The Visual Computer

B-spline free-form deformation of polygonal object as trimmed Bézier surfaces

Jieqing Feng[1], Tomoyuki Nishita[2], Xiaogang Jin[1], Qunsheng Peng[1]

[1] State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou, 310027, P.R. China
E-mail: {jqfeng, jin,peng}@cad.zju.edu.cn
[2] Department of Complexity Science and Engineering, University of Tokyo, Hongo 7-3-1, Bunkyo, Tokyo 113-0033, Japan
E-mail: nis@is.s.u-tokyo.ac.jp

Published online: November 6, 2002

Free-form deformation is a powerful shape modification tool. How to approximate or compute the real deformation of a polygonal object is still problematic. In this paper, a new solution is proposed for this problem. First, a special initial B-spline volume is defined whose Jacobian is an identity matrix. The accurate deformation is as trimmed tensor-product Bézier surfaces. The description of the trimmed surfaces is consistent with that in the industrial standard, STEP. The degrees of the Bézier surfaces are lower than the theoretical results. Compared with previous algorithms, the proposed algorithm has the advantages of both storage and runtime.

Key words: Free-form deformation – B-spline – Polygons – Bézier surface – Trimmed surface

1 Introduction

Free-form deformation, abbreviated as FFD, was first proposed by Sederberg and Parry [23]. It is an effective tool for geometric shape modification, and it is independent of the representation and topology of object. The FFD can be concisely described as follows. Firstly, the object is frozen into an intermediate flexible space, such that when the shape of the intermediate space is changed, the object will follow its change. The 3D tensor-product Bézier volume was adopted as an appropriate flexible space by Sederberg and Parry. Its initial shape is an orthogonal parallelepiped. Later the FFD was extended to include the B-spline FFD [13], extended FFD [4], rational FFD [16], direct manipulation FFD [15], NURBS FFD [17], continuous FFD [1], arbitrary topological lattice FFD [19], and Dirichlet FFD [20]. It has been applied to geometric modelling [6, 14] and computer animation [3, 5, 16]. Bechmann produced a survey about the various space deformation techniques [2].
The problem of sampling for the polygonal object deformation is unsolved as yet. Theoretically, the deformation should act upon the whole polygonal object. However, this can not be achieved, since the polygonal object can be only represented discretely. Thus, the deformation can only act on the sampling points of the polygonal object. If the number of sampling points is insufficient, aliasing phenomena will occur in the deformation result, and this will degrade the final deformation effect. Heretofore, there were three types of solutions to the problem. They are the uniform subdivision approach, the error-test approach and the accurate FFD approach.

1.1 Uniform subdivision approach

The polygonal object is uniformly subdivided on the midpoint of each edge recursively. This subdivision occurs regardless of the geometric shape of the object, and the stop criterion is decided by the user. All of the meshes in the object will be treated equally, and therefore it is a brute-force solution; but it is the solution that is adopted for most popular 3D computer animation software. The drawback of the solution is obvious: many tiny meshes must be generated if a perfect deformation result is required, and this increases the rendering and storage burden.

1.2 Error-test approach

Parry first proposed an adaptive subdivision algorithm for solving the sampling problem [22]. The

The Visual Computer (2002) 18:493–510
Digital Object Identifier (DOI) 10.1007/s00371-002-0171-1

two triangles that share a common "long side" are halved. The long side for a triangle is defined as follows. Initially, an arbitrary edge of the triangle is labelled, and then, after subdivision, the other two undivided edges are labelled as the long sides for the two newly generated triangles. Parry adopted two criteria to decide the subdivision of a triangle: the size of the triangle as projected on a screen and the triangle's "curvature". Griessmair and Purgathofer proposed another solution for the polygonal object deformation [13]. In their method, the middle point of an edge after deformation is checked to determine whether it is far away from its true position. If the disparity is greater than a predefined threshold, the edge will be halved. Inspired by the existing work on domain discretisation in finite element analysis, Nimscheck proposed an advancing-front meshing algorithm for FFD [21]. Each edge of the initial polygon is recursively subdivided based on either its post-deformation length or the disparity between its endpoint tangents. Then a new vertex is inserted for triangulation, which approximates the ideal deformation result. Besides subdivision, Gain merged the triangles that are approximately coplanar [12]. This method can decrease the number of meshes in the deformed object. All of the four methods described above follow a similar strategy: sampling, testing and subdivision. According to the sampling theory, some singular cases cannot be avoided [12]. These are inappropriate for processing by the above methods. For example, for a wave-shaped curve, if the normals to both its ends are the same, the above methods may detect it falsely.

1.3 Accurate FFD approach

The accurate FFD methods proposed by the present authors can avoid the problems of the error-test approach [10, 11]. Traditionally, the FFD acts on a point, whereas accurate FFD acts on each triangle. The deformation result is a triangular Bézier surface according to the Bernstein polynomial composition [8]. There are two weaknesses in accurate FFD methods. Firstly, a triangular Bézier surface is not directly supported by current modelling systems and rendering pipelines. In general, these support tensor-product Bézier surfaces. Sometimes the rendering procedure can be accelerated by the hardware. To make full use of the graphics hardware ability and make it compatible with the current modelling system, the triangular Bézier surface needs to be tessellated into triangular meshes or converted into three tensor-product Bézier surfaces in advance, which is an additional cost. Another weakness is the degree of the triangular Bézier surface. It is equal to the degree of the B-spline volume. When a triangular Bézier surface is converted into three tensor-product Bézier surfaces, the degree of the resulting surfaces is twice the degree of the triangular Bézier surface. This is the maximal degree according to the results in [7, 8]. For example, when the degree of the B-spline volume in the accurate FFD is $2 \times 2 \times 2$, the degree of the triangular Bézier surface is 6. The degree of the corresponding tensor-product Bézier surface will be 6×6. This high degree means high computational, storage and display costs.

In this paper, a new accurate FFD method is proposed. First, a new definition of the initial B-spline volume is given. It is an end-interpolation B-spline volume, whose Jacobian matrix is an identity matrix. Then, a robust subdivision algorithm for the polygonal object is proposed, which is free of degenerate polygons. The deformation result is trimmed tensor-product Bézier surfaces whose description is consistent with that in the industrial standard, STEP [24]. Furthermore the degree of the tensor-product Bézier surface is always lower than the theoretical results [8]. Experimental results and theoretical analysis show that the new accurate FFD has advantages both in computational cost and storage, and can make use of graphics hardware to render the trimmed surfaces.

2 Overview of the proposed algorithm

This research is inspired by the Bernstein polynomial composition [8]. The composition between a planar quadrilateral and a Bézier volume is a tensor-product Bézier surface. With this result, the sampling problem in the first FFD method can be solved [23]. Since the Bézier volume can only produce global deformation, the B-spline volume is preferable in practical use. As we know, the B-spline volume can be converted into piecewise continuous Bézier volumes [9]. Thus, the Bernstein polynomial composition can be employed here for the B-spline FFD of the polygonal object. With proper subdivision of the polygonal object in the parametric space of the B-spline volume, the B-spline FFD can be converted as Bézier FFDs. According to the results in [8], the degree of the resultant tensor-product Bézier surface

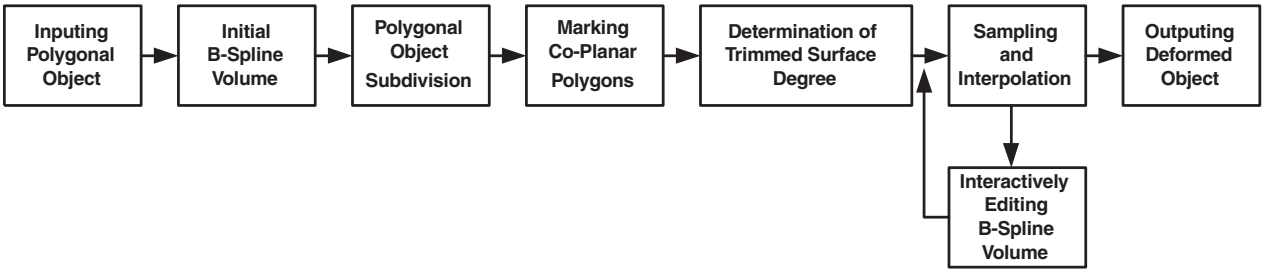


Fig. 1. The flow chart of the proposed algorithm

is equal to the tensor product of the degree of the Bézier volume. According to our analysis, this degree will never be reached in the work in this paper; that is, the degrees of the resultant tensor-product Bézier surfaces are always lower than this. Unlike the previous algorithms in [8, 18], we proposed a fast polynomial interpolation algorithm to compute the control points of the tensor-product Bézier surface. The flow chart of the proposed algorithm is shown in Fig. 1.

The initial shape of the B-spline volume can be either arbitrary or regular (orthogonal parallelepiped). An initial B-spline volume with an arbitrary shape is convenient for interactive shape modification. However, it is discommodious to define or automatically generate such an initial B-spline volume with an arbitrary shape. In practical use, an orthogonal parallelepiped is preferable, whose extension is identical with the object bounding box. Many computer animation systems adopt such an approach. In this paper we consider the regular case.

First, a new initial B-spline volume is introduced in this paper. The proposed initial B-spline volume has two desirable properties. To start with, the point coordinates in the object space are identical with its local coordinates corresponding to the B-spline volume, and the Jacobian of the B-spline is an identity matrix. After the initial B-spline volume is defined, the polygonal object is subdivided in the parametric space according to the knot vectors distribution. Each sub-polygon in the object after subdivision lies within a "knot box" (see definition in Sect. 3.2) of the B-spline volume. In fact, the B-spline volume limited in a knot box is a Bézier volume. The next step is to classify the coplanar sub-polygons in each knot box. At the same time, the degree of the resultant tensor-product Bézier surface is determined. For the coplanar sub-polygons in a knot box, it is not necessary to compute each sub-polygon deformation. We will only compute the deformation of the planar rectangular bounding box of the coplanar sub-polygons. The deformation of the coplanar sub-polygons in a knot box is the trimmed tensor-product Bézier surfaces. The trimmed curves are the edges of the coplanar sub-polygons. The control points of the tensor-product Bézier surface are evaluated through polynomial interpolation, where the interpolation matrix is constant.

3 B-spline FFD polygonal object as trimmed Bézier surfaces

The proposed algorithm is described in detail in this section. The section is organized according to the flow chart in Fig. 1.

3.1 A new definition of initial B-spline volume

3.1.1 The initial B-spline volume and local coordinates

From a mathematical point of view, FFD can be described as two mappings: $\Re^3 \rightarrow \Re^3$. The first mapping is from the object space to the parametric space, and it freezes the object in the B-spline volume. Let the mapping be noted as $L : (x, y, z) \rightarrow (u, v, w)$. The mapping L is computed once after the initial B-spline volume is defined. During the deformation, (u, v, w) is fixed for the point (x, y, z). The second mapping is the B-spline volume itself $R : (u, v, w) \rightarrow (\tilde{x}, \tilde{y}, \tilde{z})$. The relationship between (x, y, z) and $(\tilde{x}, \tilde{y}, \tilde{z})$ is linked by the triple (u, v, w), namely, the local coordinates of point (x, y, z).

For the given initial B-spline volume, there are two ways to compute the local coordinates (u, v, w). One way is called *forward mapping*, which is a linear

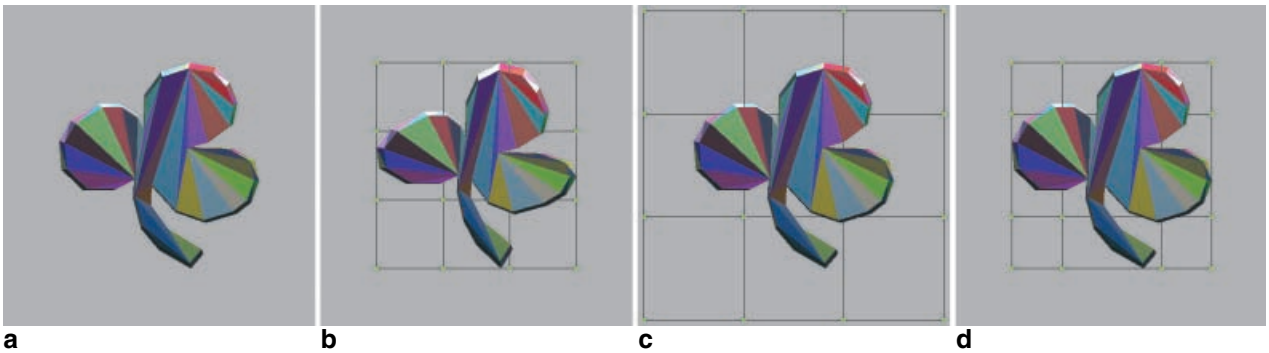


Fig. 2a–d. The degree of the B-spline volume is $2 \times 2 \times 1$; its lattice is $4 \times 4 \times 2$. **a** The original object. **b** The initial B-spline volume in end interpolation. Its control points are evenly distributed. There is warp in the deformation. **c** The uniform B-spline volume with its identity Jacobian. **d** The end-interpolation B-spline volume converted from (c). The object in both (c) and (d) has no warp when it is deformed by the initial B-spline volume.

transformation from (x, y, z) to (u, v, w). It is a fast way since only the linear transformation is computed. However, the object may be warped after it is deformed by the initial B-spline volume. An example of this is given in Fig. 2b, which shows the deformation of the object in Fig. 2a by the initial B-spline volume. The reason for this is that the initial B-spline volume in general is non-linear. The previous accurate FFD algorithms adopted this mapping technique [10, 11]. The second way is called *backward mapping*, where for each point (x, y, z) on the object, a triple (u, v, w) is computed such that $R(u, v, w) = (x, y, z)$. In general, there is no analytical solution for this problem and numerical method is adopted. By using the backward mapping, there is no warp when the object is deformed by the initial B-spline volume, but it is a time-consuming procedure, and the local coordinates are non-linearly distributed in general. It is not suitable for the accurate FFD algorithms. Could we design an initial B-spline volume that has the advantages of both of these two kinds of mapping? The answer is yes. We will give a constructive method in the following section.

3.1.2 Definition of uniform B-spline volume

Let the initial B-spline volume $R(u, v, w)$ be noted as

$$R(u, v, w) = \sum_{i=0}^{n_u-1} \sum_{j=0}^{n_v-1} \sum_{k=0}^{n_w-1} R_{ijk} N_{i,k_u}(u) N_{j,k_v}(v) N_{k,k_w}(w), \quad (1)$$

where n_u, n_v and n_w are the numbers of the control points, k_u, k_v and k_w are the degrees, $N_{i,k_u}(u)$, $N_{j,k_v}(v)$ and $N_{k,k_w}(w)$ are the B-spline functions along the u, v and w directions respectively. R_{ijk} are the control points, namely, the lattice in general. The object bounding box aligned to the coordinates axis is noted as $(x_{\min}, y_{\min}, z_{\min})$ and $(x_{\max}, y_{\max}, z_{\max})$. For simplicity, the x-, y- and z-coordinates are mapped as u, v and w respectively. Here the uniform B-spline volume means that both the knot vectors and the control points are uniformly distributed in three parametric directions. The knot vectors $\{u_i\}$, $\{v_j\}$ and $\{w_k\}$ are defined as follows:

$$\begin{cases} u_i = x_{\min} + (i - k_u) * (x_{\max} - x_{\min})/(n_u - k_u) \\ \qquad (i = 0, 1, \ldots, k_u + n_u)\,; \\ v_j = y_{\min} + (j - k_v) * (y_{\max} - y_{\min})/(n_v - k_v) \\ \qquad (j = 0, 1, \ldots, k_v + n_v)\,; \\ w_k = z_{\min} + (k - k_w) * (z_{\max} - z_{\min})/(n_w - k_w) \\ \qquad (k = 0, 1, \ldots, k_w + n_w)\,. \end{cases}$$

From the definition, we know that the valid parametric domain is identical with the object bounding box; that is, $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$. Next, the control point $R_{ijk} = (x'_{ijk}, y'_{ijk}, z'_{ijk})$ is defined by the following equation:

$$\begin{cases} x'_{ijk} = x_{\min} + i * (x_{\max} - x_{\min})/(n_u - 1) \\ \qquad (i = 0, 1, \ldots, n_u - 1)\,; \\ y'_{ijk} = y_{\min} + j * (y_{\max} - y_{\min})/(n_v - 1) \\ \qquad (j = 0, 1, \ldots, n_v - 1)\,; \\ z'_{ijk} = z_{\min} + k * (z_{\max} - z_{\min})/(n_w - 1) \\ \qquad (k = 0, 1, \ldots, n_w - 1)\,. \end{cases}$$

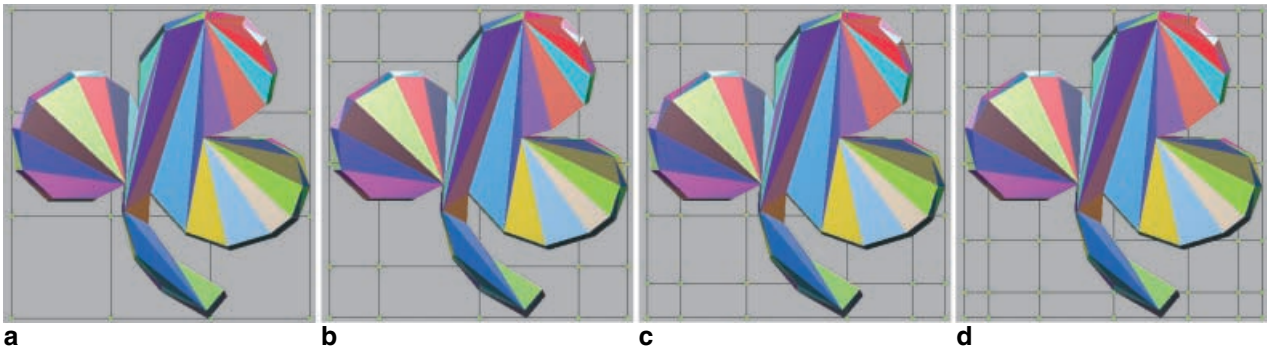


Fig. 3a–d. The control point configurations for the end-interpolation B-spline volume of degree $3 \times 3 \times 1$. The numbers of the control points are noted under each subfigure. **a** $4 \times 4 \times 2$; **b** $5 \times 5 \times 2$; **c** $6 \times 6 \times 2$; **d** $7 \times 7 \times 2$

In general, the extension of $R(u, v, w)$ with the above control points $\{(x'_{ijk}, y'_{ijk}, z'_{ijk})\}$ is smaller than the object bounding box. If the object is deformed by $R(u, v, w)$, there will be no warp introduced except for constrictions. To make the extension of $R(u, v, w)$ identical with the object bounding box, the control points should be scaled. Let $(x'_{\min}, y'_{\min}, z'_{\min})$ and $(x'_{\max}, y'_{\max}, z'_{\max})$ be noted as the corner points of the $R(u, v, w)$ extension, which correspond to $R(x_{\min}, y_{\min}, z_{\min})$ and $R(x_{\max}, y_{\max}, z_{\max})$. Three scalars can be computed by the following equations:

$$\begin{cases} s_x = (x_{\max} - x_{\min})/(x'_{\max} - x'_{\min}) ; \\ s_y = (y_{\max} - y_{\min})/(y'_{\max} - y'_{\min}) ; \\ s_z = (z_{\max} - z_{\min})/(z'_{\max} - z'_{\min}) . \end{cases}$$

After x'_{ijk}, y'_{ijk} and z'_{ijk} are scaled by using scalars s_x, s_y and s_z respectively, the extension of the new B-spline volume is identical with the object bounding box. Furthermore, its Jacobian is an identity matrix. This means that when the object is deformed by such an initial B-spline volume, the object will remain unchanged. One example is shown in the Fig. 2c. It is the deformation of the object in Fig. 2a by a uniform B-spline volume as defined above.

3.1.3 Definition of end-interpolation B-spline volume

In practical applications, an end-interpolation B-spline volume is preferable, since its convex hull of control points is tighter than that of the uniform B-spline volume. It can be accomplished by using the knot-insertion algorithm for the end knots [9]. An example is shown in Fig. 2d, which is a conversion result of Fig. 2c. Besides being free of warp, another advantage is that the object point coordinates are identical with its local coordinates. Thus, linear-transformation computational costs are saved.

Since the above end-interpolation B-spline volume results from a uniform one, an interesting result can be derived according to the knot-insertion algorithm. Let us illustrate the result with the 1D case. Supposing $R(u)$ is an end-interpolation B-spline curve converted from a uniform one. It has n control points $\{R_i\}_{i=0}^{n-1}$ and its degree is k. Let R_0R_1 be noted as vector $\boldsymbol{u}$. If $n = k+1$, it is a Bézier curve. The control points are evenly distributed. Else, the results are described by Eq. (2). Thus the end-interpolation B-spline volume can be defined in a straightforward way. Some examples are shown in Fig. 3.

$$\begin{cases} R_0R_1 = R_{n-2}R_{n-1} = \boldsymbol{u} \\ R_1R_2 = R_{n-3}R_{n-2} = 2\boldsymbol{u} \\ \vdots = \vdots = \vdots \\ R_{k-1}R_k = R_{n-k-1}R_{n-k} = k\boldsymbol{u} \\ R_sR_{s+1} = k\boldsymbol{u} \ \ (other\ s) \quad . \end{cases} \tag{2}$$

With the above steps, we defined an initial B-spline volume. It has the following properties. Its parametric domain is identical with the object bounding box. It has end interpolation, and it is linear. Its Jacobian is an identity matrix. When the object is embedded into its parametric domain by using either forward mapping or backward mapping, the deformation results are the same. Each object point coordinate is identical to its local coordinate.

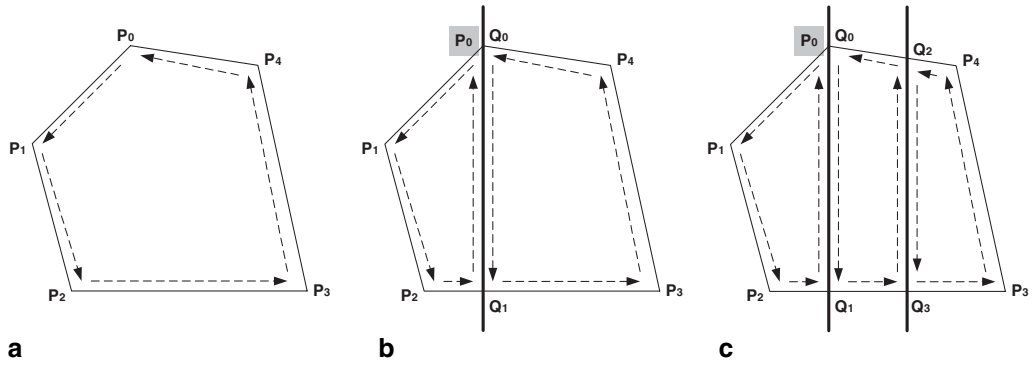


Fig. 4. a Original polygon and its cycle-linked list. **b** Knot plane crosses a polygon vertex P_0. The vertex P_0 is marked as the intersection and linked with another intersection point Q_1. **c** The final linked list after all intersection points are evaluated

3.2 Subdivision of a polygonal object in the parametric space

Before introducing the subdivision algorithm, we first define the knot box. A knot box is a 3D knot interval for the B-spline volume. For example, $[u_i, u_{i+1}] \times [v_j, v_{j+1}] \times [w_k, w_{k+1}]$ is a knot box for Eq. (1). It can be indexed by (i, j, k) for $k_u \le i < n_u, k_v \le j < n_v, k_w \le k < n_w$. The B-spline volume limited on a knot box is a Bézier volume. The purpose of the subdivision is to make each polygon in the object lie in a knot box after subdivision. The polygons after subdivision are called *sub-polygons* throughout the rest of paper. The proposed subdivision algorithm is described as follows:

```
for ( each parametric direction u,v,w )
{
 for ( each polygon in the object )
 {
  form the cycle-linked list for the vertex of the
  polygon;
  find indices i0 and i1 such that knot plane ui
  ( i0 < i < i1 ) crosses the polygon;
  for ( each knot plane ui ( i0 < i < i1 ) )
  {
   compute the intersections between each
   knot plane and polygon;
   insert the intersections into
   a cycle-linked list;
  }
  extract the sub-polygons from
  the cycle-linked list;
 }
}
```

3.2.1 Computing intersections between the knot planes and the polygon

We assume that the object is composed of convex polygons. Let $\{P_i\}_{i=0}^{n-1}$ be an n-sided polygon. It may be from the original object or from previous subdivision steps. First a cycle-linked list is built for the polygon vertex. An example is shown in Fig. 4a. Then the knot indices i_0 and i_1 are found such that the plane determined by knot u_i, v_i or w_i $(i_0 < i < i_1)$ crosses the current polygon. For each knot plane crossing the polygon, there are two intersections on the polygon. The two intersections should be linked to each other for traversal purposes in the next step. If the knot plane crosses a polygon vertex, then the crossed polygon vertex is re-marked as an intersection point, and this is linked with the other intersection point, as shown in Fig. 4b. When the sub-polygons are extracted from the linked cycle list, each polygon vertex will be visited once and each intersection will be visited twice. Thus each point will be assigned with a `Counter`: `Counter` = 1 for the polygon vertex; `Counter` = 2 for the intersection.

Here, if a knot plane crosses a polygon edge or one intersection is coincident with another, then there is no intersection between the knot plane and the polygon. Examples of crossing an edge or special vertex are shown in Fig. 5a,b. If we do not check such singular cases carefully, degenerate sub-polygons will be produced. The degenerate polygons can be neglected in our previous methods, since they only cause some wasteful computations [10, 11]; but, in

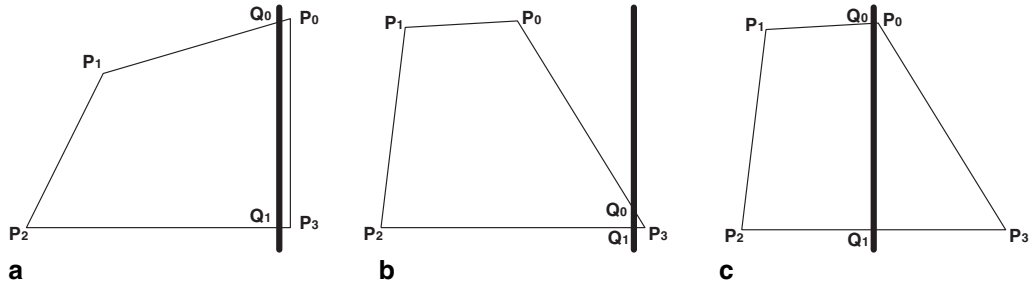


Fig. 5a–d. Singular cases in subdivision, with the knot plan drawn as a *bold line*: **a** if the knot plane were not regarded as crossing edge P_0P_3 because of a numerical error, then a 0-width polygon $P_0Q_0Q_1P_3$ would be generated; **b** if the knot plane were not regarded as crossing an extreme vertex P_3, a degenerate triangle $Q_0Q_1P_3$ would be produced; **c** if the knot plane were not regarded as crossing vertex P_0, the polygon $P_0Q_0Q_1P_3$ will possess a 0-length edge P_0Q_0

the method proposed here, they cause fatal errors in the following process steps. This is the reason why we check carefully for any singular cases in the subdivision.

3.2.2 Extracting the sub-polygons from the linked cycle list

For convenience of extracting sub-polygons, the data structure of the polygon vertex and the intersection is proposed as follows:

```
class node {
 double Position[3];
   // vertex or intersection coordinates
 short int  Counter;
   // "1" if vertex ; "2" if intersection
 class node *NextNode;
   // pointer to the next points:
   // vertex or intersection;
 class node *Intersection;
   // NULL if object vertex
}
```

The algorithm for extracting sub-polygons begins from a `node` whose `Counter` is 1. The key step is to determine the next node from the linked cycle list. There will be some degenerate sub-polygons generated in the old subdivision algorithm [10]. The traversal algorithm to extract a sub-polygon is described in the following. When all sub-polygons are extracted from the linked list, all `Counters` are 0. According to the algorithm, there are three sub-polygons extracted in Fig. 4c. The three sub-polygons in correct vertex order are: $P_1P_2Q_1Q_0$, $Q_0Q_1Q_3Q_2$ and $Q_3P_3P_4Q_2$.

```
ExtractSubPolygon()
{
 pHead = GetFirstNode();
 pHead->Counter--;
 if(pHead->NextNode->Counter == 0) {
   pNode = pHead->Intersection;
   status = INTERSECTION_EDGE;
 }
 else {
   pNode = pHead->NextNode;
   status = POLYGON_EDGE;
 }

 while(pNode != pHead) {
   pNode->Counter--;
   if(status == POLYGON_EDGE) {
     if(pNode->Intersection == NULL)
       pNode = pNode->NextNode;
     else {
       pNode = pNode->Intersection;
       status = INTERSECTION_EDGE;
     }
   }
   else {
     pNode = pNode->NextNode;
     status = POLYGON_EDGE
   }
 }
}
```

Compared with the subdivision algorithm in [10], the proposed algorithm has two improvements. The new algorithm is more robust than the old one. There are no degenerate sub-polygons produced, which are "0-area" or "0-edge" sub-polygons. Another improvement is that the new algorithm produces fewer sub-polygons than before since no-triangulation step is necessary. Some examples are shown in

Figs. 7–13, and a comparison of the sub-polygon numbers is shown in Table 2 in Sect. 4.

3.3 Marking coplanar sub-polygons and determining surface degrees

3.3.1 B-spline deformation of a quadrilateral rather than a triangle

After subdivision, each sub-polygon lies in a knot box. In the previous results [10, 11], all sub-polygons are triangles. The accurate deformation of a triangle is a triangular Bézier surface, whose degree equals the degree of the B-spline volume. Unfortunately, most graphics APIs (such as OpenGL and Direct3D) and graphics hardware cannot support the display of a triangular Bézier surface directly. The triangular Bézier surface needs to be tessellated into triangular meshes or converted into three tensor-product Bézier surfaces in advance. Another inconvenience is that the triangular Bézier surface cannot be integrated into the current systems because most current geometric modelling and computer animation systems do not support the data structure of triangular Bézier surfaces directly. To overcome the above problems, it is necessary to consider tensor-product Bézier surfaces as an accurate deformation result.

According to the results in [8], the B-spline deformation of a quadrilateral in a knot box is a tensor-product Bézier surface, whose degree is the tensor product of the degree of the B-spline volume. The brute-force solution is that the sub-polygons are tessellated into quadrilaterals, and then the quadrilaterals are deformed as tensor-product Bézier surfaces. This is an inefficient solution. Firstly, there are many triangles among the sub-polygons in general. Each triangle will be tessellated as three quadrilaterals; that is, there will be three tensor-product Bézier surfaces generated. This is a heavy burden in terms of both computation and storage space. Secondly, some sub-polygons in a knot box may be coplanar, and it is inefficient to deform them one by one. In fact they are parts of a "large" tensor-product Bézier surface. If such a "large" surface can be computed, trimmed surfaces can be adopted for representation of the coplanar sub-polygons deformation. Thus, both computational cost and storage space are saved.

3.3.2 Marking coplanar sub-polygons in a knot box

To mark coplanar sub-polygons, the data structure `Normal`is defined as follows. The coplanar sub-polygons in a knot box will be marked and then classified into the `Normal` structure.

```
class Normal
{
  short i,j,k;
   // The index of the knot box;
  double n[3], d;
   // Plane equation of coplanar polygons;
  double s[3], t[3];
   // Local frames for the bounding box;
  short ns, nt;
   // Degree of resulting Bezier surface.
  RECTANGLE box;
   // 2D bounding box of sub-polygons in s-t plane.
  POLYGON *List;
   // Pointer to the coplanar sub-polygon list
}
```

Each sub-polygon has a plane equation. It can be described by a normalized normal (n_x, n_y, n_z) and scalar d as follows:

$$xn_x + yn_y + zn_z + d = 0.$$

If a sub-polygon is degenerate, there will be no plane equation. In practical implementations, this will cause the algorithm to become unstable, since its normal will be the zero-vector. Thus, a robust subdivision algorithm is necessary in the previous step. If two sub-polygons are coplanar, their normalized normals (n_x, n_y, n_z) and scalars d should be equal. Because of numerical error, equal comparisons are implemented as the following equations:

$$\begin{cases} \|\boldsymbol{n}_1 \cdot \boldsymbol{n}_2 - 1\| < \varepsilon_{\boldsymbol{n}} \\ \|d_1 - d_2\| \quad\ < \varepsilon_d \end{cases}.$$

In our implementation, $\varepsilon_{\boldsymbol{n}} = 2.2 \times 10^{-8}$ and $\varepsilon_d = 1.1 \times 10^{-6}$. By using these two thresholds, if two polygons in the original object are coplanar, then the sub-polygons generated from them will be also coplanar. No visual artefact will occur because of the numerical error. By using the above two criteria, the coplanar sub-polygons in a knot box will be classified into the same `Normal` structure. In general, there are many `Normal` structures in a knot box, whose number is smaller than the number of sub-polygons in the knot box.

3.3.3 Determining the degree of the tensor-product Bézier surface according to the normal

In this section, the degree of the "large" tensor-product Bézier surface is determined for each `Normal` structure in a knot box. According to the

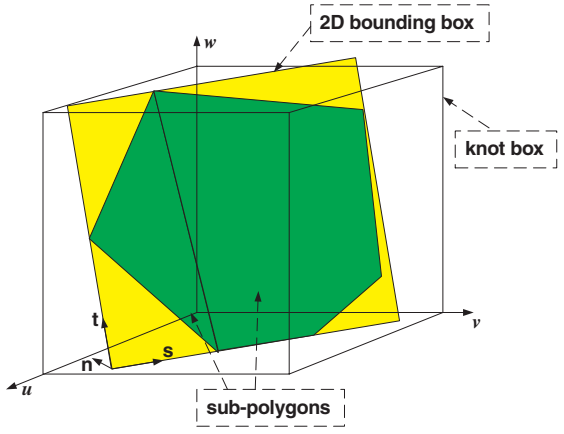


Fig. 6. The local coordinate system $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$ and the 2D bounding box of the sub-polygons in the $\boldsymbol{s}$-$\boldsymbol{t}$ plane

theoretical results in [8], the degree of the tensor-product Bézier surface, which is the accurate deformation of a quadrilateral by the $R(u, v, w)$ in Eq. (1), is $(k_u + k_v + k_w) \times (k_u + k_v + k_w)$. In our algorithm, the degrees of large surfaces are always lower than this value, and it could be known for special cases. For example, if a quadrilateral lies in the knot plane $u = \tilde{u}$, with its normal $(n_x, n_y, n_z) = (1, 0, 0)$, then the degree of its corresponding tensor-product Bézier surface is $k_v \times k_w$ or $k_w \times k_v$. In fact, the surface is the part of the iso-parameter surface $u = \tilde{u}$.

How can we reduce the surface degrees for general cases? Let us illustrate this with Fig. 6. Supposing the normal $\boldsymbol{n} = (n_x, n_y, n_z)$ has no zero component, or $n_x \cdot n_y \cdot n_z \neq 0$, and the degree of the B-spline volume is satisfied: $k_u \leq k_v \leq k_w$. Then a normalized vector $\boldsymbol{s}$ orthogonal to the normal $\boldsymbol{n}$ can be defined as

$$\boldsymbol{s} = (s_x, s_y, 0) = \frac{(-n_y, n_x, 0)}{\|(-n_y, n_x, 0)\|} .$$

Together with another normalized vector $\boldsymbol{t} = \boldsymbol{n} \times \boldsymbol{s}$, the three vectors $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$ define a local coordinate system. The sub-polygons in the `Normal` structure lie in the $\boldsymbol{s}$-$\boldsymbol{t}$ plane of the local coordinate system. In the $\boldsymbol{s}$-$\boldsymbol{t}$ plane, we can compute a 2D bounding box for sub-polygons in the `Normal`structure, which is shown as the yellow rectangle in Fig. 6. We can prove that the degree of the Bézier surface is $(k_u + k_v) \times (k_u + k_v + k_w)$, which is the accurate deformation of the rectangle. The reason is that along the $\boldsymbol{s}$-direction in the local coordinate system, the degree of the surface is independent of the parametric w-direction of B-spline volume.

For a given B-spline volume, the accurate deformation results can be classified by seven cases. The local coordinate system $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$ and surface degree are listed in Table 1. This covers all possible configurations of sub-polygons in a `Normal`structure. Till now, we have determined the degree of Bézier surfaces and the rectangle (2D bounding box of the coplanar sub-polygons in the $\boldsymbol{s}$-$\boldsymbol{t}$ plane) for each `Normal`structure. The Bézier surface accompanying the sub-polygons forms the trimmed-surface representation of the accurate B-spline deformation.

3.4 Sampling and interpolation

So far, the preprocessing mechanisms have been introduced. These can be implemented once. The

Table 1. Determining the local coordinate system and the degrees of the Bézier surface according to the normal surface

Normal $\boldsymbol{n}$	$\boldsymbol{s}$	$\boldsymbol{t}$	degrees
$(\pm 1, 0, 0)$	$(0, 1, 0)$	$(0, 0, \pm 1)$	$k_v \times k_w$
$(0, \pm 1, 0)$	$(0, 0, 1)$	$(\pm 1, 0, 0)$	$k_w \times k_u$
$(0, 0, \pm 1)$	$(1, 0, 0)$	$(0, \pm 1, 0)$	$k_u \times k_v$
$(0, n_y, n_z)$	$(1, 0, 0)$	$(0, n_z, -n_y)$	$k_u \times (k_v + k_w)$
$(n_x, 0, n_z)$	$(0, 1, 0)$	$(n_z, 0, -n_x)$	$k_v \times (k_u + k_w)$
$(n_x, n_y, 0)$	$(0, 0, 1)$	$(n_y, -n_x, 0)$	$k_w \times (k_u + k_v)$
(n_x, n_y, n_z)	$\frac{(0, -n_z, n_y)}{\|(0, -n_z, n_y)\|}$ if $(k_u \geq k_v, k_u \geq k_w)$	$\boldsymbol{n} \times \boldsymbol{s}$	$(k_v + k_w) \times (k_u + k_v + k_w)$
	$\frac{(n_z, 0, -n_x)}{\|(n_z, 0, -n_x)\|}$ if $(k_v \geq k_u, k_v \geq k_w)$		$(k_u + k_w) \times (k_u + k_v + k_w)$
	$\frac{(-n_y, n_x, 0)}{\|(-n_y, n_x, 0)\|}$ if $(k_w \geq k_u, k_w \geq k_v)$		$(k_u + k_v) \times (k_u + k_v + k_w)$

work described here, which is to compute the control points of the tensor-product Bézier surface, is processed by an interactive method.

3.4.1 Solving the functional composition of the Bernstein polynomial

There are two methods of computing the control points of the Bézier surface introduced in Sect. 3.3.3. One solution is through the shifting operators and the generalized de Casteljau algorithm, which is similar to the work in [10]. It is a numerically stable but time-consuming algorithm. Firstly, the generalized de Casteljau algorithm is time consuming in itself. Secondly, the B-spline volume must be converted into piecewise continuous Bézier volumes during the interaction, and this leads to a heavy computational burden. Another disadvantage is that the source code to solve the functional composition is complex when the resultant surface is of tensor-product form. The second solution is through polynomial interpolation, which was used for our previous accurate FFD [11]. This solution is much faster than the first one, is suitable for real-time interaction and the source code is very simple. We will choose it to solve the functional composition between the rectangle and the B-spline volume.

From a knowledge of numerical analysis, we know that a tensor-product Bézier surface $R(s,t)$ of degree $n_s \times n_t$ can be interpolated if $(n_s+1)\times(n_t+1)$ sampling points are given on the surface, where the sampling points are distributed in the same way as the surface control points. The interpolation can be accomplished by solving the linear equation system. There are two interpolation approaches. One approach is that the sampling points are arranged as in the 1D case. This is similar to the work for interpolating the triangular Bézier surface [11]. It is inefficient, since the equation size of tensor-product surface is large and many of the submatrices are $\mathbf{0}$. Another approach is to convert the tensor-product surface-interpolation problem into two stages of curve-interpolation problems. This is more efficient [9], so we will choose the latter solution.

3.4.2 Polynomial interpolation for tensor-product Bézier surface

First, the sampling points are computed on the surface. The procedure is to choose sampling points uniformly distributed in the parametric domain of the Bézier surface; that is, $R(i/n_s, j/n_t)$ for $i = 0, 1, \ldots, n_s$ and $j = 0, 1, \ldots, n_t$. In our context, such sampling points are computed in two steps: the first step is to compute the sampling points in the rectangle (2D bounding box in the Sect. 3.3.3) and then compute the $R(i/n_s, j/n_t)$ from the B-spline volume by using the de Boor–Cox algorithm. We need not convert the B-spline volume into piecewise Bézier volumes as in [10, 11], and this results in the need for many fewer computations.

There is a subtlety in the evaluation of the sampling points on the B-spline volume. Because of the choice of the local coordinate system $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$ in Sect. 3.3.3, the 2D bounding box in a `Normal`structure may not lie totally in the corresponding knot box. If we compute the points on the B-spline volume by using de Boor–Cox algorithm directly, it may give a wrong interpolation result. The reason lies in that the functional composition between a rectangle and a B-spline volume limited on a knot box is equivalent to a composition between a rectangle and a Bézier volume. If the rectangle lies partly outside of the Bézier volume, we also compute the sampling points using this Bézier volume for the sake of consistency. In our algorithm, the de Boor–Cox algorithm has a slight modification: the sampling points are always evaluated in the same knot box index by `(i,j,k)` in the `Normal`structure. This can be viewed as the generalized de Boor–Cox algorithm.

The two-step polynomial interpolations should be started from a parametric direction whose degree is lower than a corresponding one. Without loss of generality, suppose $n_s \le n_t$. Then, for $j = 0, 1, \ldots, n_t$, we should solve the following linear equations:

$$\begin{pmatrix} B_{0,n_s}(0) & B_{1,n_s}(0) & \cdots & B_{n_s,n_s}(0) \\ B_{0,n_s}\left(\frac{1}{n_s}\right) & B_{1,n_s}\left(\frac{1}{n_s}\right) & \cdots & B_{n_s,n_s}\left(\frac{1}{n_s}\right) \\ \vdots & \vdots & \ddots & \vdots \\ B_{0,n_s}\left(\frac{n_s}{n_s}\right) & B_{1,n_s}\left(\frac{n_s}{n_s}\right) & \cdots & B_{n_s,n_s}\left(\frac{n_s}{n_s}\right) \end{pmatrix} \begin{pmatrix} \widetilde{R}_{0,j} \\ \widetilde{R}_{1,j} \\ \vdots \\ \widetilde{R}_{n_s,j} \end{pmatrix} = \begin{pmatrix} R\left(\frac{0}{n_s}, \frac{j}{n_t}\right) \\ R\left(\frac{1}{n_s}, \frac{j}{n_t}\right) \\ \vdots \\ R\left(\frac{n_s}{n_s}, \frac{j}{n_t}\right) \end{pmatrix}. \quad (3)$$

The left-hand $(n_s+1)\times(n_s+1)$ matrix in Eq. (3) is constant for a given degree n_s, which is independent of the underlying curve and is only related to the de-

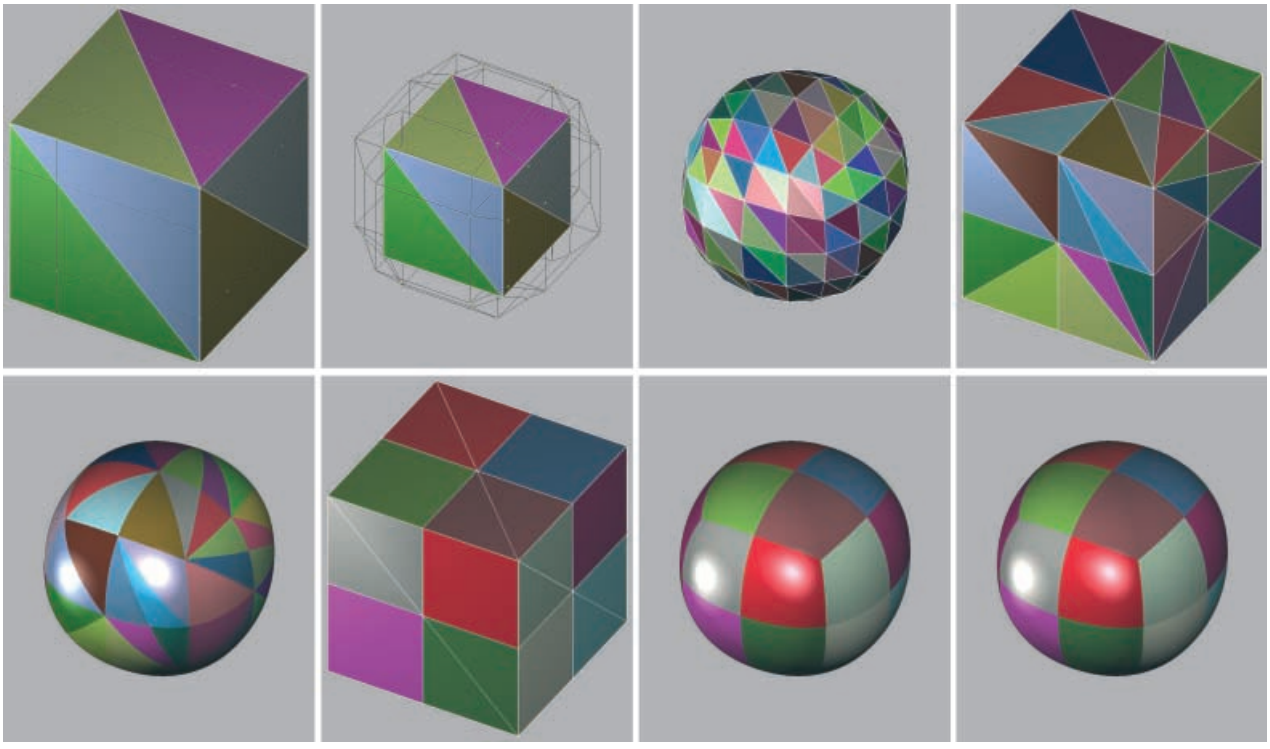


Fig. 7. The degree of the B-spline volume is $2 \times 2 \times 2$. The number of control points is $4 \times 4 \times 4$

gree. Let this be noted as $\mathbf{B}_k$ for degree k. Obviously, this can be inverted. The inverse matrix of $\mathbf{B}_k$ can be precomputed. In our implementation, $\mathbf{B}_k^{-1}$ is evaluated for $k \leq 12$ in advance. The reason for taking degree 12 is that we seldom use a B-spline volume whose degree is greater than $4 \times 4 \times 4$. After $\widetilde{R}_{ij}$ are obtained, the control points R_{ij} can be obtained for $i = 0, 1, \ldots, n_s$ by:

$$\left(R_{i,0} R_{i,1} \cdots R_{i,n_t}\right)^{\mathbf{T}} = \mathbf{B}_{n_t}^{-1} \left(\widetilde{R}_{i,0} \widetilde{R}_{i,1} \cdots \widetilde{R}_{i,n_t}\right)^{\mathbf{T}} . \quad (4)$$

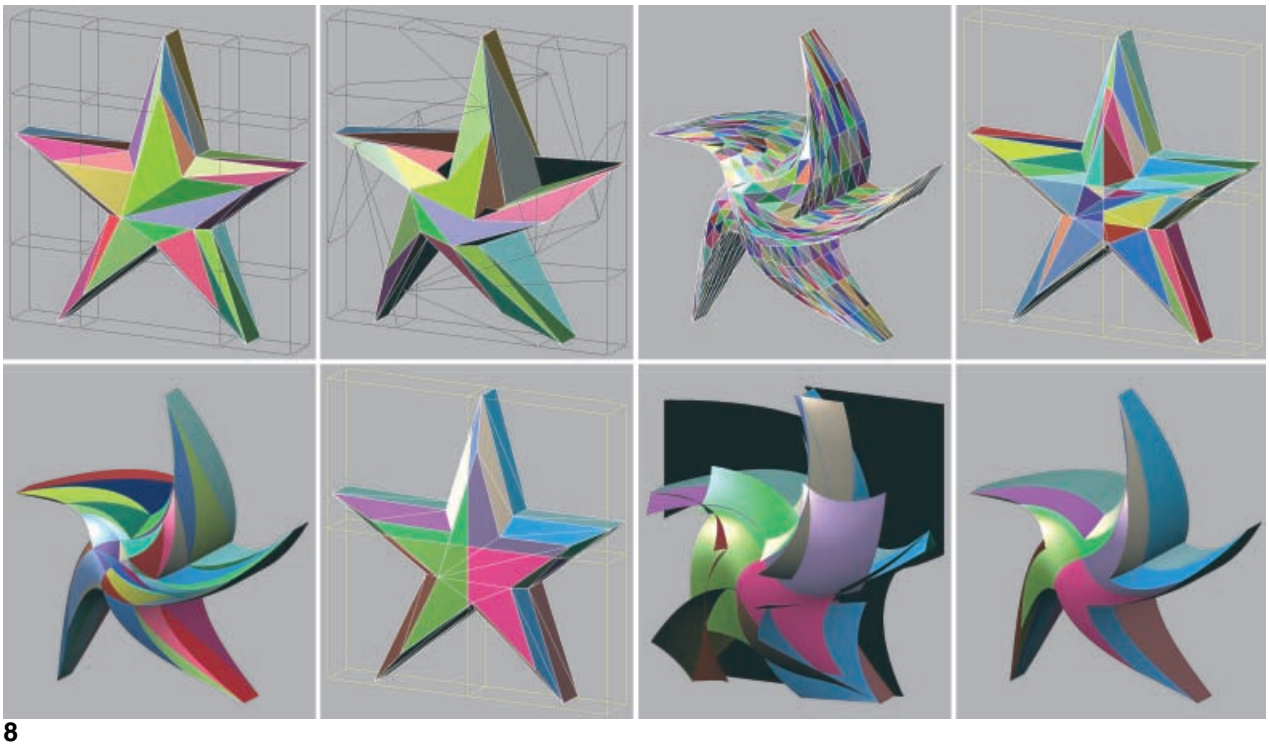
Combining with the sub-polygons, the trimmed tensor-product Bézier surfaces are the accurate deformation result.

4 Implementation, results and discussions

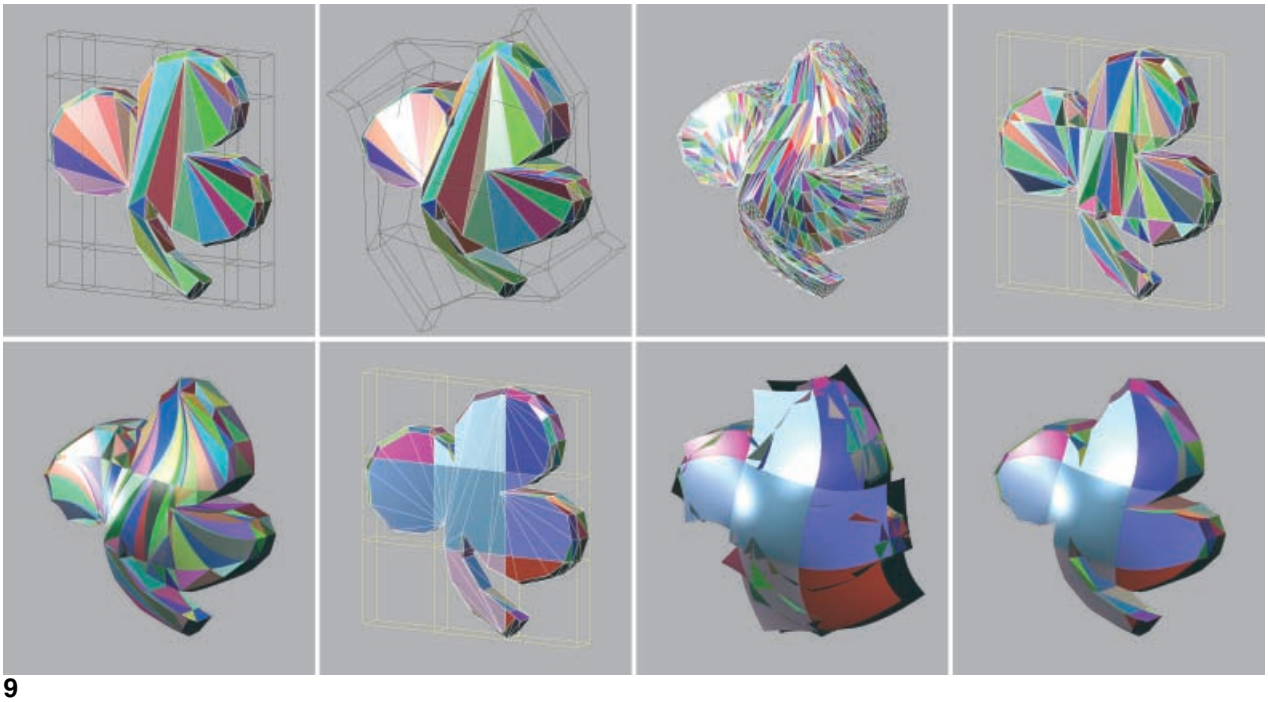
4.1 Implementation and comparison results

We have implemented the proposed algorithm on a PC with an AMD Athlon 1 GHz CPU. OpenGL was adopted as graphics API. The trimmed NURBS surfaces can be displayed directly by using the functions in the OpenGL Utility Library. In each of Figs. 7–13, there are 8 subfigures, and these are as follows:

- The original object with the initial B-spline volume lattice.
- The traditional free-form deformation result without additional sampling.
- The free-form deformation result where the object is uniformly subdivided.
- The previous object subdivision method [10]. The subdivision results in triangular meshes.
- The previous accurate deformation result [11]. The results are triangular Bézier surfaces that correspond with the triangles in the 4th subfigure
- The object subdivision result by using the proposed method. The coplanar sub-polygons in a knot box are displayed by using the same color, which will be deformed as the trimmed surfaces from a "large" Bézier surface.
- The "large" Bézier surface before being trimmed.
- The final result, the trimmed Bézier surfaces.



8



9

Fig. 8. The degree of the B-spline volume is $2 \times 2 \times 1$. The number of control points is $4 \times 4 \times 2$
Fig. 9. The degree of the B-spline volume is $2 \times 2 \times 1$. The number of control points is $5 \times 5 \times 2$

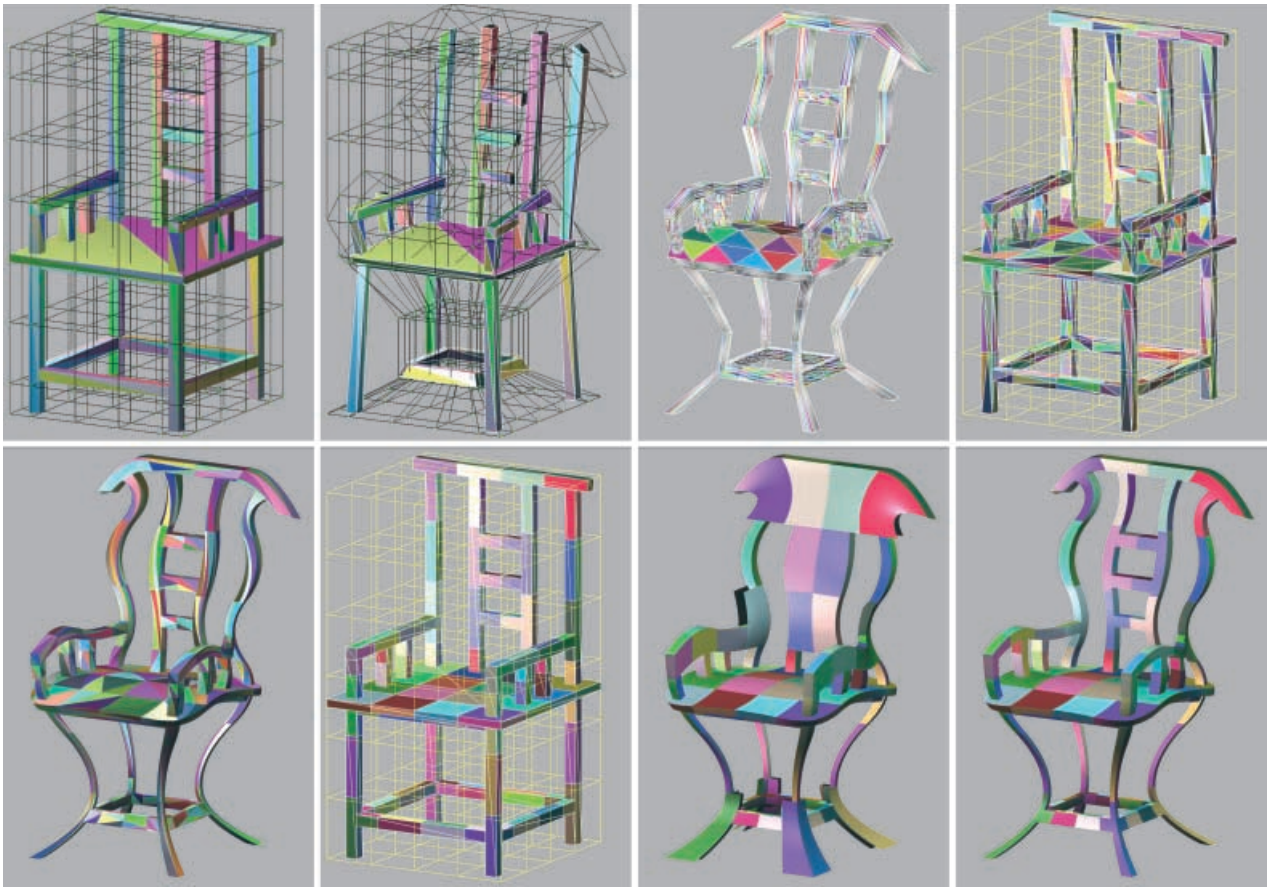


Fig. 10. The degree of the B-spline volume is $2 \times 2 \times 2$. The number of control points is $6 \times 8 \times 6$

Of course, the final result in the 8th subfigure is the same as the 5th subfigure except for their mathematical expression. A comparison of the numbers of sub-polygons, the numbers of Bézier triangles and tensor-product Bézier surfaces and the run-time between the proposed method and our previous method [11] is given in Table 2.

4.2 Discussion

From implementation comparisons, we derived that the proposed algorithm is faster than the algorithm in [11]. The run-time acceleration results from two facts. Firstly, the number of tensor-product Bézier surfaces is much less than that of the triangular

Table 2. Implementation comparisons

Fig.	Degree of B-spline volume	Lattice	Triangles in original object	Sub-polygons number Old:New	Bézier patches Old:New	Run-time (s) comparison Old:New
7	$2 \times 2 \times 2$	$4 \times 4 \times 4$	12	60 : 36	60 : 24	0.071/0.000
8	$2 \times 2 \times 1$	$4 \times 4 \times 2$	56	142 : 96	142 : 47	$0.070 : 0.010 = 7.0$
9	$2 \times 2 \times 1$	$5 \times 5 \times 2$	200	518 : 344	518 : 157	$0.261 : 0.05 = 5.22$
10	$2 \times 2 \times 2$	$6 \times 8 \times 6$	260	1436 : 840	1636 : 318	$1.442 : 0.060 = 24.03$
11	$2 \times 2 \times 2$	$6 \times 9 \times 3$	178	814 : 438	814 : 194	$0.781 : 0.030 = 26.03$
12	$2 \times 2 \times 2$	$6 \times 5 \times 6$	96	840 : 390	840 : 207	$0.751 : 0.051 = 14.73$
13	$2 \times 2 \times 2$	$7 \times 3 \times 6$	956	1980 : 1424	1980 : 1154	$1.722 : 0.671 = 2.57$

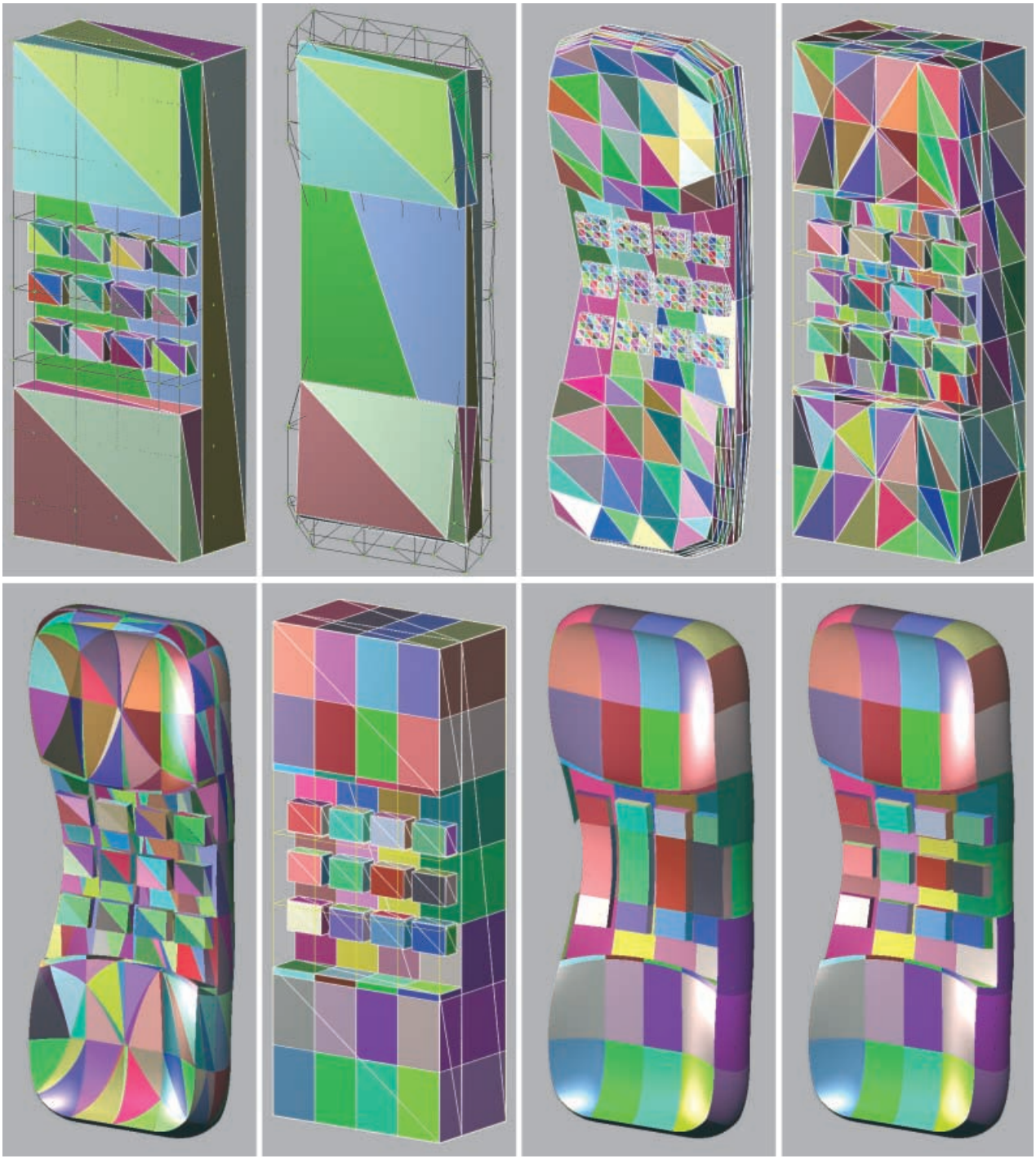


Fig. 11. The degree of the B-spline volume is $2 \times 2 \times 2$. The number of control points is $6 \times 8 \times 3$

Bézier surfaces in [11], and secondly, the resultant tensor-product Bézier surface has a low degree. Of course, the run-time acceleration times are also influenced by the initial object configuration. If the object has many polygons that are parallel to the coordinate planes or if it has many coplanar polygons, then run-time acceleration will be evident, as in Figs. 10 and 11. Otherwise, acceleration will be

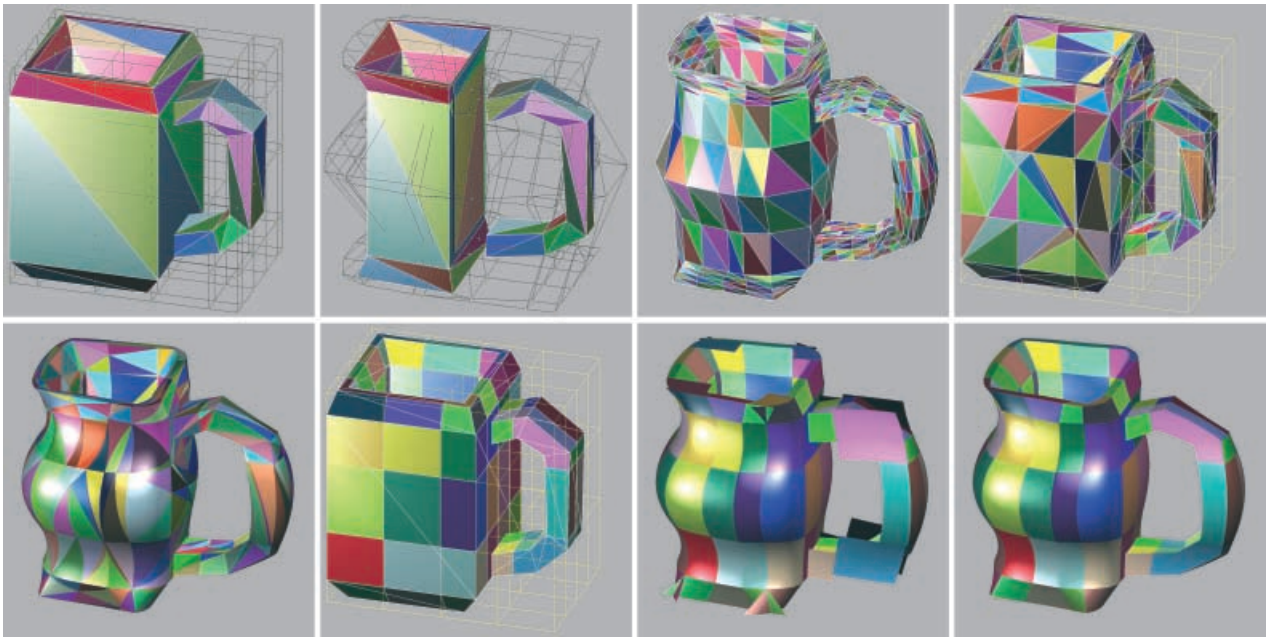


Fig. 12. The degree of the B-spline volume is $2 \times 2 \times 2$. The number of control points is $6 \times 5 \times 6$

less evident, as in Fig. 13. The storage comparison is approximately equal to the run-time comparison, since the polynomial interpolation method is adopted for both algorithms. The number of sampling points is equal to the number of control points of the Bézier surfaces or Bézier triangles since matrix multiplication can be omitted, compared with the evaluation of the sampling points on the B-spline volume. Thus we can say that the storage cost of the proposed algorithm is also lower than that of the previous algorithm [11]. The third advantage of the proposed algorithm is that the deformation result is a trimmed tensor-product Bézier, where the trimmed surface description is consistent with that in the industrial standard, STEP. Thus, the algorithm can be integrated into current geometric modelling systems and computer animation systems without any difficulty. Finally, the source code to solve the functional composition between the rectangle and the B-spline volume is very simple compared with the generalized de Casteljau algorithm.

Real-time interaction is always beneficial in computer graphics. For a simple object, the proposed algorithm is fast enough to achieve real-time editing. For interactively deforming a complex object, however, it may not be such a good choice, since it is time consuming compared with the traditional FFD method. For such cases, FFD acting directly on the vertex should be adopted for the interaction. If the deformation is far away from the real deformation result, uniform subdivision for a polygonal object can be conducted to solve the sampling problem. An accurate FFD of the polygonal object is recommended for the final deformation result. We chose this strategy in our implementation.

5 Conclusion

In this paper, a new accurate B-spline free-form deformation algorithm for a polygonal object is proposed. First, a new initial B-spline volume definition is given. Its Jacobian is an identity matrix. Then the object is subdivided in the parametric space of the B-spline volume. The sub-polygons so obtained are free of degenerate sub-polygons. After subdivision, the coplanar sub-polygons are combined, and the degree of the resulting surface is then determined. Finally the tensor-product Bézier surface is evaluated through polynomial interpolation, and the deformation results in the trimmed tensor-product Bézier surfaces.

The algorithm has advantages in both run-time and storage over the previous one [11]. The algorithm

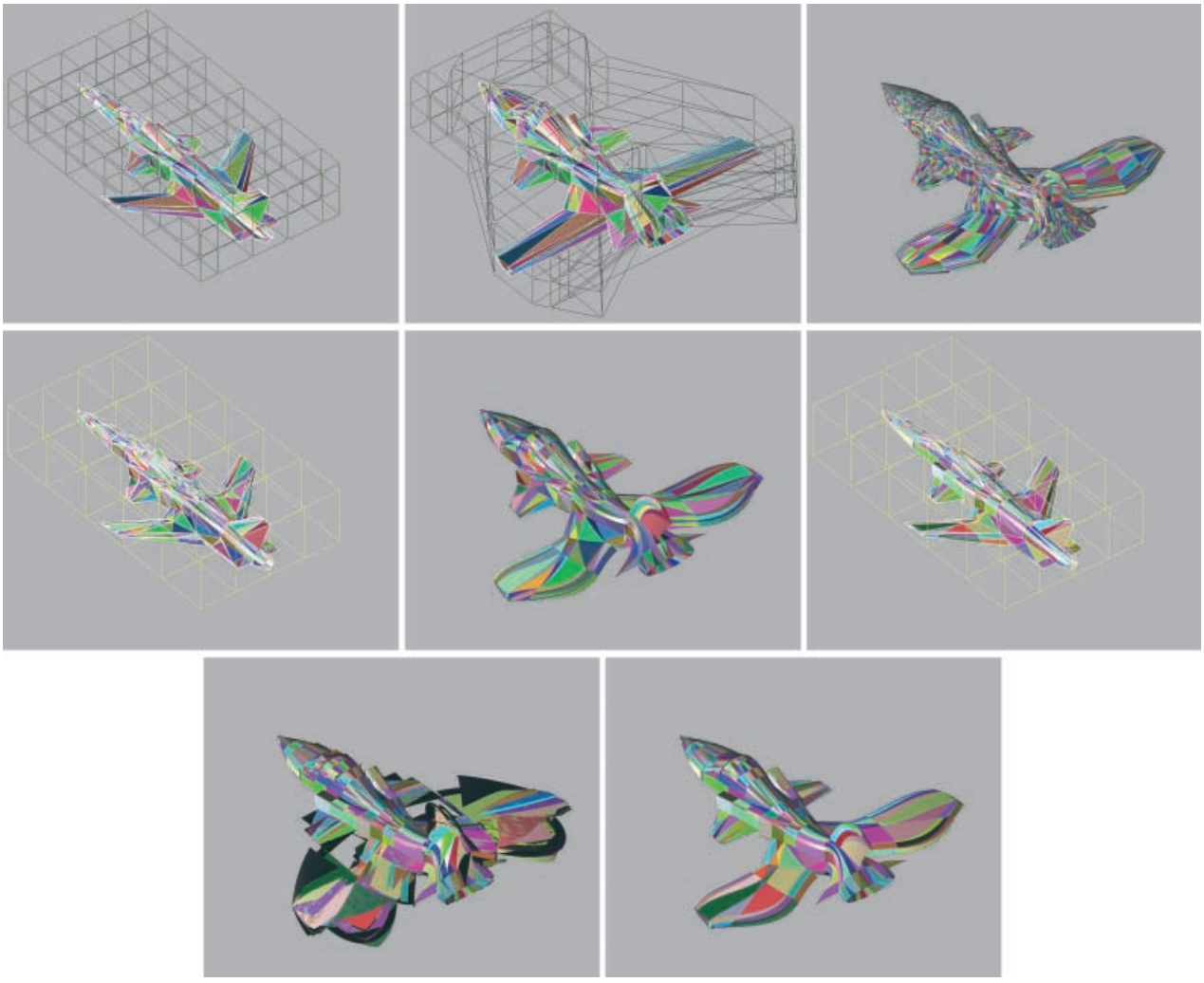


Fig. 13. The degree of the B-spline volume is $2 \times 2 \times 2$. The number of control points is $7 \times 3 \times 6$

can also be integrated into geometric modelling and computer animation systems without difficulty, since the deformation results are the trimmed tensor-product Bézier surfaces, whose representation is in accordance with that of the industrial standard, STEP. The display of the resultant surface can be accelerated if the graphics API is supported by hardware.

Acknowledgements. The work is supported by National Natural Science Foundation of China (No.69903008) and the Scientific Foundation of China for Innovative Research Groups (No.60021201). The authors also thank Dr. Neil Dodgson from the Computer Laboratory at the University of Cambridge, who supplied us with the reference [21]. We also thank the reviewers who gave helpful suggestions on improving the quality of the paper.

References

1. Bechmann D, Bertrand Y, Thery S (1997) Continuous free form deformation. Comput Networks ISDN Syst 29(14):1715–1725
2. Bechmann D (1998) Multidimensional free-form deformation tools. In: de Sousa A, Hopgood B (eds) State of the art report (Eurographics '98). Blackwell, Lisbon, pp 102–110
3. Chadwick JE, Haumann DR, Parent RE (1989) Layered construction for deformable animated characters. ACM Comput Graph (SIGGRAPH '89) 23(3):243–252
4. Coquillart S (1990) Extended free-form deformation: a sculpturing tool for 3D geometric modeling. ACM Comput Graph (SIGGRAPH '90) 24(4):187–193
5. Coquillart S, Jancene P (1991) Animated free-form deformations: an interactive animation technique. ACM Comput Graph (SIGGRAPH '91) 25(4):23–26

6. Davis OR, Burton RP (1991) Free-form deformation as an interactive modeling tool. J Imaging Technology 17(4):181–187
7. DeRose T (1988) Compositing Bézier simplex. ACM Trans Graph 7(3):198–221
8. DeRose T, Goldman R, Hagen H, Mann S (1993) Functional composition algorithms via blossoming. ACM Trans Graph 12(3):113–135
9. Farin G (1990) Curves and surfaces for computer-aided geometric design, 2nd edn. Academic Press, New York
10. Feng J, Heng P, Wong T (1998) Accurate B-spline free-form deformation of polygonal objects. J Graph Tools 3(3):11–27
11. Feng J, Peng Q (2000) Accelerating accurate B-spline free-form deformation through polynomial interpolation. J Graph Tools 5(1):1–8
12. Gain J, Dodgson N (1999) Adaptive refinement and decimation under free-form deformation. Eurographics UK '99, Cambridge. http://people.cs.uct.ac.za/∼jgain/publications/EUUK99.ps.gz. Cited 17 July 2002
13. Griessmair J, Purgathofer W (1989) Deformation of solids with trivariate B-Splines. In: Hansmann W, Hopgood FRA, Strasser W (eds) Proceedings of Eurographics'89. Elsevier Science Publisher, North-Holland, pp 137–148.
14. Gudukbay U, Ozguc (1990) Free-form solid modeling using deformation. Comput Graph 14(3/4):491–500
15. Hsu W, Hughes J, Kaufman H (1992) Direct manipulation on free-form deformation. ACM Comput Graph (SIGGRAPH'92) 26(2):177–184
16. Karla P, Mangli A, Thalmann NM, Thalmann D (1992) Simulation of facial muscle actions based on rational free-form deformation. Comput Graph Forum (Eurographics '92) 2(3):59–69
17. Lamousin H, Waggenspack W (1994) NURBS-based free-form deformation. IEEE Comput Graph Appl 14(9):59–65
18. Liu W, Mann S (1997) An optimal algorithm for expanding the composition of polynomials. ACM Trans Graph 16(2):155–178
19. MacCracken R, Joy K (1996) Free-form deformation with lattice of arbitrary topology. In: Rushmeier H (ed) SIGGRAPH '96 Conference Proceedings. Addison-Wesley, Reading, Mass., pp 181–183
20. Moccozet L, Thalmann NM (1997) Dirichlet free-form deformations and their application to hand simulation. In: Thalman NM, Thalmann D (eds) Proceedings of ComputerAnimation'97. IEEE Computer Society Press, Geneva, pp 93–102
21. Nimscheck UM (1995) Rendering for free-form deformations Technical report 381, Computer Laboratory, University of Cambridge, Oct. 1995
22. Parry S (1996) Free-form deformations in a constructive solid geometry modeling system. PhD thesis, Department of Civil Engineering, Brigham Young University, Utah, Apr. 1986.
23. Sederberg T, Parry S (1986) Free-form deformation of solid geometric models. ACM Comput Graph (SIGGRAPH'96) 20(4):537–541
24. Vergeest SM (1991) CAD surface data exchange using STEP. Comput-Aided Des 23(1):269–281

Photographs of the authors and their biographies are given on the next page.

JIEQING FENG is an associate professor at the State Key Lab. of CAD&CG, Zhejiang University, People's Republic of China. He received his BSc degree in applied mathematics from the National University of Defense Technology in 1992, PhD in computer graphics from Zhejiang University in 1997. His research interests include space deformation, computer-aided geometric design and computer animation.

TOMOYUKI NISHITA is a professor in the Department of Information Science at University of Tokyo, Japan, since 1998. He taught at Fukuyama University from 1979 to 1998. He was an associate researcher in the Engineering Computer Graphics Laboratory at Brigham Young University from 1988 to 1989. His research interests center on computer graphics, including lighting models, hidden-surface removal, and antialiasing. Nishita received his BE, ME and PhD in engineering in 1971, 1973, and 1985 respectively, from Hiroshima University.

XIAOGANG JIN is a professor of the State Key Lab. of CAD&CG, Zhejiang University, People's Republic of China. He received his BSc degree in computer science in 1989, MSc and PhD degrees in applied mathematics in 1992 and 1995 respectively, all from Zhejiang University. His research interests include implicit surface modeling, space deformation, computer animation and realistic image synthesis.

QUNSHENG PENG is a Professor of computer graphics at Zhejiang University. His research interests include realistic image synthesis, computer animation, scientific data visualization, virtual reality, features modeling and parametric design. Professor Peng graduated from Beijing Mechanical College in 1970 and received a PhD from the Department of Computing Studies, University of East Anglia, in 1983. He is currently the Director of the State Key Lab. of CAD & CG at Zhejiang University and serves as a member of the editorial boards of several Chinese journals.