Technical Report

UCAM-CL-TR-755
ISSN 1476-2986

Number 755

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Skin-detached surface for interactive large mesh editing

Yujian Gao, Aimin Hao, Qinping Zhao,
Neil A. Dodgson

September 2009

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Skin-detached surface for interactive large mesh editing

Yujian Gao
Yujian.Gao@cl.cam.ac.uk

Aimin Hao
ham@vrlab.buaa.edu.cn

Qinping Zhao
zhaoqp@vrlab.buaa.edu.cn

Neil A. Dodgson
Neil.Dodgson@cl.cam.ac.uk

Abstract

We propose a method for interactive deformation of large detail meshes. Our method allows the users to manipulate the mesh directly using freely-selected handles on the mesh. To best preserve surface details, we introduce a new surface representation, the *skin-detached surface*. It represents a detail surface model as a peeled "skin" added over a simplified surface model. The "skin" contains the details of the surface while the simplified mesh maintains the basic shape. The deformation process consists of three steps: At the mesh loading stage, the "skin" is precomputed according to the detail mesh and detached from the simplified mesh. Then we deform the simplified mesh following the nonlinear gradient domain mesh editing approach to satisfy the handle position constraints. Finally the detail "skin" is remapped onto the simplified mesh, resulting in a deformed detail mesh. We investigate the advantages as well as the limitations of our method by implementing a prototype system and applying it to several examples.

1 Introduction

The field of mesh deformation has attracted a lot of attention throughout recent years, and a variety of techniques have been developed and widely used in movie production, mesh editing tools and game engines. Existing mesh deformation techniques include: free-form deformation (FFD) [1, 2], RBF-based mesh deformation [3], curve-based deformation [4, 5], skeleton deformation [6], physical simulation [7] and gradient domain deformation. The computational load and memory consumption of all these approaches grow tremendously as the complexity of the mesh increases. With the development of 3D scanning techniques, people are becoming more and more critical about the detail level of 3D meshes, but it is difficult to edit and deform large meshes interactively using these methods on common PCs. In addition, except for gradient domain deformation, these techniques have one common limitation: the abundant geometry details present in the mesh might be seriously distorted in large scale deformation.

To achieve interactive editing of large detail meshes while preserving the geometric details, we propose a skin-detached mesh deformation scheme based on a new surface representation, the *skin-detached surface*. To put it simply, the *skin-detached surface* representation decomposes the mesh into a simplified surface and a group of displacement

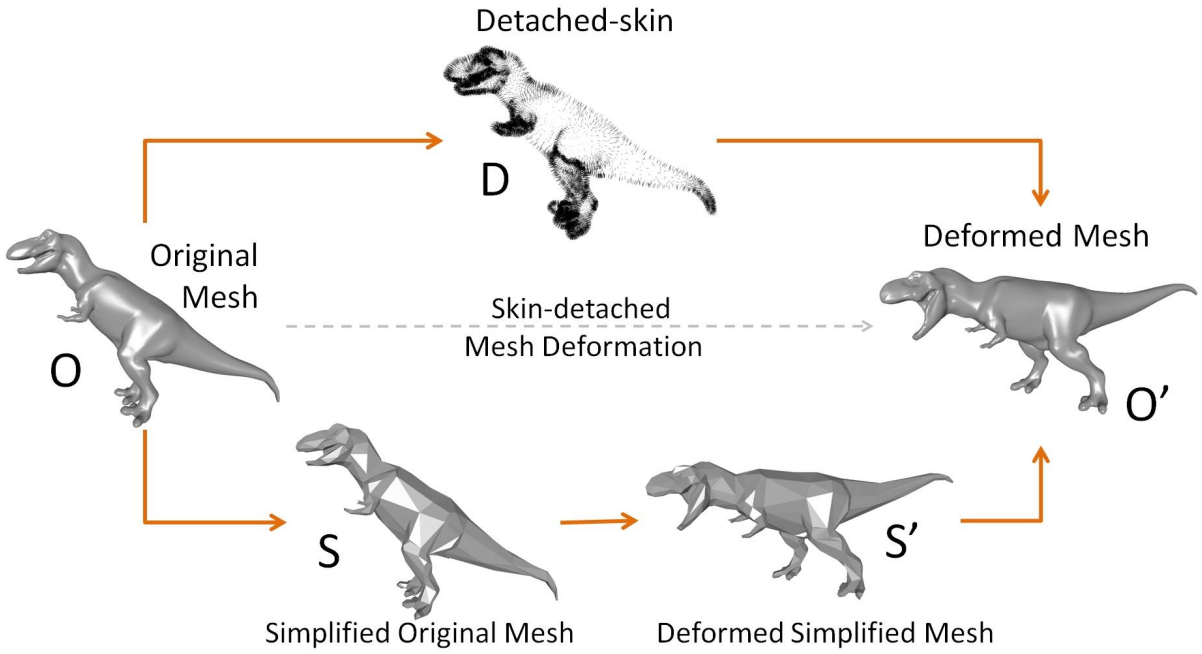


Figure 1: The workflow of skin-detached mesh deformation.

detail coefficients, allowing global deformations with fine-scale detail preservation. This representation is inspired by the key idea of displacement mapping, which was originally used for accelerating the rendering process of fine-scale detail on a mesh.

The whole deformation process is depicted in Figure 1. The main computational load of our skin-detached deformation scheme lies in extracting the displacement coefficients from the original surface in the precomputation step, and it is actually the simplified surface on which the deformation is applied, thus high performance and interactive editing on common PCs can be achieved using our method. In addition, our method completely seperates the detail preservation mechanism from the mesh deformation process, resulting in an advantage that it can be easily combined with different mesh deformation approaches to satisfy various types of requirements while preserving the mesh detail.

In summary, there are two main contributions in this paper: 1. We introduce a new surface representation, *skin-detached surface*. This representation separates the details of the surface from the basic shape, and can be easily applied to efficient manipulation or animation of large detail meshes. The main challenge of designing the *skin-detached surface* representation lies in the diversity of mesh topologies. 2. We propose an efficient surface deformation scheme based on our surface representation, which integrates the strengths of the state-of-the-art deformation techniques and multiresolution techniques to achieve both detail-preserving deformation and high performance.

2 Background

Among the various existing mesh deformation approaches, gradient domain mesh deformation appears to be the one which can best represent the state-of-the-art deformation solutions. While the other deformation approaches have the common limitation of detail

distortion, the gradient domain mesh deformation directly encodes the geometric details in differential coordinates, the so called Laplacian coordinates. By preserving the computed Laplacian coordinates, the mesh details can be well preserved during deformation. Another advantage of gradient domain mesh deformation is that it provides an intuitive way of mesh editing. The users can directly drag the handles on the mesh to desired positions to deform the mesh. By casting the detail preservation as an energy minimization problem, the deformation can be propagated through the mesh to achieve high quality deformation result.

Many researchers have contributed in improving gradient domain mesh deformation. The existing techniques can be categorized into two classes: linearization methods and nonlinear optimization techniques. Linearization methods dominated in the early stage development. This class generally includes Poisson mesh editing [8], harmonic propagation methods [9, 10], Laplacian mesh editing [11, 12, 13], as-rigid-as possible (ARAP) [14] methods, volume graph laplacian [15] and multi-grid deformation algorithm [16]. Nonlinear methods tackle the limitation of solving large rotation in lineariazation methods by automatically deriving the rotation from pure translation, which reduces the user burden significantly. Current nonlinear methods include subspace gradient deformation [17], dual Laplacian coordinates method [18] and subdivision surface deformation method [19].

In recent years, with the development of 3D scanning techniques, highly detail surface models are becoming commonplace. This directly results in the emergence of an issue with current deformation techniques, i.e., how to achieve detail-preserved deformation for large meshes with high performance. To the best of our knowledge, this problem has not been solved well yet. To tackle this problem, our method combines the strengths of gradient domain techniques and multiresolution techniques to achieve visually pleasing deformation and high performance. Our method is similar in spirit to the displacement mapping technique.

Displacement mapping was introduced by Cook [20] as a rendering technique in computer graphics. It utilizes a texture or height map to produce an effect where the actual geometric position of points over the surface are displaced. The idea of displacement mapping is widely used in multiresolution techniques, of which one representative technique is *displaced subdivision surfaces* [21]. In this technique, Lee et al. enhanced the expressive power of integrating displacement mapping into the subdivision framework. Based on Lee's work, Zhou et al. [19] proposed a GPU-based method for interactive deformation of subdivision surfaces, which upgrades the performance of the previous deformation methods. The difference between our deformation method and Zhou's method is that we maintain the original topology of the mesh and can recover the mesh details accurately. There are several advantages of maintaining the original topology of mesh: 1. The users can save the current deformation for further edit. 2. For textured models, the texture coordinates can be preserved after the deformation is applied. 3. The deformation results can be easily used for mesh interpolation or animation. In addtion, the simplified mesh in our method has much lower complexity than the subdivision surfaces in Zhou's [19], which endows our method with higher performance.

3 Method overview

As depicted in Figure 1, our method consists of the following steps:

- **Mesh simplification** ($O \longrightarrow S$) After loading the original mesh into memory, the first step of processing is simplifying the mesh. This step is done by system automatically without user's intervention. The simplified mesh will then be used for "skin" coefficients computation as well as actual shape deformation.

- **Skin coefficients computation** ($O \ominus S \longrightarrow D$) Computing "skin" coefficients of the original mesh based on the simplified mesh is a nontrivial task. The main challenge lies here is that all the geometric relationships of different meshes should be taken into consideration to make our method as general as possible. In the following sections, we investigate whether it is possible to handle all of the situations using automatic computation, and describe some exceptional cases which are out of the capability of our algorithm.

- **User-directed mesh deformation** ($S \longrightarrow S'$) The users can directly manipulate the handles on the mesh in this step to achieve their desirable results. We employ the nonlinear gradient domain deformation technique to respond to users' input, deforming the mesh according to the position constraints and the volumetric constraint.

- **Detail mesh reconstruction** ($S' \oplus D \longrightarrow O'$) After each iteration of the user's manipulation, we reconstruct the details of the mesh by composing the precomputed "skin" coefficients and the simplified mesh, obtaining the deformed detail mesh. This step is basically a reverse of the "skin" computation step.

In the following sections, we elaborate each step of our method with more technical details.

3.1 Mesh simplification

To obtain the nude mesh without "skin" on it, we first simplify the original mesh to a

(a) (b) (c)

Figure 2: (a): The original Armadillo. (b): The simplified model using only quadric error. (c): The simplified model by our system, note the claws and ears are better simplified.

coarse level. We simplify the original mesh using a sequence of edge collapse transformations prioritized according to the quadric error metric which was first proposed by Garland and Heckbert [22].

To handle the case in which one vertex has multiple set of displacement coefficients (see the next section), we restrict some of the candidate edge collapses. We set the main objective as ensuring that the resulting simplified mesh should have a locally similar space of normals with the original mesh. We borrow the 1-ring neighborhood test method introduced by Aaron et al. [21] for this task. If the test fails on any face in the 1-ring neighborhood, the edge collapse transformation is denied.

Figure 2(b) and Figure 2(c) give the simplified meshes of Armadillo model (Figure 2(a)) without and with 1-ring neighborhood test respectively. The original Armadillo contains 345,944 triangles while the simplified meshes contain only 1,000 triangles each.

3.2 Skin-detached surface representation

A skin-detached surface representation consists of a simplified mesh and a group of displacement coefficients (see Figure 3). The large green triangle is one of the triangles in the simplified bunny mesh, and the blue mesh above it is the corresponding part of the detail bunny. Each vertex of the detail mesh is displaced from a specific point (the black points) on the simplified triangle along the normal direction of the triangle.

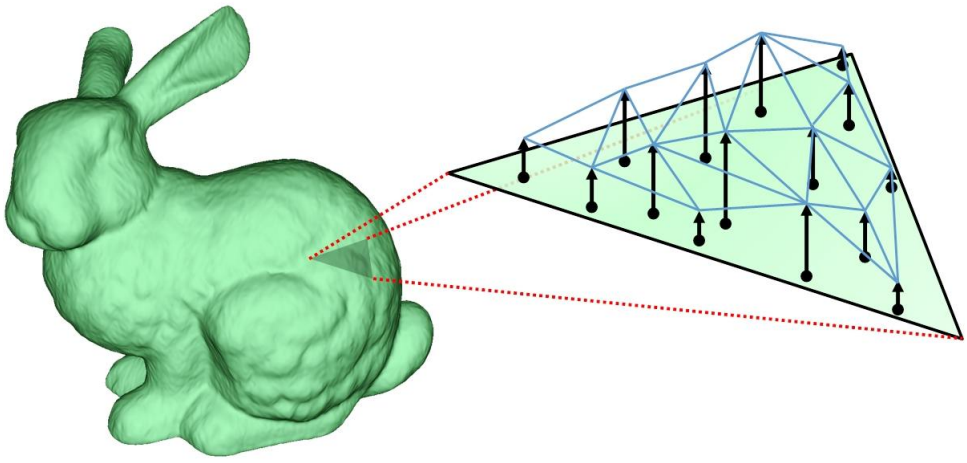


Figure 3: Vertices' displacements from the coarse level triangle of the simplified mesh.

We define a set of displacement coefficients for a specific vertex V as:

- ***Triangle index*** This parameter stores the index of the simplified triangle in which the vertex V falls along the normal direction. To put it iconically, it is the piece of flesh where the "skin" point V attaches.

- ***UVW coordinates*** *UVW coordinates* supply the information of where the droppoint (the black points in Figure 3) lies in the simplified triangle. Suppose P_1, P_2, P_3 are the three sequenced vertices of the simplified triangle, then we can compute the coordinates of the droppoint by $V' = u \cdot P_1 + v \cdot P_2 + w \cdot P_3$.

- ***Distance*** The distance between the vertex V and the point of fall V'. Given the distance d and the triangle normal $\hat{n}$, we can reconstruct the original vertex position by $V = V' + d \cdot \hat{n}$.

Note that our method maintains the original topology of the mesh in the displacement coefficients which is different from the *displaced subdivision surfaces*, therefore the reconstruction of the detail mesh using our method is absolutely accurate.

Figure 3 shows only the ideal relationship between the detail mesh and the simplified triangle, but the situation can be much more complex in practice. Therefore we study the diverse relationships between vertices of the detail mesh and triangles of the simplified mesh, and conclude them into three categories as following.

Case one

Figure 4 shows the 2D view of the first case. It is the ideal case that the vertex V of the detail mesh falls within one and just one triangle of the simplified mesh. It is the same case as shown in Figure 3. The *normal volume* is a volume that is swept by the translated triangle along its normal direction. The vertex V of the detail mesh lies in and only in the normal volume of triangle 3. This case is the simplest case, and the displacement coefficient of the vertex V can be computed straightforwardly.

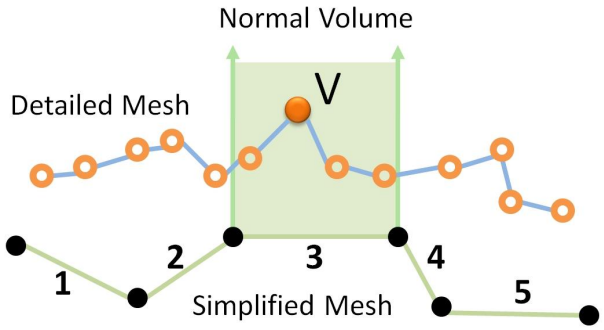


Figure 4: First case: The vertex falls within one and only one triangle along the triangle's normal (2D view).

Figure 5 shows the 3D view of the first case. Calculating the set of displacement coefficients for vertex V can be cast as a problem of Ray/Triangle intersection. We denote the triangle $\triangle P_1P_2P_3$ as T and the plane in which T lies as π. To get the intersection of R and T, we first need to determine the intersection of R and π. In order to obtain the UVW coordinates of vertex V during the intersection computation, we use a method that first computes the parametric coordinates of the intersection point in the plane π, and then determines whether the intersection point lies in or out of the triangle T. The parametric equation of plane π is given by:

$$V(s,t) = P_1 + s \cdot (P_2 - P_1) + t \cdot (P_3 - P_1) = P_1 + s\hat{u} + t\hat{v}, \tag{1}$$

where s and t are real numbers, and $\hat{u} = (P_2 - P_1)$ and $\hat{v} = (P_3 - P_1)$ are edge vectors of T. A point $V' = V(s,t)$ is in the triangle T when $s \geq 0$, $t \geq 0$, and $s + t \leq 1$. So, given V', we just have to find the (s,t) coordinate for it, and then check these inequalities to verify inclusion in T. Further, a point $V' = V(s,t)$ is on an edge of T if one of the

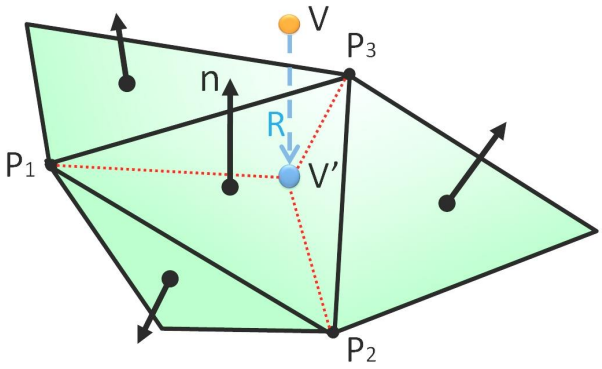


Figure 5: 3D view of the first case.

conditions $s = 0$, $t = 0$, or $s + t = 1$ is true (each condition corresponds to one edge). Also, the three vertices are given by: $P_1 = V(0, 0)$, $P_2 = V(1, 0)$, and $P_3 = V(0, 1)$.

To solve for (s, t), we use a 3D generalization of Hill's *"perp-dot" product* [23]. For plane π with a normal vector n, and any vector a in the plane (i.e., a satisfies $n \cdot a = 0$), the *generalized perp operator* on π is: $a^{\perp} = n \times a$. We use this *generalized perp operator* to solve the plane's parametric equation for the intersection point V'. Put $\hat{w} = V' - P_1$, we want to solve the equation: $\hat{w} = s \cdot \hat{u} + t \cdot \hat{v}$ for s and t. Take the dot product of both sides with $\hat{v}^{\perp}$ to get: $\hat{w} \cdot \hat{v}^{\perp} = s\hat{u} \cdot \hat{v}^{\perp} + t\hat{v} \cdot \hat{v}^{\perp} = s\hat{u} \cdot \hat{v}^{\perp}$, and solve for s. Similarly, taking the dot product with $\hat{u}^{\perp}$, we can solve for t. We get:

$$s = \frac{\hat{w} \cdot \hat{v}^{\perp}}{\hat{u} \cdot \hat{v}^{\perp}} = \frac{\hat{w} \cdot (\hat{n} \times \hat{v})}{\hat{u} \cdot (\hat{n} \times \hat{v})}, \quad t = \frac{\hat{w} \cdot \hat{u}^{\perp}}{\hat{v} \cdot \hat{u}^{\perp}} = \frac{\hat{w} \cdot (\hat{n} \times \hat{u})}{\hat{v} \cdot (\hat{n} \times \hat{u})}. \tag{2}$$

Since the intersection is computed many times, we want better performance than the cross products used in Equation 2. Therefore we apply the left association identity (for any three vectors a, b, c, we have $(a \times b) \times c = (a \cdot c)b - (b \cdot c)a$.) to the above equations, resulting in:

$$\hat{u}^{\perp} = \hat{n} \times \hat{u} = (\hat{u} \times \hat{v}) \times \hat{u} = (\hat{u} \cdot \hat{u})\hat{v} - (\hat{u} \cdot \hat{v})\hat{u}, \tag{3}$$

$$\hat{v}^{\perp} = \hat{n} \times \hat{v} = (\hat{u} \times \hat{v}) \times \hat{v} = (\hat{u} \cdot \hat{v})\hat{v} - (\hat{v} \cdot \hat{v})\hat{u}. \tag{4}$$

Then we can compute s and t using only dot products as:

$$s = \frac{(\hat{u} \cdot \hat{v})(\hat{w} \cdot \hat{v}) - (\hat{v} \cdot \hat{v})(\hat{w} \cdot \hat{u})}{(\hat{u} \cdot \hat{v})^2 - (\hat{u} \cdot \hat{u})(\hat{v} \cdot \hat{v})}, \quad t = \frac{(\hat{u} \cdot \hat{v})(\hat{w} \cdot \hat{u}) - (\hat{u} \cdot \hat{u})(\hat{w} \cdot \hat{v})}{(\hat{u} \cdot \hat{v})^2 - (\hat{u} \cdot \hat{u})(\hat{v} \cdot \hat{v})}. \tag{5}$$

where the two denominators are the same and only need to be calculated once. Since we precompute and store the normal vectors for all triangles when the mesh is first loaded, this ray/triangle intersection algorithm would not compute a cross product at all, making it very efficient. After obtaining the (s, t), we can simply compute the UVW coordinates by $u = 1 - s - t, v = s, w = t$. The *triangle index* and *distance* coefficients are very easy to obtain.

Case two

Figure 6 shows the 2D view of the second case. In this case, the vertex V does not fall within any triangle. This case usually happens when the vertex V is near to a summit

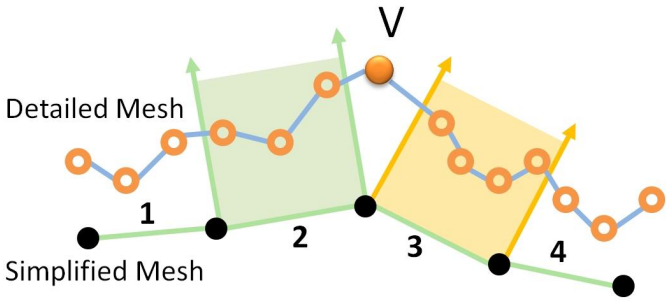


Figure 6: Second case: The vertex does not fall within any triangle (2D view).

of the mesh. It is more complex than the first case since it is not clear which simplified triangle should be associated with the vertex V.

To deal with this case, we tested several ways of selecting one associated triangle for V, e.g., associating the vertex with one of the closest triangles, but none of them results in visually pleasing deformation. Therefore we adopt a strategy that associates the vertex V with multiple triangles. We first traverse over the simplified mesh to find the vertex P_3 which is the closest to vertex V (see Figure 7), then we associate vertex V with all the neighbour triangles of vertex P_3.

As illustrated in Figure 7, we take the highlighted triangle $\triangle P_1P_2P_3$ (T) as an example. First, we emit a ray from vertex V along the normal direction of T, and compute the intersection point V' on the extended plane of T (the big blue triangle in Figure 7) by applying the same Ray/Triangle intersection scheme introduced in case one. The only difference with the first case is that the intersection vertex V' must lie outside of T, therefore it must be one of the following cases: $s \leq 0$ or $t \leq 0$, or $s + t \geq 1$. But it does not affect our computation, and we can still compute the UVW coordinates by $u = 1 - s - t, v = s, w = t$. In this way, we associate vertex V with triangle T, and the other neighbour triangles are associated in the same way, resulting in multiple set of displacement coefficients for one vertex. When the simplified mesh is deformed, the deformed position of vertex V is determined by all these associated triangles.

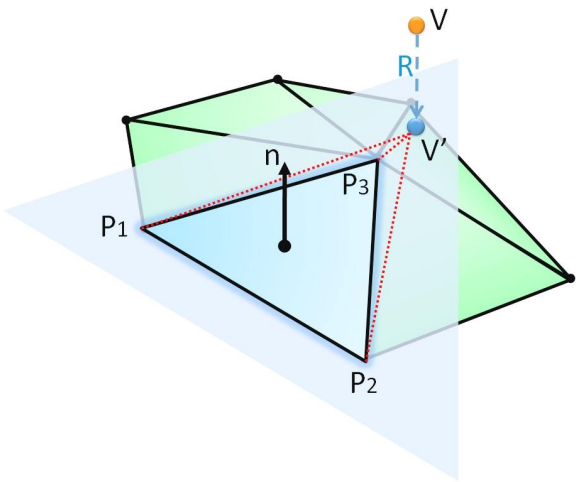


Figure 7: 3D view of the second case.

Case Three

Figure 8 shows the 2D view of the third case. In this case, vertex V falls within multiple triangles along their normals. This case usually happens when the vertex v is near to a concave part of the mesh. It is actually the general case of the first case, and we only need a scheme for selecting triangles to associate.

We first compute the displacement coefficients for each of the triangles within which the vertex V falls. Then we sort these triangles according to their corresponding *distance* coefficients. Note that the *distance* coefficient is a signed value, therefore we sort the coefficients by just their absolute values. Suppose the shortest *distance* of all these triangles is d_{min}, we then associate vertex V with all of the triangles which have the *distance* coefficient less than $2 \times d_{min}$. The rest of the computation is the same with the first case, and we also store multiple sets of displacement coefficients for one vertex like the second case.

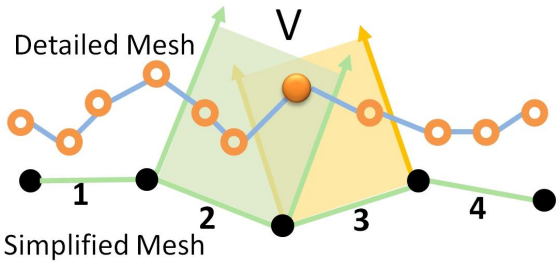


Figure 8: Third case: The vertex falls within multiple triangles along their normals (2D view).

3.3 Dual laplacian mesh deformation

To ease the users' manipulation of mesh deformation, we employ the dual Laplacian mesh deformation technique proposed by Au et al. [18]. The dual Laplacian method encodes both the local parameterization and geometric details in the dual mesh domain, thus it can reduce the distortion caused by large angle deformation or the translations of handles.

The dual mesh consists of the vertices located at the centroids of all the original mesh faces, and the connectivity of these vertices is determined based on the original face adjacency. Therefore the valence of dual mesh vertices will always be three for a triangular mesh, which means the one-ring neighborhood of a vertex will always be coplanar. Consequently the dual Laplacian coordinate of a vertex is exactly in the normal direction of the triangle formed by the one-ring neighborhood. The advantage of this is that it can avoid the so-called tangential drift effect. Precisely, a vertex i can be determined using:

$$\hat{v}_i = \hat{p}_i + h_i \hat{n}_i = \sum_{j \in 1,2,3} \hat{w}_{i,j} \hat{v}_{i,j} + h_i \hat{n}_i, \tag{6}$$

where $\hat{p}_i$ is the normal projection of vertex i onto the triangle $\triangle \hat{v}_{i,1} \hat{v}_{i,2} \hat{v}_{i,3}$ formed by its one-ring neighborhood, h_i is the signed height of vertex i from the triangle, and $\hat{n}_i$ is the outward normal. Then $\hat{p}_i$ is expressed by its barycentric coordinates $\hat{w}_{i,j}$. We can derive the dual Laplacian coordinate of vertex i easily as:

$$\hat{\delta}_i = h_i \hat{n}_i = \hat{v}_i - \sum_{j \in 1,2,3} \hat{w}_{i,j} \hat{v}_{i,j}. \tag{7}$$

We can put it into the matrix form as $\hat{L}DV = \hat{\delta}$ where $\hat{L}$ is the topological Laplacian of the mesh [24], and D is the dual mesh construction matrix constructed from the vertex-face incident matrix [25]. Let V^k and $\hat{\delta}^k$ be the vertex coordinates matrix and the dual Laplacian coordinates computed at interation k respectively. We iteratively update them using two linear steps until convergence:

- Compute the new vertex position V^{k+1} with $\hat{\delta}^k$ by solving the following linear least-squares problem:

$$min \|\hat{L}DV^{k+1} - \hat{\delta}^k\|^2 + \|CV^{k+1} - p\|^2, \tag{8}$$

  where C is a vector consists of 0 and 1 to represent which vertex is selected as handle and p is the handles' positional constraints.

- Update the dual Laplacian coordinates using:

$$\hat{\delta}_i^{k+1} = h_i \hat{n}_i^{k+1}, \tag{9}$$

  where $\hat{n}_i^{k+1}$ is the unit normal of the triangle formed by one-ring neighborhood of v_i, computed using the V^{k+1} obtained above.

We set the termination condition as that the MSE (mean square error) of the vertex positions between two successive iterations is less than a given threshold. When the termination condition is satisfied, the algorithm stops and the deformed mesh is obtained.

3.4 Detail mesh reconstruction

The detail mesh reconstruction procedure is basically a reverse procedure of the *skin-detached surface* computation, it is like wrapping the "skin" back onto the simplified mesh after deformation is applied. Given a deformed simplified mesh and the displacement coefficients for all the original vertices, we can efficiently compute the deformed position of vertex V of the detail mesh. For the first case, the deformed position of vertex V can be calculated as:

$$V_{deformed} = u \cdot P_{i,1} + v \cdot P_{i,2} + w \cdot P_{i,3} + d \cdot \hat{n}_i, \tag{10}$$

where $i, (u, v, w)$ and d are the set of displacement coefficients of vertex V. $(P_{i,1}, P_{i,2}, P_{i,3})$ and $\hat{n}_i$ are the three vertices and the normal of the triangle i from the deformed simplified mesh.

For both the second and the third cases, we adopt a weighted mean value scheme to reconstruct the deformed vertex V. The weight of each set of displacement coefficients is determined by the area of the corresponding simplified triangle:

$$V_{deformed} = \frac{\sum_{i=1}^{n} S_{\triangle_i} (u \cdot P_{i,1} + v \cdot P_{i,2} + w \cdot P_{i,3} + d \cdot \hat{n}_i)}{\sum_{i=1}^{n} S_{\triangle_i}}, \tag{11}$$

where n stands for the count of corresponding triangles and $S_{\triangle_i}$ is the area of the ith corresponding triangle of the simplified mesh.

4 Experiment

We implemented a prototype system of the described algorithm on a 3.0Ghz PC with 2GB of memory using OpenGL and C++. With this system, the users can directly select vertices on the mesh as handles and manipulate them to achieve desirable deformation results. We employed *openmesh* [26] as the basis of our system for the sake of efficiency. *Openmesh* is a generic and efficient data structure for representing and manipulating polygonal meshes. It wraps complex internal structure in an easy-to-use interface and maximize the time efficiency as well as the memory usage. We embedded it in our system easily by adding the properties of displacement coefficients to geometric element classes in *openmesh*.

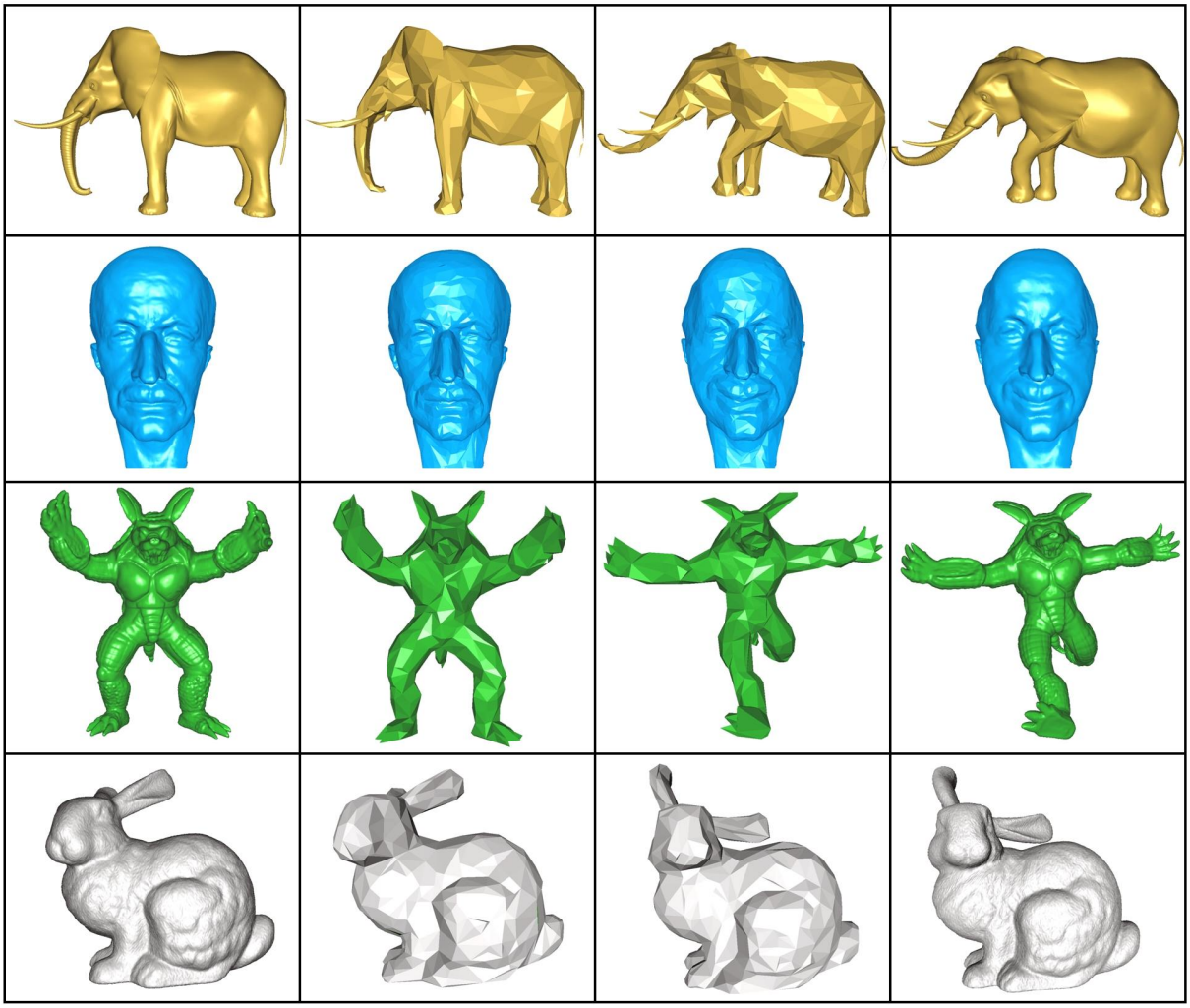


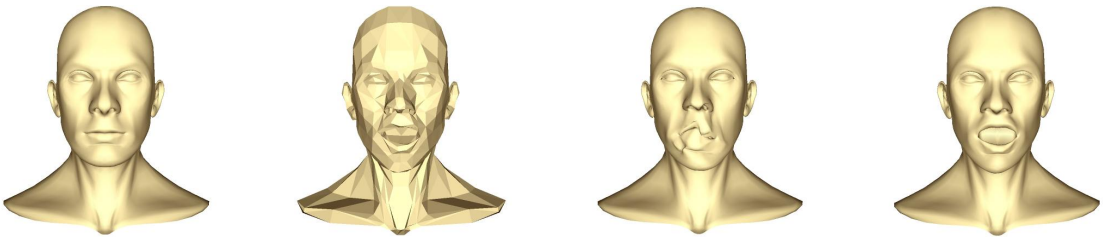
Figure 9: 1st column: Original detail meshes; 2nd column: Simplified meshes; 3rd column: Deformed simplified meshes; 4th column: Detail mesh deformation results.

At the loading stage, the system first reads the mesh that needs to be deformed into memory, and then it will do two things automatically: simplifying the mesh to a certain coarse level (in our implementation, we set the default face count of the simplified mesh as 1000. The users can change the number due to their specific requirements), then

computing and storing the "skin" of the original mesh based on the simplified mesh. After all these computations, the users can intuitively edit the large mesh. At each time the users drop the handle, the system reconstructs the detail mesh according to the deformed simplified mesh. Since the computation of mesh simplification, "skin" decomposition and detail mesh reconstruction are hided in a black box, it appears to the users as if they are editing the detail mesh directly.

Figure 9 shows the deformation results of four large detail mesh using our system. The face counts of the original four meshes are 157K, 398K, 332K and 278K respectively. With conventional gradient domain mesh deformation techniques, the computation of these meshes would be complex and time-consuming. This makes the deformation work quite difficult for the artists. But with our method, the artists can edit large meshes interactively. For the four different models, we set the vertex number of the simplified models to be different according to the complexity of their shapes. Specially, for the head model, due to people's sensibility to human facial detail, we simplified the model to 5,000 triangles. Our system achieved visually pleasing deformation results for these models.

During our experiment, we found that there are some exceptional cases in which our method can not achieve correct deformation. Figure 10 gives a case with a human head model. Figure 10(a) shows the original head model with mouth closed. Figure 10(b) and 10(c) show the deformed simplified model and the reconstructed detail model whereas Figure 10(d) shows the desirable result. When we tried to drag the lower lip from the upper lip, the shape of mouth was heavily distorted. The reason for this phenomenon is that when we computed the *skin-detached surface* representation of the head model, some vertices of the upper lip were mistakenly attached to the simplified lower lip, and vice versa. Therefore, when we reconstructed the detail mesh of the head, these vertices were determined by improper simplified triangles and then the distortion happened. Generally speaking, when we want to split a group vertices that are close to each other, we need to be careful enough about the computation of *skin-detached surface* representation. So we believe that using only automatic method is not enough to cover all the situations for this task.



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

Figure 10: (a) Original human head; (b) Deformed simplified model; (c) Reconstructed detail model; (d) Desirable result.

Table 4 provides some performance statistics of the four examples shown in Figure 9. For these large models, traditional deformation methods typically take several seconds or longer for a single manipulation of moving one handle, and take even longer for the recomputation each time when the users change the handles. But we can see that our skin-detached mesh deformation system can achieve real-time performance for all these examples. The frame rates in the table were calculated when the handles are already

Model	$\#V_{simplified}$	$\#V_{detail}$	FPS	Precomputation Time
Elephant	700	157,160	86	21min
Max-Planck Head	5,000	398,043	34	64min
Armadillo	1,000	331,904	75	52min
Detail Bunny	500	277,804	92	44min

Table 1: Performance for the examples used in this paper, including the vertex numbers of simplified meshes and detail meshes, the frame rates of deformation and the precomputation time cost.

selected and the user is manipulating certain handles. If the user adds new handles or removes old handles, then we need to re-compute the inverse for some matrices and the frame rate will decrease. But fortunately these matrices' dimensions are proportional to the vertex count of the simplified mesh, which is much smaller than the detail mesh. For all the above examples, this recomputation process takes less than one second. The topological Laplacian and the dual mesh construction matrices are fixed during deformation and are not affected by the handle selections.

Note that the real performance bottleneck of our method lies in the precomputation step of generating *skin-detached surface* representation. In its last column, table 1 also provides the precomputation time of each model. Although the precomputation takes relatively long time for large models, it is acceptable since for each model we only need to compute the *skin-detached surface* representation once, and we can save the displacement coefficients for further usage. In addition, we believe that the performance of precomputation can be improved by transferring the parallel computation onto GPU, which would be an area for future work.

5 Conclusions and future work

We have described a method for interactive deformation of large detail mesh. With our method, the user can directly manipulate large meshes using freely-selected surface points as handles. The most important feature of our algorithm is that it combines the strengths of the nonlinear gradient domain mesh deformation techniques and the multiresolution techniques to achieve both visually pleasing deformation and high performance. Specifically, by separating the detail recovering mechanism from the mesh deformation procedure, our system achieves the advantage that it can be easily combined with different mesh deformation approaches to satisfy various requirements. While significant computation is needed for large detail mesh deformation with traditional methods, our method, designed with *skin-detached surface* representation, distributes the main computational load to the precomputation step and achieves real-time performance on common PCs.

In mesh deformation, it is well known that designing an algorithm suitable for all the topological structures of different meshes is nearly impossible. As illustrated in the experiment section, there are still some limitations of our method. As a topic of future research, we plan to explore the solution to the exceptional cases of our system. Besides, our current system only supports triangular mesh deformation, and we are also interested in developing techniques for multi-component models or non-manifold models. Another

area for future work is improving the current performance of precomputation by applying the parallel GPU computation to the *skin-detached surface* computation.

References

[1] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Computer Graphics*, 20(4):151–160, 1986.

[2] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 630–634, New York, NY, USA, 2004. ACM.

[3] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005.

[4] Karan Singh and Eugene L. Fiume. Wires: A geometric deformation technique. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 405–414, July 1998.

[5] Qunsheng Peng, Xiaogang Jin, and Jieqing Feng. Arc-length-based axial deformation and length preserved animation. In *CA '97: Proceedings of the Computer Animation*, page 86, Washington, DC, USA, 1997. IEEE Computer Society.

[6] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[7] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 205–214, New York, NY, USA, 1987. ACM.

[8] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heungyeung Shum. Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 644–651, New York, NY, USA, 2004. ACM.

[9] Rhaleb Zayer, Christian Rössl, Zachi Karni, and Hans-Peter Seidel. Harmonic guidance for surface deformation. *Computer Graphics Forum*, 24(3):601–609, 2005.

[10] Tiberiu Popa, Dan Julius, and Alla Sheffer. Material-aware mesh deformations. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*, page 22, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Marc Alexa. Local control for mesh morphing. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 209, Washington, DC, USA, 2001. IEEE Computer Society.

[12] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, David Levin, Christian Rössl, and Hans-Peter Seidel. Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International*, pages 181–190. IEEE Computer Society Press, 2004.

[13] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 179–188. Eurographics Association, 2004.

[14] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Transaction on Graphics*, 24(3):1134–1141, 2005.

[15] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. *ACM Transaction on Graphics*, 24(3):496–503, 2005.

[16] Lin Shi, Yizhou Yu, Nathan Bell, and Wei-Wen Feng. A fast multigrid algorithm for mesh deformation. *ACM Transaction on Graphics*, 25(3):1108–1117, 2006.

[17] Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Subspace gradient domain mesh deformation. *ACM Transaction on Graphics*, 25(3):1126–1134, 2006.

[18] Oscar K.C. Au, Chiew-Lan Tai, Ligang Liu, and Hongbo Fu. Dual Laplacian editing for meshes. *IEEE Transaction on Visualization and Computer Graphics*, 12(3):386–395, 2006.

[19] Kun Zhou, Xin Huang, Weiwei Xu, Baining Guo, and Heung-Yeung Shum. Direct manipulation of subdivision surfaces on gpus. *ACM Transaction on Graphics*, 26(3):91, 2007.

[20] Robert L. Cook. Shade trees. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231, New York, NY, USA, 1984. ACM.

[21] Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced subdivision surfaces. In *Proceedings of ACM SIGGRAPH 2000*, pages 85–94, July 2000.

[22] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[23] F. S. Hill, Jr. The pleasures of "perp dot" products. *Graphics gems IV*, pages 138–148, 1994.

[24] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[25] G. Taubin. Dual mesh resampling. *Graphical Models*, 64(2):94–113, March 2002.

[26] Botsch Steinberg Bischoff, M. Botsch, S. Steinberg, S. Bischoff, L. Kobbelt, and Rwth Aachen. Openmesh – a generic and efficient polygon mesh data structure. In *In OpenSG Symposium*, 2002.