EUROGRAPHICS '97 / D. Fellner and L. Szirmay-Kalos
(Guest Editors)

Volume 16, (1997), Number 3

Interactive Rendering of Deforming NURBS Surfaces

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

Computer Graphics and Media Laboratory
Department of Computing
The Hong Kong Polytechnic University, Hong Kong
{cswbli, cswhlau}@comp.polyu.edu.hk

Department of Computer Science
University of Alberta
Alberta, Canada
mark@cs.ualberta.ca

Abstract

The non-uniform rational B-splines (NURBS) has been widely accepted as a standard tool for geometry representation and design. Its rich geometric properties allow it to represent both analytic shapes and free-form curves and surfaces precisely. Moreover, a set of tools is available for shape modification or more implicitly, object deformation. Existing NURBS rendering methods include de Boor algorithm, Oslo algorithm, Shantz's adaptive forward differencing algorithm and Silbermann's high speed implementation of NURBS. However, these methods consider only speeding up the rendering process of individual frames. Recently, Kumar et al. proposed an incremental method for rendering NURBS surfaces, but it is still limited to static surfaces. In real-time applications such as virtual reality, interactive display is needed. If a virtual environment contains a lot of deforming objects, these methods cannot provide a good solution. In this paper, we propose an efficient method for interactive rendering of deformable objects by maintaining a polygon model of each deforming NURBS surface and adaptively refining the resolution of the polygon model. We also look at how this method may be applied to multi-resolution modelling.

1. Introduction

Non-uniform rational B-splines (NURBS) [1,2] have become a de-facto industry standard for representing curves and surfaces. They receive widespread acceptance and popularity because they posses many important advantages. For examples, they provide a common mathematical form for representing and designing standard analytic shapes and free-form curves and surfaces. A variety of shapes can be produced simply by manipulating the control points and weights. In addition, they have a powerful geometric toolkit including knot insertion, knot refinement and knot removal, which can be used throughout the design process. Applications of NURBS include free-form surface modelling in CAD systems, virtual reality and computer animation. In free-form surface modelling and virtual reality, interactive rendering of deforming NURBS surfaces is important. In virtual reality, in particular, we may want to model scenes with many deformable objects moving around and the efficient rendering of these objects is critical. Even in computer animation, it is an advantage to be able to preview a deforming object as some of its parameters are being modified. Because most hardware graphics accelerators can only handle polygons, the standard way to render NURBS surfaces is *polygonisation* that is tessellating a NURBS surface into polygons for rendering. Therefore, the most obvious approach to address the problem is to accelerate the polygonisation process.

In this paper, we present an efficient method for interactive rendering of deforming NURBS surfaces. The method accelerates the rendering speed with two techniques. Firstly, we consider the frame-to-frame changes of the polygon models generated in consecutive frames as the object deforms. Instead of executing the polygonisation process in every frame, we maintain a polygon model of the deforming

NURBS surface and incrementally update the polygon model to represent the deforming NURBS surface at each individual frame. Secondly, as the NURBS surface deforms and the polygon model is updated incrementally, some parts of the polygon model may become too coarse to represent the NURBS surface due to the increase in local curvatures. This will result in a polygonal appearance of the surface. On the other hand, other parts of the polygon model may become too fine to represent the surface due to the decrease in local curvatures. This, on the contrary, will result in a waste of processing time. To overcome this problem, we suggest a technique for incrementally refining the resolution of the pre-generated polygon model. We also discuss how this technique can be used in multi-resolution modelling.

The rest of the paper is organised as follows. Section 2 describes previous work on rendering NURBS surfaces. Section 3 presents the incremental technique for rendering deforming NURBS surfaces and section 4 presents the technique for resolution refinement of the pre-generated polygon model. Section 5 discusses how our method can be used in multi-resolution modelling. Section 6 shows some experimental results and discusses the advantages and limitations of our method. Finally, section 7 presents conclusions of our work and discusses some possible future work.

2. Related Work

Many methods have been proposed to speed up the process of rendering NURBS surfaces [3,4,5,6,7,8,9,10]. In order to make use of the polygon rendering capabilities of existing hardware graphics accelerators, the majority of the methods developed are based on polygonisation. For example, the recursive de Boor algorithm [6] evaluates a NURBS surface in parameter space. Boehm in [4] proposed a subdivision method using knot insertion. Cohen et al. [5] also proposed a similar subdivision method called the Oslo algorithm. This method divides a surface into polygons by simultaneously inserting multiple knots into the surface. However, it performs a large number of knot insertion operations, which are computationally very expensive. A better method is to convert the NURBS surface into Bézier patches by inserting multiple knots into each interior knot of the surface followed by either de Casteljau's [11] or Horner's [12] Bézier subdivision method. Here, fewer knots are needed to be inserted. The Mesa 3D graphics library [13] is an implementation of such technique. Other methods proposed to accelerate the polygonisation process, such as Shantz's adaptive forward differencing algorithm [9] and Silbermann's high speed implementation of NURBS [10], are based on simplifying the evaluation of the NURBS equations. All the methods mentioned above concentrate on accelerating the polygonisation of individual NURBS surfaces without concern for their incremental evolution in successive frames. Hence, the complete polygonisation process is executed in every frame.

Kumar et al. in their recent paper [14] proposed an incremental method for displaying NURBS surfaces. The method generates a polygon model from the NURBS surface by converting the surface into Bézier segments followed by a Bézier subdivision process. The method uses the viewpoint as a function of coherence between successive frames for the polygonisation of the NURBS surface. Since there is usually only a small change in view point between two consecutive frames, the method minimises the number of triangles generated within the given time frame by incrementally polygonising segments that will become visible in the next frame and deleting those that will become invisible. However, this method does not consider the situation when a NURBS surface is continuously deforming. In such situation, the complete polygonisation process still needs to be executed. As such, this method is more suitable for displaying static NURBS surfaces.

3. Incremental NURBS Surface Deformation

As mentioned earlier, existing NURBS rendering methods do not consider the coherence of a deforming NURBS surface between two successive frames, and the complete polygonisation process is executed in every frame. If there are a lot of deforming objects in the scene and in particular, if a fine tessellation is required, these rendering methods may be too costly to execute in real-time. The objective of

this work is to speed up the rendering of deforming NURBS surfaces. The main idea is to maintain two representations of the NURBS surface, the surface model itself and the polygon model approximating the surface model. As the surface deforms, the polygon model is not regenerated. Instead, all the polygon vertices in the original polygon model are updated, such that the updated model approximates the deformed surface model.

3.1 Updating the Polygon Model

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

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} P_{i,j} R_{i,j}(u,v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} R_{i,j}(u,v)} \qquad (1)$$

where $w_{i,j}$ are the weights, $P_{i,j}$ form a control net, and $R_{i,j}(u,v)$ are the basis functions.

A polygonal representation of the NURBS surface can be obtained by evaluating equation (1) with some fixed u and v parameters. If a control point $P_{s,t}$ is moved to $\overline{P_{s,t}}$, the displacement vector can then be defined as $\vec{V} = \overline{P_{s,t}} - P_{s,t}$. The incremental difference between the two polygonal representations of the surface before and after the control point movement can be shown as follows:

$$\begin{aligned}\overline{S}(u,v) - S(u,v) &= \frac{(\overline{P_{s,t}} - P_{s,t}) w_{s,t} R_{s,t}(u,v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} R_{i.j}(u,v)} \\ &= \alpha_{s,t}\vec{V}\end{aligned}$$

where $S(u,v)$ and $\overline{S}(u,v)$ are the polygonal representations of the surface before and after the control point movement respectively. $\alpha_{s,t}$ is a constant for each particular set of u and v parameter values and defined as:

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

If the subdivision resolution remains the same before and after the deformation, all the scalar coefficients $\alpha_{s,t}$ can be pre-computed. Therefore, the cost of updating a polygon vertex from the current frame to the next is only two operations, one multiplication and one addition. In addition, with the *local control property* of NURBS, it is known that the translation of a particular control point $P_{s,t}$ of a NURBS surface would only affect the region $[u_s, u_{s+q}) \times [v_t, v_{t+q})$ where q is the order of the NURBS surface. Hence, we only need to update the polygon vertices within this local region to minimise the polygon model update time.

3.2 The Incremental Rendering Process

Our rendering process can be divided into two phases, *the initialisation phase* and *the polygon model update phase*.

- **Initialisation Phase**: During this phase, we perform a polygonisation process to produce the initial polygon model of the NURBS surface. We also calculate the scalar coefficients for all the control points and store them for subsequent deformation. However, if memory is tight, the computation of the scalar coefficients may be performed on demand upon the movement of one or more control points and the computed scalar coefficients are then stored for subsequent deformation.
- **Polygon Model Update Phase**: During this phase, we update all the polygon vertices affected by the displaced control point by multiplying the displacement vector with the pre-computed set of scalar coefficients. When the deformation involves multiple control points, we may consider each of them in sequence.

4. Resolution Refinement

With the technique proposed in Section 3 above, we may now render a deforming NURBS surface very efficiently. However, since the polygon model representing the NURBS surface is pre-generated, the resolution of the polygon model is fixed throughout the deformation. The problem with this is that if the curvature of the surface is increased by a large amount during a deformation process, the resolution of the polygon model may become too coarse to represent the deformed surface and polygonal appearance may result. On

the other hand, if the curvature of the surface is decreased by a large amount during a deformation process, the resolution of the polygon model may become higher than necessary to represent the deformed surface. In this situation, many polygons may be projected to a single pixel resulting in a waste of processing time. To overcome this problem, we propose a technique for inserting or deleting polygons incrementally according to the change in local curvature of the surface. Interactive frame rates can therefore be maintained by avoiding complete re-evaluation for the new polygon vertices. The method is based on the de Casteljau algorithm [11].

4.1 Insertion and Deletion of Polygons

To increase the resolution of a region, we divide a polygon (or more precisely, a square) into four smaller polygons. The major computational cost of this operation is to create a new set of scalar coefficients for the new vertex. However, calculating the scalar coefficients with equation (2) is expensive. Hence, we derive a new technique which generates new scalar coefficients incrementally from the existing scalar coefficients instead of re-calculating them with equation (2). This technique minimises the overhead for resolution refinement.

After a NURBS surface is decomposed into Bézier patches, a polygon model can then be obtained by applying the de Casteljau subdivision method [12] on each of the Bézier patches. For simplification, we derive the formula by considering a rational Bézier curve. The derivation of the formula for rational Bézier patches goes in a similar way.

Let $P_0, P_1, \ldots, P_n \in \mathbf{R}^4$ and $0 \le u \le 1$, where $P_0, P_1, \ldots, P_n$ are the homogeneous Bézier points of a rational Bézier curve of degree *n*. The Bézier curve can be defined by the de Casteljau Algorithm as follows:

$$P_i^r(u) = (1-u)P_i^{r-1}(u) + uP_{i+1}^{r-1}(u) \quad \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n-r \end{cases} \tag{3}$$

where $P_i^0(u) = P_i$. Then $P_0^n(u)$ is the point with parameter value *u* on the Bézier curve P^n.

Now, let us assume a Bézier curve segment has *r - 1* level of refinement and a further subdivision step is applied on it with *u = t (where $0 \le t \le 1$)*. Then, we will obtain two Bézier curve segments:

$$P_i^r(t) = (1-t)P_i^{r-1}(t) + tP_{i+1}^{r-1}(t), \text{ and}$$
$$P_{n-i}^r(t) = (1-t)P_{n-i}^{r-1}(t) + tP_{n-i-1}^{r-1}(t)$$

where $P_i^r(t)$ and $P_{n-i}^r(t)$ are the Bézier points of the first and second curve segments respectively. These two curve segments behave the same in the refinement process except that they have different parameter ranges.

After the surface deformation, we have $\overline{P_0^{r-1}}(u), \overline{P_1^{r-1}}(u), \ldots, \overline{P_n^{r-1}}(u)$ as the Bézier points of the original Bézier curve and the new positions of $P_i^r(t)$ and $P_{n-i}^r(t)$ can be calculated as:

$$\overline{P_i^r(t)} = (1-t)\overline{P_i^{r-1}}(t) + t\overline{P_{i+1}^{r-1}}(t), \text{ and}$$
$$\overline{P_{n-i}^r}(t) = (1-t)\overline{P_{n-i}^{r-1}}(t) + t\overline{P_{n-i-1}^{r-1}}(t)$$

The movement of $P_i^r(t)$ and $P_{n-i}^r(t)$ are then expressed as:

$$\overline{P_i^r(t)} - P_i^r(t)$$
$$= (1-t)\left(\overline{P_i^{r-1}}(t) - P_i^{r-1}(t)\right) + t\left(\overline{P_{i+1}^{r-1}}(t) - P_{i+1}^{r-1}(t)\right)$$

and

$$\overline{P_{n-i}^r}(t) - P_{n-i}^r(t)$$
$$= (1-t)\left(\overline{P_{n-i}^{r-1}}(t) - P_{n-i}^{r-1}(t)\right) + t\left(\overline{P_{n-i-1}^{r-1}}(t) - P_{n-i-1}^{r-1}(t)\right)$$

We can replace the movement of control points with the scalar coefficients and displacement vectors as follows:

$$\alpha_i^r(t)\vec{V} = (1-t)\alpha_i^{r-1}(t)\vec{V} + t\alpha_{i+1}^{r-1}(t)\vec{V}, \text{ and}$$
$$\alpha_{n-i}^r(t)\vec{V} = (1-t)\alpha_{n-i}^{r-1}(t)\vec{V} + t\alpha_{n-i-1}^{r-1}(t)\vec{V}$$

Finally, we may generalise the above two equations for the scalar coefficients to the de Casteljau form:

$$\alpha_i^r(u) = (1-u)\alpha_i^{r-1}(u) + u\alpha_{i+1}^{r-1}(u) \quad \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n-r \end{cases} \tag{4}$$

Therefore, the set of new scalar coefficients can be calculated by the combination of adjacent

scalar coefficients stored at existing vertices. Instead of applying the de Casteljau scheme, in our implementation, we re-express equation (4) in monomial form by the Horner's subdivision scheme as follows to obtain better performance.

$$\alpha_i^r = (\cdots((\binom{n}{0}(1-u)\alpha_0^0 + \binom{n}{1}u\alpha_1^0)(1-u) + \binom{n}{2}u^2\alpha_2^0)(1-u)+\cdots)(1-u)+\binom{n}{n}u^n\alpha_n^0$$

The reason for this is that de Casteljau scheme requires $O(N^2)$ computation while Horner's scheme requires $O(N)$ only.

In contrast, when the resolution of a region is to be reduced because of the decrease in local curvature, we can apply the de Casteljau algorithm in reverse to construct the regressed Bézier segments. We cannot simply delete the extra vertices, because it will affect the integrity of the Bézier segments.

With this technique, we can now modify the resolution of the pre-generated polygon model dynamically. In practice, due to the frame-to-frame coherence, the number of polygon vertices needed to be inserted or deleted from the polygon model between two consecutive frames is expected to be small most of the time.

4.2 The Curvature Test

Because we use a pre-generated polygon model to approximate a deforming NURBS surface, we should therefore make sure that the approximation is precise enough at any time while the surface is deforming. To increase the precision of the approximation, regions of higher curvature should be represented with more polygons. On the other hand, to save processing time, regions of low curvature may be represented with fewer polygons. To determine the curvature of a surface, we measure the deviation of the surface from a planar polygon. The result of this curvature test is then used as an indicator to quantify the precision of our approximation. It may also be used as a stopping criterion for subdividing the NURBS surface.

In our subdivision process, the NURBS surface is firstly converted to Bézier patches, which are then adaptively subdivided into quadtree nodes based on their local curvatures. The curvature test is carried out in the following steps starting from the root patch of the quadtree:

1) We check the flatness of each boundary curve of the Bézier patch by determining whether the control points of the curve lie within a given tolerance from the line segment formed by joining the two ending control points. If a curve is found to fail in the test, we subdivide the patch and repeat the check on the sub-patches.

2) If the flatness of all the boundary curves is within the specified tolerance, we then check the flatness of the Bézier patch by determining whether the total distance of all the control points of the patch from the plane defined by any three corner points of the patch is within a certain tolerance. If it is, the patch is considered as sufficiently flat and no further subdivision is required. If it is not, we subdivide the patch and repeat the check on the sub-patches.

3) If the patch passes both tests 1 and 2 above, we output the approximating planar rectangle formed by the four corner vertices.

4.3 Crack Prevention

When adaptively refining the resolution of a polygon model, it is possible that neighbouring polygon patches may not be at the same resolution level as shown in Figure 1(a). The additional vertices inserted at the boundaries between high and low resolution polygon patches can cause cracks to appear between the patches. To solve this problem, we adopt Barsky's crack prevention method [15] to ensure the visual continuity of these patches. The idea is that cracks can be prevented if the shared boundaries of neighbouring polygon patches become flat at the same resolution level, and they can use the same line segment to represent the boundary. Hence, this line segment is passed down to any sub-patches sharing the boundary. When a polygon patch is to be subdivided, the flat boundary segment is used to "correct" the locations of the inserted vertices to prevent cracking as shown in Figure 1(b).

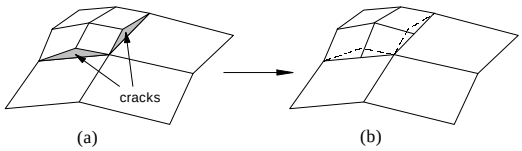

Figure 1. *Crack prevention*.

5. Multi-Resolution Modelling

Because a distant object occupies a relatively smaller screen area than a nearby object, most of the details in the distant object are not visible to the viewer. Multi-resolution modelling is a technique to optimise the rendering performance by representing a nearby object with a detailed model, but a distant object with a simpler one. To determine the resolution of an object model, our recent work [16] identifies two importance factors: the *static visual importance* (SVI) and *the dynamic visual importance* (DVI). SVI refers to the visual importance of a point on an object calculated according to its geometric importance. For example, a point at a corner of an object has a higher SVI value than a point on a flat surface. DVI refers to the visual importance of a point on an object calculated according to some dynamic viewing and animation parameters. For example, a region of an object has a higher DVI value if it intersects the viewer's line of sight.

Although there are many methods for generating multi-resolution models [17,18,19,20, 21], most of them are slow and some of them may not even preserve the geometry of the object model. To generate accurate models without sacrificing performance, a common method is to pre-generate a few key models of the object at different resolutions. The run-time object distance determines which model to use for rendering. Although this method is fast and simple, it has two major limitations. Firstly, when the object crosses the threshold distance, there is a sudden change in model resolution and an objectionable visual discontinuity effect can be observed. In [21], Turk proposed to have a transition period during which a smooth interpolation between the two successive models is performed to produce models of intermediate resolutions. This method, however, further increases the computational cost during the transition period because of the need to process two models at the same time. Secondly, because the multi-resolution models are pre-generated, the object's DVI is not considered. The result of this is a uniform increase or decrease in model resolution, and hence the generated models are not optimised for the given viewing and animation parameters of individual frames. Such optimisation is important when an object covers a large depth range, for example, a piece of landscape or a large building. Even though there may only be a small region of the object lying close to the viewer or inside the viewer's line of sight, we still need to use the high resolution model for rendering so that the details in the closest part of the object will not be lost. Sometimes, a small object may also have similar problems if, for example, it is close to the viewer.

To overcome these limitations, related methods for managing large terrain models can be used [22,23,24]. These methods basically divide a large terrain surface into square blocks. All the polygons inside a block are arranged in a quadtree structure with the root node representing the whole block and the leaf nodes representing individual polygons. Each successive lower level of the tree represents a four-time increase in the number of polygons and two-time increase in resolution. With such a data structure, it is possible to calculate the DVI values of individual high level nodes during run-time. A node with high DVI value can be rendered using its lower level subnodes, i.e., with more details, while a node with low DVI value can be rendered using its higher level nodes. However, this kind of method requires the object to be regularly subdivided. In [25], Hoppe proposed an idea called progressive meshes. This method can be used to perform limited selective refinement by considering the DVI of the object. Our recent work [16] subdivides each object into regions to allow more flexible selective refinement with better performance. However, both of these methods do not consider deforming objects.

To apply our method for rendering deforming NURBS surfaces to multi-resolution modelling, we need to be able to determine the resolution of the deforming surface according to its SVI and DVI. The importance of a Bézier patch, *i*, can be calculated as follows:

$$P_{i,imp} = P_{i,SVI} \bullet P_{i,DVI}$$

where $P_{i,SVI}$ and $P_{i,DVI}$ are the SVI and DVI values of patch i respectively. This patch importance value, $P_{i,imp}$, can be used to determine the actual subdivision level, and hence the resolution, of patch *i*. The higher this value is, the more subdivisions will need to make to the patch.

When the surface is not deforming, $P_{i,SVI}$ remains unchanged. When the surface is deforming, the SVI value of the Bézier patch, *i*, can be updated from the result of the curvature test, R_i, described in section 4.2 as follows:

$$P_{i,SVI} = K_{SVI} \bullet R_i$$

where K_{SVI} is a constant. The DVI value of the patch can be calculated according to some real-time viewing and animation parameters. For example, if we consider the distance of the patch from the viewer and the distance of the patch from the viewer's line of sight, the DVI value of the patch can be calculated as follows:

$$P_{i,DVI} = I_{dist} \bullet I_{sight}$$

where I_{dist} and I_{sight} are the importance values (between zero and one) based on the distances of the patch from the viewer and from the viewer's line of sight respectively. The actual calculation of these and other parameters can be found in many references [26,16,27].

6. Results and Discussions

We implemented the new method in OpenGL and with the Manchester NURBS library [28]. We compared the performance of the new method with two other polygonisation methods on a SGI Indigo^2 workstation with a 200MHz MIPS 4400 CPU and Extreme graphics accelerator. The two polygonisation methods are the Oslo algorithm and the method used in the Mesa graphics library, which is converting the NURBS surface into Bézier segments followed by Horner's subdivision. Figure 2 shows the results of the comparison. Curve (a) shows the performance of the Oslo algorithm. Curve (b) shows the performance of the method used by the Mesa graphics library. Curves (c) and (d) show the performances of our method with and without resolution refinement respectively. Compared to the method used by the Mesa graphics library, our method without resolution refinement is roughly 15 times faster, and is roughly 3 times faster with resolution refinement when the total number of polygons used to represent the surface changes by about 2% due to changes in refinement. The extra computational cost is due to the curvature tests and the calculation of the scalar coefficients for the inserted vertices. Note that the performance shown in figure 2 does not include the polygon rendering time.

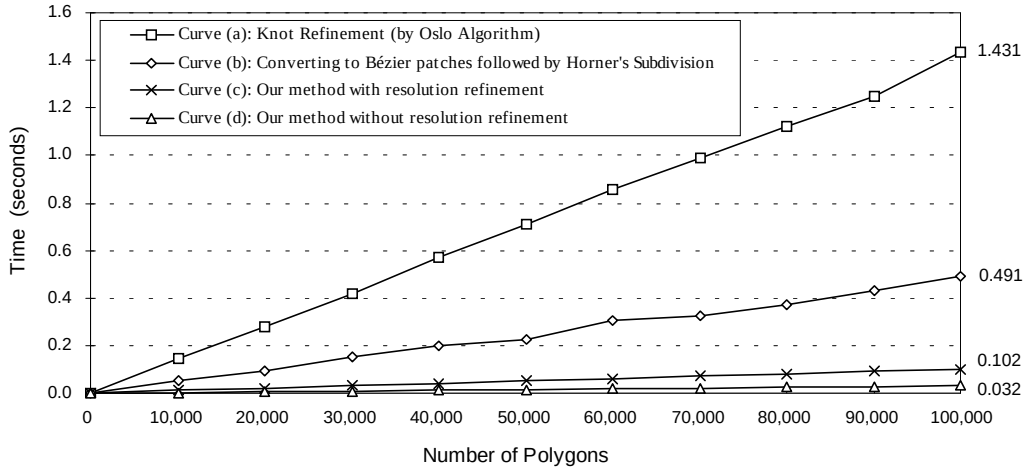

Figure 2. *Performance comparison of different NURBS rendering.*

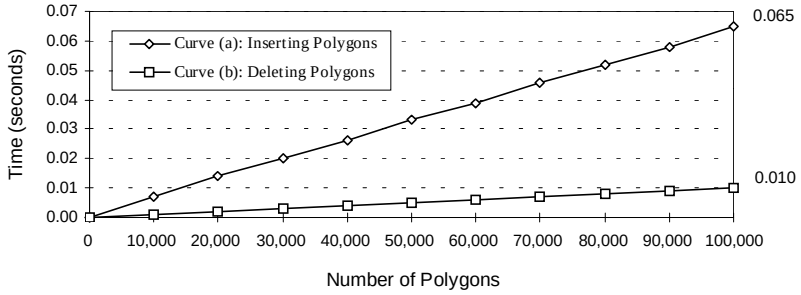

Figure 3. *Performance of polygon insertion and deletion.*

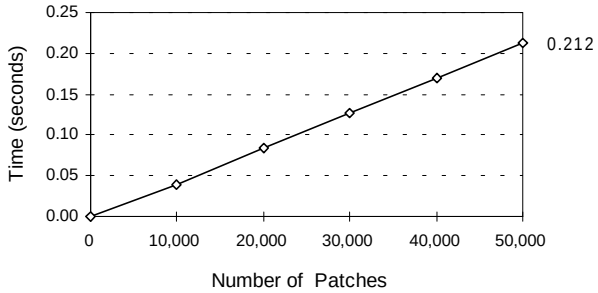

Figure 4. *Performance of the curvature test.*

Figure 3 shows the performance of inserting polygons and deleting polygons from the model. From the diagram, we can see that it is more expensive to insert polygons than to delete polygons from the model. This is because more vertices are involved in calculating the sets of scalar coefficients when inserting vertices than when deleting vertices. Figure 4 shows the performance of the curvature test with respect to the number of Bézier patches being tested.

Figure 5 shows some images generated with our method. Figures 5(a) and 5(b) are generated without resolution refinement. While the flat regions have too many polygons, the regions of high curvature may have too few. Figures 5(c) and 5(d) are generated with resolution refinement and contain the same total number of polygons as figures 5(a) and 5(b). We can see that figure 5(d) has a much smoother appearance than figure 5(b). Figures 5(e), 5(f), 5(g) and 5(h) are generated when the surface is at some distance away from the viewer. Figures 5(e) and 5(f) give a closer look of the surface while figures 5(g) and 5(h) show the actual size of the surface.

In practice, as the surface deforms, we need to update the positions of all affected polygon vertices and perform the curvature test on all affected patches during every frame time. However, we may not need to insert or delete polygons in every frame. Even if we need to modify the resolution of a patch, there are usually only a small number of insertions or deletions in a single frame. As such, our method can be very efficient. The major limitation of our method is that, because the polygon model is arranged in a quadtree structure, cracks may appear at the boundaries between patches of different resolutions, and extra processing time is needed to prevent this to happen. We are now working on another subdivision method which can eliminate crack prevention operation.

7. Conclusions

We have presented a method for interactive rendering of deforming objects modelled by NURBS surfaces. The method makes use of the incremental property of the deforming surface in successive frames by updating the geometry and refining the resolution of a polygon model pre-generated from the deforming NURBS surface instead of executing the polygonisation process in every frame. Results show that our method is more efficient than existing methods.

We are currently developing a set of tools to be incorporated into the MR Toolkit [29] to support the rendering of deformable objects. We are also considering adding some animation support based on the method described in this paper.

8. References

1. L. Piegl, "On NURBS: A Survey", *IEEE Computer Graphics & Applications*, pp. 55-71, Jan.1991.
2. L. Piegl and W. Tiller, *The NURBS Book*, Springer-Verlag, 1995.
3. S. S. Abi-Ezzi and L. A. Shirman, "Tessellation of Curved Surfaces under Highly Varying Transformations", *Eurographics'91*, pp.385-397, 1991.
4. W. Boehm, "Inserting New Knots into B-spline Curves", *Computer Aided Design*, **12**(4), pp.199-201, 1980.
5. E. Cohen, T., Lyche and R. F. Riesenfeld, "Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics", *Computer Graphics and Image Processing*, **14**, October 1980.
6. C. de Boor, "On Calculating B-splines", *Journal of Approximation Theory*, **6**(1), pp.50-62, July 1972.
7. J. M. Lane, L. C. Carpenter, J. T. Whitted and J. F. Blinn. "Scan Line Methods for Displaying Parametrically Defined Surfaces", *CACM*, **23**(1), pp.23-24, 1980.
8. T. Nishita, T. W. Sederberg and M. Kakimoto. "Ray Tracing Trimmed Rational Surface Patches", In *ACM SIGGRAPH'90*, **24**(4), pp. 337-345, 1990.
9. M. Shantz and S. L. Chang, "Rendering Trimmed NURBS with Adaptive Forward Differencing", In *ACM SIGGRAPH'88*, pp.189-198, August 1988.
10. M. J. Silbermann, "High Speed Implementation of Nonuniform Rational B-Splines (NURBS)", *SPIE Vol. 1251 Curves and Surfaces in Computer Vision and Graphics (1990)*, pp.338-345, 1990.
11. Paul de Casteljau, "Courbes et Surfaces à Pôles", *S. A. André Citroen*, Paris, 1959.
12. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*, Academic Press Inc., 1990.
13. P. Brain, "The Mesa 3-D Graphics Library", *URL http://www.ssec.wisc.edu/~brianp/Mesa.html.*
14. S. Kumar, D. Manocha and A. Lastra, "Interactive Display of Large-Scale NURBS Models", *Proceedings of 1995 Symposium on Interactive 3D graphics*, pp. 51-58, 1995.
15. B. A. Barsky, T. D. DeRose and M. D. Dippé, "An Adaptive Subdivision Method With Crack Prevention for Rendering Beta-spline Objects", *Report UCB/CSD 87/348*, Dept. of Computer Science, U.C. Berkeley, 1987.
16. R. Lau, D. To and M. Green, "An Adaptive Multi-Resolution Modeling Technique Based on Viewing and Animation Parameters", *IEEE VRAIS'97*, pp.20-27, March 1997.
17. M. DeHaemer and M. Zyda. "Simplification of Objects Rendered by Polygonal Approximations", *Computers & Graphics*, **15**(2), pp.175-184, 1991.
18. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle. "Mesh Optimization", In *ACM SIGGRAPH'93*, **27**, pp.19-26, August 1993.
19. J. Rossignac and P. Borrel. "Multi-Resolution 3D Approximations for Rendering Complex Scenes", *Technical Report RC 17697 (#77951)*, IBM Research Division, T.J. Watson Research Cener, 1992.
20. W. Schroeder, J. Zarge and W. Lorensen. "Decimation of Triangle Meshes", In *ACM SIGGRAPH'92*, **26**, pp.65-70, July 1992.
21. G. Turk, "Re-tiling Polygonal Surfaces", In *ACM SIGGRAPH'92*, **26**, pp.55-64, July 1992.
22. J. Falby, M. Zyda, D. Pratt and R. Mackey. "NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation", *Computers & Graphics*, **17**(1), pp.65-69, 1993.
23. B. Herzen and A. Barr. "Accurate Triangulations of Deformed, Intersecting Surfaces", In *ACM SIGGRAPH'87*, **21**, pp.103-110, July 1987.
24. P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust and G. Turner. "Real-Time, Continuous Level of Detail Rendering of Height Fields", In *ACM SIGGRAPH'96*, pp. 109-118, August 1996.
25. H. Hoppe. "Progressive Meshes", In *ACM SIGGRAPH'96*, pp.99-108, August 1996.
26. T. Funkhouser and C. Séquin. "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments", In *ACM SIGGRAPH'93*, **27**, pp. 247-254, August 1993.
27. T. Ohshima, H. Yamamoto and H. Tamura. "Gaze-Directed Adaptive Rendering for Interacting with Virtual Space", In *IEEE VRAIS'96*, pp.103-110, July 1996.
28. W. T. Hewitt and D. Yip. *The NURBS Procedure Library*. Technical report, Computer Graphics Unit, University of Manchester, 1992.
29. University of Alberta, "MR Toolkit", *URL http://www.cs.ualberta.ca/~graphics/MRToolkit.html.*

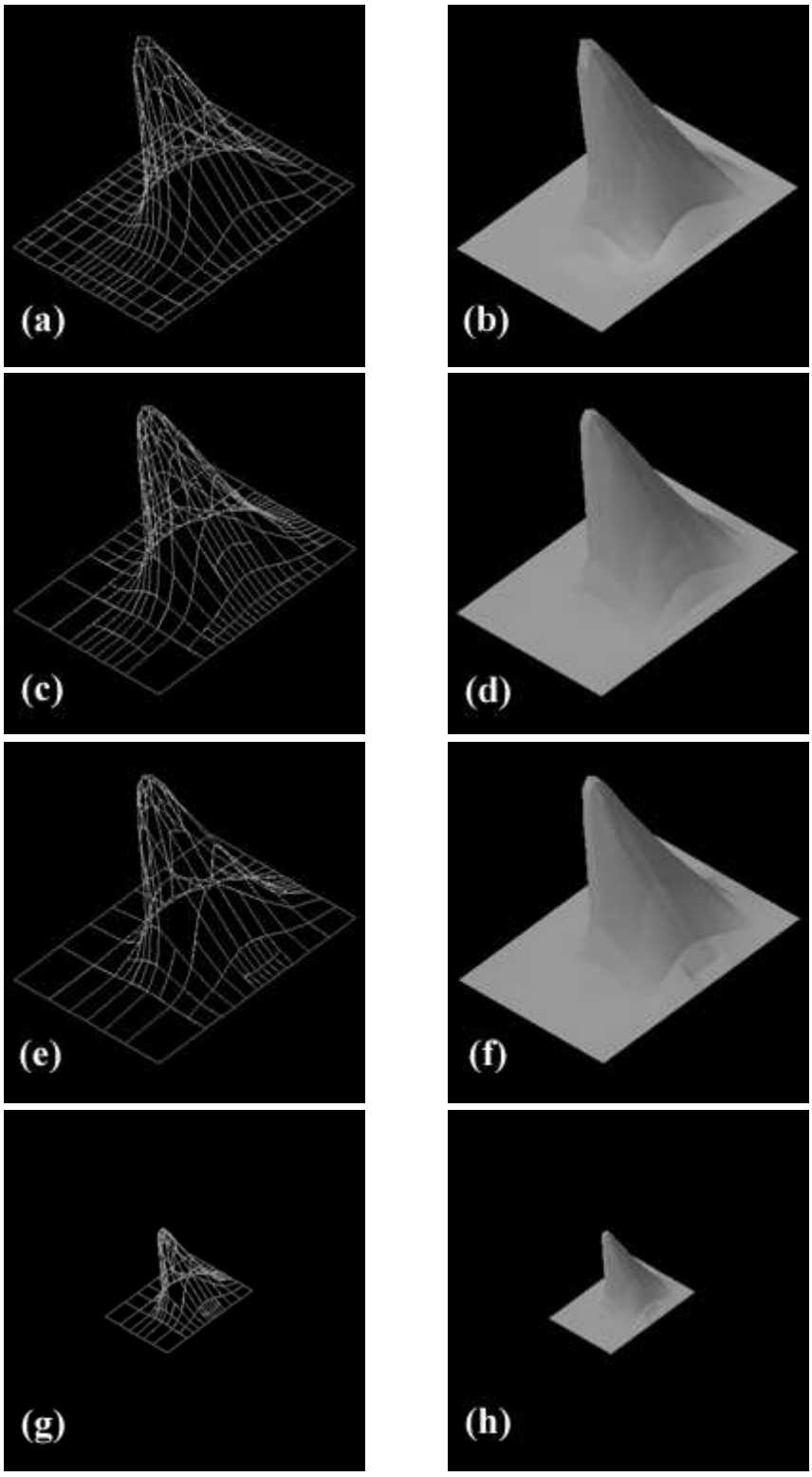

Figure 5. *Images generated with our method.*