Computer Aided Geometric Design 35–36 (2015) 69–81

Contents lists available at ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

# GPU-based smooth free-form deformation with sharp feature awareness

Yuanmin Cui, Jieqing Feng *

*State Key Laboratory of CAD&CG, Zhejiang University, China*

ARTICLE INFO

*Article history:*
Available online 24 March 2015

*Keywords:*
Smooth FFD
Sharp feature preserving
GPU
Cubic triangular Bézier patch
Normal field

ABSTRACT

In an accurate free-form deformation of a polygonal object, only the linear geometry, e.g., triangles or planar polygons, is deformed as triangular Bézier patches or trimmed tensor product Bézier patches; the related normal field is not considered. Thus, the geometry appearance and shading of the deformed object are typically not smooth. In this paper, both the linear geometry and normal of a polygonal object are simultaneously considered in the framework of accurate free-form deformation. First, each triangle and its normal field are deformed as two cubic triangular Bézier patches. Then, the curved geometry corresponding to the deformed triangles is locally adjusted to tone the smoothness of the geometry appearance according to the deformed normal field. The deformed normal field is adjusted accordingly. As a result, a smooth free-form deformation with visually plausible smooth geometry and shading is obtained. Furthermore, the sharp features in the polygonal object can be preserved. Because the curved geometry and normal field adjustments are local operations, all of the above computations can be performed in parallel on a GPU. The experimental results show that the method can deform a complex polygonal object as a smooth object in real time while preserving sharp features.



## 1. Introduction

Free-form deformation (FFD) is a prevalent shape manipulation and shape animation method in computer graphics and geometric modeling (Sederberg and Parry, 1986). Classic FFD is conducted on the sampled points of the geometric model. However, the approach tends to produce an aliased deformation result when using a low sampling density.

As an alternative, accurate FFD (Feng et al., 1998, 2002; Feng and Peng, 2000) deforms the planar polygons as a set of triangular Bézier patches or trimmed tensor-product Bézier patches based on the functional composition of Bernstein polynomials (DeRose, 1988; DeRose et al., 1993). The deformation result is accurate for polygonal objects in theory. The accurate FFD considers only the geometry of the original model, excluding its normal field. As a result, the deformed object is only position continuous ($C^0$) for its geometry. Its geometric appearance (e.g., silhouettes and the common edge of two patches) and shading are not smooth because the normal field is discontinuous.

In mathematics, the normal is a differential attribute of the surface. In many graphics applications, the geometry and normal of a polygonal object are independently defined. Each vertex is equipped with one or more normals. In general, the independent normal is an approximation of the potentially true normal. The sharp features, such as sharp edges and corners, can also be preserved by assigning several normals to one vertex. For example, Phong shading can achieve smooth shading

* Corresponding author.
*E-mail address:* jqfeng@cad.zju.edu.cn (J. Feng).

http://dx.doi.org/10.1016/j.cagd.2015.03.002
0167-8396/


**Fig. 1.** Examples of classic FFD, accurate FFD and smooth FFD.

effect at a low computational cost via linear normal interpolation across a triangle. To alleviate the unsmooth silhouette problem in Phong shading, the PN-triangle (Vlachos et al., 2001) method and Phong tessellation (Boubekeur and Alexa, 2008) method alter the linear geometry of a triangle as a curved geometry according to the related normal field.

In this paper, a new GPU-based smooth FFD with sharp feature awareness is proposed for polygonal objects in the framework of accurate FFD, where both the geometry and normal are simultaneously considered. The deformations of the linear geometry (triangle) and linear normal field defined on the triangles are approximated as two cubic triangular Bézier patches for efficiency. As a result, the geometry appearance and shading of the deformed objects are visually plausible smooth, and the sharp features can also be well preserved. Fig. 1 shows examples of various FFD results. The main contributions of the paper are summarized as follows:

- Both the linear geometry and its normal field are considered in the framework of accurate FFD.
- The deformed object exhibits a continuous shading mimicking a $G^1$ surface over smooth edges (implemented as $C^0$ normal field over a $C^0$ geometry), while sharp edges exhibit a discontinuous normal field.
- All of the computations are local and can be implemented fully in parallel on a GPU.

## 2. Related work

FFD, which was first proposed by Sederberg and Parry (1986), is an intuitive model manipulation and soft object animation method. The main concept of FFD is to embed the object into an intermediate space, e.g., a Bézier volume. Users first edit the shape of the intermediate space; then, the space deformation is transferred to the embedded object, whereas the topological connectivity of the object remains unchanged. There are many successive studies regarding FFD. Most of these studies focus on improving the interactive means of FFD (Coquillart, 1990; Hui, 2002; MacCracken and Joy, 1996; McDonnell et al., 2007; Xu et al., 2013). Gain and Bechmann (2008) provided a detailed survey of these methods.

Traditional FFD and its extensions deform the sampled vertices of the model. Thus, the quality of the deformation result depends on the sampling density of the vertex. As a solution to the sampling problem, adaptive upsampling approaches (Gain and Dodgson, 1999; Griessmair and Purgathofer, 1989; Parry, 1986) are more efficient than the naive uniform upsampling approach on CPU. The adaptive upsampling approaches consider the polygon size or surface curvature and upsample the model if necessary. But they cannot handle certain special or pathological cases well, and are difficult to be ported on GPU. Accurate FFD, which was proposed by Feng et al. (1998, 2002), Feng and Peng (2000), is an alternative approach to solving the sampling problem. However, it is computationally intensive, and it also consumes considerable bandwidth, i.e., transferring a large amount of data from the CPU to the GPU after the intensive computations are performed in the CPU. Thus, the algorithms are not interactive or performed in real time in practical applications.

In the recent years, GPUs have been widely adopted for FFD implementations due to their tremendous parallel computing power. Chua and Neumann (2000) proposed an OpenGL-oriented hardware evaluator sub-system to accelerate FFD evaluations. However, none of the GPU vendors integrate this type of dedicated sub-system into their GPUs. In contrast, GPUs have evolved into general-purpose many-core processors. Schein and Elber (2006) implemented a GPU-accelerated FFD using the NVIDIA CG language. Jung et al. (2011) achieved the same goal using NVIDIA CUDA and embedded it to the X3D system. Hahmann et al. (2012) proposed a GPU-based, volume-preserving FFD. They employed the multilinear property of volume constraint and derived an explicit solution. The GPU acceleration component implemented by CUDA is 6.5-times faster than its CPU counterpart.

Cui and Feng (2013, 2014) proposed GPU-based accurate FFD of polygonal objects, the results of which are represented in terms of trimmed tensor product Bézier patches or triangular Bézier patches. They are sufficiently efficient to meet the real-time or interactive demands of large-scale models. However, the deformation is only performed on the linear geometry, without considering the normal of the model. The actual normal of the resulting Bézier patch is adopted for rendering. Due to the piecewise linear continuity of the polygonal object, the deformed object is only $C^0$ continuous. As a result, both the geometric appearance and shading effect are unsmooth. The PN-triangle method (Vlachos et al., 2001) decouples the linear geometry and normal information of a polygonal object to achieve visually plausible smooth geometry and shading, in which it adopts cubic and quadratic triangular Bézier surfaces to represent the geometry and normal, respectively. Phong tessellation (Boubekeur and Alexa, 2008) proposed by Boubekeur et al. uses scalar tags to solve the sharp edge problem.

The decoupled approaches inspired us to propose a novel method to address the above smoothness problem in the accurate FFD of polygonal objects.

## 3. Overview of accurate FFD in terms of triangular Bézier patches

The GPU-based accurate FFDs (Cui and Feng, 2013, 2014) of polygonal objects adopt trimmed tensor product Bézier patches and triangular Bézier patches as the deformation result, respectively. In this paper, we adopt the framework of accurate FFD using triangular Bézier patches (Feng et al., 1998; Feng and Peng, 2000; Cui and Feng, 2014) since it is more efficient than the one using trimmed tensor product Bézier patches. Some basic notation is introduced below.

$\mathbf{R}(u, v, w)$ is a B-spline volume of degree $n_u \times n_v \times n_w$ with $m_u \times m_v \times m_w$ control points:

$$\mathbf{R}(u, v, w) = \sum_{i=0}^{m_u-1} \sum_{j=0}^{m_v-1} \sum_{k=0}^{m_w-1} \mathbf{R}_{ijk} N_{i,n_u}(u) N_{j,n_v}(v) N_{k,n_w}(w) \tag{1}$$

where $\{\mathbf{R}_{ijk}\}_{i=0,\ j=0,\ k=0}^{m_u-1,m_v-1,m_w-1}$ are the control points, $\{N_{i,n_u}(u)\}_{i=0}^{m_u-1}$, $\{N_{j,n_v}(v)\}_{j=0}^{m_v-1}$ and $\{N_{k,n_w}(w)\}_{k=0}^{m_w-1}$ are normalized B-spline basis functions, and $\{u_i\}_{i=0}^{n_u+m_u}$, $\{v_i\}_{j=0}^{n_v+m_v}$ and $\{w_k\}_{k=0}^{n_k+m_k}$ are the knot vectors along the $u$, $v$ and $w$ directions, respectively. Every three-dimensional region $[u_i, u_{i+1}] \times [v_j, v_{j+1}] \times [w_k, w_{k+1}]$ is called a knot box, where $n_u \le i < m_u$, $n_v \le j < m_v$ and $n_w \le k < m_w$, respectively.

As described in Feng et al. (1998), Feng and Peng (2000), Cui and Feng (2014), each polygon of the model is first clipped against the knot boxes such that the generated sub-polygons lie inside of a knot box. Second, the generated sub-polygons are triangulated. The accurate FFD of such a sub-triangle in a knot box governed by $\mathbf{R}(u, v, w)$ is a triangular Bézier patch (Feng et al., 1998; Feng and Peng, 2000), whose degree is $n = n_u + n_v + n_w$. Let the triangular Bézier patch be denoted as $\mathbf{P}(u, v, w)$:

$$\mathbf{P}(u, v, w) = \sum_{\substack{i+j+k=n \\ 0 \le i,j,k \le n}} \mathbf{P}_{ijk} B_{ijk}^n(u, v, w), \quad u, v, w \ge 0, \quad u + v + w = 1 \tag{2}$$

where $\{B_{ijk}^n(u, v, w) = \frac{n!}{i!j!k!} u^i v^j w^k \mid i + j + k = n\}$ are the Bernstein basis functions defined on a 2D simplex, i.e., a triangle. Its control points are $\{\mathbf{P}_{ijk} \mid i + j + k = n\}$, which can be efficiently computed via polynomial interpolations (Feng and Peng, 2000).

## 4. Smooth FFD with sharp feature awareness

The input of the proposed method is a polygonal mesh with vertex, vertex normal and face information. Each vertex has one or more normals. A vertex with only one normal is called a “smooth vertex”, and a vertex with multiple normals is called a “sharp vertex”. An edge with two smooth end vertices is called a “smooth edge”, and an edge with at least one sharp end vertex is called a “sharp edge”.

### 4.1. Fitting deformed normal field as triangular Bézier patches

As described above, the previous accurate FFD methods (Cui and Feng, 2013, 2014) generate unsmooth shading effect due to unsmooth deformed geometry, as shown in Fig. 1(d). To achieve a smooth shading result, the deformed geometry should be equipped with a $C^0$ and discontinuous normal field at a smooth edge and sharp edge, respectively, in the accurate FFD framework.

The normal field generation is formulated as a constrained fitting problem, where the fitted triangular Bézier patch should pass the constraint normals along the boundary curves. Such constrained problems are common in spline modeling, for example, Liu et al. (2014) use cubic Bézier spline constrained fitting to calculate deformed cubic Bézier splines. As illustrated in Fig. 2, the solid normals denote the constraint normals, whereas the dashed normals denote the fitting normals on the original object. If the degree of the adopted triangular Bézier patch is $k$, the numbers of constraint and fitting normals are $3k$ and $m = (n + 1)(n + 2)/2$, respectively, where $n = n_u + n_v + n_w$. In this paper, the degree of the normal triangular Bézier patch is 3, which provides flexibility and feasibility for practical applications. All sampled normals on the deformed object can be obtained by deforming the sampled normals from the linear normal field defined on the triangle (Gain and Dodgson, 1999), which can be formulated as follows:

$$\mathbf{N}_{\bar{X}} = (\mathbf{J} \cdot \bar{\mathbf{J}}^*)^{\mathrm{T}} \mathbf{N}_X \tag{3}$$

where $\mathbf{N}_X$ is the original normal, $\mathbf{N}_{\bar{X}}$ is the deformed normal, and $\mathbf{J}$ is the Jacobian of the embedding function, $\bar{\mathbf{J}}$ is the Jacobian of the deformation function $\mathbf{R}(u, v, w)$ in Eq. (1), and $\bar{\mathbf{J}}^*$ is its adjoint matrix.

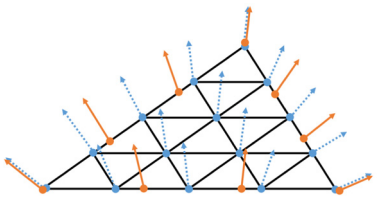


**Fig. 2.** Illustration of the fitting and constraint normals sampled on the triangle.

The above constrained normal field fitting can be formulated as follows. First, the $m$ fitting normals can be expressed as

$$\begin{pmatrix} B^3_{300}(U_1) & B^3_{210}(U_1) & \cdots & B^3_{003}(U_1) \\ B^3_{300}(U_2) & B^3_{210}(U_2) & \cdots & B^3_{003}(U_2) \\ \vdots & \vdots & \ddots & \vdots \\ B^3_{300}(U_m) & B^3_{210}(U_m) & \cdots & B^3_{003}(U_m) \end{pmatrix} \begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_9 \end{pmatrix} = \begin{pmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \vdots \\ \mathbf{Q}_m \end{pmatrix} \tag{4}$$

where $\{U_k = (u_k, v_k, w_k)\}_{k=1}^m$ are the barycentric parameters of uniformly sampling normals; $\{\mathbf{P}_k\}_{k=0}^9$ are the control points of the cubic triangular Bézier patch for the normal field; and $\{\mathbf{Q}_k\}_{k=1}^m$ are the deformed sampling normals corresponding to $\{U_k = (u_k, v_k, w_k)\}_{k=1}^m$. Eq. (4) can be rewritten as

$$\mathbf{MP} = \mathbf{Q} \tag{5}$$

Then, the $3k(k = 3)$ constraint normals can be expressed as:

$$\begin{pmatrix} B^3_{300}(U_{s_1}) & B^3_{210}(U_{s_1}) & \cdots & B^3_{003}(U_{s_1}) \\ B^3_{300}(U_{s_2}) & B^3_{210}(U_{s_2}) & \cdots & B^3_{003}(U_{s_2}) \\ \vdots & \vdots & \ddots & \vdots \\ B^3_{300}(U_{s_9}) & B^3_{210}(U_{s_9}) & \cdots & B^3_{003}(U_{s_9}) \end{pmatrix} \begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_9 \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{Q}}_1 \\ \bar{\mathbf{Q}}_2 \\ \vdots \\ \bar{\mathbf{Q}}_9 \end{pmatrix} \tag{6}$$

where $\{U_{s_k} = (u_{s_k}, v_{s_k}, w_{s_k})\}_{k=1}^9$ are the barycentric parameters of the constraint normals, which are uniformly distributed along the boundaries of the triangular domain, and $\{\bar{\mathbf{Q}}_\mathbf{k}\}_{k=1}^9$ are the deformed constraint normals. Eq. (6) can be rewritten as

$$\bar{\mathbf{M}}\mathbf{P} = \bar{\mathbf{Q}} \tag{7}$$

The constraint normal fitting is to minimize $(\mathbf{MP} - \mathbf{Q})^{\mathrm{T}}(\mathbf{MP} - \mathbf{Q})$ subject to the constraint $\bar{\mathbf{M}}\mathbf{P} = \bar{\mathbf{Q}}$. Using the method of Lagrange multipliers, we obtain

$$\begin{pmatrix} \mathbf{M}^{\mathrm{T}}\mathbf{M} & \bar{\mathbf{M}}^{\mathrm{T}} \\ \bar{\mathbf{M}} & \mathbf{O} \end{pmatrix} \begin{pmatrix} \mathbf{P} \\ \mathbf{\Lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{M}^{\mathrm{T}} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{Q} \\ \bar{\mathbf{Q}} \end{pmatrix} \tag{8}$$

where $\mathbf{\Lambda}$ is the vector of 9 Lagrange multipliers, $\mathbf{O}$ is a zero matrix, and $\mathbf{I}$ is an identity matrix.

All the triangular Bézier patches of the normal field are of degree 3. Thus, Eq. (8) can be combined for all normal patches as follows:

$$\begin{pmatrix} \mathbf{M}^{\mathrm{T}}\mathbf{M} & \bar{\mathbf{M}}^{\mathrm{T}} \\ \bar{\mathbf{M}} & \mathbf{O} \end{pmatrix} \begin{pmatrix} \mathbb{P}^N \\ \Lambda^N \end{pmatrix} = \begin{pmatrix} \mathbf{M}^{\mathrm{T}} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbb{Q}^N \\ \bar{\mathbb{Q}}^N \end{pmatrix} \tag{9}$$

$\mathbb{P}^N$ is the control points of all cubic triangular Bézier surfaces of normal field, $\Lambda^N$ is all of the Lagrange multipliers, and $\mathbb{Q}^N$ and $\bar{\mathbb{Q}}^N$ are the sampled fitting normals and sampled constraint normals on all of the normal patches, respectively. The analytical solution to Eq. (9) is:

$$\begin{pmatrix} \mathbb{P}^N \\ \Lambda^N \end{pmatrix} = \begin{pmatrix} \mathbf{M}^{\mathrm{T}}\mathbf{M} & \bar{\mathbf{M}}^{\mathrm{T}} \\ \bar{\mathbf{M}} & \mathbf{O} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{M}^{\mathrm{T}} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbb{Q}^N \\ \bar{\mathbb{Q}}^N \end{pmatrix} \tag{10}$$

Due to the fixed sampling parameters, the product of the first two matrices on the right of Eq. (10) is independent of the specific model. Thus they can be pre-computed and loaded if necessary. Furthermore, the last 9 rows of the matrix product are about the solutions of Lagrange multipliers and can be ignored. As a result, the control points of all cubic normal patches can be denoted as:

$$\mathbb{P}^N = \mathbf{M}_r \begin{pmatrix} \mathbb{Q}^N \\ \bar{\mathbb{Q}}^N \end{pmatrix} \tag{11}$$

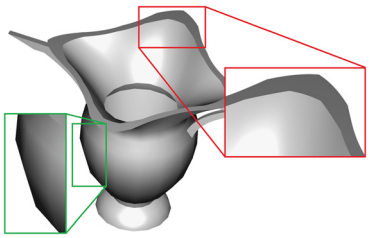


**Fig. 3.** Shading result using constrained fitting normal field of the deformed model in Fig. 1(d) and its geometry artifacts.

where $\mathbf{M}_r$ are the first 10 rows of

$$\begin{pmatrix} \mathbf{M}^{\mathrm{T}}\mathbf{M} & \bar{\mathbf{M}}^{\mathrm{T}} \\ \bar{\mathbf{M}} & \mathbf{O} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{M}^{\mathrm{T}} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{pmatrix}$$

In this manner, we can obtain the continuous normal fields across the deformed model. Fig. 3 shows the shading result of Fig. 1(d).

### 4.2. Improvement of the deformed geometry

The shading in Fig. 3 is smooth except for the geometry artifacts, i.e., the unsmooth silhouettes and patch boundaries. Various solutions can be used to alleviate the artifacts for the Phong shading of polygonal objects (Vlachos et al., 2001; Boubekeur and Alexa, 2008). A method inspired by the PN-triangles (Vlachos et al., 2001) is proposed to improve the silhouettes, patch boundaries, and keep the sharp edges.

In the PN-triangle algorithm (Vlachos et al., 2001), a cubic triangular Bézier patch is adopted to replace the corresponding triangle to obtain a smooth geometry appearance in Phong shading. In the accurate FFD, the deformed object is represented in terms of triangular Bézier patches (Feng et al., 1998; Feng and Peng, 2000; Cui and Feng, 2014), whose degree is the degree of the B-spline volume, and is typically higher than 3. High degree will result in high computational costs. Furthermore, the corresponding geometry adjustment becomes complex because there are many more control points. According to our experiments, the cubic triangular Bézier patch is feasible and sufficiently flexible to approximate the geometry of the accurate FFD result. Similar to the constrained normal field fitting approach in Section 4.1, the geometry can be approximated by cubic triangular Bézier patches using the following equation:

$$\mathbb{P}^V = \mathbf{M}_r \begin{pmatrix} \mathbb{Q}^V \\ \bar{\mathbb{Q}}^V \end{pmatrix} \tag{12}$$

where $\mathbb{P}^V$ is the control points of all cubic triangular Bézier patches of the deformed object. $\mathbb{Q}^V$ and $\bar{\mathbb{Q}}^V$ are those sampled fitting points and sampled constraint points on all cubic triangular Bézier patches, respectively. The sampling strategy is the same as that of the normal field in Section 4.1.

Using the above method, a $C^0$ continuous geometry and a $C^0$/discontinuous normal field across the smooth and sharp edges are obtained. Next, the geometry is adjusted to obtain a smooth geometry appearance along smooth edges and to preserve shape features along sharp edges.

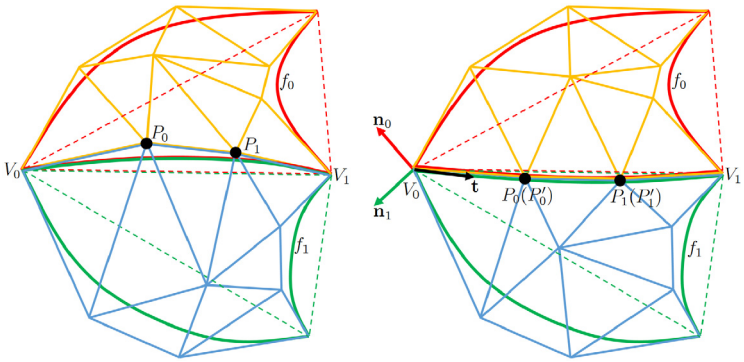
The adjustment of triangular Bézier patches sharing a smooth edge is similar to the PN-triangle method. As shown in Fig. 4(a), the PN-triangle approach projects $(2V_0 + V_1)/3$ into the tangent plane at $V_0$ (Vlachos et al., 2001), whereas the proposed method projects the corresponding edge control point $P_0$ onto the tangent plane at $V_0$. Thus, a visually plausible smooth geometry ($C^0$ in fact) across the smooth edge is obtained. In Fig. 4(a), red and green dashed lines indicate two original triangles, and red and green solid curve nets indicate cubic triangular Bézier patches, which are the adjustment results.

Similarly, the geometry adjustment of patches sharing a sharp edge is shown in Fig. 4(b). The two corresponding control points $P_0$ and $P_0'$ in the neighboring triangular Bézier patch $f_0$ and $f_1$ sharing a sharp edge are projected to the vector $\mathbf{t} = \mathbf{n}_0 \times \mathbf{n}_1$, where $\mathbf{n}_0$ and $\mathbf{n}_1$ are two distinct normals at the same vertex $V_0$. This method yields a $C^0$ continuous geometry across a sharp edge, which can preserve the sharp feature along the sharp edge.

After the above adjustment, visually plausible smooth geometries, such as silhouettes and edges, will be obtained. The sharp features along the sharp edges are also preserved. An example is shown in Fig. 5. It is worth noting that the generated geometry is different than the original discrete polygonal mesh in the rest-pose because of the higher-order interpretation of the polygonal mesh.

### 4.3. Consideration of knot box clipping

If there is only one knot box, the above deformed geometry and normal field methods are suitable. However, for the case of general B-spline volumes, the input polygonal object will be clipped against the knot boxes, and thus, all of the sub-triangles will lie in one knot box. If the above adjustment method is directly applied to the subdivided polygonal object,




(a) Smooth edge (b) Sharp edge

**Fig. 4.** Adjustment of control points of the deformed geometry. (For interpretation of the references to colour in this figure, the reader is referred to the web version of this article.)

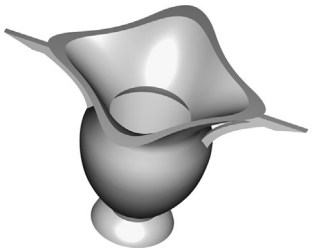
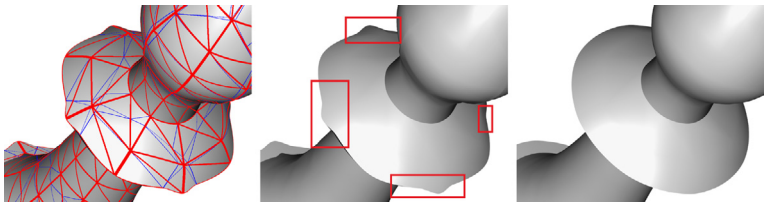


**Fig. 5.** Shading result after adjusting the deformed geometry in Fig. 3.



(a) Shading result with boundary curves (b) Shading result (c) Visually plausible smooth and fair deformed geometry

**Fig. 6.** Unfairness near the clipping points in the deformed geometry and the improvement. (For interpretation of the references to colour in this figure, the reader is referred to the web version of this article.)

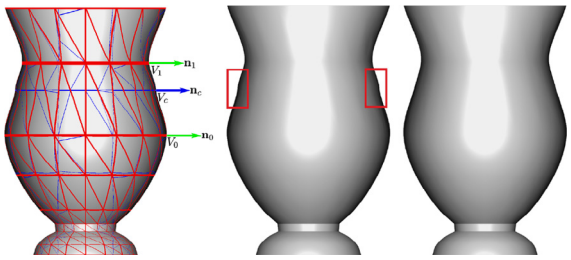
the deformed geometry does not appear fair near the knots. An example is shown in Fig. 6. Fig. 6(a) is the deformed geometry by a B-spline volume with multiple knot boxes. Both the geometry and shading are smooth. However, the geometry appearance is not fair. The bold curves are the deformed curves corresponding to the original triangles on the model; the thin curves are the deformed curves of the sub-triangles resulting from knot box clipping and subsequent triangulation. The unfair parts are indicated in the red rectangles in Fig. 6(b). Since the sub-triangles from one original triangle are co-planar, the constrained fittings for these deformed sub-triangles will lead to unfairness.

*4.3.1. Adjustment of the deformed geometry clipped against knot boxes*

There is a heuristic solution to this unfairness problem. For each triangle in the original object, the corresponding PN-triangle is generated first. If the triangle is subdivided against the knot boxes, the clipped vertices are moved to the corresponding positions on the PN-triangle. Then, we apply the constrained fitting approach to the modified sub-triangles. As a result, we can obtain a visually plausible smooth and fair deformed geometry. The improved result of the example in Fig. 6(a) is shown in Fig. 6(c).

*4.3.2. Normal adjustment at the clipped vertex*

Intuitively, the normal at the clipped vertex can be calculated via barycentric coordinate interpolation. But it would lead to an unfairness problem in the normal field as shown in Fig. 7(a) and 7(b). The normals of $V_0$ and $V_1$ are $\mathbf{n}_0$ and $\mathbf{n}_1$, respectively. Both of the normals are horizontally rightward. Thus, the interpolated normal $\mathbf{n}_c$ of the subdivided vertex $V_c$




(a) Shading result with boundary curves (b) Shading result (c) Improved result of 7(a)

**Fig. 7.** The normal at the clipped vertex will lead to bumps and the related solution.

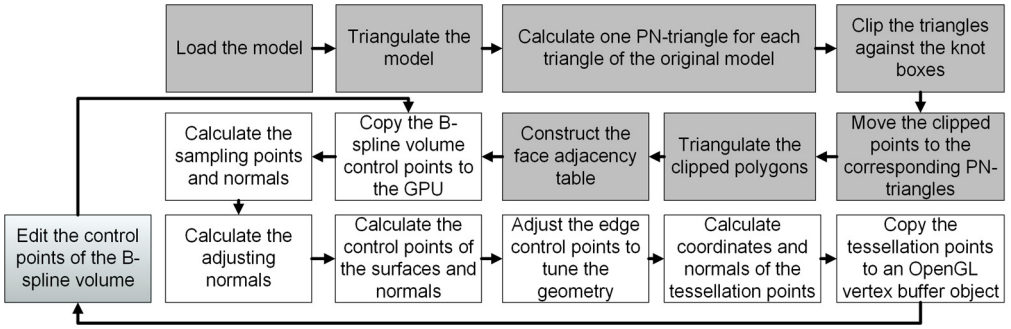



**Fig. 8.** Flowchart of the proposed algorithm.

is also horizontally rightward. There is no artifact when using the normal $\mathbf{n}_c$ to determine the normal field for rendering. However, if it is used to adjust the control points for smoothing the deformed geometry, i.e., the edge control points adjacent to $V_c$, it will lead to abnormal bumps on the deformed geometry.

The solution to this problem is to define a reasonable normal at each clipped vertex. The normal at the subdivided vertex can take the corresponding normal $\mathbf{n}^*$ on the quadratic triangular Bézier normal field of the PN-triangle (Vlachos et al., 2001). After using $\mathbf{n}^*$ in the geometry adjustment scheme, the adjusted geometry for the example in Fig. 7(a) is shown in Fig. 7(c), where the abnormal bumps disappear. The adjusted normal field is only for smoothing the deformed geometry in Section 4.2, where some edge control points are projected onto the planes or tangent lines determined by the normals. The normal field for rendering the deformed object is the normal field constructed in Section 4.1, where the normal at the clipped vertex is obtained via barycentric coordinate interpolation.

## 5. Parallel implementation on the GPU

The proposed algorithm contains a CPU execution component and a GPU execution component. As a data- and computing-intensive algorithm, most of the computing overhead is implemented on the GPU. Because the algorithm is local, it can fully utilize the parallel computing power of the GPU. The flowchart of the proposed algorithm is shown in Fig. 8. The gray boxes indicate the steps that are executed on the CPU. The white boxes are the steps executed on the GPU and are the core of the proposed algorithm. NVIDIA CUDA is used to perform the GPU computations.

### 5.1. Parallel computing sampling normals and points

The first step in the proposed algorithm is to sample normals and points on the deformed object. More specifically, we must calculate all elements in $\mathbb{Q}^N$ and $\bar{\mathbb{Q}}^N$ in Eq. (11) and all elements in $\mathbb{Q}^V$ and $\bar{\mathbb{Q}}^V$ in Eq. (12), where $\mathbb{Q}^N$ and $\bar{\mathbb{Q}}^N$ are the fitting and constraint normals respectively, and $\mathbb{Q}^V$ and $\bar{\mathbb{Q}}^V$ are the fitting and constraint points respectively.

$\mathbb{Q}^V$ and $\bar{\mathbb{Q}}^V$ can be directly obtained via B-spline volume evaluations. Two methods can be used for this: the de Boor Cox algorithm and the matrix approach (Farin, 2002). We adopted the latter method because it is less computationally burdensome and is more suitable for GPU implementation compared to the former method. The deformation of a sampled normal in $\mathbb{Q}^N$ and $\bar{\mathbb{Q}}^N$ can be evaluated by Eq. (3).

In our implementation, one CUDA thread is used to calculate one sampling point and one sampling normal. Because each cubic patch contains the same number of sampling points, the task arrangement is straightforward.

### 5.2. Parallel computing constrained fitting patches for the geometry and normal field

Having obtained $\mathbb{Q}^V$, $\bar{\mathbb{Q}}^V$, $\mathbb{Q}^N$ and $\bar{\mathbb{Q}}^N$, the next step is to calculate the control points of the normal patches and geometry patches, i.e., the matrix $\mathbb{P}^N$ in Eq. (11) and the matrix $\mathbb{P}^V$ in Eq. (12), respectively. This can be accomplished by matrix multiplication. cuBLAS, which is a library of optimized implementations of BLAS on a GPU, is adopted for these matrix multiplications.[1] The elements of $\mathbb{P}^N$, $\mathbb{Q}^N$, $\bar{\mathbb{Q}}^N$, $\mathbb{P}^V$, $\mathbb{Q}^V$ and $\bar{\mathbb{Q}}^V$ are 3-dimensional points, which are not supported by cuBLAS. Thus, Eqs. (11) and (12) should be rewritten and combined as the following feasible form to fully explore the parallel computing power of the GPU:

$$\begin{pmatrix} \mathbb{P}_x^V & \mathbb{P}_y^V & \mathbb{P}_z^V & \mathbb{P}_x^N & \mathbb{P}_y^N & \mathbb{P}_z^N \end{pmatrix} = \mathbf{M}_r \begin{pmatrix} \mathbb{Q}_x^V & \mathbb{Q}_y^V & \mathbb{Q}_z^V & \mathbb{Q}_x^N & \mathbb{Q}_y^N & \mathbb{Q}_z^N \\ \bar{\mathbb{Q}}_x^V & \bar{\mathbb{Q}}_y^V & \bar{\mathbb{Q}}_z^V & \bar{\mathbb{Q}}_x^N & \bar{\mathbb{Q}}_y^N & \bar{\mathbb{Q}}_z^N \end{pmatrix} \tag{13}$$

### 5.3. Calculating the adjusted normals for the subdivided geometry

As described in Section 4.3.2, each sub-triangle has three adjustment normals, one for each vertex. These vertex normals are taken from the corresponding points on the normal field of the PN-triangle of the original triangle. Then, they undergo deformation via Eq. (3) for the silhouette adjustment. Here we use one CUDA thread to handle one normal computation.

### 5.4. Adjusting the edge control points in parallel on the GPU

The first task in this step is finding 1-ring neighbors of the current triangle. A face adjacency table can accomplish this task. The topological connectivity of the model remains unchanged during the deformation. Thus, the face adjacency table can be constructed once in the pre-processing step and copied from the main memory to the GPU memory. The face adjacency table is maintained in GPU via a hash table.

The topological connectivity of the model will be modified after the subdivision against the knot boxes (Step 4 in Fig. 8) and the triangulation of sub-polygons (Step 6 in Fig. 8). Thus, the face adjacency table must be reconstructed after subdivision and triangulation (Step 7 in Fig. 8), copied to the GPU memory, and used by the CUDA kernel.

After obtaining the adjusted normals, control points of the geometry patches, and the face adjacency table, we can adjust the edge control points to tone the geometry using the method in Section 4.2 in parallel. Here, one CUDA thread is used to adjust one patch.

### 5.5. GPU tessellations of the geometry and normal surfaces

The triangular Bézier patches for the geometry and normal field have now been obtained. However, current GPUs do not support the rendering of triangular Bézier patches directly. Thus, we must tessellate the patches into triangles for rendering. Here, we use a uniform tessellation method in Cui and Feng (2014) to calculate the tessellated points and normals to fully explore the tremendous computing power of the GPU. Similar to Cui and Feng (2014), we combine the point and normal evaluations as in Eq. (13) for efficiency:

$$\begin{pmatrix} \mathbb{R}_x^V & \mathbb{R}_y^V & \mathbb{R}_z^V & \mathbb{R}_x^N & \mathbb{R}_y^N & \mathbb{R}_z^N \end{pmatrix} = \mathbf{B}_q \begin{pmatrix} \mathbb{P}_x^V & \mathbb{P}_y^V & \mathbb{P}_z^V & \mathbb{P}_x^N & \mathbb{P}_y^N & \mathbb{P}_z^N \end{pmatrix} \tag{14}$$

where $\mathbf{B}_q$ is the evaluation matrix of cubic triangular Bézier patches. The result can be rendered using an OpenGL Vertex Buffer Object (VBO).

## 6. Implementation results and comparison

The proposed method is implemented on a PC with an Intel Core i5 760 CPU@2.8GHz, 4 GB of main memory and an NVIDIA GeForce GTX 465 GPU. The operating system is Arch Linux x86_64. The CPU and GPU components of our method are written using C++ and CUDA, respectively.

We will compare the proposed smooth FFD with a uniform upsampling method and Cui and Feng (2013, 2014) from the aspects of rendering result, efficiency and approximation errors, tessellation in the follows.

### 6.1. Comparison of the rendering results

The deformation results are shown in Figs. 9–12. Each triangular Bézier patch is tessellated into 100 triangles. There are 4 sub-figures in each of the 4 examples: (a) is the original model; (b) is the result of accurate FFD (Cui and Feng, 2013, 2014); (c) is the result of smooth FFD; (d) is the textured shading effect of (c).

These figures illustrate that the results obtained with smooth FFD are superior to those obtained with the other methods. The highlight is smooth because the normals that we use are independent of the model geometry. The smooth geometry and sharp features benefit from our geometry and normal adjustment method.

[1] NVIDIA, cuBLAS, nvidia developer zone. https://developer.nvidia.com/cublas.

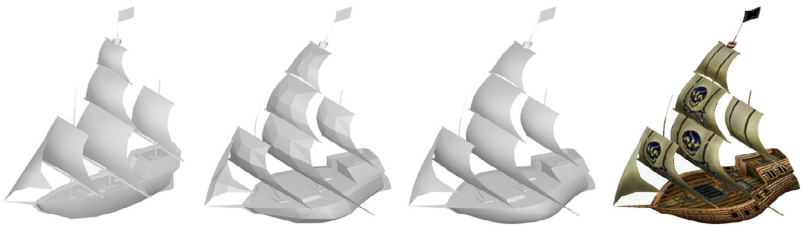



**Fig. 9.** Deformation of the Ship model by a $3 \times 3 \times 3$ B-spline volume with $5 \times 8 \times 5$ control points.

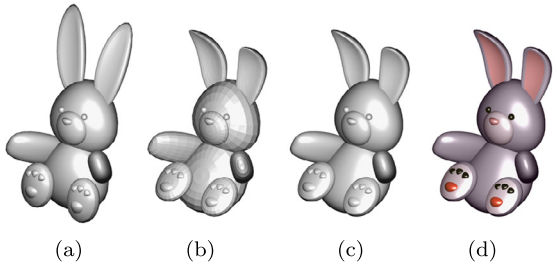



**Fig. 10.** Deformation of the Doll model by a $2 \times 2 \times 2$ B-spline volume with $5 \times 5 \times 5$ control points.

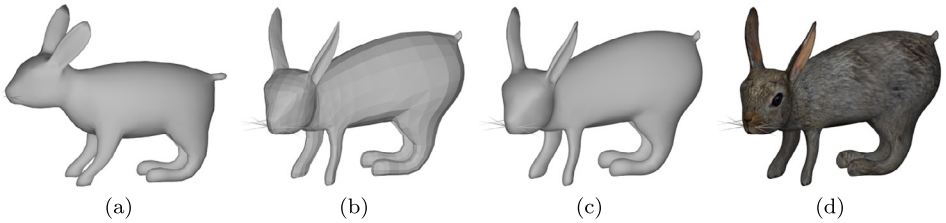



**Fig. 11.** Deformation of the Rabbit model by a $2 \times 2 \times 2$ B-spline volume with $5 \times 8 \times 5$ control points.

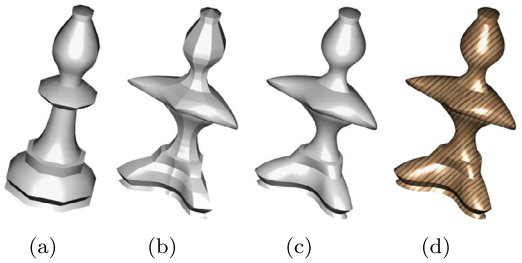



**Fig. 12.** Deformation of the Chess model by a $2 \times 2 \times 2$ B-spline volume with $5 \times 9 \times 5$ control points.

### *6.2. Comparison of efficiencies*

The efficiencies obtained with smooth FFD and Cui and Feng (2013, 2014) are compared in Table 1. The model we use is shown in Fig. 13(a); this model has 46,742 faces. As in Section 6.1, each triangular Bézier patch is tessellated into 100 sub-triangles. The degree of the B-spline volume is $2 \times 2 \times 2$, with $5 \times 5 \times 5$ control points. Table 1 illustrates that the speed of our method is lower than that of Cui and Feng (2014) due to additional geometry adjustment, normal field computation and adjustment, but higher than that of Cui and Feng (2013). The proposed method is sufficiently fast to handle large models in real time.

### *6.3. Approximation error tests*

In the proposed smooth FFD, both the geometry and normal are obtained approximately via constrained fitting. Thus, there are approximation errors in the geometry and normal compared with the accurate FFD. Since the smooth parts of a

**Table 1**
Comparison of the efficiencies of our method, (Cui and Feng, 2013, 2014) (ms).

| Step | Our method | (Cui and Feng, 2013) | (Cui and Feng, 2014) |
|---|---|---|---|
| Copy control points to the GPU | 0.009 | 0.009 | 0.009 |
| Calculate the sampling points | 8.469 | – | 6.701 |
| Calculate the constraint points | 2.879 | – | – |
| Calculate the control points | 3.971 | 11.608 | – |
| Calculate the adjusting normals | 1.162 | – | – |
| Adjust the control points | 2.003 | – | – |
| Calculate the tessellation points | 4.861 | 24.33 | 7.507 |
| Copy the results to the VBO | 3.092 | – | 3.301 |
| Render | 10.816 | 10.709 | 10.773 |
| Total | 37.262 | 46.656 | 28.291 |

(a) Shading of smooth FFD (b) Shading of UUS algorithm

**Fig. 13.** Shading results for smooth FFD and UUS algorithm.

polygonal object can be regarded as a piecewise linear approximation of a potentially smooth shape, which is unknown in general, thus it is difficult to evaluate the approximation errors.

Here, we use a Cube model and a Utah teapot model consists of 36 bicubic Bézier patches to test the approximation error, because the geometry and normal of both the input objects are defined accurately. Both of the two models are normalized into $[-1, 1]^3$. The deformations of the Cube model by Cui and Feng (2013, 2014) are accurate for both the geometry and normal. The accurate deformation result of Utah teapot is obtained via original FFD of uniformly sampled points and normals. The input of smooth FFD of the Utah teapot is a mesh generated via de Casteljau subdivisions. The error test results are shown in Fig. 14, where the first and second columns are the Cube model deformed by a $2 \times 2 \times 2$ and $3 \times 3 \times 3$ B-spline volume, respectively. The third column is the Utah teapot deformed by a $2 \times 2 \times 2$ B-spline volume. For clarity, the shading differences are inverted and enhanced by 10 times, as shown in Fig. 14(d).

The statistics of geometry error, normal error and volume error are given in Tables 2–4, respectively. The geometry error is the Euclidean distances between the corresponding vertices, and the normal error is the angles between the corresponding normals. The volume error of Utah teapot is not listed here because it's not a closed model.

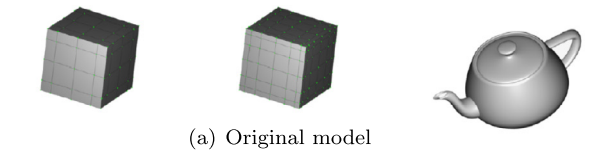
According to Fig. 14 and Tables 2–4, the proposed smooth FFD can generate good approximations of accurate FFD from the geometry, normal, and volume aspects. Among them, only maximum normal error of the Utah teapot model is a little bit large. The maximum errors only occur at the spout end and lip which are high curvature parts. Here the input mesh of the Utah teapot is obtained via uniform sampling approach, thus it is not a good polygonal approximation to its original smooth model. It leads to the maximum normal approximation error as above. That is to say the maximum error comes from the polygonal mesh approximation to the smooth object, which is out of scope of the paper.

### *6.4. Comparison of smooth FFD and a uniformly upsampling method*

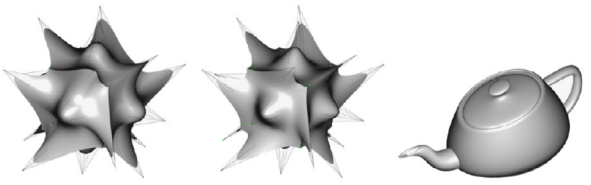
Similar deformation results for polygonal objects can be obtained using a modified GPU-based Uniform UpSampling method (similar to the UUS method in Cui and Feng (2013)): step 1: using PN-triangles to replace the triangles in the original object; step 2: uniformly upsampling the PN-triangles for both the geometry and normal patches; step 3: deform all sampled points and normals to obtain the deformation result. Here, the Snail model in Fig. 13(a) is adopted again. The deformation results are similar because the Snail model consists of lots of small triangles. The runtime statistics are collected in Table 5. The modified GPU-based UUS method is not as efficient as smooth FFD when the number of tessellated triangles is relatively large. The rendering times are not considered because they are the same in both algorithms. This comparison is consistent with the comparison in Cui and Feng (2013). However, if the triangles of the model is relatively large, we can find that the shading of the UUS algorithm is not as good as the proposed algorithm because the former uses a quadratic normal field on each PN-triangle (Vlachos et al., 2001), as shown in Fig. 15.

### *6.5. Comparison of smooth FFD and an adaptive tessellation shader method*

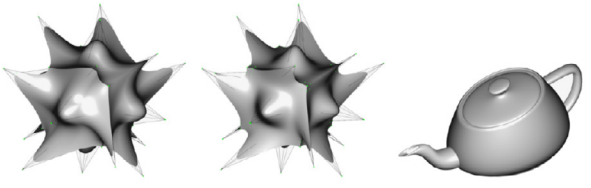
Tessellation shader which has been added to OpenGL since version 4.0 is a suitable tool to tessellate the generated cubic triangular Bézier surface patches. So, we also implemented an adaptive tessellation of cubic triangular patches using



(a) Original model

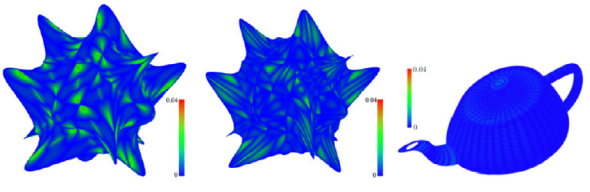


(b) Smooth FFD results of (a)

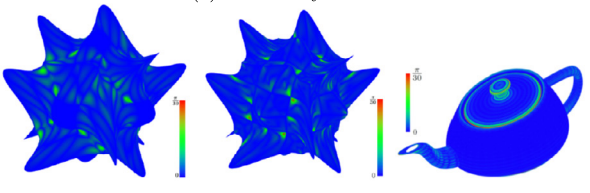


(c) Accurate FFD results of (a)

(d) Shading differences between (b) and (c)



(e) Geometry errors



(f) Normal errors

**Fig. 14.** Error testing.

**Table 2**
Geometry approximation errors.

| | Average error | Maximum error |
|---|---|---|
| Column 1 in Fig. 14 | 0.00448184 | 0.0274765 |
| Column 2 in Fig. 14 | 0.00297672 | 0.0261029 |
| Column 3 in Fig. 14 | 0.000906093 | 0.00710446 |

tessellation shader. The tessellation factors of three edges of a patch are proportional to their lengths. This adaptive approach shares the same steps with smooth FFD until the control points of all cubic triangular Bézier patches are obtained. After that, the adaptive tessellation shader method is performed to render these patches directly, while the smooth FFD will calculate the tessellation points by cuBLAS first, then copy the result to OpenGL VBO, and render the sub-triangles finally.

**Table 3**
Normal approximation errors.

| | Average error | Maximum error |
|---|---|---|
| Column 1 in Fig. 14 | 0.394331° | 4.17587° |
| Column 2 in Fig. 14 | 0.387851° | 5.23632° |
| Column 3 in Fig. 14 | 0.566609° | 23.3726° |

**Table 4**
Volume approximation errors.

| | Degree of the B-spline volume | Model volume after smooth FFD($v'$) | Accurate volume($v$) | $\lvert v' - v \rvert / v$ |
|---|---|---|---|---|
| cube | $2 \times 2 \times 2$ | 16.641 | 16.6449 | 0.023107% |
| | $3 \times 3 \times 3$ | 11.7615 | 11.7651 | 0.0304927% |

**Table 5**
Runtime comparison between smooth FFD and modified GPU-based UUS (ms).

| Number of sub-triangles in each patch | Smooth FFD | UUS | Smooth FFD/UUS |
|---|---|---|---|
| 100 | 26.418 | 18.120 | 1.457947 |
| 121 | 27.251 | 21.357 | 1.275975 |
| 144 | 28.954 | 24.859 | 1.164729 |
| 169 | 30.894 | 30.383 | 1.016819 |
| 196 | 31.786 | 34.636 | 0.917716 |
| 225 | 32.773 | 39.224 | 0.835534 |
| 256 | 36.183 | 44.120 | 0.820104 |

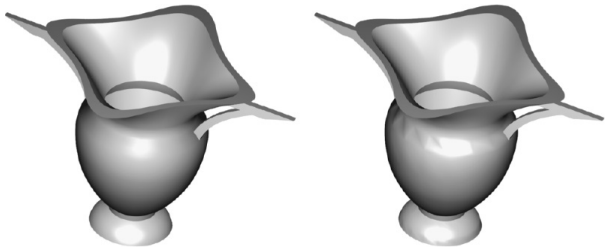
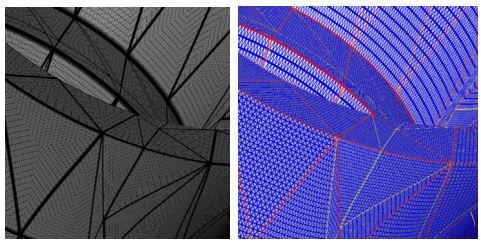



**Fig. 15.** Shading results for smooth FFD (a) and UUS algorithm (b). Each Bézier patch is tessellated into 100 sub-triangles.



(a) Adaptive tessellation (b) Smooth FFD

**Fig. 16.** Comparison of tessellation granularity.

The Vase model shown in Fig. 1 is adopted as the test example. The tessellation granularities of two methods are shown in Fig. 16. In the tessellation shader method, the longest edge are tessellated into 27 segments. For fair comparison, in the smooth FFD, all the edges are tessellated uniformly into 27 segments. The time comparison of them is shown in Table 6. As we can see, the efficiency of the tessellation shader method is not as efficient as that of the smooth FFD. The main reason is that the smooth FFD is a perfect SIMD task, it can make full use of the GPU streaming processors. Of course, if the object is composed of tiny, slim and large triangles, it is possible that the adaptive tessellation approach may perform better than the uniform tessellation.

**Table 6**
Comparison of the times for the smooth FFD and adaptive tessellation shader method (ms).

| | Smooth FFD | Tess shader method |
|---|---|---|
| Calculate the tess points | 0.456 | – |
| Copy results to the VBO | 0.576 | – |
| Render | 3.031 | 6.082 |
| Total | 4.063 | 6.082 |

## 7. Conclusion and future work

In this paper, we proposed a GPU-based smooth FFD with sharp features awareness that addresses the unsmoothness of the normal field and the geometry artifact problems in the framework of accurate FFD. The algorithm can produce a high-quality deformation result. It is a highly parallelizable GPU algorithm and is able to deform a relatively large-scale model in real time. The algorithm is intuitive and can be implemented easily. It can handle relatively coarse meshes and generate smooth deformation results.

The approach can still be improved in several aspects. First, the uniform tessellation of the cubic triangular Bézier patches will generate many unnecessary small triangles. An efficient adaptive tessellation algorithm via GPGPU is an alternative to our method. Second, the approximation error of the smooth FFD for polygonal object is worth to be analyzed in theory. A feasible error bound is useful to guide the discretization of a smooth object, which is essential for generating high-quality deformation result.

## Acknowledgements

The authors would like to thank the anonymous reviewers who gave valuable suggestions to improve the quality of the paper. This work was supported by the National Natural Science Foundation of China under Grant Nos. 61170138 and 61472349.

## References

Sederberg, T.W., Parry, S.R., 1986. Free-form deformation of solid geometric models. SIGGRAPH Comput. Graph. 20 (4), 151–160.
Feng, J., Heng, P.-A., Wong, T.-T., 1998. Accurate B-spline free-form deformation of polygonal objects. J. Graph. Tools 3 (3), 11–27.
Feng, J., Nishita, T., Jin, X., Peng, Q., 2002. B-spline free-form deformation of polygonal object as trimmed Bézier surfaces. Vis. Comput. 18, 493–510. http://dx.doi.org/10.1007/s00371-002-0171-1.
Feng, J., Peng, Q., 2000. Accelerating accurate B-spline free-form deformation of polygonal objects. J. Graph. Tools 5 (1), 1–8.
DeRose, T.D., 1988. Composing Bézier simplexes. ACM Trans. Graph. 7 (3), 198–221.
DeRose, T.D., Goldman, R.N., Hagen, H., Mann, S., 1993. Functional composition algorithms via blossoming. ACM Trans. Graph. 12 (2), 113–135.
Vlachos, A., Peters, J., Boyd, C., Mitchell, J.L., 2001. Curved PN triangles. In: Proceedings of the 2001 Symposium on Interactive 3D Graphics. I3D '01. ACM, New York, NY, USA, pp. 159–166.
Boubekeur, T., Alexa, M., 2008. Phong tessellation. ACM Trans. Graph. 27 (5), 141:1–141:5. http://dx.doi.org/10.1145/1409060.1409094.
Coquillart, S., 1990. Extended free-form deformation: a sculpturing tool for 3D geometric modeling. SIGGRAPH Comput. Graph. 24 (4), 187–196. http://dx.doi.org/10.1145/97880.97900.
Hui, K., 2002. Free-form design using axial curve-pairs. Comput. Aided Des. 34 (8), 583–595. http://dx.doi.org/10.1016/S0010-4485(01)00132-4.
MacCracken, R., Joy, K.I., 1996. Free-form deformations with lattices of arbitrary topology. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '96. ACM, New York, NY, USA, pp. 181–188.
McDonnell, K.T., Qin, H., Zhao, X., 2007. PB-FFD: a point-based technique for free-form deformation. J. Graph. Tools 12 (3), 25–41. http://cvc.cs.sunysb.edu/Publications/2007/TQ07.
Xu, G., Hui, K.-C., Ge, W.-B., Wang, G.-Z., 2013. Direct manipulation of free-form deformation using curve-pairs. Comput. Aided Des. 45 (3), 605–614. http://dx.doi.org/10.1016/j.cad.2012.09.004.
Gain, J., Bechmann, D., 2008. A survey of spatial deformation from a user-centered perspective. ACM Trans. Graph. 27 (4), 107:1–107:21.
Gain, J.E., Dodgson, N.A., 1999. Adaptive refinement and decimation under free-form deformation. In: Proceedings of the Eurographics UK 17th Annual Conference. Eurographics UK '99, Cambridge, pp. 13–15.
Griessmair, J., Purgathofer, W., 1989. Deformation of solids with trivariate B-splines. In: Hopgood, F.R.A. (Ed.), Proceedings of Eurographics, pp. 137–148.
Parry, S.R., 1986. Free-form deformation in a constructive solid geometry modeling system. Ph.D. thesis. Brigham Young University, Provo, UT, USA. uMI order no. GAX86-16254.
Chua, C., Neumann, U., 2000. Hardware-accelerated free-form deformation. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware. HWWS '00. ACM, New York, NY, USA, pp. 33–39.
Schein, S., Elber, G., 2006. Real-time free-form deformation using programmable hardwares. Int. J. Shape Model. 12, 179–192.
Jung, Y., Graf, H., Behr, J., Kuijper, A., 2011. Mesh deformations in X3D via CUDA with freeform deformation lattices. In: Shumaker, R. (Ed.), Virtual and Mixed Reality – Systems and Applications. In: Lecture Notes in Computer Science, vol. 6774. Springer, Berlin, Heidelberg, pp. 343–351.
Hahmann, S., Bonneau, G.-P., Barbier, S., Elber, G., Hagen, H., 2012. Volume-preserving FFD for programmable graphics hardware. Vis. Comput. 28, 231–245. http://dx.doi.org/10.1007/s00371-011-0608-5.
Cui, Y., Feng, J., 2013. Real-time B-spline free-form deformation via GPU acceleration. Comput. Graph. 37 (1–2), 1–11.
Cui, Y., Feng, J., 2014. Real-time accurate free-form deformation in terms of triangular Bézier surfaces. Appl. Math. J. Chin. Univ. 29 (4), 455–467.
Liu, S., Jacobson, A., Gingold, Y., 2014. Skinning cubic Bézier splines and Catmull–Clark subdivision surfaces. ACM Trans. Graph. 33 (6).
Farin, G., 2002. Curves and Surfaces for CAGD: A Practical Guide, 5th edition. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.