GPU による非一様 B スプライン曲面の高速かつ高品質な表示手法

金井　崇
東京大学大学院総合文化研究科

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パラメトリック形式の自由曲面は，現在でも多くの CAD システムにおいて実装されている．中でも，B スプライン曲面は，基底関数のローカルサポートや任意の数の制御点に対する定義など，よく知られている多くの利点により，形状モデリングによく使われている．特に設計・製造の分野では，設計により得られるこれらパラメトリック曲面の評価に，視覚評価ツール [5, 6] が用いられる．これらのツールは，曲面の滑らかさや曲率の変化，凸凹を視覚的に評価するために用いられる．このようなアプリケーションのためには，より正確な幾何量の評価が必要となる．

一方で，B スプライン曲面を含むパラメトリック曲面の表示には，通常それをテセレート近似したポリゴンが利用される．レンダリングは，頂点の位置と法線を用いてライティングの計算をし，三角形の頂点ごとに計算された色を線形補間することで行う．よって，曲面上の任意の点におけるこれらの色は，その点における正確な位置や法線を利用しているわけではなく，正しい値を表示しているとはいえない．本研究では，近年進展著しいプログラマブルグラフィックスハードウェア（GPU）を利用して，非一様 B スプライン曲面の高品質な表示のための手法を提案する．我々の手法の特徴は，GPU のフラグメントプログラムの中で，フラグメント単位で正確な非一様 B スプライン曲面の位置や法線ベクトルを計算することにある．このため，オブジェクトのスケール非依存で高品質な表示をすることができる．

我々の手法における技術的な貢献は，B スプライン基底関数の計算を非常に素直な方法で GPU のフラグメントプログラムの中で実装していることにある．ノットベクトルの中から，入力パラメータについて適切なノット区間を計算することにより，関数値の計算コストを減らすことができる．これにより，曲面の評価を含めたリアルタイムレンダリングを実現できる．

2　関連研究

本研究に最も関連深い研究が，Guthe らによって発表されている [4]．彼らは，トリム曲面を含む NURBS をリアルタイムレンダリングする手法を提案している．トリムされる箇所は，最初のパスでサンプルされ，トリムテクスチャとして格納される．2 パス目で，一様にサンプルされたグリッドポリゴン上の頂点毎に曲面が評価される．フラグメントプログラムの中で，トリムテクスチャを利用してトリミングが行われ表示が行われる．この方法の一つの問題点として，入力データをすべて双 3 次の有理ベジエ曲面に変換しているため，次数が 3 よりも大きい NURBS はすべて近似曲面として扱われている．よって，近似を行った時点で，オリジナルの曲面で定義されている幾何学的特徴がすでに失われてしまっている可能性がある．また，もう一つの問題点として，各パッチ毎に描画処理を行っているため，パッチの数が増えるごとに計算量は増加する．

GPU を利用した曲面の表示に関していくつかの研究が行われている [1, 9, 2, 3] が，これらの手法は，しかしながら，最終的にはポリゴンを描画しており，1 節で挙げたような曲面評価の正確性の問題が完全に解決するわけではない．これに対し，細分割曲面に対するフラグメントベースの曲面評価手法が提案されている [10]．本手法は，この手法を基本とし，より一般的な B スプライン曲面への拡張を行っている．

3 非一様 B スプライン曲面の GPU による実装

本節では，非一様 B スプライン曲面に対する，フラグメントベースでの評価を GPU 上で実装するための手法について説明する．なお本節では，まず B スプライン曲線をもとにした B スプライン基底関数の実装方法について述べる．曲面の実装は，基本的には曲線の実装の単純な拡張により行われる．曲面では 2 方向のパラメータに対して二つの B スプライン基底関数を計算する必要があるが，曲線では 1 方向のパラメータだけで計算できる．

p 次の非一様 B スプライン曲線は以下の式で表される（表記は [7] にもとづく）：

$$\mathbf{C}_p(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i, \quad a \le u \le b, \tag{1}$$

ここで $\mathbf{P}_i$ は制御点を，$N_{i,p}(u)$ は非周期かつ非一様なノットベクトル ($m+1$ 個のノット, a, b は $p+1$ 個):

$$\begin{aligned} U &= \{u_0, \ldots, u_m\} \\ &= \{a, \ldots, a, u_{p+1}, \ldots, u_{m-p-1}, b, \ldots, b\}, \end{aligned}$$

により定義される p 次の B スプライン基底関数を示す．

GPU 上に実装する上での問題点として，現在の GPU では，任意の回数に対するループ演算をすることができない：すなわち，for ループでの繰り返し回数を a とすると，a はプログラムの中で固定された回数として定義される必要がある．外部入力や関数の引数として a の値を取得するようなことはできない．これを基底関数の計算に当てはめてみると，B スプライン曲線・曲面における制御点の数 $n+1$（式 (1)）は，GPU 上では固定されている必要があるため，式 (1) を GPU 上でそのまま計算することはできない．

そこでここでは，ノット区間を見つけることを基本とする B スプライン基底関数の実装方法 [7] をもとにする．これは，非ゼロとなる B スプライン基底関数の値が必ず $p+1$ 個となることを利用するものである．

3.1 B スプライン基底関数の計算と曲線・曲面

B スプライン基底関数 $N_{i,p}(u)$ は次式で定義される:

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1 & \text{if} \quad u_i \le u < u_{i+1}, \\ 0 & \text{otherwise}, \end{cases} \\ N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) \\ &\quad + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u). \end{aligned} \tag{2}$$

上記の式 (2) を GPU で実装する上で注目すべき性質として，$u_i \le u < u_{i+1}$ となるようなノット区間 i を見つけることができれば，$p+1$ 個の B スプライン基底関数 $N_{i-p,p}, \ldots, N_{i,p}$ だけを計算すれば良い [7]．これら以外の関数値はすべて 0 となる．よって，式 (2) を計算する手順は以下のようになる．

1. ノットベクトル U のうち，$u_i \le u < u_{i+1}$ となる i を見つける．
2. B スプライン基底関数 $N_{i-p,p}, \ldots, N_{i,p}$ を計算する．

ここで，もう一つの問題として，GPU の中で次数の違いをどのように扱うか，ということがある．前述のとおり，GPU 上では可変変数を使ってループ演算を扱うことができない．従って，次数 p も，フラグメントプログラムの計算ルーチンの中では固定されている必要がある．

ここでの我々の解決案として，次数ごとに別々の計算ルーチンを用意する．すなわち，プログラムの中で，次数により場合分けすることで別々の関数を呼び出すことにする．

ノット区間の決定． ノット区間 i の決定は，あらかじめソートされている 1 次元配列より一つの要素を抽出する操作に相当する．これは線形探索や二分探索により計算できる．我々はこの計算に [8] の二分探索アルゴリズムを簡略化したものを用いている．すなわち，ある入力パラメータ u に対し，ノットベクトルの 1 次元配列 U を二分探索で辿っていき，繰り返し計算の中でその範囲を絞っていく．繰り返し計算の回数は扱えるノットベクトルの数に依存する．例えば 5 回ならば $2^5 = 32$ 個, 6 回ならば

$2^6 = 64$ 個のノットベクトルを扱うことができる．アルゴリズムの計算量は $O(\log n)$ となる．このアルゴリズムの中では $u = u_m$ のときに範囲が定まらないが，別途同一条件を設ける．すなわち，$u = u_m$ のとき，ノット区間を $m - p - 1$ として値を返す．

ノットベクトル U は，1 次元テクスチャとして格納し，必要に応じてフラグメントプログラム内で取得する．テクスチャへの格納方法に関しては 3.2 節で記述する．

B スプライン基底関数の計算.　u が i のノット区間にあるものとすると，B スプライン基底関数は逆三角形法 [7] により求めることができる:

$$\begin{array}{cccc} & & & N_{i-3,3} \\ & & N_{i-2,2} & N_{i-2,3} \\ & N_{i-1,1} & N_{i-1,2} & N_{i-1,3} \\ N_{i,0} & N_{i,1} & N_{i,2} & N_{i,3} \end{array} \quad \cdots \tag{3}$$

次数 0 の基底関数からはじめ，次数を上げていきながら，式 (2) を用いて計算する．次数 p に対し，計算すべき基底関数の数は $\sum_{i=0}^{p} i$ 個である．次数 p の基底関数の計算の中で，計算に必要なノットは $u_{i-p+1}, u_{i-p+2}, \ldots, u_{i+p}$ であり，全部で $2p$ 個である．

B スプライン曲線の計算.　基底関数の計算と同様，ノット区間 i が決定しているものとすると，計算に必要な制御点は $\mathbf{P}_{i-p}, \mathbf{P}_{i-p+1}, \ldots, \mathbf{P}_i$ の計 $p+1$ 個である．$n+1$ 個の制御点はあらかじめテクスチャに格納しておき，フラグメントプログラムの中で，i の値に応じて取得する．B スプライン曲線 $\mathbf{C}_p(u)$ は

$$\begin{aligned} \mathbf{C}_p(u) = &N_{i-p,p}\mathbf{P}_{i-p} + N_{i-p+1,p}\mathbf{P}_{i-p+1} \\ &+ \ldots + N_{i,p}\mathbf{P}_i, \end{aligned} \tag{4}$$

のように N と $\mathbf{P}$ の線形結合により求められる．これらの計算式は，次数ごとに別々の関数として用意する．

導関数の計算.　1 階導関数 $\mathbf{C}_p{}'(u)$ は次式で計算される:

$$\begin{aligned} \mathbf{C}'_p(u) = &N'_{i-p,p}\mathbf{P}_{i-p} + N'_{i-p+1,p}\mathbf{P}_{i-p+1} \\ &+ \ldots + N'_{i,p}\mathbf{P}_i. \end{aligned} \tag{5}$$

$N'_{i,p}(u)$ は B スプライン基底関数の 1 階導関数であり，次式で与えられる（導出過程は [7] を参照のこと）:

$$\begin{aligned} N'_{i,p}(u) = &\frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) \\ &- \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u). \end{aligned} \tag{6}$$

式 (6) より，導関数の計算に必要な要素のほとんどを式 (2) の基底関数の計算と共有化できる: $N_{i,p-1}(u)$,$N_{i+1,p-1}(u)$ の計算までが式 (2) と同一であり，ノットテクスチャ，制御点テクスチャは同じものを使用できる．このことは，プログラムにおける命令数の大幅な減少をもたらす．

B スプライン曲面の計算.　p, q 次の非一様 B スプライン曲面は，u, v 各方向で決定されたノット区間をそれぞれ i, j とすると，以下の式で与えられる:

$$\mathbf{S}_{p,q}(u,v) = \sum_{k=i-p}^{i} \sum_{l=j-q}^{j} N_{k,p}(u) N_{l,q}(v) \mathbf{P}_{k,l}. \tag{7}$$

B スプライン曲面の計算手法は，曲線の場合とそれほど違いはない．すなわち，u, v 各方向ごとに B スプライン基底関数を求め，$(p+1) \times (q+1)$ 個の制御点との線形結合により曲面上の点を計算する．曲面の法線ベクトルは，u, v における各導関数を求め，外積をとる．

B スプライン曲面の他の計算方法としては，de Boor アルゴリズムがある．この方法を用いると，曲面上の点の位置を直接計算することができる．位置だけを計算するならば，上記の手法の代わりに de Boor アルゴリズムを用いても計算量はほとんど変化がない．但し，我々の方法を用いることで，B スプライン基底関数の値を，位置と導関数（法線）の計算で使い回しできる．

3.2　テクスチャの準備.

本節では，GPU 上で B スプライン曲面の評価を行うために必要な，テクスチャの生成方法について説明する．テクスチャとして必要なのは，ノットベクトルおよび制御点である．

図 1 に，ノットベクトルのテクスチャの格納方法について示す．このテクスチャは，B スプライン曲面の計算において，上記で示したほとんどの関数の中で参照される．各方向につき一つのテクスチャが必要であり，全部で二つ（u, v 方向）生成される．横

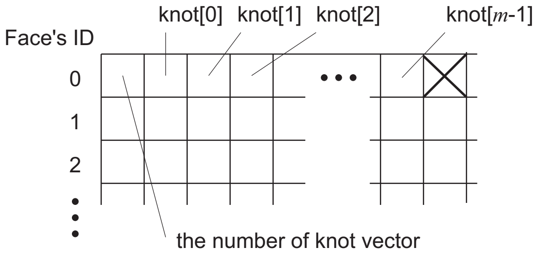

図 1　二次元テクスチャによるノットベクトルの格納方法.

軸にはノットベクトルの列を，縦軸には面の id が示されている．各ピクセルは一つの浮動小数点のみを格納できる．それぞれの横の列の先頭のピクセルには，ノットベクトルの数を格納しておく．これは，ノット区間を決定するための二分探索アルゴリズムの中で必要となる．テクスチャのサイズは（ノットベクトルの数の最大値 + 1）×（面の数）となる．

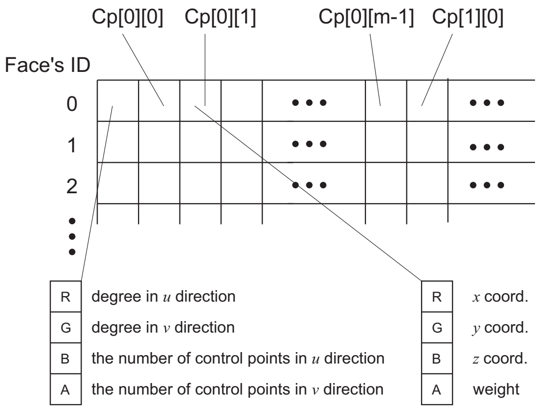

図 2　二次元テクスチャによる制御点の格納方法.

図 2 に，制御点のテクスチャの格納方法について示す．このテクスチャは，B スプライン曲面の位置および導関数の計算の際にのみ参照され，一つだけ作成される．横軸方向には $m \times n$ 個の制御点を，縦軸方向には面の id が示されている．各ピクセルは 4 つの浮動小数点を保持することができ，そのほとんどのピクセルには，一つの制御点の x, y, z 座標および重みを格納する．重みは NURBS でのみ必要となる．それぞれの行の先頭のピクセルには，u, v 方向の次数およびそれぞれの方向の制御点の数が格納される．次数は，様々な関数において利用される他，次数による場合分けにも利用される．制御点の数は，位置や導関数の計算時にのみ用いられる．テクスチャのサイズは，（制御点の数の最大値 + 1）×（面の数）となる．

3.3　B スプライン曲面の描画アルゴリズム

本節では，B スプライン曲面の描画アルゴリズムについて説明する．

まず曲面の要素のテクスチャとは別に，曲面を三角形分割したポリゴンを用意する．このポリゴンは，アルゴリズムの際のフラグメント取得のために使われる．ポリゴンの各頂点には，頂点座標の他に曲面の二次元パラメータ (u, v) と面番号を格納しておく．トリム曲面に関しても，この（トリム化された）ポリゴンさえ用意できれば，トリムなし曲面と同様に扱うことができる．アルゴリズムにトリム曲面のための特別な関数を必要としない．

図 3 に，提案するレンダリングアルゴリズムの概要を示す．用意されたポリゴンは，通常のレンダリングプロセスの中で描画される．このとき，曲面のパラメータ (u, v) はテクスチャ座標のそれぞれ x, y 座標として，面番号はテクスチャパラメータの z 座標として入力すると効率的である．ただし，テクスチャパラメータは，グラフィックス API（e.g. OpenGL）の仕様により，最大 3.0 までの値しか入力できない．そのため，ある大きな数字（例えば 10,000）で割ってから入力し，後のフラグメントプログラムの中でその数値を掛けることで元に戻す．ポリゴンの各三角形はラスタ化によりフラグメントに分解され，線形補間されたパラメータ (u, v) と面番号はフラグメントプログラムに渡される．

フラグメントプログラムの中で，曲面パラメータと面番号，および 3 つのテクスチャ（2 つのノットテクスチャと制御点テクスチャ）を入力として，曲面の評価を行う．まず，曲面パラメータとノットテクスチャより，ノット区間の決定が行われる．次に，面番号と制御点テクスチャより次数を取得する．次数によって場合分けが行われ，別々の関数を利用して計算が行われる．

なお，本手法では次数毎に別々の関数を用意する必要があるが，用意すべき関数の数を減らすには，

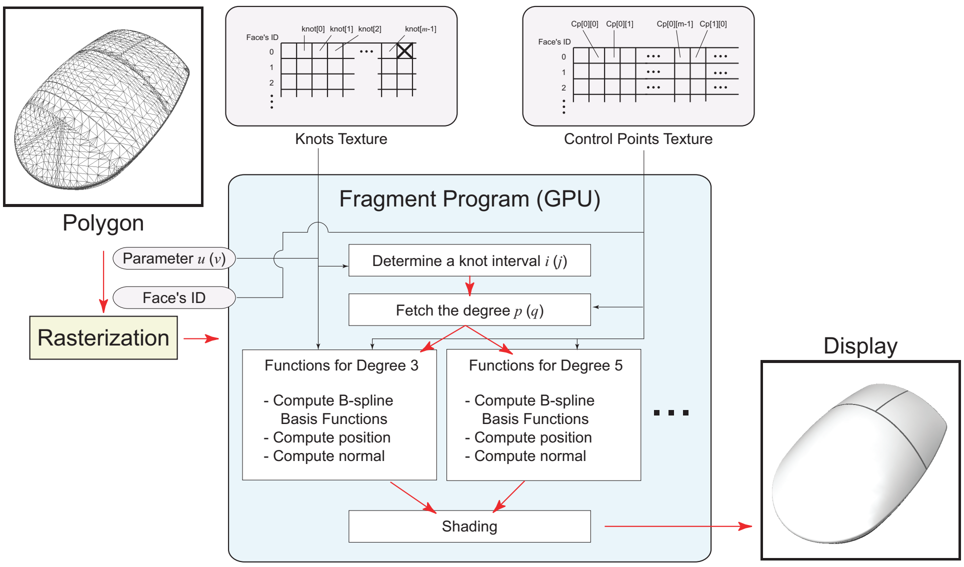

図 3　本手法における描画手順.

次のような戦略をとるのが良いと考えられる．すなわち，次数に応じてすべての関数を用意するのではなく，次数の異なる（例えば次数 3, 5, 7 ... など）いくつかの関数を作成する．用意した次数以外のものに関しては，それよりも高い次数で最も近いものに次数上げを行う．例えば次数 4 は 5 に次数上げし，次数 6 は 7 に次数上げする，など．また，u, v 方向でそれぞれ次数が異なる場合は，高い方の次数に揃える．例えば u 方向で 3 次，v 方向で 5 次の場合，両方とも 5 次にする，など．このようにする理由は，次数が高くなればなるほど計算量が増えるため計算量を抑えることと，関数の数を減らすことの両方の条件を同時に満たすためである．

各次数毎での曲面の位置と法線ベクトルの計算後，共通のシェーディング処理ルーチンに渡し，そのフラグメントにおける最終的な色を計算する．

4　結果と議論

以下に，本手法による描画結果について議論する．なお本論文では，双 3 次非一様 B スプライン曲面のみを実装している．

図 4 に isophoto [6] を利用した結果の比較を示す．isophoto は，法線と光線（光源から曲面上の点までの線）のなす角度で色分けした表示図である．法線の連続性を確かめることができるため，CAD 等のアプリケーションで良く用いられる．左図は，ポリゴンの頂点に法線ベクトルを割り当て，これを使って isophoto を表示した結果である．このとき，各フラグメントには，面の頂点の法線ベクトルを線形補間したものが割り当てられる．右図は，本手法による isophoto の表示結果である．左図の方は，線形補間による不連続な凹凸が現れており，線形補間により計算された法線ベクトルを利用しているからに他ならない．これに対し，右図の方は線が滑らかに表示されていることが確認できる．これは，フラグメント単位で計算された正確な法線ベクトルを利用していることによる．

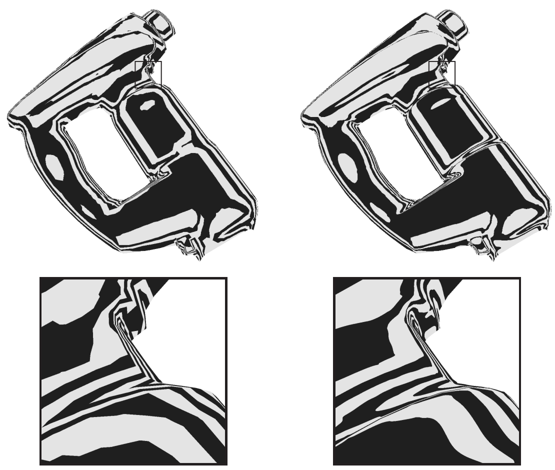

図 4　左: ポリゴンを利用した isophoto の表示結果．右: 本手法による isophoto の表示結果.

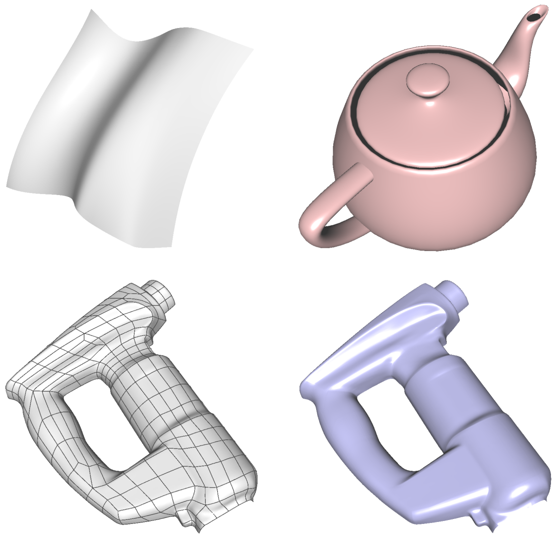

図 5　本手法による様々なモデルの表示結果．左上: B スプライン曲面パッチ．右上: B スプライン曲面で表現されたティーポットモデル．下: スプレーモデル．左図にはパッチ境界を表示している．

	#patches	#control points	#knot vectors	fps
surface	1	6×14	8×16	212
teapot	32	4×4	6×6	201
spray	266	18×18	20×20	126

表 1　本論文で使用したモデルに対するパッチ数，1 つのパッチにおける最大制御点数，最大ノットベクトル数，計算時間．"teapot" はベジエ曲面を一様 B スプライン曲面に変換している．512×512 の解像度の画面の表示により計測．

図 5 に，いくつかのモデルに対し本手法を適用した結果を示す．また，表 1 には，本論文で使用したデータの統計と計算時間を示す．計算時間は，Athlon 64 X2 4800+ CPU, GeForce 7900 GTX 512MB GPU の環境で計測している．なお，fps で示される描画速度は，モデルを回転させたときの平均速度を計測している．

いずれのモデルに対しても，実用的な描画速度を得られているのがわかる．ただし，描画速度はパッチの数に対して線形に比例しているわけではなく，むしろ，塗りつぶされるピクセルの数に対し大きな影響を受けている．これは，フラグメントプログラムでの処理に時間を取られていることが一因である．

5　結論と展望

本研究では，非一様 B スプライン曲面のフラグメント単位での評価手法と，それを用いたレンダリングを GPU 上で行うための手法を提案した．本手法により，粗いポリゴンに対してもフラグメント単位で高品質なレンダリングを行えることを示した．これは特に，視覚的曲面評価ツール等の応用に対する利点を持つことが期待できるものである．

参考文献

[1] S. Bischoff, L. P. Kobbelt, and H.-P. Seidel. Towards hardware implementation of loop subdivision. In *Proc. ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp. 41–50, 2000.

[2] J. Bolz and P. Schröder. Evaluation of subdivision surfaces on programmable graphics hardware. Technical report, California Institute of Technology, 2003.

[3] T. Boubekeur and C. Schlick. Generic mesh refinement on GPU. In *Proc. ACM SIGGRAPH/Eurographics Conference on Graphics Hardware*, pp. 99–104, 2005.

[4] M. Guthe, Ákos Balázs, and R. Klein. GPU-based trimming and tessellation of NURBS and T-spline surfaces. *ACM Transactions on Graphics (Proc. SIGGRAPH 2005)*, 24(3):1016–1023, 2005.

[5] H. Hagen, S. Hahmann, T. Schreiber, Y. Nakajima, B. Wordenweber, and P. Hollemann-Grundstedt. Surface interrogation algorithms. *IEEE Computer Graphics and Applications*, 12(5):53–60, 1992.

[6] S. Hahmann. Visualization techniques for surface analysis. In C. Bajaj ed., *Advanced Visualization Techniques*. John Wiley, 1999.

[7] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, Berlin, 2nd edition, 1997.

[8] T. J. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan. Photon mapping on programmable graphics hardware. In *Proc. ACM SIGGRAPH/Eurographics Conference on Graphics Hardware*, pp. 41–50, 2003.

[9] L.-J. Shiue, I. Jones, and J. Peters. A realtime GPU subdivision kernel. *ACM Transactions on Graphics (Proc. SIGGRAPH 2005)*, 24(3):1010–1015, 2005.

[10] Y. Yasui and T. Kanai. Surface quality assessment of subdivision surfaces on programmable graphics hardware. In *Proc. International Conference on Shape Modeling and Applications*, pp. 129–136, 2004.