Visual Comput (2005) 21: 242–251
DOI 10.1007/s00371-005-0285-3

ORIGINAL ARTICLE

Jianmin Zheng
Yiyu Cai

Making Doo-Sabin surface interpolation always work over irregular meshes

Published online: 12 May 2005

Jianmin Zheng
School of Computer Engineering,
Nanyang Technological University,
50 Nanyang Avenue, Singapore 639798
E-mail: asjmzheng@ntu.edu.sg
Tel.: +65-67906257

Yiyu Cai
School of Mechanical and Aerospace
Engineering, Nanyang Technological
University, 50 Nanyang Avenue,
Singapore 639798

Abstract This paper presents a reliable method for constructing a control mesh whose Doo-Sabin subdivision surface interpolates the vertices of a given mesh with arbitrary topology. The method improves on existing techniques in two respects: (1) it is guaranteed to always work for meshes of arbitrary topological type; (2) there is no need to solve a system of linear equations to obtain the control points. Extensions to include normal vector interpolation and/or shape adjustment are also discussed.

Keywords arbitrary topology · subdivision surfaces · interpolation · normal condition · shape control

1 Introduction

Constructing smooth surfaces of arbitrary topology is an important but difficult task in computer graphics and geometric modeling [13]. As an industry standard, non-uniform rational B-spline surfaces (NURBS) suffer from the regularity requirement of the control meshes. Recursive subdivision was introduced as an efficient technique to overcome this limitation. Starting from an initial polyhedral mesh, subdivision recursively refines the mesh by adding new vertices, edges, and faces. As the number of this processing step goes to infinity, the refined meshes finally converge to a smooth surface.

Among various subdivision schemes, the Catmull-Clark algorithm [2] and the Doo-Sabin algorithm [4] are of particular interest. This is because they are generalizations of cubic and quadratic B-splines, respectively. When an initial mesh is a regular mesh, i.e., each face in the mesh is four-sided and each vertex has valence four, the Catmull-Clark algorithm and Doo-Sabin algorithm produce a tensor-product bicubic and biquadratic B-spline surface; otherwise, when the initial mesh happens to be an irregular one, then the limit surface generated by these two subdivision schemes comprises a sequence of Bézier patches, which meet with at least tangent plane continuity. Analogous to B-spline surfaces, Catmull-Clark and Doo-Sabin surfaces generally do not pass through the vertices of the initial mesh. Instead, they just approximate the mesh.

In an interactive free-form surface design environment, control points are often used to define and modify the shape. Although the control points in approximate schemes give some guidance to the change in the surface shape, the designer is required to invoke his or her experience to achieve the desired shape. An inexperienced user would prefer to specify points on the surface directly. Therefore, it is more desirable to be able to interpolate points lying on the final surface. To force the limit surface to go through a particular set of control points, modifications of the Catmull-Clark and Doo-Sabin schemes are needed. Nasri [8] presents such a mod-

ification for the Doo-Sabin algorithm. Specifying normal vectors at those interpolation points is also possible [9]. Halstead et al. [7] propose an interpolation scheme using Catmull-Clark surfaces that minimizes a certain fairness measure. Both methods require the construction of a linear constraint on the control points for each interpolation point and thus the establishment of a system of linear equations. The initial mesh for the subdivision surface can be obtained by solving the equations. However, as pointed out in both Halstead et al. [7] and Zorin et al. [14], it is possible for the coefficient matrix in the linear system to be singular, and it is unclear under what conditions the linear system is soluble.

This paper proposes a reliable method for computing the control mesh for the Doo-Sabin subdivision surface that interpolates the specified points on a given mesh. Nasri's approach is modified so that the linear system to be solved for the interpolation constraints is *always* soluble. Moreover, instead of constructing a linear system and solving the equations, our method performs an updating iteration directly applied to each control vertex. This has the advantage that the programming task becomes very simple, and more importantly, when the number of points of the input mesh is large, the memory storage will not become a problem. Therefore, the importance of the new method is twofold: first, the method provably always works in all situations. This is of theoretical interest. Second, the computation is simple and significantly faster compared to previous methods. This therefore makes the method suitable for interactive shape modification in practical applications.

The paper is organized as follows: in Sect. 2, we review Doo-Sabin subdivision surfaces and a related interpolation algorithm. Section 3 then describes the construction of our new method. The incorporation of normal vector interpolation and shape adjustment into the new method is discussed in Sect. 4. Section 5 gives the conclusion.

2 Doo-Sabin subdivision surfaces

The Doo-Sabin subdivision algorithm generalizes the subdivision rules for biquadratic B-splines to an arbitrary closed mesh. That is, in a configuration of faces, edges, and vertices [10], each vertex is called the control point, which has a position in space; each edge is a line segment bounded by two vertices; each face is a loop of edges, each of which shares a vertex with the next around the loop; and each edge is shared by exactly two faces. Denser meshes are generated repeatedly from the previous ones. In each subdivision step, for every face with m vertices $V_1, \cdots, V_m$, the new corresponding vertices $V'_1, \cdots, V'_m$ are computed by $V'_i = \sum_{j=1}^{m} \alpha_{ij} V_j$, where

$$\alpha_{ij} = \begin{cases} \dfrac{m+5}{4m}, & i = j \\ \dfrac{3+2\cos(2(i-j)\pi/m)}{4m}, & i \neq j \end{cases} \tag{1}$$

Then a new face of type F is created by connecting $V'_1, \cdots, V'_m$ to replace the old one. In addition to the type-F face, a new four-sided face of type E is formed for every edge of the old mesh by connecting the images of the edge endpoints on each of the faces sharing the edge; also, a new face of type V is formed for every vertex of the old mesh by connecting the images of the vertex on each of the faces surrounding the vertex. Figure 1 illustrates these types of new faces. The Doo-Sabin surface is the limit of the sequence of such refined meshes. An example of a Doo-Sabin surface is shown in Fig. 3, where (a) is the initial mesh, (b) is the mesh after one Doo-Sabin subdivision, and (c) is the limit surface.

The Doo-Sabin algorithm has the following properties:

- The surfaces can be of arbitrary genus, since the refinement rules can be applied to a mesh of arbitrary topological type.
- After the first subdivision step, every vertex of the new mesh will have valence four, and thus, the number of non-four-sided faces will remain constant and equal to the number of non-four-sided faces and non-four-valent vertices in the initial configuration. Beyond these faces, the remaining faces present a rectangular topology with four-sided faces and four-valent vertices; thus, they will converge to a biquadratic B-spline.
- The technique using the discrete Fourier transform and eigenanalysis [4] shows that the Doo-Sabin surface is tangent-plane continuous everywhere and interpolates the centroids of all faces at any step of subdivision.
- Note that faces in Doo-Sabin meshes are not necessarily planar. However, if a face is planar, its corres-

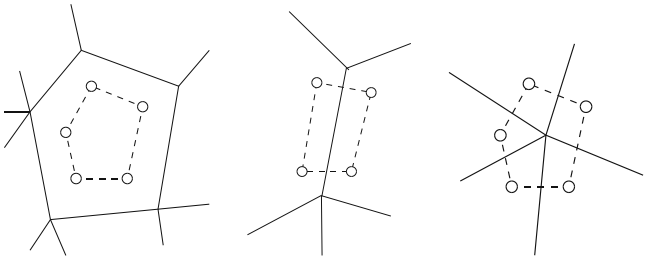

Fig. 1. Three types of faces in the Doo-Sabin scheme

ponding new type-F face lies on this plane. As a consequence, a planar face and the tangent plane of the Doo-Sabin surface at the centroid of the face coincide.

The interpolation problem is stated as follows: given a polyhedron $\widehat{\mathcal{I}}$ with a set of vertices $\mathbb{I} = \{I_i : i = 1, \cdots, N\}$ and a subset $\mathbb{SI}$ of $\mathbb{I}$, we want to construct a smooth surface interpolating the vertices in the subset $\mathbb{SI}$.

Based on the properties of the Doo-Sabin subdivision algorithm, Nasri [8] developed an interpolation approach using Doo-Sabin subdivision. The basic idea is to construct an auxiliary polyhedron $\widehat{\mathcal{M}}^0$ with a set of vertices $\mathbb{M} = \{V_i : i = 1, \cdots, N\}$ having the same topology as $\widehat{\mathcal{I}}$, where the vertices V_i are to be determined. The sets $\mathbb{I}$ and $\mathbb{M}$ have a one-to-one correspondence. Every interpolation vertex I_i in the subset $\mathbb{SI}$ maps to a vertex V_i in $\mathbb{M}$ such that I_i is the centroid of the type-V face corresponding to V_i after the first subdivision of $\widehat{\mathcal{M}}^0$. This ensures that the limit surface interpolates I_i. Obviously, the above requirement introduces a linear constraint between I_i and the elements of the set $\mathbb{M}$. For each non-interpolation vertex I_j in $\mathbb{I}\backslash\mathbb{SI}$ – the complement set of $\mathbb{SI}$ – it also corresponds to a certain V_j in $\mathbb{M}$. Geometrically, V_j can simply be set to be I_j. In this way, a system of simultaneous vector-valued linear equations is derived: $\mathbb{I} = [C]\mathbb{M}$, where $[C]$ is the coefficient matrix. Solving this linear system gives the desired vertex set $\mathbb{M}$, thus accomplishing the interpolation goal.

3 New interpolation method

There are limitations to Nasri's approach [8]; for example, it involves solving a system of linear equations, and it is unclear under what conditions the coefficient matrix in the linear system is nonsingular [7] and [14]. Even if the matrix is well-conditioned, the complexity of solving a huge system of linear equations prevents the use of the method for interactive modeling. In this section, we present a modification to overcome these deficiencies.

3.1 Build the interpolation constraints

We first rewrite the Doo-Sabin refinement formula:

$$V_i' = \frac{V_i + A}{2} + \Delta_i \tag{2}$$

where $A = \frac{1}{m}\sum_{j=1}^{m} V_j$ is the centroid of the face with vertices $\{V_1, \cdots, V_m\}$, and

$$\Delta_i = \sum_{j=1}^{m} \frac{1 + 2\cos\left(\frac{2(i-j)\pi}{m}\right)}{4m}(V_j - V_i) \tag{3}$$

is a perturbation term which is introduced to make the shape smoother [3]. Nasri [8] applied this refinement rule to an auxiliary mesh $\widehat{\mathcal{M}}^0$ to generate a refined mesh $\widehat{\mathcal{M}}^1$, and made the interpolation points coincide with the centroids of some faces of $\widehat{\mathcal{M}}^1$. Note that it is the latter that guarantees the interpolation property no matter how $\widehat{\mathcal{M}}^1$ is created. Instead of using the Doo-Sabin refinement rule (Eq. 2) in the first subdivision step, we use a modified rule for the first subdivision:

$$V_i' = \frac{V_i + A}{2}. \tag{4}$$

Now let's use this rule to establish the constraints for the auxiliary mesh $\widehat{\mathcal{M}}^0$. Without loss of generality, consider one vertex V in $\widehat{\mathcal{M}}^0$ and its neighborhood, corresponding to an interpolation point I. Assume there are m faces meeting at V. Refer to Fig. 2 for the notation being used. We relabel the vertices in the neighborhood of V by E_1, $V_1^1, V_2^1, \cdots, V_{m_1-3}^1$, E_2, $V_1^2, \cdots, V_{m_2-3}^2$,$E_3, \cdots, E_m$,$V_1^m, \cdots, V_{m_m-3}^m$, where E_i is the point that shares an edge with V, and m_i is the number of vertices of the ith face around V, whose vertices are V, E_i, $V_1^i, \cdots, V_{m_i-3}^i$, E_{i+1}. After one subdivision step using the modified formula (Eq. 4), we obtain a new mesh $\widehat{\mathcal{M}}^1$, which we call the *first refined mesh*. If we denote by W_i the image of V on the ith face surrounding V, then

$$W_i = \left(\frac{1}{2} + \frac{1}{2m_i}\right)V + \frac{E_i + E_{i+1}}{2m_i} + \frac{1}{2m_i}\sum_{j=1}^{m_i-3} V_j^i. \tag{5}$$

From $\widehat{\mathcal{M}}^1$, the Doo-Sabin subdivision rule is then used for the subsequent refinements to make the limit shape smooth. The type-V face, with vertices $W_1, \cdots, W_m$ in $\widehat{\mathcal{M}}^1$, will shrink and eventually converge to its centroid. So

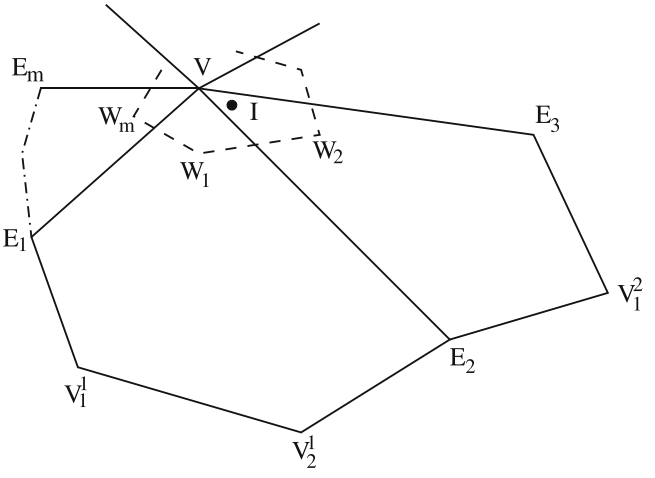

Fig. 2. The neighborhood around a vertex V

the requirement of interpolating vertex I imposes a simple linear constraint: $I = \frac{1}{m}\sum_{i=1}^{m} W_i$. Substituting Eq. 5 into this equation leads to

$$\left(m + \sum_{i=1}^{m} \frac{1}{m_i}\right) V + \sum_{i=1}^{m} \left(\frac{1}{m_i} + \frac{1}{m_{i-1}}\right) E_i + \sum_{i=1}^{m} \frac{1}{m_i} \sum_{j=1}^{m_i-3} V_j^i = 2mI. \tag{6}$$

Grouping the equations for all interpolation vertices and using the simple setting mentioned in Sect. 2 for the remaining non-interpolation vertices, we arrive at a system of linear equations with N equations and N unknowns. These are constraints on the auxiliary mesh $\widehat{\mathcal{M}}^0$.

3.2 Solve for the auxiliary mesh

Unlike Nasri's approach, the linear system we obtain is always nonsingular. In fact, examine Eq. 6, where the interpolation vertex I corresponds to the control point V. The sum of the coefficients of all E_i and V_j^i is $\sum_{i=1}^{m}\left(1 - \frac{1}{m_i}\right) < m$ and thus is less than the coefficient of V. Thus, the linear system is diagonally dominant. It can be easily and robustly solved by a direct method such as Gaussian elimination.

As an alternative to direct methods, the Gauss-Seidel iterative method is also often used, especially when the coefficient matrix is sparse, large, and diagonally dominant [6]. The convergence is guaranteed by the diagonal dominance. Therefore, the Gauss-Seidel method is quite preferable in our case. In particular, the iteration can be written as

$$V = 2mC\, I - C\sum_{i=1}^{m}\left(\frac{1}{m_i} + \frac{1}{m_{i-1}}\right) E_i - C\sum_{i=1}^{m}\frac{1}{m_i}\sum_{j=1}^{m_i-3} V_j^i \tag{7}$$

where $C = 1 \Big/ \left(m + \sum_{i=1}^{m}\frac{1}{m_i}\right)$. This iteration is convergent because the linear system (Eq. 6) is diagonally dominant. The initial setting for the iteration is naturally chosen to be the given polyhedron $\widehat{\mathcal{I}}$. Once all updates for vertices are within a prescribed range or the iterative number exceeds a predetermined one, the iteration stops, and the current vertices define the auxiliary mesh.

One important fact of the Gauss-Seidel method is that during each iteration, it uses updated values as soon as they are available. Therefore, in Eq. 7, the most recent vertices are used. This implies that one does not need to keep the old values in the course of programming. On the other hand, this inherently sequential characteristic of the Gauss-Seidel method makes the iteration depend on the order in which the equations are updated. In our current implementation, we examine for simplicity the equations according to the order in which the vertices are stored. However, if one wants an order-independent iteration or wants the updates to be done in parallel, the Gauss-Seidel method can be replaced by the Jacobi method, which has a similar iterative formula (Eq. 7) but uses all values from the previous iteration.

No matter whether the Gauss-Seidel or the Jacobi method is adopted, the iterative approach (Eq. 7) is attractive. It makes the programming task simple. The user can ignore the underlying mathematics. The right side of the Eq. 7 is just a linear combination of the vertices in the local neighborhood of V, which is similar to the way of computing new vertices in subdivision algorithms.

3.3 Adjust the first refined mesh

From the initial input polyhedron, the preceding section creates an auxiliary polyhedron. Performing the modified subdivision (Eq. 4) on it yields the first refined mesh $\widehat{\mathcal{M}}^1$. The interpolation points are assumed to lie on the centroids of corresponding type-V faces of $\widehat{\mathcal{M}}^1$. However, this assumption does not always hold due to the fact that the iterative method just gives approximate solutions. It is therefore necessary to adjust the vertices of $\widehat{\mathcal{M}}^1$ so that the centroids match the interpolation points.

Observe that all faces in $\widehat{\mathcal{M}}^1$ corresponding to the interpolation points are separated. Thus, we only need to study one interpolation point and its corresponding type-V face. Assume the interpolation point is I and the type-V face consists of vertices $W_1, \cdots, W_m$. Now we give each vertex W_i a perturbation ε_i such that the perturbed face has I as its centroid, i.e.,

$$I = \frac{1}{m}\sum_{i=1}^{m}(W_i + \varepsilon_i)\,. \tag{8}$$

Meanwhile, we hope the amount of perturbation is as small as possible. We minimize the following objective function:

$$\sum_{i=1}^{m} \varepsilon_i \cdot \varepsilon_i \to \min. \tag{9}$$

This gives a solution $\varepsilon_i = \dfrac{1}{m}\sum_{i=1}^{m}(I - W_i) = I - \dfrac{1}{m}\sum_{i=1}^{m} W_i$.

Note that vertices $W_1, \cdots, W_m$ may be nonplanar. To control the tangent plane easily, we can further force the perturbed vertices to lie on a plane. To do so, we choose a plane that comes close to all the vertices. The unit normal vector $\mathbf{n}$ can be computed [5]:

$$\mathbf{n} = \sum_{i=1}^{m} W_i \times W_{i+1} \Big/ \left\| \sum_{i=1}^{m} W_i \times W_{i+1} \right\|. \tag{10}$$

Thus, we need to add the following constraints:

$$(W_i + \varepsilon_i - I) \cdot \mathbf{n} = 0, \quad i = 1, \cdots, m. \tag{11}$$

To solve the constrained minimization problem (Eq. 9) with Eqs. 8 and 11, we introduce Lagrange's multipliers λ and μ_i ($i = 1, \cdots, m$), where λ is a vector, and all μ_i are scalar numbers. Including the constraints in the objective function, we obtain a single unconstrained objective function:

$$f = \sum_{i=1}^{m} \varepsilon_i \cdot \varepsilon_i + \lambda \cdot \sum_{i=1}^{m} (\varepsilon_i + W_i - I) + \sum_{i=1}^{m} \mu_i (W_i + \varepsilon_i - I) \cdot \mathbf{n}. \tag{12}$$

Taking the partial derivatives of f with respect to ε_k and setting the derivatives to zero yield

$$\frac{\partial f}{\partial \varepsilon_k} = 2\varepsilon_k + \lambda + \mu_k \mathbf{n} = 0, \quad k = 1, \cdots, m, \tag{13}$$

which gives

$$\varepsilon_k = -\frac{1}{2}(\lambda + \mu_k \mathbf{n}). \tag{14}$$

Substituting Eq. 14 into Eq. 11 yields

$$\mu_k = 2(W_k - I) \cdot \mathbf{n} - \lambda \cdot \mathbf{n}. \tag{15}$$

Further substituting Eqs. 14 and 15 into Eq. 8 leads to

$$\lambda - (\lambda \cdot \mathbf{n})\mathbf{n} = \frac{2}{m} \left[\sum_{i=1}^{m} (W_i - I) - \left(\sum_{i=1}^{m} (W_i - I) \cdot \mathbf{n} \right) \mathbf{n} \right]. \tag{16}$$

We finally obtain the perturbation vector

$$\begin{aligned} \varepsilon_k &= -\frac{1}{2} \left(\lambda + 2 \left[(W_k - I) \cdot \mathbf{n} \right] \mathbf{n} - (\lambda \cdot \mathbf{n}) \mathbf{n} \right) \\ &= ((W_k - I) \cdot \mathbf{n}) \mathbf{n} - \frac{1}{2} (\lambda - (\lambda \cdot \mathbf{n}) \mathbf{n}) \\ &= \left[\left(\frac{1}{m} \sum_{i=1}^{m} W_i - W_k \right) \cdot \mathbf{n} \right] \mathbf{n} + \left(I - \frac{1}{m} \sum_{i=1}^{m} W_i \right). \end{aligned} \tag{17}$$

In the right side of the above equation, the second term is just the simple optimal solution without the coplanar constraint, and the first term is used to correct the normal vector. When all the vertices are on a plane, this correction becomes zero.

3.4 Algorithm

Based on the analysis above, our interpolation algorithm first constructs an auxiliary polyhedron $\widehat{\mathcal{M}}^0$ from the input polyhedron $\widehat{\mathcal{I}}$ by an iterative scheme. After that, the modified subdivision is applied to $\widehat{\mathcal{M}}^0$, generating the first refined mesh $\widehat{\mathcal{M}}^1$. Next, the vertices in $\widehat{\mathcal{M}}^1$ are adjusted. Finally, the application of the Doo-Sabin subdivision algorithm to the perturbed $\widehat{\mathcal{M}}^1$ results in a smooth surface that interpolates the input vertices.

It should be clarified that the auxiliary polyhedron $\widehat{\mathcal{M}}^0$ is not the Doo-Sabin mesh in a strict sense, since our first refinement rule is different from the Doo-Sabin rule; however, $\widehat{\mathcal{M}}^1$ is the Doo-Sabin mesh. This is similar to the situation in Brunet [1] and Nasri [9].

The algorithm to generate the Doo-Sabin mesh $\widehat{\mathcal{M}}^1$ is described as follows:

```
Input:
    a polyhedron Î,
    a subset 𝕀 indicating the vertices to be interpolated,
    ITE_MAX – the maximum number of iterations,
    EPS – prescribed tolerance
Procedure:
step 1.copy polyhedron Î to create M̂^0 /* initializing */
step 2./* compute the auxiliary mesh */
      for (i = 0; i < ITE_MAX; i++) {
          set PASS = TRUE
          for (each vertex I in 𝕀) {
              use Eq. 7 to update the corresponding V
              if (the update for V > EPS) set PASS =
    FALSE
          }
          if (PASS == TRUE) break;
      }
step 3./* perform the first subdivision */
      apply Eq. 4 to M̂^0 to generate M̂^1.
step 4./* adjust the first refined mesh */
      for (each I in 𝕀) {
          find the corresponding type-V face f in M̂^1
          compute an average unit normal vector n using
    Eq. 10
          compute W_c = (1/m) Σ_{i=1}^{m} W_i
          for (each vertex W_k in face f) {
              compute the perturbation ε_k by
                    ε_k = [(W_c − W_k) · n]n + (V − W_c)
              W_k = W_k + ε_k /* update the vertex */
          }
      }
step 5.output M̂^1.
```

The algorithm has been implemented using C++ on the MS Windows platform. Since the first refined mesh $\widehat{\mathcal{M}}^1$ is to be perturbed in step 4, the maximum number *ITE_MAX* in step 2 can be chosen to be a fairly small number. In our implementation, it is chosen to be 5, which gives a pretty good approximation in most cases.

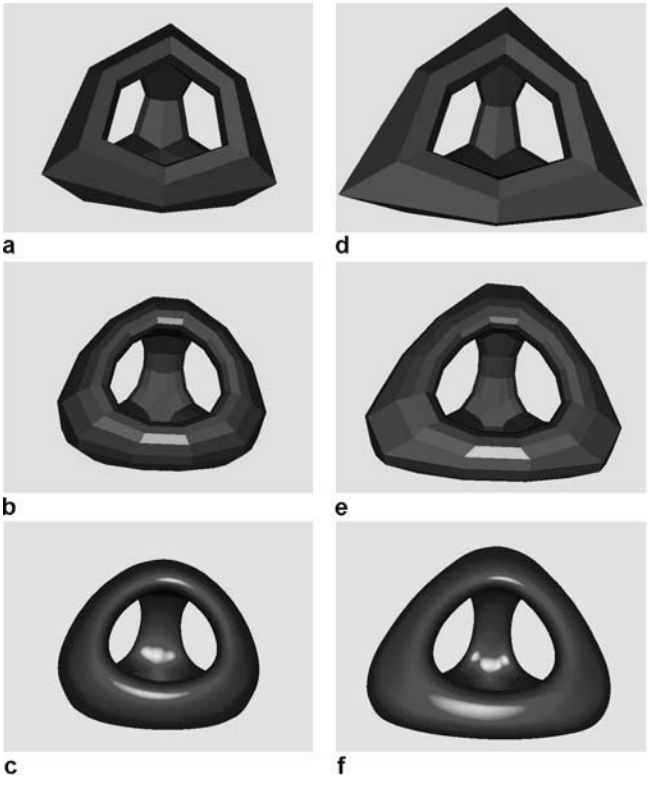

Fig. 3. Approximation versus interpolation

The right column of Fig. 3 illustrates the process of the interpolation algorithm. To interpolate the vertices of the input mesh given in Fig. 3(a), an auxiliary mesh is computed, which is shown in Fig. 3(d). The perturbed first refined mesh $\widehat{\mathcal{M}}^1$ and the limit surface are shown in Figs. 3(e) and (f), respectively.

3.5 Examples

We present two examples to demonstrate the effectiveness of the new algorithm. Nasri's algorithm was implemented for comparison. In the implementation, Gaussian elimination was used as the solver of linear systems. We ran both algorithms on a 1.6 GHz Intel Pentium 4 with 512 MB of RAM.

Figure 4(a) shows a model that contains 562 vertices. This model was taken from the repository of "The Princeton Shape Benchmark" [11]. We wanted to construct a smooth surface interpolating all of the vertices of the model. Our algorithm took about 4.9 ms to create the auxiliary mesh in Fig. 4(b) and the perturbed first refined mesh was created within 37 ms. The auxiliary mesh was obtained after only three iterations with an iterative termination threshold of 0.3%. The interpolating limit surface is shown in Fig. 4(c). Nasri's algorithm took 1.4 s to generate the auxiliary mesh shown in Fig. 4(d). Its corresponding limit surface is shown in Fig. 4(e).

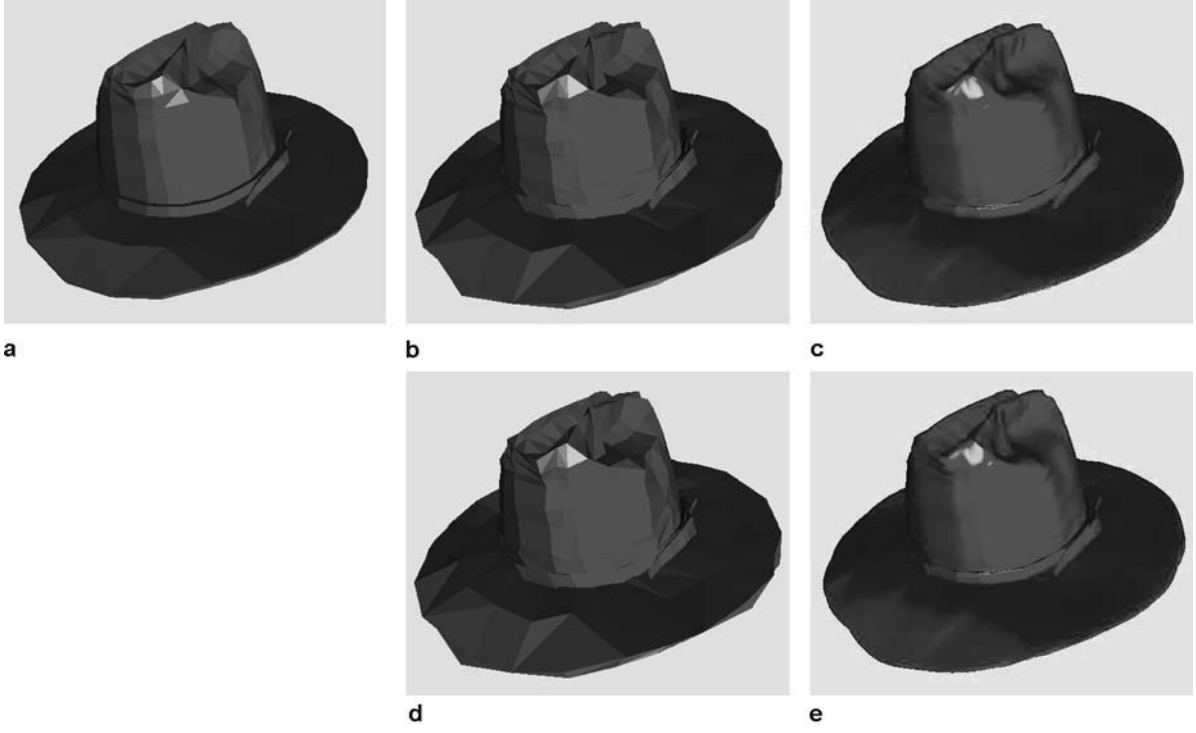

Fig. 4. Example 1: the new method versus Nasri's method

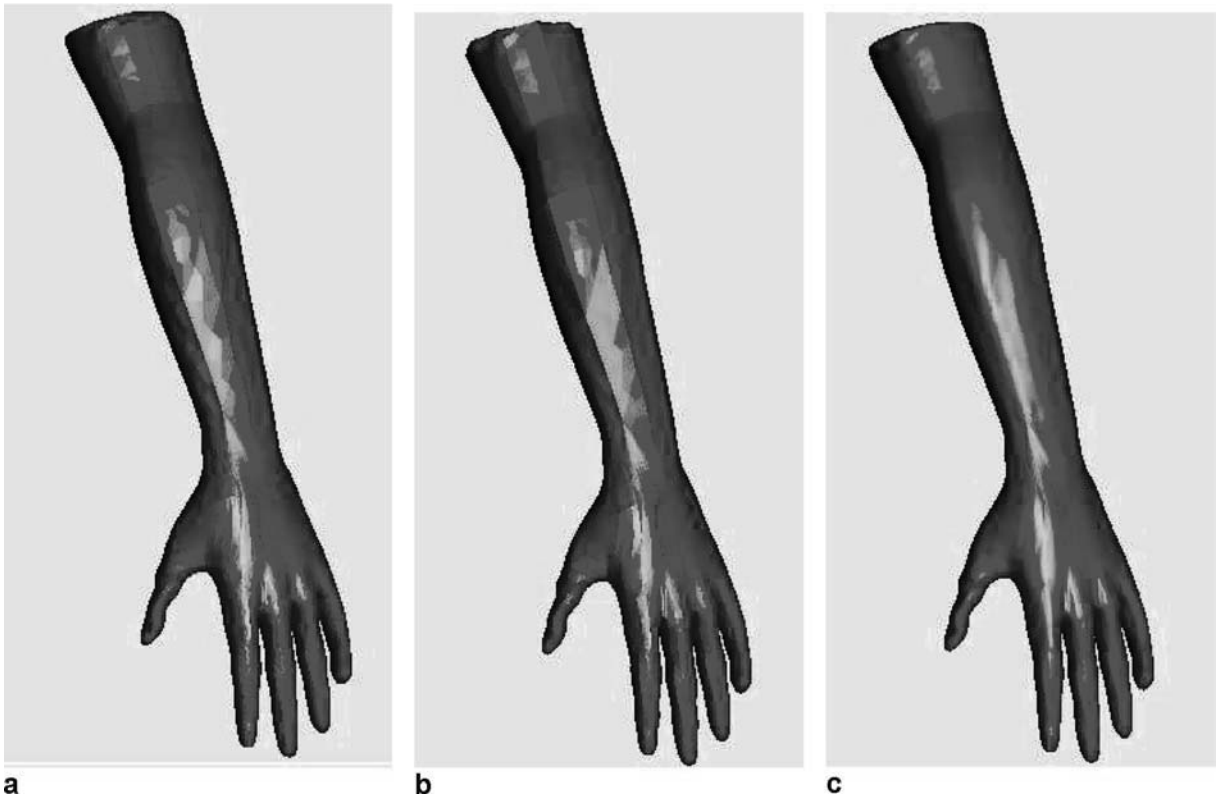

Fig. 5. Example 2: interpolating a model containing 11 540 vertices with the new method

Figure 5 shows another example. The input mesh in Fig. 5(a) has 11 540 vertices. Our algorithm took only 79 ms to generate the auxiliary mesh, which is shown in Fig. 5(b). The auxiliary mesh was obtained after two iterations. In the second iteration, the maximal update for the vertices was less than 0.3% of the model's size. The algorithm took another 491 ms to create the perturbed first refined mesh $\widehat{\mathcal{M}}^1$, and the limit surface, which interpolates the initial 11 540 vertices, is shown in Fig. 5(c). We also ran Nasri's method on this input mesh but failed to create the auxiliary mesh because of the large size of the linear system.

Remark 1 For Nasri's method, Halstead et al. [7] indicated that the linear system could possibly be singular. At this moment, we do not have such examples, but at least we can say that so far it is still unclear in which situations the coefficient matrix in the linear system of Nasri's method is nonsingular [14]. By contrast, the linear system of our method is provably well-conditioned in all situations. In practice, the superiority of our method is even more obvious. If the maximum iteration number is set to a fixed number, the computational complexity of our algorithm is linear in the number of vertices to be interpolated. This may explain why our method is significantly faster than Nasri's method. In the case where there is a large number of vertices to be interpolated, our method can still quickly generate the auxiliary mesh, but Nasri's method may fail in practice even though its linear system is soluble theoretically. The above second example is just such an example.

Remark 2 The accuracy of our approach is also worth mentioning. In our algorithm, the vertices of the auxiliary mesh are obtained by the iteration formula given by Eq. 7, which is derived from a diagonally dominant linear system. The diagonal dominance ensures convergence and makes the computation numerically stable. The above examples have shown that a few iterations can achieve good approximation to the accurate solution of the linear system. Moreover, the accurate solution is actually not required in our method because the perturbation step will later compensate for the approximation error and any accuracy loss in performing the iterations. Once the auxiliary mesh is obtained, the subdivision surface is uniquely determined, which *exactly* interpolates the given interpolation points. To compute the subdivision surface, we can recursively subdivide the mesh until the refined mesh approximates the limit surface within some prescribed tolerance. An alternative approach is to develop a direct and exact evaluation for Doo-Sabin subdivision surfaces as suggested by Stam in his paper for the exact evaluation of Catmull-Clark subdivision surfaces [12]. The evaluation is

then applied to the perturbed first refined mesh, which is a Doo-Sabin subdivision mesh. In this way, any point on the limit surface can be exactly computed. Therefore, this approach could be used for high-precision interpolation.

4 Normal interpolation and shape adjustment

In free-form shape design and modeling, the ability to specify planes tangent at the interpolated points is an attractive property for local shape control. Some extensions of Nasri's original approach using Doo-Sabin subdivision surfaces have been developed to offer improvement in these contexts [1, 9]. The basic idea is to modify the mesh obtained after the first subdivision step. In this section, we show that these extensions can be easily included in our new approach.

4.1 Match normal conditions

Besides interpolating part or all of the vertices of the input polyhedron, the surface is now also required to have specified normal vectors at some interpolated vertices. To achieve this goal, Nasri modified his original approach by rotating each type-V face in the first refined mesh such that the normal to the rotated face is the same as the given face. A space transformation is determined for each type-V face. The new vertices are computed by applying the corresponding transformation.

In our approach, matching the normal conditions is very simple. Just replace the average normal in Eq. 17 by the required normal vector. This, in fact, even simplifies the computation because we do not need to compute the average normal (Eq. 10) in this situation. The modification of the vertices is automatically completed.

Figure 6 demonstrates the interpolation both with and without normal constraints. In Fig. 6, (a) and (d) show the same input mesh. We let the leftmost vertex and the rightmost vertex of the mesh be the tagged vertices to be interpolated. In (d), we further specify two vectors, which are labeled by the short green lines, to be the normal vectors of the tangent planes at those two interpolation vertices. By applying the algorithm to both of these input meshes, an exactly identical auxiliary mesh is created. However, the perturbed first refined meshes are different, which are displayed in (b) and (e). This is because the input mesh in (d) has normal constraints. Note that in (a), (b), (d) and (e), black lines have been added to highlight the edges of the meshes. Finally, the respective interpolating surfaces are shown in (c) and (f).

Compared to Nasri's approach, our method is more straightforward. Moreover, the constraints given by Eq. 11 in our approach actually specify the tangent plane. In other words, the normal vector $\mathbf{n}$ and the negative vector $-\mathbf{n}$ denote the same thing. However, in Nasri's approach, one needs to explicitly specify the normal direction that determines the transformation. The choice of positive or negative direction results in a different transformation. As pointed out in Nasri's paper [9], care must be taken when computing the cross-product vectors, as any change in direction will lead to an unexpected rotation.

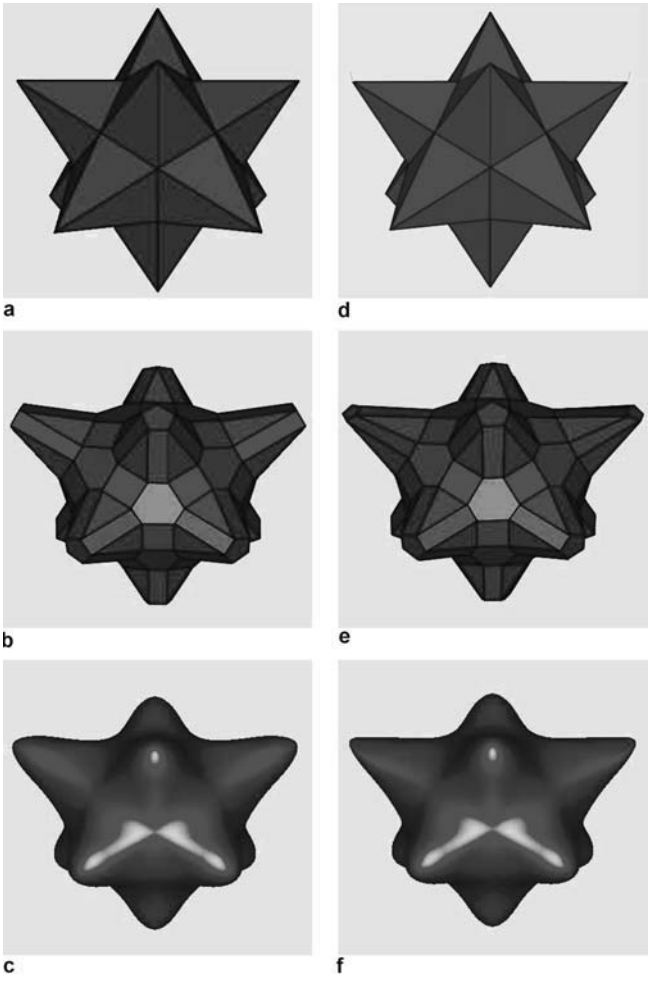

Fig. 6. Interpolation with and without normal control

4.2 Including shape handles

To increase the capability of adjusting the shape of subdivision surfaces, Brunet associated each vertex with a scalar number S called the *shape handle* [1]. Brunet used Nasri's approach to construct the auxiliary polyhedron, and then subdivided the polyhedron once to get the first refined mesh $\widehat{\mathcal{M}}^1$. Thus, each type-V face in $\widehat{\mathcal{M}}^1$ corresponds to one shape handle. Assume the kth type-V face consists of vertices W_{kl}, $l = 1, 2, \cdots$. Then, for these W_{kl}, a hometetic transformation can be carried out: $W'_{kl} = D_k + S_k(W_{kl} - D_k)$, where D_k is the centroid of the face (see Fig. 7) and S_k is the shape handle corresponding to the

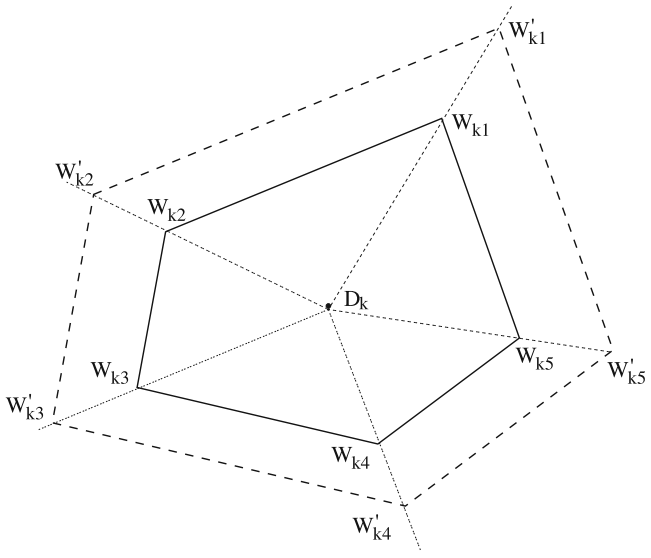

Fig. 7. A hometetic transformation

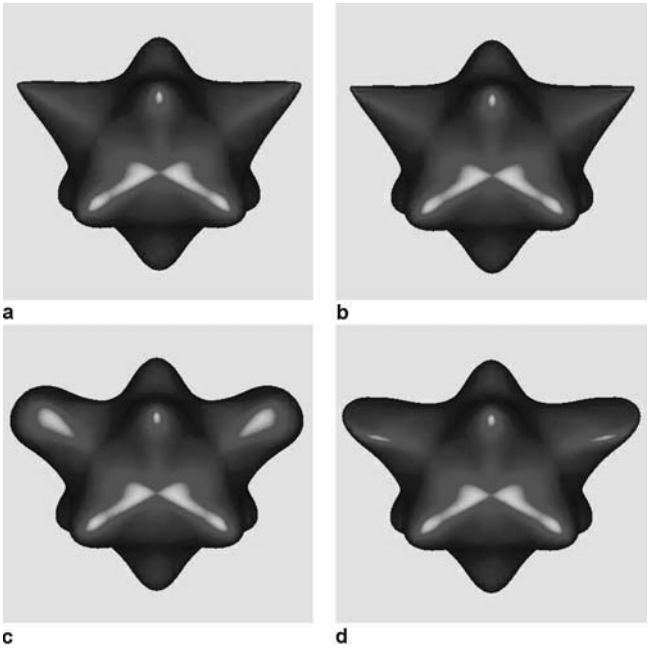

Fig. 8. Shape adjustment near the interpolation vertices

kth face. The shape handles are used to locally control the shape of the subdivision surface. For example, large values (> 1) of S_k produce a flat spot in the area near D_k.

Incorporating this scheme into our approach is straightforward. After we obtain the perturbed first refined mesh $\widehat{\mathcal{M}}^1$, what we need to do is just to apply Brunet's method to further adjust the vertices of $\widehat{\mathcal{M}}^1$. Figures 8(a) and 8(b) show the results of applying Brunet's scheme to Figs. 6(b) and 6(e) with the shape handles $S_k = 0.4$ at the interpolation vertices. The shapes look slim compared to those in Figs. 6(c) and 6(f). If the value of the shape handles is changed to 2.5, the respective shapes become Figs. 8(c) and 8(d), which look round or fat.

Finally, it should be emphasized that in the above two procedures, and in our previous perturbing step in Sect. 3.3, the topological information regarding the faces of the first refined mesh $\widehat{\mathcal{M}}^1$ is not altered. What changes is only the spatial position of the vertices.

5 Conclusions

We have described a simple method for automatic surface fitting through the vertices of an arbitrary topological mesh using Doo-Sabin subdivision surfaces. The method makes the coefficient matrix of the interpolation equations diagonally dominant. This ensures that the method always works and that the computation is numerically stable. Moreover, our approach of using iterative schemes instead of matrix inversion makes computation easy and programming simple. The experimental results show that the new method is significantly faster that the previous method in Nasri [8]. Our method is also capable of constraining the surface to have a specified tangent plane at the interpolation point, and it can provide the user shape parameters to locally adjust the shape. All of these features make it feasible for our method to be used to design and model complicated shapes.

Although this paper only discusses interpolation for closed meshes, extension to open meshes is straightforward. The future work includes the following considerations:

- The ordering in Gauss-Seidel iteration affects the convergence speed. How to choose an appropriate order for iteration is worthy of investigation.
- Our work provides one way to generate the first refined mesh that plays a key role in maintaining the position and normal interpolation. The aim of our approach is to ensure that the interpolation always works. However, we should point out that there exist other possibilities (and thus a lot of degrees of freedom) in the creation of the first refined mesh from the auxiliary mesh. How to make use of these degrees of freedom and also the shape handles to create visually pleasing shapes is an interesting problem.

Acknowledgement The authors thank the reviewers for their helpful suggestions. This work was funded by a research grant (SUG 8/04) at Nanyang Technological University. The authors would also like to thank Singapore's Agency for Science, Technology and Research for their support for this project.

References

1. Brunet P (1988) Including shape handles in recursive subdivisionsurfaces. Comput Aided Geom Des 5(1):41–50
2. Catmull E, Clark J (1978) Recursively generated B-spline surfaces on arbitrary topological meshes. Comput Aided Des 10:350–355
3. Doo D (1978) A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In: Proceedings of the Interactive Techniques in Computer Aided Design, Bologna, IEEE Computer Society, pp 157–165
4. Doo D, Sabin M (1978) Behaviour of recursive division surfaces near extraordinary points. Comput Aided Des 10:356–360
5. Foley J, Dam, A, Feiner S, Hughes J (1997) Computer graphics: principles and practice, 2nd edn. Addison-Wesley, Boston
6. Golub G, van Loan C (1989) Matrix computations, 2nd edn. John Hopkins University Press, Baltimore
7. Halstead M, Kass M, DeRose T (1993) Efficient, fair interpolation using Catmull-Clark surfaces. In: Computer Graphics, Proceedings of SIGGRAPH 93 27:35–44
8. Nasri A (1987) Polyhedral subdivision methods for free-form surfaces. ACM Trans Graph 6(1):29–73
9. Nasri A (1991) Surface interpolation on irregular networks with normal conditions. Comput Aided Geom Des 8(1):89–96
10. Sabin M (1991) Cubic recursive division with bounded curvature. In: Laurent PJ, Le Méhauté A, Schumaker LL (eds) Curves and surfaces. Academic Press, Boston, pp 411–414
11. Shilane P, Min P, Kazhdan M, Funkhouser T (2004) The Princeton shape benchmark. In: Proceedings of Shape Modeling International, Genove, Italy, pp 167–178
12. Stam, J (1998) Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In: Computer Graphics Proceedings, SIGGRAPH 98, pp 395–404
13. Zorin D, Schröder P (2000) Subdivision for modeling and animation. In: Course Notes of SIGGRAPH 2000. ACM, Boston
14. Zorin D, Schröder P, Sweldens W (1996) Interpolating subdivision for meshes with arbitrary topology. In: Computer Graphics, SIGGRAPH 96 Proceedings 30:189–192

JIANMIN ZHENG is an assistant professor in the School of Computer Engineering at Nanyang Technological University. Before joining NTU, he was a research faculty in Computer Science Department at Brigham Young University, USA. He was also a faculty at Zhejiang University, China, where he received his BS and PhD. He is a member of ACM SIGGRAPH. His research interest includes computer aided geometric design, CAD/CAM, computer graphics, animation, digital imaging and visualization.

DR YI-YU CAI is an Associate Professor with the College of Enginering, Nanyang Technological University (NTU), Singapore. He leads a strategic research program on Virtual Reality and Softcomputing in School of Mechanical & Aerospace Engineering. He is also deputy director of Bioinformatics Research Center in NTU. His research interests include CAD/CAM, geometric modeling, virtual reality, bioinformatics, and medical imaging.