GhostMesh: Cloud-Based Interactive Mesh Editing

Zhong Zhou*, Fei Dou, Yi Li, and Lin Zhang

State Key Laboratory of Virtual Reality Technology and Systems
Beihang University, Beijing, China
{zz,zhanglin}@vrlab.buaa.edu.cn

Abstract. Interactive deformation of complex meshes is a challenging task in modeling and animation, due to high computation and storage requirements. The emerging cloud computing technology, as well as cloud storage, provides a possible consumer-level solution so that interactive mesh editing could work in light-weighted clients with public cloud services. In this paper we present a system for efficient interactive editing of complex models. Building over a client-server-based distributed architecture, our system transfers computationally heavy tasks (e.g., computing of blending weights, construction of handles) and storage from the client to a cloud computing environment, and allows the user to edit a complex mesh that stored in the cloud server by interactively modifying a simplified mesh locally. When the user specifies coarse deformations on the client, user manipulations are sent to the cloud server, and the original complex mesh is deformed accordingly in the background. Experiments conducted on various complex models show that our system achieves very efficient performance, and the most important, remains fully compatible with the classical WYSIWYG deformations.

Keywords: interactive graphics, shape deformation, 3D modeling.

1 Introduction

The modeling and animation of complex geometric models has become a hot topic today. Although much progresses have been made over the years on 3D modeling systems, effective interactive deformation of complex objects is still a challenging task, due to limitations of computation capability and storage resources.

In previous research, a fair amount of techniques have been developed for the purpose of overcoming these challenges. Among them, space deformation and multi-resolution shape deformation are most representative. To reduce the computational cost of shape deformation, space deformation methods deform a low polygon-count polyhedron in which a shape is embedded, instead of explicitly deforming the shape itself. The main advantages of space deformation techniques lie in the simplicity and speed. Multi-resolution shape deformation is

* Corresponding author.

B. Huet et al. (Eds.): PCM 2013, LNCS 8294, pp. 384–395, 2013.

another preferable choice for interactive mesh editing. Different from space deformation methods that introduce an additional ambient space, multi-resolution techniques, such as subdivision surfaces, organize the objective meshes at different levels of detail and perform hierarchical editing to achieve efficiency. However, both space deformation methods and multi-resolution deformation techniques still strongly depend on the local computing power, meaning that deformations of large complex objects would be still not available on a device with low computing power or storage.

Cloud computing has been a hot topic of computing paradigm in the last years. It transfers heavy computing and storage from the client to a cloud computing environment. Enterprises currently employ cloud services to improve the scalability of their services and to deal with bursts in resource demands. In this work, we are interested in an interactive mesh editing system based on cloud services. In particular, we aim at enabling the user to edit large complex models with the help of cloud server even using a terminal with low computing power and storage. Furthermore, we desire our system to provide interactive manipulation for users.

Our solution is a client-server-based system that transfers heavy computing and storage from the client to the cloud server. Specifically, a coarse deformation is specified on the client side by interactively editing a simplified model instead of the original complex model, while computationally heavy tasks, such as the computing of blending weights for handles and the deformation of the original complex model, are performed on the cloud server. We propose efficient strategies to classify and stream blending weights and user manipulations. As a result, the system allows the user to deform a large complex mesh only by editing a small simplified mesh interactively in local.

2 Related Work

In the past decades, a fair amount of research on surface deformation has been developed. Variational methods [1,2] require post time optimization, which are still too slow to deform high-resolution objects at high framerate. Variational harmonic maps [3] restrict the degrees of freedom to harmonic deformations of a specified cage although they are faster. Other methods using a weighted blend of handle transformations such as least squares [4], dual quaternions [5], and linear blend skinning (LBS) [6] which is most frequently used for real-time character deformation, are fast at pose time. Bounded biharmonic weights [7] is one of the techniques using LBS for efficient pose-time computation with great deformation result. Each vertex is transformed according to the affine transformations of the handles, which are linearly averaged with different weights.

Multi-resolution shape deformation is a very effective way to perform interactive mesh editing. B-spline is presented as a pioneering work on hierarchical editing by Forsey and Bartels [8], in which the user should decide where to add detail and to manipulate the corresponding controls. Wavelets enable multi-resolution editing on objects that can be represented by the hierarchical basis

functions [9]. Subdivision surfaces [10] is a preferable choice for interactive mesh editing, due to the possibility to edit mesh at varying levels of detail. For purpose of higher efficiency, a fair amount of research on simplification techniques has long been developed. Progressive mesh, first proposed by Hoppe [11], is a multi-resolution technique using edge collapse, which simplifies models by iteratively contracting edges with precise results and high performance. Among numerous edge collapse algorithms, quadric error metric (QEM) [12], which estimates the error introduced by a pair collapse as the distance from a vertex to a quadratic surface, is the one with high performance and usually the most precise results.

There have been numerous shape editing systems for designing of 3D models. iWIRES [13] and LineFFD [14] are analyze-and-edit approaches. They detect important features of a model to extract a descriptive set of lines and then users can utilize the feature lines to deform the model. Systems like Teddy [15] and FiberMesh [16] for quickly and easily designing freeform surfaces with a collection of 3D curves provide convenience for users. A user draws several 2D freeform strokes interactively on the screen and the system automatically convert them to 3D curves and constructs plausible 3D polygonal surfaces afterwards. When using Wires [17] or SilSketch [18], the user applies the deformation by giving nothing but the 'over-sketch' of feature lines to the system. The lines have to be drawn manually and the system will deform the shape according to them.

3 User Interface

The user interface of GhostMesh consists of a button panel and a rendering window (Fig. 1). The rendering window is embedded with controls for navigation and the capability of drawing viewport-aligned strokes. By clicking option buttons on the button panel and holding some meta keys, the user can drag the mesh along the horizontal and the vertical axies, rotate it and scale the current projection by clicking (left or right) and dragging the mouse.

To edit a large mesh, the system first streams a simplified version of the original large mesh from the cloud server to the client. The user determines an appropriate view, places control handles on the surface. Then the system binds the mesh to these handles. The user interactively manipulates these handles and the system deforms the mesh accordingly.

The system has three main tools for the mesh editing: handle tool, ROI tool, and deformation tool. The user switches between these tools via menu selection or a keyboard shortcut.

3.1 Handle Tool and ROI Tool

The handle tool lets the user construct a number of handles on the mesh. We aim to provide users a flexible and interactive way to manipulate large 3D mesh on the client side. In our system, three common handle types are supported: point, skeleton and control curve. A control curve is described as a set of points.

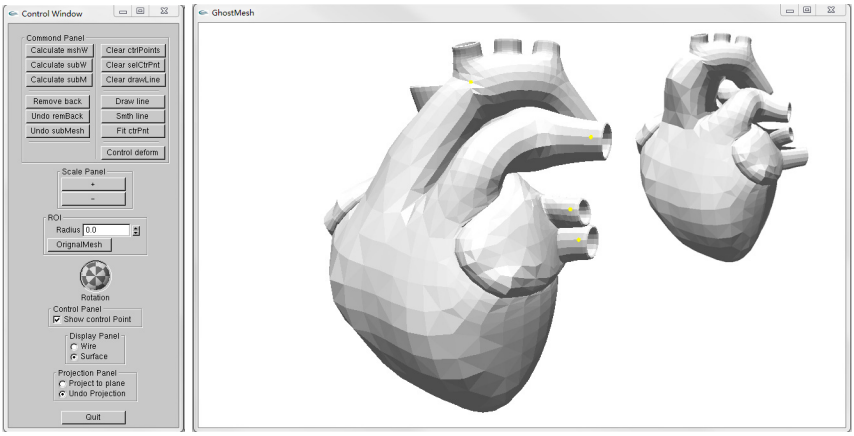


Fig. 1. User interface of our system

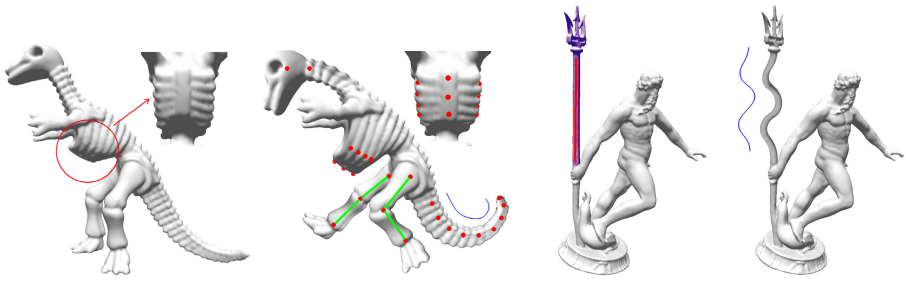


Fig. 2. Our mesh editing tool in action. GhostMesh supports points, skeletons, and control curves arranged in an arbitrary configuration. The user can specify translations, rotations and scales at handles (middle left). A sequence of point handles can be selected automatically by drawing a stroke on the surface (middle right), which would be deformed by drawing another control curve (rightmost).

To construct handles, the user clicks or drags on the simplified surface to specify a number of points. Fig. 2 shows illustration of our handle tools.

The ROI tool lets the user specify a certain part of mesh to conduct deformations on. After constructing handles, an optional ROI around specified handles can be computed by diffusion on the surface, and the ROI can be interactively adjusted to fit the target deformation.

3.2 Deformation Tool

The deformation tool lets the user deform the mesh by manipulating handles. One of our user interface for mesh deformation is a usual direct manipulation

method: the user grabs and drags a point (i.e., point handle, point of a skeleton and point on a curve), specifies translations, rotations and scales on the point, and the mesh deforms smoothly within the specified ROI. Generally, interactive manipulations of the handles (dragging a point or drawing a control stroke) are usually carried out on a 2D plane due to the limitation of 2D input devices and displays, as well as the 2D depth perception. Therefore, the current implementation always moves the points parallel to the screen. If the user wants more control, new handles must be added on the surface.

Another of our deformation tools is drawing a control stroke to manipulate a curve handle. Using this tool, the user draws a stroke which suggests a modification of the curve, and the shape is deformed according to the deformed curve handle. It is worth noting that because of the sketching strokes can contain noise, our system allows the user to smooth a control stroke or the deformed curve handle directly by brushing on the target curve with the mouse, which is intuitive and convenient.

By manipulating handles with our deformation tools, the simplified mesh is deformed in real-time, showing the user a coarse deformation. At the same time, the system streams the manipulation to the server side, and the original large mesh is deformed accordingly in the background.

4 Algorithm

To implement our interactive mesh editing and the described interfaces, we propose a cloud service which consists of four main parts: handle optimization, weight computing, data transmission, and deformation computation.

Before describing our algorithms, we introduce notations. Let $M = (V, E, F)$ be a given triangular mesh. V denotes the set of vertices, E denotes the set of edges and F denotes the set of faces. Let $\mathbf{e}_{ij}$ be the edge linking two distinct vertices $\mathbf{v}_i$ and $\mathbf{v}_j$. Let H_j $(j = 1, ..., m)$ be the control handles.

4.1 Handle Selection and Optimization

In current implementation we choose some of the mesh vertices as handle members. To select handle members, the user clicks or drags on the simplified mesh M_S to specify a number of points.

It is obvious that handles constructed on the original large mesh are more close to the user intention (see Fig. 3). Therefore, when the user picks 3D points on the simplified mesh M_S, the system streams these points to the cloud server, constructing handles on the complex surface.

When click or drag on M_S, 3D intersection points p_i are selected on faces f_{p_i} of the simplified mesh (3D mouse picking). In current implementation, any two of points p_i are guaranteed not in the same face of the simplified mesh M_S, avoiding redundancy. A path line L formed by a discrete set of points $\{p_i\}$ is constructed (we take the single point selected by clicking as a line with size 1). L is usually not a subset of the vertices of mesh M_S and the original large mesh

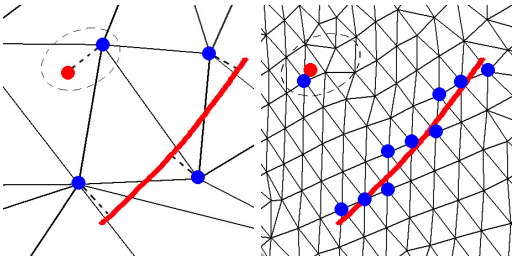


Fig. 3. Comparison between constructing handles on simplified (left) and original (right) surface. In both left and right figures, red shapes are user interactions, and blue points are the possible result handles.

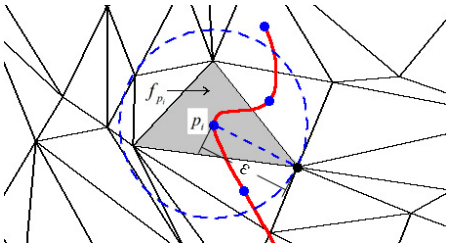


Fig. 4. Initial parameters of collecting p'_i

M. We define a line $L' = \{p'_i, p'_i \in V\}$ as minimizers of a shape-aware functional subject to:

$$\arg\min_{p'_i, i=1,\ldots,|L|} \sum_{i=1}^{|L|} \sum_{p_j \in \mathcal{N}_i^+} \|p'_i - p_j\|_2 \tag{1}$$

with $|L|$ the size of set $L = \{p_i\}$, and $\mathcal{N}_i^+$ the set of p_i and its geodesic neighbors in L. In practice we compute line L' by collecting p'_i through a flood filling process performed on M, starting at the first vertex of face f_{p_i} and restricted to the euclidean distance ε from it to p_i (Fig. 4).

When the user draws a stroke on the surface, the sampling density of handle members depends on that of the 3D mouse picking during dragging, and can be controlled according to the practical requirement. A skeleton point is constructed by moving a 3D intersection point inside into the volume.

4.2 Weight Computing

Real-time performance of weight computing for blending is critical for user experience as well. The less the user has to wait, the better the user experience will be. In our system, to shorten time spent waiting for binding, weights are

computed by the powerful cloud server. In current implementation, we use bounded biharmonic weights [7], which produces smooth and intuitive deformations, for real-time blending-based deformations. Since handles are selected on the original mesh M, handle members are usually not a subset of the vertices of the simplified mesh M_S. As discussed in [7], this is not consistent with the requirement of computation of bounded biharmonic weights. A direct solution is recomputing a new collapse list for PM that keeps all p_i' during edge collapse, meaning more computation and re-streaming of a new simplified mesh. Therefore, in current implementation, we directly discretize face f_{p_i} by adding edges between each vertex of f_{p_i} and the handle member p_i'.

As will be discussed further below, to make our system more efficient, some of the weights are transmitted over unreliable channels. Then missing weight of vertex v_i is estimated by interpolation as the formula below:

$$w_j(i) = \sum_{k \in \mathcal{N}(i)} \mu_{ik} \cdot w_j(k), \tag{2}$$

where w_j is the weight function associated with handle H_j, the function $\mathcal{N}(i)$ is the 1-ring neighbors of vertex v_i, and $\sum \mu_{ik} = 1$. In current implementation, the choice of neighbor weights

$$\mu_{ik} = \begin{cases} \frac{1}{|\mathcal{N}(i)|-1} \cdot \left(1 - \frac{\|v_i - v_k\|_2}{\sum_{l \in \mathcal{N}(i)} \|v_i - v_l\|_2}\right) \\ 0 \qquad \text{if } w_j(k) \text{ is missing} \end{cases} \tag{3}$$

defines the nature of w_j.

4.3 Data Transmission

Since manipulations are carried out on the simplified mesh on the client, all of them should be sent to the cloud server for the desired corresponding deformation on the original mesh. In order to facilitate data transmission and maintenance, we divide user manipulations into two types: intermediate and key manipulation. There are mainly two cases producing intermediate manipulations. The first case is the user chooses a subset of handles and specifies a same kind of manipulation (i.e., translations, rotations or scales) on them *multiple times* to get a desire deformation. The second case is the system records continuous changes of handles caused by just one user manipulation, such as the trajectory of moving a point by dragging. The relative key manipulation is the final result of a series of intermediate manipulations. A single manipulation is also considered a key manipulation. In fact, the same result of shape editing can be obtained by only computing these key manipulations in order, and the intermediate manipulations are only details of user editing process. Fig. 5 shows an illustration of the user's manipulations.

In our system, data transmission between the cloud server and the client includes: progressive meshes, creation of handles, ROI, blending weights, manipulations of handles and other auxiliary information such as weight computing

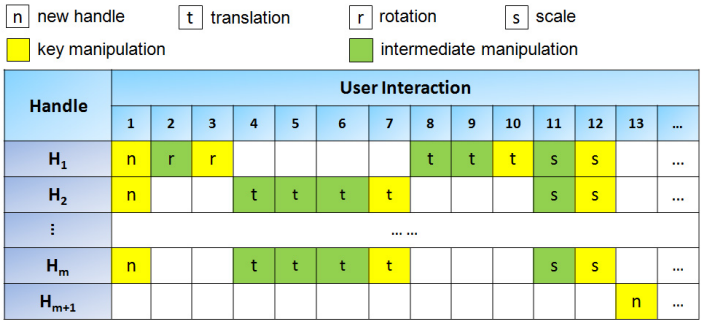


Fig. 5. Illustration of different types of manipulations. A key manipulation is a single manipulation or the final result of a series of intermediate manipulations of a subset of handles.

progress. In order to make our system more efficient, we send these data differently on hybrid transmissions to take the advantages of both TCP and UDP. The progressive meshes, creation of handles, ROI parameters, greater blending weights, and key manipulations of handles are encoded and transmitted over reliable channels, while the smaller blending weights and intermediate manipulations are delivered over unreliable channels.

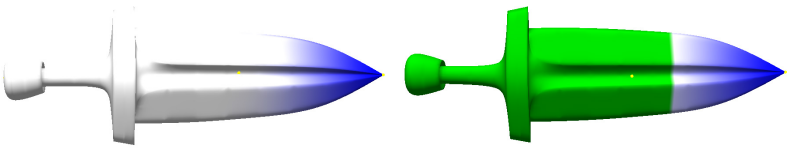
We use QEM [12] for streaming progressive meshes, a simplified mesh is firstly streamed from the cloud server to the client. The user specifies handle selecting and ROI parameters on this simplified mesh, and the system sends them to the cloud server. Then the server constructs handles, computes blending weights accordingly, and returns weights to the client.

In a typical mesh editing system, each handle has the maximum effect on its immediate region and its influence disappears in distant parts of the object, in other words, weights decay quickly. According to this phenomenon we send weights greater than an user-defined threshold δ over reliable channels, otherwise over unreliable channels (Fig. 6). There is no need to request retransmission of lost weight data. In practice we interpolate the missing weights with formula 2 through a simple dynamic process starting at the vertex with the most neighbors that have weight.

During editing, the client encodes the user manipulation, and sends them to the server. The encoded manipulation data includes the following information:

- The manipulation ID (numbered in ascending order);
- The manipulation type (i.e., intermediate or key);
- IDs of manipulated handles;
- The result parameters of each handle.

Since the key and intermediate manipulations are transmitted via reliable and unreliable channels respectively, two sorted lists are maintained on the server, and they can be easily retrieved and traversed as one according to the manipulation IDs. The cloud server processes only the key manipulations in order, or



(a) plot of weights (b) plot of weight groups

Fig. 6. 3D plot of weights of a point handle on the tip of the Knife model and their groups. Note that weights are smooth and local (a). Those smaller weights (in yellow) are transmitted over unreliable channels, and the others over reliable channels. Parameter value $\delta = 0.2$.

the intermediate manipulations as well, to computes the desired deformations of the complex original mesh.

5 Evaluation

In this section, we introduce the experiments results to evaluate GhostMesh. We implement the real-time 3D interactions and the proposed data transmission algorithm on a Windows XP PC with Intel Core2 Xeon 3.07GHz CPU and 4GB DDR3 RAM, and a Core2 Duo Laptop with a 2.5GHz CPU and 4GB RAM. Clients are tested on the Laptop. Laptop client communicates with the PC server through Wi-Fi.

Table 1. Statistics for the various examples in the paper. The binding time is measured in seconds. LOD is the percentage of visible vertices after mesh simplification.

Model	# of vertices	# of faces	Binding time per handle in different LODs			
			25%	50%	75%	100%
Heart	7349	14676	0.296	0.755	1.403	2.241
Sword	18001	35966	0.785	3.978	6.143	9.277
Dinosaur	56194	112384	3.446	11.167	19.681	30.193
Neptune	99996	200000	9.453	23.738	45.664	69.157

We list in Table 1 the binding time measurements of our unoptimized code. The computational overhead of blending weights can be reduced significantly by using a simplified mesh. The data size of weights is determined by the number of vertices and handles. Take the simplified *Neptune* model with 50K vertices and 100K triangles (LOD=50%) as an example, the data size of weights is about 0.19MB per handle without compression, and it takes an average of 0.561s to send that of 25 handles from the server to the client in our experiment.

Fig. 7 gives a comparison of deformation results between the one using the original weights and the other one using the interpolated weights. As is shown, only small distortions can be seen at a loss ratio even as high as 50% (review

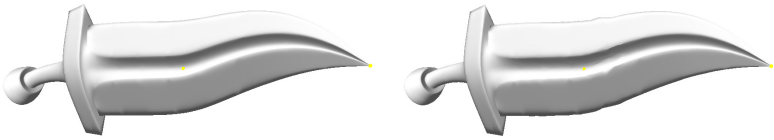


Fig. 7. Comparison between deformation results of using the original weights (left) and the interpolated weights (right). For parameters value of $\delta = 0.2$ and the loss ratio is 50%.

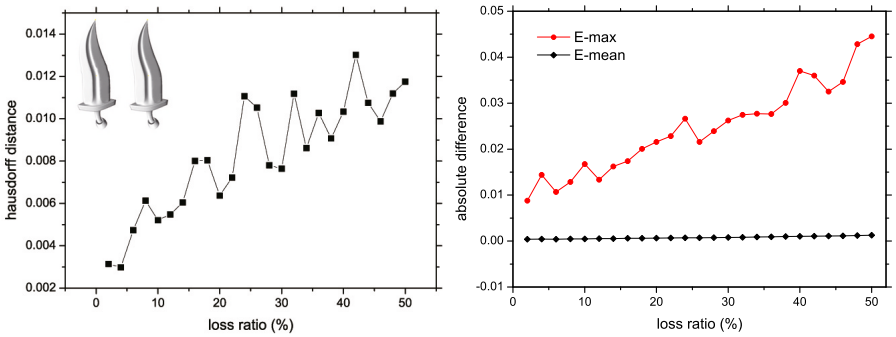
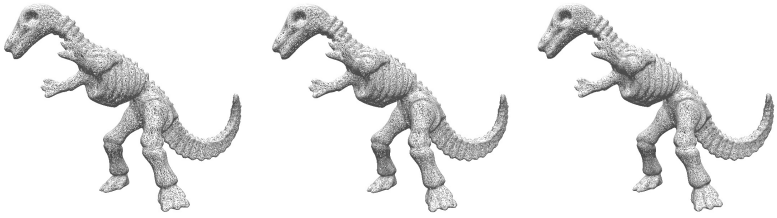
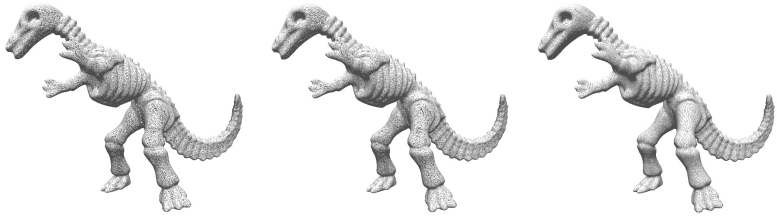


Fig. 8. The geometric distortion of the deformed Sword model and weight interpolation error with different data loss ratios. The geometric distortion is measured in Hausdorff Distance (left). E-max and E-mean are respectively the max and mean absolute difference between our interpolation and the original bounded biharmonic weights (right).



(a) LOD=30% (b) LOD=40% (c) LOD=50%



(d) LOD=60% (e) LOD=70% (f) LOD=100%

Fig. 9. Editing results of simplified mesh with different LOD (a-e), and the corresponding deformation result computed by the cloud server

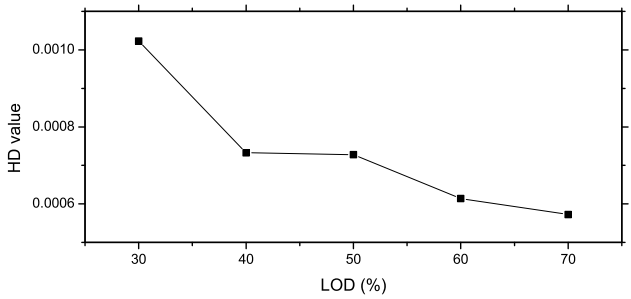


Fig. 10. The geometric difference between deformations of simplified and the original meshes with different LOD, measured in Hausdorff Distance

Fig. 6(a) for the plot of weights). Fig. 8 shows our analysis results on the weight interpolation and the geometric distortion according to different loss ratios. As expected, errors and distortions are very limited, even though with the increasing of data loss, the error and distortion raises as well.

Fig. 9 shows the editing results of simplified mesh with different LOD (a-e), and the corresponding deformation result computed by the cloud server. As we can see, deformation results are very similar. Furthermore, we show our analysis results on the geometric difference between the client (Fig. 9(a-e)) and the server (Fig. 9(f)) according to different LOD in Fig. 10. Consistent with Fig. 9, the geometric differences are very small, meaning the system deforms the original complex mesh in a way very close to user's desire.

6 Conclusion and Future Work

We have presented a system for cloud-based interactive mesh editing, namely GhostMesh. The system allows the user to edit a complex mesh that stored in the cloud server by interactively modifying a simplified mesh locally. We proposed efficient strategies to classify and stream blending weights and user manipulations. As a result, the system can transfer heavy computing and storage from the client to a cloud computing environment, allowing the user to edit a large complex model on the remote cloud server even using a lightweight terminal with low computing power and small storage space. Mobile-based augmented reality is an emerging technology that provides immersive experiences over wireless networks. In future work, we wish to extend our system to include multiple mobile users in a collaborative virtual environments. We also plan to compare with other applications and perform user studies in the future.

Acknowledgments. This work is supported by the National 863 Program of China under Grant No.2012AA011801, the Natural Science Foundation of China under Grant No.61170188, and the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant No.20121102130004.

References

1. Igarashi, T., Moscovich, T., Hughes, J.F.: As-rigid-as-possible shap manipulation. ACM Trans. Graph. 24(3), 1134–1141 (2005)
2. Botsch, M., Sorkine, O.: On linear variational surface deformation methods. IEEE TVCG 14(1), 213–230 (2008)
3. Ben-Chen, M., Weber, O., Gotsman, C.: Variational harmonic maps for space deformation. ACM Trans. Graph. 28(3) (2009)
4. Schaefer, S., Mcphail, T., Warren, J.: Image deformation using moving least squares. ACM Trans. Graph. 25(3), 533–540 (2006)
5. Kavan, L., Collins, S., Zara, J., OSullivan, C.: Geometric skinning with approximate dual quaternion blending. ACM Trans. Graph. 27(4) (2008)
6. Magnenat, N., Laperriére, R., Thalmann, D.: Joint-dependent local deformations for hand animation and object grasping. In: Graphics Interface, pp. 26–33 (1988)
7. Jacobson, A., Baran, I., Popović, J., Sorkine, O.: Bounded biharmonic weights for real-time deformation. ACM Trans. Graph., 78:1–78:8 (2011)
8. Forsey, D.R., Bartels, R.H.: Hierarchical B-spline refinement. ACM SIGGRAPH Computer Graphics 22(4), C205–C212 (1988)
9. Gortler, S.J., Cohen, M.F.: Hierarchical and variational geometric modeling with wavelets. In: Proceedings of ACM Symposium on Interactive 3D Graphics, pp. 43–54 (1995)
10. Kobbelt, L.P., Bareuther, T., Seidel, H.P.: Multiresolution shapedeformations for meshes with dynamic vertex connectivity. Computer Graphics Forum 19(3), 249–260 (2000)
11. Hoppe, H.: Progressive meshes. In: Proc. of SIGGRAPH 1996, pp. 99–108 (1996)
12. Garland, M., Heckbert, P.: Surface Simplification Using Quadric Error Metrics. In: Proceeding of SIGGRAPH 1997, pp. 209–216 (1997)
13. Gal, R., Sorkine, O., Mitra, N.J., Cohen-Or, D.: iWIRES: an analyze-and-edit approach to shape manipulation. ACM Trans. Graph. 28(3) (2009)
14. Schemali, L., Thiery, J.M., Boubekeur, T.: Automatic Line Handles for Freeform Deformation. Eurographics (Short), 81–84 (2012)
15. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: a sketching interface for 3D freeform design. In: Proceedings of SIGGRAPH 1999 (1999)
16. Nealen, A., Igarashi, T., Sorkine, O., Alexa, M.: FiberMesh: designing freeform surfaces with 3D curves. ACM Trans. Graph. 26(3) (2007)
17. Singh, K., Fiume, E.: Wires: a geometric deformation technique. In: Proceedings of SIGGRAPH 1998, pp. 405–414 (1998)
18. Zimmermann, J., Nealen, A., Alexa, M.: SilSketch: automated sketch-based editing of surface meshes. In: Proceedings of ACM Eurographics Workshop on Sketch-Based Interfaces and Modeling, pp. 23–30 (2007)