Similarity based Interpolation using Catmull-Clark Subdivision Surfaces

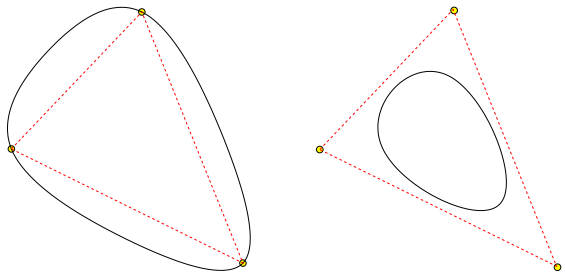
Shuhua Lai and Fuhua (Frank) Cheng
Graphics & Geometric Modeling Lab, Department of Computer Science
University of Kentucky, Lexington, Kentucky 40506-0046

Abstract. A new method for constructing a smooth Catmull-Clark subdivision surface (CCSS) that interpolates the vertices of a mesh with arbitrary topology is presented. The new method handles both open and closed meshes. Normals or derivatives specified at any vertices of the mesh (which can actually be anywhere) can also be interpolated. The construction process is based on the assumption that, in addition to interpolating the vertices of the given mesh, the interpolating surface is also similar to the limit surface of the given mesh. Therefore, construction of the interpolating surface can use information from the given mesh as well as its limit surface. This approach, called *similarity based interpolation*, gives us more control on the smoothness of the interpolating surface and, consequently, avoids the need of shape fairing in the construction of the interpolating surface. The computation of the interpolating surface's control mesh follows a new approach, which does not require the resulting global linear system to be solvable. An approximate solution provided by any fast iterative linear system solver is sufficient. However, interpolation of the given mesh is guaranteed. This is an important improvement over previous methods because with these features, the new method can handle meshes with large number of vertices efficiently. Although the new method is presented for CCSSs, the concept of similarity based interpolation can be used for other subdivision surfaces as well.

Keywords: subdivision, subdivision surfaces, Catmull-Clark subdivision surfaces, interpolation

1 Introduction

Given a 3D mesh, there exist infinitely many smooth surfaces that interpolate the mesh vertices. Any of them can be used as a solution to the interpolation problem. But, to a shape designer, usually only one of them is the surface he really wants. That surface, called the *designer's concept surface*, is a piece of important information for the interpolation process. If that information is available to the interpolation system, then by constructing an interpolating surface whose shape is '*similar*' to the designer's concept surface, we get the best result one can get for the interpolation process. We call an interpolation process *similarity based interpolation* if the interpolation also depends on establishing '*similarity*' with a *reference surface*. In the above case, the reference surface is the designer's concept surface.



(a) Interpolation (b) Approximation

Figure 1: They are similar!

The result of a similarity based interpolation depends on the quality of the reference surface. The closer the shape of the reference surface to the designer's concept surface, the better the result. The designer's concept surface usually is not available to the interpolation system. But it is reasonable to assume that the given mesh carries a shape similar to the designer's concept surface. Afterall, these are vertices the user extracted from his concept surface. Consequently, limit surface of the given mesh, when viewed as the control mesh of a Catmull-Clark subdivision surface, would be similar to the designer's concept surface. Therefore, using the limit surface as the reference surface in the interpolation process, i.e., constructing an interpolating surface of a given mesh that is also similar to the limit surface of the given mesh, we

should get an interpolating surface that is relatively close to the designer's concept surface. This interpolation concept has not been studied with subdivision surfaces before, although interpolation using subdivision surfaces has already been studied for a while.

1.1 Previous Work: A Brief Review

There are two major ways to interpolate a given mesh with a subdivision surface: *interpolating subdivision* [3, 6, 5, 8, 10] or *global optimization* [4, 7]. In the first case, a subdivision scheme that interpolates the control vertices, such as the Butterfly scheme[3], Zorin et al's improved version [10] or Kobbelt's scheme [6], is used to generate the interpolating surface. New vertices are defined as local affine combinations of nearby vertices. This approach is simple and easy to implement. It can handle meshes with large number of vertices. However, since no vertex is ever moved once it is computed, any distortion in the early stage of the subdivision will persist. This makes interpolating subdivision very sensitive to the irregularity in the given mesh. In addition, it is difficult for this approach to interpolate normals or derivatives.

The second approach, *global optimization*, usually needs to build a global linear system with some constraints. The solution to the global linear system is an interpolating mesh whose limit surface interpolates the control vertices in the given mesh. This approach usually requires some fairness constraints, such as the energy functions presented in [4], in the interpolation process to avoid undesired undulations. Although this approach seems more complicated, it results in a traditional subdivision surface. For example, the method in [4] results in a Catmull-Clark subdivision surface (CCSS), which is C^2 continuous almost everywhere and whose properties are well studied and understood. The problem with this approach is that a global linear system needs to be built and solved. Hence it is difficult to handle meshes with large number of vertices.

There are also subdivision techniques that produce surfaces to interpolate given curves or surfaces that near- (or quasi-)interpolate given meshes. But those techniques are either of different natures or of different concerns and, hence, will not be discussed here.

1.2 Overview

In this paper, we will address some of the problems with current vextex interpolation techniques by similarity based interpolation technique developed for CCSSs. Given a 3D mesh P with arbitrary topology, the new method calculates a control mesh Q whose CCSS interpolates the vertices of P. The CCSS of Q is constructed with the additional assumption that its shape is *similar* to a reference surface, the limit surface of P. A shape fairing process is not required in the construction process of the interpolating surface. The computation of the control mesh Q follows a new approach which does not require the resulting global linear system to be solvable. An approximate solution provided by any fast iterative linear system solver is sufficient. Hence, handling meshes with large number of vertices is not a problem. However, interpolation of the given mesh is guaranteed. The new method can handle both closed and open meshes. The interpolating surface can interpolate not only vertices of a given mesh, but also derivatives and normals anywhere in the parameter space of the surface.

The remaining part of the paper is arranged as follows. In Section 2, the similarity based interpolation technique for closed meshes is presented. A technique that works for open meshes is presented in Section 3. Implementation issues and test results are presented in Section 4. A summary is presented in Section 5.

2 Similarity based Interpolation

2.1 Mathematical Setup

Given a 3D mesh with n vertices: $P = \{\mathbf{P}_1, \mathbf{P}_2, \cdots, \mathbf{P}_n\}$, the goal here is to construct a control mesh Q whose CCSS interpolates P (the vertices of P, for now). The construction of Q follows the following path. First, we perform one or more levels of Catmull-Clark subdivision on P to get a finer control mesh G. G satisfies the following property: each face of G is a quadrilateral and each face of G has at most one *extra-ordinary vertex*. The vertices of G are divided into two categories. A vertex of G is called a *Type I vertex* if it corresponds to a vertex of P. Otherwise it is called a *Type II vertex*. Q is then defined as a control mesh with the same number of vertices and the same topolgy as G. We assume Q has m vertices $Q = \{\mathbf{Q}_1, \mathbf{Q}_2, \cdots, \mathbf{Q}_m\}$, $m > n$, and the first n vertices correspond to the n Type I vertices of G (and, consequently, the n vertices of P). These n vertices of Q will also be called Type I vertices and the remaining $m - n$ vertices Type II vertices. This way of setting up Q is to ensure the parametric form developed for a CCSS patch [9] can be used for the limit surface of Q, denoted $\mathbf{S}(Q)$, and we have enough degree of freedom in our subsequent work. Note that m is usually much bigger than n. The remaining job then is to determine the position of each vertex of Q.

In previous methods [7, 4] the n Type I vertices of Q are set as independent variables, the $m - n$ Type II vertices are represented as linear combinations of the Type I vertices. Since $m - n$ is bigger than n, this

setting leads to an over-determined system. Without any freedom in adjusting the solution of the system, one has no control on the shape of the resulting interpolating surface $\mathbf{S}(Q)$ even if it carries undesirable undulations. In this paper, instead, the $m-n$ Type II vertices are set as independent variables and the n Type I vertices are represented as linear combinations of the Type II vertices. This approach provides us with enough degrees of freedom to adjust the solution of the resulting linear system and, consequently, more control on the shape of the interpolating surface $\mathbf{S}(Q)$.

As discussed in the previous section, the interpolating surface $\mathbf{S}(Q)$ should be similar to the limit surface of P. The limit surface of P, denoted $\mathbf{S}(P)$, usually is smaller than the interpolating surface of P, $\mathbf{S}(Q)$. To establish a better similarity relationship, we scale up $\mathbf{S}(P)$ so that dimension of the scaled limit surface is the same as $\mathbf{S}(Q)$. The required scaling factors s_x, s_y and s_z for such a task can be determined by the condition that the bounding box of the scaled limit surface is the same as the bounding box of the interpolating surface. This can easily be done by comparing the maxima and minima of the vertices of P in all three directions with the maxima and minima of their corresponding limit points. The example shown in Figure 2 illustrates this process. Figure 2(b) is the limit surface of the mesh shown in Figure 2(a). The scaled limit surface is shown in Figure 2(c). Figure 2(d) is the interpolating surface of the mesh shown in Figure 2(a). We can see that the interpolating surface is more similar to the scaled limit surface than the original limit surface. In the subsequent discussion, it should be understood that each reference to the similarity between the interpolating surface and the limit surface of P actually means the similarity between the interpolating surface and the scaled limit surface of P. The control mesh of the scaled limit surface, called $\hat{G}$, can be obtained by scaling G with s_x, s_y and s_z. The scaled limit surface will be denoted $\mathbf{S}(\hat{G})$.

2.2 Interpolation Requirements

Recall that Type I vertices of Q are those vertices that correspond to vertices of P. Hence, each vertex of P is the limit point of a Type I vertex of Q. We assume the limit point of $\mathbf{Q}_i$ is $\mathbf{P}_i$, $1 \le i \le n$. Then for each Type I vertex $\mathbf{Q}_i$ $(1 \le i \le n)$, we have

$$\mathbf{Q}_i = C_i \cdot \widetilde{Q} + c\mathbf{P}_i \tag{1}$$

where $\widetilde{Q} = \{\mathbf{Q}_{n+1}, \mathbf{Q}_{n+2}, \cdots, \mathbf{Q}_m\}$ is the vector of Type II vertices. Vector C_i and constant c depend on the topology of P and the degree of vertex $\mathbf{P}_i$. C_i and c can be easily obtained using the formula for calculating the limit point of a CCSS [9, 4]. The conditions in (1) are called *interpolation requirements*, because they have to be exactly satisfied.

Note that the interpolation requirements in (1) form a system of linear equations. By solving this system of linear equations, we solve the interpolation problem [7]. But in this case one tends to get undesired undulations on the resulting interpolationg surface [4].

2.3 Similarity Constraints

Two CCSSs are said to be *similar* if their control meshes have the same topology and they have similar ith derivatives $(1 \le i < \infty)$ everywhere. The first condition of this definition is a sufficient conditon for the second condition to be true, because it ensures the considered CCSSs have the same parameter space. The CCSSs considered here, $\mathbf{S}(Q)$ and $\mathbf{S}(\hat{G})$, satisfy the first condition. Hence, we have the sufficient condition to make the assumption that $\mathbf{S}(Q)$ and $\mathbf{S}(\hat{G})$ are similar. In the following, we assume $\mathbf{S}(Q)$ and $\mathbf{S}(\hat{G})$ are similar in the sense of the above definition.

With explict parameterization of a CCSS available [9], it is possible for us to consider derivatives of $\mathbf{S}(Q)$ and $\mathbf{S}(\hat{G})$ at any point of their parameter space. However, to avoid costly integration of derivative expressions, we will only consider derivatives sampled at the following points

$$\{(1/2^i, 1/2^j) \mid 0 \le i, j \le \infty\} \tag{2}$$

for each patch of $\mathbf{S}(Q)$ and $\mathbf{S}(\hat{G})$. In the above similarity definition, two derivatives are said to be *similar* if they have the same direction. In the following, we use the similarity condition to set up constraints in the construction process of $\mathbf{S}(Q)$.

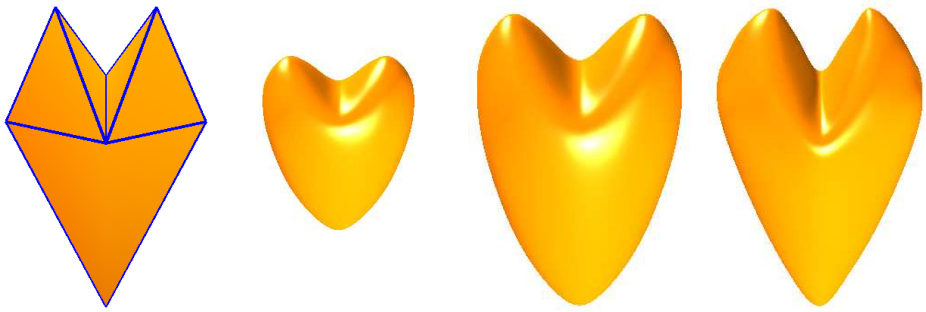
Given two surfaces, let $\mathbf{D}_u$ and $\mathbf{D}_v$ be the u and v derivatives of the first surface and $\hat{\mathbf{D}}_u$ and $\hat{\mathbf{D}}_v$ the u and v derivatives of the second surface. These derivatives are *similar* if the following conditin holds:

$$\mathbf{D}_u \times \hat{\mathbf{D}}_u = \mathbf{0} \quad \text{and} \quad \mathbf{D}_v \times \hat{\mathbf{D}}_v = \mathbf{0} \tag{3}$$

A different condition, shown below, is used in [4, 7].

$$\mathbf{D}_u \cdot (\hat{\mathbf{D}}_u \times \hat{\mathbf{D}}_v) = 0 \quad \text{and} \quad \mathbf{D}_v \cdot (\hat{\mathbf{D}}_u \times \hat{\mathbf{D}}_v) = 0 \tag{4}$$

These two conditions are not necessarily equivalent. Our test cases show that (3) gives better interpolating surfaces. This is because (4) only requires the corresponding derivatives to lie in the same tangent plane, no restrictions on their direcitons. As a result, using (4) could result in unnecessary undulations. Note that (3) requires directions of $\mathbf{D}_u$ and $\mathbf{D}_v$ to be the same as that of $\hat{\mathbf{D}}_u$ and $\hat{\mathbf{D}}_v$, respectively.



(a) (b) (c) (d)

Figure 2: Similariy between the interpolating surface and the limit surface: (a) given mesh, (b) limit surface, (c)scaled limit surface, (d) interpolating surface.

Conditions of the type shown in (3) are called *similarity constraints*. These constraints do not have to be satisfied exactly, only to the extent possible. The interpolation method used in [7] considers interpolation requirements only. The method in [4] also includes fairness constraints to avoid undesired undulations and artifacts.

2.4 Global Linear System

If the derivatives of $\mathbf{S}(Q)$ and $\mathbf{S}(\hat{G})$ are sampled at a point in (2) then, according to (3) and the derivative of the parametric form of a CCSS patch [9], we would have

$$(V^T \cdot Q) \times (V^T \cdot \hat{G}) = \mathbf{0} \tag{5}$$

where V is a constant vector of scalars whose values depend on the type of the derivative and the point where the sampling is performed. This expression actually contains 3 equations, one for each component. Replace the Type I vertices $\mathbf{Q}_1, \mathbf{Q}_2, \cdots, \mathbf{Q}_n$ in the above expression with (1) and combine all the similarity constraints, we get a system of linear equations which can be represented in matrix form as follows:

$$\mathrm{D} \cdot X = C$$

where X is a vector of length $3(m-n)$, whose entries are the x, y and z components of $\widetilde{Q}$. D usually is not a square matrix. Hence we need to find an X such that $(\mathrm{D} \cdot X - C)^T \cdot (\mathrm{D} \cdot X - C)$ is minimized. This is a quadratic programming problem and can be solved using a linear least squares method. It is basically a process of finding a solution of the following linear system:

$$\mathrm{A} \cdot X = B \tag{6}$$

where $\mathrm{A} = \mathrm{D}^T\mathrm{D}$ and $B = \mathrm{D}^T C$. A is a symmetric matrix. Hence only half of its elements need to be calculated and stored. Once X is known, i.e., $\widetilde{Q}$ is known, we can find $\mathbf{Q}_1, \mathbf{Q}_2, \cdots, \mathbf{Q}_n$ using (1).

The matrix D could be very big if many sample points or constrains are used. Fortunately, we do not have to calculate and store the matrix D and the vector C. Note that A and B can be writen as

$$\mathrm{A} = \sum D_i (D_i)^T \quad \textit{and} \quad B = \sum D_i c_i$$

where $(D_i)^T$ is the ith row of D and c_i is the ith entry of C. Note that the number of rows of D can be as large as possible but the number of columns is fixed, $3(m-n)$. Suppose the ith constraint (See eq. (5)), with $\mathbf{Q}_1, \mathbf{Q}_2, \cdots, \mathbf{Q}_n$ replaced, is written in vector form as $U^T \cdot X = u$. Then U^T is the ith row of matrix D and u is the ith entry of C. Hence rows of matrix D and entries of C can be calculated independently from (5) for each constraint of each sample point. Therefore, A and B can be accumulatively calculated, constraint by constraint. No matter how many sample points are used, and no matter how many constraints are considered for every sample point, only a fixed amount memory is required for the entire process and the size of matrix A is always the same, $3(m-n) \times 3(m-n)$.

Note that the solution of (6) only determines the positions of Type II vertices of Q, i.e. $\mathbf{Q}_{n+1}, \mathbf{Q}_{n+2}, \cdots, \mathbf{Q}_{n+m}$. Type I vertices of Q, $\mathbf{Q}_1, \mathbf{Q}_2, \cdots, \mathbf{Q}_n$, are represented as linear combinations of $\mathbf{Q}_{n+1}, \mathbf{Q}_{n+2}, \cdots, \mathbf{Q}_{n+m}$ in the interpolation requirements defined in (1). Since interpolation of the vertices of P by the interpolating surface is determined by the interpolation requirements (1) only, this means as long as we can find a solution for (6), the task of

constructing an interpolating surface that interpolates the vertices of P can always be fulfilled, even if the solution is not precise. Hence, an exact solution to the linear system (6) is not a must for our method. An approximate solution provided by a fast interative linear system solver is sufficient. As a result, the new method can handle meshes with large number of vertices efficiently. This is an important improvement over previous methods.

With the similarity assumption, the surface interpolation problem is basically a process of using an iterative method to find an approximate solution for the global linear system (6). The scaled mesh $\hat{G}$ is a good initial guess for the interative process because $\hat{G}$ is actually very close to the control mesh of the interpolating surface we want to obtain. In our implementation, the Gauss-Seidel method is used for the iterative process. The iterative process would converge to a good approximate solution very rapidly with this initial guess. However, it should be pointed out that there is no need to carry out the iterative process to a very precise level. According to our test cases, a residual tolerance of the size $\epsilon = 10^{-6}$ does not produce much noticable improvement on the quality of the interpolating surface than a residual tolerance of the size $\epsilon = 10^{-2}$, while the former takes much more time than the latter. Therefore a relatively large residual tolerance can be supplied to the iterative linear system solver to prevent it from running too long on the iterative process, while not improving the quality of the interpolationg surface much. This is especially important for processing meshes with large number of vertices.

2.5 Additional Interpolation Requirements

In addition to the *interpolation requirements* considered in (1), other *interpolation requirements* can be included in the global linear system as well. One can also modify or remove some of the *interpolation requirements* in (1). For example, if we wants the first $u-$derivative of the interpolating surface at $\mathbf{P}_i$ to be $\mathbf{D}_u$, we need to set up a condition similar to (5) as follows:

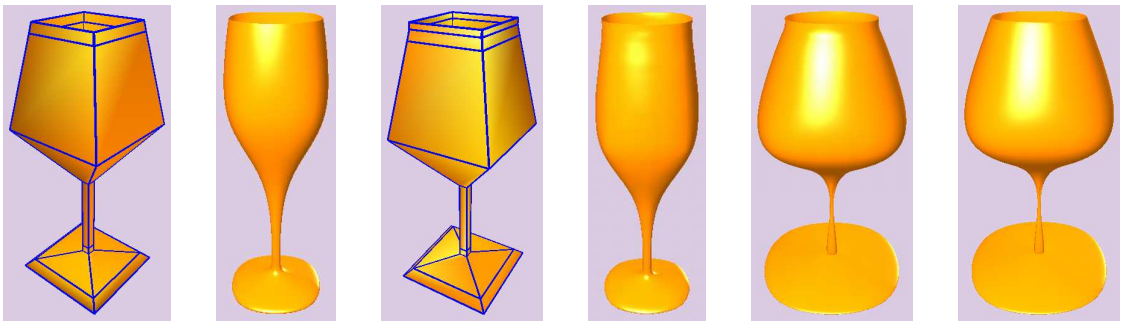
$$(V^T \cdot Q) \times \mathbf{D}_u = \mathbf{0}$$

where V is a constant vector. The difference here is, this is not a similarity constraint, but an interpolation requirement. However, if we want a particular normal to be interpolated, we should set up interpolation requirements for the u derivative and the v derivative whose cross product equals this normal, instead of setting up an interpolation requirement for the normal directly, to avoid the involvement of non-linear equations in our system. Then by combining all the new interpolation requirements with the original interpolation requirements in (1), we get all the expressions for vertices that are not considered independent variables in the linear system in (6). Note that including a new interpolation requirement in the interpolation requirement pool requires us to change a variable vertex in Q to a non-variable vertex. Actually, interpolation requirements can be specified for any points of the interpolating surface, not just for vertices of P. This is possible because we have a parametric representation for each patch of a CCSS [9]. For example, if we want the position of a patch at $(1/2, 1/4)$ to be $\mathbf{T}$, we can set up an interpolation requirement of the form: $V^T \cdot Q = \mathbf{T}$ where V is a constant vector whose values depend on $(1/2, 1/4)$. Therefore the interpolating surface can interpolate positions, derivatives and normals anywhere in the parameter space.

3 Handling Open Meshes

The interpolation process developed in the previous section can not be used for open meshes, such as the one shown in Fig. 3(a), directly. This is because boundary vertices of an open mesh have no corresponding limit points, nor derivatives, therefore, one can not set up interpolation requirements for these vertices, as required by the new interpolation process. One way to overcome this problem is to add an additional ring of vertices along the current boundary and connect the vertices of this ring with corresponding vertices of the current boundary to form an additional ring of faces, such as the example shown in Figure 3(c). The newly added vertices are called *dummy vertices*. We then apply the interpolation method to the extended open mesh as to a closed mesh except that there are no interpolation requirements for the dummy vertices. This technique of extending the boundary of a given mesh is similar to a technique proposed for uniform B-spline surface representation in [1].

Note that in this case, the interpolation process does not use the limit surface of the given mesh, but rather the limit surface of the extended mesh as a reference surface. Therefore, the shape of the interpolating surface depends on locations of the dummy vertices as well. Determining the location of a dummy vertex, however, is a tricky issue, and the user should not be burdened with such a tricky task. In our system, this is done by using locations of the current boundary vertices of the given mesh as the initial locations of the dummy vertices and then solving the global linear system in (6) to determine their final locations.



(a) (b) (c) (d) (e) (f)

Figure 3: Interpolating an open mesh: (a) given mesh; (b) limit surface of (a); (c) extended version of (a); (d) limit surface of (c); (e) interpolating surface of (a) that is similar to (d); (f) interpolating surface of (c) with additional requirements.

This approach of generating dummy vertices works fine because dummy vertices only affect similarity constraints. Figure 3(e) is a surface that interpolates the mesh given in Fig. 3(a) and uses 3(d) as a reference surface.

The above setting of the dummy vertices usually is not enough to create an interpolating surface with the desired boundary shape. Additional requirements (not constraints) are needed in the interpolation process. As explained in Section 2.5, a platform that allows us to define additional requirements can be created by treating the dummy vertices as non-variables in (6). We can then specify new derivative conditions or normal conditions to be satisfied at the original boundary vertices. With the additional interpolation requirements, a designer has more control on the shape of the interpolating surface in areas along the boundary and, consequently, can generate an interpolating surface with the desired boundary shape. For example, Figure 3(f) is an interpolating surface of the mesh given in Figure 3(a), but generated with additional interpolation requirements. The interpolating surface obviously looks more like a real glass now.

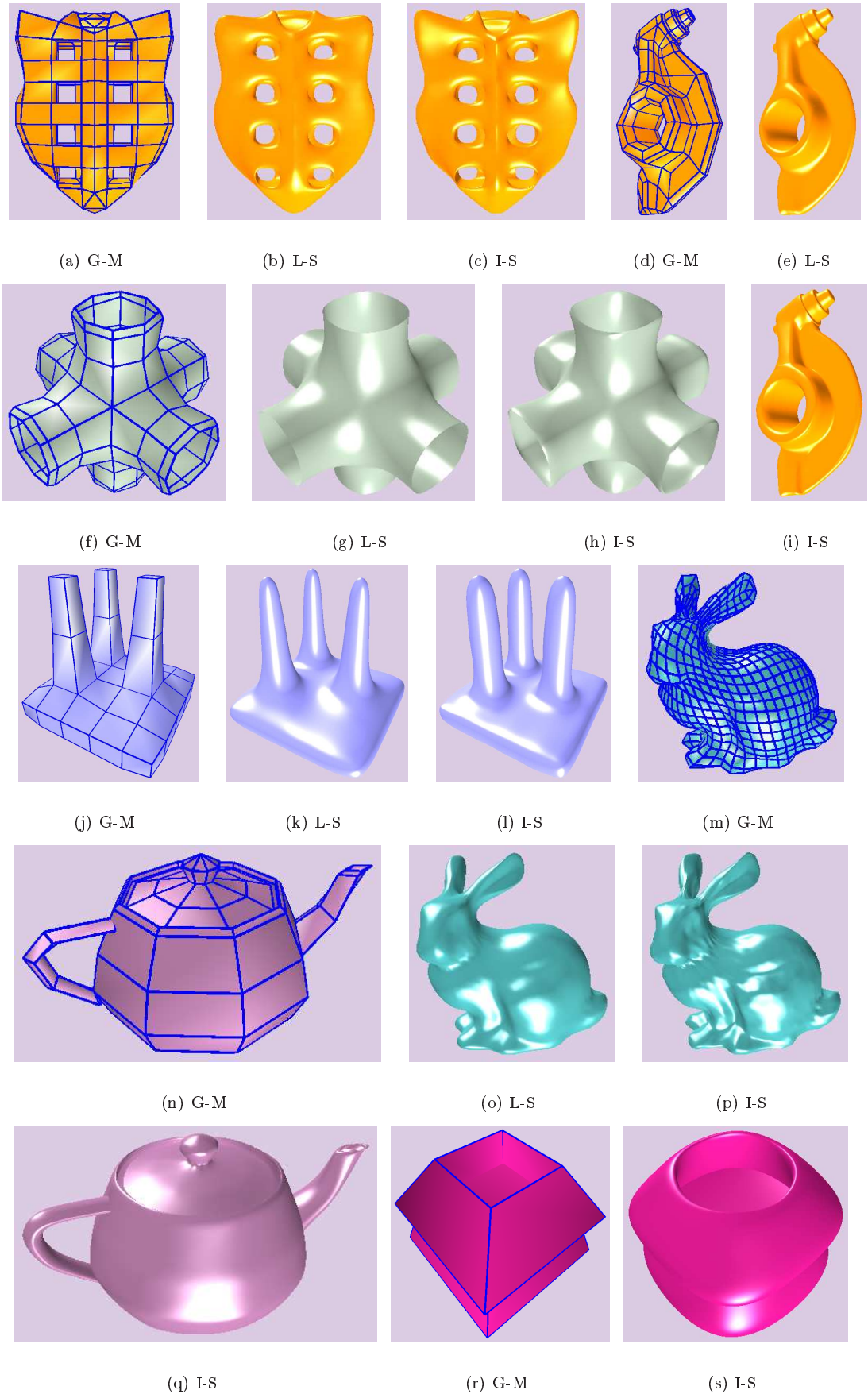
4 Test Results

The proposed approach has been implemented in *C++* using *OpenGL* as the supporting graphics system on the Windows platform. Quite a few examples have been tested with the method described here. All the examples have extra-ordinary vertices. Some of the tested results are shown in Figures 2, 3 and 4. Due to limited space, limit surface of the Utah Teapot which is well known and limit surface of the mesh shown in Figure 4(r) which is very simple are not shown here. For all other cases, the limit surfaces (not scaled) of the given meshes and the interpolating surfaces are both shown so that one can tell if these surfaces are indeed similar to each other in the least squares sense.

In our implementation, only one subdivision is performed on the given mesh for each example and the first, second and third derivatives in u and v directions are used to construct interpolation constraints and build the global linear system. These derivatives are sampled at points with parameters $(\frac{1}{2^i}, \frac{1}{2^j})$, $i, j = 0, 1$ or ∞, for each patch. That is, 9 points are sampled for each patch, which is good enough for most cases. For bigger patches one can use more sample points because patches do not have to be sampled uniformly.

The original Utah teapot consists of four separate parts: lid, handle, body and spout. The mesh shown in Figure 4(n) is actually a set of four meshes, one for each component of the original Utah teapot. Each mesh is an open mesh. Although each of these meshes can be interpolated separately, Figure 4(q) is generated by regarding them as a single mesh. The mesh shown in Figure 4(f) is another example of an open mesh with disconnected boundaries. Figure 4(h) is the interpolating surface without using additional interpolation requirements in the construction process.

As can be seen from Figure 4, all the resulting interpolating surface are very smooth and visually pleasing, except the interpolating surface shown in Figure 4(p). The surface has some undulations around the neck, but we do not think they are caused completely by our method. We believe this is more of a problem with the general interpolation concept. Note that the input



(a) G-M (b) L-S (c) I-S (d) G-M (e) L-S

(f) G-M (g) L-S (h) I-S (i) I-S

(j) G-M (k) L-S (l) I-S (m) G-M

(n) G-M (o) L-S (p) I-S

(q) I-S (r) G-M (s) I-S

Figure 4: Interpolating meshes with arbitrary topology (G-M: given mesh; L-S: limit surface; I-S: interpolating surface.

mesh, Figure 4(m), has some abrupt changes of vertex positions and twists in the neck area. This is also reflected by some visible undulations in the neck area of the limit surface, Figure 4(o), even though they are not as clear as in the interpolating surface. An approximation curve/surface, like a spline curve, can be regarded as a low pass filter [10], which makes the given control polygon or mesh smoother. An interpolation curve/surface, on the other hand, can be regarded as a high pass filter, which magnifies undulations or twists in the input mesh. Since a limit surface is an approximation surface, it reduces the impact of abrupt vertex location changes and twists in the input mesh while the interpolating surface enhances it. This is why the undulations are more obvious in Figure 4(p) than in Figure 4(o).

The new interpolation method can handle meshes with large number of vertices in a matter of seconds on an ordinary PC (3.2GHz CPU, 512MB of RAM). For example, the mesh shown in Figure 4(m) has 1,022 vertices and 1024 faces, and only takes 51 seconds to interpolate it. The rocker arm shown in Figure 4(d) has 354 vertices and 354 faces, and only takes 4 seconds to interpolate it. The teapot model shown in Figure 4(n) has 138 vertices and 128 faces, and it takes less than 1 second to interpolate it. For smaller meshes, like Figures 4(a), 4(j), 4(r) and 4(f), the interpolation process is done almost in real time. Hence our interpolation method is suitable for interactive shape design, where simple shapes with small or medium-sized control vertex sets are constructed using design or interpolation methods, and then combined using CSG trees to form complex objects.

5 Summary

A new interpolation method for meshes with arbitrary topology using general CCSSs is presented. The development of the method is based on the assumption that the interpolating surface should be *similar* to the limit surface of the given mesh. Our test results show that this approach leads to good interpolation results even for complicated data sets.

The new method has several special properties. First, by using information from the vertices of the given mesh as well as its limit surface, one has more control on the smoothness of the interpolating surface. Hence, a *surface fairing* process is not needed in the new method. Second, there is no system solvability problem for the new method. The global linear system that the new method has to solve does not require an exact solution, an approximate solution is sufficient. The approximate solution can be provided by any fast iterative linear solver. Consequently the new method can process meshes with large number of vertices efficiently. Third, the new method can handle both open and closed meshes. It can interpolate not only vertices, but normals and derivatives as well. These normals and derivative can be anywhere, not just at the vertices of the given mesh. Therefore, the new method is general.

Acknowledgement. Data set of Figure 4(m) is downloaded from research.microsoft.com/∼hoppe and the original data sets of Figures 4(r) and 4(f) are downloaded from mrl.nyu.edu/∼dzorin.

References

[1] Barsky B A, End conditions and boundary conditions for uniform B-spline curve and surface representation, *Computers in Industry*, 1982, 3(1/2):17-29.

[2] Catmull E, Clark J, Recursively generated B-spline surfaces on arbitrary topological meshes, *Computer-Aided Design*, 1978, 10(6):350-355.

[3] Dyn N, Levin D, and Gregory J A, A butterfly subdivision scheme for surface interpolation with tension control, *ACM Transactions on Graphics*, 1990, 9(2):160-169.

[4] Halstead M, Kass M, DeRose T, Efficient, fair interpolationusing Catmull-Clark surfaces, *ACM SIGGRAPH*, 1993:35-44.

[5] Levin A, Interpolating nets of curves by smooth subdivision surfaces, *Computer Graphics Proceedings (SIGGRAPH)*, Annual Conference Series, 1999, 57-64.

[6] Kobbelt L, Interpolatory subdivision on open quadrilateral nets with arbitrary topology, *Computer Graphics Forum*, Eurographics, V.15, 1996.

[7] Nasri A H, Surface interpolation on irregular networks with normal conditions, *Computer Aided Geometric Design*, 1991, 8:89-96.

[8] Schaefer S, Warren J, A Factored Interpolatory Subdivision Scheme for Quadrilateral Surfaces, *Curves and Surface Fitting*: Saint Malo 2002, 373-382.

[9] Stam J, Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values, *Proceedings of SIGGRAPH* 1998:395-404.

[10] D. Zorin, P. Schröder, W. Sweldens, Interpolating Subdivision for meshes with arbitrary topology, *ACM SIGGRAPH*, 1996:189-192.