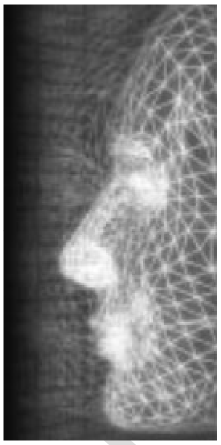
COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2005; **16:** 1–13
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cav.071

Efficient rendering of deformable objects for real-time applications



*By Gary K. L. Cheung, Rynson W. H. Lau and Frederick W. B. Li**

Deformable objects can be used to model soft objects such as clothing, human faces and animal characters. They are important as they can improve the realism of the applications. However, most existing hardware accelerators cannot render deformable objects directly. A tessellation process is often used to convert a deformable object into polygons so that the hardware graphics accelerator may render them. Unfortunately, this tessellation process is computationally very expensive. While the object is deforming, the tessellation process needs to be performed repeatedly to convert the deforming objects into polygons. As a result, deformable objects are seldom used in real-time applications such as virtual environments and computer games. Since trimmed NURBS surfaces are often used to represent deformable objects, in this paper we present an efficient method for incremental rendering of deformable trimmed NURBS surfaces. A trimmed NURBS surface typically deforms through the deformation of the trimmed NURBS surface and/or the trimming curve. Our method handles both trimmed surface deformation as well as trimming curve deformation. Experimental results show that our method performs significantly faster than the method used in OpenGL and can be used in real-time applications, such as computer games. Copyright © 2005 John Wiley & Sons, Ltd.

Received: 1 March 2004; Accepted: 1 October 2004

KEY WORDS: deformable object handling; real-time rendering; incremental rendering; trimmed NURBS surfaces

Introduction

Deformable objects have been considered as important to applications such as virtual environments and computer games, as they can help improve the realism of these applications. They can be used to model soft objects, such as clothing, facial expression, and human and animal characters. In particular, non-uniform rational B-splines (NURBS)[1,2] are often employed to represent such objects as they can produce a variety of shapes simply by manipulating their control points and weights. However, these NURBS surfaces are seldom used in interactive applications that demand real-time rendering performance because of their high rendering cost. Many studies have been carried out to address this problem. Most of the methods developed are based on tessellation.[3–8] This tessellation process subdivides the NURBS surfaces into polygons so that the hardware graphics accelerator may render the polygons in real time. However, this tessellation process is computationally very expensive. As a NURBS surface is deforming, this process must be executed in each frame to reflect the change of the object shape. Since in many real-time applications such as computer games we may want to have many deformable objects in the environment, it would be difficult to render these objects in real time using existing rendering methods.

Earlier, we proposed an efficient method for rendering deforming NURBS surfaces.[9] The method precomputes a polygon model and a set of deformation coefficients for each deformable NURBS surface. During run time, it incrementally updates the pre-computed polygon model of each deforming surface and progressively refines the resolution of the model according to the change in the surface curvature. We have shown that

*Correspondence to: F. W. B. Li, Department of Computing, Hong Kong Polytechnic University, Hong Kong.
E-mail: csbor@comp.pdyu.edu.uk

Contract/grant sponsor: City University of Hong Kong; contract/grant numbers: 7001391; 7100264.

this method is much more efficient than existing methods. Recently, we have applied this method to develop a distributed virtual sculpting system.[10,11] We have also extended the method to cover various types of deformable parametric free-form surfaces.[12] However, we have learnt from our experience that in order to represent real objects, such as human faces with eyes and mouths, many NURBS surfaces need to be used. This is because our method can only support regular NURBS surfaces. To represent an object with holes, we need to combine many regular NURBS surfaces to model a single object. To overcome this limitation, our objective in this work is to extend the method to support trimming,[13] which is a technique to allow arbitrary regions of a NURBS surface to be cut out, resulting in a non-regular NURBS surface.

In this paper, we describe a method to extend our NURBS rendering method[9] to support trimming. The rest of the paper is outlined as follows. The next section provides a brief survey on related work. The third section describes how we handle the deformable trimmed NURBS surfaces. The fourth section presents our method for tessellating trimmed NURBS surfaces. The fifth section describes how to incrementally update trimming curves and trimmed NURBS surfaces as they deform. The sixth section describes how we may perform direct surface deformation. The seventh section summarizes our method. The eight section demonstrates the performance of our method through some experiments. The final section briefly concludes the paper.

Related Work

Efficient rendering of trimmed NURBS surfaces has been a challenging research area for decades. There are many methods proposed for tessellating trimmed NURBS surfaces into polygons for rendering. Lien *et al.*[14] propose an adaptive forward differencing method for rendering NURBS surfaces, which was later extended by Shantz and Chang[15] to evaluate trimmed NURBS surfaces incrementally to produce polygon models for rendering. As the method may adjust the forward differencing step adaptively, it could optimize the number of surface points generated according to the surface curvature or other approximation criteria. Chang *et al.*[16] further enhance the method to use integer-based calculation. Rockwood *et al.*[8] propose an alternative method to accelerate the tessellation process. It first converts the surface to Bézier patches with knot insertion. For those trimmed patches, it further subdivides them into a list of *uv*-monotone patches. Each patch is then tessellated into polygons by the coving and tiling process. A variant of this method has been implemented in the OpenGL library.

Abi-Ezzi and Subramaniam[3] tessellate trimmed NURBS surfaces in a similar way to Rockwood *et al.*,[8] but it further minimizes the number of patches needed to be tessellated by culling out the invisible patches dynamically during run time. This culling process basically uses a pre-processed cone-of-normal structure[17] to reduce the complex normal calculation in run-time. Kumar *et al.*[6] improve the performance of the tessellation process by avoiding the operation of subdividing Bézier patches into *uv*-monotone patches. Instead, their method directly tessellates the Bézier patches into polygons for rendering. However, to allow fast back-patch culling, it needs to pre-compute the pseudo-normal surfaces for all Bézier patches. Kumar *et al.*[7] enhance their earlier method[6] by constructing the super-surfaces on the Bézier patches to allow a further reduction in the number of polygons in the resulting polygon model.

Unfortunately, all the methods discussed above cannot handle deforming trimmed NURBS surfaces efficiently. While the tessellation process has a high computational cost, it has to be performed repeatedly in every frame as the surfaces are deforming. On the other hand, if the trimming curve is undergoing deformation, methods adopting the *uv*-monotone approach need to regenerate a new set of *uv*-monotone patches before the tessellation can be performed.

Recently, Kahlesz *et al.*[5] proposed an adaptive tessellation method, which recursively subdivides a surface into a quad-tree hierarchy according to some approximation criteria. The location of the trimming polylines on the surface is then determined. If a leaf quad-tree node is found to be untrimmed, it will be selected as an output polygon for the resulting polygon model. Otherwise, the node is further subdivided for further processing or tessellated by constrained Delaunay triangulation to generate the output polygons. This method was further enhanced by building a seam graph structure on the tessellated trimmed NURBS surface for multi-resolution modelling.[4] A seam graph structure is actually a progressive mesh-like[18] structure. It allows the application to select an appropriate resolution of the tessellated polygon model for rendering. However, the progressive mesh and the seam graph structure are expensive to construct and these data structures will need to be reconstructed whenever the surface or the trimming curve deforms. Hence, this method, like most of the other methods, is not suitable for interactive rendering of deformable trimmed NURBS surfaces.

Handling of Deformable Trimmed NURBS Surfaces

In our earlier work, we developed a technique for efficient rendering of deformable NURBS surfaces.[9,19] 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. To handle trimming curves efficiently, we may also model them in a similar way.

Handling of the NURBS Surfaces

A NURBS surface is a rational generalization of the tensor product non-rational B-spline surface defined as follows:

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} P_{i,j} w_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{s=0}^{n}\sum_{t=0}^{m} w_{s,t} N_{s,p}(u) N_{t,q}(v)} \tag{1}$$

where $w_{i,j}$ are the weights and $P_{i,j}$ form a control net. $N_{i,p}(u)$ and $N_{j,q}(v)$ are the B-spline basis functions.

The polygon model of a surface can be obtained by evaluating the surface equation with some discrete parametric values. If a control point $P_{i,j}$ is moved to $\overline{P_{i,j}}$ with a displacement vector $\vec{V}_{i,j} = \overline{P_{i,j}} - P_{i,j}$, the incremental difference between the two polygon models of the surface before and after the control point movement is

$$\overline{S)(u,\, v)} - S(u,\, v) = \frac{\left(\overline{P_{i,j}} - P_{i,j}\right) w_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{s=0}^{n}\sum_{t=0}^{m} w_{s,t} N_{s,p}(u) N_{t,q}(v)} = \alpha_{i,j}\vec{V}_{i,j} \tag{2}$$

where $S(u,v)$ and $\overline{S(u,v)}$ are the polygon models of the surface before and after the control point movement, respectively. $\alpha_{i,j}$ are referred to as the *deformation coefficients* and defined as follows:

$$\alpha_{i,j}(u,v) = \frac{w_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{s=0}^{n}\sum_{t=0}^{m} w_{s,t} N_{s,p}(u) N_{t,q}(v)} \tag{3}$$

Each deformation coefficient $\alpha_{i,j}$ is a constant for each particular pair of (u,v). Hence, if the resolution of the polygon model does not change during the deformation, we may pre-compute the deformation coefficients and update the polygon model incrementally as shown in equation (2).

However, when a surface deforms, its curvature is likely changed and we need 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.[20] Each Bézier patch is then subdivided into a polygon model, which is maintained in a quad-tree hierarchy, by applying the de Casteljau subdivision formula.[21] to the Bernstein polynomials in both u and v directions. For example, in u, we have

$$P_i^r(u) = (1-u)\frac{w_i^{r-1}}{w_i^r} P_i^{r-1}(u) + u\frac{w_{i+1}^{r-1}}{w_i^r} P_{i+1}^{r-1}(u) \tag{4}$$

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

The difference in equation (4) before and after the deformation can be simplified to obtain a de Casteljau formula as follows:

$$\alpha_i^r(u) = (1-u)\alpha_i^{r-1}(u) + u\alpha_{i+1}^{r-1}(u) \tag{5}$$

for $r=1,\ldots,n$, $i=0,\ldots,n-r$. Equation (5) 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 have implemented this based on Horner's formula, of average complexity $O(n)$ as opposed to $O(n^2)$ when based on de Casteljau's formula.

Handling of theTrimming Loops

A trimmed NURBS surface is defined by a NURBS surface together with a set of trimming loops. Each trimming loop consists of a set of NURBS curves, which are defined over the parametric space of the NURBS surface. A NURBS curve, $C(t)$, may be defined as

$$C(t) = \frac{\sum_{i=0}^{n} P_i w_i N_{i,p}(t)}{\sum_{s=0}^{n} w_s N_{s,p}(t)} = \sum_{i=0}^{n} \beta_i(t) P_i \tag{6}$$

where P_i denotes the control points of the NURBS curve. $\beta_i(t)$ are the *deformation coefficients* of the NURBS curve and are defined as

$$\beta_i(t) = \frac{w_i N_{i,p}(t)}{\sum_{s=0}^{n} w_s N_{s,p}(t)} \tag{7}$$

To trim a NURBS surface, we subdivide the NURBS curves by the node boundaries of the polygon hierarchy representing the NURBS surface. This produces a list of *polylines* for each quad-tree node in the polygon hierarchy that are inscribed in the node.

Tessellation of Trimmed Nodes

To render a trimmed NURBS surface, we select the appropriate quad-tree nodes in the polygon hierarchy according to the dynamically changing viewing parameters, such as the distance of a node from the viewer and its angular distance from the viewer's line of sight.[22] The selected quad-tree nodes are then tessellated according to their types. There are three types of nodes: *visible non-trimmed node*, *invisible non-trimmed node* and *trimmed node*. They are processed as follows:

- A visible non-trimmed node is a node lying within the interior of the NURBS surface and does not contain any trimming curve segments. For this type of node, we just need to split it into two triangles along the diagonal of the node.
- An invisible non-trimmed node is a node lying completely outside the NURBS surface, i.e., trimmed out by a trimming curve, and does not contain any trimming curve segments. For this type of node, we may just ignore it and do not render it.
- A trimmed node is a node containing at least one trimming curve segment, i.e., with part of the node trimmed out by the trimming curve(s). For this type of node, we further classify it into one of the three types: *convex trimmed node*, *monotonic chain trimmed node* and *complex trimmed node*. They are described in the following subsections.

Convex Trimmed Nodes

A convex trimmed node is a node with a convex trimmed region. To identify such a node, we make use of the strong convex hull property of the NURBS curve definition. 'Strong convex hull' refers to the convex hull formed by the control points of the NURBS curve, where the whole curve is always contained inside the convex

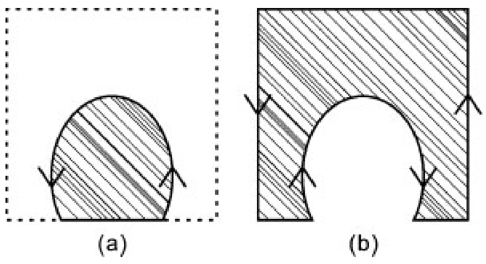


Figure 1. Trimming out of (a) exterior region and (b) interior region.

hull. We perform an angle test on the control points of the curve segment to verify the convexity of the trimmed region in the node. Once a convex trimmed node is identified, there are two possibilities, as shown in Figure 1. Either the exterior or the interior region of the trimming curve segment is trimmed out as shown in Figure 1(a) and Figure 1(b), respectively. The former represents a *convex trimmed node* and can be tessellated by a simple triangulation algorithm. The latter may represent either a *monotonic chain trimmed node* or a *complex trimmed node*.

Note that once a node containing a trimming curve segment is identified to form a convex trimmed node, there is no need to perform any node reclassification when there is a resolution refinement on the trimming curve. This is because the convex hull property and the variation diminishing property[1] are independent of the resolution of the curve.

Monotonic Chain Trimmed Nodes

A polyline segment is a monotonic chain with respect to axis L if the polylines of the segment have at most two intersections to any L' perpendicular to L. In our method, the monotonic chain is respected[Q1] to the u Q1

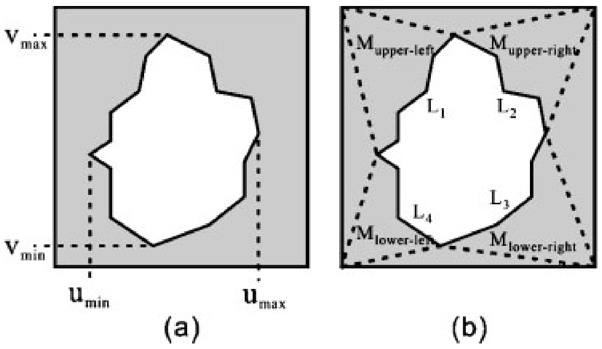


Figure 2. Monotonic chain trimmed node: (a) maxima and minima, and (b) partitioning of the node into four regions according to the maxima and minima.

and the v axes in the parametric space as uv-monotonic polylines. A monotonic chain trimmed node is then a combination of the uv-monotonic polylines and the corners of the trimmed node as shown in Figure 2. By tessellating these monotonic regions as a whole, we can both reduce the number of node subdivisions and minimize the number of resulting polygons. As shown in Figure 2(a), such a case exists when there are u-monotonic and v-monotonic polylines inscribed in a node, in which the monotonic polylines are connected at their maxima, u_{max} and v_{max}, and minima, u_{min} and v_{min}. In other words, there are four uv-monotonic polylines in the node. Referring to Figure 2(b), these uv-monotonic polylines have the following properties.

In u direction:

$$\forall x\{x : (u_{min} \le u_x \le u_{x+1} \le u_{max}) \wedge (u_x, u_{x+1}) \in L_{upper}\}$$

$$\forall x\{x : (u_{max} \ge u_x \ge u_{x+1} \ge u_{min}) \wedge (u_x, u_{x+1}) \in L_{lower}\}$$

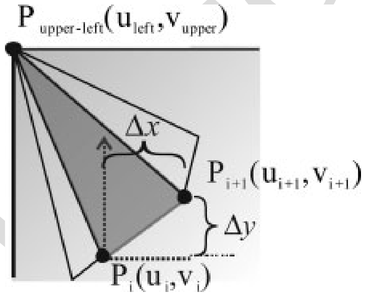
In v direction:

$$\forall y\{y : (v_{min} \le v_y \le v_{y+1} \le v_{max}) \wedge (v_y, v_{y+1}) \in L_{left}\}$$

$$\forall y\{y : (v_{max} \ge v_y \ge v_{y+1} \ge v_{min}) \wedge (v_y, v_{y+1}) \in L_{right}\}$$

where L_{upper} and L_{lower} denote the upper uv-monotonic polylines ($L_1 \cup L_2$) and the lower uv-monotonic polylines ($L_3 \cup L_4$), respectively. These polylines are partitioned by {u_{max}, u_{min}}. L_{left} and L_{right} denote the left uv-monotonic polylines ($L_1 \cup L_4$) and the right uv-monotonic polylines ($L_2 \cup L_3$), respectively. These polylines are partitioned by {v_{max}, v_{min}}.

All four uv-monotonic regions share the same properties, except that they have different orientations. To show how we tessellate the node, we consider the upper-left uv-monotonic region $M_{upper\text{-}left}$ as shown in Figure 3. Since a uv-monotonic region is convex in nature, the simplest way to tessellate the region is by joining the corner vertex of the region to each vertex of the monotonic polylines within the region. As an example, the corner vertex of $M_{upper\text{-}left}$ as shown in Figure 3 is $P_{upper\text{-}left}$, which has the coordinate (u_{left}, u_{upper}). Since the uv-monotonic polylines are inscribed in the node, we can have $u_{left} \le u_{min}$ and $u_{upper} \ge u_{min}$. The tessellation is done by adding lines from the corner vertex $P_{upper\text{-}left}$ to each vertex P_i on the uv-monotonic polylines of $M_{upper\text{-}left}$.



$P_{upper\text{-}left}(u_{left}, v_{upper})$
Δx
$P_{i+1}(u_{i+1}, v_{i+1})$
Δy
$P_i(u_i, v_i)$

Figure 3. Tessellation of the upper-left uv-monotonic region.

To show that this tessellation process work correctly, i.e., the lines added would not cross each other, we consider two consecutive sample points of a trimming curve segment, i.e., the two endpoints of a polyline. Refer to Figure 3 as an example. The two endpoints are P_i and P_{i+1}, with P_{i+1} being the next point of P_i. The two points form a triangle with $P_{upper\text{-}left}$, with coordinates (u_{left}, u_{upper}), (u_i, v_i) and (u_{i+1}, v_{i+1}). If the orientation of these three vertices is always anticlockwise, their determinant should then be positive. To evaluate the determinant, we express P_{i+1} as $(\Delta x + u_i, \Delta y + v_i)$, where $(\Delta x, \Delta y)$ is the offset from P_i to P_{i+1}. Based on the monotone property, Δx and Δy are always positive in $M_{upper\text{-}left}$. The determinant of the three vertices is calculated as follows:

$$\begin{aligned}
&\begin{vmatrix} u_i - u_{left} & (\Delta x + u_i) - u_{left} \\ v_i - v_{upper} & (\Delta y + v_i) - v_{upper} \end{vmatrix} \\
&= [\Delta y(u_i - u_{left}) + (u_i - u_{left})(v_i - v_{upper})] - \\
&\quad [\Delta x(v_i - v_{upper}) + (u_i - u_{left})(v_i - v_{upper})] \\
&= \Delta y(u_i - u_{left}) - \Delta x(v_i - v_{upper})
\end{aligned} \tag{8}$$

As $u_{left} \le u_{min}$ and $\Delta y \ge 0$, the first part, $\Delta y(u_i - u_{left})$, of equation (8) must be positive. In addition, as $v_{upper} \ge v_{min}$ and $\Delta x \ge 0$, the second part, $-\Delta x (v_i - v_{upper})$, of equation (8) must also be positive. Therefore, the result of equation (8) must always be positive.

By combining the four uv-monotonic regions together, the result of the tessellation process may become as shown in Figure 4. Since this tessellation process requires the identification of the two pairs of maxima and minima, the complexity is $O(n)$ bounded. In addition, the process traverses each vertex of the polylines at most once, which is also $O(n)$ bounded. Note that for a monotonic trimmed node, if the resolution of the trimming curve segment is increased, we only need to incrementally check the newly inserted vertices for the maxima and minima. On the other hand, if the resolution of the trimming curve segment is decreased,

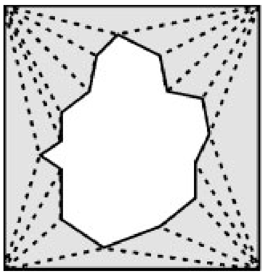


Figure 4. The tessellation of a monotonic chain trimmed node.

we may need to identify the maxima and minima again.

Complex Trimmed Nodes

When a node is identified as neither a convex trimmed node nor a monotonic chain trimmed node, it is considered as a complex trimmed node. Normally, if a node contains a highly irregular trimming curve, it is likely a complex trimmed node. To handle this kind of node, we further subdivide each complex trimmed node, which involves dividing the trimming curve by the boundaries of individual child nodes. Hence, this subdivision process essentially reduces the irregularity of the trimming curve and allows more and more subnodes to be classified as convex trimmed nodes or monotonic chain trimmed nodes for triangulation.

Since each child node contains on average about one-quarter of the original trimming curve, such reduction in irregularity is very effective. According to our experiments, more than 80% of the complex trimmed nodes will be subdivided into convex trimmed nodes or monotonic chain trimmed nodes during the first subdivision. Only 20% of the nodes need further subdivisions. From our experience, most nodes require only one or two levels of subdivision to partition a complex trimmed node into convex trimmed nodes and/or monotonic chain trimmed nodes.

Deformation of Trimming Curves and Trimmed Surfaces

A critical development of this project is that the new method should support real-time deformation of both the trimming curves and the NURBS surfaces. We note that the deformation of a trimming curve does not affect the topology or the shape of the trimmed NURBS surface. On the other hand, the deformation of the trimmed NURBS surface will affect the shape of the surface itself, but does not affect the topology of the trimming curve. As such, both types of deformation are only loosely related to each other, and we can handle each of them separately.

Trimming Curve Deformation

Usually, when a trimming curve is being deformed, only part of the NURBS surface is affected. It will be expensive to perform the retriangulation of the trimming curve against the polygon hierarchy of the NURBS surface in every frame while the trimming curve is deforming. Based on the local modification property of NURBS curves, any deformation driven by modifying a control point P_i of a NURBS curve will only affect the curve segment within the parametric range $[t_i, t_{i+p})$, where p is the order of the trimming curve. Hence, we may limit the update operation to within this trimming curve segment defined by t_i and t_{i+p}. To allow an efficient update of the corresponding polylines of the curve segment, we may simply check each of the polylines to see if its two endpoints satisfy the following condition:

$$(t_i \leq t_{start} \wedge t_{end} \leq t_{i+p})$$

Only those polylines satisfying this condition are affected by the deformation and hence need to be updated. Once we have identified the affected polylines, we would update the parametric positions of their vertices. We then perform resolution refinement and compute the new sets of deformation coefficients for the newly inserted vertices. The updated polylines are then remapped to the NURBS surface for retessellation.

Trimmed Surface Deformation

When a trimmed NURBS surface deforms, the curvature of the surface may be affected and the curvature of the trimming curves within the deformed region of the surface may also be affected. We handle this in a way somewhat similar to how we handle the deformation of the trimming curve. However, the scope of the update is relatively smaller. First, we update the affected region of the NURBS surface. Second, we update the vertex positions of the affected polylines on the NURBS surface. Third, we perform resolution refinement on the affected trimming curve segment. Finally, we retessellate the resulting polylines with the corresponding nodes.

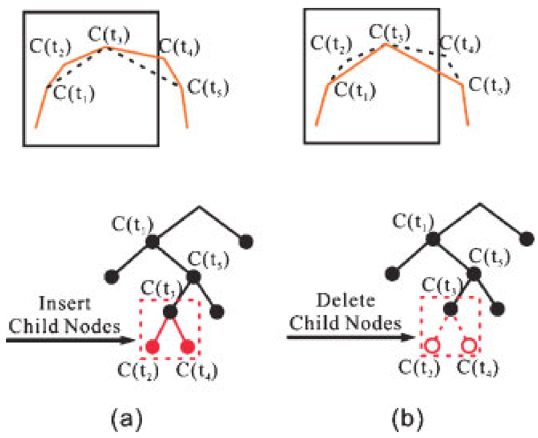


Figure 5. Resolution refinement of a trimming curve segment: (a) resolution increase and (b) resolution decrease.

Figure 5 illustrates two examples of resolution refinement of a trimming curve segment C with reference to a node in the polygon hierarchy, where $C(t_i)$ denotes a vertex of the polylines representing C. In Figure 5(a), we assume that the polylines can no longer approximate C owing to the increase in the surface curvature. Hence, we need to increase the resolution of C by inserting vertices $C(t_2)$ and $C(t_4)$ to the polylines. On the other hand, if the resolution of C is found to be too high due to the decrease in the surface curvature as shown in Figure 5(b), we may delete vertices $C(t_2)$ and $C(t_4)$ to decrease the resolution of C. By observation, an update to $C(t_3)$ will affect the polylines between $C(t_1)$ to $C(t_5)$. In fact, $C(t_1)$ and $C(t_5)$ are the previous vertex and the next vertex to $C(t_3)$, respectively. Hence, $C(t_1)$ and $C(t_5)$ may be considered as the updated range of $C(t_3)$. Generally speaking, whether a resolution refinement process involves inserting or deleting vertices, the update range always falls between $(\max\{t : t < t_i\}, t_i)$ and $(t_i, \max\{t : t_i < t\})$.

Note that the change in curvature of the trimming curve segment due to the deformation of the NURBS surface is usually very small. Hence, only a very small number of vertices may need to be inserted or deleted from the polylines in order to maintain a good polygonal representation of the trimmed surface. Figure 6 shows a trimmed NURBS surface before and after deformation. We can see that both the resolution of the surface itself and the resolution of the trimming curve are refined according to the change of the surface curvature. Note that any potential crack appearing at the T-junction point on the polygonal representation could be resolved by edge flattening.[9]

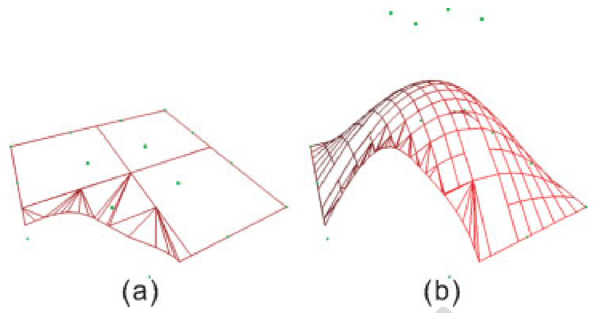


Figure 6. Resolution refinement of a trimmed NURBS surface: (a) before deformation and (b) after deformation.

Direct Deformation

In our method, the deformation of a trimmed NURBS surface is performed by repositioning the control points of the surface. However, in some applications, we may need to perform the deformation directly by manipulating the surface points. With our method, we will need to convert the deformation of the surface points into the deformation of the control points. For example, consider when a baseball is being battered. The region of the baseball hit by the bat will be deformed according to the force of the bat acting on the region. To model this situation, we would need to convert the movement of the region hit by the bat into appropriate control point movement. Since there can be many solutions to this problem, it may be difficult to find an optimal solution in real time.

To address the problem, we may adopt Piegl's shape modification technique[2] here, which provides a deterministic and optimal way to map the change of surface points into the change of control points. Suppose that we need to move a surface point $P = S(u', v')$ by a distance d in the direction $\vec{V}$ to $\bar{P}$. We then map this movement into the corresponding movement of a control point $P_{k,h}$. The new position of $P_{k,h}$, i.e. $\overline{P_{k,h}}$, is expressed as $\overline{P_{k,h}} = P_{k,h} - \gamma\vec{V}$, where γ is a scalar value and is calculated as follows.

Consider

$$|\bar{P} - P| = d = \gamma|\vec{V}|R_{k,h}(u', v') \qquad (9)$$

where $R_{k,h}(u', v')$ is the basis function of control point $P_{k,h}$ with the fixed parameter (u', v') substituted. We may obtain

$$\gamma = \frac{d}{|\vec{V}|R_{k,h}(u', v')} \qquad (10)$$

The control point $P_{k,h}$ with basis function $R_{k,h}(u, v)$ whose maximum value lies closest to (u', v') will be selected as the most appropriate control point to produce an equivalent movement made by surface point P. To determine $P_{k,h}$ efficiently, instead of computing the maximum value of $R_{k,h}(u, v)$, k and h are chosen such that

$$|u' - s_k| = \min |u' - s_i| \quad \text{and} \quad |v' - t_h| = \min |v' - t_j| \tag{11}$$

where s_i and t_j are the *node values* along u and v directions, respectively, in the parametric domain of the surface. They are defined as

$$s_i = \frac{1}{n}\sum_{r=1}^{n} u_{i+r} \quad \text{and} \quad t_j = \frac{1}{m}\sum_{r=1}^{m} v_{j+r} \tag{12}$$

where u_{i+r} and v_{j+r} are the u and v knot values, respectively. Note that the node values are the average of knots which are interior to the domain, where $R_{k,h}(u, v)$ is non-zero.

To ensure that a maximum translation will occur at P, a further operation needs to be carried out. This is done by inserting new knots $\hat{u}$ and $\hat{v}$ along u and v directions, respectively, to force s_{k+1} and t_{h+1} to have the values u' and v', respectively. Obviously, the two new knots are defined as

$$\hat{u} = nu' - \sum_{r=2}^{n} u_{k+r} \quad \text{and} \quad \hat{v} = mv' - \sum_{r=2}^{m} v_{h+r} \tag{13}$$

The resulting $P_{k+1,h+1}$ would then be displaced to produce the desired surface deformation. Note that if either $\hat{u}$ or $\hat{v}$ is already a knot of the surface, we will use the existing knot instead of inserting a new one.

Summary of the Proposed Method

This section summarizes the proposed method for rendering deformable trimmed NURBS surfaces. The method is basically composed of three routines. One is executed as a pre-process. It initializes the NURBS surface and the trimming curves. The other two are executed as the surface or the trimming curves are deforming. They update the affected region of the surface. The three routines are summarized as follows.

Initialization Routine

- The NURBS surface is first decomposed into a list of Bézier patches. Each Bézier patch is tessellated adaptively to form a quad-tree structure. While tessellating the surface, the deformation coefficients of the vertices are computed.
- Likewise, the NURBS trimming curves are also decomposed into a list of Bézier curve segments. Each Bézier trimming curve segment is subdivided adaptively into polylines and organized as a binary-tree structure. The deformation coefficients for the vertices of the trimming polylines are computed.
- By tracing the parametric coordinates of each polyline, the Bézier patch and the quad-tree nodes that the polylines are located can be determined.
- The quad-tree nodes are then classified as visible non-trimmed nodes, invisible non-trimmed nodes and trimmed nodes.
- Trimmed nodes are further classified as convex trimmed nodes, monotonic chain trimmed nodes and complex trimmed nodes.

Trimmed Surface Update Routine

- Based on the local modification property of the NURBS surface, we can easily identify the deformed region and update the positions of the corresponding polygon vertices with the precomputed deformation coefficients.
- When the current level of subdivision is too high or too low to represent the required resolution, a resolution refinement process will be performed to adjust the subdivision.
- If the deformed region contains some trimming curves, we may identify the overlapped trimming curve segments and refine the resolution of the corresponding polylines and compute new deformation coefficients.

Trimming Curve Update Routine

- Based on the local modification property of the NURBS curve, we can easily identify the deformed trimming curve segment. We refine the resolution of the corresponding polylines and compute new deformation coefficients.
- We then retrace the deformed curve segment and determine the Bézier patches and the quad-tree nodes where the corresponding polylines are located.

Results and Discussion

We have implemented the new method in C++ with OpenGL. We have tested it on various aspects through a number of experiments. First, we tested the performance of our method against the method used by OpenGL, i.e., the variant of Rockwood's method.[8] Second, we measured the performance of individual operations used in our method. Third, we studied how the rendering performance of our method changed according to the output quality (or resolution). All the experiments presented here were performed on a Pentium 4 2.0 GHz machine with 256 MB RAM.

Experiment I

In this experiment, we compare our method with the method used in OpenGL in rendering a human face model as shown in Figure 7. It is constructed by a single NURBS surface containing 837 control points and nine separated trimming regions. The four diagrams in Figure 7 show the model before and after deformation, in shaded and wireframe rendering. Another interesting model that we have experimented with is shown in Q2 Figure 8. It is the classical teapot modelled with the proposed method. Since the performance of rendering this teapot model is similar to that of rendering the human face model, we will not show it here. Q2

The deformation events were triggered by randomly moving either 40% of the trimming curve control points or 5% of the surface control points in each frame. From Figure 9, we can see that our method in each consecutive frame is roughly 2.5 and 3 times faster than the OpenGL method for trimming curve deformation and for trimmed surface deformation, respectively. As we deform the trimmed surface and the trimming curves continuously, a complete evaluation is required for the

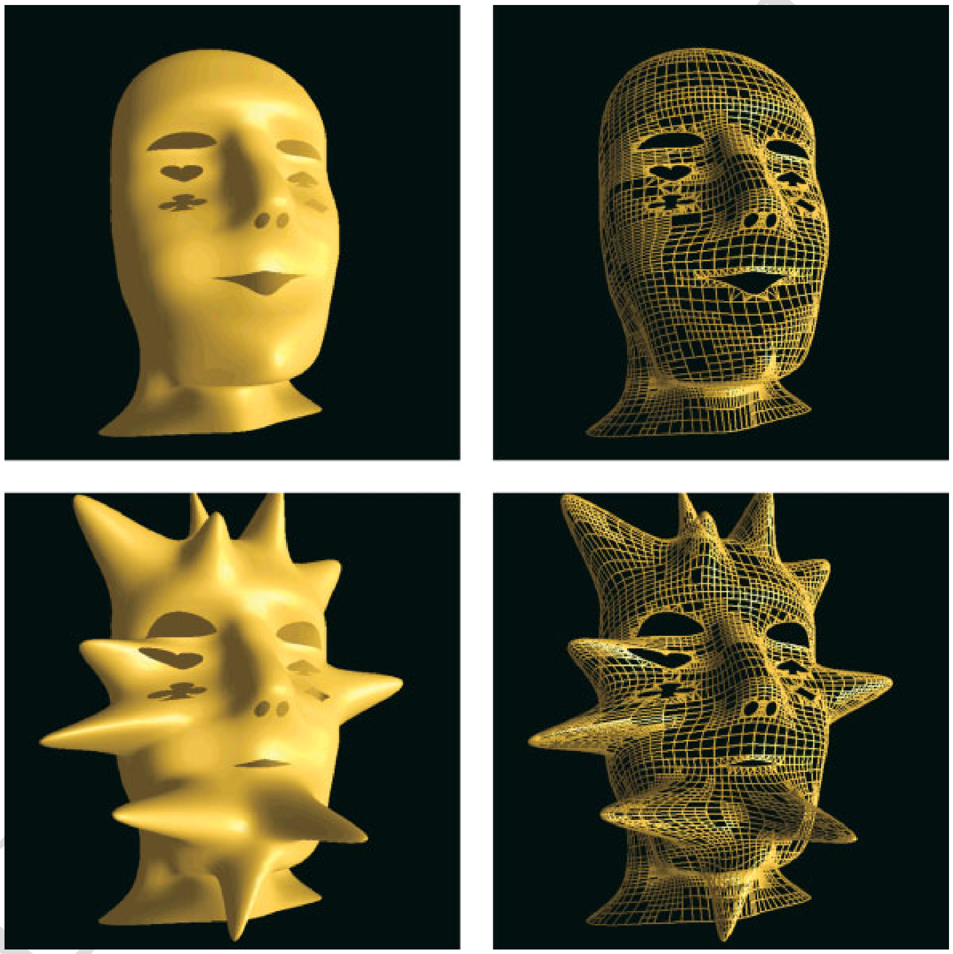


Figure 7. A human face modelled by a trimmed NURBS surface: (upper) shaded and wireframe before deformation, and (lower) shaded and wireframe after deformation on the eyes, mouth and face.

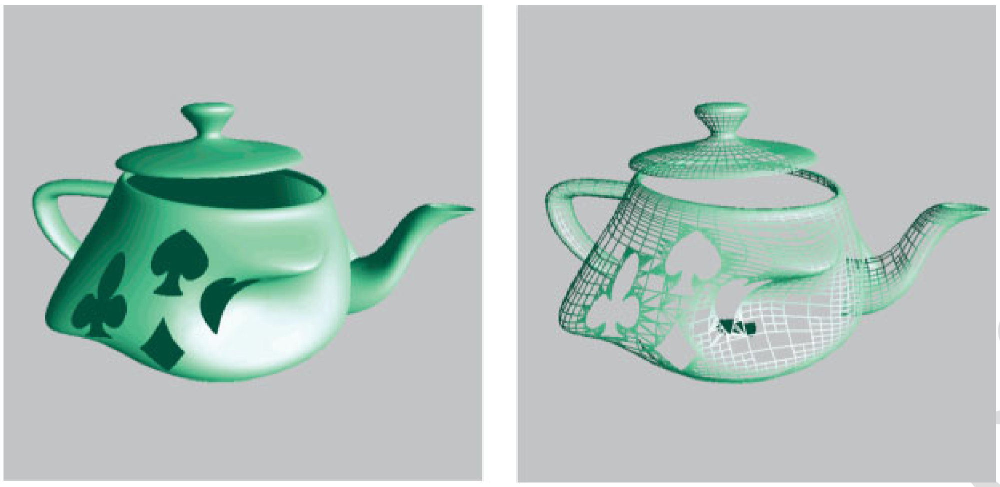


Figure 8. A teapot modelled by a trimmed NURBS surface: shaded and wireframe.

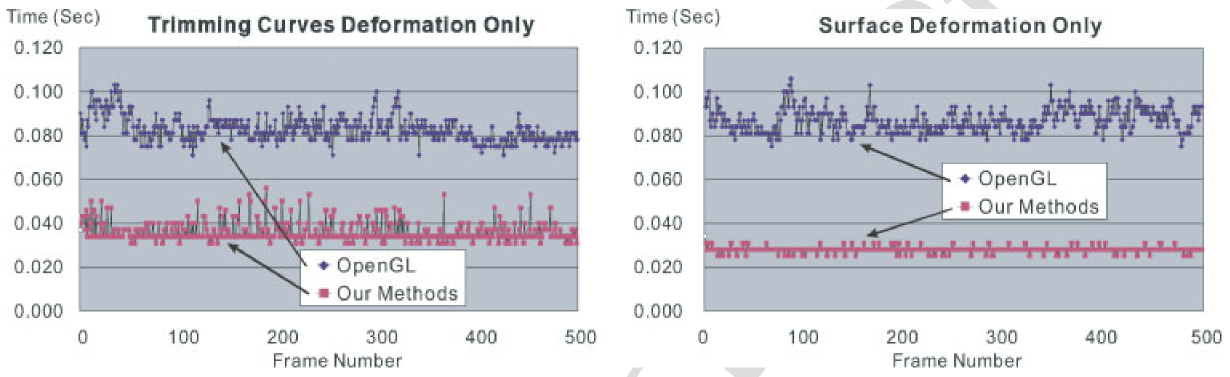


Figure 9. Performance comparison between our method and the method used by OpenGL.

OpenGL method. However, our method only needs to perform an incremental update of the affected region and is thus more efficient.

If we compare trimming curve deformation with trimmed surface deformation as shown in Figure 9, we can see that the computation time of trimming curve deformation is relatively fluctuating. This is because whenever a trimming curve is deformed, we need to perform the retessellation process. The efficiency of this retessellation process is mainly affected by the curvature of the trimming curve, since a smoother trimming curve will generate fewer polylines. For randomly deforming trimming curves, their curvatures are expected to vary significantly. As a result, the computation time would be unstable. In contrast, the surface deformation does not generate a massive update in trimmed surface patches and hence the computation time is relatively stable.

Experiment 2

In this experiment, we measure the performance of individual operations used in our method. The result is shown in Figure 10. *Tessellation* involves the classification and triangulation of all the trimmed nodes. *Coordinate update* involves the incremental updating of the trimming curves and the trimmed surface. *Rendering* refers to the time taken to render all the polygons by the OpenGL engine in each frame. The reason for the tessellation process having a large fluctuation in computation time has already been explained. We can see that the tessellation process only needs to be executed occasionally during surface deformation. For trimming curve deformation, as the surface coordinates of the polylines cannot be calculated with the deformation coefficients, they should be recomputed from the coordinates of the

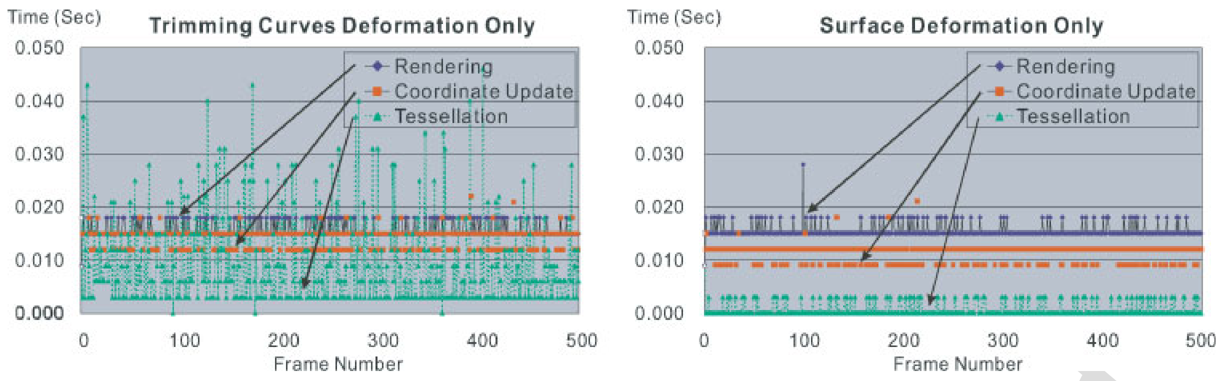


Figure 10. Computational costs of individual operations.

incrementally updated trimming curve. Hence, it is in general more computationally intensive for trimming curve deformation than for trimmed surface deformation. When a trimming curve deforms, we need to perform re-evaluation on the curve to generate the updated polylines representing the deformed curve. However, in practice, this process may not significantly degrade the overall rendering performance as the region of the deformation, and hence the effort spent on the update, is usually relatively small compared with the deformation of the whole surface.

Experiment 3

Since our method also supports multi-resolution modelling, i.e., we may vary the resolution of the output model according to some viewing factors such as distance from the viewer, in this experiment we study how the performance of our method changes as a result of the change in the model resolution. To do this, we first investigate the change in the computation time of the tessellation process due to the change in the number of triangles being tessellated as the surface deforms, as shown in Figure 11. We can see from the diagram that the computation times increase roughly linear to the increase in the number of triangles being tessellated. Hence, as we reduce the resolution of a trimmed surface, we should expect to see a reduction in the computational cost of tessellating the surface.

The second part of this experiment measures the rendering time of a deforming trimmed NURBS surface at different resolutions. The results are shown in Figure 12. When the tessellated surface is at 100% resolution, the surface contains 17,000 triangles. In Figure 12(a), we can see that as the surface deforms in such a way that the resolution of the deformed surface drops to 75% and then 50%, the tessellation time also decreases roughly linearly. A similar result is also observed for trimming curve deformation as shown in Figure 12(b).

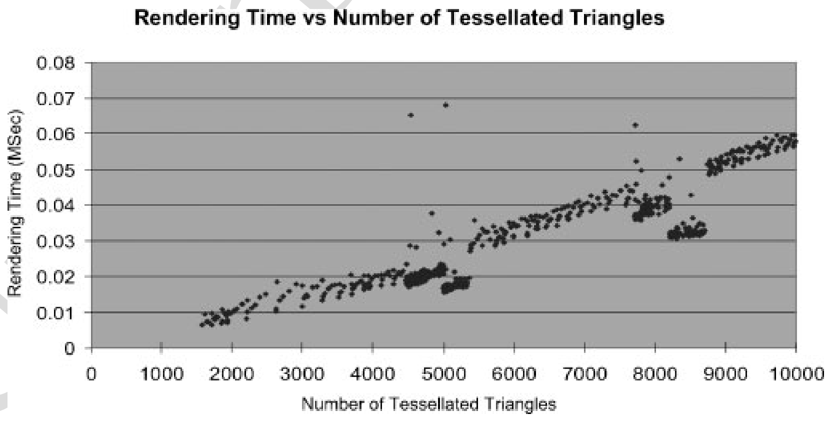


Figure 11. Tessellation time against rendering quality.

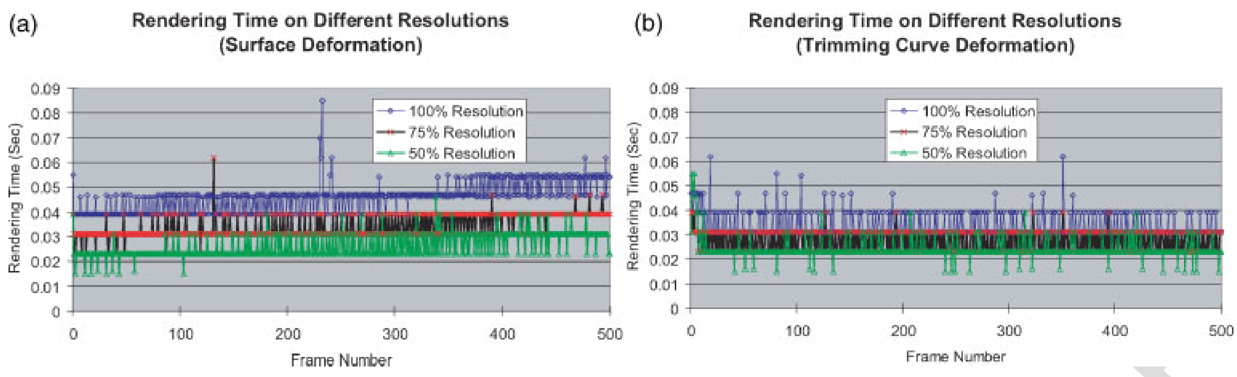


Figure 12. Update time of surface and curve deformation under different resolutions.

Although we have shown that our method proposed here is very efficient in rendering deformable trimmed NURBS surfaces, it has one major limitation. The triangulation method that we use to triangulate monotonic chain trimmed nodes, as discussed above, produces degenerate long, thin triangles. Unfortunately, the Delaunay triangulation method cannot be used here, even though it can produce triangles of more optimal shape. This is because as a trimmed NURBS surface or a trimming NURBS curve deforms, we need to dynamically retriangulate all the affected trimming curves in every frame. The Delaunay triangulation method will be too computationally expensive to be used here.

Conclusion

Deformable objects can improve the realism of real-time applications such as computer games but they are costly to render while they are deforming. In this paper, we have introduced a novel method for efficient rendering of deformable objects represented by trimmed NURBS surfaces. In order to efficiently update the NURBS surface and the trimming curves as the surface and/or the trimming curves are deforming, we propose a regional update mechanism and an efficient method for dynamically tessellating the NURBS surface with the trimming curves. We have shown that the new method for rendering deformable trimmed NURBS surfaces is more efficient than the method used in OpenGL during the curve/surface deformation. As a future work, we are hoping to develop an efficient method for triangulating the monotonic chain trimmed nodes that would produce triangles of better shape.

ACKNOWLEDGEMENTS

The work described in this paper was partially supported by an SRG grant and a DAG grant from City University of Hong Kong (project numbers 7001391 and 7100264).

References

1. Farin G. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press: New York, 1997.
2. Piegl L, Tiller W. *The NURBS Book* (2nd edn). Springer: Berlin, 1997.
3. Abi-Ezzi S, Subramaniam S. Fast dynamic tessellation of trimmed NURBS surfaces. In *Proceedings of Eurographics'94*, 1994; 108–126.
4. Guthe M, Meseth J, Klein R. Fast and memory efficient view-dependent trimmed NURBS rendering. In *Proceedings of Pacific Graphics'02*, June 2002; 204–213.
5. Kahlesz F, Balázs A, Klein R. Multiresolution rendering by sewing trimmed NURBS surfaces. In *Proceedings of ACM Symposium on Solid Modeling and Applications*, June 2002; 281–288.
6. Kumar S, Manocha D, Lastra A. Interactive display of large-scale NURBS models. In *Proceedings of Symposium on Interactive 3D Graphics*, 1995; 51–58.
7. Kumar S, Manocha D, Zhang H, Hoff K. Accelerated walk-through of large spline models. In *Proceedings of Symposium on Interactive 3D Graphics*, 1997; 91–101.
8. Rockwood A, Heaton K, Davis T. Real-time rendering of trimmed surfaces. In *Proceedings of ACM SIGGRAPH'89* 1989; 107–116.
9. Li F, Lau RWH, Green M. Interactive rendering of deforming NURBS surfaces. In *Proceedings of Eurographics'97*, September 1997; 47–56.
10. Li F, Lau RWH, Ng F. Collaborative distributed virtual sculpting. In *Proceedings of IEEE VR*, March 2001; 217–224.
11. Li F, Lau R, Ng F. VSculpt: a distributed virtual sculpting environment for collaborative design. *IEEE Transactions on Multimedia* 2003; **5**(4): 570–580.

12. Li F, Lau RWH. Incremental polygonization of deforming NURBS surfaces. *Journal of Graphics Tools* 1999; **4**(4): 37–50.
13. Farin G, Hansford D. *The Essentials of CAGD*. A. K. Peters: Wellesley, MA, 2000.
14. Lien S, Shantz M, Pratt V. Adaptive forward differencing for rendering curves and surfaces. In *Proceedings of ACM SIGGRAPH'87*, 1987; 111–118.
15. Shantz M, Chang S. Rendering trimmed NURBS with adaptive forward differencing. In *Proceedings of ACM SIGGRAPH'88*, 1988; 189–198.
16. Chang S, Shantz M, Rocchetti R. Rendering cubic curves and surfaces with integer adaptive forward differencing. In *Proceedings of ACM SIGGRAPH'89*, 1989; 157–166.
17. Shirmun L, Abi-Ezzi S. The cone of normals technique for fast processing of curved patches. In *Proceedings of Eurographics'93*, 1993; 261–272.
18. Hoppe H. Progressive meshes. In *Proceedings of ACM SIGGRAPH'96*, August 1996; 99–108.
19. Li F, Lau RWH. Real-time rendering of deformable parametric free-form surfaces. In *Proceedings of ACM VRST*, December 1999; 131–138.
20. Cohen E, Lyche T, Riesenfeld R. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing* 1980; **14**: ●●●–●●●[Q3]. Q3
21. de Casteljau P. *Courbes et Surfaces à Pôles*. S. A. André Citroen, 1959.
22. Lau RWH, To D, Green M. Adaptive multi-resolution modeling technique based on viewing and animation parameters. In *Proceedings of IEEE VRAIS*, March 1997; 20–27.

Authors' biographies:

Gary Cheung received his BSc degree in computer studies from the City University of Hong Kong in 2000. He is currently a research student in the Department of Computer Engineering and Information Technology, City University of Hong Kong. His research interests include computer graphics and surface rendering.

Rynson W. H. Lau received a (top) first-class honours degree in computer systems engineering in 1988 from the University of Kent, England, and a PhD degree in computer graphics in 1992 from the University of Cambridge, England. 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, England, on a defence project on image processing. Rynson Lau's research interests include computer graphics, virtual reality and multimedia systems.

Frederick W. B. Li received a BA (honours) in computing studies in 1994 and an MPhil in computer graphics in 1998, both from the Hong Kong Polytechnic University, and a PhD degree in computer graphics in 2001 from the City University of Hong Kong. He is currently an assistant professor at the Hong Kong Polytechnic University. He was the research manager of an R&D project funded by the Hong Kong Government Innovation and Technology Commission, PCCW, and Sun Microsystems from 2001 to 2003. Frederick Li's research interests include surface modelling, virtual reality, computer animation and computer networking.