Fast Mesh Interpolation and Mesh Decomposition with Applications

Paper ID: 0369
Category: Technical Paper

Abstract A new approach for constructing a smooth subdivision surface to interpolate the vertices of an arbitrary mesh is presented. The construction process does not require setting up any linear systems, not any matrix computation, but is simply done by iteratively moving vertices of the given mesh locally until control mesh of the required interpolating surface is reached. The new interpolation method has the simplicity of a local method in effectively dealing with meshes of large number of vertices. It also has the capability of a global method in faithfully resembling the shape of a given mesh. Furthermore, the new method is fast and does not require a fairing step in the construction process because the iterative process converges to a unique solution at an exponential rate. Another important result of this work is, with the new iterative process, each mesh (surface) can be decomposed into a sum of simpler meshes (surfaces) which carry high and low frequency information of the given model. This mesh decomposition scheme provides us with new approaches to some classic applications in computer graphics such as texture mapping, de-noising/smoothing/sharpening, and morphing. These new approaches are demonstrated in this paper and test results are included.

1 Introduction

Constructing a smooth surface from a set of sampled or scanned data points is an important problem in many areas such as computer graphics, reverse engineering, geometric modeling, computer vision and animation. Interpolation is the most widely used technique in this smooth surface construction process. However, powerful interpolation techniques that have the advantages of both a global method and a local method and, consequently, can handle meshes of any size and any topology while capable of faithfully reproducing the shape of any given mesh, are not available yet. In this paper, we try to fill the gap with an iterative interpolation method. The new method has the advantages mentioned above. Besides, it is very fast and very easy to implement. Subdivision surfaces are the representation scheme used in this interpolation technique.

Subdivision surfaces [1, 11, 14] have become popular recently because of their capability in modeling/representing any complex shape with only one surface [2]. Subdivision surfaces cover both *parametric forms* [12, 13] and *discrete forms*. Parametric forms are good for design and representation and discrete forms are good for machining and tessellation (including FE mesh generation). Therefore we have a representation scheme that is good for almost all applications. Powerful interpolation techniques using subdivision surfaces as a representation scheme certainly are needed for subdivision surface based applications.

Traditional interpolation techniques [3, 4, 5, 6, 8, 9, 10, 15] are mainly aimed at one purpose: *shape reconstruction*. The iterative interpolation technique proposed in this paper also provides us with a means to look at several classic applications from a different angle and, consequently, allows us to solve these applications with different approaches. This is because with the new technique each mesh (surface) can be decomposed into a sum of simpler meshes (surfaces). By manipulating the decomposed items we can control low-frequency and high-frequency information of the given mesh (surface) and, consequently, the overall shape or local details of the mesh (surface).

Overall, the main contributions of this paper can be summarized as follows:

- presenting an iterative interpolation technique that can effectively deal with meshes of any size and topology; the iterative process converges at an exponential rate; no linear system is used, no matrix computation is required and no fairing process is needed in the construction of the interpolation surface.

- showing that each mesh (or surface) can be decomposed into a sum of simpler meshes (surfaces) which provides alternative approaches to several classic applications in computer graphics.

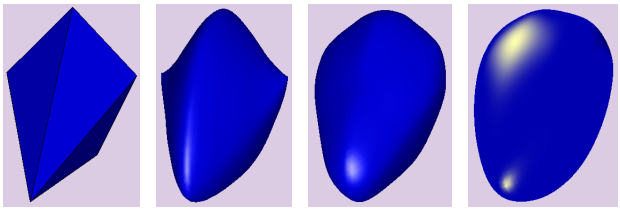
2 Previous Work

There are two major methods for interpolating a given mesh with a subdivision surface: *local method* [3, 5, 6, 9, 10, 15] or *global method* [4, 8]. In a local method, vertices are moved to new positions defined as affine combinations of nearby vertices iteratively. Interpolating subdivision is the most well-known local interpolation method. In this case, a subdivision scheme such as the Butterfly scheme[3], Zorin et al's improved version [15] or Kobbelt's scheme [6], is used to generate the interpolating surface. 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. When the mesh vertices are dense enough, the undesired artifacts would not be so clear to see. But when the mesh vertices are not so dense, the effect of undesired artifacts becomes obvious on the resulting interpolating surfaces (see Figure 1 for an example where the Butterfly scheme [3], the improved Butterfly scheme [15] and the technique proposed in this paper are compared on a given mesh with five vertices).

A *global method*, on the other hand, usually needs to build a global linear system with some constraints [4]. The solution to the global linear system is a control mesh whose limit surface interpolates the vertices of the given mesh. This approach usually requires some fairness constraints [4] in the interpolation process to avoid undesired undulations. Although this approach seems more complicated, it results in a traditional subdivision surface which resembles the shape of the given mesh more faithfully. The problem with this approach is that it is difficult to handle meshes with large number of vertices because it needs to set up a global linear system.

There are also techniques that produce subdivision surfaces to interpolate given curves or surfaces that near- (or quasi-)interpolate given meshes [7]. But those techniques are either of different natures or of different concerns, hence will not be discussed here.

As far as we know, none of the existing methods has the advantages of both a local method and a global method. Hence we do not have a subdivision surface based interpolation approach that can handle meshes of any size and any topology while capable of faithfully reproducing the shape of any given mesh. Besides, none of the previous techniques were designed for other applications but mesh interpolation.



(a) Given Mesh (b) Butterfly (c) New Butterfly (d) New Method

Figure 1: Comparison of Butterfly (b), modified Butterfly (c), and new (d) methods.

3 Mathematical Setup

Given a mesh M and a subdivision scheme, our task is to find a smooth subdivision surface to interpolate M. We use the following notations in the paper: A refers to the matrix that calculates all the limit points of M with respect to the given subdivision scheme; $\mathbf{S}(M)$ refers to the limit surface of M; $\mathbf{I}(M)$ refers to the subdivision surface that interpolates M and limit points of $\mathbf{I}(M)$'s control points equal M, i.e., if P is the control mesh of $\mathbf{I}(M)$ then we must have $M = \mathrm{A} * P$. We also assume the subdivision scheme considered here is the Catmull-Clark scheme. Hence, $\mathbf{I}(M)$ and $\mathbf{S}(M)$ are Catmull-Clark subdivision surfaces. However, the techniques presented here work for other subdivision schemes as well.

Let $M = M_0$ be the given mesh. Then the task is to find $\mathbf{I}(M_0)$. If we can find a smooth offset surface $\mathbf{L}$ that moves $\mathbf{S}(M_0)$ to $\mathbf{I}(M_0)$ everywhere, i.e.,

$$\mathbf{L} + \mathbf{S}(M_0) = \mathbf{I}(M_0) \tag{1}$$

then the interpolation problem is solved. The question is: how should $\mathbf{L}$ be constructed?

Note that the above definitions and the fact that A is invertible (see Appendix B), and $\mathbf{I}(\cdot)$ is a linear operator imply that $\mathbf{S}(M_0) = \mathbf{I}(\mathrm{A} * M_0)$. Hence $\mathbf{L}$ can be regarded as an interpolating surface $\mathbf{I}(M_1)$ where

$$M_1 = M_0 - \mathrm{A} * M_0$$

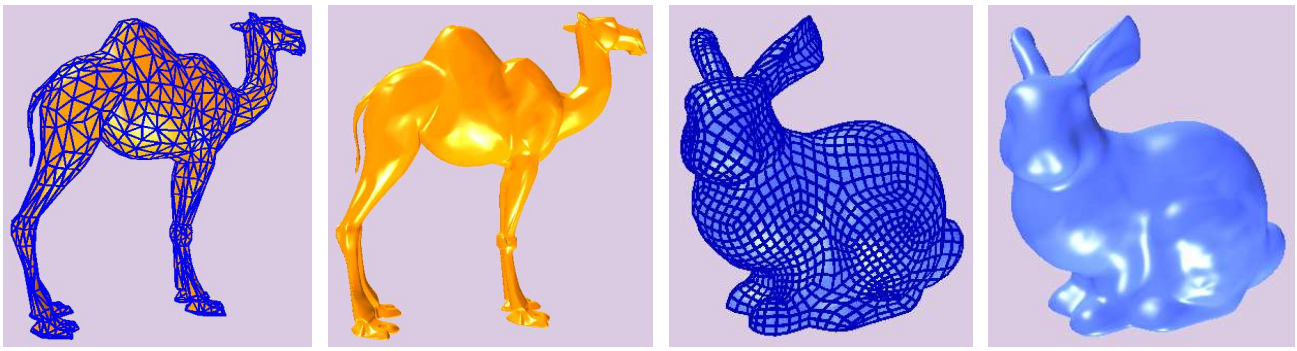
is the difference between M_0 and its limit points $\mathrm{A} * M_0$. M_1 has the same topology as M_0. Hence $\mathbf{I}(M_1)$ and $\mathbf{I}(M_0)$ can be constructed exactly the same way.

By iteratively repeating the above process we get a sequence of meshes M_i such that

$$\begin{cases} M_{i+1} = M_i - \mathrm{A} * M_i \\ \\ \mathbf{I}(M_{i+1}) + \mathbf{S}(M_i) = \mathbf{I}(M_i) \end{cases} \qquad (0 \le i \le \infty) \tag{2}$$

From the above equation we have

$$\mathbf{I}(M) = \sum_{i=0}^{n} \mathbf{S}(M_i) + \mathbf{I}(M_{n+1}). \tag{3}$$



(a) Given Triangular Mesh (b) Interpolation Surface (c) Given Quadrilateral Mesh (d) Interpolation Surface

Figure 2: Examples of mesh interpolation.

We can also easily get M_i as follows

$$M_i = (\mathrm{E} - \mathrm{A})^i M, \tag{4}$$

where E is an identity matrix. It can be proven (see Appendix C) that $\lim_{i\to\infty} M_i = \mathbf{0}$. As a result, $\mathbf{I}(M_i)$ approaches $\mathbf{0}$ when i tends to ∞. In addition, because subdivision is a linear process, we have $\sum \mathbf{S}(M_i) = \mathbf{S}(\sum M_i)$. Therefore, the control mesh P, whose limit surface interpolates the given mesh M, can be calculated as follows.

$$P = \sum_{i=0}^{\infty} (\mathrm{E} - \mathrm{A})^i M. \tag{5}$$

On the other hand, because A is invertible (see Appendix B), there can be only one mesh (having the same topology as M) whose limit surface interpolates the given M. Consequently a fairing process is not needed in the construction of the interpolating surface because there exists only one such surface that interpolates the given mesh. Traditionally, people tried to directly find $\mathrm{A}^{-1}M$ by solving a linear system [4, 8]. For meshes with large number of vertices, it is difficult to set up and solve the huge corresponding global linear systems even with a fast solver such as the Gauss-Seidel method. With the new technique, no matrix is required to be calculated, no linear system is needed to be set up and solved. The work is done simply by iteratively moving mesh vertices around locally (see eq. (7)) until some given error tolerance is reached (see Section 4). Hence there is no problem to deal with meshes with large number of vertices.

More importantly, just like Fourier series and multiresolution represention [21, 22], any mesh (or surface) now can be decomposed into a sum of simpler meshes (or subdivision surfaces). For example, a given subdivision surface $\mathbf{S}(M)$ can be represented as an infinite series of subdivision surfaces as

$$\mathbf{S}(M) = \mathbf{I}(\mathrm{A} * M) = \sum_{i=0}^{\infty} \mathbf{S}((\mathrm{E} - \mathrm{A})^i * \mathrm{A} * M)$$

and for a given mesh M, we have

$$M = \sum_{i=0}^{\infty} (\mathrm{E} - \mathrm{A})^i * \mathrm{A} * M. \tag{6}$$

In the above infinite series, each term $(\mathrm{E} - \mathrm{A})^i \mathrm{A} M$ contains part of the information on M. Terms with smaller indices contain more information on the overall shape of M while terms with bigger indices contain more subtle details on the shape of M. They can be regarded as low and high frequency information on mesh M. Hence the above two equations transform a space domain model into a frequency domain representation. Because each term itself is a mesh (or surface), it can be furthermore decomposed using the above two equations to obtain more subtle details of the original model. Like Fourier series, this representation can be used for applications in other areas of graphics and modeling, such as fairing, smoothing, sharpening, low pass or high pass filtering etc. For example, for any $n < \infty$, $\sum_{i=0}^{n} (\mathrm{E} - \mathrm{A})^i \mathrm{A} M$ gives us a smoother model than M, and the smaller of n, the smoother of the resulting model. On the other hand, $\sum_{i=0}^{n} (\mathrm{E} - \mathrm{A})^i M$ sharpens model M, and the bigger of n, the sharper of the resulting model. In the following sections, several direct and straightforward applications of the above representation will be discussed.

4 Fast Iterative Interpolation

Eq. (5) should not be used to construct the interpolating surface directly, because it requires costly matrix multiplications. Actually matrix A is not needed

in the construction of the interpolating surface at all. Note that if we define $P_n = \sum_{i=0}^{n}(\mathrm{E} - \mathrm{A})^i M$, $0 \leq n < \infty$, then through simple algebra we have

$$P_{n+1} = P_n + M - \mathrm{A} * P_n. \tag{7}$$

The geometric meaning of the above formula is: in the iteraive process, the new location of a mesh vertex can be obtained by moving that vertex by an offset, and the offset is the difference between the original location of that vertex (in the given mesh M) and the limit point of that vertex (in the current mesh P_n). Note that in eq. (7), $\mathrm{A} * P_n$ represents the limit points of vertices in P_n and as we know, all limit points can be directly calculated according to Catmull-Clark subdivision rules [4]. Therefore one can use eq. (7) to find the control mesh $P = P_\infty$ of the interpolating surface iteratively without using any matrices or linear systems at all.

In addition, as we mentioned above, P is the only mesh that has the same topology as M and whose limit surface interpolates M. Therefore there is no need for a fairing process either. Traditional interpolation techniques [4, 8] need a fairing process because extra vertices are added in the interpolation process. These extra vertices, with possibly improperly assigned positions, could lead to undulations in the interpolating surface because they need to be interpolated as well.

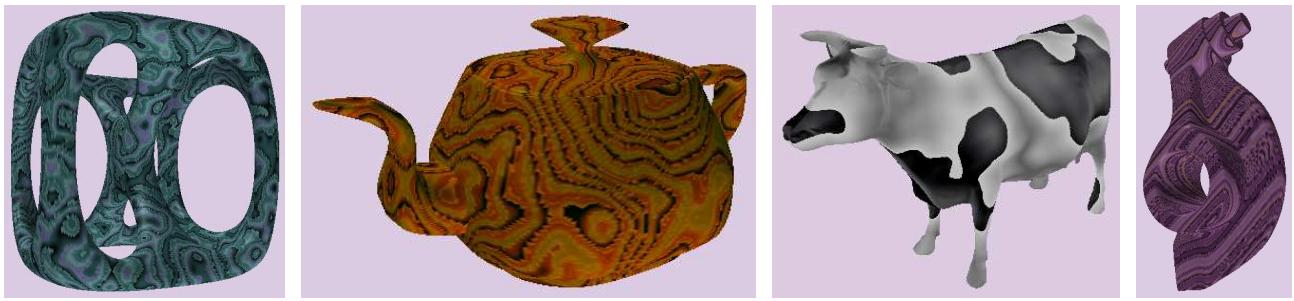
The computation from P_n to P_{n+1} in eq. (7) is a local affine combination process. Hence this is a local method. On the other hand, P_n converges to $\mathrm{A}^{-1}M$, which means every vertex in M contributes to all the vertices in P_∞. Hence this is a global method as well. Consequently, we have a method that not only has the simplicity of a local method in effectively dealing with meshes of large number of vertices, but also has the capability of a global method in faithfully resembling the shape of a given mesh.

As is proven in the Appendix C, eq. (7) converges at an exponential rate. Hence good interpolation results can be obtained in just a few iterations. Nevertheless, error can be explicitly calculated as $||M - \mathrm{A} * P_n||$ and the iteration stop criterion can be determined based on some given error tolerance. Because it converges rapidly, the new interpolation technique is even suitable for interactive shape design. Figure 2 shows some test results. We can see from these examples that smooth and visually pleasant interpolation shapes can be obtained for complicated meshes with dense or not-so-dense vertices. All the test cases are done in less than one second without GPU support.

5 Procedural Texture Mapping

3D Procedural texture mapping, instead of looking up an image to do the mapping, generates texture for a model by assigning different colors to vertices of the model. The color of each vertex is determined only by the vertex's location in 3D space. Procedural texture mapping usually uses the Perlin noise [23] function to make rendered models interesting by adding irregularity to a procedural texture. A key component in the construction of Perlin noise is the interpolation function which makes it possible to smoothly add discrete noises of different amplitudes and frequencies together [23]. Traditionally, in 3D case, Perlin noise is constructed at each integer lattice point in a 3D array, such as the case of solid texturing [25] or hypertexture [24]. As far as we know, procedural texture mapping has never been applied to smooth subdivision surfaces by directly assigning colors to points of the surfaces to generate natural-looking models. We believe the main reason of this is because a powerful technique that can smoothly interpolate all the noises of an irregular mesh is not available yet. With the fast iterative interpolation technique presented in the above section, this is no longer the case. We can now do procedural texture mapping on a given 3D model by mapping the model's 3D locations directly into colors. Hence a 3D array for storing noises is not needed any more.

For any given 3D mesh G_0, perform n times of subdivision to get $n+1$ meshes G_i, $(0 \leq i \leq n)$. For each G_i, a Perlin noise value $\frac{1}{2^i} f(2^i x, 2^i y, 2^i z)$ is generated for each vertex with position (x, y, z), where f is a random noise function. Note that G_{i+1} has many more vertices than G_i, hence the distribution of noises on G_{i+1} is denser than G_i. All the noises of mesh G_i form a mesh N_i. N_i has the same topology as G_i, and the x component of each N_i's vertex is set to the noise value at the corresponding vertex of G_i. N_i's y and z components are not needed, hence these values do not have to be defined. By applying our fast iterative interpolation technique to N_i, we get $\mathbf{I}(N_i)$. Because all $\mathbf{I}(N_i)$ are smooth subdivision surfaces, they can be added together (through subdivision or parametrization [12]) to form a new surface. Let $\mathbf{N} = \sum_{i=0}^{n} \mathbf{I}(N_i)$. After normalization, $\mathbf{N}_x$, which is the x component of $\mathbf{N}$, contains noises for the subdivision surface $\mathbf{S}(G_0)$ everywhere. For a given list of colors, three B-splines $\mathbf{B}(t) = (B_r(t), B_g(t), B_b(t))$ are constructed to interpolate all the colors of the given list. Now one can render $\mathbf{S}(G_0)$ by assigning colors (procedural texture) to all points of the surface. The color of each point $\mathbf{P}$ of $\mathbf{S}(G_0)$ is obtained by finding the spline values $\mathbf{B}(t_p)$, where t_p is the noise value $\mathbf{N}_x$ at point $\mathbf{P}$. One can



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

Figure 3: Examples of procedural texture mapping.

also render mesh G_i directly by assigning generated colors to vertices of G_i only.

Different from methods that only generate Perlin noises at given vertices, our new method can generate more noises of different amplitudes and different frequencies for irregular models at more positions, and can blend them smoothly by interpolation. As a result, it creates procedural texture with better visual effect. More importantly, a huge array is not needed in the procedural texture mapping process. Figure 3 shows several examples of procedural texture mapping generated using our method. All of them are generated with only basic Perlin noise functions.

6 De-noising

With the proliferation of 3D scanning devices, fairing, smoothing and denoising of noisy meshes have become more and more important. Several important works have been done in this area [19, 17, 16, 18]. In this section, we present a new denoising technique which is a straightforward application of our interpolation formula. Our purpose here is simply to show the versatility of our interpolation method, hence we did not compare our results with other denoising methods. Nevertheless, our method is easy to understand, easy to implement, and can achieve relatively good denoising results.

Consider a finite portion of the series defined in eq. (6). Define $T_0 = M$ and T_k recursively as follows:

$$T_{k+1} = \sum_{i=0}^{m} (\mathrm{E} - \mathrm{A})^i * \mathrm{A} * T_k, \qquad k \geq 0 \qquad (8)$$

When $m < \infty$, T_{k+1} is a smoother version of T_k because some high frequency information is not included. Through simple computation, we have $T_{k+1} = \mathrm{D} * \mathrm{A} * T_k$ (hence $T_k = \mathrm{D}^k * \mathrm{A} * T_0$), where $\mathrm{D} = \mathrm{E} - (\mathrm{E} - \mathrm{A})^{m+1}$.

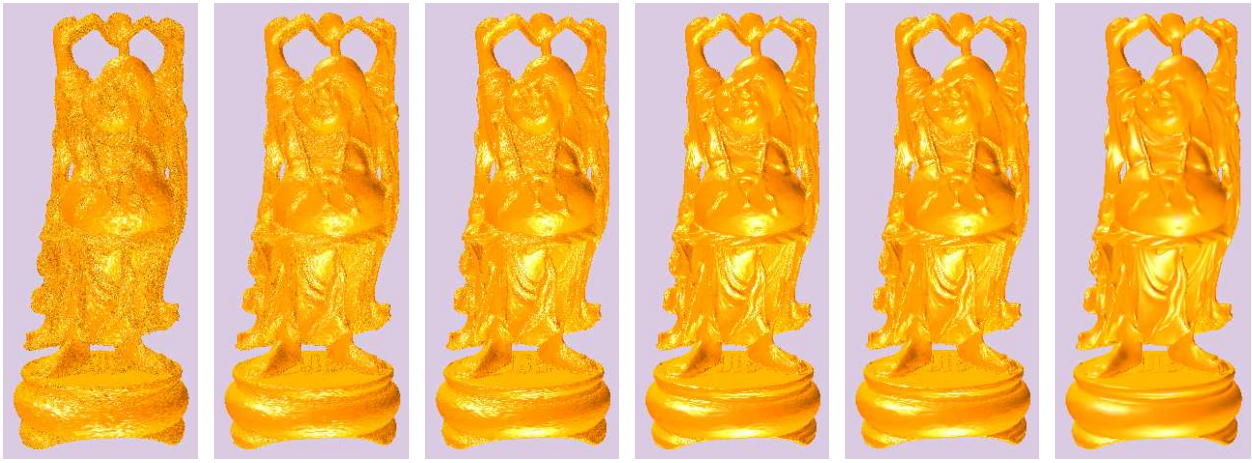
When $m = 0$ in eq. (8), $T_k = \mathrm{A}^k * \mathrm{A} * T_0$. Because all the eigenvalues of A are between 0 and 1 (see Appendix B), $\mathrm{A}^k * \mathrm{A} * M$ converges to a point when k tends to infinity. Hence using matrix A^k for smoothing would make the mesh shrink. Mesh shrinking leads to model details losing, hence A^k should not be used for de-noising directly. However, by compensating A^k with some details in each step, it would shrink much slower, consequently keep more details, and meanwhile smooth out undesired noises. For example, when $m = 1$, $T_k = (\mathrm{A}^k + (\mathrm{A}^k - \mathrm{A}^{2k})) * \mathrm{A}T_0$ which compensates A^k by a small mount of details $\mathrm{A}^k - \mathrm{A}^{2k}$.

For any m, the biggest eigenvalue of D is always 1 with corresponding eigenvector $[1, 1, 1, ..., 1]^T$ and all other eigenvalues are between 0 and 1. Hence D^k can be used to smooth meshes. Note that even though eventually $\mathrm{D}^k \mathrm{A} M$ shrinks to a single point when k approaches infinity, it shrinks at a speed much slower than $\mathrm{A}^k \mathrm{A} M$. This is because if the second biggest eigenvalue of A is λ, then the second biggest eigenvalue of D is $1 - (1 - \lambda)^{m+1}$, which is much bigger that λ and becomes even bigger when m gets bigger. Because T_k shrinks slower, it maintains more details in the denoising process while smoothing out undesired noises.

m can be used as a parameter to control how much details should be kept in the de-noising process. The smaller of m, the smoother the resulting model and the less details are kept. When m gets bigger, more details, and possibly more noises are kept. In our test cases, setting m to 2 or 3 is good enough for fast and good de-noising (smoothing) while maintaining enough details of the input model.

Similar to interpolation, which sharpens a mesh with dramatic changes, eq. (8) can be modified to sharpen a mesh T_0 gently as follows.

$$T_{k+1} = T_k + \sum_{i=m}^{n} (\mathrm{E} - \mathrm{A})^i * T_k, \qquad k \geq 0$$



(a) Noise added (b) 1 iteration (c) 2 iterations (d) 3 iterations (e) 4 iterations (f) Original model

Figure 4: Examples of mesh denoising.

where $T_0 = M$ and $n \geq m > 0$. From the above equation, we can see that T_{k+1} is a sharper version of T_k because it adds extra high frequency information of T_k to T_{k+1}.

Figure 4 demonstrates the capability of eq. (8) in mesh smoothing. From Figure 4 we can see that with only 3 or 4 times of iteration, pretty good denoising results can already be obtained, even for complicated models. Comparing the original model shown in Figure 4(f) with the figure shown in Figure 4(e), we can see that many subtle details are kept meanwhile noises are smoothed out.

7 Morphing

For two given meshes M and Q, the task here is to find a smooth transition from Q to M. We assume M and Q have the same topology. If this is not the case, simply resample them using one of the resampling techniques such as [20].

According to eq. (6), we have

$$Q = M + \sum_{i=0}^{\infty} (\mathrm{E} - \mathrm{A})^i \mathrm{A}(Q - M).$$

$(Q - M)$ can be regarded as the difference of the two models, hence our goal is to transform this difference smoothly so that it approaches zero eventually. By taking different finite terms in the above summation, we get a transition from M to Q. However, this approach would not give us a smooth morphing from M to Q because, as we mentioned above, $(\mathrm{E}-\mathrm{A})^i$ shrinks at an exponential rate.

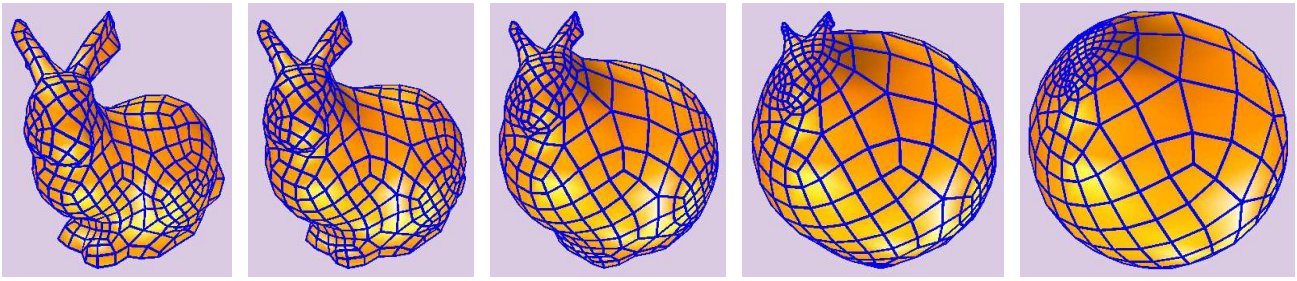
Let $Q_0 = Q$ and for a given $m > 0$, define Q_k recursively as follows:

$$Q_{k+1} = M + \rho \sum_{i=0}^{m} (\mathrm{E} - \mathrm{A})^i \mathrm{A}(Q_k - M), \qquad (9)$$

where $0 < \rho \leq 1$ is used to control the morphing speed. The smaller of ρ, the faster the morphing from Q to M. When $m < \infty$, $(Q_{k+1} - M)$ is a smoother version of $(Q_k - M)$. Similar to the discussision in the above section, we know $(Q_k - M)$ approaches zero when k tends to infinity. As a result, eq. (9) provides us with a sequence of meshes Q_k, $k \geq 0$, transforming Q to M. For a properly selected m and ρ in the above equation for each k, the sequence of meshes Q_k provides a smooth morphing from Q to M.

In the morphing process, Q_{k+1} gets its information from both M and Q_k. The bigger of k, the more information from M. Therefore, eq. (9) can be regarded as a linear operation on meshes M and Q_k.

Traditionally, given two meshes M and Q with the same topology, morphing is done simply by performing a linear operation on M and Q: $u*Q+(1-u)*M$. This single-vertex-based linear combination method sometimes does not produce meaningful and desired morphing results if features of the meshes are not aligned properly. For example, if M and Q are symmetric about the x-axis, then the morphing process will produce a flat shape when $u = 0.5$ that certainly is not desired. Eq. (9) provides an alternative approach for morphing, which can be regarded as a multiple-vertex-based linear combination process. The bigger of k, the more vertices of Q_0 are involved in determining vertex positions of Q_k. Therefore, the morphing process



(a) Given model (b) Morphing (early) (c) Morphing (middle) (d) Morphing (late) (e) Morphing (end)

Figure 5: Examples of mesh morphing.

is not so heavily affected by features of the meshes. Consequently, eq. (9) produces better morphing results than simple linear morphing methods, especially when mesh features are not very well aligned.

Again, eq. (9) should not be used to do the morphing directly because of costly matrix multiplications. A formula similar to eq. (7) should be derived for local and iterative computation of the morphing. Figure 5 shows a morphing example of a bunny to a 3D sphere using the technique presented above. As can be seen from this example, the morphing process is very smooth, un-natural looking artifacts usually found in a simple-vertex-based morphing process are nowhere around.

8 Summary and Future Work

An iterative mesh interpolation method and an accompanying mesh decomposition technique with its applications in several classic areas of computer graphics are presented. The iterative mesh interpolation method generates a subdivision surface to interpolate a given mesh by iteratively modifying vertices of the given mesh locally until control mesh of the required interpolating surface is reached. The new method is fast and does not require any matrix computation, or linear systems or a fairing step in the construction process of the interpolating surface. Hence, an iterative method seems to be the ultimate solution for mesh interpolation because it has all the advantages one would like to see for a mesh interpolation process.

The mesh decomposition technique decomposes a given mesh (surface) into an infinite series of simpler meshes (surfaces) which can be regarded as high or low frequency information of the given mesh. Hence we can use decomposed items to control high-frequency and low-frequency information of the mesh (surface) and, consequently, overall shape or local details of the mesh (surface). Manipulating or balancing such pieces of information is a core work in many classic areas of computer graphics. Hence, a mesh decomposition provides a new solution or an alternative solution to some of the classic areas of computer graphics. As far as we know this is the first ever attempt to use a mesh decomposition to solve problems in texture mapping, denoising and morphing.

One of our future research subjects is to investigate other possible applications of the mesh decomposition. Areas that will be considered include mesh compression, feature identification and mesh simplification. Another subject of our future research is to compare the performance of the approaches provided by a mesh decomposition with other methods in the literature to study their effectiveness and possible improvements. In addition, the matrix A in eq. (6), instead of being a subdivision matrix, could be set to other matrices, as long as their eigen values are in $(0, 1]$. Hence it is possible to design an A and use eq. (6) to solve problems in computer graphics with specified requirements.

References

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

[2] DeRose T, Kass M, Truong T, Subdivision Surfaces in Character Animation, *Proceedings of SIGGRAPH*, 1998: 85-94.

[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 interpolation using Catmull-Clark surfaces, *ACM SIGGRAPH*, 1993:35-44.

[5] Levin A, Interpolating nets of curves by smooth subdivision surfaces, *ACM SIGGRAPH*, 1999, 57-64.

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

[7] Litke N, Levin A, Schröder P, Fitting subdivision surfaces, *Proceedings of the conference on Visualization* 2001:319-324.

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

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

[10] Peters J, C1-surface splines. *SIAM Journal on Numerical Analysis* 1995, 32(2):645-666.

[11] Sederberg T W, Zheng J, Sewell D, Sabin M, Non-uniform recursive subdivision surfaces, *Proceedings of SIGGRAPH*, 1998:19-24.

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

[13] Reif, U. A unified approach to subdivision algorithms near extraordinary points. *Computer Aided Geometric Design* 12, 2, 153C174, 1995.

[14] Adi Levin, Modified Subdivision Surfaces with Continuous Curvature, *Proceedings of SIGGRAPH*, 1035-1040, 2006.

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

[16] S. Fleishman, I. D. Tel and D. Cohen-Or, Bilateral mesh denoising, *SIGGRAPH 2003*.

[17] Mathieu Desbrun, Mark Meyer, Peter Schröder, Alan Barr, Implicit fairing of irregular meshes using diffusion and curvature flow, *SIGGRAPH 1999*, p.317-324.

[18] T. Jones , F. Durand , M. Desbrun, Non-iterative, feature-preserving mesh smoothing, *ACM Transactions on Graphics*, v.22 n.3, 2003.

[19] Leif Kobbelt, Discrete Fairing, In *Proc. of the 7th IMA Conf. on the Mathematics of Surfaces*, 1997.

[20] E. Praun, H. Hoppe. Spherical parametrization and remeshing, *SIGGRAPH* 2003, P.340-349.

[21] Denis Zorin, Modeling with multiresolution subdivision surfaces, *SIGGRAPH* 2006, P.30-50.

[22] M. Lounsbery , Tony DeRose , Joe Warren, Multiresolution analysis for surfaces of arbitrary topological type, *ACM Transactions on Graphics*, P.34–73, 1997.

[23] Ken Perlin, Improving Noise, *SIGGRAPH 2002*.

[24] Ken Perlin, Hypertexture, *SIGGRAPH 1989*.

[25] Darwyn Peachey, Solid texturing of complex surfaces, *SIGGRAPH 1985*, p.279-286.

Appendix

A Construction of Matrix A

The matrix A is not needed in the implementation. We show it here only for proof purpose. The matrix shown here is for the generalized Catmull-Clark subdivision scheme. The matrix for other schemes can be constructed similarly.

For generalized Catmull-Clark subdivision scheme, new face points and edge points are calculated the same way as the standard Catmull-Clark subdivision scheme, but the new vertex points are calculated differently, using the following formula

$$\mathbf{V}' = \frac{n-2}{n}\mathbf{V} + \frac{1}{n^2}\sum_{j=1}^{n}(\alpha\mathbf{V} + (1-\alpha)\mathbf{E}_j) + \frac{1}{n^2}\sum_{j=1}^{n}\mathbf{f}_j$$

where $0 \leq \alpha \leq 1$ and $\mathbf{f}_j$ are new face points after one subdivision. When $\alpha = 0$, we get the standard Catmull-Clark subdivision scheme. The limit point [4] of a vertex $\mathbf{V}_i$ of degree n_i can be calculated as:

$$\mathbf{V}_i^{\infty} = \frac{1}{n_i(n_i+5)}(b_{ii}\mathbf{V}_i + \sum_j b_{ij}\mathbf{E}_j + \sum_j b_{ij}\mathbf{F}_j)$$

where

$b_{ii} = (n_i - 1)n_i + n_i\alpha + \sum \frac{4}{d_{ij}}$
$b_{ij} = (2 - \alpha + \frac{4}{d_{ij}} + \frac{4}{d_{ji}})$, if $(\mathbf{V}_\mathrm{i}, \mathbf{V}_\mathrm{j})$ is an edge
$b_{ij} = 4/d_{ij}$, if $(\mathbf{V}_\mathrm{i}, \mathbf{V}_\mathrm{j})$ is a diagonal line of a face
$b_{ij} = 0$, if $\mathbf{V}_\mathrm{i}$ and $\mathbf{V}_\mathrm{j}$ do not belong to the same face

Note that in the above formula the surrounding faces could be not-four-sided (see figure 6). d_{ij} is the number of sides of the face of which $(\mathbf{V}_i, \mathbf{V}_j)$ is an edge or a diagonal line. Note that d_{ij} and d_{ji} could have different values because faces adjacent to the edge (V_i, V_j) could have different number of sides. But if $(\mathbf{V}_i, \mathbf{V}_j)$ is a diagonal line of a face, then $d_{ij} = d_{ji}$. According to the above definition, we have

$$\mathrm{A}_{ij} = \frac{1}{n_i(n_i+5)}b_{ij}.$$

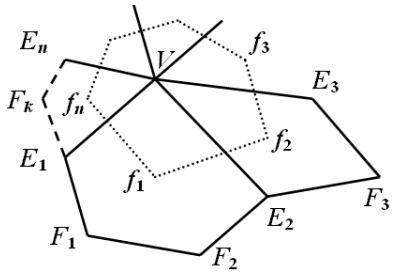


Figure 6: Vertex $\mathbf{V}$ and its neighboring vertices.

B Matrix A's eigen values $\lambda_i \in (0, 1]$

It is easy to verify that each entry of A is non-negative and the sum of each row equals one. Hence all eigen values λ_i of A are all ≤ 1. Therefore, to prove $\lambda_i \in (0, 1]$, we only need to show that all A's eigen values are positive. A common coefficient $1/n_i/(n_i + 5)$ can be factored out for each row of A. If we define matrix B as $\mathrm{B}_{ij} = b_{ij}$, then it is easy to verify that B is symmetric, and $\mathrm{A} = diag(1/n_i/(n_i + 5)) * \mathrm{B}$. Hence we just need to prove all the eigen values of B are positive, which is equivalent to prove that B is positive definite.

We need to show $\mathbf{X}^T\mathrm{B}\mathbf{X} > 0$ for any non-zero vector $\mathbf{X}$. $\mathbf{X}^T\mathrm{B}\mathbf{X}$ can be expanded as follows.

$$\mathbf{X}^T\mathrm{B}\mathbf{X} = \sum_{all\ E} 2b_{ij}x_ix_j + \sum_{all\ D} 2b_{ij}x_ix_j + \sum_i b_{ii}x_i^2$$

$$= \sum_{all\ E}(b_{ij} - \frac{4}{d_{ij}} - \frac{4}{d_{ji}})(x_i + x_j)^2 + \sum_{all\ F}\frac{4}{d_{ij}}(x_i + x_j + \cdots + x_p)^2$$

$$+ \sum_i (b_{ii} - \sum_{(\mathbf{V}_i,\mathbf{V}_j)\ \in E}(b_{ij} - \frac{4}{d_{ij}} - \frac{4}{d_{ji}}) - \sum_{(\mathbf{V}_i,\mathbf{V}_j)\in E}\frac{4}{d_{ij}})x_i^2$$

$$\geq \sum_{all\ E}(2 - \alpha)(x_i + x_j)^2 + \sum_i (n_i^2 - 3 * n_i + 2n_i\alpha)x_i^2$$

where E denotes all edges, D denotes all diagonal lines of all faces and F denotes all faces of the given mesh. Let $\sigma_i = n_i^2 - 3 * n_i + 2n_i\alpha$. Because when $n_i \geq 3$, $\sigma_i \geq 0$ and $2 - \alpha > 0$, to prove $\mathbf{X}^T\mathrm{B}\mathbf{X} > 0$, we just need to show $\sigma_i > 0$. Obviously, when $n_i \geq 4$, we have $\sigma_i > 0$. Hence B is positive definite. Also we can claim that B is positive definite if there exists at least one vertex $\mathbf{V}_k$ in the given mesh such that $n_k \geq 4$. This can be proven by contradiction. Suppose this is not the case, then there exists an $\mathbf{X} \neq 0$ such that $\mathbf{X}^T\mathrm{B}\mathbf{X} = 0$. It is easy to see that $x_k = 0$ for otherwise we would have $\mathbf{X}^T\mathrm{B}\mathbf{X} \geq \sigma_k x_k^2 > 0$. In addition, all x_j where $(\mathbf{V}_k, \mathbf{V}_j)$ is an edge must be 0 as well because otherwise we have $\mathbf{X}^T\mathrm{B}\mathbf{X} \geq (2-\alpha)(x_k + x_j)^2 = (2 - \alpha)x_j^2 > 0$. Similarly, all vertices directly or indirectly connected to $\mathbf{V}_k$ are all equal to 0. Because M is a connected mesh, all x_i are 0, which contradicts the assumption $X \neq 0$. Hence if there exists at least one vertex whose valance is bigger than 3, then B is positive definite as well.

When all n_i are 3 in a mesh and $\alpha = 0$, B might not be positive definite. However, we can change the value of α to convert the standard Catmull-Clark subdivision scheme into a general Catmull-Clark subdivision scheme, such that B is positive definite. It is easy to verify that when $\alpha > 0$ and $n_i \geq 3$, we have $\sigma_i > 0$. Therefore let $\alpha \in (0, 1]$, say $\alpha = 0.5$, then B becomes positive definite.

Because B is positive definite, all the eigenvalues of A are positive. Therefore A is invertible and all its eigenvalues are in $(0, 1]$.

C Proof of Convergence

Because all eigenvalues of A are $\in (0, 1]$, all the eigenvalues of $(\mathrm{E}-\mathrm{A})$ are $\in [0, 1)$. Hence $(\mathrm{E}-\mathrm{A})^i$ converges to $\mathbf{0}$ when i tends to infinity. As a result, $\sum_{i=0}^{n}(\mathrm{E}-\mathrm{A})^i$ converges to A^{-1} when n tends to infinity.

It can also be proven that eq. (7) or eq. (5) converges at an exponential rate. To prove this it is sufficient to show that $||P_\infty - P_n||$ converges to $\mathbf{0}$ at an exponential rate.

$$\begin{aligned}
&||P_\infty - P_n|| \\
&= ||\textstyle\sum_{i=n+1}^{\infty}(\mathrm{E} - \mathrm{A})^i M|| \\
&= ||(\mathrm{E} - \mathrm{A})^{n+1} * \mathrm{A}^{-1} * M|| \\
&= ||\mathbf{X}\Lambda^{n+1}\mathbf{X}^{-1} * \mathrm{A}^{-1} * M|| \\
&\leq ||\Lambda^{n+1}|| * ||\mathbf{X}|| * ||\mathbf{X}^{-1}|| * ||\mathrm{A}^{-1}|| * ||M|| \\
&= ||\Lambda^{n+1}|| * c
\end{aligned}$$

where c is a constant and Λ is the diagonalized matrix of $(\mathrm{E} - \mathrm{A})$. Suppose the biggest eigenvalue of matrix $(\mathrm{E} - \mathrm{A})$ is λ, then $||\Lambda^{n+1}|| \leq \lambda^{n+1}$. As we know, $0 \leq \lambda < 1$. Hence we have

$$||P_\infty - P_n|| \leq c * \lambda^{n+1},$$

which means P_n converges to P_∞ at an exponential rate.