# Virtual 3D Sculpturing with a Parametric Hand Surface

Janis P.Y. Wong [§] Rynson W.H. Lau [†] Lizhuang Ma [‡]

[§] Computer Graphics and Media Laboratory, Deparment of Computing
The Hong Kong Polytechnic University, Hong Kong
Email: cspywong@comp.polyu.edu.hk

[†] Department of Computer Science
City University of Hong Kong, Hong Kong
Email: rynson@cs.cityu.edu.hk

[‡] State Key Laboratory of CAD&CG
Zhejiang University, Hangzhou 310027, P.R. China

## Abstract

*Many techniques have been developed for 3D object deformation. These techniques have been widely used in most CAD/CAM systems. Intuitive while efficient methods for interactive 3D object deformation in a VR environment, however, are rare. One of our current research projects is to develop such a technique based on the use of a sensor glove. The idea is to create a hand surface interpolating through all the data points of the sensor glove. Through mapping the vertices of an object model to the hand surface, the object model may be deformed simply by changing the hand gesture. An initial method for implementing this idea was presented in our recent paper. In this paper, we discuss some of the limitations of the earlier method and present a refined method that overcomes most of the limitations. The resulting method is both efficient and intuitive as demonstrated by the results of our experiments.*

## 1. Introduction

Parametric surfaces are widely used for object modeling. Free-Form Deformations (FFDs) [1] embed the object model to be deformed inside a parallelepipedical 3D lattice defined by a tensor product piecewise tricubic Bezier volume. The deformation of the 3D lattice will result in the deformation of the embedded object model. Extended FFDs [2] do not require enclosing the object with a regular 3D lattice, but rather, arbitrarily shaped lattices are built by combining several tricubic Bezier volumes, each represents an individual FFD. However, extra effort is needed to maintain the continuity between these volumes. NURBS-based FFDs [3] extend FFDs by redefining the Bezier volumes by non-uniform rational B-splines ones. The resultant lattice is non-uniformly subdivided and thus different weightings can be given to different regions of the model to create different deformation effects. These techniques have been implemented with a 3D pointing input device [4] and a 3D tracker [5]. However, all the above techniques concern modeling through deforming the control lattice instead of the object itself, which is not intuitive and is only suitable for expert users who have knowledge in parametric object modeling.

Direct manipulation of FFDs [6], on the other hand, allows the user to move the sample points on the object model and compute the necessary changes to the control points of the FFD. This method is more intuitive. However, similar to other algorithms, it concerns the modification of object points one by one. In addition, it is computationally very expensive because this method is based on the technique of least square calculation. Furthermore, the problem could be very complex when more than one vertex point is to be moved at the same time, and the solution may not be unique.

JDCAD [7] is a system built on the MR Toolkit [8]. It allows interactive 3D object creation together with key-frame animation. The system made use of both a 3D magnetic tracker and a keyboard as input devices. Unlike the work described in this paper, their work aimed at providing a better interface for object creation by assembling primitive volumes instead of creation by using FFD techniques.

Kurmann in [9] proposed a system for interactive spatial modeling to support architectural design in a VR environment. By using the mouse or other possible 3D input devices, the user may interact with the system in a 3D environment. This enables the user to formulate design ideas in a 3D space. Similar to the JDCAD system, this system makes use of predefined objects, such as building elements or furniture, to create the scene. It also introduced the concept of positive (solid) and negative (space) volumes for scene creation. This concept was created to coincide with architectural design needs.

The systems discussed so far concern mainly scene assembling but not of artistic sculpturing in an interactive environment. Kameyama designed a system [10] for creating deformable virtual objects using a 3D position tracker and a tactile feedback device. The system provides functions for creating a virtual object, but there is no means for the manipulating or modeling of an existing 3D model.

In this paper, we present a method for virtual sculpturing based on the use of a sensor glove for deforming virtual objects. This technique extends and refines our earlier paper described in [11]. As will be shown, the new method is intuitive and efficient. The rest of this paper is organized as follows. In section 2, we summarize our previous work on virtual sculpturing and highlight the differences between our earlier method and the method presented here. We then describe the creation and the update of the hand surface in section 3. In section 4, we discuss the mapping of object points to the hand surface and in section 5, we describe how the object is deformed. In section 6, we show and discuss some experimental results of our method. Finally, we present the conclusion of our work in section 7 of this paper.

## 2. Previous Work

In this project, we are developing an intuitive method for direct object creation and modification using the sensor glove. The idea is to create a parametric hand surface, which interpolates through all the key data points of the sensor glove. These data points indicate the finger joint positions of the user’s hand. The hand surface is then mapped to the object model to be deformed. Based on changing the user’s hand gesture, the object model can be deformed accordingly. The major advantage of this approach is that multiple control points of the object surface can be modified at the same time in an intuitive manner.

In our earlier paper [11], we described our initial method for implementing the new approach. In that method, the hand surface is represented by a Bezier surface. Before the object deformation process, the user is required to open his hand widely with the palm surface facing the object to be deformed. This facilitates the creation of a coplanar surface, called the base plane. In other words, the base plane is defined from the co-domain of the hand surface in flat state. This base plane is used for mapping each of the vertices of the object model being modified to a parametric coordinate of the hand surface. To do this, we first project each vertex to the base plane in a direction parallel to the normal vector of the base plane, and the base plane is uniformly subdivided to determine the parameter coordinate of the hand surface to which the vertex maps. When the hand surface changes due to the change of the user' s hand gesture, the positions of the affected object vertices need to be updated. For each vertex of the object model, the normal vector, $N(u_p, v_p)$, at the corresponding parametric coordinate, $H(u_p, v_p)$, of the hand surface is multiplied by the projected distance parameter, $d_p$, of the vertex from the base plane, resulting in a moving vector, $d_p \cdot N(u_p, v_p)$. This moving vector is then added to $H(u_p, v_p)$ to determine the new position of the object vertex.

However, $N(u_p, v_p)$ may change very rapidly and the deformation of the object model may become difficult to control. In addition, self-intersection of the object model may sometimes occur. To reduce this unwanted effect, a blending parameter, $\alpha$, is introduced to reduce the effect of $N(u_p, v_p)$ on the deformation. The new position of the object vertex is then determined as follows:

$$Q = H(u_p, v_p) + d_p[(1-\alpha)N(u_p, v_p) + \alpha N_H^0]_I$$

where $N_H^0$ is the normal vector of the base plane, and $[V]_I$ denotes the unit normal vector of *V*. By selecting different values of $\alpha$, the user could achieve different effects. When $\alpha = 0$, the deformation will be completely based on the normal vector of the hand surface. When $\alpha = 1$, the mapping is similar to FFDs in which the deformation is a linear extrusion of the hand surface along $N_H^0$. When $\alpha$ decreases from 1 to 0, the corresponding mapping becomes more and more sensitive to $N(u_p, v_p)$. The deformation of the object model also becomes less and less coherent to the shape of hand surface.

Although the method is simple to implement, it suffers from a number of constraints. In this paper, we address some of these constraints and propose alternatives for overcoming them.

For fast prototyping, the hand surface is represented by a Bezier type tensor product surface. As inherited from the properties of Bezier representations, this surface suffers from the undesired global control of deformation. This means that a little movement on any part of the hand surface, say a fingertip, will lead to the change of shape of the whole hand surface. In addition, in our previous method, we set the control points of the hand surface to the data points of the sensor glove. This may only offer an indirect deformation of the object model. To improve the modeling of the user's hand, in the new method, we propose to fit a bicubic B-spline tensor product surface to the data points. We also discuss an efficient computation and rendering method (see section 3).

In our earlier method, the user is required to set his/her hand in a coplanar state before the object deformation process, so that the base plane of the hand surface may be created. However, when we experimented with the sensor glove, we found that it was in fact very difficult to set all joint positions of the user's hand on a single plane. Any deviation of the finger joints from the base plane may lead to errors in the initial parametric mapping from object vertices to the hand surface. In addition, because the mapping is via the base plane, the amount of movement that the user's hand can make from the initial coplanar state is small. In this paper, we present a new mapping method based on ray-projection to replace the parallel projection used in our previous work. With the new mapping method, mapping is via a 3D triangulated hand model, and therefore, the user is no longer required to set his/her hand in a coplanar state before the deformation process. This is discussed further in section 4.

Finally, since in our earlier method, deforming the object model is based on the local hand surface normal, $N(u_p, v_p)$, and the distance parameter, *d*, this may cause undesirable object crossover. To counteract this, we suggested a weighting of $N(u_p, v_p)$ by the normal vector of the base plane. This produces a pushing effect on the whole object rather than modifying the shape of it. In section 5, we introduce a new modification technique, which can adapt to different hand gestures and alleviate the crossover problem.

## 3. Parametric Hand Surface Model

The development of our hand surface model is based on the CyberGlove™ from Virtual Technologies, Inc., but extension of the method to other sensor gloves is expected to be straightforward. In the following subsections, we briefly introduce the CyberGlove™, and then we present an efficient method for creating the hand surface.

### 3.1. Collection of Data from the CyberGlove™

The CyberGlove™ that we use has 18 sensors. The *Virtual*Hand™ software library [12,13] provides functions for reading the transformation matrices of $5 \times 3$ articulated rigid segments comprising the virtual hand model. These matrices hold the transformations to successive joint origins. The first dimension refers to the 5 fingers and the second dimension refers to the first, second and third joints nearest to the fingertip. Each of the segments is in its local coordinate. Applying the specific transformation matrix to one joint, the new matrix holds the coordinate system for the next joint. The hand model is thus formed as an articulated model with each segment in its local coordinate system. From these transformation matrices, we can determine $5 \times 3$ joint positions. Since the 18-sensor CyberGlove do not explicitly measure the positions of the fingertips, these data must be determined by some other means. As for most normal human hand motion, the joints nearest to the fingertips are correlated to the neighbouring joint. Therefore, $5 \times 1$ fingertip positions can be predicted by this phenomenon [14]. We then deduce the centre of palm from the wrist and the joints nearest to the wrist. By using a 3D magnetic tracker, the wrist position and the hand orientation can be obtained. The first joint position of each finger is thus deduced to be at the position 2/3 from the wrist to the joints nearest to the wrist, which positions have already been obtained from the transformation matrices (Figure 1).

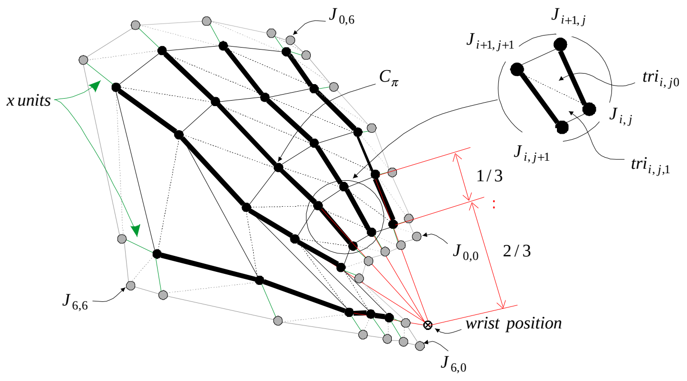


**Figure 1: The 3D triangulated hand model.**

Now, we have $5 \times 5$ joint data points representing the feature gesture of the hand. We then extend the data points to $7 \times 7$ simply by extending the last segments in each of the dimensions on the edge by *x* unit(s). The border part is used to prevent unmatched gaps when partial modification is applied to an object. This can produce an effect of modeling the elastic property of the object model. The more elastic the object, the smaller the parameter *x* and the narrower the border. Detail of the modification is discussed in section 5.

## 3.2. Efficient Hand Surface Modeling

In our implementation, we create the hand surface with a bicubic B-spline tensor product surface. A bicubic B-spline tensor product surface is defined as follows:

$$\begin{bmatrix} J_{0,0} & \cdots & J_{0,6} \\ \vdots & \ddots & \vdots \\ J_{6,0} & \cdots & J_{6,6} \end{bmatrix} = \begin{bmatrix} Nu_0^3(u_0) & \cdots & Nu_8^3(u_0) \\ \vdots & \ddots & \vdots \\ Nu_0^3(u_6) & \cdots & Nu_8^3(u_6) \end{bmatrix} \cdot \begin{bmatrix} P_{0,0} & \cdots & P_{0,8} \\ \vdots & \ddots & \vdots \\ P_{8,0} & \cdots & P_{8,8} \end{bmatrix} \cdot \begin{bmatrix} Nv_0^3(v_0) & \cdots & Nv_0^3(v_6) \\ \vdots & \ddots & \vdots \\ Nv_8^3(v_0) & \cdots & Nv_8^3(v_6) \end{bmatrix} \qquad (1)$$

where $J_{i,j}$ are the positions obtained from the CyberGlove. $Nu_m^3(u_i)$ and $Nv_n^3(v_j)$ are the cubic basis functions at knots $u_i$ in *U*-dimension and $v_j$ in *V*-dimension, respectively. $P_{m,n}$ are the control points of the hand surface interpolating $J_{i,j}$.

In order to improve the efficiency of creating the hand surface with a bicubic B-spline tensor product surface, we do not determine the inverse of the matrices implicitly. Instead, we simplify a technique proposed by [15]. Since the hand surface is a tensor product surface, we may let:

$$\begin{bmatrix} E_{0,0} & \cdots & E_{0,8} \\ \vdots & \ddots & \vdots \\ E_{6,0} & \cdots & E_{6,8} \end{bmatrix} = \begin{bmatrix} Nu_0^3(u_0) & \cdots & Nu_8^3(u_0) \\ \vdots & \ddots & \vdots \\ Nu_0^3(u_6) & \cdots & Nu_8^3(u_6) \end{bmatrix} \cdot \begin{bmatrix} P_{0,0} & \cdots & P_{0,8} \\ \vdots & \ddots & \vdots \\ P_{8,0} & \cdots & P_{8,8} \end{bmatrix} \qquad (2)$$

Equation (1) becomes,

$$\begin{bmatrix} J_{0,0} & \cdots & J_{0,6} \\ \vdots & \ddots & \vdots \\ J_{6,0} & \cdots & J_{6,6} \end{bmatrix} = \begin{bmatrix} E_{0,0} & \cdots & E_{0,8} \\ \vdots & \ddots & \vdots \\ E_{6,0} & \cdots & E_{6,8} \end{bmatrix} \cdot \begin{bmatrix} Nv_0^3(v_0) & \cdots & Nv_0^3(v_6) \\ \vdots & \ddots & \vdots \\ Nv_8^3(v_0) & \cdots & Nv_8^3(v_6) \end{bmatrix} \qquad (3)$$

As both Equations (2) and (3) represent seven one-dimension B-spline, the interpolation process of the two-dimension tensor product B-spline surface is now reduced to fourteen one-dimension B-spline interpolating processes.

Now, consider solving the one-dimension interpolation problems. Consider Equation (3), for each $(a+1)$th column, where $a = 0,...,6$, the problem could be seen as follows:

Given *7* data points $J_{a,i}$ (where $i = 0,...,6$). Our goal is to find the cubic B-spline curve $S_a^v(v)$ with knot vector $\{0,0,0,0,v_1,v_2,v_3,v_4,v_5,1,1,1,1\}$ and 9 unknown control points $E_{a,n}$ (where $n = 0,...,8$) such that $S_a^v(v_i) = J_{a,i}$ and also the curve satisfies end point interpolation condition, i.e., $S_a^v(0.0) = J_{a,0}$ and $S_a^v(1.0) = J_{a,6}$.

To further improve the efficiency, the interpolation of data points are chosen to coincide with the interior knots defining the hand surface. Recalling the definition of a cubic B-spline [16], for each data point coincides with the interior knot, there are only three nonzero cubic basis functions at each of these knots.

$$S(v_i) = N_i^3(v_i)E_i + N_{i+1}^3(v_i)E_{i+1} + N_{i+2}^3(v_i)E_{i+2}$$

By setting $\alpha_i = N_i^3(v_i)$, $\beta_i = N_{i+1}^3(v_i)$ and $\gamma_i = N_{i+2}^3(v_i)$ the following tridiagonal system can then be set up, which can be solved efficiently by numerical methods [17] without evaluating the inverse of a matrix.

$$\begin{bmatrix} J_{a,1} - \alpha_1 E_{a,1} \\ J_{a,2} \\ J_{a,3} \\ J_{a,4} \\ J_{a,5} - \gamma_5 E_{a,7} \end{bmatrix} = \begin{bmatrix} \beta_1 & \gamma_1 & 0 & 0 & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & 0 & 0 \\ 0 & \alpha_3 & \beta_3 & \gamma_3 & 0 \\ 0 & 0 & \alpha_4 & \beta_4 & \gamma_4 \\ 0 & 0 & 0 & \alpha_5 & \beta_5 \end{bmatrix} \cdot \begin{bmatrix} E_{a,2} \\ E_{a,3} \\ E_{a,4} \\ E_{a,5} \\ E_{a,6} \end{bmatrix}$$

The end tangents $D_{a,0} = J_{a,1} - J_{a,0}$ and $D_{a,6} = J_{a,6} - J_{a,5}$ determine the end conditions of the B-spline as,

$$\begin{cases} E_{a,0} = J_{a,0} \\ E_{a,1} = E_{a,0} + \frac{1}{3} D_{a,0} \end{cases} \text{and} \begin{cases} E_{a,8} = J_{a,6} \\ E_{a,7} = J_{a,6} - \frac{1}{3} D_{a,6} \end{cases}$$

Equation (2) could be solved in a similar manner to resolve the $9 \times 9$ control points $P_{m,n}$, and therefore, a B-spline tensor product control hand surface captures the feature hand gesture is defined.

To further improve the performance, we may also incorporate the incremental technique [18] for rendering of the deforming hand surface.

## 4. Mapping of Object Vertices to the Hand Surface

To improve the mapping accuracy, we replace the original parallel projection method with the ray-projection method for mapping object vertices to the hand surface.

## 4.1. Ray-projection

Initially, a point called the centre of projection, $P_c$, is determined at a certain distance, $d_c$, from the palm centre (Figure 2). This distance will affect the level of deformation to an object model. The detail of the effect will be discussed in section 5.

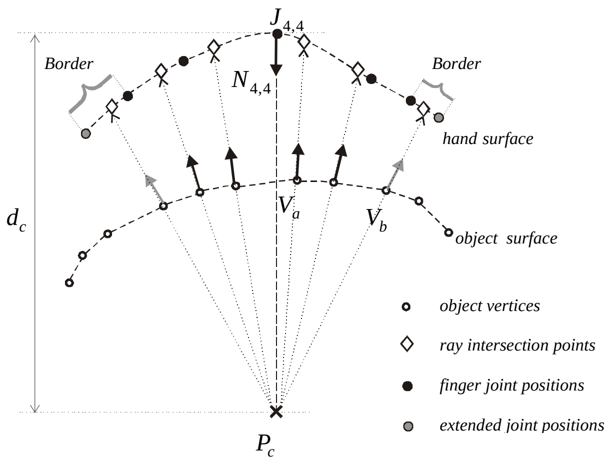


**Figure 2: Ray-projection.**

Ray-projection is referred to the projection of a ray directed from the centre of projection $P_c$, through each of the object vertices, $V_a$ where $a = 0,...,n-1$ of an object consisting of *n* vertices. This ray is projected onto the 3D triangulated hand model as shown in Figure 2. Notice the mapping is unique as long as there is no self-intersection, relative to the centre of projection, occurred in the hand gesture. The 3D triangulated hand model is constructed by triangulating the data points, $J_{i,j}$. The reader will recall that the hand is modelled by a tensor product surface. By joining neighbouring joint points in *U* and *V* dimensions, a tetrahedron mesh with $6\times6$ tetrahedrons is formed. Since it is not guaranteed that the four vertices of a tetrahedron are on a single plane, we further break each of the tetrahedrons into two triangles,

$$\begin{cases} tri_{i,j,0} : \Delta J_{i,j} J_{i,j+1} J_{i+1,j+1} \\ tri_{i,j,1} : \Delta J_{i,j} J_{i+1,j+1} J_{i+1,j} \end{cases}$$

Hence, a 3D triangulated hand model with $6\times6\times2$ triangles is formed (Figure 1) and each triangle, $tri_{i,j,\#}$, defines a plane, $T_{i,j,\#}$, as follows,

$$T_{i,j,\#} : \left(P \cdot N_{i,j,\#}\right) + d_{i,j,\#} = 0 \qquad (4)$$

where $i = 0,...,6,\ j = 0,...,6$ and $\# = 0,1$. $P$ is any point on the plane $T_{i,j,\#}$. $N_{i,j,\#}$ is the normal of the plane and $d_{i,j,\#}$ is the offset of the plane from the origin.

A ray, $R_a$, through a particular vertex, $V_a$, is defined as:

$$R_a : P = V_a + k \cdot \left[V_a - P_c\right]_I \qquad (5)$$

To test whether a ray, $R_a$, intersects the plane, $T_{i,j,\#}$, we need to solve the scalar variable, *k*, as follows,

$$k = -\frac{d_{i,j,\#} + N_{i,j,\#} \cdot V_a}{N_{i,j,\#} \cdot \left[V_a - P_c\right]_I} \qquad (6)$$

If $N_{i,j,\#} \cdot \left[V_a - P_c\right]_I$ equals zero, $R_a$ and $T_{i,j,\#}$ will be parallel to each other. Otherwise, they will intersect at some *k*. Followed by a transformation to the 2D local coordinate defined by $T_{i,j,\#}$, clipping [19] is applied to determine whether $R_a$ intersects within $tri_{i,j,\#}$. Once we identified which triangle and where within the triangle the ray intersects, the corresponding parametric coordinate of the hand surface can be determined using the barycentric coordinate method [14].

## 4.2. Two-Pass Projection

Notice that the 3D triangulated hand model consists of $6\times6\times2$ triangles and each defines a plane. Therefore, in the worst case, for an object consisting of *n* vertices, the mapping process could consist of a maximum of $6\times6\times2\times n$ projections, intersection and clipping tests. These processes are very computationally expensive. To minimize the number of projections and tests, we separate the process into two steps: *determination of intersection triangles* and *determination of intersection points.*

### 4.2.1 Determination of Intersection Triangles

The first step is to determine which triangle a ray intersects by ray-projecting each of the object vertices to a single plane $\pi$ and not directly onto each of the $6\times6\times2$ triangles of the 3D triangulated hand model as described previously. The plane $\pi$ is set up by including the centre of the palm, $J_{4,4}$, and setting it's normal the same as that of the centre of the palm, $N_{4,4}$. We then ray-project all $7\times7$ hand data points onto $\pi$ and a corresponding 2D base mesh is constructed (Figure 3). That is, the 2D base mesh is coplanar and is totally contained in the plane $\pi$.

To map the object vertices to the control hand surface, we ray-projected each of them, $V_a$, to $\pi$.

Followed by a transformation to the 2D local coordinate defined by $\pi$, clipping [19] is applied to determine which triangle of the 2D base mesh the vertex projected. Notice that the intersection of the triangle on the 2D base mesh indicates the intersection of the corresponding triangle on the 3D triangulated hand model. Therefore, by this initial projection, we can determine which triangle of the 3D triangulated hand model the vertex intersects in one projection and one clipping process.

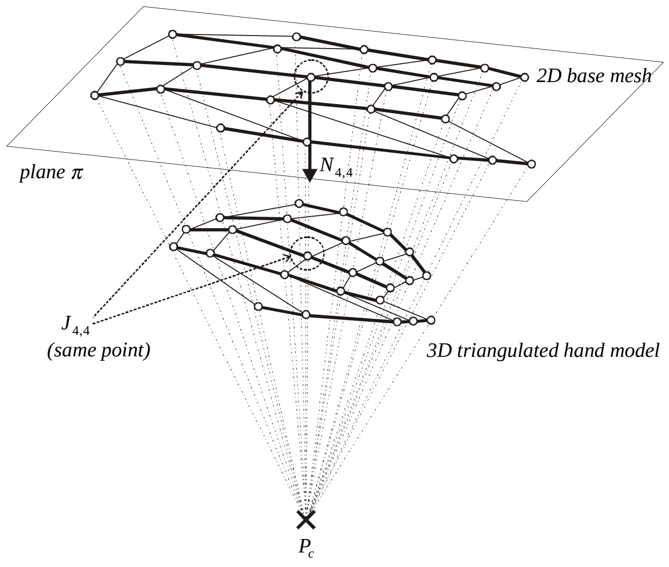


**Figure 3: Two-pass projection.**

### 4.2.2 Determination of Intersection Points

The second step is to determine the real intersection on the corresponding triangle of the 3D triangulated hand model the ray intersects. This is done by first solving $k$ in equation (6) and subsequently, applying a transformation to the 2D local coordinate defined by the plane, $T_{i,j,\#}$, containing the real triangle the ray intersects. Finally, the parametric coordinate $(u,v)$, which the vertex is mapped, can be evaluated by the barycentric coordinate method [14] and this is what we discussed in section 4.1

## 5. More realistic Sculpturing Mapping

In the previous section, we have proposed to replace the parallel projection method by the ray-projection method for mapping object vertices to the hand surface. Correspondingly, the shape modification method needs to be changed also.

### 5.1. Sculpturing Mapping

Recall our previous work [12], the position of an object vertex $V_a$ is deformed by:

$$V'_a = S'(u_a,v_a) + d_a \cdot N'(u_a,v_a)$$

where $N'(u_a,v_a)$ is the unit normal vector of the hand surface at $(u_a,v_a)$. $S'(u_a,v_a)$ is the 3D coordinate of the hand surface at $(u_a,v_a)$ and $d_a$ is the directional distance of the vertex $V_a$ to the base plane with the hand in flat state.

With ray-projection, $N'(u_a,v_a)$ is redefined by the unit vector from $S'(u_a,v_a)$ to $V'_a$ which is equal to the unit vector from $S'(u_a,v_a)$ to $P_c$. That is, $[V'_a - S'(u_a,v_a)]_I = [P_c - S'(u_a,v_a)]_I$ (Figure 4). Thus:

$$V'_a = S'(u_a,v_a) + d_a \cdot [P_c - S'(u_a,v_a)]_I \quad (7)$$

where $d_a = |V_a - S(u_a,v_a)|$ (8)

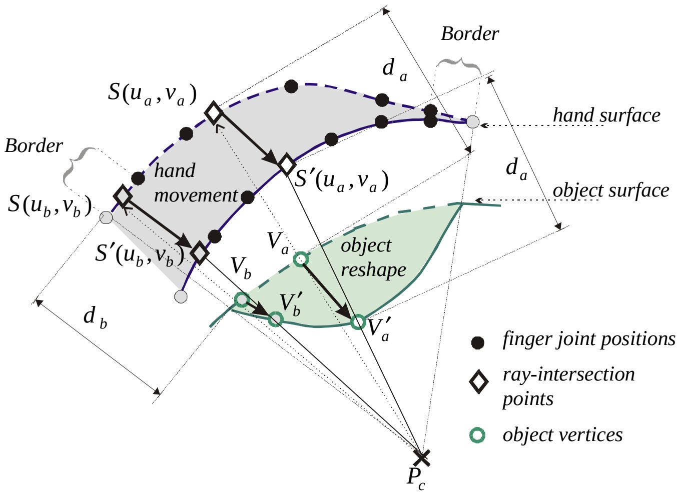


**Figure 4: Object Vertex Modification.**

Notice in border parts, the change is applied with weightings, *wt*, which progressively alleviate the modeling effect on the object vertices and prevent undesirable sudden change (Figure 4). At the edges, the weighting is proportional to the ratio of the parametric distance of a vertex from the border, either in *U*-direction or *V*-direction, to the width of the border. For the vertices mapped to the corner borders, the weightings in both directions are multiplied. An object vertex, $V_b$, mapped to the border is deformed as:

$$V_b' = wt * \left(S'(u_b, v_b) + d_b \cdot [P_c - S'(u_b, v_b)]_I\right) + (1 - wt) * V_b^0$$

where $V_b^0$ is the initial 3D coordinate of vertex $V_b$ before any modification is applied to the object.

## 5.2. Flexible Modification Region

With ray-projection, the modification region can be very flexible. It can be changed either by changing the gesture or by changing the location of the centre of projection. In our system, the modification region can be defined in three ways.

With parallel projection (Figure 5a), the modification region will be the same size as the hand surface. Note that the directional distance, $d_a$, is the magnitude of the vector from the hand surface to the object vertex (Equation 8) and the surface normal is replaced by the direction of the same vector. Therefore, $d_a \cdot N(u_a, v_a)$ equals $V_a - S(u_a, v_a)$. Equation (7) will then become,

$$V_a' = S'(u_a, v_a) + (V_a - S(u_a, v_a))$$

That is, $\Delta V_a = \Delta S(u_a, v_a)$ which implies that the change of the hand surface applies directly to the corresponding vertices of the object. This leads to a similar effect as touching the object.

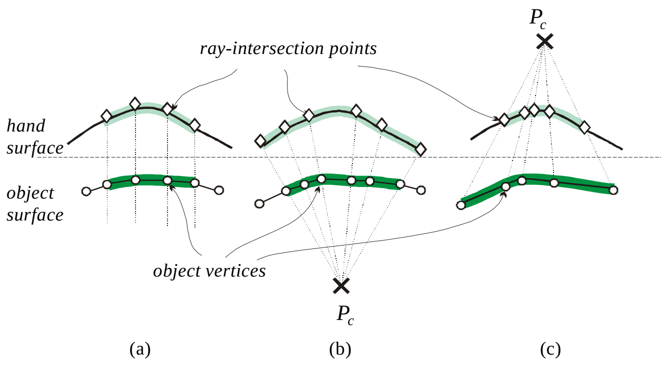


**Figure 5: Modification Region - (a) Parallel projection, (b) Reduced ray-projection, and (c) Enlarged ray-projection.**

When the centre of projection, $P_c$, is set in front of both the hand surface and the virtual object, a reduced region of modification could be resulted as shown in Figure 5b and is demonstrated in Plate 1. As the distance between the point of the hand surface where an object vertex mapped to and the palm centre increases, the magnitude of modification made to the object vertex reduces. In this case, it is resulted in a reduction modeling effect.

By setting the centre of projection, $P_c$, at the back of the hand surface, an enlarged region of modification could be resulted as shown in Figure 5c and is demonstrated in Plate 2. Conversely to the case above, the magnitude of change applied to the object vertex increases, as the distance between the point of the hand surface and the palm centre increases. This produces a magnifying modeling effect.

## 6. Results and Discussions

We have implemented the algorithm in C++ with Open Inventor and OpenGL on a SGI Indigo$^2$ workstation with a 200MHz MIPS 4400 CPU and the Extreme graphics accelerator. We have tested the system with five different models and some of the results are shown in plates 1 and 2. Plate 1 shows the sequence of grasping an apple model. Notice that the centre of projection is set in front of both the hand and the object, thus producing a reduction effect. Plate 2 demonstrates the deformation of a face model. The centre of projection is set at the back of the hand and thus producing a magnifying effect. In each plate, there is a cluster of coloured points indicating the corresponding mapping of object vertices to the hand surface. The white points indicate vertices mapped to the border. The cluster is used to reveal the control of different regions of the hand surface that can be applied to the object model. Plates 1a and 2a show the models in their initial shape. Plates 1b, 1c, 2b and 2c show the sculpturing effect applied to the models while the hand is changing its shape. The grasping effect shown in Plate 1c is an enhanced effect upon our previous work.

| Model | No. of Vertices | Stage 1: Mapping of object vertices to hand surface | | Stage 2: Modification of Virtual object | |
|---|---|---|---|---|---|
| | | Total Time | Average time (per vertex) | Total Time | Average Time (per vertex) |
| Button | 701 | 0.35s | 0.50ms | 0.35s | 0.050ms |
| Teapot (high res.) | 2081 | 1.00s | 0.48ms | 0.11s | 0.055ms |
| Teapot (low res.) | 529 | 0.26s | 0.49ms | 0.03s | 0.055ms |
| Apple | 867 | 0.45s | 0.52ms | 0.05s | 0.050ms |
| Face | 2278 | 0.86s | 0.38ms | 0.13s | 0.055ms |

**Table 1: Performance of the new method.**

To evaluate the performance of our system, we measured the time taken in the two main stages during sculpturing as shown in table 1. In stage 1, object vertices are mapped to the hand surface. It takes approximately 0.5ms per vertex. In stage 2, the vertices are updated according to the change of the user's hand gesture. It takes approximate 0.05ms per vertex. In all experiments, the time for fitting the tensor product surface to the data points is only 0.005s to 0.01s. Thus it is concluded that the new method is very efficient and is applicable to VR sculpturing.

## 7. Conclusions

In this paper, we have presented the enhancement to our previous work by introducing the efficient B-spline tensor product surface to model the hand surface. The mapping of the vertices of the object model to the hand surface is refined by ray-projection. Together with the rectification of the distance parameter, the new method can successfully alleviates undesirable self-intersections of the object model. In addition, with the flexible projection mapping, the algorithm produces different levels of effect, which is a valuable feature for virtual sculpturing. As demonstrated by our experimental results, the new method is very efficient and the deformation operation is intuitive for use in VR application.

## Acknowledgments

We would like to thank K.H. Chan, Frederick Li, George Fung of the Computer Graphics & Media Laboratory and Boris Leung for their suggestions in the project and the preparation of the paper. We are also grateful to the anonymous reviewers for their valuable comments.

## References

[1] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled Simulation in Virtual reality with MR Toolkit. *ACM Transactions on Information Systems*, **11**(3):135-142, May 1992.

[2] *CyberGlove™ User's Manual*, Copyright © 1991-1994 Virtual Technologies.

[3] D. Kurmann and M. Engeli. Modelling Virtual Space in Architecture. *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pp. 77-82, July 1996.

[4] F. Li, R. Lau, and M. Green. Interactive Rendering of Deformating NURBS Surfaces. *EUROGRAPHICS'97*, **16**(3):C-47-56, September 1997.

[5] G. Burdea, J. Zhuang, E. Roskos, D. Silver, and N. Langrana. A Portable Dextrous Master with Force Feedback. *Presence: Teleoperators and Virtual Environments*, **1**(1):18-28, 1992.

[6] G. Engeln-Mullges and F. Uhlig. *Numerical Algorithms with C*. Springer, pp. 89-92, 1996.

[7] G. Farin. Curves and Surfaces in Computer Aided Geometric Design, 4th ed. Academic Press, 1997.

[8] H. Lamousin and W. Waggenspack. NURBS-Based Free-Form Deformations. *IEEE Computer Graphics and Applications*, pp. 59-65, November 1994.

[9] H. Spoelder and F. Ulllings. Two-Dimensional Clipping: A Vector-based Approach. *Graphics GEMS*. Academic Press, pp. 121-128, 1990.

[10] J. Liang and M. Green. Geometric modeling using six degrees of freedom input devices. *Proceedings of 3rd International Conference on CAD and Computer Graphics*, pp. 217-222, August 1993.

[11] K. Kameyama. Virtual Clay Modeling System. *ACM Symposium on Virtual Reality Software and Technology*, pp. 197-200, September 1997.

[12] L. Ma, R. Lau, J. Feng, Q. Peng, and J. Wong. Surface Deformation Using the Sensor Glove. *ACM Symposium on Virtual Reality Software and Technology*, pp. 189-196, September 1997.

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

[14] S. Coquillart. Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling. *ACM Computer Graphics* (*SIGGRAPH'90*), **24**(4):187-193, August 1990.

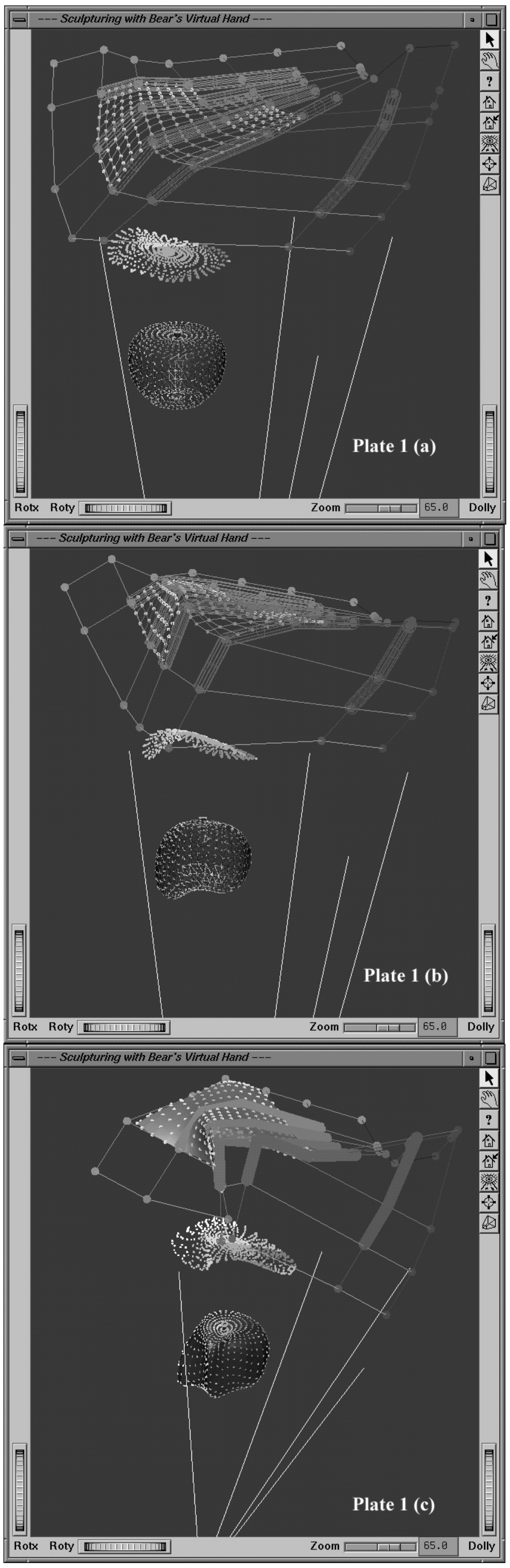
[15] T. A. Galyean and J. F. Hughes. Sculpting: An interactive volumetric modeling technique, *ACM Computer Graphics* (*SIGGRAPH'91*), **25**(4):267-274, 1991.

[16] T. Murakami and N. Nakejima. Direct and Intuitive Input Device for 3D shape Deformation. *Proceedings of ACM CHI'94*, pp. 465-470, 1994.

[17] T. Sederberg and S. Parry. Free Form Deformation. ACM Computer Graphics (SIGGRAPH'86), **20**(4):151-160, 1986.

[18] VitualHand™ Software Library Reference Manual, Copyright © 1994 Virtual Technologies.

[19] W. Hsu, J. Hughes, and H. Kaufman. Direct Manipulation of Free-Form Deformations. *ACM Computer Graphics* (*SIGGRAPH'92*), **26**(2):177-184, July 1992.


--- Sculpturing with Bear's Virtual Hand ---
Plate 1 (a)
Rotx Roty Zoom 65.0 Dolly
--- Sculpturing with Bear's Virtual Hand ---
Plate 1 (b)
Rotx Roty Zoom 65.0 Dolly
--- Sculpturing with Bear's Virtual Hand ---
Plate 1 (c)
Rotx Roty Zoom 65.0 Dolly

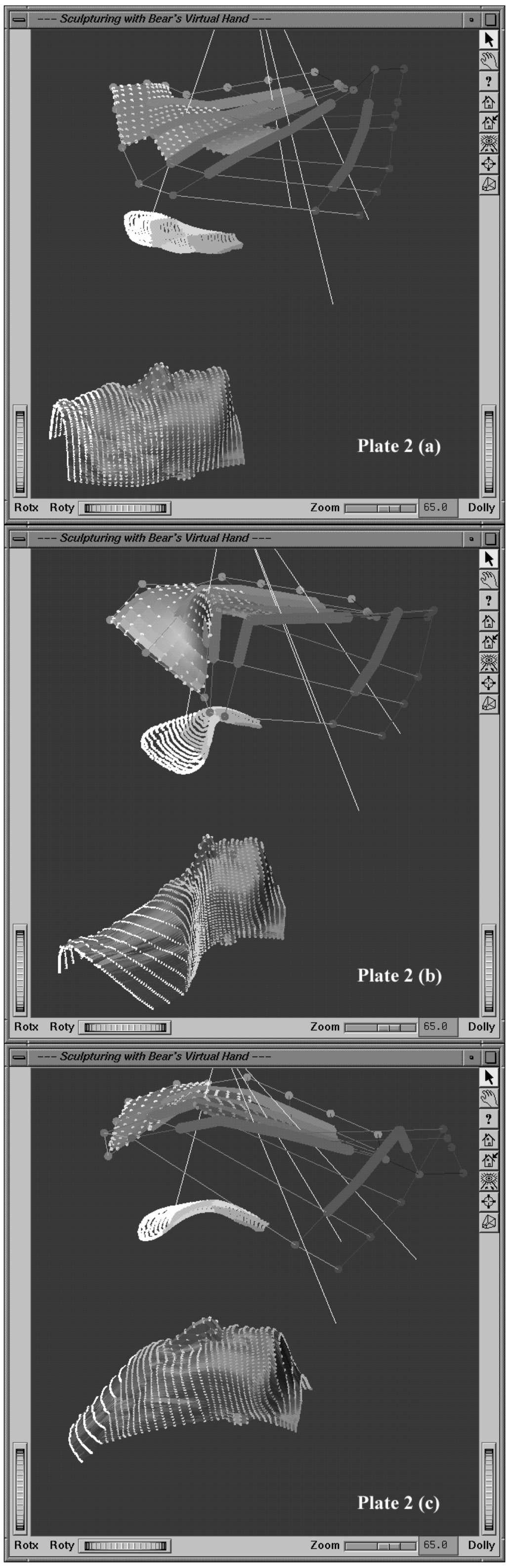

--- Sculpturing with Bear's Virtual Hand ---
Plate 2 (a)
Rotx Roty Zoom 65.0 Dolly
--- Sculpturing with Bear's Virtual Hand ---
Plate 2 (b)
Rotx Roty Zoom 65.0 Dolly
--- Sculpturing with Bear's Virtual Hand ---
Plate 2 (c)
Rotx Roty Zoom 65.0 Dolly

**Plate 1: Apple Model** **Plate 2: Face Model**

**Virtual 3D Sculpturing with a Parametric Hand Surface**