Incremental Rendering of Deformable Trimmed NURBS Surfaces

Gary K. L. Cheung[†]
cgary@cs.cityu.edu.hk

Rynson W.H. Lau[†‡]
rynson@cs.cityu.edu.hk

Frederick W.B. Li[†]
borbor@cs.cityu.edu.hk

[†] Department of Computer Engineering & Information Technology, City University of Hong Kong, Hong Kong
[‡] Department of Computer Science, City University of Hong Kong, Hong Kong

ABSTRACT

Trimmed NURBS surfaces are often used to model smooth and complex objects. Unfortunately, most existing hardware graphics accelerators cannot render them directly. Although there are a lot of methods proposed to accelerate the rendering of such surfaces, majority of them are based on tessellation, which is developed primarily for handling non-deforming objects. For an object that may deform in run-time, such as clothing, facial expression, human and animal character, the tessellation process will need to be performed repeatedly while the object is deforming. However, as the tessellation process is very time consuming, interactive display of deforming objects is difficult. This explains why deformable objects are rarely used in virtual reality applications. In this paper, we present a efficient method for incremental rendering of deformable trimmed NURBS surfaces. This method can handle both trimmed surface deformation and trimming curve deformation. Experimental results show that our method performs significantly faster than the method used in OpenGL.

Categories and Subject Descriptors

I.3.5 [**Computational Geometry and Object Modeling**]: Curve, surface, solid, and object representations; I.3.5 [**Methodology and Techniques**]: Interaction techniques

Keywords

Deformable objects, NURBS surfaces, trimmed surfaces, real-time rendering

1. INTRODUCTION

Deformable objects have been considered as important to virtual reality applications, as they may model clothing, facial expression, human and animal characters. In particular, Non-Uniform Rational B-Splines (NURBS) [4, 16] are often employed to represent such objects as they can be used to produce a variety of shapes simply by manipulating their control points and weights. However, NURBS surfaces are seldom used in interactive applications that demand real-time rendering performance because of their high rendering cost. There have been a lot of work carried out to address this problem. Most of the methods developed are based on tessellation [1, 6, 8, 9, 10, 17]. This tessellation process subdivides the NURBS surfaces into polygons so that the hardware graphics accelerator, if present, may render the polygons in real-time. However, this 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. Existing rendering methods would be difficult to render these objects in real-time.

Earlier, we proposed an efficient method for rendering deforming NURBS surfaces [13]. The method pre-computes 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 this method is much more efficient than existing methods. Recently, we have applied this method to develop a distributed virtual sculpting system [14, 15]. 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 of this project is to extend the method to support trimming [5], 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 [13] to support trimming. The rest of the paper is outlined as follows. Section 2 provides a brief survey on related work. Section 3 describes how we handle the deformable trimmed NURBS surfaces. Section 4 presents our method for tessellating trimmed NURBS surfaces. Section 5 describes how to incremental update trimming curves and trimmed NURBS surfaces as they deform. Section 6 demonstrates the performance of the method through some experiments. Finally, section 7 briefly concludes the paper.

2. 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. In particular, Shantz et al. [18] propose an adaptive forward differencing technique to evaluate the points on the surface incrementally to produce a polygon model 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. Rockwood et al. [17] 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 et al. [1] tessellate trimmed NURBS surfaces in a way similar to [17], but it further minimizes the number of patches needed to be tessellated by culling out the invisible patches dynamically during run-time. Kumar et al. [9] improve the performance of the tessellation process by avoiding the operation of subdividing Bézier patches into uv-monotone patches. Instead, it directly tessellates the Bézier patches into polygons for rendering. However, to allow fast back-patch culling, it needs to compute pseudo normal surfaces for all Bézier patches. In [10], they enhanced their method in [9] by constructing super-surfaces on the Bézier patches to allow a further reduction in the number of polygons of the resulting polygon model.

Unfortunately, all the methods mentioned above cannot handle deforming trimmed NURBS surfaces efficiently. Regardless of the high computational cost of the tessellation process, 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 will even need to re-generate a new set of uv-monotone patches before the tessellation process.

Recently, Kahlesz et al. [8] developed an adaptive tessellation method. It recursively subdivides a surface into a quad-tree hierarchy according to some approximation criteria. 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 be 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 surfaces for multi- resolution modeling [6]. A seam graph structure is actually a progressive mesh [7] like structure. It allows the application to select an appropriate resolution of the tessellated polygon model for rendering. However, the construction of the seam graph structure is itself an expensive process. Hence, this method, like most of the other methods, is not suitable for interactive rendering of trimmed NURBS surfaces.

3. HANDLING OF DEFORMABLE TRIMMED NURBS SURFACES

In our earlier work, we developed a technique for efficient rendering of deformable NURBS surfaces [11, 13]. 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 also model them in a similar way.

3.1 Handling of the NURBS Surfaces

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

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

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

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

$\alpha_{i,j}$ 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 precompute the deformation coefficients and update the polygon model incrementally as shown in Equation (1).

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 [2]. Each Bézier patch is then subdivided into a polygon model, which is maintained in a quadtree hierarchy, by applying the de Casteljau subdivision formula [3] to the Bernstein polynomials in both u and v directions. We refer to this as the *polygon hierarchy.* For example, in u, we have:

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

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

The difference of Equation (3) before and after the deformation can be simplified to get a de Casteljau formula as follows:

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

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.

3.2 Handling of the Trimming Loops

A trimmed NURBS surface is defined by a set of trimming loops together with the NURBS surface itself. 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:

$$C(t) = \sum_{i=0}^{n} \beta_i(t) P_i \tag{5}$$

where P_i denotes the control points. $\beta_i(t)$, similar to the case of NURBS surfaces, represents the set of *deformation coefficients* of the NURBS curve. It is defined as:

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

To trim a NURBS surface, we subdivide the NURBS curves against the polygon hierarchy of the NURBS surface to form *polylines*. By comparing the intersection points between the polylines and the polygon hierarchy in the parametric space, we may obtain a list of polylines for each quadtree node in the polygon hierarchy that are inscribed in the node.

4. TESSELLATION OF TRIMMED NODES

To render a trimmed NURBS surface, according to the current viewing parameters, we select the appropriate quadtree nodes in the polygon hierarchy and tessellate them based on their types. We classify all the nodes into three types: *visible non-trimmed nodes*, *invisible non-trimmed nodes* and *trimmed nodes*. For a visible non-trimmed node, we just need to split it into 2 triangles along its diagonal. For an invisible non-trimmed node, as it is completely trimmed out by some trimming loop, we would just ignore it and would not render it. For a trimmed 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.

4.1 Convex Trimmed Nodes

A convex trimmed node is a node with a convex trimmed region. To classify such a node, we make use of the strong convex hull property of the NURBS curve definition. 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 such a node is identified, there are two possible cases as shown in Figure 1. Either the exterior or the interior node region is trimmed out by the trimming curve as shown in figures 1(a) and 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*.

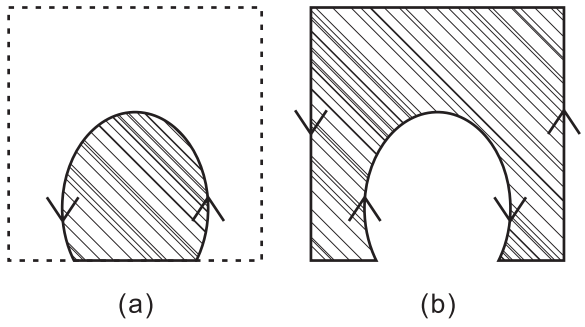


Figure 1: Note trimming: (a) exterior node region is trimmed out, and (b) interior node region is trimmed out.

4.2 Monotonic Chain Trimmed Nodes

A polyline segment is a monotonic chain with respect to axis L if the polylines of the inscribed trimming curve segment have at most 2 intersections to any L' perpendicular to L. It is similar to the monotone definition but a monotone refers to a polygon while a monotonic chain refers to the polylines. In our method, the monotonic chain is respected to the u 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 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}, as shown in figure 2(b). In other words, there are four uv-monotonic polylines in the node, which have the following properties:

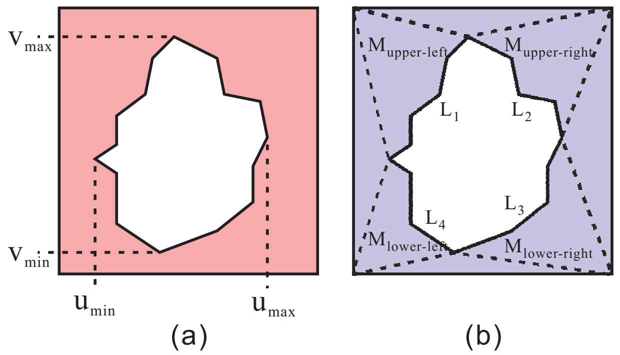


Figure 2: Handling of a monotonic chain trimmed node: (a) determining maxima and minima, and (b) partitioning of the node into four regions according to the maxima and minima.

In u direction:

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

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

In v direction:

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

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

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, $M_{upper-left}$, as shown in figure 3. Since a uv-monotonic region is convex in nature, a simplest way to tessellate it is by joining each vertex of the monotonic polylines within the region to the nearest corner point of the region. As an example, the nearest corner point of $M_{upper-left}$ as shown in figure 3 is $P_{upper-left}$, which has the coordinate (u_{left}, v_{upper}). Since the uv-monotonic polylines are inscribed in the node, we can have $u_{left} \leq u_{min}$ and $v_{upper} \geq v_{min}$. The tessellation is done by adding lines from corner point $P_{upper-left}$ to each vertex P_i on the uv-monotonic polylines of $M_{upper-left}$.

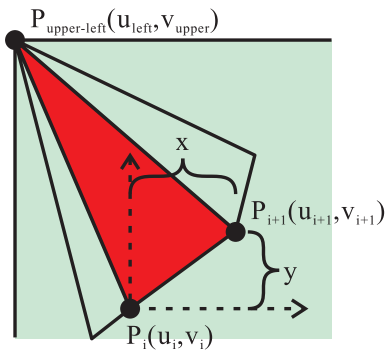


Figure 3: Tessellation of the upper-left *uv*-monotonic region.

To show that this tessellation process would 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 end points of a polyline. Refer to figure 3 as an example. The two end points 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 $P_{upper-left}$ and the coordinates of the triangle are (u_{left}, v_{upper}), (u_i, v_i) and (u_{i+1}, v_{i+1}). If the orientation of these 3 vertices is always turning left, their determinant should then be positive. To evaluate the determinant, we re-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}. According to the monotone property, Δx and Δy will always be positive in $M_{upper-left}$. The determinant of the 3 vertices is calculated as follows:

$$\begin{aligned} & \begin{vmatrix} u_i - u_{left} & (\Delta x + u_i) - u_{left} \\ v_i - v_{upper} & (\Delta y + v_i) - v_{upper} \end{vmatrix} \\ = \; & [\Delta y(u_i - u_{left}) + (u_i - u_{left})(v_i - v_{upper})] - \\ & [\Delta x(v_i - v_{upper}) + (u_i - u_{left})(v_i - v_{upper})] \\ = \; & \Delta y(u_i - u_{left}) - \Delta x(v_i - v_{upper}) \end{aligned} \quad (7)$$

As $u_{left} \leq u_{min}$ and $\Delta y \geq 0$, the first part, $\Delta y(u_i - u_{left})$, of equation 7 must be positive. In addition, as $v_{upper} \geq v_{min}$ and $\Delta x \geq 0$, the second part, $-\Delta x(v_i - v_{upper})$, of equation 7 must also be positive. Therefore, the result of equation 7 must always be positive.

By combining the 4 uv-monotonic regions together, the result of the tessellation process may become as shown in figure 4. Since this tessellation process requires to identify the 2 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. However, if a node is identified as a non-monotonic chain trimmed node, i.e., it is a complex trimmed node, a further subdivision is required.

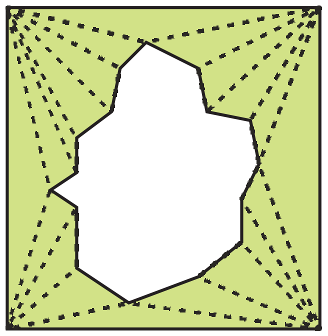


Figure 4: The resultant tessellation of a monotonic chain trimmed node.

4.3 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 most likely a complex trimmed node. To handle this kind of nodes, we need to further subdivide each of them into child nodes. This may likely also involve subdividing the trimming curve to 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. On average, each child node contains about one-fourth of the original trimming curve. Such a reduction in irregularity is very effective. According to our experiments, more than 80% of nodes in a surface would be subdivided into convex trimmed nodes or monotonic chain trimmed nodes. Only 20% of the nodes need to be further subdivided. From our experience, most nodes require only 1 or 2 levels of subdivision to partition a complex trimmed node into convex trimmed nodes and/or monotonic chain trimmed nodes.

5. OBJECT DEFORMATION

One of the critical developments in our research is that our method supports real-time deformation of both the trimming curves and the trimmed 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 only affect the shape of the surface itself; it does not affect the shape 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.

5.1 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. According to the local modification property of the NURBS curves, any deformation driven by moving 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 end points 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.

5.2 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 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 due 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 $(t_i, min\{t : t_i < t\})$.

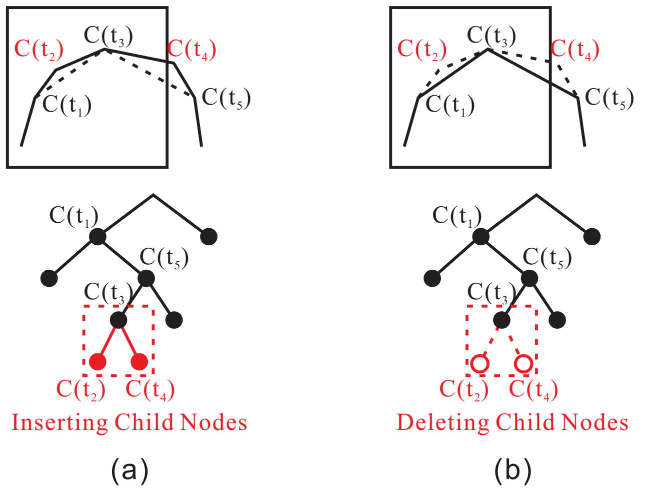


Figure 5: Resolution refinement of a trimming curve segment: (a) resolution increase, and (b) resolution decrease.

Note that the curvature change 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.

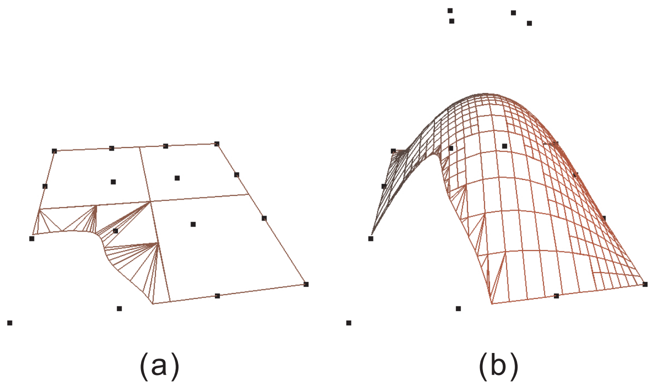
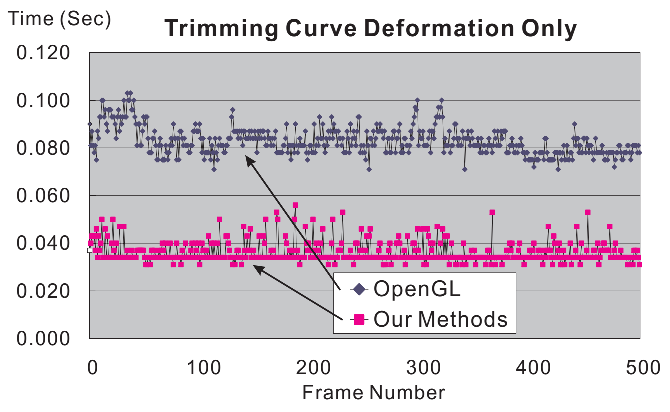


Figure 6: Resolution refinement of a trimmed NURBS surface: (a) before deformation, and (b) after deformation.



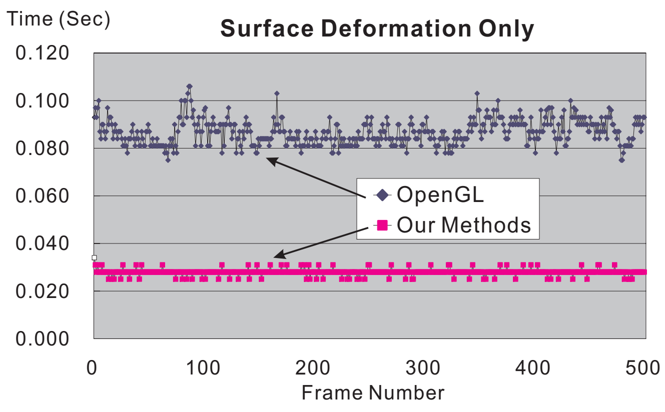
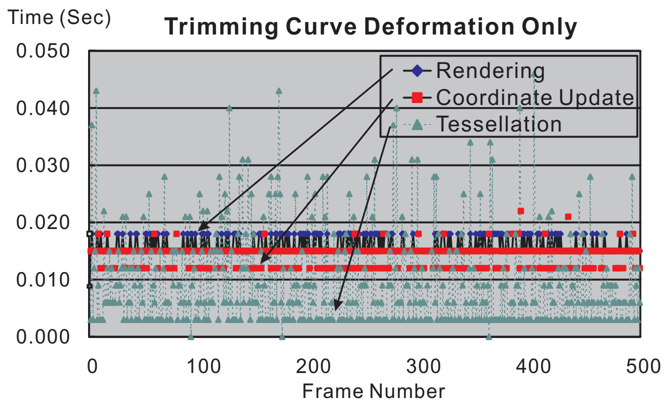


Figure 7: Performance comparison between our method and the method used by OpenGL.



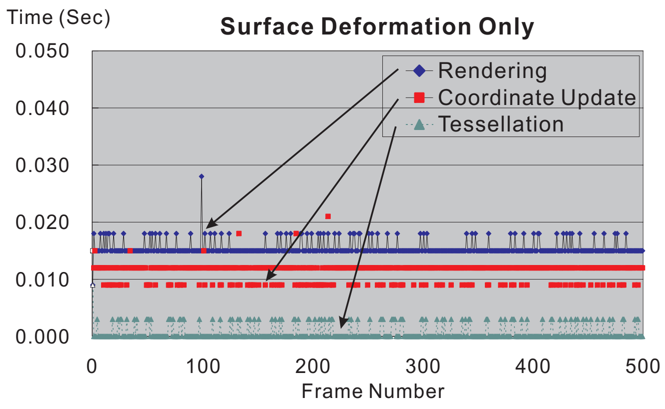


Figure 8: Computational costs of individual operations.

6. RESULTS AND DISCUSSIONS

We have implemented the new method in C++ and OpenGL. All the experiments presented here have been performed on a Pentium 4 2.0GHz machine with 256 MB RAM. 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. Figure 9 shows a human face model that we used to test the proposed rendering method. The model is constructed by a single NURBS surface containing 837 control points and 9 separate trimming curves. The figure shows the model before and after deformation, in shaded and wireframe rendering. Figure 10 shows another test model of a classical teapot after deformation.

To demonstrate the performance of our method, we have compared it with the method used by OpenGL, i.e., the variant of Rockwood's method [17], using the human face model shown in figure 9. Figure 7 shows the rendering times of the two methods, 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. The reason for this is that as we deform the trimmed surface and the trimming curves continuously, a complete evaluation is required in each frame for the OpenGL method. However, our method only needs to perform an incremental update to the affected region and is thus more efficient.

If we compare trimming curve deformation with trimmed surface deformation as shown in figure 7, we can see that the performance 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 less polylines. For the 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 of trimmed surface patches and hence, the computation time is relatively stable.

Figure 8 shows the performance of individual operations used in our method. Tessellation refers to the classification and triangulation of all the trimmed nodes. Coordinate update refers to 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. We can see that the tessellation process only needs to be executed occasionally in the surface deformation case. 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 incrementally updated trimming curve. Hence, this process 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.

7. CONCLUSION

In this paper, we have introduced an interactive rendering method for deformable trimmed NURBS surfaces. In order to efficiently update the trimmed 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.

8. ACKNOWLEDGMENTS

The work described in this paper was partially supported by two SRG grants (Project Numbers: 7001391 and 7001465) and a DAG grant (Project Number: 7100264), all from City University of Hong Kong.

9. REFERENCES

[1] S. Abi-Ezzi and S. Subramaniam. Fast Dynamic Tessellation of Trimmed NURBS Surfaces. In *Proc. of Eurographics '94*, pages 108–126, 1994.

[2] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics. *Computer Graphics and Image Processing*, 14, October 1980.

[3] P. de Casteljau. *Courbes et Surfaces à Pôles.* S. A. André Citroen, 1959.

[4] G. Farin. *NURBS from Projective Geometry to Practical Use.* A K Peters, Natick, MA, 1999.

[5] G. Farin and D. Hansford. *The Essentials of CAGD.* A K Peters, Natick, MA, 2000.

[6] M. Guthe, J. Meseth, and R. Klein. Fast and Memory Efficient View-Dependent Trimmed NURBS Rendering. In *Proc. of of Pacific Graphics '02*, pages 204–213, June 2002.

[7] H. Hoppe. Progressive Meshes. In *Proc. of ACM SIGGRAPH '96*, pages 99–108, August 1996.

[8] F. Kahlesz, A. Balázs, and R. Klein. Multiresolution Rendering By Sewing Trimmed NURBS Surfaces. In *Proc. of ACM Symposium on Solid Modeling and Applications*, pages 281–288, June 2002.

[9] S. Kumar, D. Manocha, and A. Lastra. Interactive Display of Large-Scale NURBS Models. In *Proc. of Symposium on Interactive 3D Graphics*, pages 51–58, 1995.

[10] S. Kumar, D. Manocha, H. Zhang, and K. Hoff. Accelerated Walkthrough of Large Spline Models. In *Proc. of Symposium on Interactive 3D Graphics*, pages 91–101, 1997.

[11] F. Li and R. Lau. Incremental Polygonization of Deforming NURBS Surfaces. *Journal of Graphics Tools*, 4(4):37–50, 1999.

[12] F. Li and R. Lau. Real-Time Rendering of Deformable Parametric Free-Form Surfaces. In *Proc. of ACM VRST*, pages 131–138, December 1999.

[13] F. Li, R. Lau, and M. Green. Interactive Rendering of Deforming NURBS Surfaces. In *Proc. of Eurographics '97*, pages 47–56, September 1997.

[14] F. Li, R. Lau, and F. Ng. Collaborative Distributed Virtual Sculpting. In *Proc. of of IEEE VR'01*, pages 217–224, March 2001.

[15] F. Li, R. Lau, and F. Ng. VSculpt: A Distributed Virtual Sculpting Environment for Collaborative Design. *IEEE Trans. on Multimedia (to appear)*, 2003.

[16] L. Piegl and W. Tiller. *The NURBS Book Second edition.* Springer-Verlag, Tyler, TX, 1997.

[17] A. Rockwood, K. Heaton, and T. Davis. Real-Time Rendering of Trimmed Surfaces. In *Proc. of ACM SIGGRAPH '89*, volume 23, pages 107–116, 1989.

[18] M. Shantz and S. Chang. Rendering Trimmed NURBS with Adaptive Forward Differencing. In *Proc. of ACM SIGGRAPH '88*, pages 189–198, 1988.

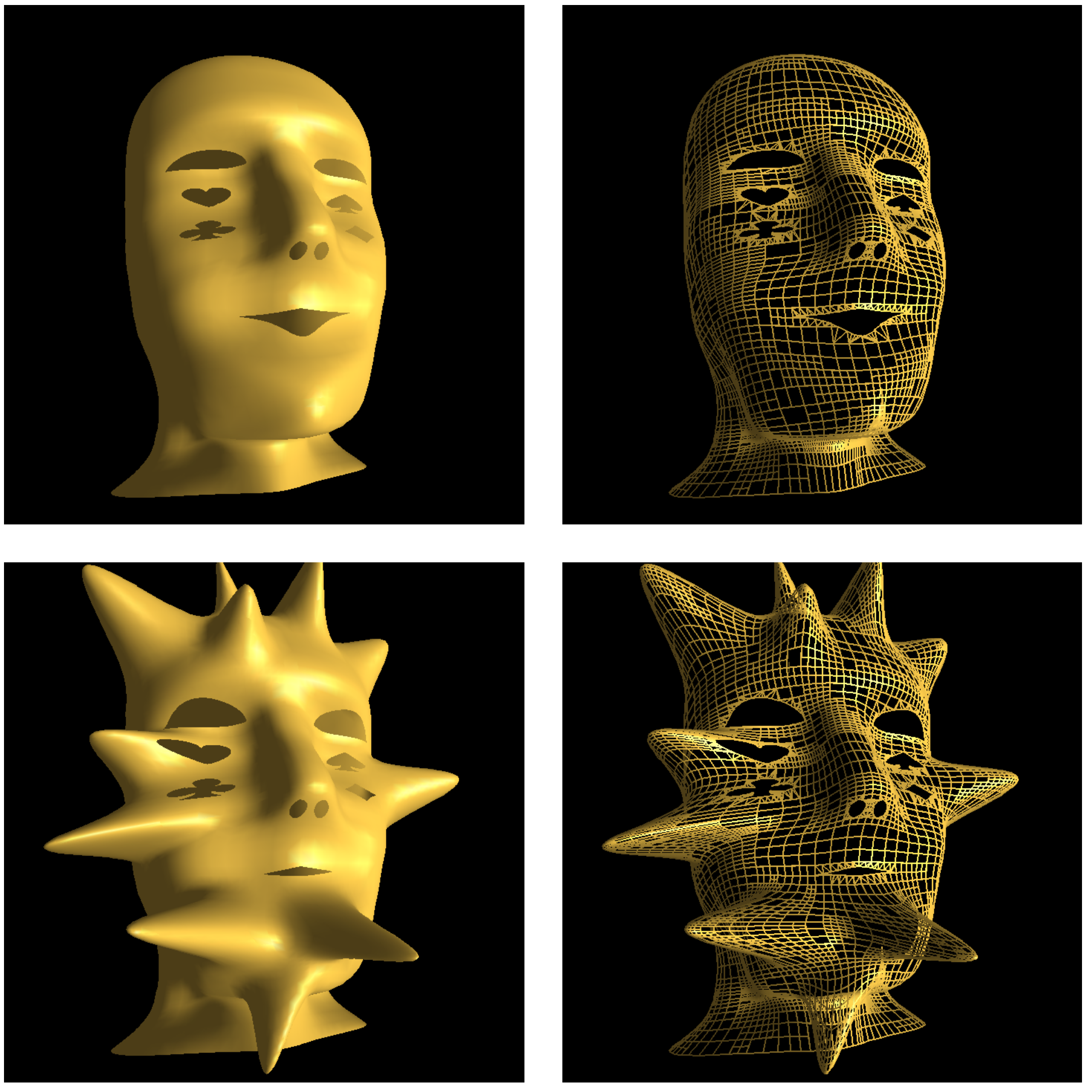


Figure 9: A human face modeled by a trimmed NURBS surface: (upper) shaded and wireframe before deformation, and (lower) shaded and wireframe after deformation on the eyes, mouth and face.

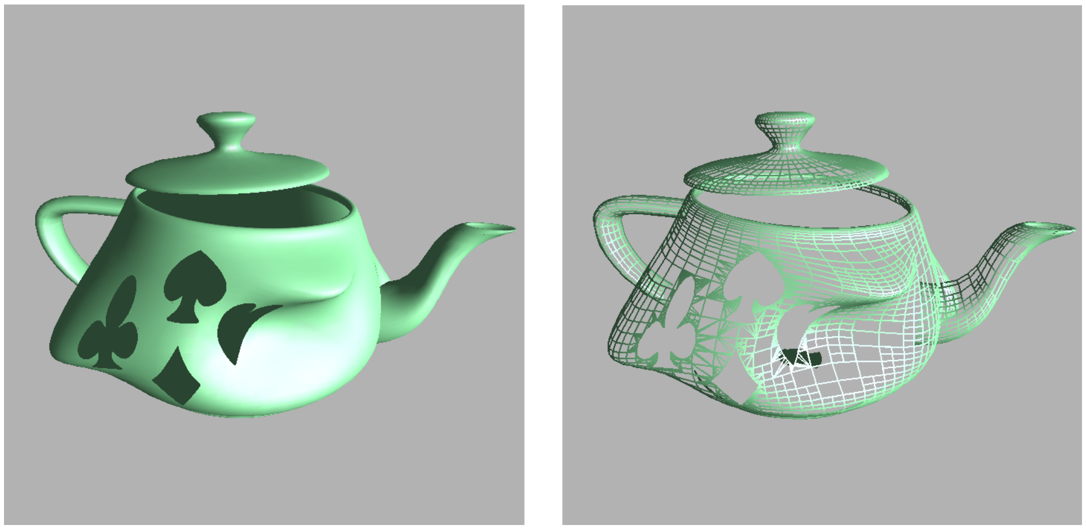


Figure 10: A teapot modeled by a trimmed NURBS surface: shaded and wireframe before deformation.