To appear in Proceedings of IEEE VR Conference, March 2001.

Collaborative Distributed Virtual Sculpting

Frederick W. B. Li Rynson W. H. Lau Frederick F. C. Ng

Department of Computer Science
City University of Hong Kong, Hong Kong
{borbor, rynson, fuk}@cs.cityu.edu.hk

Abstract

A lot of effort is now being put into developing collaborative distributed virtual environments. However, very few projects address collaborative virtual sculpting in which the shapes of the target objects are likely changing continuously. Some major issues including user interaction, data transmission, concurrent object editing by multiple clients and rendering of deforming objects must be addressed in a real-time context. In this paper, we propose a framework for collaborative virtual sculpting in a distributed virtual environment. The system is based on a hybrid model which merges the client-server and the peer-to-peer architectures to allow fast data replication. To support real-time deformation and rendering of deformable objects, we model each of these objects using NURBS surfaces and render them using the real-time deformable NURBS rendering method that we have developed. We will present a data structure for the transmission of these deformable objects. We will also introduce the idea of editing region and the corresponding locking mechanism for simultaneous editing of the same object by multiple clients. Toward the end of the paper, we will show some performance results of the prototype system.

1 Introduction

With the introduction of distributed virtual environments [1], we may now interact with each other in a virtual environment via local networks or through the internet, without physically travel. This encourages collaborative works from international participants, who may live in different geographical locations.

In [2, 3], we proposed a framework to support distributed virtual walkthrough over the Internet, in which progressive multiresolution modeling [4], caching mechanisms and prefetching mechanisms were used to optimize the system performance and to minimize the amount of date sent over the network. In this paper, we extend the work to support collaborative virtual sculpting. The proposed method provides a multi-user collaborative environment for participants at different geographical locations to modify the shapes of 3D objects. The objective of our work is to develop a distributed design environment, in which a geographically separated team can participate and visualize complex sculpting work together through the internet. Such a system will be extremely useful in reducing the cost of the product design process in manufacturing and in sculpting art works.

In the proposed framework, we model each deformable object with NURBS surfaces [5]. Although polygon meshes are widely used in object modeling, the vertex data are very often large in size and cause long transmission time. This reduces the interactivity of collaborative sculpting where model updates are needed to be sent frequently. NURBS surfaces, however, can be represented in a much more compact form, and they can be deformed simply by changing the positions of the control points. Our distributed virtual sculpting method proposed here makes use of two methods that we have developed. First, we apply our deformable NURBS rendering method [6, 7] to accelerate the rendering of deforming objects in the client machines. Our approach to real-time rendering of deformable objects, which are composed of NURBS surfaces, was to incrementally update a pre-generated polygon model of each deforming surface and progressively refine the resolution of the polygon model according to the change in surface curvature. Second, we apply our virtual sculpting method [8] to allow direct object modification using the $CyberGlove^{TM}$. In this sculpting method, a parametric hand surface is created by interpolating all the key data points of the $CyberGlove^{TM}$ and mapped to the object to be deformed. The object can then be deformed according to the change of the user's hand gesture in an intuitive way.

The rest of paper is organized as follows. Section 2 gives a brief survey on the related work. Section 3 reviews our deformable NURBS rendering method. Section 4 discusses the proposed framework for collaborative sculpting in a distributed environment. Section 5 presents a linear data struc-

ture to serialize the deformable surfaces for transmission. Section 6 introduces the idea of virtual sculpting, editing region and the corresponding locking mechanism. It also shows both the client-server and the client-client interactions. Section 7 presents some performance results of our prototype system and evaluates the new method. Finally, Section 8 briefly concludes the paper.

2 Related Works

Several frameworks and application systems for supporting distributed virtual environments have been proposed. They include **DIVE** [9], **SIMNET** [10] **NPSNET** [11], **MASSIVE** [12], **VLNET** [13] and **COVEN** [14, 15]. These systems mainly address issues on user interaction, data replication and optimization of data transmission to provide applications for visualization, simulation, training and entertainment.

Systems developed to support virtual sculpting are mainly for use in a single user environment. Galyean and Hughes developed a voxel based technique for virtual sculpting [16]. By using a 3D tracker, the user may edit a volumetric object by removing/clearing the voxels. The resultant voxel data is then converted to a polygon mesh using the marching-cube algorithm. Another system is THRED (Two Handed Refining Editor) [17, 18] developed by Shaw et al. to incorporate both hands, each tracked by a 3D tracker, in sculpting polygonal surfaces. The dominant hand selects and manipulates vertices, while the less dominant hand sets the position and orientation of the scene and the level of subdivision of the surface. Kameyama [19] proposed a Virtual Clay Modeling System. The system uses a special input device with a 3D tracker and a tactile sensor. The tactile sensor is made of arrays of pressure sensors and is covered by a soft rubber pad. By pushing at the tactile sensor, the user may deform an object using his hands. However, the resulting object is in the form of grid surface data and must be converted to solid model data for use in design or manufacturing systems. The 3DIVS [20] and two-handed direct manipulation interface [21] are design environments which allow a user, while wearing a pair of PINCH gloves, to use both hands to manipulate virtual objects. Users can perform a variety of actions by applying different PINCH gestures.

Research effort on developing distributed systems for collaborative virtual sculpting are very limited. In [22], Nishino et al. proposed a method for sharing interactive deformation in collaborative 3D modeling. In the method, the object for virtual sculpting is modeled by implicit surfaces. Each client has its own replica of the object. A client may edit an object only if it can obtain an update right of the object from a central server. While a client is editing the object, it broadcasts the update parameters to all the participating clients to update their copies of the object. However, due to the cost of tessellation, the object is not retessellated while it is deforming. After the client finishes the editing, it releases the update right by acknowledging the server. In [23], Anupam and Bajaj proposed a collaborative geometric and scientific design environment called **Shastra**. Each participant works on a shared hierarchical design graph of objects. This method enables direct collaboration by partitioning the design graph into zones. In regulated mode, a particular user is responsible for a zone and other users are denied to access that zone. In unregulated mode, a user can manipulate a "hot spot" in the design graph by gaining a prior exclusive control on a FIFO manner.

3 Deformable NURBS Rendering Method

The idea of our deformable NURBS rendering method is that we maintain two data structures of each deforming surface, the surface model itself and a polygon model representing the surface model [6, 24]. As the surface deforms, the polygon model is not regenerated through tessellation. Instead, it is incrementally updated to represent the deforming surface. Two techniques are fundamental to our method: *incremental polygon model updating* and *resolution refinement*.

3.1 Incremental Polygon Model Updating

Incremental polygon model updating is based on the incremental property of the polygonal representation of a NURBS surface obtained by evaluating the surface equation with some discrete parametric values. If a control point $P_{s,t}$ is moved to $\overline{P}_{s,t}$ with a displacement vector $\overrightarrow{V} = \overline{P}_{s,t} - P_{s,t}$, the incremental difference d between the two polygonal representations of the surface before and after the control point movement is as follows:

$$d = \overline{S} - S = \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{m,1} & \cdots & \alpha_{m,n} \end{bmatrix} \overrightarrow{V} \qquad (1)$$

where $1 \leq s \leq m$, $1 \leq t \leq n$, S and $\overline{S}$ are the polygon models of the surface before and after the control point movement, respectively. Each $\alpha_{s,t}$ is a *deformation coefficient*, a constant for each particular pair of (u, v) values. Hence, if the resolution of the polygon model representing the surface remains unchanged before and after the deformation, we may precompute the deformation coefficients and update the polygon model incrementally by the deformation coefficients and the displacement vector as shown in Equation (1). This technique is very efficient since we need to perform only one addition and one multiplication on each affected vertex of the polygon model to produce the updated

polygon model. In addition, its performance is independent of the complexity of the deforming surface.

In the implementation, the incremental polygon model updating is carried out in two stages, the *preprocessing* stage and the *run-time stage*. In the preprocessing stage, we tessellate the surface to obtain an initial polygon model. We also compute a set of deformation coefficients, $\alpha_{s,t}$, for each control point. As the surface deforms in run-time, the polygon model is incrementally updated with the set of deformation coefficients and the displacement vector of the moving control point.

3.2 Resolution Refinement

The resolution refinement technique refines the resolution of the polygon model and to generate new deformation coefficients incrementally according to the change in local curvature of the surface. Consider the fact that a NURBS surface can be subdivided into a polygon model by applying the de Casteljau subdivision formula [6, 24]. We may subtract the surface equation in the form of de Casteljau subdivision formula before the deformation, $C(u)$, and after the deformation, $\overline{C}(u)$, and substitute the resulting formula with the deformation coefficients α and the displacement vector $\overrightarrow{V}$ of the moving control point. We then have:

$$\overline{C}(u) - C(u) = C(\alpha) \tag{2}$$

Equation (2) shows the result by applying the de Casteljau subdivision formula to the Bernstein polynomials of the surface in the u parametric direction only. The other parametric direction is also in a similar form.

Hence, if the resolution of the polygon model needs to be increased, the new deformation coefficients can be calculated from adjacent deformation coefficients stored at existing vertices using the de Casteljau formula. If the resolution of the polygon model needs to be decreased, we may delete some quad-tree nodes from the polygon model.

3.3 Performance and Extensions

This efficient rendering method for deformable NURBS surfaces accelerates the rendering process by incrementally updating pre-computed polygon models without referring to the defining equation of the NURBS surfaces. Results showed that the method was roughly 3 to 15 times faster than existing methods. Recently, we extended the rendering method to cover different types of parametric surfaces [7] and developed two additional techniques, *incremental crack prevention* and *hierarchical surfaces*, to further improve the performance and flexibility of the method. The incremental crack prevention technique considers the fact that cracks would appear only in the regions with resolution change and hence, we only need to execute crack prevention on the nodes that we have performed the resolution refinement algorithm. This can avoid the execution of a complete crack prevention process from the root quadtree node recursively down to the leaf quadtree nodes in each frame time.

In hierarchical surfaces, we consider the fact that an object may be modeled by many surface patches but our original rendering method may perform resolution refinement only within each individual surface patch. The inter-patch resolution refinement was not supported. This limits the minimum resolution of the polygon model to the number of surface patches constructing the object. However, when the object is flat or too far away from the viewer, it may be more desirable to represent the object with even fewer polygons. The hierarchical surface technique addresses this problem by combining adjacent surface patches hierarchically to form a multi-resolution polygon model to represent the whole object. Hence, we can produce a polygon model with optimized resolution for each deforming object.

4 System Overview

There are two approaches to incorporate NURBS surfaces to represent deforming objects in a collaborative distributed environment. The first approach is to generate a polygon model for each deforming object at the server and send the polygon models to the clients for rendering. The advantage of this approach is that it offloads the time consuming tessellation process from the clients to the server, and thus speeds up the rendering performance of the clients. However, the transmission process may need to be performed in every frame to ensure that the clients will have the most updated polygon models for rendering. This will generate incredible workload to the network. In addition, it will also make it very difficult to support adaptive polygon resolution at the clients because the server will otherwise need to generate many polygon models according to the view parameters of individual clients.

The second approach is to replicate the surface parameters of the deforming objects at the clients and let the clients tessellate and render these objects. As an object deforms, the corresponding updated surface parameters are sent to the clients, and the clients may retessellate the polygon model of each deforming object to the appropriate resolution to optimize the rendering speed. With this approach, if traditional tessellation methods are used, it may be too slow to be interactive. The overall performance of the system will likely be degraded. Intuitively, we can replace the tessellation process with our deformable NURBS rendering method [6, 7] to allow interactive rendering of deformable objects at the clients. However, the relatively lengthy processes in creating the polygon model and the deformation coefficients cause extra setup time at each client before sculpting can begin.

In the new method, we distribute the tasks of the deformable NURBS rendering method in an optimal way to both the server and the client. The method extends our previous work on distributed virtual environments [2, 3]. However, instead of using a pure client-server model, the new system is based on a hybrid model which merges the client-server and the peer-to-peer architectures. Every participant can be a server or a client. An *object server* is the owner of a deformable object for virtual sculpting. It is responsible for creating an initial base model and the corresponding deformation coefficients of each deformable object that it owns. A client equipped with a $CyberGlove^{TM}$ may modify the shape of an object and broadcasts the updated control points to all the participants including the object server. The main components of our system is shown in Figure 1.

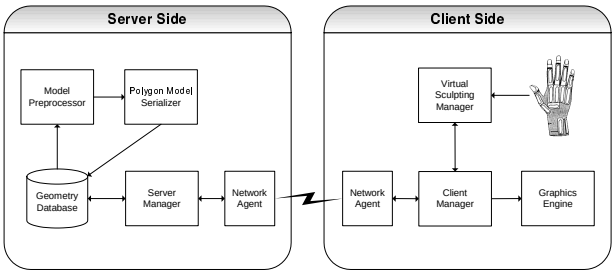

Figure 1. Main components of our system.

The client consists of 4 main components. Their functions are described as follows:

- *Client Manager*: It coordinates all components at the client and handles all user inputs. For each object to be deformed, the client manager first requests the base model and the set of deformation coefficients from the server to construct the initial polygon model and other relevant data structures. It then passes the information to the graphics engine to be maintained there.

- *Virtual Sculpting Manager*: It generates a parametric hand surface for the $CyberGlove^{TM}$ and performs virtual sculpting based on the user's hand gesture. The updated control points of the deforming object are then sent to the graphics engine via the client manager.

- *Network Agent*: It handles all the communications between the client and the server, including object requests and object updates. It also broadcasts updated control points to the other clients.

- *Graphics Engine*: It is responsible for maintaining and updating all the object models downloaded, including deformable models. It generates output images for display in every frame.

The server consists of 4 main components. Their functions are described as follows:

- *Server Manager*: It coordinates all components at the server and handles all clients' requests and updates.

- *Model Preprocessor*: It generates a base model and a set of deformation coefficients for each deformable object. The data are then sent to the polygon model serializer.

- *Polygon Model Serializer*: It constructs a linear data packet from the base model and the set of deformation coefficients. The data packet is then stored locally and sent to the clients upon their requests.

- *Network Agent*: It handles all the communications between the server and the clients, including object requests and object updates.

5 The Surface List

In our deformable NURBS rendering method, the polygon model is maintained in a quadtree format. This pointer based structure is inefficient for both transmission and storage. To overcomes these limitations, we have adopted the linear quadtree structure proposed by Gargantini [25] so that efficient transmission and caching mechanisms can be implemented in the system. To traverse the quadtree nodes, we make use of the indexing scheme proposed by Balmelli et al. [26]. We also propose a new serialized structure, called the surface list, to optimize the size of the data stream.

A linear quadtree is a pointerless scheme to store a generic quadtree in the form of a one-dimensional array of nodes. Given a particular node $p > 0$, the node index in the array can be obtained by the following equations:

$$\text{Parent node index:} \quad \left\lfloor \frac{p-1}{4} \right\rfloor \tag{3}$$

$$\text{Child node indices:} \quad 4p + i,\ i = 1 \ldots 4 \tag{4}$$

In [26], Balmelli et al. suggested to store a balanced quadtree in a linear array of nodes. It is simple, but in practice, most quadtrees are likely to be unbalanced. Thus, extra spaces are consumed to hold the empty nodes. To allow efficient transmission and caching of polygon models, we propose a revised structure to optimize the size of the data stream that holds the quadtree.

The structure of a surface list is shown in Figure 2. It consists of three main sections, the *node presence list*, *packed vertices*, and *packed deformation coefficients with control point ID's*. The node presence list is an array of boolean values encoding the presence of nodes in the linear quadtree using Balmelli's index scheme. The packed

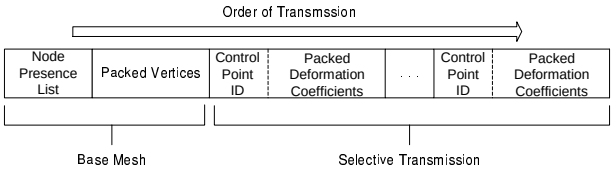

Figure 2. The data structure of the surface list.

vertices are the list of (x, y, z) coordinates of the present quadtree nodes. The nodes are arranged in ascending order with respect to their node indices. The first and second sections are referred to as the *base mesh*. The last section is a repeated sequence of records of packed deformation coefficients with its associated control point ID. This section is similar to the section of packed vertices, except that it is a linear array of scalar values of deformation coefficients. The control point ID indicates which control point that the set of the deformation coefficients belongs to.

To transmit a surface list to the client, the server may send it either as a whole or progressively. For progressive transmission, the server transmits only a subset of elements from the node presence list, packed vertices and packed deformation coefficients to the client at each time. The selection of elements are in ascending order of the elements of each section of the surface list. The client may then progressively refine the resolution of the object model upon receiving these progressive records.

When many clients request for objects at the same time, instead of sending the entire set of deformation coefficients with the base model in the initial stage, they can be sent selectively at run-time upon the clients' requests. This can reduce the network traffics and therefore minimize the latency of object requests.

6 Distributed Virtual Sculpting

We have recently developed an intuitive method for virtual sculpting [8], which is for object modification using the $CyberGlove^{TM}$. The idea is to create a *hand surface* using the bicubic B-spline tensor product interpolating all key data points of the $CyberGlove^{TM}$. These data points indicate the finger joint positions of the user's hand. The object to be deformed is then mapped to the hand surface by *ray-projection*. In ray-projection, the user specifies a center of projection, P_c. A set of rays are projected from P_c through individual object vertices onto the hand surface to establish the mapping. By changing the user's hand gesture, the object model can be deformed accordingly. We can also adjust the location of P_c to have a direct, reduced or enlarged modification region of the object model as shown in Figure 3(a), 3(b) and 3(c), respectively.

6.1 Defining and Locking of Editing Region

In a collaborative distributed environment, multiple clients may sometimes want to edit the same object simultaneously. If we lock the entire object once a client requests to edit it, other clients would not be able to participate in the sculpting. This leads to a bottleneck in collaborative sculpting. To enhance the collaboration and the availability of objects for sculpting, we introduce the idea of *editing region*. The editing region is a sub-region of an object being edited. When a client manipulates an editing region of a deformable object, the other regions of the object will not be affected by this manipulation, and can be edited by other clients simultaneously. A client may modify one or more editing region(s) provided that the regions are not currently edited by other clients.

To define the editing region, we consider the local modification property [5] of NURBS surfaces. If the position of a control point $P_{i,j}$ is changed, only the shape of the surface within the parameter region $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$ is affected, where p and q are the degrees of the NURBS surfaces along u and v parameter directions, respectively. We refer to this region as the *deformation region*. An *editing region* is the union of deformation regions of the control points that fall inside the hand surface. Figure 4 shows the virtual sculpting of a human head model with the $CyberGlove^{TM}$ in our prototype system. The 6x6 polygon mesh associated with the virtual hand represents the hand surface, while the grey region on the human head model indicates the editing region of the virtual hand.

To implement the locking mechanism for editing, we maintain at each client an *editing list* containing a set of boolean flags, each corresponding to a control point of the object for sculpting. A bit is set to TRUE if the corresponding control point is currently manipulated by one of the clients in the collaborative environment. When a client wants to modify the shape of the object, it first determines the editing region that it is interested in and compares the editing list with the set of control points of the editing region. If all control points in the editing list are set to FALSE (i.e., they are all available), the client will be granted the right to edit that region and a *locking message* will be broadcasted to all clients to update the corresponding bits of their editing lists. To resolve concurrent requests for the same editing region from multiple clients, a time stamp is attached to the locking message. Only the locking message with the earliest time stamp can acquire the requested editing region. When the client finishes the editing, it broadcasts a lock *release message* to all clients to clear those bits of the editing lists. On the other hand, if a client wants to modify the shape of an object and finds that part of or the whole editing region is locked, the client may need to wait until the region is released. We adopt such a distributed

locking scheme instead of a central server-based scheme to avoid the bottleneck caused by the large number of editing requests.

6.2 Client-Server Interactions

The distributed virtual sculpting process consists of two stages, the pre-processing stage and run-time stage. The client-server interactions occur in the pre-processing stage while the client-client interactions occur in the run-time stage. Figure 5 shows the main processes involved in the client-server interactions. In the pre-processing stage, the server generates an initial polygon model, a hierarchical surface, a set of deformation coefficients, and a surface list of each deformable object in the distributed environment. Upon a client's request, the server sends the surface definition, and the surface list consisting of the *base model* and the set of deformation coefficients to the client. After the client has received the base model, it refines the base model for rendering according to its own view parameters.

6.3 Client-Client Interactions

During run-time, if a client is editing an object, it is responsible for broadcasting *update messages* to all the participating clients. Each update message contains the ID's of the set of control points of the editing region and the corresponding new (x, y, z) coordinates of those control points. After a participating client has received an update message, it performs the incremental polygon model updating and resolution refinement according to its current view parameters and renders the resulting polygon model for display. Figure 6 shows the main processes and the broadcast messages involved in the client-client interactions.

7 Results and Discussions

We have implemented the new method with C++, OpenGL and Open Inventor. The server and the client modules communicate using TCP/IP with the BSD Sockets Library. The virtual sculpting manager was implemented with the VirtualHand Library and GesturePlus Library from Virtual Technology. Figure 7 shows a human head model that we have used for sculpting. The human head model is a NURBS model with 400 control points. We apply knot insertion [5] to subdivide it into 400 Bézier surface patches and construct a hierarchical surface on these patches to form the base model for transmission. We run the test on a SGI Onyx^2 and a SGI Octane machine, each with one CPU activated. They are physically connected on the same network segment of 10Mbit Ethernet LAN with an ATM backbone

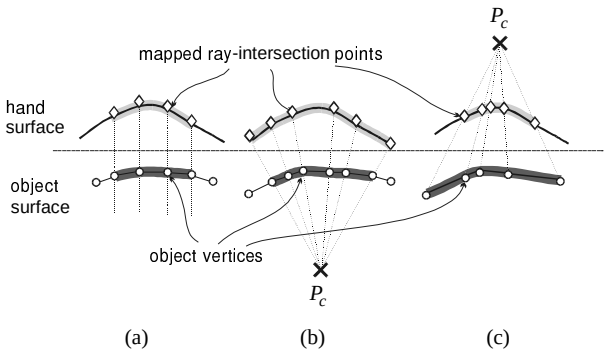

Figure 3. Ray Projection with different centers of projection. (a) Direct modification region, (b) Reduced modification region, and (c) Enlarged modification region.

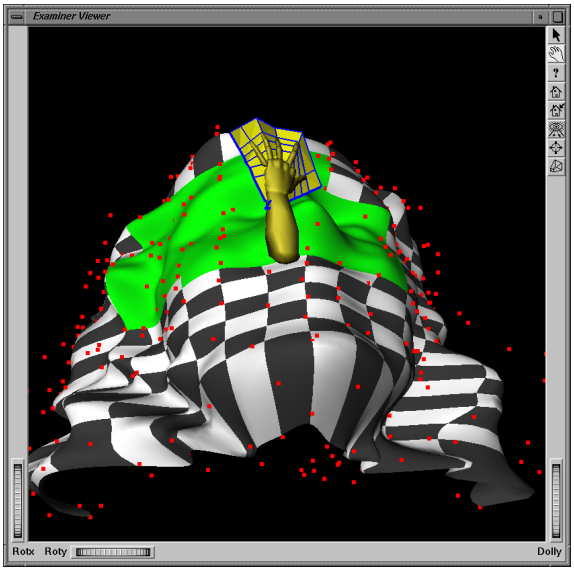

Figure 4. Virtual sculpting of a human head model and the current editing region.

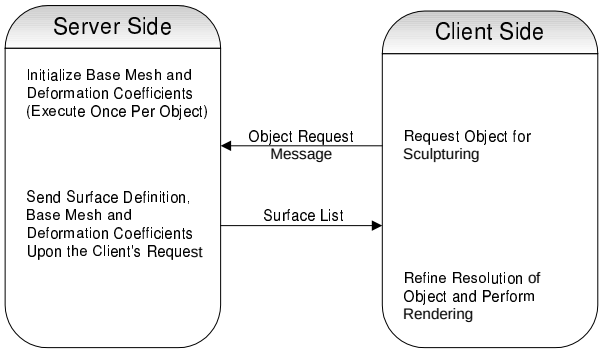

Figure 5. Client-Server Interactions.

and are separated by two switches. The most time consuming data transmission task in our method is sending the surface list. Table 1 shows the record sizes and the transmission times of individual parts of the surface list. We sent the data with their native data types without compression. The total packet size of the data stream is 74.75 Kbytes and is transmitted in 0.0940 seconds. At the time of the test, the bandwidth available to us was about 4Mbits. If we assume that the Internet bandwidth is roughly 0.2Mbits, the time needed to transmit the complete surface list will be 4.7 seconds. On the other hand, instead of sending the set of deformation coefficients together with the base model in the initial stage, they can be sent selectively and separately at run-time upon the clients' requests. This can reduce network traffics and system latency when many clients make object requests at the same time.

In our method, we use a distributed editing list instead of a central server to control simultaneous editing. This can reduce both the workload of the server and the network latency when the clients raise their requests continuously. Whenever a client raises a request, it only needs to lookup its local copy of the editing list, and the locking message is sent only if the requested editing region is available. Hence, unlike Nishino's method [22], our method generates network traffics for successful requests only. For the tessellation process, both Nishino's method and our method delegate the tessellation task to the client. This can save sending polygon models from the server to the clients. In addition, each client may have a different polygon resolution for the deforming object according to its own view parameters, to enhance the rendering performance. We further accelerate the tessellation process by applying incremental polygon model updating and progressive resolution refinement. This can avoid the re-execution of the time consuming tessellation process in each frame time.

8 Conclusion

In this paper, we have presented a framework for collaborative virtual sculpting in a distributed virtual environment. We have introduced a serialized quadtree structure, called the surface list, for transmitting deforming objects through the network for virtual sculpting. We have also presented the idea of editing region and the corresponding locking mechanism to allow simultaneous virtual sculpting by multiple users. Results show that our method can provide a reasonably fast response for clients to perform virtual sculpting even over the Internet.

Acknowledgments

The work described in this paper was partially supported by a CityU grant (Project Number 7001071).

References

[1] M. Macedonia and M. Zyda, "A Taxonomy for Networked Virtual Environments," *IEEE Multimedia*, vol. 4, no. 1, pp. 48–56, 1997.

[2] J. Chim, M. Green, R. Lau, H. Leong, and A. Si, "On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments," *ACM Multimedia*, pp. 171–180, September 1998.

[3] J. Chim, R. Lau, A. Si, H. Leong, D. To, M. Green, and M. Lam, "Multi-Resolution Model Transmission in Distributed Virtual Environments," *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pp. 25–34, November 1998.

[4] R. Lau, M. Green, D. To, and J. Wong, "Real-Time Continuous Multi-Resolution Method for Models of Arbitrary Topology," *PRESENCE: Teleoperators and Virtual Environment*, vol. 7, no. 2, pp. 22–35, 1998.

[5] L. Piegl and W. Tiller, *The NURBS Book*. Springer-Verlag, 1995.

[6] F. Li, R. Lau, and M. Green, "Interactive Rendering of Deforming NURBS Surfaces," *Proceedings of Eurographics '97*, vol. 16, pp. 47–56, September 1997.

[7] F. Li and R. Lau, "Real-Time Rendering of Deformable Parametric Free-Form Surfaces," *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pp. 131–138, December 1999.

[8] J. Wong, R. Lau, and L. Ma, "Virtual 3D Sculpting," *Journal of Visualization and Computer Animation*, vol. 11, no. 3, pp. 155–166, 2000.

[9] C. Carlsson and O. Hagsand, "DIVE - a Multi-User Virtual Reality System," *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp. 394–400, 1993.

[10] J. Calvin, A. Dicken, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The SIMNET Virtual World Architecture," *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp. 450–455, 1993.

[11] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz, "NPSNET: A Network Software Architecture for Large Scale Virtual Environments," *Presence: Teleoperators and Virtual Environments*, vol. 3, no. 4, pp. 265–287, 1994.

[12] S. Benford, J. Bowers, L. Fahlen, and C. Greenhalgh, "Managing Mutual Awareness in Collaborative Virtual Environments," *Proceedings of Virtual Reality Software and Technology*, pp. 223–236, 1994.

[13] I. Pandzic, T. Capin, E. Lee, N. Thalmann, and D. Thalmann, "A Flexible Architecture for Virtual Humans in Networked Collaborative Virtual Environments," *Proceedings of Eurographics '97*, vol. 16, pp. 177–188, September 1997.

[14] V. Normand and J. Tromp, "Collaborative Virtual Environments: the COVEN Project," *Proceedings of the Framework for Immersive Virtual Environments Conference (FIVE'96)*, 1996.

Part Name	Data Type	Size	Transmission Time
Node Presence List	short	1.33 Kbytes	0.0028 sec
Packed vertices	float	19.17 Kbytes	0.0225 sec
Packed Deformation Coefficients	float	54.25 Kbytes	0.0687 sec
Total		74.75 kBytes	0.0940 sec

Table 1. Data sizes and transmission times of different parts of the surface list.

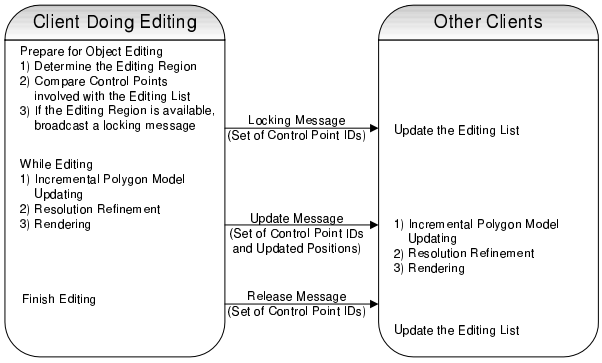

Figure 6. Client-Client Interactions.

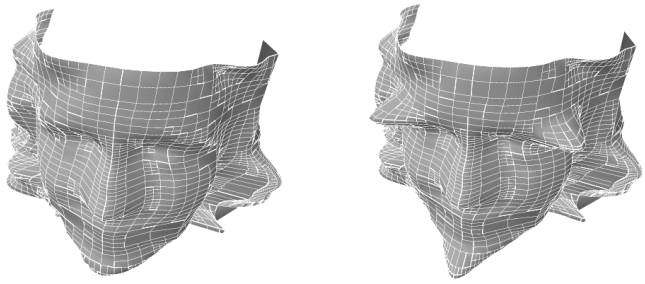

Figure 7. Model in our experiment.

[15] V. N. et al., "The COVEN Project: Exploring Applicative, Techical and Usage Dimensions of Collaborative Virtual Environments," *Presence: Teleoperators and Virtual Environments*, vol. 8, no. 2, pp. 218–236, 1999.

[16] T. Galyean and J. Hughes, "Sculpting: An Interactive Volumetric Modeling Technique," *Proceedings of ACM SIGGRAPH '91*, vol. 25, no. 4, pp. 267–274, 1991.

[17] C. Shaw and M. Green, "Two-Handed Polygonal Surface Design," *Proceedings of UIST 1994*, pp. 205–212, November 1994.

[18] C. D. Shaw and M. Green, "THRED: A Two-Handed Design System," *Multimedia Systems Journal*, vol. 5, pp. 126–139, March 1997.

[19] K. Kameyama, "Virtual Clay Modeling System," *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pp. 197–200, September 1997.

[20] F. Kuester, M. Duchaineau, B. Hamann, K. Joy, and A. Uva, "3DIVS: 3-Dimensional Immersive Virtual Sculpting," *Proceedings of the Workshop on New Paradigms in Information Visualization and Manipulation in Conjunction with the Eighth ACM Interaction Conference on Information and Knowledge Management*, pp. 92–96, November 1999.

[21] L. Cutler, B. Fröhlich, and P. Hanrahan, "Two-Handed Direct Manipulation on the Responsive Workbench," *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pp. 107–114, April 1997.

[22] H. Nishino, K. Utsumiya, A. Sakamoto, K. Yoshida, and K. Korida, "A Method for Sharing Interactive Deformations in Collaborative 3D Modeling," *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pp. 116–123, December 1999.

[23] V. Anupam and C. Bajaj, "Shastra: Multimedia Collaborative Design Environment," *IEEE Multimedia*, vol. 1, no. 2, pp. 39–49, 1994.

[24] F. Li and R. Lau, "Incremental Polygonization of Deforming NURBS Surfaces," *Journal of Graphics Tools*, vol. 4, no. 4, pp. 37–50, 1999.

[25] I. Gargantini, "An Effective Way to Represent Quadtrees," *Communications of the ACM*, vol. 25, pp. 905–910, December 1982.

[26] L. Balmelli, J. Kovačević, and M. Vetterli, "Quadtrees for Embedded Surface Visualization: Constraints and Efficient Data Structures," *Proceedings of IEEE International Conference on Image Processing (ICIP)*, vol. 2, pp. 487–491, October 1999.