In Proceedings of ACM Symposium on Virtual Reality Software and Technology, pp. 131-138, Dec. 1999.

Real-Time Rendering of Deformable Parametric Free-Form Surfaces

Frederick W. B. Li Rynson W. H. Lau

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

Abstract

Deformable objects are required to improve the realism of virtual reality applications. They are particularly useful in modeling clothes, facial expression, human and animal characters. A common method to render these objects is by tessellation. However, the tessellation process is computationally very expensive. If the object deforms, we need to retessellate the surface every frame, as its shape changes from one frame to the next. This computational burden poses a significant challenge to the real-time rendering of deformable objects. Consequently, deformable objects are seldom incorporated in existing virtual reality systems. In this paper, we present an incremental method for rendering deformable objects modeled by parametric free-form surfaces. We also introduce two new frame coherence techniques for crack prevention and parameter caching. Finally, we present a single hierarchical data structure which provides a multi-resolution representation of the object model.

1 Introduction

Virtual reality systems provide a means for a user to experience and interact in an artificial environment. As the computational power of computers increases, users are demanding for an ever increasing level of realism in virtual reality. In particular, deformable objects are used in modeling clothing, facial expression, human and animal characters. Very often, parametric free-form surfaces [1, 2], such as Bézier surfaces and B-spline surfaces, are used to model deformable objects because their shapes can be easily modified by moving one or more control points. To give the user a realistic perception and instant response, it is very important for virtual reality systems to be able to render deformable objects in real-time. However, this goal is difficult to achieve because of three reasons.

First, because majority of the current hardware graphics accelerators can only handle polygons, tessellation becomes a standard way of rendering parametric free-form surfaces. There are a lot of tessellation methods proposed [3, 4, 5, 6, 7, 8]. Basically, these methods apply some tessellation algorithm to the defining equation of the surface to produce a polygon model for rendering. However, the tessellation process is usually computationally very expensive. While the object is deforming, this process must be executed in each frame to reflect the change of shape. Second, the performance of the tessellation process generally decreases as the degree or complexity of the defining equation of the surface increases. Third, in many applications, we may want to have many deformable objects in an environment. Interactive rendering may become difficult to achieve.

One common way to accelerate the rendering speed of object models is to employ a *multi-resolution method* [9] for object modeling, which minimizes the number of polygons to represent the objects. However, because most multi-resolution methods proposed are for rigid objects and often require lengthly pre-processing, they cannot be used to model deformable objects directly.

In this paper, we present an incremental rendering method for deformable parametric free-form surfaces. The new method focuses on tackling the complexity and un-determination in real applications, where many deformable objects may be involved and their deformations may not be known in advance. It can be applied to different kinds of parametric surfaces. The rest of the paper is organized as follows. Section 2 reviews previous works on rendering of parametric free-form surfaces. Section 3 describes our incremental method for rendering deformable parametric free-form surfaces. Sections 4 and 5 introduce frame coherent techniques on incremental crack prevention and parameter caching respectively. Section 6 discusses how we may construct hierarchical representations of both the parametric surface and the corresponding polygon model. Finally, section 7 presents some experimental results and section 8 concludes the work presented in this paper.

2 Previous Works

There are many methods developed for rendering parametric free-form surfaces. Most of them are based on tessellation, which makes use of the polygon rendering capability of existing hardware graphics accelerators. They include uniform subdivision methods from Rockwood et al. [8] and Kumar et al. [6], and adaptive subdivision methods from Filip [5] and Abi-Ezzi et al. [3, 4]. The adaptive subdivision

method generally produces fewer polygons than the uniform subdivision method. However, it requires extra processing time for flatness test and crack prevention. To minimize the number of polygons generated, Kumar et al. [6] incrementally tessellates surface segments that will become visible in the next frame and deletes polygons that will become invisible. The method takes advantage of the small change in the user's viewpoint between two consecutive frames. In [7], Kumar et al. enhances their original method by adopting adaptive subdivision and parallelization to their method. To further improve the rendering performance, the *multi-resolution modeling* technique can be used to minimize polygon count by representing objects at variable resolution according to the distance of the object from the viewer. The idea is to use a lower resolution model to represent a distant object so that less time is spent on drawing insignificant details. There are many methods proposed for generating multi-resolution representations of arbitrary polygon models [10, 11, 12, 13, 14, 15, 16]. Others generate multi-resolution representations of parametric free-form surfaces [17, 18].

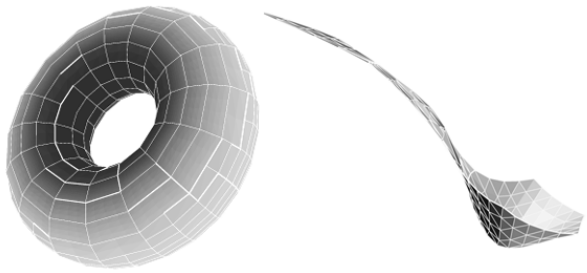
Most of the methods discussed above, however, do not facilitate the rendering of deformable objects. They accelerate only the tessellation of the surfaces without concerning the incremental evolution from one frame to another. Hence, the complete tessellation process must be executed in every frame as the object deforms. The multi-resolution methods for polygon models, on the other hand, implicitly assume that the objects for rendering are rigid, so that they can precompute simplified key models or preprocess some data structures for real-time resolution refinement. Although Zorin et al. [18] proposes a method for interactive multi-resolution mesh editing, the method still requires the polygon model to be rebuilt through the analysis and synthesis procedures when the object deforms.

In [19], we proposed an incremental rendering method for deformable NURBS surfaces. The method accelerates the rendering process by generating polygon models and refines their resolution incrementally without referring to the defining equation of the NURBS surfaces. Results showed that the method was roughly 3 to 15 times faster than existing methods. In this paper, we extend the rendering method to cover different types of parametric surfaces. The contributions of this paper include:

- Generalization of the polygon model updating process and the resolution refinement process.
- Introducing a hierarchical method for organizing the surface and the corresponding polygon model to optimize the number of polygons needed to represent the surface.
- Introducing two frame coherence techniques, incremental crack prevention and parameter caching.

3 Rendering Deformable Parametric Free-Form Surfaces

Many different parametric free-form surfaces have been developed including Bézier surfaces, B-spline surfaces, Bézier triangles and multivariate objects. They are used in different applications according to their strengths and limitations. For example, the tensor-product Bézier surfaces and the NURBS surfaces are commonly used in CAD/CAM applications, and scattered data interpolation and approximation. This is because they possess simple geometric definitions and regular shapes, and their computational costs are relatively low compared to other parametric surfaces. However, these surfaces may not be sufficient to model higher dimensional volumes or objects with complicated shapes. The triangular surfaces and multivariate surfaces or volumes are developed to tackle this issue. Common examples include Bézier triangles, tensor-product Bézier volumes and tetrahedral Bézier volumes.



(a) (b)

Figure 1: Examples of different kinds of surfaces.

The main idea of our method is that we maintain two data structures of the deforming 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. Our method is based on two fundamental techniques, *incremental polygon updating* and *resolution refinement*. To show how our method can be applied to different types of parametric free-form surfaces, we categorize the surfaces according to the mathematical properties of their defining equations. They are the surfaces defined by the multi-dimensional tensor-product of univariate Bernstein polynomials (S_A) and the generalized Bernstein polynomials over the barycentric coordinates (S_B). They are defined as follows:

$$S_A(u,v,w,\cdots) = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n}\cdots B_i^l(u)B_j^m(v)B_k^n(w)\cdots P_{ijk\cdots}$$

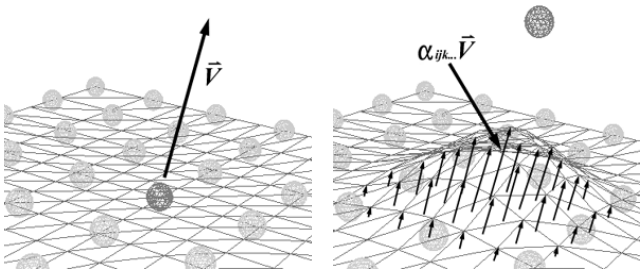
$$S_B(u,v,w,\cdots) = \sum_{i+j+k+\cdots=n} B_{ijk\cdots}^n(u,v,w\cdots)P_{ijk\cdots}$$

where $B_i^l(u)B_j^m(v)B_k^n(w)\cdots$ and $B_{ijk\cdots}^n(u,v,w\cdots)$ are the basis functions. $P_{ijk\cdots}$ form the control net.

Figures 1(a) and 1(b) show sample objects defined by the multi-dimensional tensor-product of univariate Bernstein polynomials and the generalized Bernstein polynomials over the barycentric coordinates, respectively.

3.1 Incremental Polygon Model Updating

At start, we generate a polygon model to represent the initial state of the deformable surface. As the surface deforms, we incrementally update the existing polygon model to produce the deformed polygon model. To show how this technique works, we consider the polygonal representation of a surface obtained by evaluating the surface equation with some discrete parametric values. If a control point $P_{ijk\cdots}$ is moved to $\overline{P}_{ijk\cdots}$ with the displacement vector $\overrightarrow{V} = \overline{P}_{ijk\cdots} - P_{ijk\cdots}$, the incremental difference between the two polygonal representations for each of the two types of surfaces before and



(a) (b)

Figure 2: Incremental polygon model updating.

after the control point movement, is as follows.

For the multi-dimensional tensor-product of univariate Bernstein polynomials:

$$\overline{S}_A(u,v,w,\cdots) - S_A(u,v,w,\cdots) = (\overline{P}_{ijk\cdots} - P_{ijk\cdots})B_i^l(u)B_j^m(v)B_k^n(w)\cdots = \alpha_{ijk\cdots}\overrightarrow{V}$$

where $\alpha_{ijk\cdots} = B_i^l(u)B_j^m(v)B_k^n(w)\cdots$

For the generalized Bernstein polynomials over the barycentric coordinates:

$$\overline{S}_B(u,v,w,\cdots) - S_B(u,v,w,\cdots) = (\overline{P}_{ijk\cdots} - P_{ijk\cdots})B_{ijk\cdots}^n(u,v,w\cdots) = \beta_{ijk\cdots}\overrightarrow{V}$$

where $\beta_{ijk\cdots} = B_{ijk\cdots}^n(u,v,w\cdots)$.

It is obvious that the two deformation coefficients $\alpha_{ijk\cdots}$ and $\beta_{ijk\cdots}$ are constants for each particular set of $(u,v,w,\cdots)$ parameter values. Hence, if the resolution of the polygon model representing the surface remains unchanged after the deformation, we may precompute the deformation coefficients and update the polygon model incrementally by the deformation coefficients and the displacement vector. This technique is very efficient since we need to perform only one addition and one multiplication on each affected vertex of the polygon model to produce the deformed one. In addition, its performance is independent of the complexity of the deforming surface.

In the implementation, the incremental polygon model updating is carried out in two stages, the *pre-processing* stage and the *run-time stage*. In the pre-processing stage, we tessellate the surface to obtain a polygon model. We also evaluate a set of deformation coefficients $\alpha_{ijk\cdots}$ or $\beta_{ijk\cdots}$ for each control point. As the surface deforms in run-time, the polygon model is incrementally updated with the set of deformation coefficients and the displacement vector of the moving control point. Figure 2(a) shows a surface deformation by moving a control point with displacement vector $\overrightarrow{V}$. Figure 2(b) shows the incremental updating of the affected polygon vertices.

3.2 Resolution Refinement

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

For a surface defined by the multi-dimensional tensor-product of univariate Bernstein polynomials, we may subdivide it into a polygon model by applying the de Castlejau subdivision formula to all Bernstein polynomials at different parametric directions. For example, in the u direction, we have:

$$P_{ijk\cdots}^{rjk\cdots}(u) = (1-u)P_{ijk\cdots}^{r-1,j,k,\cdots}(u) + uP_{i+1,j,k,\cdots}^{r-1,j,k,\cdots}(u) \quad (1)$$

for $r = 1,\ldots,l$, $i = 0,\ldots,l-r$ and all $j,k,\cdots$, where $(l,m,n,\cdots)$ is the degree of the surface. The other parametric directions have similar recursions.

If the surface deforms, the above equation becomes:

$$\overline{P}_{ijk\cdots}^{rjk\cdots}(u) = (1-u)\overline{P}_{ijk\cdots}^{r-1,j,k,\cdots}(u) + u\overline{P}_{i+1,j,k,\cdots}^{r-1,j,k,\cdots}(u) \quad (2)$$

By subtracting Equation(1) from Equation(2), we obtain:

$$\overline{P}_{ijk\cdots}^{rjk\cdots}(u) - P_{ijk\cdots}^{rjk\cdots}(u) = (1-u)(\overline{P}_{ijk\cdots}^{r-1,j,k,\cdots}(u) - P_{ijk\cdots}^{r-1,j,k,\cdots}(u)) + u(\overline{P}_{i+1,j,k,\cdots}^{r-1,j,k,\cdots}(u) - P_{i+1,j,k,\cdots}^{r-1,j,k,\cdots}(u)) \quad (3)$$

We then substitute the deformation coefficients and the control point displacement vector into Equation(3) to become:

$$\alpha_{ijk\cdots}^{rjk\cdots}(u)\overrightarrow{V} = (1-u)\alpha_{ijk\cdots}^{r-1,j,k,\cdots}(u)\overrightarrow{V} + u\alpha_{i+1,j,k,\cdots}^{r-1,j,k,\cdots}(u)\overrightarrow{V} \quad (4)$$

Since equation(4) is true for any vector $\overrightarrow{V}$, we must have

$$\alpha_{ijk\cdots}^{rjk\cdots}(u) = (1-u)\alpha_{ijk\cdots}^{r-1,j,k,\cdots}(u) + u\alpha_{i+1,j,k,\cdots}^{r-1,j,k,\cdots}(u) \quad (5)$$

for $r = 1,\ldots,l$, $i = 0,\ldots,l-r$ and all $j,k,\cdots$. Equation(5) indicates that deformation coefficients can be generated incrementally by the de Casteljau subdivision formula.

On the other hand, a surface defined by the generalized Bernstein polynomials is evaluated by applying the generalized de Casteljau subdivision formula. The generalized Bernstein polynomials of degree n in generalized de Casteljau form are:

$$P_{\mathbf{i}}^r(\mathbf{u}) = uP_{\mathbf{i+e1}}^{r-1}(\mathbf{u}) + vP_{\mathbf{i+e2}}^{r-1}(\mathbf{u}) + wP_{\mathbf{i+e3}}^{r-1}(\mathbf{u}) + \cdots \quad (6)$$

for $r = 1,\ldots,n$ and $|\mathbf{i}| = n-r$, where $\mathbf{e1} = (1,0,0,\cdots)$, $\mathbf{e2} = (0,1,0,\cdots)$, $\mathbf{e3} = (0,0,1,\cdots)$ and $\mathbf{u} = (u,v,w,\cdots)$. Similar to the way we derived Equation(5), we subtract the generalized de Casteljau formula of the generalized Bernstein polynomials before deformation from that after deformation, and then eliminate the displacement vector $\overrightarrow{V}$ on both sides of the equation. We obtain:

$$\alpha_{\mathbf{i}}^r(\mathbf{u}) = u\alpha_{\mathbf{i+e1}}^{r-1}(\mathbf{u}) + v\alpha_{\mathbf{i+e2}}^{r-1}(\mathbf{u}) + w\alpha_{\mathbf{i+e3}}^{r-1}(\mathbf{u}) + \cdots \quad (7)$$

for $r = 1,\ldots,n$ and $|\mathbf{i}| = n-r$.

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

3.3 Multiple Control Point Movement

When deforming an object such as in virtual sculpting [20] or character animation, there is often more than one control point moving at the same time. We have investigated the computational cost of multiple control point movement. Since the number of control points of a surface defined by Bernstein polynomials is proportional to the degree of the surface, the costs are therefore bounded by this degree. For a surface defined by piecewise polynomials, such as B-spline surfaces, although the number of control points is not bounded by the degree of the surface, this type of surfaces possesses a local modification property [21]. More explicitly, a control point $P_{i,j}$ affects only the shape of the surface within the parameter region $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$, where p and q are the degrees of the surface along u and v parameter directions, respectively. Consequently, the shape of a patch segment with the parameter range $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ is controlled by the set of control points $P_{m,n}$ where $m = i-p, \ldots, i$ and $n = j-q, \ldots, j$. Hence, when deformation occurs, we need to perform at most $(p+1) \times (q+1)$ multiplications and additions to compute the final position of a vertex within this patch segment.

From the above discussion, we may conclude that the computational cost of multiple control point movement is bounded by the degree of the surface. Because surfaces of low degrees, say 3 or 4, are usually used to model objects, the computational cost of multiple control point movement is therefore very low.

4 Incremental Crack Prevention

Since the tessellation of the deforming surface is obtained by adaptively subdividing quadrilaterals, we need to prevent crack formation along the boundaries where polygon patches are at different subdivision levels. In our previous work [19], we adopted Barsky's crack prevention method [22]. Although the method is simple, it is computationally expensive. The algorithm must traverse the polygon hierarchy starting from the root node. For each quad-tree node, we search its neighboring nodes and compare their subdivision levels. If the subdivision level of a neighboring node is lower than that of the current node, the mid-point and the end-points of the common edge are passed down to the child nodes. The vertices of these child nodes are then modified to fix the crack. This is done recursively until a sufficiently flat quad-tree node is reached.

To avoid identifying neighboring nodes and comparing subdivision levels for each quad-tree node during run-time, we introduce a new incremental crack prevention procedure. As the object moves near or further away from the viewer, the resolution of the polygon model may need to be adjusted to maintain the visual quality of the object. We mark the quad-tree nodes as SPLIT if they need to be subdivided, or as MERGE if their child nodes need to be merged, while we are performing the resolution refinement. Since cracks only occur at edges where the polygons at either side have changed resolution, we only need to check those nodes marked with SPLIT or MERGE. We have identified all possible neighboring conditions of a SPLIT or MERGE node and the corresponding actions needed to be taken to fix the cracks. They are shown in Figure 3. We define two edge fixing functions, *adjustEdge()* and *restoreEdge()*. *adjustEdge()* flattens an edge by making the middle vertex of the edge collinear with its corner vertices, while *restoreEdge()* restores the middle vertex of an edge to the position before it was adjusted by *adjustEdge()*. When the neighboring node is at a lower or the same subdivision level, *adjustEdge()* is called to update the corner vertices of the child nodes along the common edge by both the mid-point and end-points of that edge. When the neighboring node has a higher subdivision level, its polygon vertices must have previously been adjusted to prevent cracks. We therefore call *restoreEdge()* to restore the corner vertices of these polygons along the common edge to their original positions. The incremental crack prevention algorithm is summarized as follows:

- If the current node is a SPLIT node and has a higher or equal subdivision level than its neighboring node, we fix the common edge using *adjustEdge()*.
- If the current node is a MERGE node and has a lower subdivision level than its neighboring node, we fix both the common edge and the middle perpendicular edge of the neighboring node using *adjustEdge()*. We also repeat this process on the child nodes of the neighboring node along the edge.
- If the current node is a SPLIT node and has a lower subdivision level than its neighboring node, we fix the common edge of the neighboring node by *restoreEdge()*. However, we fix the middle perpendicular edge and the child nodes of the neighboring node along the common edge using *adjustEdge()*.

This algorithm works well if the object does not deform. If the object deforms, the vertices of the quad-tree nodes will be updated. Hence, the modified vertices from *adjustEdge()* for crack prevention are no longer valid. In this situation, the incremental crack prevention algorithm will not work. To solve this problem, we introduce a *fast crack prevention procedure*. We add a state variable called *edgeAdjusted* for each edge of a quad-tree node. We mark an edge as *edgeAdjusted* if it has been modified by *adjustEdge()*. To perform fast crack prevention, we traverse the polygon hierarchy and fix the edges of the quad-tree nodes marked as *edgeAdjusted* with *adjustEdge()*. This procedure is run prior to the incremental crack prevention algorithm. It is very efficient since it does not require the traversal of neighboring nodes and the comparison of their subdivision levels.

5 Parameter Caching

During run-time, many parameters are produced in different stages, such as the curvature of individual quad-tree node for resolution refinement and vertex normal for shading. Some of them are used very frequently but may change infrequently. We may cache this kind of parameters to enhance the rendering performance.

In practical situations, most deformable objects in the virtual environment may only be deforming for a short period of time once a while. They are basically rigid for the rest of the time, when, for example, the virtual actors stop their motions. At this time, some parameters of the deformable surfaces will stay unchanged and can be cached for use in successive frames. In our implementation, we choose two essential parameters for caching, the *node curvature*, which indicates if a quad-tree node needs to be split or merged, and the *vertex normals*, which are needed for shading the polygon model.

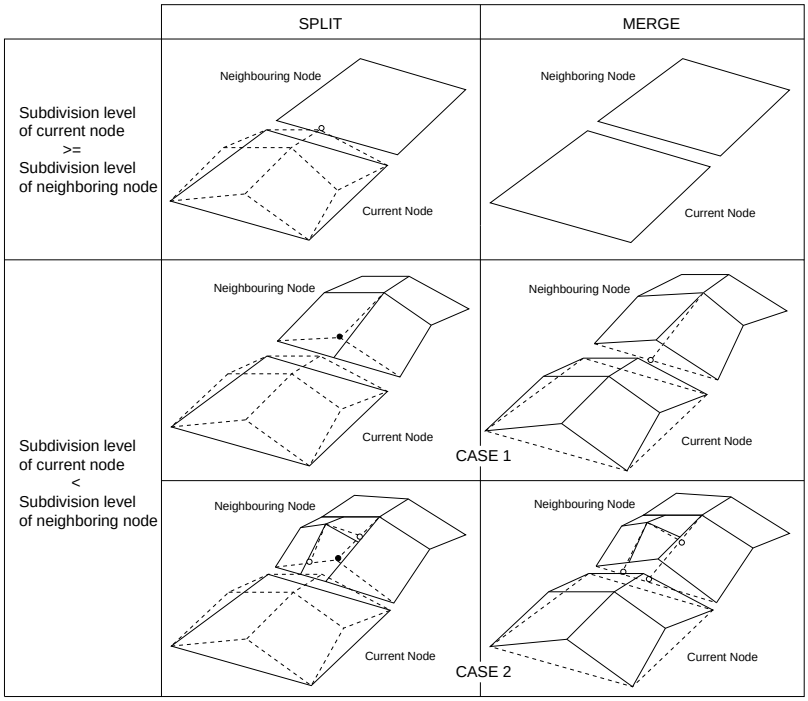
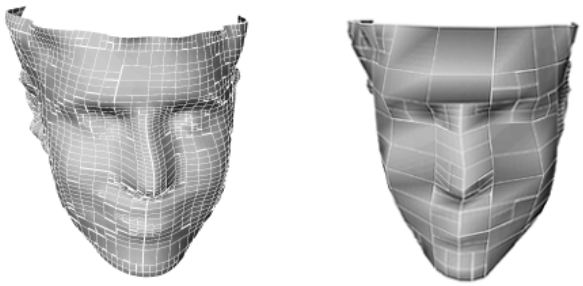


Figure 3: Crack prevention operations under different neighboring conditions.

On the other hand, when an object deforms, we need to execute a fast crack prevention procedure to fix the modified vertices of all the quad-tree nodes. This process involves a lot of operations for searching neighboring nodes and comparing subdivision levels. We may avoid these operations by caching the *edgeAdjusted* state variable in each quad-tree node. This state variable implicitly records the result of the subdivision level comparison of a quad-tree node with its neighboring nodes. Without caching this variable, the time spends on the fast crack prevention will be approximately equal to the time required for executing the Barsky's crack prevention algorithm.

6 Constructing a Hierarchical Surface

In our original method [19], resolution refinement is performed within a single surface patch only. The inter-patch resolution refinement is not considered. In practice, many objects are constructed by more than one surface patch, such as the head model in Figure 4(a), it is constructed by 400 Bézier patches. In such situation, the minimum number of polygons to represent the object is therefore 400, the number of surface patches constructing it. However, when the object is flat or too far away from the viewer, it may be more desirable to represent the object with even fewer polygons. Here we introduce a hierarchical technique to address this problem. It is by combining adjacent surface patches hierarchically to form a multi-resolution polygon model to represent the complete surface.



(a) 2171 polygons (b) 316 polygons

Figure 4: Head model before and after resolution reduction.

The construction of a hierarchical surface is a preprocessing task. It involves the construction of a surface hierarchy and a polygon hierarchy. To construct the surface hierarchy, we consider each set of 2 x 2 surface patches as a group. For each group, we merge the surface patches inside it to create a parent surface patch. This is done by assigning the corner control points of each surface patch inside the group as the control points of the parent surface patch. We may neglect the interior control points because if the surface patches are merged, the interior control points are assumed to be located within a certain tolerance from the plane formed by the corner control points. However, around the surface boundary, we may not be able to group the surface patches in a 2 x 2 manner. We can select fewer surface patches to form the parent surface patch, which will then have fewer

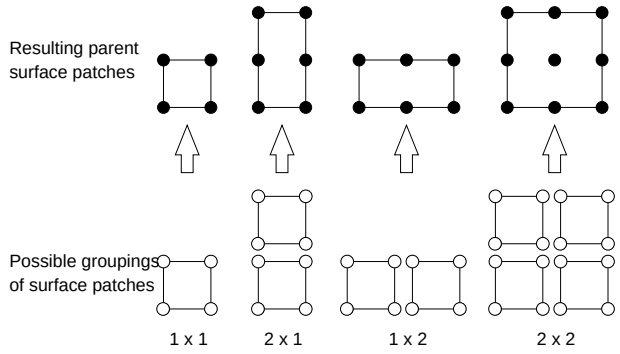


Figure 5: The construction of the parent patch. (White circles are the original control points while black circles are the final ones.)

control points. Figure 5 shows all possible parent surface patches created due to different numbers of child surface patches. In addition, while assigning the control points, we should also assign their associated deformation coefficients to the parent surface patch to allow the incremental polygon model updating technique to work with the hierarchical surface. Each newly formed parent surface patch becomes a quad-tree node. We repeat this construction process on the surface patches recursively until a single surface hierarchy is created.

To construct the polygon hierarchy, since we already have many hierarchical polygon models each representing a single surface patch, we only need to merge these hierarchical polygon models into a single polygon hierarchy. The construction process is similar to constructing the surface hierarchy. We merge each set of 2 x 2 hierarchical polygon models as a group recursively until a single root node is formed.

During run-time, if an object deforms, we incremental update the hierarchical polygon model in a bottom-up manner. For all the affected leaf nodes in the hierarchy, we update their control points with the appropriate deformation coefficients and the displacement vector of the moving control point. We then recursively perform the same operation on the parent quad-tree nodes. Since in practice, there is usually only a small subset of the hierarchy affected by the movement of a control point, the bottom-up updating is much more efficient than the top-down one. After the incremental updating, we perform resolution refinement and crack prevention to produce the final polygon model for rendering in the current frame. Figure 4(b) shows the head model rendered at low resolution using the hierarchical technique.

7 Results and Discussions

We have implemented the new method in C++ with OpenGL. We tested its performance on a SGI Onyx2 with four 195MHz R10000 CPUs and an InfiniteReality^2 graphics accelerator. However, only one processor was activated during all our experiments. We tested our method using a man face model with 400 control points and a woman face model with 1600 control points as shown in Figure 8.

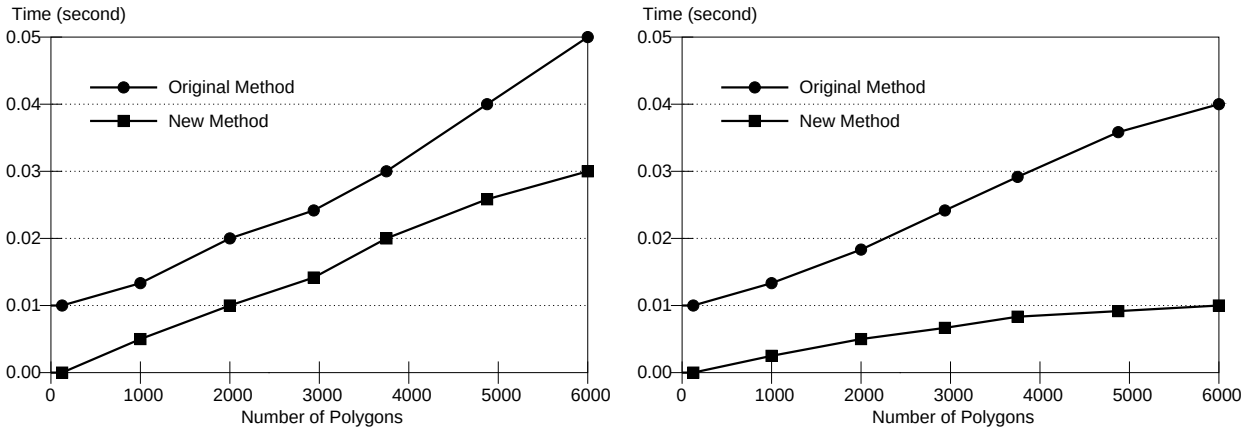
Figure 6 compares the performance of the new method with the original method [19]. We tested the run-time performances of both methods with and without object deformation as shown in figure 6(a) and 6(b), respectively. If the object deforms, both methods will perform incremental polygon model updating, resolution refinement and the crack prevention procedure(s) to generate a new polygon model to represent the deformed surface. In general, the performance of the new method is roughly 2 times higher than that of the original method. On the other hand, if the object does not deform, both methods will only perform the resolution refinement and the crack prevention procedure. (Note that we still need to perform resolution refinement because the distance or other factors of the object may change, which may affect the resolution of the surface.) The performance of the new method is now 3 times higher than that of the original method.

Figure 7 shows the comparison of the new method with our original method [19], the Kumar's method [6] and the surface evaluation method [23]. We tested the methods by moving different number of control points simultaneously. Figures 7(a) and 7(b) show the result using the man and the woman face models, respectively. The models are rendered with 3600 polygons. Results show that the new method runs faster than our original method. And the performance of the new method is bounded while that of our original method is unbounded. This improvement is due to the reduction of time spent on incremental polygon model updating. In our original method, the time spent on incremental polygon model updating increases in proportional to the number of simultaneous moving control points, while in the new method, the rendering time is limited by the degree of the NURBS surface. Our results also show that the new method runs 2 to 15 times faster than the surface evaluation method and 5 times faster than Kumar's method.

At first glance, the new method may appear to make less improvement than our original method does. However, consider the fact that in a virtual environment, there can be many deformable objects moving around. A triple in performance already implies that the system can now handle triple amount of deformable objects of same complexity or the same number of deformable objects but of higher complexities.

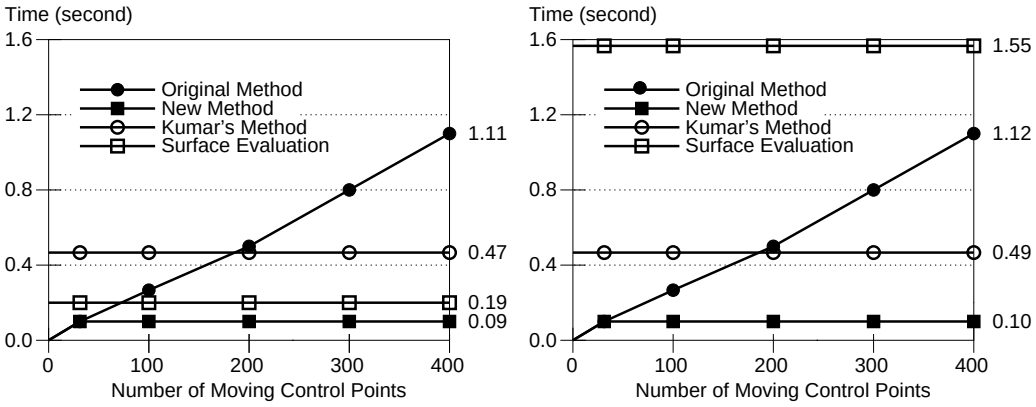
8 Conclusion

We have presented a method for interactive rendering of deforming objects modeled by different kinds of spline surfaces and multivariate objects. The new method is an extension of our earlier work on rendering of deformable NURBS surfaces. We have improved the performance of the original method by introducing techniques for incremental crack prevention and for parameter caching. In addition, the new method reduces computation time in handling multiple control point movement. We have also developed a hierarchical data structure to provide a multi-resolution representation of the object model. Our experiments have demonstrated that the new method has significant performance improvement over our original method and some of the popular methods available.



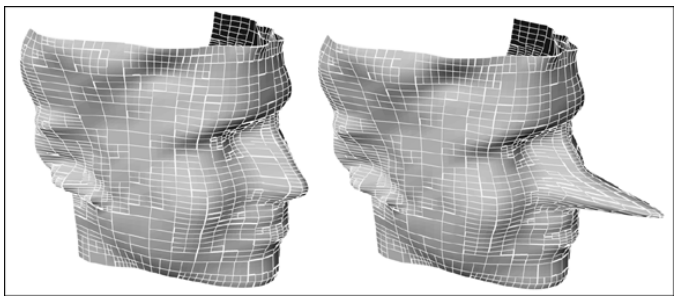
(a) With deformation (b) Without deformation

Figure 6: Performance comparison of the new method with the original method.

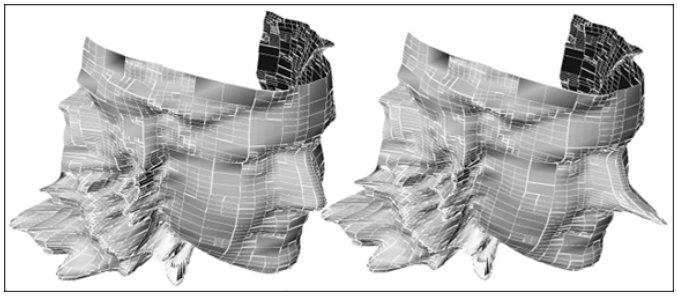


(a) (b)

Figure 7: Performance comparison of different rendering methods.



(a) Man model



(b) Woman model

Figure 8: Models used in our experiments.

References

[1] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1993.

[2] J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*. A K Peters, Ltd., 1993.

[3] S. Abi-Ezzi and L. A. Shirman, "Tessellation of Curved Surfaces Under Highly Varying Transformations," *Proceedings of Eurographics '91*, pp. 385–397, 1991.

[4] S. Abi-Ezzi and S. Subramaniam, "Fast Dynamic Tessellation of Trimmed NURBS Surfaces," *Proceedings of Eurographics '94*, pp. 108–126, 1994.

[5] D. Filip, "Adaptive Subdivision Algorithms for a Set of Bézier Triangles," *Computer-Aided Design*, pp. 74–78, 1986.

[6] S. Kumar, D. Manocha, and A. Lastra, "Interactive Display of Large-Scale NURBS Models," *Proceedings of Symposium on Interactive 3D Graphics*, pp. 51–58, 1995.

[7] S. Kumar, D. Manocha, H. Zhang, and K. Hoff, "Accelerated Walkthrough of Large Spline Models," *Proceedings of Symposium on Interactive 3D Graphics*, pp. 91–101, 1997.

[8] A. Rockwood, K. Heaton, and T. Davis, "Real-Time Rendering of Trimmed Surfaces," *Proceedings of ACM SIGGRAPH '89*, vol. 23, no. 3, pp. 107–117, 1989.

[9] E. Puppo and R. Scopigno, "Simplification, LOD and Multiresolution - Principles and Applications," *Eurographics'97 Tutorial Notes*, 1997.

[10] M. DeHaemer and M. Zyda, "Simplification of Objects Rendered by Polygonal Approximations," *Computers & Graphics*, vol. 15, no. 2, pp. 175–184, 1991.

[11] H. Hoppe, "Progressive Meshes," *Proceedings of ACM SIGGRAPH '96*, vol. 30, pp. 99–108, August 1996.

[12] H. Hoppe, "View-Dependent Refinement of Progressive Meshes," *Proceedings of ACM SIGGRAPH '97*, vol. 31, pp. 189–198, August 1997.

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

[14] D. Luebke and C. Erikson, "View-Dependent Simplification of Arbitrary Polygonal Environments," *Proceedings of ACM SIGGRAPH '97*, vol. 31, pp. 199–208, August 1997.

[15] J. Rossignac and P. Borrel, "Multi-Resolution 3D Approximations for Rendering Complex Scenes," Tech. Rep. RC 17697 (#77951), IBM Research Division, T.J. Watson Research Center, 1992.

[16] J. Xia and A. Varshney, "Dynamic View-Dependent Simplification for Polygonal Models," *Proceedings of IEEE Visualization '96*, pp. 327–334, October 1996.

[17] D. Forsey and D. Wong, "Multiresolution Surface Reconstruction for Hierarchical B-splines," *Graphics Interface '98*, pp. 57–64, 1998.

[18] D. Zorin, P. Schrőder, and W. Sweldens, "Interactive Multiresolution Mesh Editing," *Proceedings of ACM SIGGRAPH '97*, vol. 31, pp. 259–268, 1997.

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

[20] J. Wong, R. Lau, and L. Ma, "Virtual 3D Sculpturing with a Parametric Hand Surface," *Proceedings of Computer Graphics International '98*, pp. 178–186, 1998.

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

[22] B. Barsky, T. DeRose, and M. Dippé, "An Adaptive Subdivision Method With Crack Prevention for Rendering Beta-spline Objects," Tech. Rep. UCB/CSD 87/348, Dept. of Computer Science, U.C. Berkeley, 1987.

[23] C. de Boor, "On Calculating B-splines," *Journal of Approximation Theory*, vol. 6, pp. 50–62, July 1972.