VSculpt: A Distributed Virtual Sculpting Environment for Collaborative Design

Frederick W. B. Li, Rynson W. H. Lau, *Member, IEEE*, and Frederick F. C. Ng

***Abstract*—A collaborative virtual sculpting system supports a team of geographically separated designers/engineers connected by networks to participate in designing three-dimensional (3-D) virtual engineering tools or sculptures. It encourages international collaboration at a minimal cost. However, in order for the system to be useful, two factors need to be addressed: intuitiveness and real-time interaction. Although a lot of effort has been put into developing virtual sculpting environments, only limited work addresses collaborative virtual sculpting. This is because in order to support real-time collaborative virtual sculpting, many challenging issues need to be addressed. In this paper, we propose a collaborative virtual sculpting framework, called VSculpt. Through adapting some techniques we developed earlier and integrating them with some techniques developed here, the proposed framework provides a real-time intuitive environment for collaborative design. In particular, it addresses issues on efficient rendering and transmission of deformable objects, intuitive object deformation using the *CyberGlove* and concurrent object deformation by multiple clients. We demonstrate and evaluate the performance of the proposed framework through a number of experiments.**

***Index Terms*—Collaborative environments, deformable object rendering, distributed collaboration, virtual sculpting.**

I. Introduction

With THE introduction of distributed virtual environments, we may now interact and work with each other via a local network or through the internet, without physically travel. This encourages collaborative work from international participants living at different geographical locations. In [1] and [2], we proposed a framework to support distributed virtual walkthrough over the Internet, in which progressive multiresolution modeling, caching and prefetching mechanisms were used to minimize the amount of data sent over the network. However, the system does not support collaboration nor interaction among the participants.

In this paper, we present a framework for distributed virtual sculpting, called VSculpt. The objective of this work is to develop a distributed design environment in which a geographically separated team can manipulate and visualize complex sculpting work together through the internet. The proposed framework will reduce the cost and turnaround time of the product design process in manufacturing or in sculpting artwork. However, in order to develop such a collaborative environment, many challenging issues need to be addressed, including real-time processing, rendering, and efficient transmission of deformable objects. An intuitive sculpting method is also needed to allow the user to deform an object with hands. In addition, synchronization techniques and control mechanisms are needed so that multiple clients may perform object sculpting simultaneously. As will be discussed in Section II, research work on collaborative virtual sculpting is very limited, due to the many unsolved issues. In fact, we are not aware of any systems that support interactive collaborative sculpting. We develop VSculpt to address the above issues. The main contributions of this paper include

- a framework to support interactive collaborative sculpting in a distributed environment, by adapting some of the techniques we developed earlier and integrating them with techniques proposed here;
- a data structure for progressive transmission of deformable objects.
- a technique to support concurrent editing of a virtual object by multiple clients;
- a communication protocol to support synchronized transmission and object deformation.

The rest of this paper is organized as follows. Section II gives a survey on related work. Section III gives an overview of VSculpt and its architecture. Section IV summarizes our deformable NURBS rendering method. Section V shows how the data structures of a deformable object may be organized for progressive transmission. Section VI introduces the idea of editing region and proposes a locking mechanism to support collaborative sculpting. It also shows the client–server and the client–client interactions. Section VII presents some performance results of our prototype system and evaluates the new method. Finally, Section VIII briefly concludes the paper.

II. Related Work

Several frameworks and application systems have been proposed to support distributed virtual environments. They include **DIVE** [3], **SIMNET** [4], **NPSNET** [5], **MASSIVE** [6], **VLNET** [7], and **COVEN** [8]. These systems mainly address issues on user interaction, data replication and optimization of data transmission in order to support various applications including 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 [9]. Using a three-dimensional (3-D) tracker, a user may edit a

Manuscript received August 4, 2001; revised July 17, 2002. This work was supported in part by a DAG grant from City University of Hong Kong under Project 7100153 and by a CERG grant from the Research Grants Council of Hong Kong under Project CityU 1080/00E). The associate editor coordinating the review of this paper and approving it for publication was Prof. Ryoichi Komiya.

The authors are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (e-mail: rynson@cs.cityu.edu.hk).

Digital Object Identifier 10.1109/TMM.2003.814795

volumetric object by removing/clearing some 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) [10] developed by Shaw *et al.* to incorporate both hands, each tracked by a 3-D tracker, in editing polygonal surfaces. While the dominant hand selects and manipulates vertices, the less dominant hand sets the position and orientation of the scene and the level of subdivision of the surface. Kameyama [11] proposed a Virtual Clay Modeling System. The system uses a special input device with a 3-D 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/her hands. Because the resulting object is in the form of grid surface data, it must be converted to a solid model before it can be used in a design or manufacturing system. The **3DIVS** [12] and the two-handed direct manipulation interface [13] are design environments that allow a user, when 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.

Effort to develop distributed systems for collaborative virtual sculpting is very limited. In [14], Nishino *et al.* proposed a method for sharing interactive deformation in collaborative 3-D 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 can edit the object only if it can obtain an update right of the object from a central server. While a client is sculpting 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 as it is deforming. After the client finishes the sculpting, it releases the update right by acknowledging the server. In [15], 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, when a particular user is responsible for a zone, 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. In **COVEN** [8], the concept of "interaction agent" is introduced, which is shared by several participants to manage a collaborative interaction situation. However, it is only a conceptual idea and no concrete solution is available.

There are several limitations in existing distributed sculpting systems. First, they use a central server to control and grant the editing right to the clients. This central server may become a bottleneck and degrade both the performance and the interactivity of the whole sculpting environment. Second, they do not support concurrent sculpting of the same object. However, in some design applications, it is desirable for multiple users to edit the same object together. Finally, these systems totally rely on the client machines to perform the rendering task. Although it can save both the workload of the server and the amount of data transmitted through the network, the expensive retessellation process may seriously affect the performance of the system. Alternatively, some systems simply do not perform the retessellation process, sacrificing the quality of the output images.

VSculpt addresses these three problems through the introduction of a distributed object locking mechanism, the editing region, and a distributed rendering and transmission technique for deformable objects. We will describe these in details later in the paper.

III. Architectural Overview of VSculpt

A. Overview of VSculpt

In VSculpt, each object is modeled using NURBS surfaces. Although polygon meshes are widely used in object modeling, the vertex data is very often large in size and therefore time consuming for transmission. This reduces the interactivity of collaborative sculpting where model updates are sent over the network 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. However, as a NURBS surface deforms, we need to retessellate it into polygons for rendering. Because retessellation is a very expensive task, we adapt our real-time NURBS rendering method [16], [17] here to accelerate the rendering of deforming objects in the client machines. Initially, we tessellate the NURBS surfaces of each object into a polygon model and compute all the deformation coefficients. These data structures are then packed into a linear data structure, called *NURBS stream*, to facilitate efficient rendering and progressive transmission. Details of this can be found in Sections IV and V.

When some users want to initiate collaborative sculpting, they first identify the object for sculpting and the corresponding object server will distribute the object to all the relevant client machines in the form of a NURBS stream. In order for the user to be able to sculpt the object in an intuitive manner, we adapt our virtual sculpting technique here [18] to allow direct object modification with the user's own hands. Each user participate in the collaborative sculpting will wear one or a pair of *CyberGloves*. Each *CyberGlove*[1] is basically an electronic glove that captures the user's hand and finger gesture. The system will map the *CyberGlove* to the object for sculpting, so that the user may deform the object by flexing the hand(s). In order to provide a more flexible environment for sculpting, a *ray-projection* technique is used to allow the user to dynamically change the mapping between the *CyberGlove* and the object surface. The *CyberGlove* can be mapped to the whole object to allow coarse deformation or to only a small region of the object to allow fine deformation. Details of this can be found in Section VI.

When a user selects a region of the object for deformation, the control points that affect the shape of this region are determined. As the user flexes the hand(s) to deform the region, the new positions of the control points will be distributed to all participating clients in the form of update messages. When a client receives an update message, it can update the data structures to reflect the change in object shape. In the case when a NURBS surface

[1]*CyberGlove* is a trademark of Immersion Corporation, San Jose, CA 95131 USA.

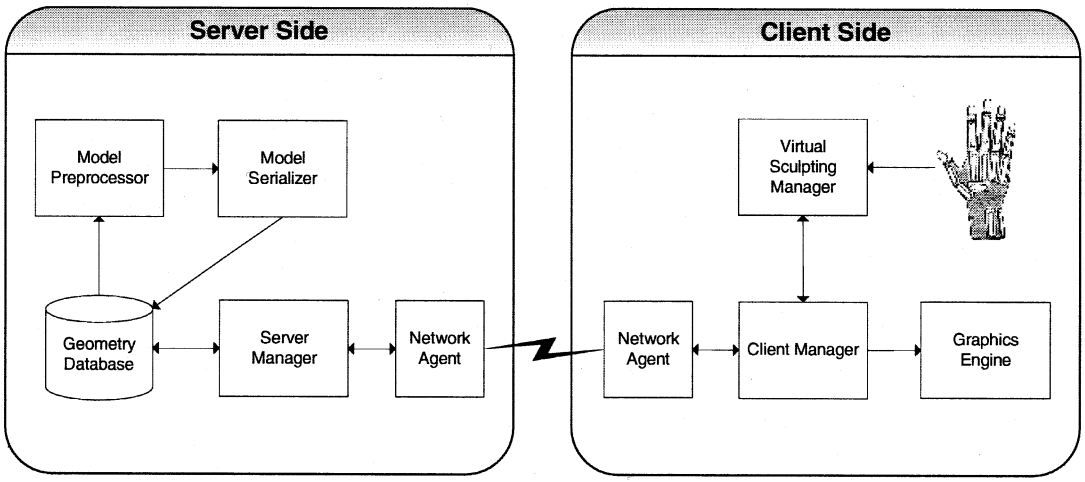


Fig. 1. Main components of VSculpt.

needs to be refined and the refinement information is not available locally, the client may either compute it locally or request the server for it. Details of the client–server and client–client communications can be found in Sections VI-B and VI-C.

B. Architecture of VSculpt

VSculpt 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 deformable objects for sculpting. It is responsible for constructing the NURBS stream of each object. A client equipped with a *CyberGlove* may modify the shape of an object and broadcasts the updated control points to all the clients including the object server. Fig. 1 shows the main components of VSculpt.

The server module consists of four main processes. The *Server Manager* coordinates all other components at the server and handles all clients' requests and updates. The *Model Preprocessor* computes a polygon model and a set of deformation coefficients for each deformable object. The data are then sent to the *Model Serializer*, which constructs a NURBS stream from the polygon model and the set of deformation coefficients. The NURBS stream is then stored in the database for later transmission to the clients upon their requests. Finally, the *Network Agent* handles all the communications between the server and the clients, including object requests and object updates.

The client also consists of four main processes. The *Client Manager* coordinates all components at the client and handles all user inputs. It is responsible for requesting the NURBS streams from the server to construct the relevant data structures and then passes the information to the graphics engine to be maintained there. The *Virtual Sculpting Manager* generates a parametric hand surface for the *CyberGlove* 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. The *Network Agent* handles all the communications between the client and the server, including object requests and object updates. It also sends update messages to other clients. Finally, the *Graphics Engine* is responsible for maintaining and updating all the object models downloaded, including deformable models. It generates output images for display in every frame.

IV. Rendering of Deforming Objects

In our earlier work, we developed a technique for efficient rendering of deformable NURBS surfaces [16], [17]. The basic idea of this method is to maintain two data structures of each surface, the surface model and a polygon model representing the surface model. As the surface deforms, the polygon model is not regenerated through tessellation. Instead, it is incrementally updated to represent the deforming surface. There are two techniques fundamental to our method: *incremental polygon model updating* and *resolution refinement*.

A. Incremental Polygon Model Updating

In this technique, we incrementally update a precomputed polygon model to represent each deforming surface. To show how it works, we consider the polygonal representation of a 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 between the two polygonal representations of the surface before and after the control point movement is as follows:

$$\overline{S}(u, v) - S(u, v) = \frac{\left(\overline{P}_{s,t} - P_{s,t}\right) w_{s,t} R_{s,t}(u,v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} R_{i,j}(u,v)} = \alpha_{s,t} \overrightarrow{V} \tag{1}$$

where $S(u, v)$ and $\overline{S}(u, v)$ are the polygon models of the surface before and after the control point movement, respectively. $\alpha_{s,t}$ is called the *deformation coefficient* defined as follows:

$$\alpha_{s,t}(u, v) = \frac{w_{s,t} R_{s,t}(u,v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} R_{i,j}(u,v)}. \tag{2}$$

The deformation coefficient $\alpha_{s,t}$ is a constant for each particular pair of (u, v). 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 as shown in (1). This technique is very efficient since we need to perform only one vector addition and one scalar-vector multiplication on each affected vertex of the polygon model. Another advantage is that the performance of the method is independent of the surface complexity.

B. Resolution Refinement

When a surface deforms, its curvature is also changed. If the curvature is increased or decreased by a large amount during the deformation, the resolution of the polygon model may become too coarse or higher than necessary to represent the deforming surface, respectively. To overcome this problem, we proposed a *resolution refinement* technique to refine the resolution of the polygon model and to compute new deformation coefficients incrementally according to the change in the surface curvature.

A NURBS surface is first converted into a set of Bézier patches using knot insertion [19]. Each Bézier patch is then subdivided into a polygon model by applying the de Casteljau subdivision formula [20] to the Bernstein polynomials in both u and v directions. For example, in u, we have

$$P_i^r(u) = (1-u)\,\frac{w_i^{r-1}}{w_i^r}\,P_i^{r-1}(u) + u\,\frac{w_{i+1}^{r-1}}{w_i^r}\,P_{i+1}^{r-1}(u) \quad (3)$$

where $w_i^r(u) = (1-u)w_i^{r-1}(u) + uw_{i+1}^{r-1}(u)$ and $r = 1, \ldots, n$, $i = 0, \ldots, n-r$. $[w_iP_i \;\; w_i]^T$ are the homogeneous Bézier points with $P_i \in \mathbf{R}^3$, w_i are the weights, and n is the degree of the surface. The v direction has similar recursion.

If we compute the difference of (3) before and after the deformation and then simplify it, we get a de Casteljau formula as follows:

$$\alpha_i^r(u) = (1-u)\alpha_i^{r-1}(u) + u\alpha_{i+1}^{r-1}(u) \quad (4)$$

for $r = 1, \ldots, n$, $i = 0, \ldots, n-r$. Equation (4) indicates that the deformation coefficients can be generated incrementally by the de Casteljau subdivision formula.

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. To achieve a better performance, we implemented this based on the Horner's formula, of average complexity $O(n)$ as opposed to $O(n^2)$ when based on the de Casteljau's formula.

V. Deformable Object Transmission

In our deformable NURBS rendering method, we maintain a polygon model of each deformable object in a quadtree structure. The algorithms for this pointer-based tree structure are recursive in nature. In addition, the navigation methods associated with these algorithms are often restricted to preorder, inorder or postorder tree traversal. If an operation, such as the crack prevention process in our rendering method, requires information from neighboring nodes for the comparison of subdivision levels, an additional tree traversal operation is needed to locate the neighboring node starting from the root node, unless extra pointers are provided to link each node to its neighboring nodes. To overcome this limitation, we propose a new linear data structure, called *NURBS streams*, which are used to maintain and transmit the precomputed polygon models of the NURBS surfaces.

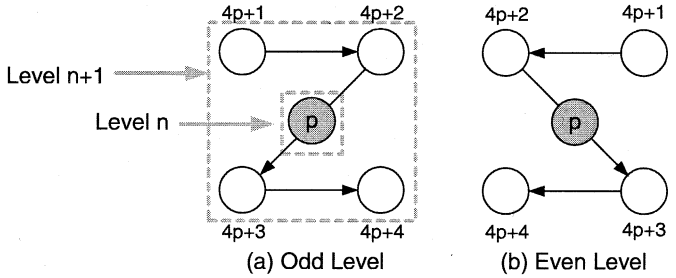


Fig. 2. Z-ordering indexing scheme.

A NURBS stream is based on the linear quadtree structure proposed by [21] using the z-ordering indexing scheme [22]. It allows constant navigation time between any node pairs and supports progressive transmission. Fig. 2 illustrates the z-ordering indexing scheme for a linear quadtree. It shows a top view of the two possible spatial organizations of a parent node p with its child nodes $4p+1$, $4p+2$, $4p+3$ and $4p+4$. A quadtree is assumed to start from level 1, i.e., the root node. Each quadtree node is assigned with an unique index. When assigning an index to a node, if the child node is residing at an odd level, we apply the spatial organization shown in Fig. 2(a); otherwise, we apply the one shown in Fig. 2(b).

A. Indexing Scheme

A linear quadtree is a pointerless scheme to store a generic quadtree in the form of a linear array of nodes. The quadtree nodes are ordered by both the z-ordering indexing scheme and their residing quadtree levels. Each potential node, whether it exists or not, is assigned with a static and unique index. For example, the index of the root node is "0" and a node residing at a deeper quadtree level has a greater index value. Given a node of index p, we can determine the following:

$$\text{Parent node index:} \quad \left\lfloor \frac{p-1}{4} \right\rfloor \quad (5)$$

$$\text{Child node indices:} \quad 4p+i, \qquad i = 1 \cdots 4 \quad (6)$$

$$\text{Node level:} \quad \lfloor \log_4 (3p+1) \rfloor \quad (7)$$

$$\text{Western neighbor:} \quad p - \Delta h^d(n_x) \quad (8)$$

$$\text{Eastern neighbor:} \quad p + \Delta h^d(n_x+1) \quad (9)$$

$$\text{Northern neighbor:} \quad p - \Delta v^d(n_y) \quad (10)$$

$$\text{Southern neighbor:} \quad p + \Delta v^d(n_y+1) \quad (11)$$

where (n_x, n_y) is the 2D location of the node in the grid of nodes at a particular level. Δh^d and Δv^d are the precomputed lists, called distance vectors, that give the horizontal and vertical index differences for pairs of neighboring nodes at level d, respectively.

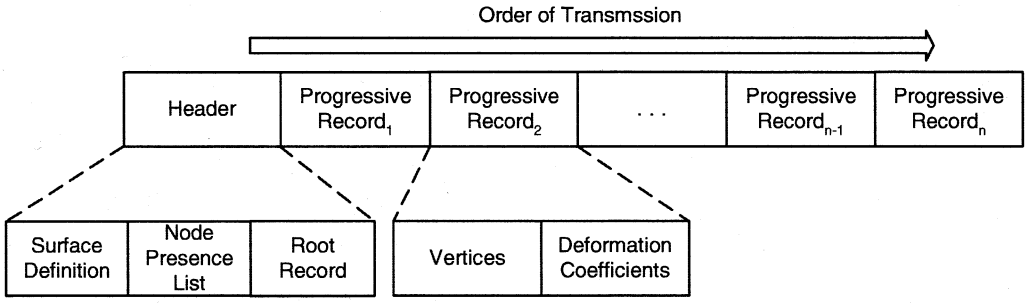


Fig. 3. Structure of a NURBS stream.

They are defined as follows:

$$\Delta h^d = (\delta_{th}(i-1)\,\phi_i\,\delta_{th}(i-1)),$$
$$\phi_i = (-\phi_{i-1} - \delta_h(i-1) - \phi_{i-1}) \tag{12}$$
$$\Delta v^d = (-\delta_{tv}(i-1)\,\phi_i - \delta_{tv}(i-1)),$$
$$\phi_i = (\phi_{i-1}\,\delta_v(i-1)\,\phi_{i-1}) \tag{13}$$

where $\delta_h(m)$ and $\delta_v(m)$ give the horizontal and vertical index differences for pairs of neighboring nodes having a relative level distance m. $\delta_{th}(m)$ and $\delta_{tv}(m)$ perform similar calculations on neighboring nodes in toroidal quadtrees.

B. The Structure of a NURBS Stream

Unlike the linear quadtree suggested by Balmelli *et al.* [22], a NURBS stream does not consume extra spaces to hold empty nodes. This is very important since most quadtrees are likely unbalanced. Fig. 3 shows the structure of a NURBS stream. It consists of a *header* and a sequence of *progressive records*. The header consists of a *surface definition*, a *node presence list*, and a *root record*. The surface definition consists of the degrees, knots and control points of a NURBS surface. In particular, the control points provide an aid for the client to modify the shape of the NURBS surface. The degrees and knots help the client determine the deformation region of each control point. 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 root record stores information of the root quadtree node. Each of the progressive records stores information of a single quadtree node. It consists a set of *vertices* and a set of *deformation coefficients* associated with the vertices. The vertices are the list of (x, y, z) coordinates of the vertices in the quadtree node. The progressive records are arranged in ascending order of node indices.

When transmitting a NURBS stream to the client, the server first sends the header to the client followed by the progressive records. As the client receives the progressive records, it may begin to refine as well as render the polygon model of the deforming NURBS surface.

VI. Collaborative Virtual Sculpting

To provide an intuitive interface for the user, our sculpting method uses the *CyberGlove* as an input device for object modification. Our idea is to create a *hand surface* using the bicubic tensor product B-spline interpolating all key data points of the *CyberGlove* [18]. 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*. With ray-projection, the user manually specifies the location of a center of projection, P_c. A set of rays are then projected from P_c through individual object vertices onto the hand surface to establish a mapping. We refer to the region of the object mapped to the hand surface as the *Editing Region*. Once the mapping is established, the user may deform the object model simply by changing the hand gesture. During the sculpting process, the user may adjust the location of P_c interactively to change the size of the editing region.

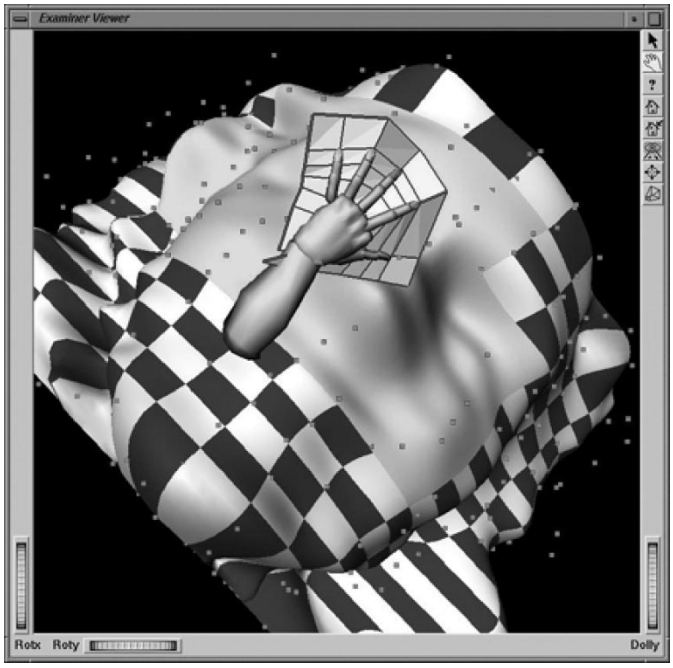
In order to support multiuser collaborative virtual sculpting in a distributed environment, we need to incorporate a flexible locking mechanism that allows any participant to define and lock a region of the object for sculpting. This locking mechanism must make sure that no other users are editing a region before locking it for a user and that there are no data inconsistencies among the participating machines. We present here a simple locking mechanism to do this. We also present the client–server and client–client interactions during a sculpting section.

A. Determining and Locking an Editing Region

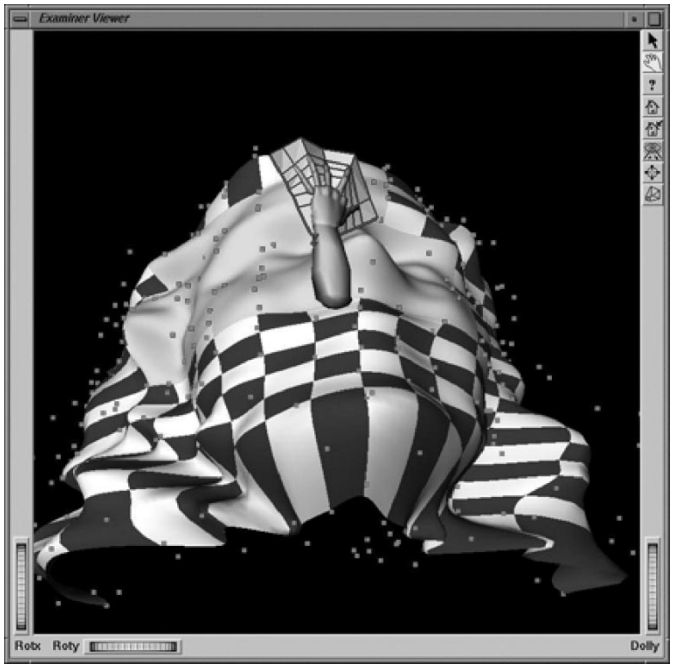
Most earlier systems for distributed virtual environments, such as **SIMNET** [4] and **MASSIVE-2** [23], do not provide an explicit locking mechanism as they do not consider collaboration. Systems that consider collaboration, such as **DIVE** [3] and **PaRADE** [24], employ a conservative concurrency control to prevent concurrent modification of distributed objects. This type of concurrency control mechanisms, however, does not provide the required interactivity for virtual sculpting.

In a collaborative environment, multiple clients may sometimes want to edit the same object simultaneously. If we allow a client to lock the entire object for sculpting, other clients may not be able to participate. This leads to a bottleneck in collaborative sculpting. To enhance the collaboration, we allow a client to lock only the editing region(s) that he/she is sculpting, instead of the whole object. When a client manipulates an editing region of a deformable object, the rest of the object will not be affected by this manipulation and can thus be edited by other clients concurrently. A client may modify one or more editing region(s) if the regions are available.

To determine the boundary of an editing region, we consider the local modification property 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 $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$



(a)



(b)

Fig. 4. Virtual sculpting of a human head model.

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 all the control points that fall inside the hand surface. Fig. 4 shows the sculpting of a human head model with the *CyberGlove* using our prototype system described here. The 6×6 polygon mesh associated with the virtual hand represents the hand surface. The grey region on the human head model is the editing region.

To implement the locking mechanism for sculpting, we maintain at each client an *editing list* containing a set of Boolean flags, each corresponding to a parameter region $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ of the object for sculpting. A bit is set to 1 if the corresponding parameter region is currently manipulated by one of the participants. When a client wants to modify the shape of an object, it first determines the editing region that it is interested in and compares the set of parameter regions in the editing region with the editing list. If all parameter regions in the editing list are currently set to 0 (i.e., they are all available), the client will be granted the right to sculpt the region and a *locking message* will be broadcasted to all clients to update the corresponding bits of their editing lists.

However, if two clients request for the same editing region at nearly the same time, both clients may find from their local editing lists that the region is available and start to sculpt the local copies of the object. This may cause inconsistency. To solve this problem, we introduce a *timeout period*. After a client has sent a locking message, it needs to wait for the timeout period. If it does not receive any locking messages with an earlier timestamp from other clients when this period expires, it may start to sculpt. However, if the client receives a locking message with an earlier timestamp after this period has already expired, it will then need to roll back the sculpting work to avoid inconsistency.

The reason for the introduction of the timeout period is that most users in general do not like the roll-back experience. The timeout period helps resolve most of the concurrent requests and hence significantly reduce the number of roll-backs required. To determine the timeout period, $t_{timeout}$, we need to consider the time needed for a client to send a locking message and for this message to be received by another client. Hence, $t_{timeout} = \alpha_t(t_{latency} + t_{message})$ where $t_{latency}$ is the network latency between the two clients and $t_{message}$ is the time taken to send the message. α_t is a tolerant factor to compensate for the fluctuation in the network performance.

When the client finishes the sculpting, it broadcasts a *lock release message* to all clients to clear the appropriate 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 this distributed locking scheme to eliminate the need for a central server, which may become the bottleneck due to the large number of editing requests.

B. Client–Server Interactions

The distributed virtual sculpting process consists of two stages, the preprocessing stage and the run-time stage. The preprocessing stage involves only client–server interactions, in which the client requests the server for objects. The run-time stage involves both client–server interactions, in which the client may occasionally request the server for object refinement information, and client–client interactions, in which a client may frequently send locking messages, update messages, and lock release messages to other clients. Fig. 5 shows the major operations in the client–server interactions. In the preprocessing stage, the server constructs a NURBS stream for each deformable object. Upon a client's request, the server sends the NURBS stream to the client as described in Section V. After the client has received the NURBS stream, it refines the polygon

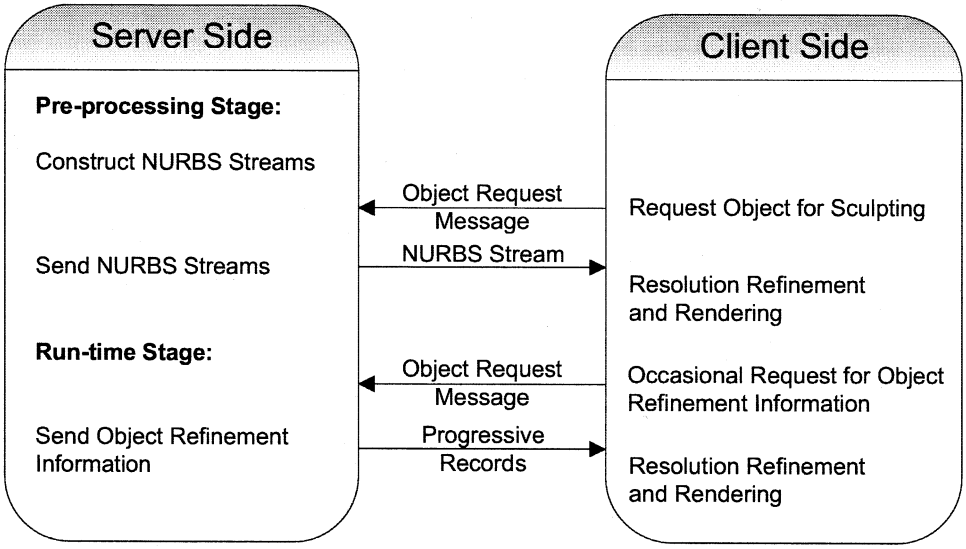


Fig. 5. Client–Server interactions.

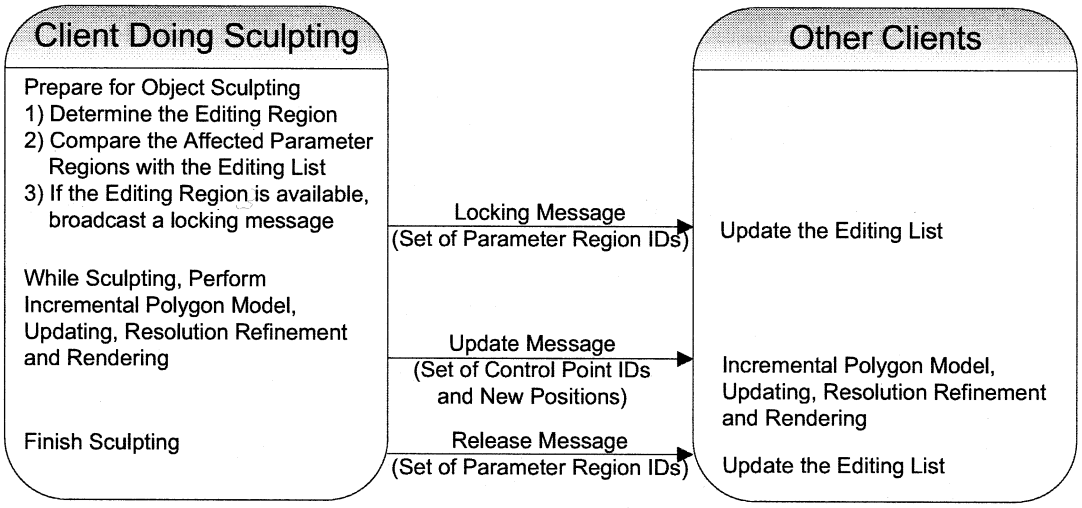


Fig. 6. Client–client interactions.

model for rendering according to its own view parameters, such as object distance, object moving speed and the viewer's line of sight [25].

Although the advantage of using NURBS surfaces is that they can be infinitely refined, it is not feasible to generate a very high resolution NURBS stream to cater for any possible view conditions as it will be very costly in terms of memory and transmission cost. Hence, during the preprocessing stage, we may only need to transmit each NURBS stream up to a resolution high enough for most view conditions. During run-time, a client may occasionally needs to refine a deformable object beyond the resolution of its local NURBS stream, it will then need to request the server for more refinement information (i.e., progressive records) of the NURBS stream.

C. Client–Client Interactions

During run-time, if a client is sculpting 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 moving control points within the editing region and the updated positions of these control points.

When a participating client receives 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. Fig. 6 shows the main processes and the broadcast messages involved in the client–client interactions.

VII. Results and Discussions

We have implemented various components of the system in C++. 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 the GesturePlus Library from Virtual Technology. The graphics engine is written in OpenGL and OpenInventor. We tested the system on a SGI Onyx^2 machine with eight 195 MHz R10000 processors and a SGI Octane machine with two 250 MHz R10000 processors, each with only one CPU activated.

TABLE I
SIZES OF NURBS STREAMS FOR VARIOUS TEST MODELS

	Model A	Model B	Model C	Model D
Number of Control Points	49	400	900	1600
Size of the NURBS Stream (Kbytes)	3.95	65.93	158.85	297.36
Number of Progressive Records in the NURBS Stream	20	408	994	1868

The Onyx^2 machine was set up as a server while the Octane machine was set up as a client. These machines are physically connected to our university network through 10 Mbps Ethernet connections. At the time of our experiments, the bandwidth available to us was about 2 Mbps.

A. Experiment 1

In this experiment, we perform a number of tests to study the performance in constructing and transmitting NURBS streams. The models used in our experiments are NURBS surfaces with different numbers of control points. We apply knot insertion [26] to subdivide each object into Bézier surface patches and construct a hierarchical surface on these patches to form a single quadtree structure. Table I shows information of each of the test models, including the number of NURBS control points, the size of the NURBS stream and the number of progressive records in the NURBS stream.

Fig. 7 compares the processing time for constructing the NURBS streams of the test models. The operations involved in constructing a NURBS stream include the generation of an initial polygon model, the construction of a hierarchical surface and the computation of the deformation coefficients. From the result, it is found that the construction time of a NURBS stream is approximately a polynomial function of the number of NURBS control points in the model, i.e., the complexity of the model. In addition, this construction time is generally too long for the NURBS stream to be constructed in real-time. If we let the client handle this process, the user may then need to wait for a long time before he/she may start visualizing or manipulating the object. The situation may be worse if the user needs to simultaneously work on more than one deformable object. Hence, in VSculpt, the server is responsible for constructing the NURBS streams and distributes them to the clients progressively to allow efficient model replication.

Fig. 8 compares the time for transmitting the NURBS streams of the test models. Results show that the deformable models can be transmitted to the clients in a very short time, and the transmission time is approximately proportional to the size of the NURBS streams. In addition, since NURBS streams support progressive reconstruction, even if the client needs to handle many deformable objects at the same time, it may still be able to visualize all the objects. This is because the client may first render the deformable objects in low resolutions and then progressively refine their resolutions as more progressive records are being received.

Finally, we would like to study the detailed transmission performance of a NURBS stream. In this test, we used a human head model (Model B) as shown in Fig. 4. The human head model is a NURBS model with 400 control points. We apply knot insertion [26] to subdivide it into 289 Bézier surface patches and then construct a hierarchical surface on these patches to form the polygon model for transmission. Table II shows the size and the transmission time of the NURBS stream. The NURBS stream is 65.93 Kbytes in size and transmitted in 0.0772 s. We have also measured the size and the transmission time of the header and of each single progressive record. The header is 5491 bytes in size and takes 6.184 ms to transmit to the client. A single progressive record is 152 bytes in size and takes 0.174 ms to transmit to the client. If we assume that the Internet bandwidth is about one-tenth of the LAN, i.e., roughly 0.2 Mbps, it will take 0.772 s to transmit the whole NURBS stream. However, since the NURBS stream supports progressive reconstruction, the client is expected to be able to visualize a coarse model of the deformable object in a time less than this transmission time.

B. Experiment 2

There are two ways to handle model refinement when a deformable object is undergoing deformation at the client. The client may either perform the resolution refinement process (Section IV-B) itself or request the server to transmit more progressive records of the corresponding NURBS stream to the client. In this experiment, we would look at the costs of them. Table III shows the computational cost of performing a single subdivision, i.e., from one parent node to produce four child nodes, and the transmission cost if the client requests the server to transmit the four progressive records of the four child nodes to the client. Results show that with the current configuration, it is cheaper for the client to simply request the server to transmit the progressive records than to compute them locally. This will also save the CPU time of the client for other time critical operations. However, if the client CPU becomes more powerful or the network bandwidth becomes too small, at some point, it may be cheaper to perform the resolution refinement locally.

C. Experiment 3

Experiment 3 evaluates the transmission performance of different kinds of messages in the client–client interactions as described in Section VI-C. They are the locking messages, lock release messages and update messages. Each locking message or lock release message contains the editing list. An update message contains the ID's and the updated positions of the set of moving control points.

We perform the experiment with the human head model as shown in Fig. 4. The model contains 529 parameter regions. For this model, each locking message or lock release message uses 4 bytes to store the timestamp and 67 bytes to store the editing list. In our measurement, it took 0.081 ms to transmit through our LAN. If we use our default assumption that the Internet is about one-tenth the bandwidth of our LAN, it will takes about 0.81 ms to transmit a locking message through the Internet.

To determine the timeout period, $t_{timeout}$, described in Section VI-A, we need to determine the latency of the network, $t_{latency}$. To do this, we can measure the round-trip time, t_{rt}, of the network using the **ping** system program. This program

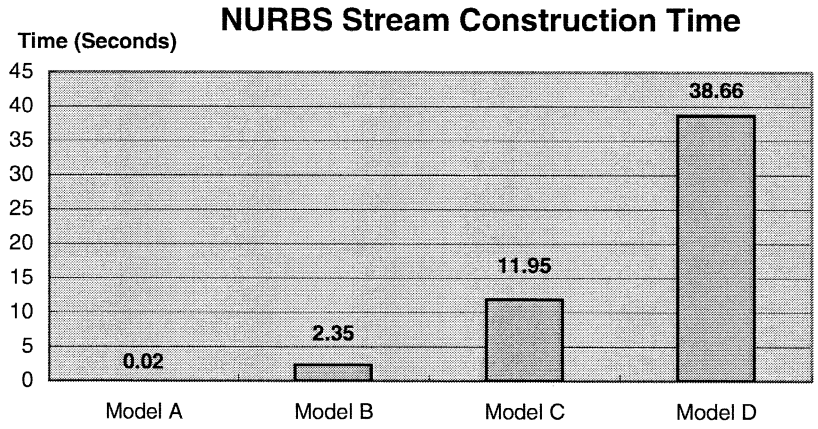


Fig. 7. Processing times for constructing the NURBS streams of the test models.

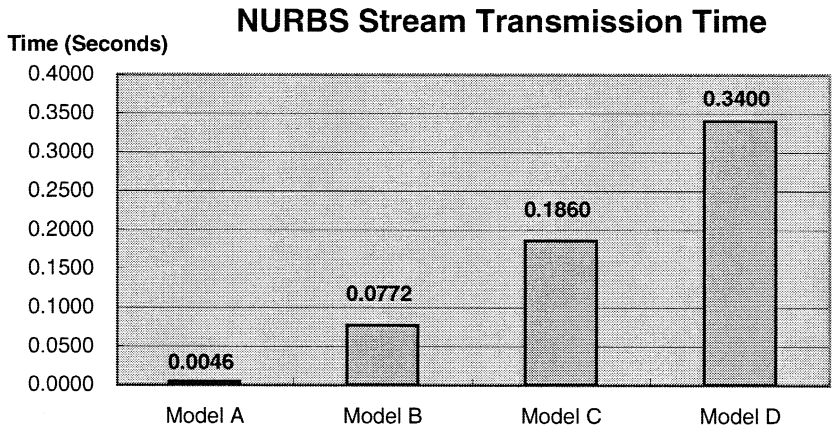


Fig. 8. Times for transmitting the NURBS streams of the test models.

TABLE II
RECORD SIZES AND TRANSMISSION TIMES OF DIFFERENT PARTS OF THE NURBS STREAM

Part Name		Data Type	Size	Transmission Time
Header	Surface Definition	Structure of NURBS	5016 bytes	5.643 ms
	Node Presence List	Short	171 bytes	0.193 ms
	Root Record	Float	304 bytes	0.348 ms
A Single Progressive Record	Vertices	Float	24 bytes	0.027 ms
	Deformation Coefficients	Float	128 bytes	0.147 ms
The Whole NURBS Stream (with 408 progressive records)			65.93 Kbytes	0.0772 seconds

sends a small packet to a specified host and times the duration taken for this packet to bounce back. $t_{latency}$ is approximately equal to half of t_{rt}. Table IV shows some example round-trip times for different network connections.

As an example, if one client is in Hong Kong and the other is in the US, $t_{latency}$ is 80 ms. To send a locking message, $t_{message}$ is 0.81 ms. If we set the tolerant factor, α_t, to 1.5, the timeout period, $t_{timeout}$, will be 121 ms. If both clients are within Hong Kong, $t_{timeout}$ will be 16.2 ms.

For the update messages described in Section VI-C, we use 2 bytes to store each control point ID and 3 bytes to store each coordinate component of the updated positions. In typical situations, the number of control points simultaneously affected by the sculpting process is no more than 50. Thus, a typical update message will be no more than 550 bytes in size. In our measurement, it took 0.63 ms to transmit through our LAN. Hence, we expect that it will take about 6.3 ms to transmit an update message through the Internet, which is a very short time. However, if there is a need to reduce the size of the update messages, geometry compression techniques [27], [28] can be applied, which will reduce the size to less than one-third of the current one. On the other hand, if we compare the transmission time of an update message with the roundtrip time presented in Table IV, the major source of delay in transmitting the update message is the

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT METHODS FOR OBTAINING THE REFINEMENT INFORMATION

Methods for Obtaining the Refinement Information	Time Taken
Generating new child nodes at the client using resolution refinement	12.96 ms
Transmitting the progressive records from the server through a LAN (approximately 2 Mbps available)	0.70 ms
Transmitting the progressive records from the server through the Internet (approximately 0.2 Mbps available)	6.95 ms

TABLE IV
EXAMPLE ROUND-TRIP TIMES FOR DIFFERENT NETWORK CONNECTIONS

Network Connection	# of Hops	RTT
LAN (2 hosts within the university network)	4	0.64 ms
Internet (Local connection, 2 hosts within HK)	10	10 ms
Internet (Overseas connection, 1 host in HK, another in US)	13	160 ms

network latency rather then the size of the message, especially if the message is to be sent to an overseas client.

D. Additional Comments on the Editing List

In our method, we use a distributed editing list instead of a central server to control simultaneous editing of the same object. This can prevent creating a bottleneck at the server and reduce the network latency when the clients raise their requests continuously. This method can also be applied in the object level, with a separate editing list to indicate the availability of individual objects. If an object is available for editing, the corresponding bit of this editing list is clear; otherwise, it is set to 1. With this locking method, whenever a client raises a request, it only needs to lookup its local copy of the editing list. The locking message is sent only if the requested object and editing region are available. Hence, unlike Nishino's method [14], our method generates network traffics for successful requests only. For the tessellation process, Nishino's method does not perform retessellation and the model is fixed at a resolution. In our case, we adapt our real-time NURBS rendering technique to allow real-time resolution refinement at the client, optimizing the rendering performance and improving the output quality.

E. User Experience

To obtain some user experience on the system, we have conducted a very preliminary experiment. We have invited two groups of users to test our prototype system. One group of users are novice users, who do not have any experience in using the *CyberGlove*. The other group of users are experienced users, who do not only have experience in using the *CyberGlove*, but also have certain knowledge on geometric modeling. We give both groups of users a short briefing session, but without any formal training, on how to operate our prototype system.

Initially, some of the novice users have some difficulties in using the *CyberGlove* to change the shape of the object model. However, they generally can operate our system smoothly after practicing for about 10 min or so. Another problem that the novice users encountered is that they get tired with using the *CyberGlove* after 20–30 min. This is probably because they tend to move their arms a lot when they are editing an object. On the other hand, most of the experienced users can operate our system smoothly in a very short period of time. They generally find our prototype system much easier to edit objects than the commercial modeling packages. From the feedback of both groups of users, our prototype system is in general very intuitive and easy to use. The *CyberGlove* provides a very *natural* interface for editing object models.

However, some of the users have commented that as they are editing an object model, they may also want to look around the object to inspect its shape from time to time. Unfortunately, since both of their hands are already occupied, there is no mechanism for them to rotate their view. At one point, we thought of attaching a 3-D tracker to the user's head, but this require the user to wear a head mounted display in order to be useful. However, existing head mounted displays have their own limitations too. Our temporarily solution to this problem is from observing how we would do in our daily life when we are physically changing the shape of an object. When we are molding an object, if we need to use both hands to do it, we would fix the object to a device. On the other hand, if we would like to be able to inspect the overall shape of an object as we edit it, we would use one hand to hold the object and the other hand to edit it. Hence, in our system, if a user wants to be able to inspect an object as he/she is editing it, then he/she can only use one hand to do the editing, freeing the other hand for controlling the view point.

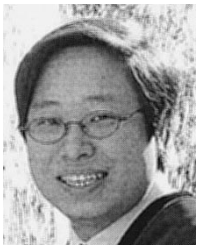
VIII. CONCLUSION

In this paper, we have presented a framework for real-time distributed virtual sculpting. The framework extends our earlier work on distributed virtual environment [1], [2] to support collaborative sculpting. To do this, we adapt our real-time techniques for rendering deformable NURBS surfaces [16], [17] and for virtual sculpting [18] into the new framework. We have introduced the NURBS streams for transmitting deforming objects through the network. We have also presented the idea of editing region and the corresponding locking mechanism to allow simultaneous object sculpting by multiple users. Results show that our method can support interactive collaborative virtual sculpting over the Internet.

REFERENCES

[1] J. Chim, M. Green, R. Lau, H. Leong, and A. Si, "On caching and prefetching of virtual objects in distributed virtual environments," *Proc. ACM Multimedia*, pp. 171–180, Sept. 1998.
[2] J. Chim, R. W. H. Lau, H. V. Leong, and A. Si, "CyberWalk: A web-based distributed virtual walkthrough environment," *IEEE Trans. Multimedia*, vol. 5, pp. 503–515, Dec. 2003.
[3] C. Carlsson and O. Hagsand, "DIVE—A multi-user virtual reality system," in *Proc. IEEE VRAIS*, 1993, pp. 394–400.
[4] J. Calvin, A. Dicken, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The SIMNET virtual world architecture," in *Proc. IEEE VRAIS*, 1993, pp. 450–455.
[5] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz, "NPSNET: A network software architecture for large scale virtual environments," *Presence: Teleop. Virtual Environ.*, vol. 3, no. 4, pp. 265–287, 1994.
[6] S. Benford, J. Bowers, L. Fahlen, and C. Greenhalgh, "Managing mutual awareness in collaborative virtual environments," in *Proc. ACM VRST*, 1994, pp. 223–236.

[7] I. Pandzic, T. Capin, E. Lee, N. Thalmann, and D. Thalmann, "A flexible architecture for virtual humans in networked collaborative virtual environments," in *Proc. Eurographics'97*, vol. 16, Sept. 1997, pp. 177–188.
[8] C. Babski *et al.*, "The COVEN Project: Exploring applicative, technical and usage dimensions of collaborative virtual environments," *Presence: Teleop. Virtual Environ.*, vol. 8, no. 2, pp. 218–236, 1999.
[9] T. Galyean and J. Hughes, "Sculpting: An interactive volumetric modeling technique," in *Proc. ACM SIGGRAPH'91*, vol. 25, 1991, pp. 267–274.
[10] C. Shaw and M. Green, "THRED: A two-handed design system," *Multimedia Syst. J.*, vol. 5, pp. 126–139, Mar. 1997.
[11] K. Kameyama, "Virtual clay modeling system," in *Proc. ACM VRST*, Sept. 1997, pp. 197–200.
[12] F. Kuester, M. Duchaineau, B. Hamann, K. Joy, and A. Uva, "3DIVS: 3-dimensional immersive virtual sculpting," in *Proc. Workshop on New Paradigms in Information Visualization and Manipulation*, Nov. 1999, pp. 92–96.
[13] L. Cutler, B. Fröhlich, and P. Hanrahan, "Two-handed direct manipulation on the responsive workbench," in *Proc. ACM Symp. Interactive 3D Graphics*, Apr. 1997, pp. 107–114.
[14] H. Nishino, K. Utsumiya, A. Sakamoto, K. Yoshida, and K. Korida, "A method for sharing interactive deformations in collaborative 3D modeling," in *Proc. ACM VRST*, Dec. 1999, pp. 116–123.
[15] V. Anupam and C. Bajaj, "Shastra: Multimedia collaborative design environment," *IEEE Multimedia*, vol. 1, no. 2, pp. 39–49, 1994.
[16] F. Li, R. Lau, and M. Green, "Interactive rendering of deforming NURBS surfaces," in *Proc. Eurographics'97*, vol. 16, Sept. 1997, pp. 47–56.
[17] F. Li and R. Lau, "Incremental polygonization of deforming NURBS surfaces," *J. Graph. Tools*, vol. 4, no. 4, pp. 37–50, 1999.
[18] J. Wong, R. Lau, and L. Ma, "Virtual 3D sculpting," *J. Visualization Comput. Animation*, vol. 11, no. 3, pp. 155–166, 2000.
[19] E. Cohen, T. Lyche, and R. Riesenfeld, "Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics," *Comput. Graph. Image Process.*, vol. 14, Oct. 1980.
[20] P. de Casteljau, "Courbes et surfaces à pôles," *S. A. André Citroen*, 1959.
[21] I. Gargantini, "An effective way to represent quadtrees," *Commun. ACM*, vol. 25, pp. 905–910, Dec. 1982.
[22] L. Balmelli, J. Kovačević, and M. Vetterli, "Quadtrees for embedded surface visualization: Constraints and efficient data structures," in *Proc. IEEE ICIP*, vol. 2, Oct. 1999, pp. 487–491.
[23] C. Greenhalgh and S. Benford, "A multicast network architecture for large scale collaborative virtual environments," in *Proc. . Eur. Conf. Multimedia Applications, Services and Techniques (ECMAST '97)*, May 1997, pp. 113–128.
[24] D. Roberts and P. Sharkey, "Maximizing concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimizing the effect of network delays," in *Proc. IEEE Workshop on Enabling Technology: Infrastructure for Collaborative Enterprise (WETICE '97)*, 1997, pp. 161–166.
[25] R. Lau, D. To, and M. Green, "An adaptive multi-resolution modeling technique based on viewing and animation parameters," *Proc. IEEE VRAIS*, pp. 20–27, 1997.
[26] L. Piegl and W. Tiller, *The NURBS Book*. Berlin, Germany: Springer-Verlag, 1995.
[27] P. Alliez and M. Desbrun, "Progressive compression for lossless transmission of triangle meshes," in *Proc. ACM SIGGRAPH'01*, Aug. 2001, pp. 195–202.
[28] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive geometry compression," in *Proc. ACM SIGGRAPH'00*, July 2000, pp. 271–278.



Frederick W. B. Li received both the B.A. (Honors) in Computing Studies and M.Phil. degrees from the Hong Kong Polytechnic University in 1994 and 1998, respectively, and the Ph.D. degree in computer graphics from the City University of Hong Kong in 2001.

He is currently the Research Manager of the "Web-based 3D collaborative engine" project funded by the Hong Kong Government Innovation and Technology Fund, Pacific Century CyberWorks (PCCW), and Sun Microsystems. His research interests include surface modeling, virtual reality, computer animation and computer networking.

Rynson W. H. Lau (M'88) received a (top) first class honors degree in computer systems engineering in 1988 from the University of Kent, Canterbury, U.K., and the Ph.D. degree in computer graphics in 1992 from the University of Cambridge, U.K.

He is currently an Associate Professor at the City University of Hong Kong. Prior to joining the university in 1998, he taught at the Hong Kong Polytechnic University. From 1992 to 1993, he worked at the University of York, U.K., on a defense project on image processing. His research interests are in computer graphics, virtual reality and multimedia systems.

Dr. Lau is a member of the IEEE Computer Society and the ACM.

Frederick F. C. Ng received the B.A. degree (Honors) in computing studies from the Hong Kong Polytechnic University in 1998.

He is currently a Research Student with the Computer Science Department, City University of Hong Kong. His research interests include computer graphics and distributed virtual environments.