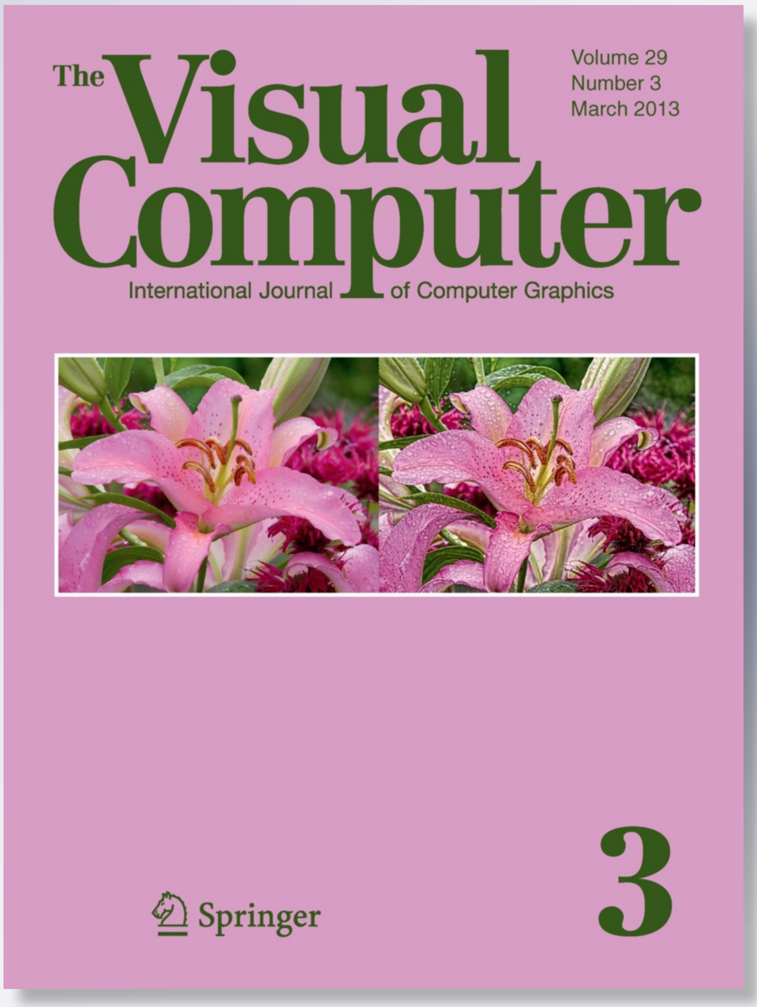

The Visual Computer
International Journal of Computer Graphics
Volume 29
Number 3
March 2013
Springer
3

Vis Comput (2013) 29:189–201
DOI 10.1007/s00371-012-0715-y

ORIGINAL ARTICLE

# Shape aware normal interpolation for curved surface shading from polyhedral approximation

**Xunnian Yang · Jianmin Zheng**

Published online: 28 April 2012

**Abstract** Independent interpolation of local surface patches and local normal patches is an efficient way for fast rendering of smooth curved surfaces from rough polyhedral meshes. However, the independently interpolating normals may deviate greatly from the analytical normals of local interpolating surfaces, and the normal deviation may cause severe rendering defects when the surface is shaded using the interpolating normals. In this paper we propose two novel normal interpolation schemes along with interpolation of cubic Bézier triangles for rendering curved surfaces from rough triangular meshes. Firstly, the interpolating normal is computed by a Gregory normal patch to each Bézier triangle by a new definition of quadratic normal functions along cubic space curves. Secondly, the interpolating normal is obtained by blending side-vertex normal functions along side-vertex parametric curves of the interpolating Bézier surface. The normal patches by these two methods can not only interpolate given normals at vertices or boundaries of a triangle but also match the shape of the local interpolating surface very well. As a result, more realistic shading results are obtained by either of the two new normal interpolation schemes than by the traditional quadratic normal interpolation method for rendering rough triangular meshes.

This work is supported by NNSF of China grant (60970077) and the ARC 9/09 Grant (MOE2008-T2-1-075) of Singapore.

X. Yang (✉)
Department of Mathematics, Zhejiang University, Hangzhou, China
e-mail: yxn@zju.edu.cn

J. Zheng
School of Computer Engineering, Nanyang Technological University, Singapore, Singapore
e-mail: ASJMZheng@ntu.edu.sg

**Keywords** Surface rendering · Triangular meshes · Gregory normal interpolation · Side-vertex normal interpolation

## 1 Introduction

In computer graphics rough triangular meshes have been widely used for the approximate representation of curved surfaces. Even though meshes can be used to approximate some smooth surfaces efficiently, flat shaded meshes have severe visual defects due to rough contours and discontinuous normal vectors for the triangles. To obtain realistic rendering results of curved surfaces represented by meshes, the curved geometry should be reconstructed, and the normal vectors of the surface should be interpolated properly.

Interpolation of a triangular mesh by a set of smoothly joined surface patches has been studied extensively in past few decades. Many literatures deal with parametric surface interpolation of triangle meshes with tangent plane or even higher order of continuity; see, for example [1–6] and references therein. Smooth surface interpolation plays an important role for geometric modeling. However, the construction of smooth surfaces from initial triangular meshes with arbitrary topology is usually very costly. It needs at least quartic surface patches to fill a triangular mesh with $G^1$ continuity. When lower order surface patches are used, triangles should then be subdivided into smaller ones. The determination of too many free parameters for high-order surface patches or subdivision of triangular meshes may arouse additional complexity or inefficiency for surface rendering.

A feasible way to render meshes with high speed as well as visually smooth quality is defining the geometry and surface normals separately [7, 8]. The sharp contour of a triangular mesh can be made smooth by local interpolation of low-order surface patches [8, 9] or tessellating triangles adaptively with local interpolating curves or surfaces [10, 11]. These local operations have the advantages of simplicity and low computational costs for contour smoothing, but the tangent planes for adjacent quadric or cubic surface patches may not be tangent continuous across the joining edges. Even though, one can still obtain visually smooth rendering results using continuous normals which are interpolated independently from the surface patches. For fast and realistic rendering purpose, the interpolating normals should satisfy the following requirements:

1. The interpolating normals should be continuous all over the surface, and they can hide the discontinuity of analytical normals of local interpolating surfaces.
2. The interpolating normals should coincide with the contour of the input mesh well, or they should approximate the analytical normals of local interpolating surfaces closely.
3. The interpolating normals should be constructed locally and simply. This rule can guarantee the efficiency and hardware implementation of normal interpolation algorithms.

### 1.1 Our approach

In this paper we present two novel methods for normal interpolation for curved surface shading along with local surface interpolation to triangular meshes. For a triangular mesh with given or estimated normal vectors at vertices, we interpolate every triangle by a cubic Bézier triangle which matches normal vectors at vertices too. As suggested in [8], cubic Bézier surfaces can be used to smooth mesh contours efficiently even though adjacent surface patches have only $C^0$ continuity in general.

Along with local interpolation of cubic Bézier triangles, independent normal vectors will be evaluated. The proposed normal interpolation methods are based on normal computation along cubic Bézier curves in space. A quadratic normal function along a cubic Bézier curve in space is obtained by transforming the control vectors of the tangent curve. When the normal curves along all boundary edges are obtained, the normal vector to any point in the interior of a triangle is obtained by a quadratic Gregory normal patch. This type of interpolating normals can also be blended with the analytical normals of the cubic Bézier patch. Another way to interpolate normal vector to any interior point in the triangle is by blending the side-vertex normal functions that are computed by three side-vertex parametric curves of the Bézier patch. Just like surface interpolation by the side-vertex method [2], the smoothness of normal patch by the side-vertex interpolation scheme will be improved further when the end derivatives of side-vertex normal functions have been set properly.

### 1.2 Overview

In Sect. 2 we review previous methods on normal interpolation. The construction of cubic Bézier patches that interpolate vertices and normals of triangles will be given in Sect. 3. We present formulae for Gregory normal interpolation and side-vertex normal interpolation in Sect. 4. The complexities of our proposed normal interpolation schemes are analyzed in Sect. 5. Examples and comparisons with some known methods are presented in Sect. 6. Section 7 concludes the paper.

## 2 Previous work

When rendering a polyhedral surface, Phong [12] first proposed to calculate intensity for each point/pixel by an illumination model using linearly interpolated normal vectors. The interpolating normals can be used to generate more realistic shading results than direct interpolation of intensities; see [13] for a comparison. However, surface shading by linear interpolation of normal vectors still suffers defects like mach bands. Instead of linearly interpolation of normal vectors, van Overveld and Wyvill [7] proposed to interpolate a quadratic normal function along a straight edge when the unit normal vectors at two ends are given. The coefficients of the normal function are obtained by the interpolation condition together with the minimization of a fairness functional. The quadratic normals can generate more realistic shading results than normals by linear interpolation. As pointed out in [14], this method may suffer unnecessary highlights for convex edges when the normal vectors at two ends are not symmetric.

Based on quadratic normal interpolation at boundary edges, Vlachos et al. [8] defined a quadratic normal function for the normal component of a so-called PN triangle. The independent normal components of PN triangles can generate visually smooth shading results for adjacent Bézier triangles which are joined with only position continuity. However, the quadratic normals may not coincide with the shape of local interpolating surfaces very well when the input normals at triangle vertices deviate from the facet normals considerably. The idea of PN triangles has also been extended to higher-order rational interpolating surfaces [15] or scalar tagged normal vectors [16]. However, these extensions still interpolate normal vectors based on selected boundary normals, and the same defect remains unresolved.

Recently, there are increasing interests in rendering subdivision surfaces with substitutes of local surface patches [17]. Boubekeur et al. [18] interpolated both sampled points and normals by quadratic patches. The geometry of a Catmull–Clark surface can be approximated by either bicubic surface patches or Gregory surface patches, and then the normal vectors are defined by independent normal patches [19, 20]. Li et al. [21] proposed to approximate Loop subdivision surfaces by quartic triangular Bézier patches and render the surfaces using independent quartic normal patches too. These interpolating surface patches and normal patches match the geometry and normals of the original surface well, and in case the subdivision surface happens to a B-spline surface they can reduce to the geometry or normal vectors of the B-spline surface. However, the derivation of approximating surfaces and interpolating normals depends on the exact surface points and their derivatives. These methods cannot be extended to render a general polyhedral surface.

Besides contour smoothing and normal interpolation using local surface patches, Alexa and Boubekeur [22] proposed subdivision shading technique to calculate independent normal vectors for subdivision surface rendering. This method produces more visually smooth results for rendering (especially interpolatory) subdivision surfaces than using the analytical normals of the subdivision surfaces themselves. The requirements of topology information for normal subdivision may prevent it from being an efficient method for fast rendering as compared with local interpolating surface patches.

## 3 Curved triangle construction

In this section we briefly introduce local surface interpolation by cubic Bézier triangles. Surface interpolation is an essential step for contour smoothing for realistic surface shading, and it serves as a basis for our proposed normal interpolation schemes too.

For a triangular mesh, we assume that the unit normal vectors at mesh vertices are already given. If only triangular meshes are known, vertex normals should be estimated by averaging facet normals, for example, by the method proposed in [23]. For a triangular mesh equipped with unit normal vectors at vertices, there are many different ways to interpolate the triangles by piecewise smooth surfaces. Among all these surfaces, the cubic Bézier triangle is a simple but efficient surface form to interpolate the vertices and the vertex normals of any triangle. Cubic Bézier triangles are low-order surface patches that can represent convex surfaces as well as surfaces with inflections. Together with a continuous normal field, cubic Bézier triangles can be used as efficient tools for surface contour smoothing [8].

Assume that $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are the three vertices of a triangle, and $\mathbf{n}_1$, $\mathbf{n}_2$, and $\mathbf{n}_3$ are the unit normal vectors corresponding to these three vertices, respectively. Any point $\mathbf{p}$ within the triangle can then be represented as a linear combination of vertices like $\mathbf{p} = w\mathbf{p}_1 + u\mathbf{p}_2 + v\mathbf{p}_3$, where $u \geq 0$, $v \geq 0$ and $w = 1 - u - v \geq 0$ are the barycentric coordinates

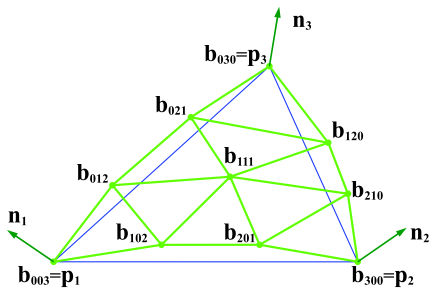


**Fig. 1** Control points for a cubic triangular Bézier patch

of the point. A cubic triangular Bézier patch defined on the domain of the triangle has the form

$$\mathbf{b}(u, v, w) = \sum_{i+j+k=3} \mathbf{b}_{ijk} \frac{3!}{i!j!k!} u^i v^j w^k, \qquad (1)$$

where $\mathbf{b}_{ijk}$ are the control points of the surface patch. The mesh formed by these control points is the control mesh of the surface (Fig. 1). For notational simplicity, a Bézier triangle $\mathbf{b}(u, v, 1-u-v)$ is also written as $\mathbf{b}(u, v)$. For a Bézier triangle, the analytical normal vector of the surface can be evaluated as follows:

$$\mathbf{n}_{bz} = \frac{\mathbf{b}_u(u, v) \times \mathbf{b}_v(u, v)}{\|\mathbf{b}_u(u, v) \times \mathbf{b}_v(u, v)\|}.$$

According to the properties of Bézier triangles [24], the three corner control points are interpolated by the Bézier surface, and the tangent planes through three corners are just determined by control points neighboring to respective corners. Then, the boundary control points for a cubic Bézier triangle which interpolates vertices and normals of a given flat triangle can be obtained from the interpolation conditions directly. As suggested in [8], the boundary control points of an interpolating Bézier triangle can be obtained by projecting trisection points of triangle edges onto the tangent planes through three corners. It yields

$$\begin{aligned}
&\mathbf{b}_{003} = \mathbf{p}_1, \quad \mathbf{b}_{300} = \mathbf{p}_2, \quad \mathbf{b}_{030} = \mathbf{p}_3,\\
&\mathbf{b}_{102} = (2\mathbf{p}_1 + \mathbf{p}_2 - \omega_{12}\mathbf{n}_1)/3,\\
&\mathbf{b}_{201} = (2\mathbf{p}_2 + \mathbf{p}_1 - \omega_{21}\mathbf{n}_2)/3,\\
&\mathbf{b}_{210} = (2\mathbf{p}_2 + \mathbf{p}_3 - \omega_{23}\mathbf{n}_2)/3,\\
&\mathbf{b}_{120} = (2\mathbf{p}_3 + \mathbf{p}_2 - \omega_{32}\mathbf{n}_3)/3,\\
&\mathbf{b}_{012} = (2\mathbf{p}_1 + \mathbf{p}_3 - \omega_{13}\mathbf{n}_1)/3,\\
&\mathbf{b}_{021} = (2\mathbf{p}_3 + \mathbf{p}_1 - \omega_{31}\mathbf{n}_3)/3,
\end{aligned}$$

where $\omega_{ij} = (\mathbf{p}_j - \mathbf{p}_i) \cdot \mathbf{n}_i$ for $i, j \in \{1, 2, 3\}$ and $i \neq j$. When all boundary control points are obtained, the center control point $\mathbf{b}_{111}$ can be computed by

$$\begin{aligned}
\mathbf{b}_{111} = {} & (\mathbf{b}_{102} + \mathbf{b}_{201} + \mathbf{b}_{210} + \mathbf{b}_{120} + \mathbf{b}_{012} + \mathbf{b}_{021})/4\\
& - (\mathbf{p}_1 + \mathbf{p}_2 + \mathbf{p}_3)/6.
\end{aligned}$$

Due to the fact that the boundary control points of an interpolating Bézier triangle are determined only by end vertices and vertex normals of triangle edges, any two Bézier triangles that interpolate two neighboring triangles will be joined seamlessly, and all Bézier triangles interpolating the same corner have a common tangent plane at the point. However, the normal vectors of two adjacent Bézier triangles are generally not continuous across the joining edges.

To obtain a visually smooth rendering result, Vlochas et al. [8] also defined an independent normal patch for the surface as

$$\mathbf{n}(u,v)=\mathbf{n}_1 w^2+\mathbf{n}_2 u^2+\mathbf{n}_3 v^2+\mathbf{h}_1 uv+\mathbf{h}_2 vw+\mathbf{h}_3 wu, \tag{2}$$

where $\mathbf{h}_1$, $\mathbf{h}_2$, and $\mathbf{h}_3$ are the reflected average end normals across planes perpendicular to three triangle edges, respectively. The quadratic normals of Eq. (2) are usually smooth over a whole surface even though they are constructed independently for each triangle, but they depend entirely on triangle boundaries, and they may not coincide with the geometry shape of local interpolating surfaces very well. There may exist some rendering defects by this kind of normal interpolation.

## 4 Shape aware normal interpolation

Along with local surface interpolation by cubic triangular Bézier patches, in this section we present two novel normal interpolation schemes for rendering curved surfaces from polyhedral approximations. Based on a new normal interpolation method along cubic curves in space, a Gregory normal patch and a side-vertex normal interpolation technique are presented or developed in the following.

### 4.1 Normal vectors along a cubic curve

As a basis for shape aware normal interpolation, we first propose a new quadratic normal function along a cubic curve in space. Without loss of generality, we assume that a cubic curve is represented by

$$\mathbf{r}(t)=\mathbf{r}_0 B_{0,3}(t)+\mathbf{r}_1 B_{1,3}(t)+\mathbf{r}_2 B_{2,3}(t)+\mathbf{r}_3 B_{3,3}(t),$$

where $B_{i,3}(t)=\frac{3!}{i!(3-i)!}t^i(1-t)^{3-i}$ are the Bernstein basis functions, $\mathbf{r}_0$, $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ are the control points of the curve. Given two unit normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ at two ends of the curve, we would compute interpolating normals along curve $\mathbf{r}(t)$ by rotating tangent vectors along the curve.

The derivative or the tangent direction of curve $\mathbf{r}(t)$ with respect to parameter $t$ can be obtained by $\frac{1}{3}\mathbf{r}'(t)=(\mathbf{r}_1-\mathbf{r}_0)B_{0,2}(t)+(\mathbf{r}_2-\mathbf{r}_1)B_{1,2}(t)+(\mathbf{r}_3-\mathbf{r}_2)B_{2,2}(t)$, so we define the normal function along curve $\mathbf{r}(t)$ as

$$\begin{aligned}\mathbf{n}(t)&=(\mathbf{r}_1-\mathbf{r}_0)^{\perp}B_{0,2}(t)\\&\quad+(\mathbf{r}_2-\mathbf{r}_1)^{\perp}B_{1,2}(t)+(\mathbf{r}_3-\mathbf{r}_2)^{\perp}B_{2,2}(t),\end{aligned} \tag{3}$$

where $(\mathbf{v})^{\perp}$ means a vector that has equal magnitude but is perpendicular to vector $\mathbf{v}$. To ensure that the normal function $\mathbf{n}(t)$ interpolates $\mathbf{n}_1$ and $\mathbf{n}_2$ at two ends, the control vectors for $\mathbf{n}(t)$ are computed as follows:

$$\begin{aligned}(\mathbf{r}_1-\mathbf{r}_0)^{\perp}&=\|\mathbf{r}_1-\mathbf{r}_0\|\mathbf{n}_1,\\(\mathbf{r}_2-\mathbf{r}_1)^{\perp}&=\mathbf{n}_v\times(\mathbf{r}_2-\mathbf{r}_1),\\(\mathbf{r}_3-\mathbf{r}_2)^{\perp}&=\|\mathbf{r}_3-\mathbf{r}_2\|\mathbf{n}_2,\end{aligned}$$

where

$$\mathbf{n}_v=\frac{(\mathbf{r}_3-\mathbf{r}_0)\times(\mathbf{n}_1+\mathbf{n}_2)}{\|(\mathbf{r}_3-\mathbf{r}_0)\times(\mathbf{n}_1+\mathbf{n}_2)\|}.$$

From the construction process we know that vector $(\mathbf{r}_2-\mathbf{r}_1)^{\perp}$ lies on a plane spanned by vectors $\mathbf{r}_3-\mathbf{r}_0$ and $\mathbf{n}_1+\mathbf{n}_2$. In particular, $\mathbf{n}(t)$ is just the normal function of curve $\mathbf{r}(t)$ when the curve and two given normal vectors all lie on the same plane and the normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ also satisfy $(\mathbf{r}_1-\mathbf{r}_0)\cdot\mathbf{n}_1=0$ and $(\mathbf{r}_3-\mathbf{r}_2)\cdot\mathbf{n}_2=0$.

Usually, cubic-curve-based normal vectors match the contour shapes of triangle meshes much better than the traditional quadratic normal interpolation proposed in [8]. See Fig. 2 for a comparison of normal interpolation along a cubic Bézier curve that interpolates end vertices and end normal

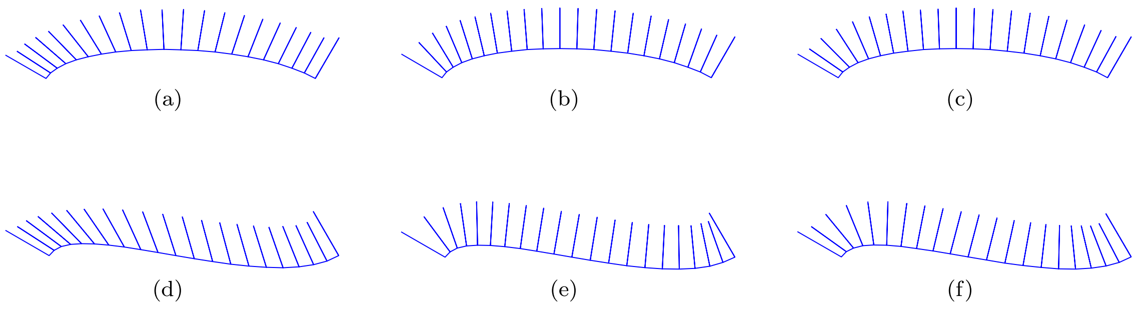


**Fig. 2** Normal interpolation along with cubic curve interpolation. (**a**) and (**d**) Normals by quadratic normal interpolation; (**b**) and (**e**) Normal interpolation by the cubic curve; (**c**) and (**f**) Interpolating normals with vanishing end derivatives

vectors of an edge. Though the interpolating quadratic normals change very smoothly along the curve, but they deviate from the curve contour apparently; see Figs. 2(a) and (d). The interpolating normals by Eq. (3) are the exact curve normals when the curve lies entirely on a plane; see Figs. 2(b) and (e).

Even though the interpolating normals by Eq. (3) coincide with the shape of local cubic interpolating curve very well, the $C^0$ continuity of normal vectors at the joint point between two edges or across the joint edge between two triangles may still be visible when the normals are computed independently for each edge or each triangle. To overcome this phenomenon, the interpolating normals should be joined even more smoothly at the joint points or joint edges. To achieve such a goal, we propose to compute normal functions that have vanishing derivatives at the ends. Then any two neighboring normal functions have the same derivative at the joint point.

Assuming that a quadratic normal function by Eq. (3) has been represented by

$$\mathbf{n}(t) = \mathbf{d}_0 B_{0,2}(t) + \mathbf{d}_1 B_{1,2}(t) + \mathbf{d}_2 B_{2,2}(t),$$

we elevate the degree and update $\mathbf{n}(t)$ a little so that it has vanishing derivative at ends. By applying the degree elevation algorithm three times [25] and modifying the control points, we have

$$\begin{aligned}\hat{\mathbf{n}}(t) = {} & \mathbf{d}_0\big[B_{0,5}(t) + B_{1,5}(t)\big] \\ & + \mathbf{d}_1\big[B_{2,5}(t) + B_{3,5}(t)\big] \\ & + \frac{\mathbf{d}_0 - \mathbf{d}_2}{10}\big[B_{3,5}(t) - B_{2,5}(t)\big] \\ & + \mathbf{d}_2\big[B_{4,5}(t) + B_{5,5}(t)\big]. \qquad (4)\end{aligned}$$

From Eq. (4) we have $\hat{\mathbf{n}}'(0) = \hat{\mathbf{n}}'(1) = 0$ immediately.

The difference between $\mathbf{n}(t)$ and $\hat{\mathbf{n}}(t)$ can be computed as

$$\begin{aligned}\mathbf{n}(t) - \hat{\mathbf{n}}(t) = {} & \Delta_1\big(B_{1,5}(t) - B_{2,5}(t)\big) \\ & + \Delta_2\big(B_{4,5}(t) - B_{3,5}(t)\big),\end{aligned}$$

where $\Delta_1 = \frac{2}{5}(\mathbf{d}_1 - \mathbf{d}_0)$ and $\Delta_2 = \frac{2}{5}(\mathbf{d}_1 - \mathbf{d}_2)$. Then we have

$$\begin{aligned}\big\|\mathbf{n}(t) - \hat{\mathbf{n}}(t)\big\| & \le \max\big\{\|\Delta_1\|, \|\Delta_2\|\big\} \sup_{t\in[0,1]} \big(\big|B_{1,5}(t) - B_{2,5}(t)\big| + \big|B_{4,5}(t) - B_{3,5}(t)\big|\big) \\ & \le 0.3125 \max\big\{\|\Delta_1\|, \|\Delta_2\|\big\} \\ & = 0.125 \max\big\{\|\mathbf{d}_1 - \mathbf{d}_0\|, \|\mathbf{d}_1 - \mathbf{d}_2\|\big\}.\end{aligned}$$

This ensures that the difference between the modified normal with the original one is small and the modified normal approximates the original one well. See Figs. 2(c) and (f) for examples of modified normal functions from previously obtained normal functions by cubic curve interpolation.

## 4.2 Gregory normal interpolation

For a triangle $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ equipped with unit normal vectors $\mathbf{n}_1$, $\mathbf{n}_2$, and $\mathbf{n}_3$ at three vertices, a cubic triangular Bézier patch $\mathbf{b}(u, v)$ can first be constructed to interpolate the triangle for contour smoothing. Simultaneously, the normal vector to any point on the Bézier patch can be computed by *Gregory normal patch* which is constructed using normal vectors along boundary curves.

It can be easily shown that boundary curves $\mathbf{b}(1-t, t, 0)$, $\mathbf{b}(0, 1-t, t)$, and $\mathbf{b}(t, 0, 1-t)$ are all cubic Bézier curves. By Eq. (3), we have normal vectors to these three curves as

$$\mathbf{e}_1(t) = \mathbf{n}_{21} B_{0,2}(t) + \mathbf{h}_1 B_{1,2}(t) + \mathbf{n}_{32} B_{2,2}(t), \qquad (5)$$

$$\mathbf{e}_2(t) = \mathbf{n}_{31} B_{0,2}(t) + \mathbf{h}_2 B_{1,2}(t) + \mathbf{n}_{12} B_{2,2}(t), \qquad (6)$$

$$\mathbf{e}_3(t) = \mathbf{n}_{11} B_{0,2}(t) + \mathbf{h}_3 B_{1,2}(t) + \mathbf{n}_{22} B_{2,2}(t), \qquad (7)$$

where $\mathbf{n}_{i1}$, $\mathbf{n}_{i2}$, and $\mathbf{h}_i$ $(i = 1, 2, 3)$ are control vectors which are derived from control polygons of three boundary curves (Fig. 3). Because the control vectors for $\mathbf{e}_1(t)$, $\mathbf{e}_2(t)$, and $\mathbf{e}_3(t)$ may not be necessarily unit ones, then it is not guaranteed that $\mathbf{n}_{i1} = \mathbf{n}_{i2}$ for $i = 1, 2, 3$, even though each pair of normals has the same direction. We define the normal vector at any interior point of the triangle using Gregory surface interpolation technique [3]. Let

$$\bar{\mathbf{n}}_1 = \begin{cases} \frac{u\mathbf{n}_{11}+v\mathbf{n}_{12}}{u+v} & \text{if } u^2 + v^2 \ne 0, \\ \mathbf{n}_1 & \text{otherwise,} \end{cases}$$

$$\bar{\mathbf{n}}_2 = \begin{cases} \frac{v\mathbf{n}_{21}+w\mathbf{n}_{22}}{v+w} & \text{if } v^2 + w^2 \ne 0, \\ \mathbf{n}_2 & \text{otherwise,} \end{cases}$$

$$\bar{\mathbf{n}}_3 = \begin{cases} \frac{w\mathbf{n}_{31}+u\mathbf{n}_{32}}{w+u} & \text{if } w^2 + u^2 \ne 0, \\ \mathbf{n}_3 & \text{otherwise.} \end{cases}$$

The Gregory normal patch is given as

$$\bar{\mathbf{n}}_{\text{gr}} = \bar{\mathbf{n}}_1 w^2 + \bar{\mathbf{n}}_2 u^2 + \bar{\mathbf{n}}_3 v^2 + 2\mathbf{h}_1 uv + 2\mathbf{h}_2 vw + 2\mathbf{h}_3 wu. \qquad (8)$$

The unit normal corresponding to $\bar{\mathbf{n}}_{\text{gr}}$ can just be obtained as $\mathbf{n}_{\text{gr}} = \frac{\bar{\mathbf{n}}_{\text{gr}}}{\|\bar{\mathbf{n}}_{\text{gr}}\|}$.

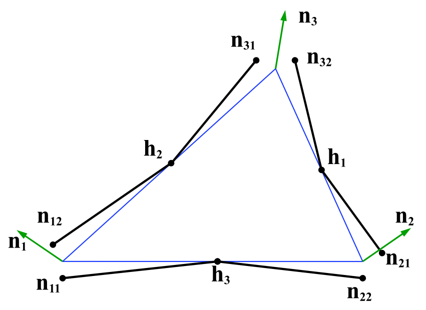


**Fig. 3** Computing normals to a Bézier triangle patch from boundary curves

To match the shape of the interpolating Bézier surface, we further also compute the unit normal vector $\mathbf{n}_{bz}$ of the surface $\mathbf{b}(u,v)$. Then we define the *mixed Gregory normal function* as

$$\mathbf{n}(u,v) = \bigl(1 - s(u,v)\bigr)\mathbf{n}_{\mathrm{gr}}(u,v) + s(u,v)\mathbf{n}_{bz}(u,v), \qquad (9)$$

where $\mathbf{n}_{bz}(u,v)$ is the analytical normal to the cubic triangular Bézier patch $\mathbf{b}(u,v)$. The coefficient $s(u,v)$ is chosen as $s(u,v) = \min\{1, \lambda(27uvw)^2\}$, where $\lambda \geq 0$ is a user defined number.

From the definition of Gregory normal patch we can easily draw a few properties of the new normal function.

1. The Gregory normal patch is continuous over the domain triangle.
2. Any two triangles sharing a common edge have equal unit normal vectors along the joint edge.
3. The normals $\mathbf{n}(u,v)$ and $\mathbf{n}_{\mathrm{gr}}(u,v)$ are equal for any point lying on the boundary of the Bézier triangle.
4. The normal $\mathbf{n}(u,v)$ equals $\mathbf{n}_{\mathrm{gr}}(u,v)$ when $\lambda = 0$, and $\mathbf{n}(u,v)$ will be close to $\mathbf{n}_{bz}(u,v)$ in the interior of the triangle when $\lambda$ is chosen a large number.

## 4.3 Side-vertex normal interpolation

The side-vertex normal interpolation technique is obtained by adapting Nielson's side-vertex surface interpolation [2] to normal interpolation on triangles. The side-vertex normal interpolation permits even higher-order smoothness across boundaries.

When a cubic Bézier triangle has been obtained that interpolates vertices and vertex normals of triangle $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$, the side-vertex interpolation normal to any point on the surface can be computed by the following four steps.

Step 1. Compute normal functions $\hat{\mathbf{e}}_i(t)$ $(i = 1,2,3)$ along the three boundary curves.

Step 2. Compute Bézier curves $\mathbf{b}_i(t)$ $(i = 1,2,3)$ through a point $\mathbf{b}(u,v)$ on the surface and either of triangle vertices $\mathbf{p}_i$, respectively.

Step 3. Compute normal functions $\hat{\mathbf{f}}_i(t)$ $(i = 1,2,3)$ along the three Bézier curves.

Step 4. Blend the normalized normals computed by three normal functions $\hat{\mathbf{f}}_i(t)$ $(i = 1,2,3)$ at the joint point.

We explain in detail how to compute normal functions and Bézier curves mentioned above in the following few paragraphs.

The normal functions $\hat{\mathbf{e}}_i(t)$ $(i = 1,2,3)$ are obtained by modifying quadratic normal functions $\mathbf{e}_i(t)$ which are originally defined in Eqs. (5)–(7). The modification rule is as in Eq. (4). Then we know that normal functions $\hat{\mathbf{e}}_i(t)$ have vanishing derivatives at the ends. If a point $\mathbf{b}(u,v)$ lies on the boundary of the Bézier triangle, the normal vector to the surface can be computed by one corresponding normal function directly.

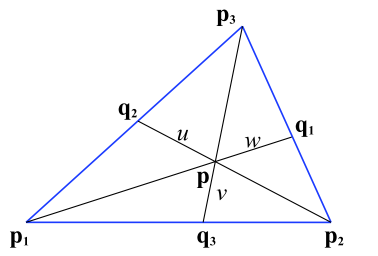


**Fig. 4** The domain of a Bézier triangle

If $u > 0$, $v > 0$, and $w > 0$, the normal vector at point $\mathbf{b}(u,v)$ should be computed by all steps listed above.

For an arbitrary point $\mathbf{p} = w\mathbf{p}_1 + u\mathbf{p}_2 + v\mathbf{p}_3$ in the domain of a Bézier triangle (Fig. 4), the lines that connect points $\mathbf{p}_1$, $\mathbf{p}_2$ or $\mathbf{p}_3$ through point $\mathbf{p}$ intersect opposite edges at $\mathbf{q}_1$, $\mathbf{q}_2$ or $\mathbf{q}_3$, respectively. Then points $\mathbf{q}_1$, $\mathbf{q}_2$, and $\mathbf{q}_3$ can be represented by

$$\mathbf{q}_1 = \frac{u}{u+v}\mathbf{p}_2 + \frac{v}{u+v}\mathbf{p}_3,$$
$$\mathbf{q}_2 = \frac{v}{v+w}\mathbf{p}_3 + \frac{w}{v+w}\mathbf{p}_1,$$
$$\mathbf{q}_3 = \frac{w}{w+u}\mathbf{p}_1 + \frac{u}{w+u}\mathbf{p}_2.$$

With simple computation, it is easy to see that the curves $\mathbf{b}_1(t)$, $\mathbf{b}_2(t)$, and $\mathbf{b}_3(t)$ on the Bézier triangle $\mathbf{b}(u,v)$ corresponding to the line segments $\mathbf{p}_1\mathbf{q}_1$, $\mathbf{p}_2\mathbf{q}_2$, and $\mathbf{p}_3\mathbf{q}_3$ on the domain triangle are just cubic Bézier curves. Representing curve $\mathbf{b}_1(t)$ with Bernstein bases, we have

$$\mathbf{b}_1(t) = R_0 B_{0,3}(t) + R_1 B_{1,3}(t) + R_2 B_{2,3}(t) + R_3 B_{3,3}(t), \qquad (10)$$

where

$$R_0 = \mathbf{b}_{003},$$
$$R_1 = \frac{u\mathbf{b}_{102} + v\mathbf{b}_{012}}{u+v},$$
$$R_2 = \frac{u^2\mathbf{b}_{201} + 2uv\mathbf{b}_{111} + v^2\mathbf{b}_{021}}{(u+v)^2},$$
$$R_3 = \frac{u^3\mathbf{b}_{300} + 3u^2v\mathbf{b}_{210} + 3uv^2\mathbf{b}_{120} + v^3\mathbf{b}_{030}}{(u+v)^3}.$$

If we replace $u$, $v$, and $\mathbf{b}_{ijk}$ within the control points of $\mathbf{b}_1(t)$ by $v$, $w$, and $\mathbf{b}_{kij}$, respectively, we will obtain the control points for curve $\mathbf{b}_2(t)$. Similarly, the control points for $\mathbf{b}_3(t)$ can be computed as those for $\mathbf{b}_1(t)$ by replacing $u$, $v$, and $\mathbf{b}_{ijk}$ with $w$, $u$, and $\mathbf{b}_{jki}$, respectively. Since the three Bézier curves all pass through point $\mathbf{b}(u,v)$, we have $\mathbf{b}(u,v) = \mathbf{b}_1(u+v) = \mathbf{b}_2(v+w) = \mathbf{b}_3(w+u)$.

The normal functions $\hat{\mathbf{f}}_i(t)$ along curves $\mathbf{b}_i(t)$ $(i = 1,2,3)$ can be computed by the control polygons and normal vectors at the ends. It is easy to verify that $\mathbf{b}_i(0) = \mathbf{p}_i$. Then the normal vectors at end points $\mathbf{b}_i(0)$ should be chosen as $\mathbf{n}_i$ for $i = 1,2,3$, respectively. As to the normals at $\mathbf{b}_i(1)$, they can be computed by normal functions $\hat{\mathbf{e}}_i(t)$ $(i = 1, 2, 3)$. The unit normal vectors at $\mathbf{b}_i(1)$ $(i = 1, 2, 3)$ are obtained by

$$\mathbf{v}_1 = \frac{\hat{\mathbf{e}}_1(\frac{v}{v+u})}{\|\hat{\mathbf{e}}_1(\frac{v}{v+u})\|}, \quad \mathbf{v}_2 = \frac{\hat{\mathbf{e}}_2(\frac{w}{w+v})}{\|\hat{\mathbf{e}}_2(\frac{w}{w+v})\|}, \quad \mathbf{v}_3 = \frac{\hat{\mathbf{e}}_3(\frac{u}{u+w})}{\|\hat{\mathbf{e}}_3(\frac{u}{u+w})\|},$$

respectively. For each side-vertex parametric curve $\mathbf{b}_i(t)$ with given normal vectors $\mathbf{n}_i$ and $\mathbf{v}_i$ at the ends, the normal function $\hat{\mathbf{f}}_i(t)$ with vanishing end derivatives can be obtained by Eq. (4).

Let $\mathbf{f}_i(t) = \frac{\hat{\mathbf{f}}_i(t)}{\|\hat{\mathbf{f}}_i(t)\|}$ $(i = 1, 2, 3)$. We define the side-vertex interpolation normal to the Bézier triangle as

$$\bar{\mathbf{n}}(u, v) = \frac{u^2v^2\mathbf{f}_1(u+v) + v^2w^2\mathbf{f}_2(v+w) + w^2u^2\mathbf{f}_3(u+w)}{u^2v^2 + v^2w^2 + w^2u^2}. \tag{11}$$

In practice, we should always use normalized normal vectors for surface rendering. Therefore, we compute normalized vector from Eq. (11) as

$$\mathbf{n}(u, v) = \frac{u^2v^2\mathbf{f}_1(u+v) + v^2w^2\mathbf{f}_2(v+w) + w^2u^2\mathbf{f}_3(u+w)}{\|u^2v^2\mathbf{f}_1(u+v) + v^2w^2\mathbf{f}_2(v+w) + w^2u^2\mathbf{f}_3(u+w)\|}. \tag{12}$$

The blending normal function $\bar{\mathbf{n}}(u, v)$ or the normalized normal function $\mathbf{n}(u, v)$ have the following properties:

1. The normal function $\mathbf{n}(u, v)$ interpolates normals $\hat{\mathbf{e}}_i(t)$ $(i = 1, 2, 3)$ at the boundary of the triangle.
2. The blending normal function $\bar{\mathbf{n}}(u, v)$ interpolates the derivatives $\mathbf{f}_i'(0)$ and $\mathbf{f}_i'(1)$ for $i = 1, 2, 3$.
3. The normal function $\mathbf{n}(u, v)$ interpolates the derivatives $\mathbf{f}_i'(0)$ and $\mathbf{f}_i'(1)$ for $i = 1, 2, 3$.

Property 1 can be drawn directly from Eq. (12). To prove the next two properties, we assume that the line segments $\mathbf{p}_i\mathbf{q}_i$ $(i = 1, 2, 3)$ are parameterized as $(u_i(t), v_i(t))$ $(t \in [0, 1])$, and let normal function $\mathbf{r}_i(t) = \bar{\mathbf{n}}(u_i(t), v_i(t))$. Similar to side-vertex surface interpolation [2], we know that $\bar{\mathbf{n}}(u, v)$ interpolates the derivatives $\mathbf{f}_i'(0)$ and $\mathbf{f}_i'(1)$ for $i = 1, 2, 3$. We have $\mathbf{r}_i'(0) = \mathbf{f}_i'(0) = \hat{\mathbf{f}}_i'(0) = 0$ and $\mathbf{r}_i'(1) = \mathbf{f}_i'(1) = \hat{\mathbf{f}}_i'(1) = 0$. Furthermore, the normal function

$$\mathbf{n}\big(u_i(t), v_i(t)\big) = \frac{\mathbf{r}_i(t)}{\|\mathbf{r}_i(t)\|}$$

has vanishing derivatives at the ends, too.

## 5 Complexity analysis

In this section we present the complexity analysis of normal interpolation by the PN triangle method, the Gregory normal patch, and the side-vertex normal interpolation scheme. Some potential methods to improve the efficiency of the new proposed normal interpolation schemes are also discussed.

For a triangular mesh that has $N$ triangles, we assume that $n$ points, together with unit normal vectors at the points, have been sampled or computed from curved triangles which are constructed from the input triangles and the input vertex normals. Because normal patches by the three normal interpolation methods are all dependent on normal functions on boundaries of triangles, we should analyze the complexities of the construction of boundary normal functions as well as normal computation by normal patches. To compute a unit normal vector from a normal function, it may need several operations of vector addition/subtraction, scalar multiplication/division, scaling of vectors, cross-product between vectors, and normalization of vectors. Noticing that one operation of vector scaling equals three scale multiplications and one cross-product between two vectors in space equals six scale multiplications, we will only counter the numbers of vector addition/subtraction, scalar multiplication/division, and vector normalization when coefficients of normal functions or interpolating normals have been computed.

Based on the formulas presented in [8], the coefficients of the normal patch of a PN triangle can be computed by the input vertices and vertex normals. For example, the computation of $\mathbf{h}_1$ in Eq. (2) needs three operations of vector addition/subtraction, 11 scalar multiplications/divisions, and one operation of vector normalization. When an interpolating normal vector is computed by Eq. (2), it needs five operations of vector addition/subtraction, 24 scalar multiplications/divisions, and one vector normalization operation. In total, there are $9N + 5n$ operations of vector addition/subtraction, $33N + 24n$ scalar multiplications/divisions, and $3N + n$ vector normalization operations for normal interpolation of a triangular mesh.

The computation of coefficients of boundary normal function $\mathbf{e}_1(t)$ of a curved triangle by Eq. (3) needs five operations of vector addition/subtraction and 12 scalar multiplications/divisions. Because the multiplication of a vector by the length of another vector is equivalent to the normalization of a vector, the computation also needs three vector normalization operations. When a vector is computed by Gregory normal patch using Eq. (8), it needs eight operations of vector addition/subtraction, 48 scalar multiplications/divisions, and one vector normalization operation. As a whole, it needs $15N + 8n$ operations of vector addition/subtraction, $36N + 48n$ scalar multiplications/divisions, and $9N + n$ vector normalization operations to compute interpolating normals for a triangle mesh. If the mixed Gregory normal patch is used, one should also compute the unit normal vector to the Bézier triangle. To compute an interpolating normal vector by Eq. (9), in comparison with Eq. (8), additional 11 operations of vector addition/subtraction, 65 scalar multiplications, and two vector normalization operations are needed. The computational complexity of mixed Gregory normal method includes $15N + 19n$ operations of vector addition/subtraction, $36N + 113n$ scalar multiplications/divisions, and $9N + 3n$ operations of vector normalization.

After the construction of boundary normal functions $\mathbf{e}_i(t)$ $(i = 1, 2, 3)$, the computation of an interpolating normal by the side-vertex normal interpolation scheme needs to compute three Bézier curves on a Bézier triangle, three normal functions along the Bézier curves, and the blending vector of three unit vectors computed by the normal functions. It needs six operations of vector addition/subtraction and totally 45 scalar multiplications to compute a Bézier curve $\mathbf{b}_i(t)$ by Eq. (10), and it needs three vector addition/subtraction operations, 30 scalar multiplications, and one vector normalization operation to compute a unit end normal vec-

**Table 1** The complexities for normal interpolation on a triangular mesh using various methods

| Method | Vector+/− | Scalar×/÷ | VecNorm |
|---|---|---|---|
| quadratic | $9N + 5n$ | $33N + 24n$ | $3N + n$ |
| Gregory | $15N + 8n$ | $36N + 48n$ | $9N + n$ |
| mixGreg.($\lambda > 0$) | $15N + 19n$ | $36N + 113n$ | $9N + 3n$ |
| Side-Vertex | $15N + 53n$ | $36N + 366n$ | $9N + 16n$ |

**Table 2** Time costs for point sampling and independent normal interpolation on a cubic Bézier patch (seconds)

| #point | PtSamp | PNtri. | Greg. | mixGr. | SideVtx |
|---|---|---|---|---|---|
| $1 \times 10^5$ | 0.047 | 0.047 | 0.094 | 0.172 | 0.546 |
| $2 \times 10^5$ | 0.078 | 0.078 | 0.187 | 0.358 | 1.077 |
| $1 \times 10^6$ | 0.436 | 0.437 | 0.936 | 1.747 | 5.413 |

**Table 3** Average normal deviation angles for the test models (degree)

| Model | #point | Phong. | PNtri. | Greg. | SideVtx |
|---|---|---|---|---|---|
| star | 4620 | 30.860 | 29.446 | 16.425 | 18.213 |
| costa | 5082 | 12.422 | 15.721 | 9.315 | 12.280 |
| cat | 155001 | 7.508 | 7.057 | 3.799 | 4.984 |
| Fig. 7 | 106260 | 24.974 | 25.790 | 15.052 | 17.501 |
| Fig. 8(a) | 98406 | 18.834 | 19.832 | 13.163 | 15.951 |
| Fig. 8(d) | 95634 | 17.144 | 17.622 | 11.536 | 13.672 |
| Fig. 8(g) | 98406 | 16.740 | 17.627 | 11.593 | 13.791 |

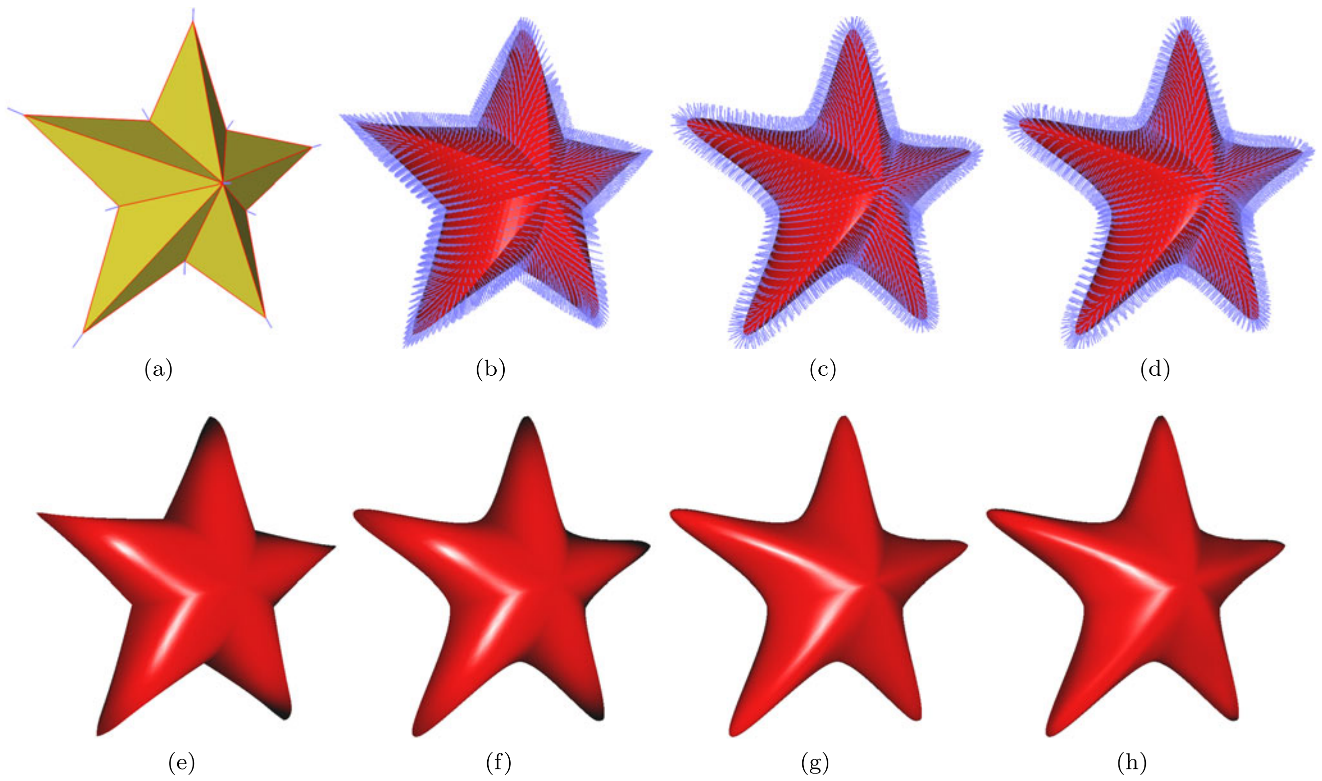


**Fig. 5** (**a**) The input triangular mesh with known vertex normals; (**b**) The Phong tessellated mesh and the interpolating quadratic normals; (**c**) The cubic surface patches with mixed Gregory normal interpolation ($\lambda = 0.5$); (**d**) The cubic surface patches with side-vertex normal interpolation; (**e**) The shading result by Phong tessellation; (**f**) The shading result by PN triangles; (**g**) The shading result by mixed Gregory normal function; (**h**) The shading result by side-vertex normal interpolation

tor $\mathbf{v}_i$ for the Bézier curve by the updated boundary normal function $\hat{\mathbf{e}}_i(t)$. To compute a unit normal vector $\mathbf{f}_i(t)$ on Bézier curve $\mathbf{b}_i(t)$, it needs eight operations of vector addition/subtraction, 32 scalar multiplications, and four vector normalization operations. Even when an interpolating normal vector is finally obtained by Eq. (12), it still needs two operations of vector addition/subtraction, 15 scalar multiplications, and one vector normalization operation. The total complexity of this normal interpolation scheme is summarized in Table 1.

From Table 1 we can see that all the mentioned normal interpolation schemes have linear complexity with respect to the triangle number and the sampling rate, but the proposed schemes need more vector operations or scalar multiplications/divisions than the quadratic normal method for the computation of interpolating normals. As can be seen in the next section, the new normal interpolation schemes can generate better rendering results.

Despite their increased computational costs, the new methods can still be used for fast surface rendering using hardware acceleration or combination with other techniques of surface rendering. Note that a mixed Gregory normal vector can be decomposed into a Gregory normal vector and a Bézier surface normal vector. The three Bézier curves and the corresponding normal functions for side-vertex normal interpolation can also be computed independently with each other. Then, the proposed normal interpolation schemes can be used for realistic and high-speed surface shading on modern computers that have parallel processing units like GPUs. An alternative way to achieve both rendering quality and speed is combining Gregory or side-vertex normal interpolation for rough tessellation and quadratic normal interpolation for detailed tessellation together.

## 6 Examples and comparisons

We have implemented the Phong tessellation, the PN triangles method, and our new proposed normal interpolation schemes by C++ code on a PC with Intel(R) Core(TM)2 CPU, T9900 @3.06 GHz 3.07 GHz, and 4 G RAM. In this section we present several interesting examples to show the shading results by different normal interpolation methods. The comparisons of time costs using various normal interpolation schemes or tessellation methods are also given.

We first compare the time costs of normal interpolation using various normal interpolation schemes on a curved Bézier triangle. We have computed 100000, 200000, and 1000000 unit normal vectors on a curved triangle using the PN triangle method, the Gregory normal patch, the mixed Gregory normal method ($\lambda = 0.5$) or the side-vertex normal interpolation scheme, respectively. The time costs are given

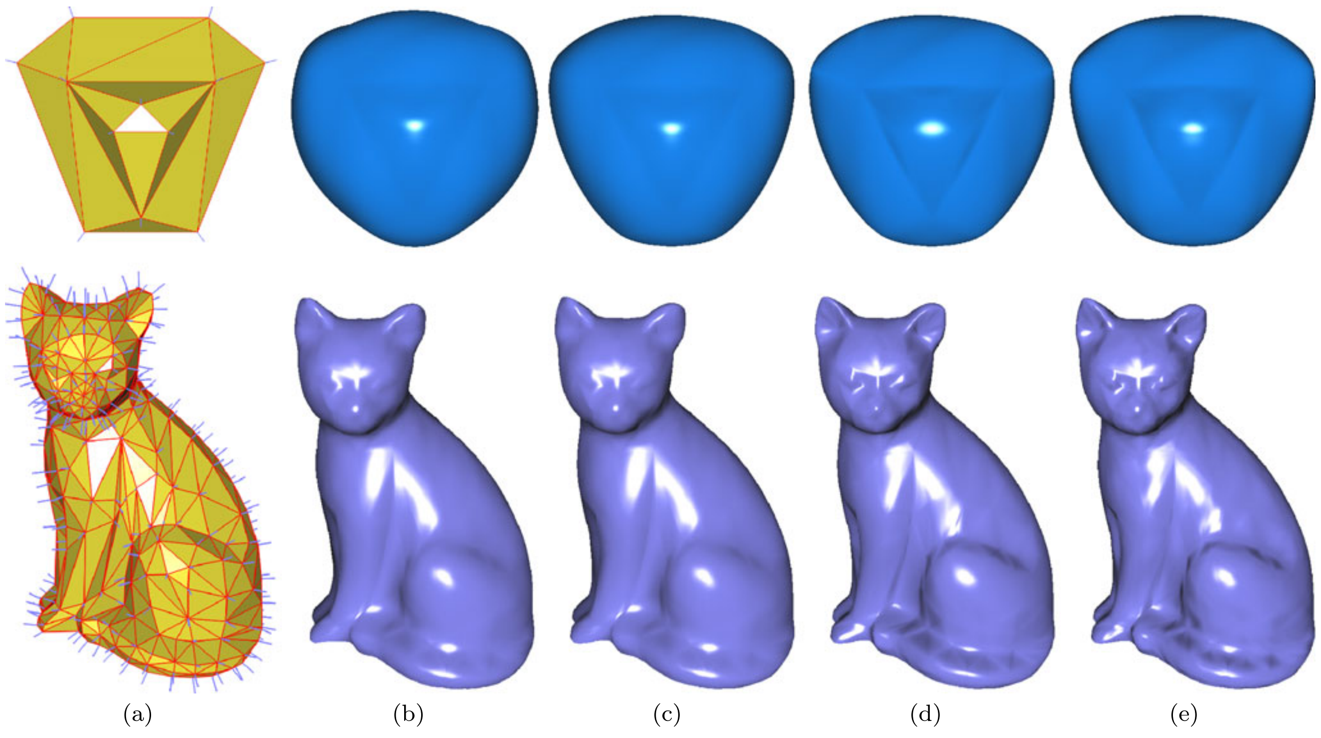


**Fig. 6** (**a**) The costa model (*up*) and cat model (*bottom*) with known vertex normals; (**b**) The shading results by Phong tessellation; (**c**) The shading results by PN triangles; (**d**) The shading results by mixed Gregory normal interpolation with $\lambda = 0.5$; (**e**) The shading results by side-vertex normal interpolation

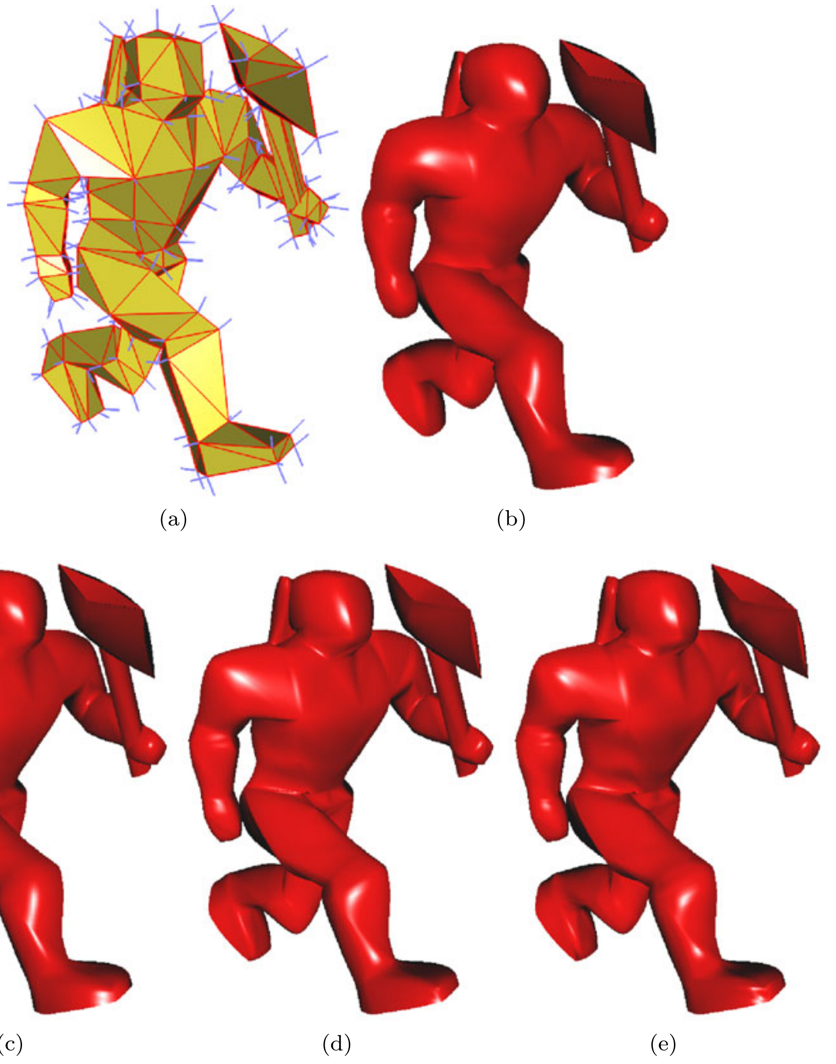


**Fig. 7** (**a**) The input triangular mesh with known vertex normals; (**b**) The shading result by Phong tessellation; (**c**) The shading result by PN triangles; (**d**) The shading result by mixed Gregory normal interpolation with $\lambda = 0.5$; (**e**) The shading result by side-vertex normal interpolation

in Table 2. In this table, the time costs for point sampling are also given for comparison purpose. From the table we can see that all normal interpolation methods are very fast even a huge number of normal vectors have to be computed, and the new methods take more time due to their increased complexities. From the table we also learn that the time costs for all the methods are approximately linear with respect to the sampling rates.

To compare the shading results, we first tessellate and render a solid star model that consists of 20 triangles. The initial vertex normals are computed from facet normals for this example (Fig. 5(a)). Figure 5(b) illustrates the surface by Phong tessellation and the quadratic normals by the algorithm proposed in [8]. Even though the normal field changes smoothly across the whole surface, the normal vectors deviate from those of interpolating surfaces evidently, especially in sharp corner regions of the star model. As a comparison, the interpolating normals by our proposed schemes match the interpolating surfaces more naturally. Figures 5(c) and (d) show the tessellated surfaces by cubic surface interpolation and normal vectors by Gregory normal interpolation or by side-vertex normal interpolation, respectively. We note that all pictures in this paper are rendered using a light set in infinity in the direction of the sight.

**Table 4** Comparison of total time costs for tessellating the test models (seconds)

| Model | #Patch. | Phong. | PNtri. | Greg. | SideVtx |
|---|---|---|---|---|---|
| star | 20 | 0.004 | 0.004 | 0.008 | 0.020 |
| costa | 22 | 0.004 | 0.004 | 0.008 | 0.023 |
| cat | 671 | 0.125 | 0.125 | 0.343 | 0.733 |
| Fig. 7 | 460 | 0.093 | 0.094 | 0.218 | 0.499 |
| Fig. 8(a) | 426 | 0.078 | 0.078 | 0.203 | 0.453 |
| Fig. 8(d) | 414 | 0.078 | 0.078 | 0.202 | 0.437 |
| Fig. 8(g) | 426 | 0.078 | 0.078 | 0.203 | 0.460 |

To compare the normal interpolation schemes quantitatively, we compute the average normal deviation angles be-

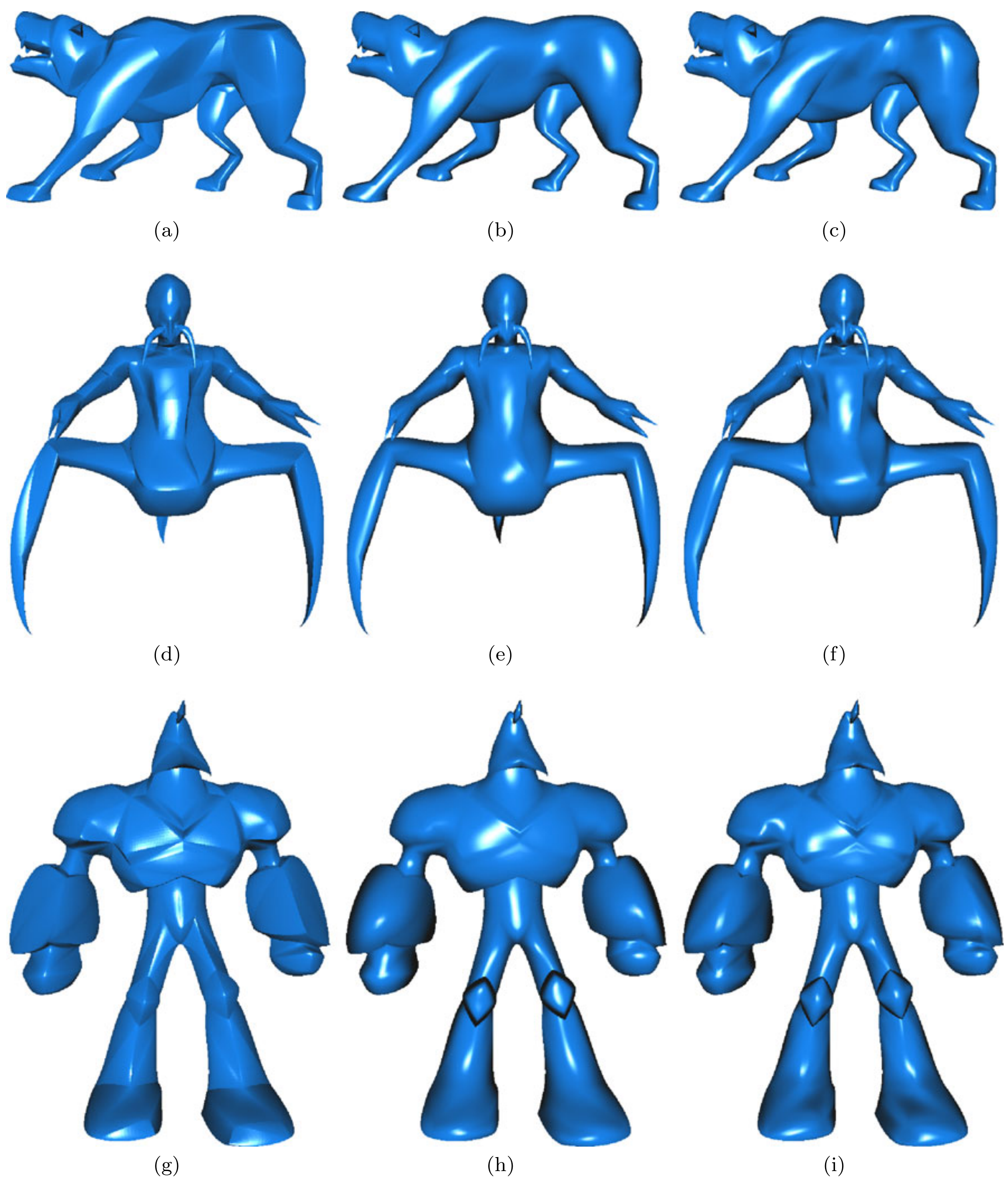


**Fig. 8** Comparisons of shading results by various normal interpolation schemes. (*left*) The triangular meshes with tessellated cubic Bézier surfaces; (*middle*) The shading results by quadratic normal interpolation; (*right*) The shading results by side-vertex normal interpolation

tween the analytical normals of local interpolating surfaces and the interpolating normals at all sample points. From Table 3 we can see that the average normal deviation angles using quadratic normal interpolation for Phong tessellation or local cubic surface interpolation are always larger than those by Gregory normal interpolation or side-vertex normal interpolation. Large normal deviation may produce unrealistic shading results. See, for example, the corners of the star model in Figs. 5(e) or (f). Our proposed new normal interpolation methods are aware of local shapes very well; see Figs. 5(g) and (h).

In Fig. 6 we present another two models rendered using Phong tessellation, cubic surface interpolation, and the proposed methods of normal interpolation. Figures 6(b) and (c) show the shading results by Phong tessellation or cubic surface interpolation, both by quadratic normal interpolation.

Though the two figures have similar appearance, the cubic surface interpolation scheme has given more naturally contour shapes. Even though the rendered pictures are visually smooth, many salient surface features have been blurred by using quadratic normal interpolation. See, for example, the protruding part of the costa model and the ears of the cat model are over-smoothed in Figs. 6(b) and (c). Our proposed normal interpolation methods can be used to render surfaces with smooth appearance across edges as well as preservation of salient geometric features; see Figs. 6(d) and (e).

As pointed out in reference [8], there may exist some long and thin triangles in a triangular mesh for the representation of sharp edges or creases on the surface. See, for example, the sharp edges within the axe model in Fig. 7. For thin triangles on a triangular mesh, the initial vertex normals may deviate from the triangle normal largely (Fig. 7(a)). The quadratic normals which are determined from vertex normals alone may deviate from normals of the interpolating surfaces considerably, too. When the tessellated surfaces have been rendered using quadratic normals, they may suffer severe shading defects near the sharp edges. See, for example, the axe edges in Figs. 7(b) and (c). By Gregory interpolation normals or side-vertex interpolation normals, all curved triangles can be rendered into highly realistic shading results with no need of special handing of sharp triangles. See Figs. 7(d) and (e) for the shading results by these two new normal interpolation methods, respectively. The initial models in this and the next figure are from reference [8].

Finally, we present a few more examples of surface rendering by our new proposed normal interpolation method. Because surface rendering by Phong tessellation or by PN triangles have similar appearance, we only plot the shading results by PN triangles for comparison purpose. Because side-vertex normal interpolation can usually generate better (or at least similar results) than Gregory normal interpolation, only rendering results by side-vertex normal interpolation are given. The tessellated results of interpolating cubic Bézier surfaces are given in the left column of Fig. 8. The middle column shows the shading results by quadratic normal interpolation, and the right column shows the shading results by our proposed side-vertex normal interpolation. From the figure we can see that quadratic normal interpolation can give visually smooth shading results, but many local salient features have been smoothed out by this method. See, for example, Fig. 8(h), we notice that salient features in the breast have been blurred while unrealistic dark contours near the knees of the model occur due to improper normals by quadratic normal interpolation. Figure 8(i) shows that our new proposed normal interpolation method can be used to render a rough surface with smooth shading result, together with well preservation of visually important surface features.

The total time costs for points sampling and normal interpolation of the above examples by various methods are given in Table 4. From the table we can see that all four methods for surface shading are very fast. In particular, the Gregory normal interpolation scheme and the side-vertex normal interpolation method take more calculation time than the other two methods. Then users can choose different normal interpolation methods with a balance of rendering quality and speed. They can also accelerate the speed further using the methods discussed in Sect. 5.

## 7 Conclusions

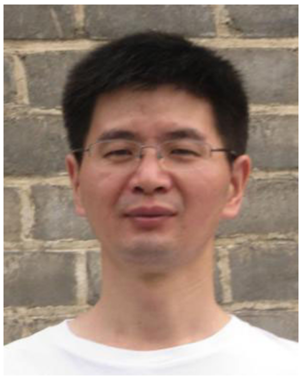
We have modified two well-known local surface interpolation methods for normal interpolation for curved surface rendering from polyhedral approximation. The interpolating normal vectors are computed along with cubic Bézier surface interpolation for every triangle on a triangular mesh with given or estimated normal vectors at vertices. The interpolating normals are joined continuously or even smoothly between neighboring triangles, and they match the shape of local interpolating surfaces very well. Salient surface features and silhouettes can be preserved well when a surface is rendered by either of the proposed normal interpolation schemes. The local interpolation of Bézier surfaces and normal vectors also makes it very convenient for high-speed hardware implementation. Though we proposed normal interpolation methods for rendering triangular meshes, the proposed technique can be extended to normal interpolation for rendering of other types of meshes, too.

**Acknowledgements** We owe thanks to anonymous referees for their helpful comments on an earlier version of the paper.

## References

1. Peters, J.: Local surface interpolation: a classification. Comput. Aided Geom. Des. **7**, 191–195 (1990)
2. Nielson, G.M.: The side-vertex method for interpolation in triangles. J. Approx. Theory **25**, 318–336 (1979)
3. Gregory, J.: Smooth interpolation without twist constraints. In: Barnhill, R.E., Riesenfeld, R.F. (eds.) Computer Aided Geometric Design, pp. 71–87. Academic Press, New York (1974)
4. Shirman, L.A., Séquin, C.H.: Local surface interpolation with Bézier patches. Comput. Aided Geom. Des. **4**, 279–295 (1987)
5. Walton, D.J., Meek, D.S.: A triangular $G^1$ patch from boundary curves. Comput. Aided Des. **28**(2), 113–123 (1996)
6. Hahmann, S., Bonneau, G.-P.: Polynomial surfaces interpolating arbitrary triangulations. IEEE Trans. Vis. Comput. Graph. **9**(1), 99–109 (2003)
7. van Overveld, C.W.A.M., Wyvill, B.: Phong normal interpolation revisited. ACM Trans. Graph. **16**(4), 397–419 (1997)
8. Vlachos, A., Peters, J., Boyd, C., Mitchell, J.: Curved PN triangles. In: Proceedings of the 2001 Symposium on Interactive 3D Graphics, pp. 159–166 (2001)
9. Boubekeur, T., Alexa, M.: Phong tessellation. ACM Trans. Graph. **27**(5), 141 (2008)

10. Wang, L., Tu, C., Wang, W., Meng, X., Chan, B., Yan, D.: Silhouette smoothing for real-time rendering of mesh surfaces. IEEE Trans. Vis. Comput. Graph. **14**(3), 640–652 (2008)
11. Schwarz, M., Stamminger, M.: Fast GPU-based adaptive tessellation with CUDA. Comput. Graph. Forum **28**(2), 365–374 (2009)
12. Phong, B.T.: Illumination for computer generated pictures. Commun. ACM **18**(6), 311–317 (1975)
13. Walia, E., Singh, C.: An analysis of linear and non-linear interpolation techniques for three-dimensional rendering. In: Intern. Conference on Geometric Modeling and Imaging—New Trends, pp. 69–76 (2006)
14. Lee, Y.-C., Jen, C.-W.: Improved quadratic normal vector interpolation for realistic shading. Vis. Comput. **17**, 337–352 (2001)
15. Fünfzig, C., Müller, K., Hansford, D., Farin, G.: PNG1 triangles for tangent plane continuous surfaces on the GPU. In: Proceedings of Graphics Interface, Canada, pp. 219–226 (2008)
16. Boubekeur, T., Reuter, P., Schlick, C.: Scalar tagged PN triangles. In: Proc. Eurographics'05, pp. 17–20 (2005)
17. Ni, T., Castaño, I., Peters, J., Mitchell, J., Schneider, P., Verma, V.: Efficient substitutes for subdivision surfaces. In: ACM SIGGRAPH 2009 Courses, Article No. 13 (2009)
18. Boubekeur, T., Schlick, C.: QAS: Real-time quadratic approximation of subdivision surfaces. In: Proceedings of Pacific Graphics 2007, pp. 453–456 (2007)
19. Loop, C., Schaefer, S.: Approximating Catmull–Clark subdivision surfaces with bicubic patches. ACM Trans. Graph. **27**(1), Article No. 8 (2008)
20. Loop, C., Schaefer, S., Ni, T., Castaño, I.: Approximating subdivision surfaces with Gregory patches for hardware tessellation. ACM Trans. Graph. **28**(5), Article No. 151 (2009)
21. Li, G., Ren, C., Zhang, J., Ma, W.: Approximation of Loop subdivision surfaces for fast rendering. IEEE Trans. Vis. Comput. Graph. **17**(4), 500–514 (2011)
22. Alexa, M., Boubekeur, T.: Subdivision shading. ACM Trans. Graph. **27**(5), Article No. 142 (2008)
23. Max, N.: Weights for computing vertex normals from facet normals. J. Graph. Tools **4**(2), 1–6 (1999)
24. Farin, G.: Triangular Bernstein–Bézier patches. Comput. Aided Geom. Des. **3**(2), 83–127 (1986)
25. Farin, G.: Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide, 5th edn. Academic Press, San Diego (2002)



**Xunnian Yang** received his bachelor's degree in mathematics from Anhui University, Hefei, China, in 1993 and Ph.D. in computer aided geometric design and computer graphics from Zhejiang University, Hangzhou, China, in 1998. He is now an associate professor in Department of Mathematics of Zhejiang University. His main research interests include geometric modeling, computer graphics, and image processing.

**Jianmin Zheng** received the B.S. and Ph.D. degrees from Zhejiang University, China. He is an associate professor in the School of Computer Engineering at Nanyang Technological University. Previously, he was a faculty member at Zhejiang University and a research faculty at Brigham Young University, Provo, Utah. His research interest includes computer aided geometric design, CAD/CAM, computer graphics, animation, visualization, and interactive digital media.