GPUによる細分割曲面の高品質表示法

金井 崇[†] 安井悠介[††]

本論文では，GPUによる細分割曲面の高品質な表示方法について提案する．細分割曲面はベジエ曲面やBスプライン曲面等のパラメトリック曲面と同様，2変数のパラメータから曲面上の任意の点の位置や法線ベクトルを計算できる．このことを利用し，フラグメント単位でそれらの正確な値を計算する．また応用として，CAD/CAMの分野で用いられている反射線を計算する手法についても述べる．我々の提案する表示のための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．

High-quality Display of Subdivision Surfaces on GPU

Takashi Kanai[†] and Yusuke Yasui[††]

In this paper, we propose a method for high-quality display of subdivision surfaces using recent programmable graphics hardware. Arbitrary positions and normal vectors of subdivision surfaces can be computed from two-dimensional parameters as parametric surfaces such as Bézier surfaces or B-spline surfaces do. Using this property, we compute these values for each fragment rasterized from polygons. We also describe a method to compute reflection lines which are used in the area of CAD/CAM as one of applications. As our framework for such a per-fragment evaluation does not depend on the level of subdivision, a precise evaluation can be established even for low levels of subdivided polygons.

1. はじめに

ベジエ曲面やBスプライン曲面に代表される自由曲面は，現在多くのコンピュータグラフィックス（CG）およびCAD/CAMにおける形状モデリングシステムで実装されている．なかでも細分割曲面（*subdivision surface*）[12]は，制御メッシュと呼ばれる多面体を規則的に分割することで，滑らかな曲面形状を表現することが可能である．

これら自由曲面の表示を行う際には，制御点等の自由曲面を構成するデータのほかに，曲面形状を近似するようなポリゴンデータを別に用意するのが一般的である．しかしながら，ポリゴンによって曲面形状を正確に表示する場合，近似の度合いを高めるためにポリゴンの数をより多くする必要があり，そのことが表示にかかる時間を遅らせる一因となる．表示の際ポリゴンの数を調節する方法（たとえばHoppeの手法[4]）もあるが，特別なデータ構造をシステムの中に持つ必要がある等，実装上の負担が大きい．

本論文では，細分割曲面，特にCatmull-Clark細分割曲面[3]の表示を，プログラミング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1つであるプログラマブルシェーダにより，GPUの多様な利用法が考えられるようになっている．また，CgやGLSL等のシェーダ用高級プログラミング言語の登場も，開発を容易にする一助となっている．

本論文の主なポイントは，フラグメントごとの浮動小数点演算を用いることで，表示に必要な細分割曲面上の位置や法線ベクトルの計算をすべてGPU上で行うことにある．これにより，粗いレベルの細分割ポリゴンにおいても，高品質な曲面表示を行うことができる．なお，我々の過去の研究[10]では，GPU上の制約により位置と法線ベクトルの計算をそれぞれ別のシェーダプログラム上に実装する必要があった．本論文では，最新のGPU（*n*VIDIA GeForce 6以降）の機能により，2つの計算を1つのプログラムにまとめることができ，その分，パフォーマンスの向上が期待できる．

† 理化学研究所ものつくり情報技術統合化研究プログラム
Integrated Volume-CAD System Research Program, RIKEN
†† 先端力学シミュレーション研究所
ASTOM Inc.

さらに，本手法の応用として，意匠形状の曲面品質評価ツールとして用いられる反射線[5),6)]の計算を，GPU上で行うための手法についても示す．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，曲面評価に適している．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．

2. 関 連 研 究

本論文では，Catmull-Clark細分割曲面[3)]に焦点を当てるが，我々の手法はLoop細分割曲面[7)]のような他の細分割曲面に対しても適用可能である．Catmull-Clark細分割曲面は，四辺形の制御ポリゴンを再帰的に細分割することにより得られる曲面であり，その極限曲面は，非正則点を除いてC^2連続の双三次一様Bスプライン曲面となる．

Bolzらは，ハードウェアを利用した細分割曲面生成のためのアルゴリズムを提案した[1),2)]．細分割曲面における位置や法線は，基底関数と制御点の線形結合に帰着でき，このことを利用してGPUによる高速な表示を行っている．しかし，この手法の出力は細分割されたポリゴンであり，フラグメントごとの位置と法線は，ポリゴンの頂点における値の線形結合により計算している．よって厳密にいえば，極限曲面の正確な計算を行っているわけではない．これに対し，我々のアルゴリズムは，フラグメントごとに正確な位置と法線の計算を行うことが可能である．

一方，Stamは，細分割曲面における任意のパラメータ値での正確な位置を計算することが可能であることを示した[9)]．また，Zorinらは，区分平滑細分割曲面への拡張を行った[11)]．我々は，Stamのアルゴリズムを GPU上に実装している．

3. GPUを用いたフラグメントごとの細分割曲面の計算

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，特にポリゴンが粗いときには，正確な曲面を表現することが難しい．

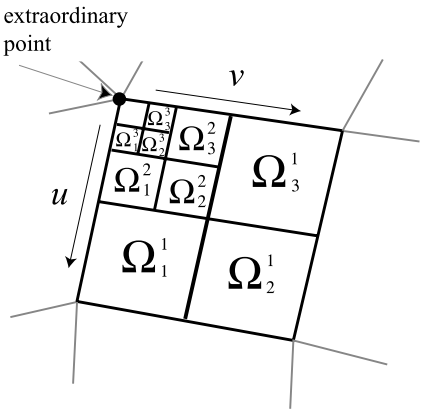


図 1 パッチ Ω^n_k 上での曲面上の位置と導関数の計算
Fig. 1 Positions and their partial derivatives are evaluated on each patch Ω^n_k.

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．この手法では，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1回細分割する．すると，それぞれのポリゴンはせいぜい1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4つのパッチに分割され，そのうち3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（**図 1**）．

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)]：

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

それぞれの位置はパッチΩ^n_kによって評価され，nとkは2つのパラメータu，vによって決定される．式(1)の$\hat{C}_0$は次のように表せる：

$$\hat{C}_0 = V^{-1} C_0, \qquad (2)$$

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 は非正則点の価数を示す．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 (1) 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：

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

これより，式 (1) を書き直すと，

$$s(u,v) = \sum_{i=1}^{2N+8} \mathbf{p}_i \,(\lambda_i)^{n-1} \sum_{j=1}^{16} x_{ijk}\, b_j(t_{k,n}(u,v)), \tag{4}$$

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 (4) 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 2 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：

$$\psi_i(u,v) = (\lambda_i)^{n-1} \sum_{j=1}^{16} x_{ijk}\, b_j(t_{k,n}(u,v)). \tag{5}$$

この ψ_i を用いて，式 (4) は以下のように書き直せる：

$$s(u,v) = \sum_{i=1}^{2N+8} \psi_i(u,v)\, \mathbf{p}_i. \tag{6}$$

$\psi_i(u,v)$ 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，制御ポリゴンの面の ID 番号と 2 つのパラメータ u，v も用意する．ここで，制御ポリゴンの面の ID 番号が必要な理由として，後述するフラグメントプログラム上での細分割曲面の位置と法線ベクトルの計算において，制御ポリゴンの面を特定する必要があるからである．

各フラグメントで制御ポリゴンの面の ID 番号 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，制御ポリゴンの面の ID 番号とパラメータは，分割前のポリゴンより簡単に引き継ぐことができる．すなわち，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．制御ポリゴンの面の ID 番号は，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 (1) 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いえる．そこで，$\hat{C}_0^T$，Λ^{n-1}，X_k をそれぞれ独立に格納する代わりに，$\hat{C}_0^T \Lambda^{n-1} X_k$ 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 2 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：

$$n = \lfloor min(-\log_2(u), -\log_2(v)) \rfloor + 1, \tag{7}$$

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：たとえば，もし n の最大値が 10 ならば，

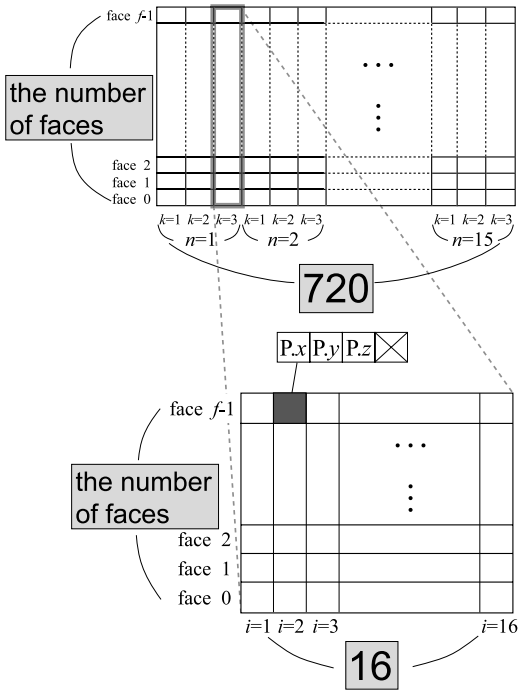


図 2 テクスチャへのデータの格納例
Fig. 2 Procedure of storing data to textures in the two-dimensional table.

$2^{-10} = \frac{1}{1024}$ 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，$2^{-15} = \frac{1}{32768} < 10^{-4}$ 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 (6) 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：

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

すると，$s(u,v)$ は以下のように計算できる：

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

各パッチにおける $P_i(1 \le i \le 16)$ 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合せを格納する．k は 3 通り，n は 15 通りあるので，各パッチについて計 45 通りの P_i の組合せが必要となる．**図 2** 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．

3.4 細分割曲面表示アルゴリズムとその実装

表示に必要な計算は，主に GPU 上のフラグメントプログラムの中で行われる．本アルゴリズムは 2 パスもしくは 3 パスによるアルゴリズムとなっている．以下に，アルゴリズムの詳細およびその実装方法について述べる．**図 3** には，例として 3 パスから構成される GPU による細分割曲面表示アルゴリズムの手順を示す．

1 回目のパスにおいて，入力した細分割ポリゴン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．フラグメントプログラムの中で，細分割ポリゴンから引き継がれた制御ポリゴンの面の ID 番号 id と，線形補間されたパラメータ u，v をそのまま浮動小数点バッファに出力する☆．これらはいったん浮動小数点テクスチャとして格納され，次のパスのフラグメントプログラムの中での位置と色の計算に利用される．

2 回目のパスでは，細分割曲面の位置と法線ベクトル，および色が計算される．ここでは，1 パス目で格納した浮動小数点テクスチャと同じ大きさの 1 枚の四辺形ポリゴンを描画する．そうすることで，フラグメントプログラム内で，各フラグメントにおいて浮動小数点テクスチャの対応するピクセル内のデータ（ここではパラメータ u，v と ID 番号 id）を取得することができる．

曲面の位置と法線，および色は，フラグメントプログラムの中で，各フラグメントにつき以下の手順で計算される．

(1) パラメータ u，v と ID 番号 id を，浮動小数点テクスチャの対応するピクセルより取得する．そのうち，u，v を利用して式 (7) より n と k を得る．n，k および id を用いて，制御点テクスチャから 16 個の P_i（式 (8)）を取得する．

(2) パラメータ u，v より，三次 B スプライン基底関数，および u，v 各方向の 1 階導関数を計算する．

☆ OpenGL では，浮動小数点バッファとして pbuffer を利用する．各ピクセルに対し 4 つの（RGBA）浮動小数点を出力できるが，このうち 2 つにパラメータ u，v を，1 つに ID 番号 id を格納する．

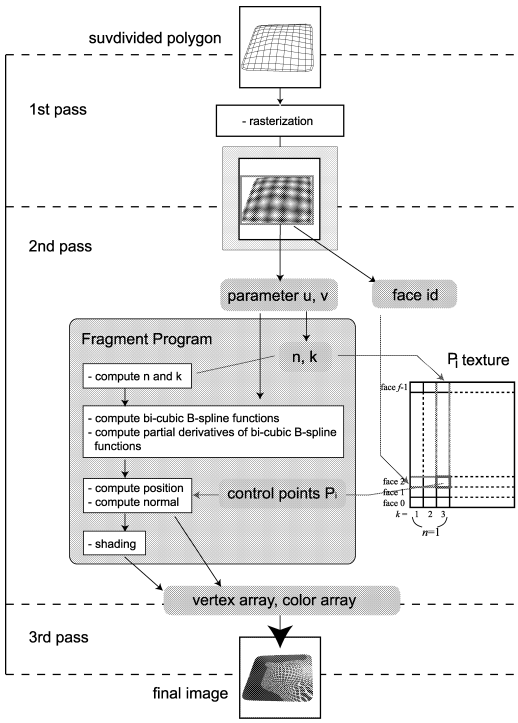


図 3 GPU 上での 3 パスによる描画アルゴリズムの概要
Fig. 3 Overview of our 3-passes display algorithm on GPU.

これらの計算に必要な基底関数の係数は，フラグメントプログラム内に定数として記述しておく．

(3) 制御点テクスチャより取得した 16 個の P_i と基底関数の線形結合より，パラメータ u，v に対する細分割曲面の位置を計算する．また，同様にして u，v 各方向の接線ベクトルを計算し，その外積をとって法線ベクトルを計算する．

(4) 細分割曲面の位置と法線ベクトルを用いてシェーディング処理を行い，色を計算する．

これらのフラグメントプログラムにおける処理の出力は，細分割曲面の位置および色のデータであり，これらのデータをそれぞれ頂点配列（*vertex array*），色配列（*color array*）として浮動小数点テクスチャに格納する☆．

3 回目のパスでは，頂点配列と色配列を利用して，色情報を付加した点群の描画を行い，最終画像を得る．

ここで，2 回目のパスのフラグメントプログラムによる出力は，細分割曲面の位置および色の 2 つのデータである．よって，複数の出力結果を別々のテクスチャに格納するための方法が必要となる．文献 10) では，この部分のパスを 2 回に分け，位置の計算と色の計算で別々のフラグメントプログラムを利用していたが，2 つのパスの間で浮動小数点テクスチャのコピーが発生するため，その分パフォーマンスが低下するという問題があった．

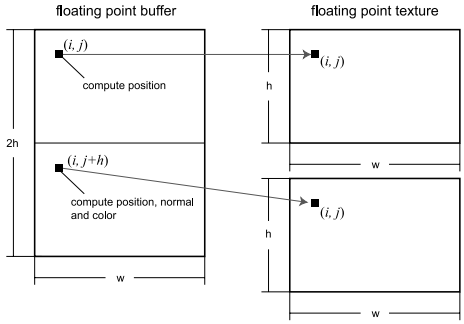


図 4 浮動小数点バッファの重ね合わせによる頂点と色の計算
Fig. 4 Calculation of vertices and colors by the combination of two floating point buffers.

本手法では，1 回のパスで計算結果を得るために，以下の 2 通りの方法を提案する：1 つ目の方法として，描画する浮動小数点バッファの領域を倍にする．すなわち，**図 4** のように，浮動小数点バッファを上下に並べ，上半分では位置の計算のみを，下半分では位置，法線ベクトル，および色の計算を行う．そして，計算されたバッファの上半分を頂点配列，下半分を色配列と見なすようにする．

2 つ目の方法として，*n*VIDIA GeForce 6 以降の GPU で採用された機能の 1 つである，MRT（*Multiple Render Targets*）を利用する．この機能では，1 つのフラグメントプログラム内で複数の出力バッファに値を指定することが可能である（最大で 4 つ）．1 つ目の出力バッファに頂点を，2 つ目の出力バッファに色を指定することで，1 回のパスにより両方の結果を得ることができる．

なお，MRT を用いた場合，上記の 2 パス目の処理を 1 パス目に含めることが可能であり，その結果，全体で 2 パスで処理することができる．

4. 計算時間に関する検討

ここでは，本手法にかかる計算時間を検討する．**図 5** に，計算時間の評価に用いたスプーン形状の細分割曲面データ（頂点数 118，面数 116）を示す．図 5 (b)–(d) は入力として用いたポリゴンであり，制御ポリゴンにそれぞれ 1 回，2 回，3 回の細分割操作を施し

☆ OpenGL では，Render To Texture 拡張と VAR（Vertex Array Range）拡張によって，格納したテクスチャをそのまま頂点配列や色配列と見なす機能があり，この機能を使うことで，フラグメントプログラムの計算結果を直接，次のパスの描画に利用することができる．

表 1 図 5 の例題における面の数，画面解像度，パスごとの計算およびフレームレート
Table 1 The number of faces, display resolution, computation time in each pass and fps for the example in Fig. 5.

Fig. 5	methods	#faces	resolution	computation time (sec.)			fps
				1st pass	2nd pass	3rd pass	
(b)	1	464	512×512	3.66×10^{-4}	3.00×10^{-2}	1.95×10^{-3}	29.98
(c)	1	1,856	512×512	1.12×10^{-3}	2.90×10^{-2}	2.05×10^{-3}	27.97
(d)	1	7,424	512×512	4.57×10^{-3}	3.29×10^{-2}	1.85×10^{-3}	25.14
(b)	2	464	512×512	2.89×10^{-2}		1.98×10^{-3}	32.53
(c)	2	1,856	512×512	2.97×10^{-2}		2.07×10^{-3}	31.48
(d)	2	7,424	512×512	3.06×10^{-2}		1.78×10^{-3}	30.01
(b)	2	464	$1{,}024 \times 1{,}024$	5.32×10^{-2}		1.96×10^{-2}	13.67
(c)	2	1,856	$1{,}024 \times 1{,}024$	5.62×10^{-2}		1.94×10^{-2}	13.15
(d)	2	7,424	$1{,}024 \times 1{,}024$	6.25×10^{-2}		1.97×10^{-2}	12.17

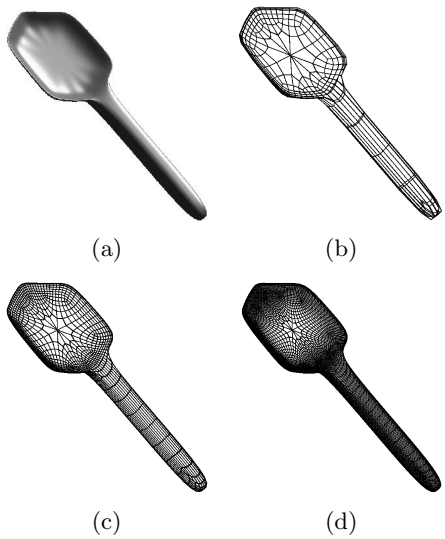


図 5 計算時間の計測に用いた例題．(a) 細分割曲面．(b) 1 回細分割ポリゴン．(c) 2 回細分割ポリゴン．(d) 3 回細分割ポリゴン
Fig. 5 Example for estimating computation times. (a) Subdivision surface. (b) Once-subdivided polygon. (c) Twice-subdivided polygon. (d) Three times subdivided polygon.

ている．なお，計算時間の計測は CPU に Athlon64 3500+，GPU に GeForce 7800 GTX（PCI-Express 接続）を搭載した PC 上で行った，コンパイラは Visual C++ を用い，シェーダのプログラミング言語として Cg[8] を利用した．

表 1 に，本手法によるスプーン形状に対し，3 つの入力細分割ポリゴン（図 5 (b)–(d)），2 つの異なる解像度（512×512，$1{,}024 \times 1{,}024$）のもとでそれぞれパスごとに計測した計算時間（秒），およびフレームレートを示す．手法（methods）の 1 は浮動小数点バッファを上下に並べた場合の手法（以下，手法 1）を，2 は MRT を利用した場合の手法（以下，手法 2）を示す．計算時間において，1st pass は入力ポリゴンのフラグメント群への分解と面番号，パラメータのテクスチャへの出力，2nd pass は，位置および法線ベクトル，色の計算およびテクスチャへの出力，そして 3rd pass は，点によるレンダリングにかかった時間をそれぞれ表している．なお，ここでの計算時間およびフレームレートはすべて，100 フレーム分の計算の平均時間として算出している．

表 1 の各パスにおける計算時間をみると，計算時間の最もかかる処理は，2 パス目の頂点，法線ベクトル，色の計算部分であることが見てとれる．2 パス目はフラグメントプログラムにより実行され，3 パス目は計算されたピクセルの数がそのまま描画される頂点の数となる．このため，2，3 パス目は，処理するピクセルの数によって計算時間が増減する．実際，描画領域を大きくすると（$512 \times 512 \rightarrow 1{,}024 \times 1{,}024$），2，3 パス目の計算時間が増えているのが確認できる．一方，入力ポリゴンの面数は 1 パス目の処理時間に影響し，2，3 パス目にはまったく影響しないことが分かる．

手法 1 と手法 2 を比べてみると，手法 2 の方が手法 1 より 3〜5 fps 程度高速に計算できることが分かる．これは，手法 2 の方がパスの数が少ないことと，手法 1 は描画領域が 2 倍になっているため，その分 2 パス目の処理に時間がかかっていることが原因である．

5. 反射線の表示への応用

本手法は，フラグメント単位で正確な細分割曲面の位置と法線を計算することができる．このため，これらの情報を必要とするような曲面の高品質な表示手法に応用できると考えられる．

その一例として，ここでは反射線[5),6)] の計算とその結果について述べる．反射線は，線光源が反射したときに観察者が目に見える曲面上の曲線のことであり，古くから工業製品の意匠設計等に用いられてきた．

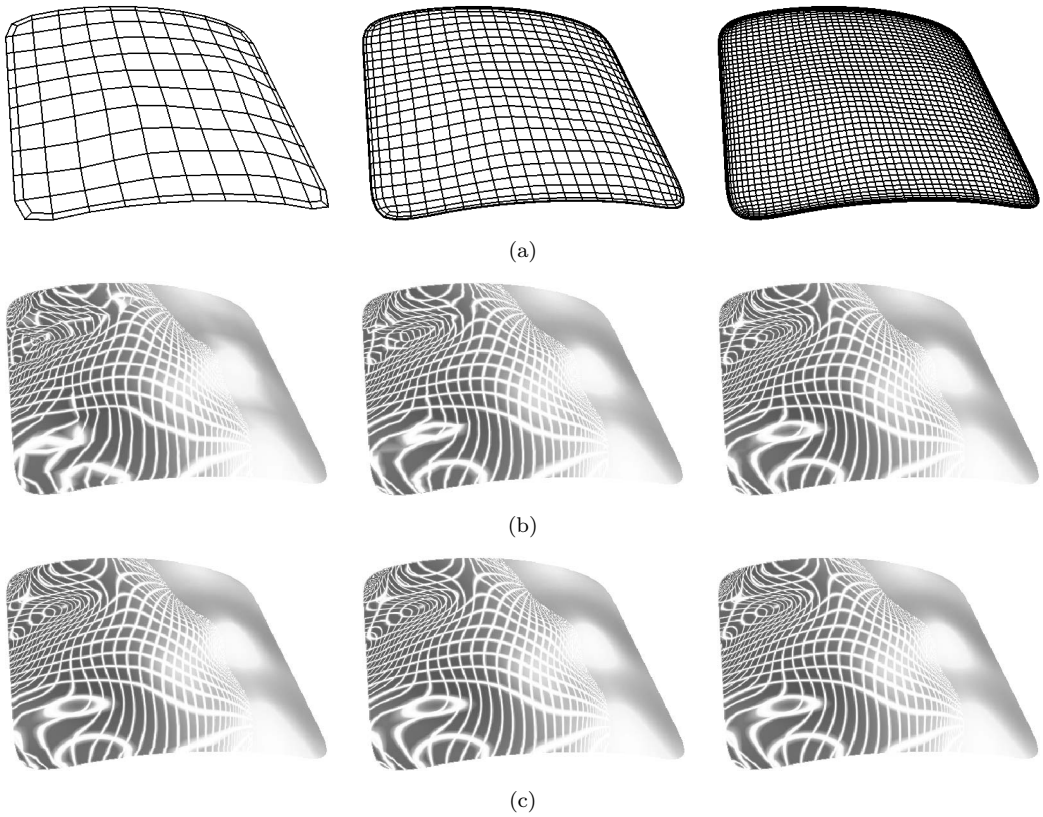


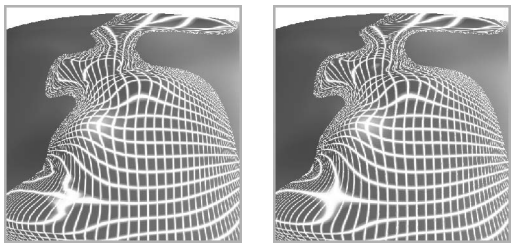
図 6 細分割ポリゴンにおける反射線の計算結果．左から右へ，それぞれ 1 回，2 回，3 回細分割ポリゴンを示す．(a) 線画表示．(b) 細分割ポリゴンを直接利用して計算した結果．(c) 本手法による結果

Fig. 6 Reflection lines with subdivided polygons. From left to right: once, twice and three times subdivided polygons. (a) Wire-frame display. (b) Calculated by using polygons directly. (c) Calculated by our algorithm.

ここでは，反射線を計算するためのアルゴリズム[10]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 1 点であれば，そのフラグメントは反射線上の 1 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．

図 6 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 6 (a) は 3 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 1 回，2 回，3 回細分割したポリゴンである．図 6 (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 6 (b) の結果から分かるように，正確性は細分割レベルに大きく依存する．3 回細分割されたポリゴンからは，1 回細分割のそれよりも良い結果が得られている．図 6 (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．

我々の手法は，視点がオブジェクトに近づいたとき



(a) (b)

図 7 反射線の拡大図表示．ここでは 3 回細分割ポリゴンを入力としている．(a) ポリゴンによる計算結果．(b) フラグメント単位での計算結果

Fig. 7 Close-up view of reflection lines. We use three times subdivided polygons in this case. (a) polygon based calculation (b) fragment based calculation.

により効果的となる．**図 7** にそのような場合の表示例を示す．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．

6. 結論と展望

本論文では，プログラマブル GPU 上での Catmull-Clark 細分割曲面の曲面表示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，正確な位置および法線をフラグメント単位で計算することで，高品質な表示ができることを反射線の計算例を通じて実証した．

今後の展望として，1 つに他の形式の曲面形式への本手法の適用があげられる．Loop 細分割曲面や Doo-Sabin 細分割曲面等の他の細分割曲面形式や，ベジエ曲面，B スプライン曲面等のパラメトリック曲面への適用が考えられる．もう 1 つに，他の表示手法への適用があげられる．CG の分野では，バンプマッピングやディスプレイスマッピング等の効果に，CAD の分野では曲率線や等高線など，曲面の幾何的性質を調べるための表示に，本手法は適用できると考えている．

参 考 文 献

1) Bolz, J. and Schröder, P.: Rapid Evaluation of Catmull-Clark Subdivision Surfaces, *Proc. 7th International Conference on 3D Web Technology* (*Web3D Symposium*), pp.11–17, ACM Press, New York (2002).
2) Bolz, J. and Schröder, P.: Evaluation of Subdivision Surfaces on Programmable Graphics Hardware (2003). submitted for publication.
3) Catmull, E. and Clark, J.: Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes, *Computer Aided Design*, Vol.10, No.6, pp.350–355 (1978).
4) Hoppe, H.: Progressive Meshes, *Computer Graphics* (*Proc. SIGGRAPH '96*), pp.99–108, ACM Press, New York (1996).
5) Kaufmann, E. and Klass, R.: Smoothing Surfaces Using Reflection Lines for Families of Splines, *Computer Aided Design*, Vol.20, No.2, pp.73–78 (1988).
6) Klass, R.: Correction of Local Irregularities Using Reflection Lines, *Computer Aided Design*, Vol.12, No.7, pp.411–420 (1980).
7) Loop, C.: Smooth Subdivision Surfaces Based on Triangles, Master's thesis, University of Utah, Department of Mathematics (1987).
8) Mark, W.R., Glanville, R.S., Akeley, K. and Kilgard, M.J.: Cg: A System for Programming Graphics Hardware in a C-like Language, *ACM Trans. Graphics* (*Proc. SIGGRAPH 2003*), Vol.22, No.3, pp.896–907 (2003).
9) Stam, J.: Exact Evaluation Of Catmull-Clark Subdivision Surfaces At Arbitrary Parameter Values, *Computer Graphics* (*Proc. SIGGRAPH '98*), pp.395–404, ACM Press, New York (1998).
10) Yasui, Y. and Kanai, T.: Surface Quality Assessment of Subdivision Surfaces on Programmable Graphics Hardware, *Proc. Shape Modeling International 2004*, pp.129–136, IEEE CS Press, Los Alamitos CA (2004).
11) Zorin, D. and Kristjansson, D.: Evaluation of Piecewise Smooth Subdivision Surfaces, *The Visual Computer*, Vol.18, No.5–6, pp.299–315 (2002).
12) Zorin, D. and Schröder, P.: Subdivision for Modeling and Animation, *SIGGRAPH 2000 Course Notes*, ACM SIGGRAPH (2000).

(平成 17 年 7 月 5 日受付)
(平成 17 年 11 月 1 日採録)

金井　崇（正会員）

昭和 44 年生．平成 10 年東京大学大学院工学系研究科精密機械工学専攻博士課程修了．同年理化学研究所基礎科学特別研究員．平成 12 年慶應義塾大学環境情報学部専任講師．平成 17 年理化学研究所ものつくり情報技術統合化研究プログラム研究員．現在に至る．形状モデリングの CAD/CAM および CG への応用に関する研究に従事．博士（工学）．精密工学会，画像電子学会，芸術科学会，ACM，IEEE CS 各会員．平成 13 年度山下記念研究賞受賞．

安井　悠介

昭和 56 年生．平成 16 年慶應義塾大学環境情報学部卒業．同年（株）先端力学シミュレーション研究所入社．現在に至る．形状モデリングの研究に従事．