseudo-highlight line*) が用いられていた．さらに，ハイライト線 (*highlight line*) [1] と呼ばれる評価線が用いられる場合も多い．これらの線は総称して特徴線 (*characteristic line*) と呼ばれる [4]．これらの線は，曲面の品質をチェックするだけでなく，曲面修正のためのガイドとしても用いられる [2, 10, 7]．ただし，これまでの手法の多くが Runge-Kutta 法 [8] のような数値積分法による数値計算を利用しており，計算安定性等の問題点があった．

本論文では，細分割曲面に対する反射線の計算および表示を，プログラマブル機能を備えたグラフィックスハードウェア (*programmable graphics hardware*, GPU) によって行うための手法を提案する．近年の GPU の発展の度合いは CPU のそれをも凌ぐ勢いである． GPU の機能の一つであるプログラマブルシェーダにより，ポリゴンの描画速度が高速になっただけでなく， GPU の多様な利用法が考えられるようになった．

我々の手法はキューブ環境マッピング (*cubic environment mapping*) [3] やゼブラマップ (*zebra mapping*) と似た方法であるが，より正確な反射線を計算できるという点で優れている．

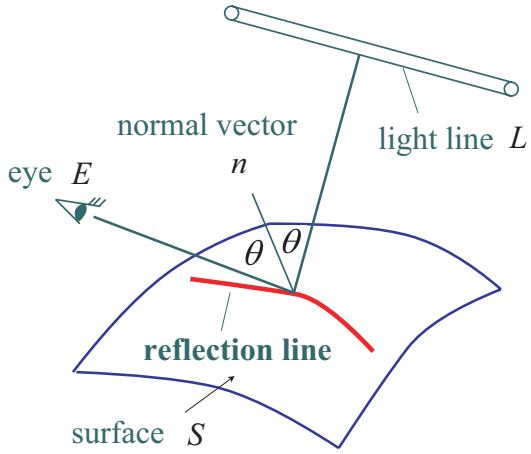

図 1: 反射線.

本論文の主なポイントは，ピクセル毎の浮動小数点演算を用いることである．これは， nVIDIA 社の GeForce FX や ATI 社の RADEON 9700/9800 のような「第二世代」と呼ばれるプログラマブル GPU に備わっている新しい機能の一つである．ここでは，反射線群の一連の計算を GPU のみを用いて行う．また我々は，細分割曲面のフラグメント単位での位置や，法線算出のための一階導関数ベクトルの計算を行うための手法をも提案する．これより，粗いレベルの細分割ポリゴンにおいても高品質な反射線を計算することができる．

2　GPU を用いた反射線の計算

本節では， GPU 上での反射線の計算手法についての基本的なアイデアと，その実装方法について述べる．

2.1　基本的アイデア

反射線 [6, 5] は，線光源が反射した時に観察者が目に見える曲面上の曲線のことである (図 1)．反射線の式は以下

*Aesthetic Shape Interrogation of Subdivision Surfaces on GPU, by Takashi KANAI (Keio University, Faculty of Environmental Information) and Yusuke YASUI (Advanced Simulation Technology Of Mechanics Inc.,ASTOM)

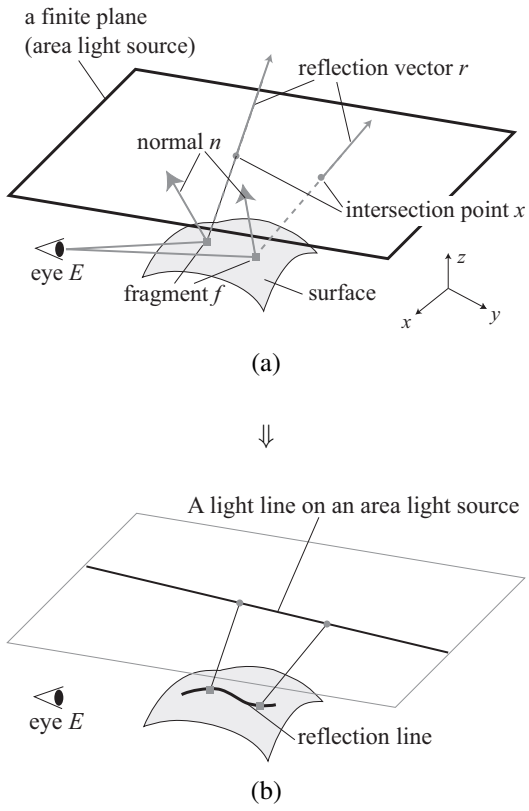

図 2: GPU を用いた反射線算出の基本的アイデア.

の通りである.

$$\frac{S-E}{|S-E|} - 2\left(\frac{S-E}{|S-E|}\cdot n\right) n = \frac{L-S}{|L-S|}, \tag{1}$$

ここで S, n, L, E はそれぞれ曲面上の一点，その点における法線ベクトル，線光源上の一点，視点を示す．式 (1) の意味は以下の通りである.

> 反射線は，曲面上の点 f から視点へのベクトルと法線ベクトル n との角度が，f から光源上の点 L へのベクトルと n の角度と等しくなるような曲面上の点列である.

上の定義は，以下のように解釈することもできる:

> 視点 E から曲面上の点 f へのベクトルに対し，反射ベクトル r は法線ベクトル n により計算される．この反射ベクトルを延長した直線と線光源との交点 x' が見つかったとき，x' に対応する曲面上の点列 f が反射線である.

我々のアイデアは後者の定義に基づく．図 2 にこの考えを図示したものを示す．線光源群を用いるかわりに，面光源を直接曲面上に置く (図 2(a))．これは線光源を等間隔に並べたものとみなすことができる.

視点 E から曲面 S 上の点 f へのベクトルに対し，反射ベクトル r は法線ベクトル n を用いて計算される．r を延長した直線が面光源の平面と交差するならば，f は反射点となるはずである．ここで x をそのような交点とする．上記の計算を曲面全体にわたり行ない，f から x への写像 $f \mapsto x$ を構築する．ここでこの写像は必ずしも一対一写像にはならないことに注意.

この写像の構築後に面光源上に線を定義する (図 2(b)). もし x がこの線上に存在すれば，それは x' として定義され，f は反射線上の一点となる.

2.2 アルゴリズム

ここでは我々の基本的なアイデアに基づき，反射線を計算するためのアルゴリズムについて述べる．このアルゴリズムはレンダリングプロセスの中で行われる.

まず，視点位置，面光源の幅と高さ，線光源群としてのテクスチャを入力とする．レンダリングプロセスの中で，ポリゴンはフラグメントに分解され，それぞれのフラグメントは曲面上の位置と法線ベクトルを持っているものとする．それらと視点位置より，フラグメント上で反射ベクトルを計算し，このベクトルと面光源との交点を計算する．線光源群に相当するテクスチャは，面光源としてマッピングされている．求めた交点に対する面光源上のテクスチャの色を取得し，それをフラグメントの色として割り当てる．もし交点が線光源上の一点であれば，そのフラグメントは反射線上の一点となる.

上記の計算を，曲面上のすべてのフラグメントに対して適用することにより，反射線が計算できる.

3 GPU を用いたフラグメント毎の細分割曲面の計算

本節では，我々のアルゴリズムを細分割曲面に適用することを考える．一般に細分割曲面のレンダリングにおいて，曲面を表示するために，制御ポリゴンを数回細分割した細分割ポリゴンが用いられる．必要であればそれぞれの頂点における極限点を計算して置き換える．一方，レンダリングプロセスの中のラスタライズ化の部分で，各フラグメントにおける位置は，プリミティブの頂点の持つ位置の線形補間により計算される．よって，この位置は正確なものではなく，ポリゴンが粗いときには問題となる．さらに，反射線の形状は，曲面の幾何に非常に敏感である．よって，我々は反射線を正確に計算するために，各フラグメントにおける極限曲面の位置と法線ベクトルが必要となる.

この問題に対処するため，我々は Stam によって提案された手法 [9] を利用することにする． Stam は，任意のパラメータにおける細分割曲面上の正確な位置の計算を可能とした．ここでは，このアルゴリズムをプログラマブル GPU の中でどのように実装すべきかを述べる.

3.1 任意パラメータでの細分割曲面の正確な計算

本節では Stam の手法 [9] を概観する． [9] では，制御ポリゴンを用いて，任意のパラメータにおける Catmull-Clark 細分割曲面の位置とその導関数ベクトルを計算できることを示している．以下にその導出方法について述べる.

ここではすべての制御ポリゴンは四辺形であるものとする．まず初めに，入力制御ポリゴンを細分割ルールにより一回細分割する．すると，それぞれのポリゴンはせいぜい一つの非正則点 (*extraordinary point*) しか持たなくなる．非正則点を四隅に持たない正則四辺形の極限曲面は一様双三次 B スプライン曲面であり，簡単にパラメータ化できる．非正則点を含む四辺形は細分割により四つのパッチに分割され，そのうち三つのパッチは同様に一様双三次 B スプライン曲面により表すことができる．

二次元パラメータ空間の点 (u, v) に対する細分割曲面の位置は次のように表現される [9]:

$$s(u,v) = \hat{C}_0^T \Lambda^{n-1} X_k b(t_{k,n}(u,v)). \tag{2}$$

それぞれの位置は Ω_k^n によって評価され， n と k は二つのパラメータ u, v によって決定される．式 (2) の $\hat{C}_0$ は次のように表せる:

$$\hat{C}_0 = V^{-1} C_0, \tag{3}$$

ここで C_0 は四辺形の周りの制御点列を表す $(2N+8)$ 次元の列ベクトルである． N は非正則点の個数を示す． V は，その列が細分割行列の固有ベクトルであるような，逆行列可能な行列である．これは $(2N+8) \times (2N+8)$ の正方行列であり，よって $\hat{C}_0$ も $(2N+8)$ 次元の列ベクトルとなる．

式 (2) において， Λ は細分割行列の固有値を含む対角行列である． Λ の i 番目の要素は，行列 V の i 番目の列に相当する固有ベクトルの固有値である． X_k は固有基底関数の係数と呼ばれ，パラメータ k に依存し $(2N+8) \times 16$ 個の要素からなる． $b(u,v)$ は二次元パラメータ (u,v) に対する 16 個の B スプライン基底関数のベクトルである． $t_{k,n}(u,v)$ は (u,v) を Ω_k^n 上のパラメータに変換する関数であり，以下のように表せる:

$$\begin{aligned} t_{1,n}(u,v) &= (2^n u - 1, 2^n v), \\ t_{2,n}(u,v) &= (2^n u - 1, 2^n v - 1), \\ t_{3,n}(u,v) &= (2^n u, 2^n v - 1). \end{aligned} \tag{4}$$

これより，式 (2) を書き直すと，

$$s(u,v) = \sum_{i=1}^{2N+8} \mathbf{p}_i \, (\lambda_i)^{n-1} \sum_{j=1}^{16} x_{ijk} \, b_j(t_{k,n}(u,v)), \tag{5}$$

ここで， $\mathbf{p}_i$ はベクトル $\hat{C}_0$ の i 番目の要素を， λ_i は Λ の i 番目の対角要素を， x_{ijk} は X_k の i 行 j 列の要素を，そして b_j は 三次 B スプライン基底関数の j 番目の要素をそれぞれ示す．

u と v に対するそれぞれの偏導関数は，式 (5) における三次 B スプライン基底関数の偏微分により計算できる．法線ベクトルは，これら二つの偏導関数の外積により計算できる．

関数 ψ_i は以下のように定義される:

$$\psi_i(u,v) = (\lambda_i)^{n-1} \sum_{j=1}^{16} x_{ijk} \, b_j(t_{k,n}(u,v)). \tag{6}$$

この ψ_i を用いて，式 (5) は以下のように書き直せる:

$$s(u,v) = \sum_{i=1}^{2N+8} \psi_i(u,v) \, \mathbf{p}_i. \tag{7}$$

$\psi_i(u,v)$ は固有基底関数と呼ばれる．結果として，任意パラメータに対する細分割曲面の計算は， $\mathbf{p}_i$ と $\psi_i(u,v)$ の線形結合として扱うことができる．

3.2 アルゴリズムにおける入力データ

我々のアルゴリズムはフラグメントプログラムの中で実行される．入力として，あらかじめ前計算されたデータをテクスチャに格納する．さらに，面の id と二つのパラメータ u, v も用意する．面の id が必要な理由としては，ラスタライズされたフラグメント上で面の情報を利用するからである．フラグメントプログラムの中で，テクスチャのルックアップのために使われる．

各フラグメントで面の id とパラメータ u, v を計算するために，あらかじめ細分割曲面を近似するポリゴンを用意する．このポリゴンは，もとの制御ポリゴンを数回細分割することによって生成される．細分割の過程の中で，面の id とパラメータは，分割前のポリゴンより簡単に引き継ぐことができる: 細分割ポリゴンの各面は，レンダリングプロセスの中でラスタライズされ，各フラグメントのパラメータは線形補間により生成される．面の id は分割前のポリゴンより引き継がれる．

細分割ポリゴンは細分割曲面 (の極限曲面) の良い近似であることが望ましい．粗いポリゴンでは，ラスタライズされた位置とパラメータにより計算された細分割曲面の位置との幾何誤差が大きくなる．これは最終的なレンダリング結果に影響を及ぼす．特に，各フラグメントで計算されたデータはポイントとしてレンダリングすることから，メッシュに隙間が生じてしまう．しかしながら，我々の実験では，たとえ折り目や角が入っていたとしても，もとの制御ポリゴンを 2 〜 3 回分割しただけの細分割ポリゴンで十分な近似が得られる．

3.3 テクスチャの準備

本節では，フラグメントプログラムでの計算に必要なテクスチャへのデータの格納方法について述べる．ここで，式 (2) を直接計算することは，フラグメントプログラムの命令数が多くなってしまうため，現在の GPU をもってしても負荷が高いと言える．そこで， $\hat{C}_0^T, \Lambda^{n-1}, X_k$ をそれぞれ独立に格納するかわりに， $\hat{C}_0^T \Lambda^{n-1} X_k$ をあらかじめ計算しておき，その結果をテクスチャに格納する．この式には，パラメータ u, v に依存する係数 n, k を含む．幸いにして，これら二つの係数に割り当てる値の範囲は限られている． n は次のように定義される:

$$n = \lfloor min(-\log_2(u), -\log_2(v)) \rfloor + 1,$$

ここで $\lfloor x \rfloor$ は x より小さい最大の整数を示す．

n の最大値を決定するということは，すなわち，どれだけ小さい u, v を許容するか，ということと等価である．

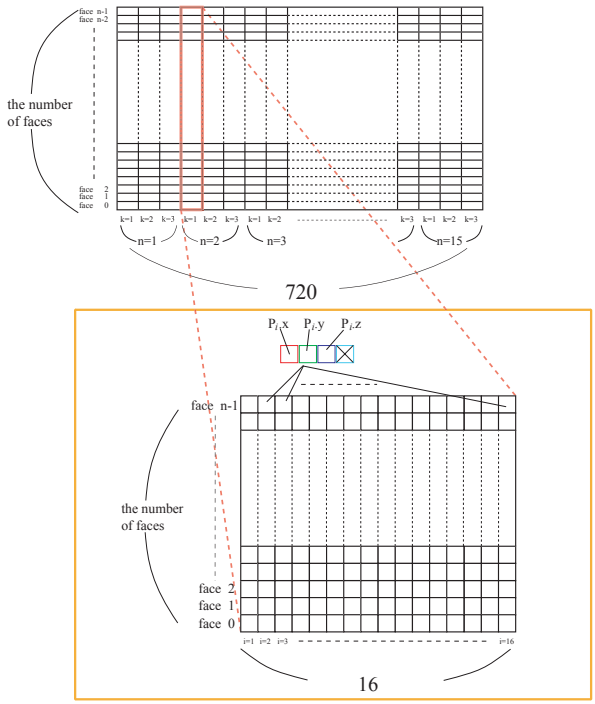

図 3: テクスチャへのデータの格納例.

例えば，もし n の最大値が 10 ならば，$\frac{1}{2^{-10}} = \frac{1}{1024}$ よりも大きいパラメータを利用することができる．ここでは n の最大値を 15 に設定した．これより，$\frac{1}{2^{-15}} = \frac{1}{32768} < 10^{-4}$ よりも大きいパラメータを利用することができ，我々の実験によれば十分な量である．また k の値は 1, 2, 3 のみである．

式 (7) で示したように， Stam は，細分割曲面の計算は固有基底関数と $\mathbf{p}_i$ の線形結合として扱うことができることを提案した [9]．この式をさらに，$\hat{C}_0$ から選択された 16 個の点と三次 B スプライン基底関数の線形結合である P_i を使って置き換える． P_i は以下のように定義される:

$$P_i = \sum_{j=1}^{2N+8} p_j(\lambda_j)^{n-1} x_{jik}. \tag{8}$$

すると， $s(u,v)$ は以下のように計算できる:

$$s(u,v) = \sum_{i=1}^{16} P_i b_i(t_{k,n}(u,v)). \tag{9}$$

各面における $P_i(1 \le i \le 16)$ を入力としてテクスチャに格納する． P_i は パラメータ n,k に依存するため， P_i の可能なすべての組み合わせを格納する． k は 3 通り， n は 15 通りあるので，各面について計 45 通りの P_i の組み合わせが必要となる．図 3 にこれらのパラメータを格納するための具体例について示す．列は面の数に相当する．行の幅は $16 \times 45 = 720$ となる．列の高さはグラフィックスハードウェアの能力によって制限を受ける: もし面の数が制限値を超えた場合は，多重ブロックを利用する．最初のブロックにデータを格納し，続いて制限を超えたデータを次のブロックに格納する．さらに容量を超えた場合は，複数のテクスチャを用いる．

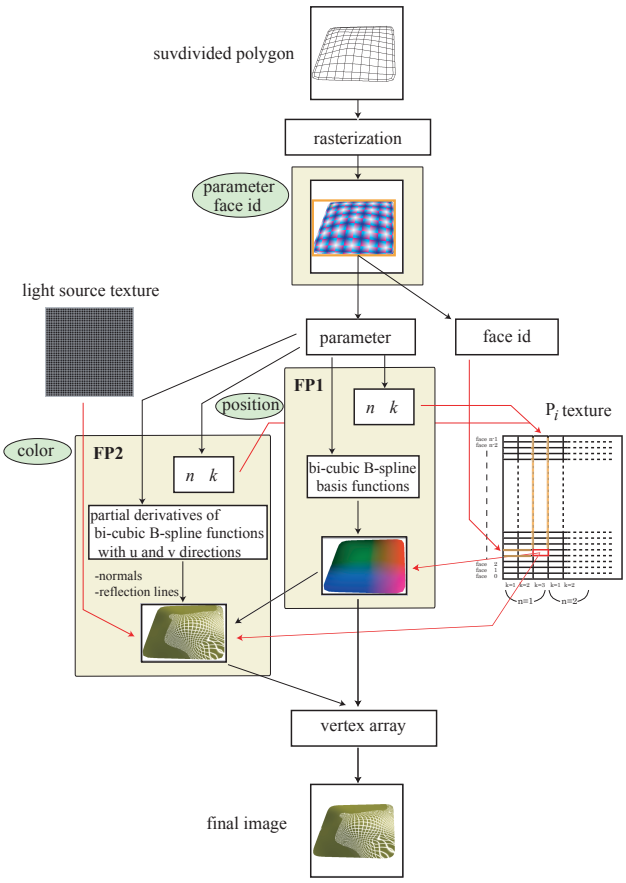

図 4: GPU 上でのアルゴリズムの概要． FP1 と FP2 はそれぞれフラグメントプログラムを示す.

3.4 アルゴリズム

図 4 に我々のアルゴリズムの概要を示す．すべての計算はフラグメントプログラムの中で行われる．アルゴリズムの入力は，前節に示した通りである．

我々は，細分割ポリゴンを入力として用意する．入力メッシュの各面はラスタライズされフラグメント群に分解される．各フラグメントにおいて，面の id と線形補間されたパラメータ u,v が入力メッシュから引き継がれる．これらは一度テクスチャに格納され，フラグメントプログラムの中で位置と色を計算するために利用される．テクスチャに格納する際，すべてのビューポートの範囲を格納するのではなく，ポリゴンを覆う最小の四角形領域を格納することで，フラグメントプログラムの計算負荷を軽減することができる．位置と色は， pbuffer (OpenGL における浮動小数点バッファ) の中で一つの四辺形を描くことで計算される． pbuffer のビューポートの大きさは，転送されたテクスチャと同じである．

曲面の位置は，フラグメントプログラムの中で次のように計算される．まず，現在のフラグメントにおけるパラメータと面の id を，転送されたテクスチャから取得する．これらを使って， P_i が格納されているテクスチャより 16 個の点を選択する．これらの点にアクセスするためのオフセット値は，フラグメントプログラム内で n と k から計算される．また，これらのパラメータに対する三次 B スプライン基底関数を直接計算する．これより，曲面の

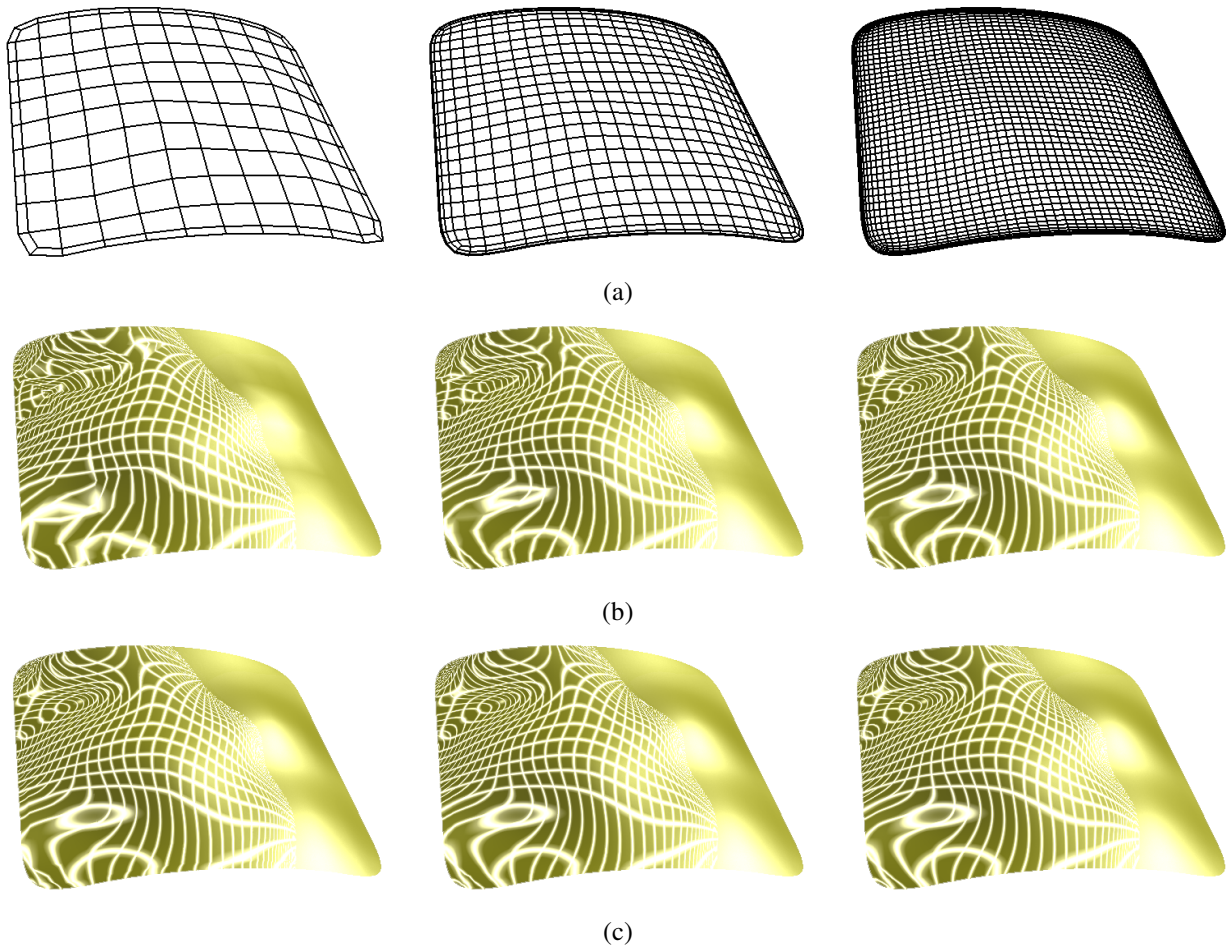

図 5: 細分割ポリゴンにおける反射線の計算結果．左から右へ，それぞれ一回，二回，三回細分割ポリゴンを示す．(a) 線画表示．(b) 細分割ポリゴンを直接利用して計算した結果．(c) 本手法による結果．

位置は基底関数と P_i の線形結合により計算される．この結果を，頂点配列にコピーするだけでなく，次の色の計算を行うフラグメントプログラムへテクスチャとして転送する．

色は，別のフラグメントプログラムにより以下のように計算する．まず，視点と曲面の位置は入力として用意する．次に，u と v 両方向の導関数ベクトルを，曲面の位置と同様にして計算する．フラグメントプログラムの中で使用しているの基底関数は，それぞれの方向で異なる．次に，これらの導関数ベクトルを用いて法線ベクトルを計算する．最後に 2 節の方法を用いて交点を計算し，色を取得する．

各フラグメントにおける結果のデータは，位置と一頂点の色である．最終的な画像は，フラグメントプログラムにおけるこれらの結果を頂点配列にコピーし，ポイントとしてレンダリングする．

4　結果と議論

図 5 に，本手法の正確性を実証するためのいくつかの結果を示す．図 5(a) は三つの細分割ポリゴンの線画表示を示す．左から右へ，それぞれ一回，二回，三回細分割したポリゴンである．図 5(b) は，細分割ポリゴンを直接利用して計算した結果を示す．すなわち，それぞれの頂点は，位置だけでなく法線ベクトルをも持ち，各フラグメントにおける位置と法線ベクトルはラスタライズ処理の際の線形補間により求める．これらの位置と法線ベクトルを用いて，フラグメントプログラムにより反射線を計算する．図 5(b) の結果からわかるように，正確性は細分割レベルに大きく依存する．三回細分割されたポリゴンからは，一回細分割のそれよりも良い結果が得られている．図 5(c) には，本手法による計算結果を示す．それぞれの図は，極限曲面とその反射線の出力結果を示している．これらの結果は，本手法が細分割レベルに依存しないことを示している．

一回，もしくは二回細分割されたポリゴンにおいて，極限曲面への近似は十分ではない．近似が十分でない場合，フラグメントプログラムによって計算された位置が，各ピクセルのそれと異なるという問題が起こる．これは，基本的にはデータの欠如に起因するものである．この問題に対処するための一つの方法として，実際のレンダリングに必要な領域よりも大きいビューポートを利用することが挙げられる．しかしながら，ビューポートを大きくすると，それだけ処理するフラグメントの数が増える．よって実際には，細分割ポリゴンが極限曲面を十分に近似するものであることが望ましい．図 5(c) の左と中央の図では，それぞれ実際よりも 1.7^2, 1.3^2 倍大きいビューポートを利用して

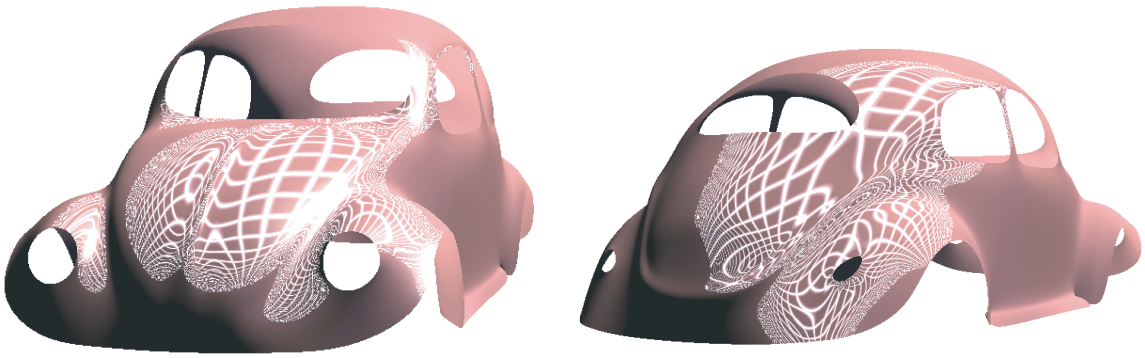

図 6: 1,776 面のモデル (“Volkswagen”, Leif P. Kobbelt 教授の提供による) に対する反射線の出力結果.

#faces	*300*	*1,200*	*4,800*
get parameter (sec.)	0.0008	0.003	0.012

表 1: 面の数と，パラメータおよび面の id の取得にかかる時間との関係.

#fragments	*300×300*	*500×500*	*700×700*
posistion (sec.)	0.03	0.07	0.13
color (sec.)	0.10	0.27	0.52

表 2: フラグメントの数と計算時間の関係.

いる．右図では，実際と同じ大きさのビューポートを使用している．

我々の手法は，視点がオブジェクトに近づいたときにより効果的となる．視点が近い場合，本手法はポリゴンによる方法に比べて正確な反射線を計算できる．それは特に，曲率が大きい部位に関して顕著に現れる．本手法は，すべてのフラグメントにおいて位置と色の計算の正確性を保証するものである．

図 6 では，より複雑な 1,776 面の制御ポリゴンに対し，本手法を適用した結果を示している．

本手法は， Athlon XP 2600+ GeForce FX 5900 Ultra の環境で試行している．本手法の計算は四つのステージからなる．最初のステージは，パラメータと面の id の取得である．二つ目のステージは位置の計算，三つ目は色の計算である．最後のステージがレンダリングである．計算時間を考えた場合，最初のステージのみが面の数に依存し，残りの三つのステージはフラグメントの数，すなわち，ポリゴンが描画されるビューポートの大きさに依存する．表 1 にパラメータと面の id の取得にかかった時間を，表 2 に位置と色の計算にかかった時間を示す．位置と色の計算のフラグメントプログラムの命令数はそれぞれ 280, 181 である．レンダリングは高速のため計算時間は無視できる．

本手法は，他のパラメトリック曲面形式 (ベジエ，クーンズ，エルミートなど) にも適用可能である．その場合フラグメントプログラムの命令数は，細分割曲面の場合に比べ大幅に少なくなることが予想できる．

5 結論と展望

本論文では，反射線を用いたプログラマブル GPU 上での細分割曲面の曲面品質評価手法について提案した．本手法により，粗いレベルの細分割ポリゴンにおいても，正確な反射線を計算できることを実証した．反射線の計算は高速である一方，細分割曲面の位置と導関数ベクトルの計算はかなり遅い．本手法の計算は，現存のグラフィックスハードウェアに全面的に依存しているが，将来的には，ハードウェアの発達により劇的に改善されることを期待する次第である．

参考文献

[1] K.-P. Beier and Y. Chen. Highlight line algorithm for real-time surface quality assessment. *Computer Aided Design*, 26(4):268–277, 1994.

[2] Y. Chen, K.-P. Beier, and D. Papageorgiou. Direct highlight line modification on NURBS surfaces. *Computer Aided Geometric Design*, 14(1):583–601, 1997.

[3] N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, 1986.

[4] M. Higashi, T. Kushimoto, and M. Hosaka. On formulation and display for visualizing features and evaluating quality of free-form surfaces. In *Proc. Eurographics '90*, pp. 299–309. Elsevier, Amsterdam, 1990.

[5] E. Kaufmann and R. Klass. Smoothing surfaces using reflection lines for families of splines. *Computer Aided Design*, 20(2):73–78, 1988.

[6] R. Klass. Correction of local irregularities using reflection lines. *Computer Aided Design*, 12(7):411–420, 1980.

[7] J. Loos, G. Greiner, and H.-P. Seidel. Modeling of surfaces with fair reflection line pattern. In *Proc. Shape Modeling International '99*, pp. 106–115. IEEE CS Press, Los Alamitos CA, 1999.

[8] W. H. Press, W. T. Vetterling, S. A. Teukolsky, and B. P. Flannery. *Numerical Recipes in C++: the Art of Scientific Computing*. Cambridge University Press, 2002.

[9] J. Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Computer Graphics (Proc. SIGGRAPH 98)*, pp. 395–404. ACM Press, New York, 1998.

[10] C. Zhang and F.-F. Cheng. Removing local irregularities of NURBS surfaces by modifying highlight lines. *Computer Aided Design*, 30:923–930, Aug. 1998.