Journal of Information & Computational Science 9: 3 (2012) 583–593
Available at http://www.joics.com

Extended T-mesh and Data Structure for the Easy Computation of T-spline*

Hongwei Lin*, Ye Cai, Shuming Gao

State Key Lab. of CAD&CG, Zhejiang University, Hangzhou 310058, China

Abstract

T-spline overcomes the topological constraints of the control net of NURBS model successfully. However, the introduction of T-junctions, L-junctions and the isolated vertices in the T-mesh makes its topological structure very flexible. As a result, not only the T-mesh is hard to be represented, but the computation and local refinement of T-spline are difficult to be implemented as well. This hinders the studies and applications of T-splines in practice. In this paper, we develop the *extended T-mesh*, which can be represented in an *obj*-like format file, and converted into the *face-edge-vertex* data structure conveniently. With such data structure, the computation of T-splines can be made much easier. Furthermore, we develop a new local refinement algorithm, by virtue of the extended T-mesh. The new algorithm is easy to be implemented, by separating the local refinement into two procedures, the mesh refinement, and blending function refinement.

Keywords: T-splines; NURBS; Extended T-mesh; Local Refinement; Geometric Design; Iso-geometric Analysis

1 Introduction

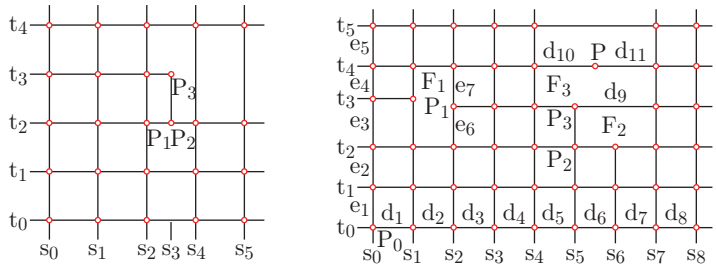
To overcome the topological constraints of the control net of NURBS model, Sederberg et al. invent the T-spline, which is a generalization of NURBS. The T-spline control mesh (called *T-mesh*) allows a row of control points to terminate, forming a T-junction, so that it is able of significantly reducing the number of superfluous control points in the NURBS model.

However, since T-spline breaks the topological constraints of NURBS model, the T-mesh can be made very flexible and complex. Taking two T-meshes presented in Ref. [1] as examples (Fig. 1), at the left T-mesh (Fig. 1 (a)), there is an L-junction at $\boldsymbol{P}_3$; at the right T-mesh (Fig. 1 (b)), there is an isolated vertex $\boldsymbol{P}$. These singular cases make the T-mesh difficult to be represented, and then the T-spline hard to be computed and refined. As a result, it hinders the wide-ranging applications of T-splines in practice.

*Project supported by the National Nature Science Foundation of China (No. 60970150, No. 60933008), and Zhejiang Provincial Natural Science Foundation of China (No. Y1090416).

*Corresponding author.
Email address: hwlin@cad.zju.edu.cn (Hongwei Lin).

March 2012



(a) L-junction at $\boldsymbol{P}_3$. (b) Isolated vertex at $\boldsymbol{P}$.

Fig. 1: Pre-images of two typical T-meshes in Ref. [1]

In this paper, we recommend using the *extended T-mesh* instead of the original one. We will show that, the extended T-mesh can be represented by a *obj*-like format file, and converted into a simple face-edge-vertex data structure easily. By the face-edge-vertex structure, the knot vectors of the blending function attaching to each vertex can be determined conveniently.

More importantly, relying on the extended T-mesh, we develop a new local refinement algorithm, which is the most important algorithm in T-spline. The new algorithm separates the local refinement into two procedures, mesh refinement and blending function refinement. By the mesh refinement, we can get the topological structure of the refined T-mesh before the blending function refinement and new control point generation. Thus, the new local refinement algorithm is much easier to be implemented than the *violation test* method proposed in Ref. [1].

1.1 Related Work

Invented by Sederberg et al., T-spline is first presented in [2], and improved in [1]. Moreover, Wang et al. develop the control point removal algorithm for T-splines [3]. Buffa et al. study the linear independence of the T-spline blending functions associated with some particular T-meshes [4]. Inspired by T-splines, He et al. develop the manifold T-spline, defined on arbitrary manifold domain of any topological type [5].

T-splines have been applied in some areas. Song et al. construct a T-spine volume for deformation [6]. Zheng et al. make use of T-spline to fit Z-map model [7]. Recently, T-spline shows its great potential in isogeometric analysis [8, 9].

2 The Extended T-mesh and Data Structure

T-spline is a Point-based spline (PB-spline). Each of its control point $\boldsymbol{P}_i$ corresponds to a blending function,

$$B_i(s,t) = N_{i0}(s)N_{i0}(t), \tag{1}$$

where, $N_{i0}(s)$ is a cubic B-spline basis function defined on the knot vector,

$$\boldsymbol{s}_i = [s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}], \tag{2}$$

and $N_{i0}(t)$ is defined on the knot vector,

$$\boldsymbol{t}_i = [t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}]. \tag{3}$$

The knot vectors $\boldsymbol{s}_i$ (2) and $\boldsymbol{t}_i$ (3) are inferred from the T-mesh neighborhood of $\boldsymbol{P}_i$ by the following rule [1]:

Rule 1. (s_{i2}, t_{i2}) are the knot coordinates of $\boldsymbol{P}_i$. Consider a ray in parameter space $\boldsymbol{R}(\alpha) = (s_{i2} + \alpha, t_{i2})$. Then s_{i3} and s_{i4} are the s coordinates of the first two $s-$edges intersected by the ray (not including the initial (s_{i2}, t_{i2})). By $s-$edge, we mean a vertical line segment of constant s (refer to Fig. 2); similarly, $t-$edge means a horizontal line segment of constant t. The other knots in $\boldsymbol{s}_i$ and $\boldsymbol{t}_i$ are found in like manner.

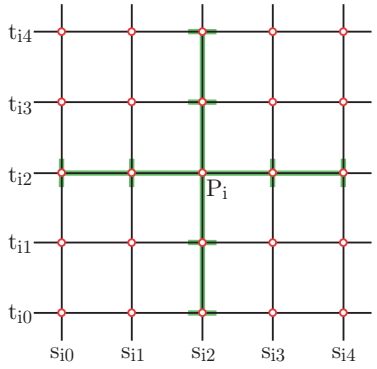


Fig. 2: Knot lines, $s-$edges, and $t-$edges (in green) for the blending function $B_i(s,t)$ [1]

The *extended T-mesh* retains the $s-$edges and $t-$edges for the blending functions at the T-junctions, L-junctions, and the isolated vertices. As illustrated in Fig. 3, there are four T-junctions at $\boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{P}_3, \boldsymbol{P}_4$, an L-junction at $\boldsymbol{P}_5$, and an isolated vertex at $\boldsymbol{P}_6$. By retaining the $s-$edges and $t-$edges of the blending functions at these vertices, we get the extended T-mesh in Fig. 3.

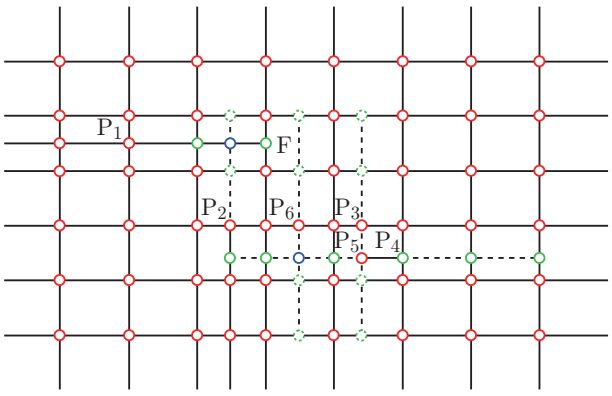


Fig. 3: Pre-image of an extended T-mesh

In the extended T-mesh, the retained $s-$edges and $t-$edges are named as *virtual edges*, which are illustrated in dashed lines in Fig. 3. On the other hand, the mesh edges in the original T-mesh are called *real edges*. Moreover, there are four kinds of vertices in the extended T-mesh, the *real vertex*, *s-vertex*, *t-vertex*, and *virtual vertex*. The *real vertex* is the vertex in the original T-mesh (displayed in red in Fig. 3), which corresponds to a control point in 3-dimensional space; the *s-vertex* (in green solid circle in Fig. 3) is the intersection between a real s-edge and a virtual edge; the *t-vertex* (in green dashed circle in Fig. 3) is the intersection between a real t-edge and a virtual edge; the virtual vertex (in blue in Fig. 3) is the intersection between two virtual edges.

One advantage of the extended T-mesh over the original T-mesh lies in that, all edges terminate at other edges, thus eliminating the L-junctions (at $\boldsymbol{P}_5$ in Fig. 3), and the isolated vertices ($\boldsymbol{P}_6$ in Fig. 3). Furthermore, the extended T-mesh can be changed as a 2-dimensional manifold, if we take the mesh face with n vertices, which is adjacent to T-junctions, as a n-gon. For example, the mesh face F in Fig. 3 can be taken as a 5-gon. Note that, the only function of the virtual vertex is to make the extended T-mesh be a 2-dimensional manifold. It has no effect in determining the knot vectors of the blending functions at real vertices.

As a 2-dimensional manifold, the extended T-mesh can be stored in an *obj*-like format file. We name it as *ebj* format, the *extended obj* format (Table 1).

Table 1: The *ebj* format file

r	s_1	t_1	x_1	y_1	z_1
r	s_2	t_2	x_2	y_2	z_2
	...	...	...		
s	s_i	t_i			
s	s_{i+1}	t_{i+1}			
	...	...	...		
t	s_j	t_j			
t	s_{j+1}	t_{j+1}			
	...	...	...		
v	s_k	t_k			
v	s_{k+1}	t_{k+1}			
	...	...	...		
f	n_1	n_2	n_3	n_4	
f	n_5	n_6	n_7	n_8	n_9
	...	...	...		

In the *ebj* format presented in Table 1, 'r' denotes the real vertex, which has knot coordinates (s,t), and the Descartes coordinates (x,y,z) in 3-dimensional space. 's', 't' and 'v' denote the s-vertex, t-vertex, and virtual vertex, respectively, which have only knot coordinates (s,t). 'f' represents the mesh face, followed by its vertex serials.

In implementation, the extended T-mesh stored in *ebj* format can be converted into the *face-edge-vertex* data structure (see Table 2). With the structure members *descar-coord*, *s-knots*, and *t-knots* in the *vertex* structure, the T-splines can be calculated conveniently.

3 Local Refinement Algorithm on the Extended T-mesh

In Ref. [1], the local refinement is performed by the *violation test* method, which tests whether there exist the following violations after the vertices are inserted into the T-mesh. That is,

- **Violation 1** A blending function is missing a knot dictated by Rule 1 for the current T-mesh.

Table 2: The *face-edge-vertex* data structure

```
// the face structure
struct face{
int ser; //the serial of the face
int vertices[]; //the serials of the vertices of the face
int edges[]; //the serials of the edges of the face
}

//the edge structure
struct edge{
int ser; //the serial of the edge
int sign[2]; //sign(1): real edge, or virtual edge
             sign(2): s-edge, or t-edge
int vertices[2]; //the serials of the vertices of the edge
int adj-face[2]; //the serials of the adjacent faces of the edge
}

//the vertex structure
struct vertex{
int ser; //the serial of the vertex
int sign; //real vertex, s-vertex, t-vertex, or virtual vertex
int adj-edges[]; //serials of the adjacent edges of the vertex
int adj-faces[]; //serials of the adjacent faces of the vertex
double knot-coord[2]; //the knot coordinates of the vertex
double descar-coord[3]; //the Descartes coordinates of the vertex
double s-knots[5]; //the knot vector along s-direction
double t-knots[5]; //the knot vector along t-direction
}
```

- **Violation 2** A blending function has a knot that is not dictated by Rule 1 for the current T-mesh.
- **Violation 3** A control point has no blending function associated with it.

And the topology phase of the violation test method for the local refinement consists of the four steps [1]:

1. Insert all desired control points into the T-mesh.
2. If any blending function is guilty of Violation 1, perform the necessary knot insertions into that blending function.
3. If any blending function is guilty of Violation 2, add an appropriate control point into the T-mesh.

4. Repeat Steps 2 and 3 until there are no more violations.

In fact, there are two procedures in the local refinement algorithm. One is the mesh refinement; the other is the blending function refinement. The above four steps in the violation test method integrate the two procedures in one circulation. In some cases, it will make the local refinement operation very complex, and hard to be implemented in computers. More seriously, it is easy to lose some violation tests, making the refinement incorrect.

Algorithm 1: Mesh Refinement

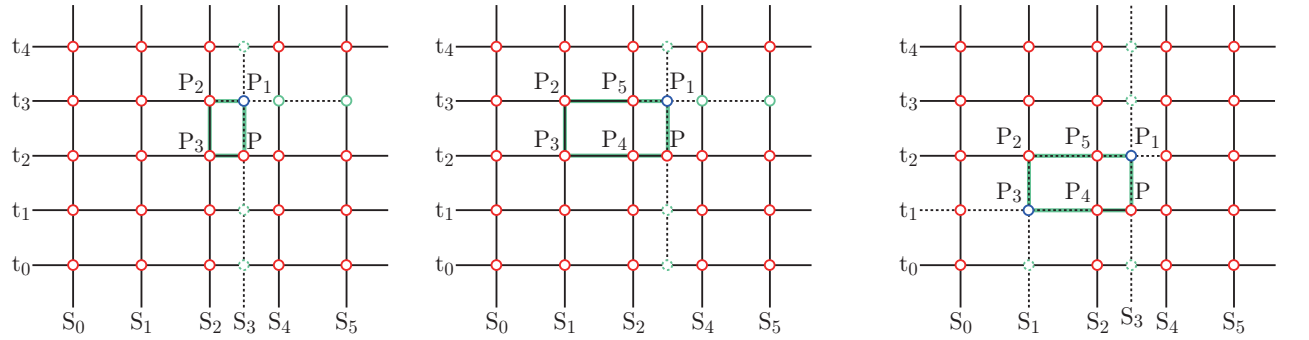
```
1  Insert the desired vertices on the real mesh edges, mark them as real vertices, and add them
   to the new real-vertex array R ;
2  while The new real-vertex array R is not empty do
3  |  P = R[1], and delete P from R; // P is the current vertex
4  |  Construct s-edges and t-edges of the basis function at the vertex P ;
5  |  for Each of the four directions at P, that is, d = inverse s-direction, s-direction, inverse
   |  t-direction, and t-direction do
   |  |  // Trace the two edges PP_1 and P_1P_2 of the basis function at P along
   |  |     the direction d
6  |  |  if The vertex P_1 is a real vertex then
7  |  |  |  Mark the edge PP_1 as a real edge ;
8  |  |  end
9  |  |  else if The vertex P_1 is a virtual vertex then
10 |  |  |  Mark P_1 as a real vertex, and add it to the array R ;
11 |  |  |  Mark the edge PP_1 as a real edge ;
12 |  |  end
13 |  |  else if The vertex P_1 is a s-vertex or t-vertex then
14 |  |  |  if P_2 is a real vertex then
15 |  |  |  |  Mark P_1 as a real vertex, and add it to the array R ;
16 |  |  |  |  Mark both edges PP_1 and P_1P_2 as real edges ;
17 |  |  |  end
18 |  |  end
19 |  end
20 end
```

To facilitate the local refinement algorithm on T-meshes, a desirable solution is to separate the two procedures, i.e., determining the topology of the refined T-mesh, before the blending function refinement and new control point generation. However, it is difficult to devise the explicit mesh refinement rule which results in the refined T-mesh same as that by the violation test method, because the topological structure of the T-mesh can be made very flexible.

Based on the fact that, the legal refined T-meshes which accommodate the same knot insertions are not unique, in this paper, we devise a simple mesh refinement algorithm (presented in Algorithm 1), by virtue of the extended T-mesh. Although it is possible for the new mesh refinement algorithm to insert more real vertices than that by the original violation test method, the new mesh refinement method is easy to be implemented, and the refined T-mesh it generates is guaranteed to be legal.

The mesh refinement algorithm (Algorithm 1) performs by checking the vertices on the knot edges of the blending function at the inserted vertex $\boldsymbol{P}$, along the four directions, s-direction, inverse s-direction, t-direction, and inverse t-direction. The vertices on these edges along the four directions can be classified into two categories, according to the connections between them and the inserted vertex $\boldsymbol{P}$, i.e., the one-ring adjacent vertices, which are adjacent to the vertex $\boldsymbol{P}$ directly, and the two-ring adjacent vertices.

We first consider the one-ring adjacent vertices, taking the extended T-meshes in Fig. 4 as examples, where the vertex $\boldsymbol{P}$ is the newly inserted real vertex. Suppose there is a virtual vertex $\boldsymbol{P}_1$ in the one-ring adjacent vertices, which is the intersection between the knot edge of the blending function at the real vertex $\boldsymbol{P}_2$ and that of the newly inserted vertex $\boldsymbol{P}$. Then, the newly inserted vertex $\boldsymbol{P}$, the virtual vertex $\boldsymbol{P}_1$, the first real vertex $\boldsymbol{P}_2$ along the current knot direction of $\boldsymbol{P}_1$, and the fourth vertex $\boldsymbol{P}_3$, form a *knot rectangle* (Fig. 4).



(a) $\boldsymbol{P}_3$ is a real vertex. (b) $\boldsymbol{P}_3$ is a real vertex, and $\boldsymbol{P}_5$ is a s-vertex. (c) Both $\boldsymbol{P}_3$ and $\boldsymbol{P}_5$ are virtual vertices.

Fig. 4: Inserting a real vertex at $\boldsymbol{P}_1$ does not change the legality of the insertion of $\boldsymbol{P}$, whether the vertex $\boldsymbol{P}_3$ is real. $\boldsymbol{P}\boldsymbol{P}_1\boldsymbol{P}_2\boldsymbol{P}_3$ constitute the *knot rectangle* (in green) in the inverse s-direction. The other knot rectangle along the s-direction can be constructed in the like manner.

According to the violation test method, it depends on the property of the vertex $\boldsymbol{P}_3$ whether there should be inserted a real vertex at $\boldsymbol{P}_1$. Specifically, if $\boldsymbol{P}_3$ is a real vertex, there should be inserted a real vertex at $\boldsymbol{P}_1$; if not, the vertex $\boldsymbol{P}_1$ should be retained as a virtual vertex. However, even the vertex $\boldsymbol{P}_3$ is not real, inserting a real vertex at $\boldsymbol{P}_1$ does not change the legality of the insertion of $\boldsymbol{P}$. Therefore, to simplify the mesh refinement algorithm, in Algorithm 1, if there exist virtual vertices in the one-ring vertices adjacent to the newly inserted vertex, we mark them as real vertices.

Moreover, the two-ring adjacent vertices are handled, taking the T-mesh in Fig. 4 (b) as an example, where $\boldsymbol{P}_1$ is the newly inserted real vertex. If at the two-ring vertices adjacent to $\boldsymbol{P}_1$, there is a real vertex $\boldsymbol{P}_2$, which is adjacent to the s-vertex $\boldsymbol{P}_5$, the blending function refinement at $\boldsymbol{P}_2$ will transfer to $\boldsymbol{P}_5$. So $\boldsymbol{P}_5$ should be a real vertex; otherwise, it will lose a real vertex at $\boldsymbol{P}_5$ according to the violation test.

In conclusion, after inserting a real vertex $\boldsymbol{P}$, the mesh refining principles in Algorithm 1 are:

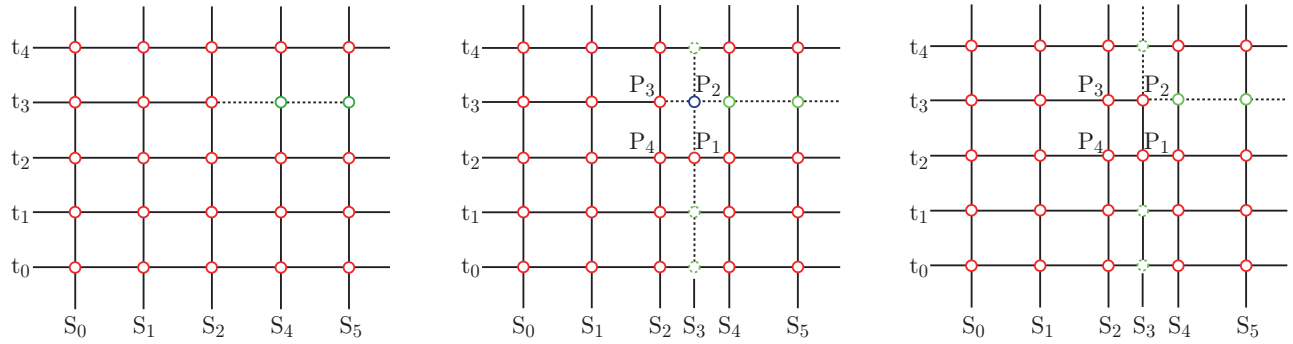
- If one of the one-ring vertices adjacent to $\boldsymbol{P}$ is a virtual vertex, change it as a real vertex;
- if one of the one-ring vertices adjacent to $\boldsymbol{P}$ is a s-vertex or t-vertex, and the two-ring vertex adjacent to the vertex is a real vertex, change the s-vertex (or t-vertex) as a real vertex.

Similar as the violation test method presented in Ref. [1], the mesh refinement algorithm (Algorithm 1) is always guaranteed to terminate, because the real vertex insertions must involve knot values that initially exist in the T-mesh, or that were added in the mesh refinement procedure. In the worst case, the algorithm would extend all partial rows of control points to cross the entire surface.

In general, it is possible that the mesh refinement algorithm proposed in this paper inserts more real vertices than the violation test method presented in Ref. [1]. However, in most typical cases, the refined mesh generated by Algorithm 1 is the same as that by the violation test method.

We illustrate Algorithm 1 with three examples (Figs. 5-7). In examples 1 and 2, the refined T-meshes by Algorithm 1 are the same as that by the violation test method [1]. In example 3, the refined T-mesh by Algorithm 1 is inserted more real vertices than that by the violation test method. Both are legal.

Example 1. The first example is the same as that in Ref. [1]. The original extended T-mesh is illustrated in Fig. 5 (a). We want to insert a new vertex at $\boldsymbol{P}_1$ (see Fig. 5 (b)). By checking its s-edges and t-edges, we find a virtual vertex at $\boldsymbol{P}_2$. It is marked as a real vertex, and the edge $\boldsymbol{P}_1\boldsymbol{P}_2$ is marked as a real edge. Next, by checking the s-edges and t-edges of the newly inserted vertex $\boldsymbol{P}_2$, we find that the vertex $\boldsymbol{P}_3$ is a real vertex. So the edge $\boldsymbol{P}_3\boldsymbol{P}_2$ should be marked as a real edge. The mesh refinement algorithm terminates till there is no unprocessed newly inserted real vertex. The legal refined mesh is demonstrated in Fig. 5 (c).



(a) Pre-image of the original extended Tmesh.

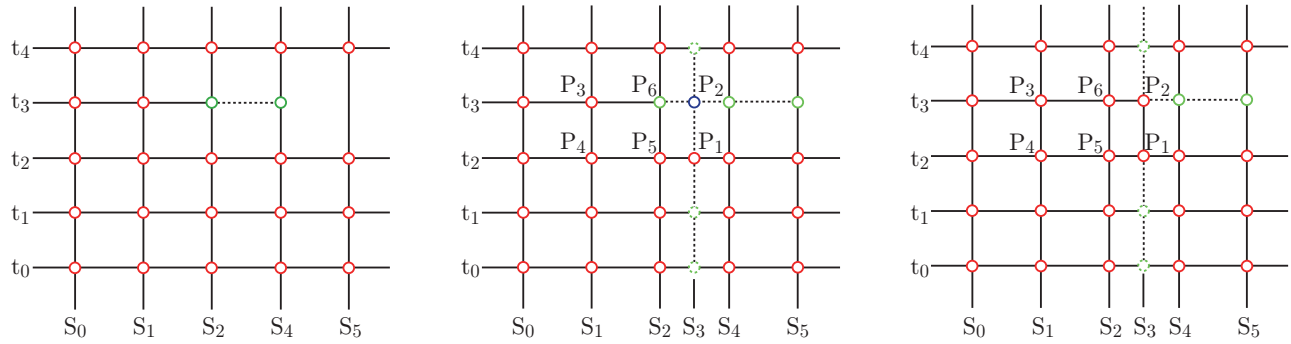
(b) Pre-image of the extended T-mesh after inserting $\boldsymbol{P}_1$.

(c) Pre-image of the legal refined T-mesh by Algorithm 1.

Fig. 5: Example 1: Insert a vertex at $\boldsymbol{P}_1$ by Algorithm 1

Example 2. The second example is illustrated in Fig. 6, where Fig. 6 (a) is the original extended T-mesh. As in Fig. 6 (b), We want to insert a vertex at $\boldsymbol{P}_1$. By tracing the knot edges of the blending function at the vertex $\boldsymbol{P}_1$, a virtual vertex $\boldsymbol{P}_2$ is found, which is marked as a real vertex. Meanwhile, the edge $\boldsymbol{P}_2\boldsymbol{P}_1$ is marked as a real edge. Next, we trace the knot edges of the blending function at the newly inserted real vertex $\boldsymbol{P}_2$. Among its two-ring adjacent vertices, there is a real vertex $\boldsymbol{P}_3$. So the vertex $\boldsymbol{P}_6$ is marked as a real vertex, and the two edges $\boldsymbol{P}_3\boldsymbol{P}_6$ and $\boldsymbol{P}_6\boldsymbol{P}_2$ are marked as real edges. By tracing the knot edges of the blending function at $\boldsymbol{P}_6$, it is not required to insert new vertex. The legal refined mesh is demonstrated in Fig. 6 (c).

Example 3. The third example presents the comparison between the refined T-meshes generated by Algorithm 1, and the violation test method, respectively, where the original T-mesh is illustrated in Fig. 7 (a). Fig. 7 (b) is the refined T-mesh after inserting the real vertex $\boldsymbol{P}_1$, using Algorithm 1, where two additional real vertices $\boldsymbol{P}_2$ and $\boldsymbol{P}_5$ are inserted. Fig. 7 (c) is the

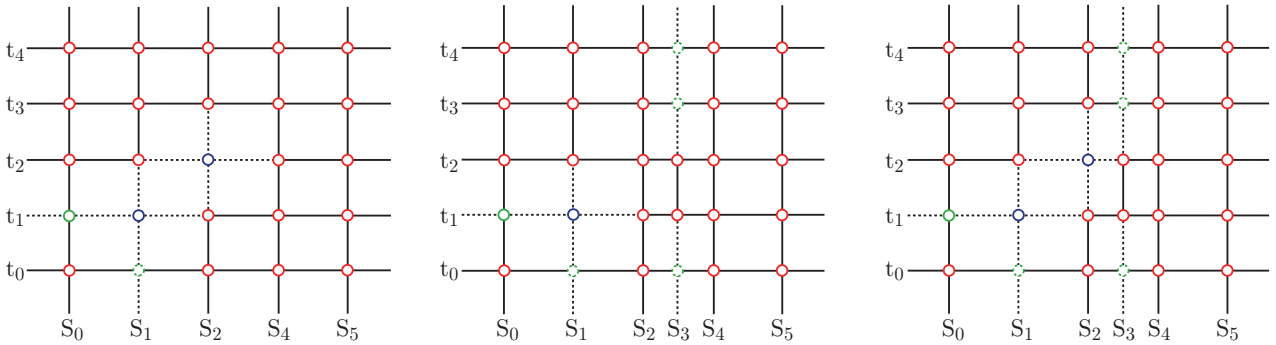


(a) Pre-image of the original extended Tmesh.

(b) Pre-image of the extended T-mesh after inserting $\boldsymbol{P}_1$.

(c) Pre-image of the legal refined T-mesh by Algorithm 1.

Fig. 6: Example 2: Insert a vertex at $\boldsymbol{P}_1$ by Algorithm 1



(a) Pre-image of the original extended Tmesh.

(b) Pre-image of the refined extended T-mesh after inserting $\boldsymbol{P}_1$ using Algorithm 1.

(c) Pre-image of the refined T-mesh after inserting $\boldsymbol{P}_1$ by the violation test method [1].

Fig. 7: Example 3: Comparison between the refined T-mesh by Algorithm 1 (b) and that by the violation test method (c) after inserting $\boldsymbol{P}_1$. The refined T-mesh by Algorithm 1 is inserted more real vertices than that by the violation test method.

refined T-mesh after inserting the real vertex $\boldsymbol{P}_1$, using the violation test method [1], where only one additional real vertex $\boldsymbol{P}_2$ is inserted for legality. This example shows that, it is possible for Algorithm 1 to insert more real vertices than the violation test method [1].

The mesh refinement algorithm (Algorithm 1) generates the legal refined T-mesh after inserting some desired real vertices. With the refined T-mesh in hand, the following blending function refinement and new control point generation is straightforward. We list it in Algorithm 2.

4 Conclusion

In this paper, we present the extended T-mesh, which can be stored in an *obj*-like format file, and converted into the *face-edge-vertex* data structure easily. With the data structure, the computation of the T-splines can be made much easier. Moreover, by the extended T-mesh, we develop a new local refinement algorithm, which is convenient to be implemented, by separating the local refinement into two procedures, mesh refinement and blending function refinement. The extended

Algorithm 2: Blending Function Refinement

// Denote the original extended T-mesh before mesh refinement as M_f, and the refined extended T-mesh as M_c

Extract the related real vertices $\boldsymbol{P}_i, i = 1, \cdots, n$ on M_f, which are at the s-edges and t-edges of the inserted vertices, and store them in a list R ;

Initialize each of the new control points at the vertices in R and the inserted vertices as {0, 0, 0} ;

for $i = 1 : n$ **do**

  $\boldsymbol{P} = R[i]$; // $\boldsymbol{P}$ is the current vertex.

  // Denote the blending function at the vertex $\boldsymbol{P}$ on M_f as $B(s,t)$; the control point at the vertex $\boldsymbol{P}$ on M_f as $\boldsymbol{C}$.

  Construct the knot vectors S and T along s-direction and t-direction, respectively, on the original mesh M_f, for the basis function $B(s,t)$;

  Insert the new knots in the refined mesh M_c to S and T, and refine the blending function $B(s,t)$, leading to,

$$B(s,t) = \sum_{i=1}^{k} c_i B_i([s_0^i, \cdots, s_4^i], [t_0^i, \cdots, t_4^i])(s,t);$$

  Add $c_i\boldsymbol{C}$ to the new control point at $(s_2^i, t_2^i), i = 1, 2, \cdots, k$;

end

T-mesh and the new local refinement algorithm on it make the processing of T-spline much easier. It will promote the wide-ranging applications of T-splines.

References

[1] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, T. Lyche. T-spline simplification and local refinement. ACM Transactions on Graphics (ACM Siggraph 2004), 23(3), 2004, 276-283

[2] T. W. Sederberg, J. Zheng, A. Bakenov, A. H. Nasri. T-splines and T-NURCCs. ACM Transactions on Graphics (ACM Siggraph 2003), 22(3), 2003, 477-484

[3] Y. Wang, J. Zheng. Control point removal algorithm for T-spline surfaces. In Myung-Soo Kim and Kenji Shimada, editors, Geometric Modeling and Processing - GMP 2006, Lecture Notes in Computer Science, vol. 4077, 2006, 385-396

[4] A. Buffa, D. Cho, G. Sangalli. Linear independence of the T-spline blending functions associated with some particular T-meshes. Computer Methods in Applied Mechanics and Engineering, 199 (23-24), 2010, 1437-1445

[5] Y. He, K. Wang, H. Wang, X. Gu, H. Qin. Manifold T-spline. In Myung-Soo Kim and Kenji Shimada, editors, Geometric Modeling and Processing - GMP 2006, Lecture Notes in Computer Science, vol. 4077, 2006, 409-422

[6] W. Song, X. Yang. Free-form deformation with weighted T-spline. The Visual Computer, 21, 2005, 139-151

[7] J. Zheng, Y. Wang, H. S. Seah. Adaptive T-spline surface fitting to Z-map models. In Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE'05, New York, NY, USA, 2005, 405-411

[8] M. R. Döfel, B. Jütler, B. Simeon. Adaptive isogeometric analysis by local h-refinement with T-splines. Computer Methods in Applied Mechanics and Engineering, Computational Geometry and Analysis, 199 (5-8), 2010, 264-275

[9] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S. Lipton, M. A. Scott, T. W. Sederberg. Isogeometric analysis using T-splines. Computer Methods in Applied Mechanics and Engineering, Computational Geometry and Analysis, 199 (5-8), 2010, 229-263