An Alternative Methodology to Represent B-spline Surface for Applications in Virtual Reality Environment

Harish Pungotra[1], George K. Knopf[2] and Roberto Canas[3]

[1]Beant College of Engineering and Technology, Gurdaspur, India hpungotr@alumni.uwo.ca
[2]The University of Western Ontario, London, Ontario, Canada gknopf@eng.uwo.ca
[3]National Research Council, London, Ontario, Canada roberto.canas@nrc-cnrc.gc.ca

ABSTRACT

B-spline representation is one of the main methods for free-form surface modeling and has become the standard for CAD systems. However, in Virtual Reality (VR) environment, when a B-spline surface deforms, the blending functions need to be continuously computed. The high computational cost of continuously calculating the blending functions for merging, collision detection and physics-based deformation system, while the model is deforming, restricts the use of B-spline representation in a VR environment. This paper presents an alternative methodology to represent B-spline surface patches for an interactive VR environment. A uniformly discretized B-spline surface patch can be represented by a set of control points and two pre-calculated B-spline blending matrices. The proposed technique exploits the fact that these B-spline blending matrices are independent of the position of control points and therefore can be pre-calculated. The blending matrices enable the algorithm to merge B-spline surface patches, accurately check the collision, and generate nodes for the mass spring system to determine deformation using the physics-based model. This technique does away with the need to calculate computationally intensive blending functions for the B-spline surfaces, and inverse of large matrices during the run-time.

Keywords: collision detection, deformable objects, physics-based modeling.
DOI: 10.3722/cadaps.2013.711-726

1 INTRODUCTION

The rapid advancement of virtual reality technology has led to the development of a variety of applications in computer graphics, gaming and entertainment, engineering analysis, surgical training, and interactive design. Several studies have suggested [4, 23, 25, 26] that virtual reality promises to be a very intuitive, creative and cost effective method for concept design. Computer Graphics (CG) and Computer Aided Design (CAD) prefer parametric surface representations, since these can represent complex surfaces with simplicity and minimal storage [17]. Another advantage of this type of representation is that the points on these parametric surfaces can be determined for any given value of parameter u and v. A tensor product parametric surface with B-spline blending functions is the most commonly used parametric representation in modern CAD software. Thus it is imperative that any Virtual Reality (VR) based concept design system should use the B-spline representation for seamless exchange between CAD and VR systems.

A B-spline surface with r and s number of control points in u and v directions, respectively, is given by the equation,

$$S(u,v)=\sum_{i=0}^{r-1}\sum_{j=0}^{s-1}B_{ik}(u)\,B_{jl}(v)\mathbf{P}_{ij} \tag{1}$$

where $\mathbf{P}_{ij}$ is the control points vector, $B_{ik}(u)$ and $B_{jl}(v)$ are the blending functions of the surface with order k and l in u and v directions, respectively. The blending functions depend upon the periodicity of the surface (in u and/or v directions), the knot vector, the degree of the surface in u and v directions, and the number of control points in u and v directions. The blending functions are independent of the position of the control points. The B-spline blending function has the property of recursion which is defined as:

$$B_{i,k}(u)=(u-u_i)\frac{B_{i,k-1}(u)}{u_{i+k-1}-u_i}+(u_{i+k}-u)\frac{B_{i+1,k-1}(u)}{u_{i+k}-u_{i+1}} \tag{2}$$

where $B_{i,1}=\begin{cases}1, & u_i\le u\le u_{i+1}\\ 0, & otherwise\end{cases}$

The blending function for a periodic B-spline is modified as:

$$B_{i,k}(u)=B_{0,k}((u-i+n+1)\bmod(n+1)) \tag{3}$$

A virtual reality-based interactive product concept design system must support visual object collision detection, physics-based modeling, the merging of surfaces representing the virtual objects, and haptic manipulation. The collision detection algorithm provides detailed information about when and how multiple virtual objects make contact and interact within the VR space. The physics-based system uses this information to determine the deformation and the reactive forces to be fed back to the user. At the same time, another essential aspect in B-spline surface modeling is the merging of the B-spline surfaces to create interesting and complex design ideas. Several tools have been presented in literature, for collision detection, physics based modeling, and merging of B-spline patches that can be used in the virtual concept design process.

The collision detection problem has been extensively researched in published literature [12, 14]. Collision detection for a B-spline surface-based deformable model is, in general, computationally intensive. It involves continuously calculating the blending functions and new equations of the deforming B-spline surface. This is the main reason why there are not many collision detection algorithms available in literature for B-spline surface-based deformable objects. Of all the collision detection algorithms for B-spline models, the majority is confined to the use of a point-based tool [5, 15]. Gao and Gibson [7] presented an algorithm which can detect collision between a B-spline model and an implicit surface-based tool. However, this algorithm is limited to implicit surface-based tools and cannot be used for collision detection between two B-spline surfaces or between a B-spline model and a point-based tool.

The deformation of the model can be simulated by a geometric- or physics-based system. There are many geometric modeling techniques in the published literature to deform a solid model [1, 28]. One major limitation of the geometric models is that these cannot realistically simulate deformation of the model, particularly if the model consists of multiple materials. A physics-based technique, on the other hand, can realistically calculate the deformation and force response of the model, based on virtual material properties [13, 20]. A physics-based deformation model gives the designer more options to try different types of materials during the concept design phase. This deformation model can also be used to validate the concept in real-time. There are many techniques available for physics-based deformation modeling. However, a mass spring system consisting of a set of particles (nodes) connected through a network of spring and dampers can provide reasonable accuracy and speed for real-time interaction.

The degree reduction and merging of Bézier/B-spline curves into a single representation has been addressed in the literature by a variety of different analytical approaches [3, 8, 16, 18, 24, 27]. Du and Schmitt [6] have described various techniques to establish the conditions of geometric continuity between two adjacent bi-parametric surface patches. The continuity conditions enable CAD systems to create shapes by stitching or joining numerous low-order bi-parametric patches. Unfortunately, many collision detection algorithms used in VR environments [12] do not permit more than a single B-spline or NURBS surface patch to be considered at any instance in time. Even when a collision detection algorithm is designed to tackle multiple B-spline surface patches [7, 9, 19], the continuity of these multiple patches will no longer be valid when the model deforms due to interaction with the tool. This is because stitching the surface patches does not automatically connect the underlying dissimilar mass-spring networks. Hence, the stitching techniques cannot be used in a VR environment when the user intends to deform the surface to obtain the desired shape. There are many analytical techniques to increase or decrease the degree of a B-spline curve [16, 18], using constrained optimization. However the constrained optimization techniques are difficult to use for the surfaces. These analytical methods used for merging of B-spline curves and surfaces cannot be used in a virtual reality environment during real-time interactions.

When using B-spline surface representation of the model in a virtual reality environment, all the processes require finding the blending function. At the same time, the inverse of large matrices is required to be computed which can be computationally intensive and sometimes impossible. The cost of computations further increases while dealing with a complex model with a high degree of deformation and having multiple contacts.

In this paper, blending matrices are presented as an alternative methodology which can be used to establish a uniform mathematical model for all aspects of B-spline surface modeling/manipulation needed in a virtual concept design process. These blending matrices can provide a general tool for the conversions between different representations of B-spline surfaces. These matrices can be used for collision detection between two or more B-spline surfaces, merging B-spline surfaces, and generating physics based deformation model. Both the model and the tool can have complex shapes, elastic or plastic properties, and multiple contacts.

2 CONCEPT OF BLENDING MATRICES

In a virtual reality environment, all aspects of B-spline surface modeling/manipulation are needed to be done in real time. These include shape control, degree control, merging of multiple B-spline patches, collision detection, and determining nodes for a mass spring system. In general, blending functions need to be computed to achieve B-spline surface modeling and manipulation. The process of calculating the blending functions for B-spline surfaces is computationally expensive and an alternative method is needed if real time interaction is to be achieved.

Eqn. (1) can be used to compute a point on a B-spline surface at any fixed (u and v) parameter values. Piegl and Tiller [17] have mentioned five steps that are needed to compute a point on a B-spline patch. These include finding the knot spans in which parameters u and v lie, computing non-zero basis functions in the u and v directions within these knot vector spans, and multiplying these non-zero values of basis function with the corresponding control points. However, this technique involves many calculations. At the same time, when a large number of points need to be generated on a B-spline surface patch, the cost of computation would be high. The computational cost can be reduced by representing the B-spline surfaces as a tensor product surface,

$$S(u,v)=[u][B_u][\mathbf{P}][B_v][v]^{\mathrm{T}} \tag{4}$$

where $[u]=[1\ u\ u^2 ......u^{k-1}]$ and $[v]=[1\ v\ v^2 ....v^{l-1}]$.

This type of representation of B-spline requires computation of these matrices for the given values of parameter u and v at which the points are to be determined. However, finding the control points which can represent these points would require computing the inverse of the large matrices, which can be computationally expensive and sometimes impossible. The Eqn. (4) can be further modified to achieve the conversions in real time. This can be done if we can represent a B-spline surface as a parametrically uniform grid of points and the underlying blending function in terms of pre-calculated blending matrices. A B-spline surface can be discretized into a set of parametrically uniform grid of nodes for various values of u and v. The matrix of discrete points $\mathbf{M}$ is given by the equation,

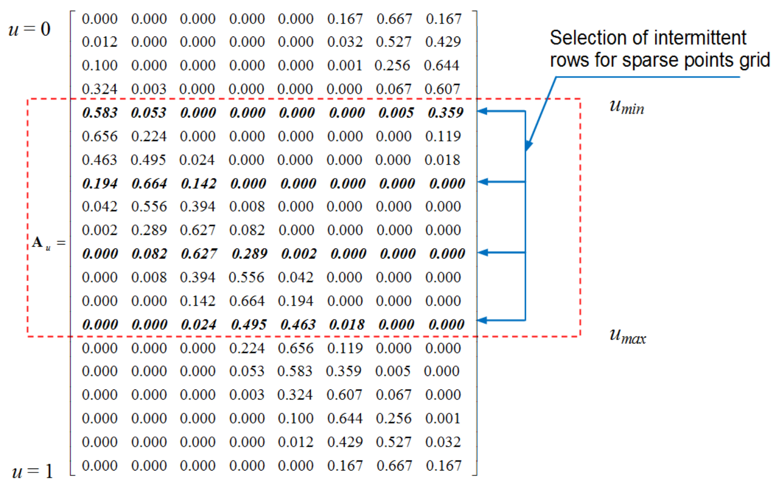
$$\mathbf{M} = \mathbf{A}_u\mathbf{P}_{ij}\mathbf{A}_v \tag{5}$$

where $\mathbf{A}_u$ and $\mathbf{A}_v$ are blending matrices, and their values depend upon the blending functions and u and v parametric values. The values of these blending matrices do not depend upon the position of the control points and hence, can be pre-calculated. Thus, there is no need to evaluate the points of interest on the surface by substituting corresponding values of u and v in the blending functions. These points can be determined by simply multiplying corresponding rows and columns of blending matrices with the control point matrix. Fig. 1 shows a typical blending matrix.

If it is required to generate points only in a particular region, the minimum and maximum values of the u and v parameters ($u_{min}, v_{min};\ u_{max}, v_{max}$) can be used to determine the rows and columns of the blending matrices to be used to generate points on the surface. Even the density of these discrete points generated on the B-spline surface can be varied by using intermittent rows and columns of the blending matrices, as shown in Fig. 1.

When the virtual model deforms, the discrete points would also change their position. This would require finding the new position of the control points which can represent the deformed model. Eqn. (5) can be re-written as,

$$\mathbf{P}_{ij} = [\mathbf{A}_u{}^{\mathrm{T}}\mathbf{A}_u]^{-1}\mathbf{A}_u{}^{\mathrm{T}}\mathbf{M}\ \mathbf{A}^{T}{}_v[\mathbf{A}_v\mathbf{A}_v{}^{\mathrm{T}}]^{-1} \tag{6}$$



$$
\begin{array}{r|cccccccc|l}
u=0 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.167 & 0.667 & 0.167 & \\
 & 0.012 & 0.000 & 0.000 & 0.000 & 0.000 & 0.032 & 0.527 & 0.429 & \\
 & 0.100 & 0.000 & 0.000 & 0.000 & 0.000 & 0.001 & 0.256 & 0.644 & \\
 & 0.324 & 0.003 & 0.000 & 0.000 & 0.000 & 0.000 & 0.067 & 0.607 & \\
 & \mathbf{0.583} & \mathbf{0.053} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.005} & \mathbf{0.359} & u_{min} \\
 & 0.656 & 0.224 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.119 & \\
 & 0.463 & 0.495 & 0.024 & 0.000 & 0.000 & 0.000 & 0.000 & 0.018 & \\
 & \mathbf{0.194} & \mathbf{0.664} & \mathbf{0.142} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \\
 & 0.042 & 0.556 & 0.394 & 0.008 & 0.000 & 0.000 & 0.000 & 0.000 & \\
 & 0.002 & 0.289 & 0.627 & 0.082 & 0.000 & 0.000 & 0.000 & 0.000 & \\
\mathbf{A}_u = & \mathbf{0.000} & \mathbf{0.082} & \mathbf{0.627} & \mathbf{0.289} & \mathbf{0.002} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \\
 & 0.000 & 0.008 & 0.394 & 0.556 & 0.042 & 0.000 & 0.000 & 0.000 & \\
 & 0.000 & 0.000 & 0.142 & 0.664 & 0.194 & 0.000 & 0.000 & 0.000 & \\
 & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.024} & \mathbf{0.495} & \mathbf{0.463} & \mathbf{0.018} & \mathbf{0.000} & \mathbf{0.000} & u_{max} \\
 & 0.000 & 0.000 & 0.000 & 0.224 & 0.656 & 0.119 & 0.000 & 0.000 & \\
 & 0.000 & 0.000 & 0.000 & 0.053 & 0.583 & 0.359 & 0.005 & 0.000 & \\
 & 0.000 & 0.000 & 0.000 & 0.003 & 0.324 & 0.607 & 0.067 & 0.000 & \\
 & 0.000 & 0.000 & 0.000 & 0.000 & 0.100 & 0.644 & 0.256 & 0.001 & \\
 & 0.000 & 0.000 & 0.000 & 0.000 & 0.012 & 0.429 & 0.527 & 0.032 & \\
u=1 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.167 & 0.667 & 0.167 &
\end{array}
$$

Fig. 1: A typical blending matrix.

Eqn. (6) can be used to find the new position of the control points representing the new matrix of discrete points. Equations (5-6) will determine the correlation between the control points and the points generated on the B-spline surface. A blending matrix can be generated depending upon some parameters of the B-spline surface. These include periodicity of the surface in the u and v directions, number of control points in the u and v directions, and the maximum number of points to be generated on the surface. Two blending matrices are required for every B-spline surface patch to generate discrete points. The size of these matrices will depend upon the number of control points and the maximum number of points to be generated in u and v directions. The maximum number of points to be generated is determined by the accuracy required for collision detection and merging of the surfaces. In general, the size of a blending matrix for generating m points on a B-spline surface having r control points in u direction will be $m \times r$. Similarly for s number of control points in the v direction, a blending matrix of size $s \times m$ will generate m points. Together, these two blending matrices can generate a maximum of m^2 points on the B-spline surface. Fig. 2 shows a discretized B-spline surface along with the set of control points $\mathbf{P}_{ij}$ [22].

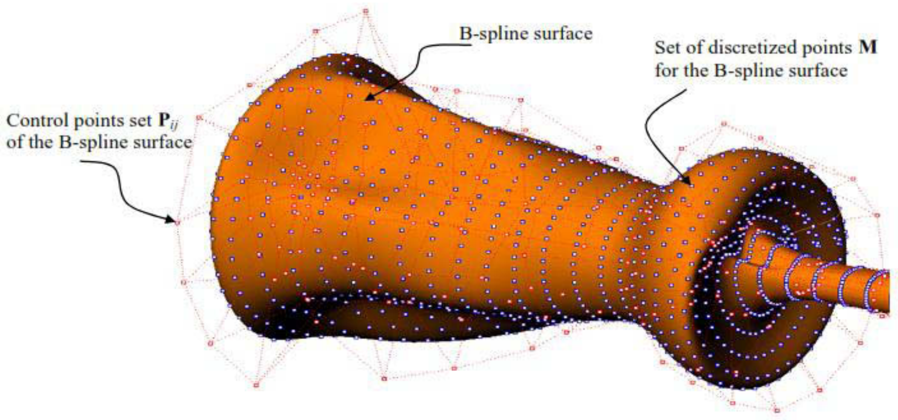

B-spline surface
Set of discretized points $\mathbf{M}$ for the B-spline surface
Control points set $\mathbf{P}_{ij}$ of the B-spline surface

Fig. 2: Discretized B-spline surface.

In the same manner, the blending matrices for finding tangents at each point generated in the u and v directions can be pre-computed. This is useful if tangential properties, such as friction, are also to be considered to realistically model material properties. The surface normal can be calculated, at any discrete point, by cross multiplication of the tangents in the u and v directions. A surface normal helps in correctly mapping the forces to the surface by calculating the forces normal to the surface.

When the virtual model deforms, due to haptic interaction, the position of the discrete points will also change. This would require computation of the new positions of control points that can represent the deformed surface. Eqn. (6) can be used to calculate the new position of the control points. This will involve computing the inverse of the blending matrices. Since the blending matrices are pre-computed and are independent of the position of the control points, their inverses can also be pre-computed. This reduces the computational cost of rendering the deformation of the B-spline surface. This inverse can be calculated by the Gaussian Elimination Method using ‘*complete pivoting*’. The complete pivot method enhances the robustness of the algorithm. As this is done during pre-processing, the time required to calculate the inverse of the matrix does not increase the computational cost during the run-time.

Thus, any B-spline surface can be represented as a set of blending matrices and control point matrix in a virtual reality environment. The user can model and manipulate the virtual object. Once the user interactively modifies the model, the information of the virtual model can be exchanged as a B-spline surface, with the existing commercially available CAD software. This type of representation allows for the pre-calculation of the underlying blending functions thereby reducing the computational cost of various interactive modifications carried out to the virtual model.

3 USING BLENDING MATRICES IN VR ENVIRONMENT

Based on the methodology proposed in this paper, the blending matrices, representing virtual objects are calculated and stored. This methodology can be used for a variety of applications in a VR environment. Some of these applications are explored in this section. Once a virtual object is modelled, it would need to be manipulated so as to get the desired shape and other features. To facilitate this modification, an efficient collision detection algorithm is required which can detect the collision between the tool and the model. As soon as the model and the tool start interacting, the physics-based model must find out the resultant deformation and the force feedback to be sent back to the user. This physics-based system can also be used to validate the concept in terms of deformation of the system, based on the virtual material properties assigned to the object. At the same time, the user may like to merge some independent surfaces to create a complex and interesting shape. For all these applications, the blending matrices can play a significant role in making the whole process computationally efficient. The following sections explore the applications and the resultant computational cost to determine the efficacy of the proposed methodology.

3.1 Collision Detection

Blending matrices can be used for collision detection of two or more B-spline surfaces or B-spline surface(s) and an implicit or point-based tool. This section briefly discusses the algorithm which efficiently uses the blending matrices for the collision detection of deformable B-spline surfaces [19]. The collision detection process starts with an intersection check of the convex hulls of the B-spline model(s) and the tool. The property of positivity ensures that a B-spline curve, or surface, always remains within the convex hull of the control points $\mathbf{P}_{ij}$ [2]. When two convex hull surfaces (or edges or an edge and a surface) intersect, the vertices (control points) of the surface (edge) are identified. The corresponding u and v parametric values associated with these vertices (control points) of the intersecting surfaces are calculated. These minimum and maximum values of u and v (u_{min}, v_{min}; u_{max}, v_{max}) set the limits on the surface to be discretized. Intermittent rows and columns of the blending matrices are used to sparsely generate points within these limits of parametric values of u and v. This region is initially large. By using lower levels of detail, the region is refined and simultaneously the intensity of the points is increased to enhance the accuracy of collision detection. There is no need to evaluate the blending functions. Similarly, by generating denser points, only at the lowest level of detail, the cost of the intersection check is reduced, as compared to the algorithm proposed by Gao and Gibson [7]. The collision detection procedure has been discussed in detail in [19]. This collision detection algorithm can detect the collision between two or more NURBS surfaces and/or between a NURBS surface and an implicit surface, a tessellated surface or a point.

To check the computational efficiency of the collision detection algorithm, different types of surfaces were used. A B-spline surface was made to collide with another B-spline surface, a sphere, a plane and a point. Fig. 3 shows two NURBS surfaces, a plane and a sphere.

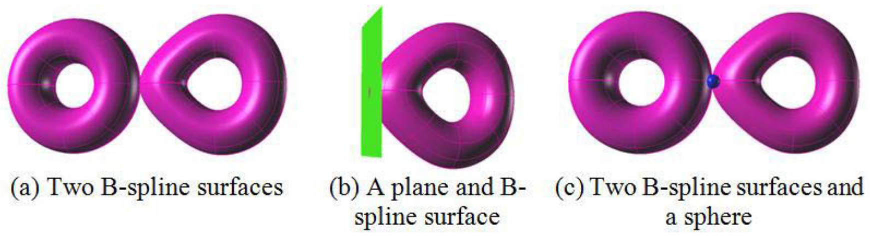

(a) Two B-spline surfaces
(b) A plane and B-spline surface
(c) Two B-spline surfaces and a sphere

Fig. 3: Primitives (plane and sphere) and B-spline surfaces (a donut and a distorted donut) for calculating time of collision detection.

The computational cost of collision detection depends upon the size of the blending matrices and the area of contact. The size of the blending matrices depends upon the number of control points and the maximum number of points that can be generated on the B-spline surface. The collision detection was carried out for different number of control points and different number of the maximum points to be generated on the B-spline surface. The collision detection was determined for B-spline surface - plane, B-spline surface - sphere, B-spline surface - point and B-spline surface - B-spline surface. The results for the computational cost for different parameters are discussed in the following sub-sections.

3.1.1 Effect of the number of control points

To determine the effect of number of control points on computational time, the number of points matrix used to determine collision was kept at 82×82 matrix. This matrix gives very good resolution. For simulation study the numbers of control points were selected in the range of 4×4 to 20×20. Most of the examples used in literature do not use more than a matrix of 12×12 for control points. Dachille [5] used a maximum of 12×12 control net with only 25×25 mesh of points for collision detection compared to an 82×82 matrix of nodes used in this study. The time taken for simulation for a B-spline surface represented by a 12×12 control point net with 25×25 mesh was 780 ms with implicit solver [5]. It used only a point to interact with the B-spline model. Gao and Gibson [7] used a resolution of 40×40 and used implicit surface rigid tools. It must be emphasized at this point that the computational time recorded in the simulation may not be the best time that the collision detection algorithm is capable of. The simulation time may be larger because of inherent inefficiencies in programming skills. A professional programming technique may be able to better utilize the resources of modern multi-core computers. Tab. 1 shows the computation times for collision detection of B-spline surface having different number of control points. The code was written in C++ and implemented with Microsoft Visual Studio 2008 on desktop computer having 6 GB RAM with Intel(R) Core(TM) i7 CPU @ 3.06 GHz running on Windows 7 Professional.

Control points net	Computational time (ms)		
	Point based tool	Sphere based tool	Plane based tool
4×4	<1	<1	2
6×6	<1	2	3
8×8	<1	3	4
10×10	2	4	8
12×12	4	5	10
14×14	5	8	15
16×16	6	12	22
18×18	8	13	26
20×20	10	17	31

Tab. 1: Computational time for collision detection of B-spline surface having different number of control points with a point, a sphere, and a plane.

As expected, the computational cost increased with increase in the number of control points used to represent the B-spline surface. As the number of control points increase, the computational cost of generating more points and subsequent generation of spheres increases. Overall, the computational time for collision detection is small. Particularly for B-spline surfaces having 12×12 control points, it takes 5 ms or less for the collision detection algorithm to check intersection with a point or sphere.

The collision detection was also carried out between two B-spline surfaces. The control points of the two B-spline surfaces were increased independently and the computational cost was determined. Fig. 4 shows the computational cost of collision detection between two B-spline surfaces. Again, the results are the average of the time taken by the algorithm to detect collision for different types of B-spline surfaces when these surfaces just start colliding. The computational cost increases with increase in the number of control points used to represent the B-spline surface. The resolution of the model was same for all the cases and was kept at 82×82. Time required for detecting collision between two B-spline surfaces represented by 8×8 control points net was 15 ms. For higher number of control points, the computational time increases, reaching to about 91 ms for B-spline surfaces when both the surfaces have 20×20 control points mesh. For smaller control points mesh, the computational time is very small and reduces to 6 ms when both the surfaces have 4×4 control points net. Thus, the collision detection algorithm can be used for real time interaction with virtual models.

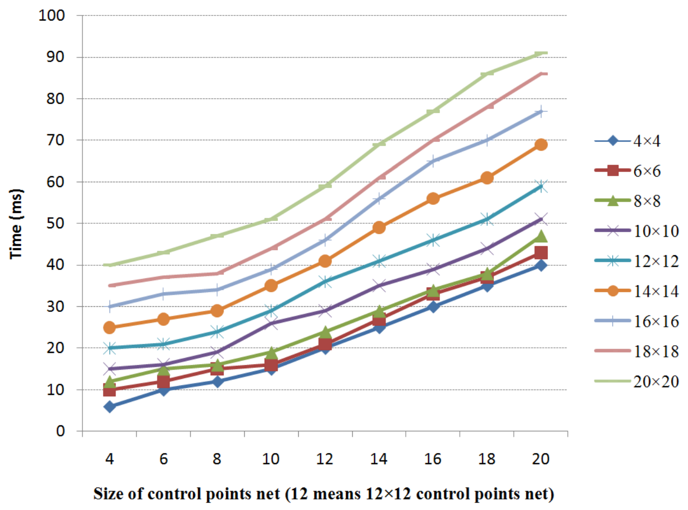


Fig. 4: Variation of the computational time (ms) of collision detection for a B-spline surface with another deformable B-spline surface for different number of control points.

3.1.2 Effect of the maximum number of points that can be generated

The number of points generated on the B-spline surface defines the resolution and accuracy of the collision detection. This also determines the maximum number of nodes that can be generated for the mass spring mesh. A larger number increases the size of the blending matrices, $\mathbf{A}_u$ and $\mathbf{A}_v$, used to generate these points using Eqn. (5). As a consequence of the large size of the blending matrices, the size of inverse of these blending matrices will also be large.

To clearly determine the effect of the maximum number of points that can be generated at the lowest level of detail, a tear shaped deformable model with 8×8 number of control point net was checked for intersection with a point, a sphere, and a plane. **Error! Reference source not found.** 5(a) shows a sphere colliding with a tear drop shape and **Error! Reference source not found.** 5(b) shows a sphere colliding with the deformed tear drop represented as B-spline surface having 8×8 control point mesh.

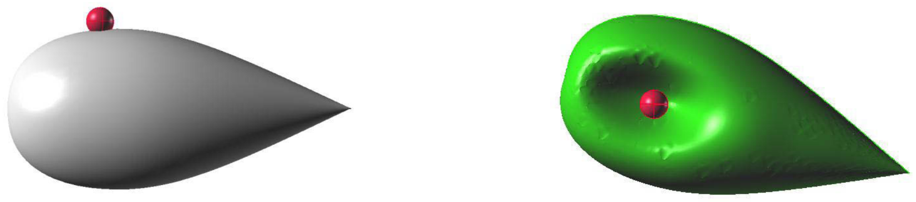

(a) At point of contact (b) After contact

Fig. 5: A sphere colliding with (a) A tear shaped B-spline surface (b) A deformed tear shaped B-spline surface.

The time taken for collision detection was computed for different shapes of B-spline surfaces. Tab. 2 shows the cost of computation for different number of the maximum points that can be generated on the B-spline surface.

Maximum points at lowest level of detail	Computational time (ms)		
	Point based tool	Sphere based tool	Plane based tool
28×28 (784 points)	<1	~1	1
28×82 (2296 points)	<1	2	2
82×82 (6724 points)	~1	3	4
82×244 (20008 points)	2	4	7
244×244 (59536 points)	3	6	15

Tab 2: Computational time for collision detection of B-spline surface having different number of the maximum points generated at lowest level of detail with a point, sphere, and plane.

When a large number of points are used, the accuracy of the collision detection is higher. The time shown in the table is average of the times noted for different B-spline surfaces. The computational cost for collision detection increases when the maximum number of points to be generated on the B-spline surface, at the lowest level of detail, is increased. When 82×82 mesh is used, the computational time is 1 ms for a point, 3 ms for a sphere and 4 ms for a plane. A mesh of size 82×82 means that a total of 6724 points can be generated on the B-spline surface. This gives high resolution for collision detection. For comparison, only 625 points (25×25 mesh of points) were used by Dachille *et al.* [5] and Gao and Gibbson [7] used 1600 points (40×40 mesh of points). This shows that the collision detection algorithm can efficiently detect collision even when a large number of points are generated to achieve higher resolution. The computational advantage is achieved due to the pre-calculated blending matrices and their inverses, as proposed in the methodology presented in this paper.

The variation of the computational cost of collision detection will be more pronounced if the resolution of two or more B-spline surfaces is changed simultaneously. During the simulation of collision detection for two B-spline surfaces the maximum number of points that can be generated was changed for both the surfaces and its effect on the computational time was recorded. The number of control points for both the surfaces were kept unchanged at 8×8. The graph representing the trend of the computation cost is shown in Fig. 6.

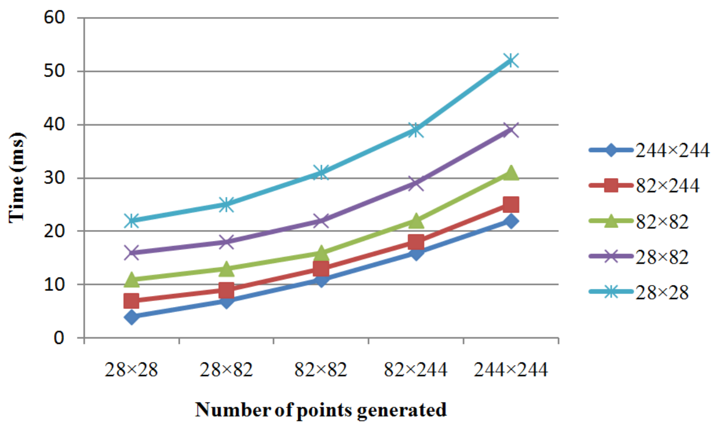


Fig. 6: Variation of the computational time (ms) of collision detection for a B-spline surface with another deformable B-spline surface with respect to the maximum number of points generated.

The computational cost for collision detection increases when the maximum number of points that can be generated at the lowest level of detail is increased. When 82×82 mesh is used, the computational time is 15 ms. A mesh of size 82×82 means that a total of 6724 points can be generated on both the B-spline surfaces. When the number of points is decreased to a mesh of 28×28, the computational time decreases to 4 ms. The simulation studies show that the collision detection can be carried out in real time while using the pre-computed blending matrices.

3.2 Merging Surfaces

The user may need to merge surfaces having a different degree or number of control points. The surfaces may not be symmetric and hence, the two surfaces may not attach to each other at a common edge. In some instances, the surfaces may be intersecting at one or more regions. These situations make it difficult and sometimes impossible to merge the surfaces even when using commercially available CAD software. The proposed methodology of blending matrices can be used to efficiently merge different B-spline surfaces.

The process of merging surfaces starts when two surfaces that need to be merged are moved closer together. The blending matrices representing the underlying surfaces are used to discretize these B-spline surface patches. These discrete points are stored as matrices $\mathbf{M}1$ and $\mathbf{M}2$. When the merging process starts, a combined matrix ô$\mathbf{M}$õ is generated by the combination of these two matrices. The number of rows and columns of matrix $\mathbf{M}$ are calculated.

As the two surface patches are combined, the number of control points of the merged B-spline surface in u and/or v directions and its knot vector will change. Consider a case where two surfaces are being joined along the u direction, having r_1 and r_2 number of control points in u direction for the first and the second surface respectively. The number of control points for the merged surface in the u direction will be given by $r = r_1 + r_2 - 1$. If the surfaces have a different number of control points in v direction, then the larger number is assumed as the new number of control points for the merged surface. If both the surfaces have the same number of control points in the v direction, then $s = s_1 = s_2$. Similarly, if the two surfaces are joined in the v direction, the number of control points in v direction will be given by $s = s_1 + s_2 - 1$.

A knot vector determines the area of influence of each control point on the B-spline surface. If the degree of the initial surfaces is not being changed, the knot vector of the two merging surfaces can be added to get the knot vector of the merged surface. The multiple knots at the common edge are reduced depending upon the type of connectivity needed at the edge. For providing only C^0 connectivity, a total of k-1 multiple knots are retained, where k is the order of the merged surface in the direction of merging. For C^1 connectivity, the number of multiple knots would be k-2 and so on. For the maximum connectivity (C^2 for a degree 3 surface) only one knot is retained at the common edge. In the second direction the knot is generally made uniform. The detailed merging process is explained in [21].

Fig. 7(a) shows two B-spline patches. Fig. 7(b) shows the discrete points generated on both the B-spline surfaces and Fig. 7(c) shows the surface generated by combining different surfaces

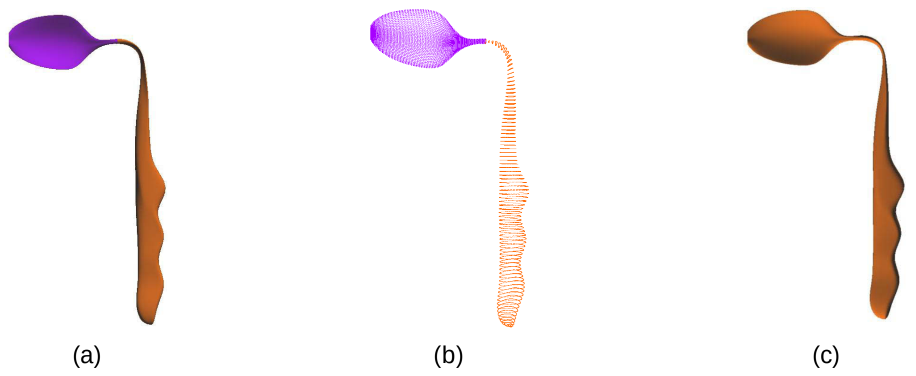


Fig. 7 (a) Bowl and handle of spoon (b) Point cloud of the bowl and handle of the spoon (c) Merged B-spline model of spoon.

The same procedure can be used to reduce or increase the degree of a surface. Similarly, if the two surfaces have different degrees, these can be merged with a common degree. In a similar fashion, the inverse of these blending matrices ($\left[\mathbf{A}_u^{\mathrm{T}}\mathbf{A}_u\right]^{-1}, \left[\mathbf{A}_v\mathbf{A}_v^{\mathrm{T}}\right]^{-1}$) is also computed and stored. These newly calculated blending matrices replace the earlier blending matrices for the two B-spline surfaces. These new blending matrices are used to determine the position of the control points which can generate the matrix of discrete points ($\mathbf{M}$). The position of the control points for the merged surface is calculated using Eqn. (6).

The computational time for merging of these surfaces depends upon the number of control points of the B-spline surfaces and the number of points generated to merge these two surfaces. It also depends upon the other parameters of the surfaces such as knot vector, degree of the surfaces and whether the surfaces are intersecting each other before merging. The computation cost of merging the surfaces is less than 4ms in most of the cases. In the example shown in Fig. 7, it took less than 2 milli-seconds to merge the spoon bowl and handle.

3.3 Physics Modeling System

A physics modeling system takes the information from the collision detection system regarding the penetration of the tool into the virtual model. Based on this information, the system needs to find the deformation of the model and resultant forces to be fed back to the user. A mass spring system gives reasonable accuracy and computational efficiency.

The blending matrices can efficiently determine the nodes for the mass spring system on the B-spline surface. Intermittent rows and columns can be used to generate an adequate number of nodes. Fig. 8 shows the generation of nodes and the mass spring system generated by using these nodes.

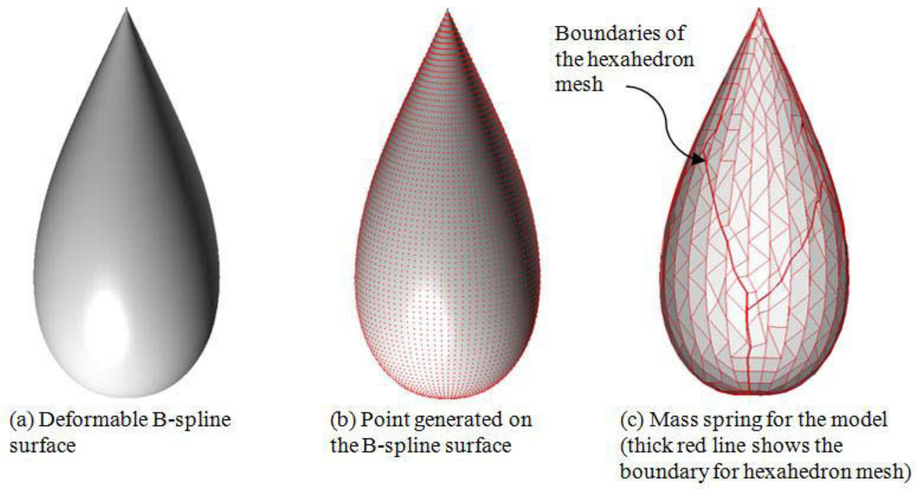

Boundaries of the hexahedron mesh
(a) Deformable B-spline surface
(b) Point generated on the B-spline surface
(c) Mass spring for the model (thick red line shows the boundary for hexahedron mesh)

Fig. 8: Generation of mass spring system.

Once the surface nodes are determined, the internal nodes can be generated through a *volumetric self organizing feature map* [11] and the material properties assigned to the virtual model.

In the sculpting of virtual objects within a VR environment, the realistic haptic system that can accurately respond to the userôs intentions is required. The external and internal forces directed at the geometric model of the three dimensional object are calculated from the haptic feedback and a collision detection algorithm. The system must be able to compute the estimated position of the dynamic model at the next time step from the current forces being applied through the haptic tool. While the object experiences external forces from the haptic device, numerical simulation of the deformation process can be achieved using the discrete Lagrange equations of motion for a dynamic node-spring system [13]. The system of equations is given by the second-order differential equation,

$$\rho_i \ddot{\boldsymbol{x}}_i(t) + \gamma_i \dot{\boldsymbol{x}}_i(t) + g_i(t) = f_i(t) \tag{7}$$

where γ_i is the velocity-dependent damping coefficient which dissipates kinetic energy in the lattice through friction; ρ_i is the point mass of node *i*, $g_i(t)$ is the total internal spring forces, and $f_i(t)$ is the external force vector applied to node *i*. At each time step φ*t* it is necessary to evaluate the current nodal forces and accelerations, the new velocities, and the new node positions using the explicit Euler time-integration procedure. It is possible to compute acceleration from Eqn. (7) at node *i* as,

$$\ddot{\boldsymbol{x}}_i(t) = (f_i(t) - \gamma_i \dot{\boldsymbol{x}}_i(t) - g_i(t)) / \rho_i \tag{8}$$

and the new velocity is given by,

$$\dot{\boldsymbol{x}}(t+\Delta t) = \dot{\boldsymbol{x}}(t) + \Delta t\, \ddot{\boldsymbol{x}}_i(t) \tag{9}$$

The new position of node ô*i'* is then calculated using,

$$\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i(t) + \Delta t\, \dot{\boldsymbol{x}}_i(t+\Delta t) \tag{10}$$

As the user applies force on the model through the haptic tool, Eqn. (7-10) determine the deformation of the model.

This mass spring mesh is created during the pre-processing stage using VSOFM [10] in order to provide more realistic virtual experience. A denser mesh of mass spring nodes increases the accuracy of the deformation behavior for the virtual model. Unfortunately, a denser mesh will also increase the computational cost for calculating the shape change and resultant forces that must be fed back to the user. To simulate realistic material behavior, the mass spring damper system requires small time-step to simulate physics based deformation of model characterized by a material from low to high stiffness. The running time depends upon the size of the node mesh for the mass spring system and the number of training cycles. The mass spring damper system can be used for local deformation and global deformation.

The computation time to generate nodes for mass spring system depends upon the size of the mass spring mesh and the number of learning cycles used to generate VSOFM mesh. More cycles (up to 1000) are needed when the VSOFM models is required to generate the mass spring mesh from a very large cloud point set. However, using intermittent rows and columns of the blending matrices, very small number of nodes is used for generation of mass spring nodes. Thus, fewer learning cycles (100) give very good results. In the simulation study, presented in this paper, 300 learning cycles were used which yielded very good result. For a 12×12×12 mass spring mesh, it takes 3.471 seconds to generate the mass spring damper model. Tab. 3 shows the computational time for generation of mass spring mesh of different sizes. Even for 20×20×20 mass spring mesh the pre-processing time is less than half minute, which is very reasonable.

Number of nodes	Pre-Processing Time (seconds)
10×10×10	2.44
15×15×15	7.88
20×20×20	28.48
25×25×25	76.90
30×30×30	213.02
35×35×35	353.19
40×40×40	958.10

Tab. 3: Pre-processing time for generation of mass spring system for different node sizes.

The computation time increase with increased size nodes for the mass spring system. Fig. 9 shows the correlation between the size of the mass spring mesh being generated and the pre-processing time for calculating the mesh by using volumetric self organizing feature map.

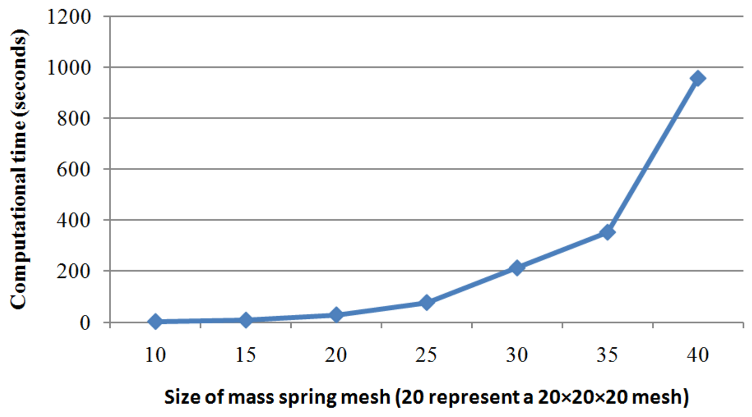


Fig. 9: Pre-processing time for calculating the mass spring mesh of different sizes.

4 CONCLUSION

Product concept generation within a virtual reality environment requires a large variety of tools that enable the user to enhance his or her creativity. However, to achieve real-time interaction, the algorithm needs to be computationally efficient. The methodology proposed in this paper to represent virtual objects, by using the B-spline blending matrices, reduces the computational cost of interacting with B-spline based virtual models in a VR environment. Once calculated, the blending matrices can be used for a variety of applications. Two or more B-spline patches can be merged in a VR environment. The same blending matrices can be used for efficient collision detection as well as generating nodes for the mass spring system. In this manner, all the aspects of B-spline manipulation work in tandem and reduce the computational cost without affecting the accuracy of various interactions. The efficiency achieved by using blending matrices would allow the use of B-spline representations of a virtual model and tool in a virtual reality environment.

ACKNOWLEDGEMENTS

This work has been supported, in part, by All India Council for Technical Education New Delhi, India, through research promotion scheme; vide Ref. No. 8023/RID/RPS-12/2011-12 and Natural Sciences and Engineering Research Council of Canada. The authors also acknowledge the support from the National Research Council of Canada in London, Ontario, Canada.

REFERENCES

[1] Basdogan, C.; Suvranue, D.; Jung, K.; Muniyandi, M.; Kim, H.; Srinivasan, M.: Haptics in minimally invasive surgical simulation and training, IEEE Trans. on Computer Graphics and Applications, 24(2), 2004, 56-64.

[2] Bloomenthal, J.; Bajaj, C.; Blin, J.; Gascuel, M.; Rockwood, A.; Wyvill, B.; Wyvill, G., Introduction to implicit surfaces, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 1997.

[3] Cheng, M.; Wang, G.: Approximate merging of multiple Bezier segments, Progress in Natural Science, 18(6), 2008, 757-762.

[4] Cheshire, D.; Evans, M.; Dean, C.: Haptic modeling an alternative industrial design methodology? In: Proceedings of EuroHaptics 2001, Birmingham, UK, 2001, 124-129.

[5] Dachille, F.; Kaufman, A.; Qin, H.: A novel haptics based interface and sculpting system for physics-based geometric design, Computer Aided Design, 33(5), 2001, 403-420.

[6] Du, W.-H.; Schmitt, F. J. M.: On the G^1 continuity of piecewise Bézier surfaces: a review with new results, Computer- Aided Design, 22(9), 1990, 556-573.

[7] Gao, Z.; Gibson, I.: Haptic sculpting of multi-resolution B-spline surfaces with shaped tools, Computer Aided Design, 38(6), 2006, 661-676.

[8] Hu, S.-M.; Tong, R.-F.; Ju, T.; Sun, J.-G.: Approximate merging of a pair of Bézier curves, Computer-Aided Design, 33(2), 2001, 125-136.

[9] Hughes, M.; DiMattia, C.; Lin, M.; Manocha, D.: Efficient and accurate interference detection for polynomial deformation. In: Proceedings of the IEEE Computer Animation Conference, Washington DC, USA, 1996, 155-166.

[10] Igwe, P. C., Deformable volumetric self-organising feature maps and physics-based modeling for concept design, Ph.D. *Thesis*, Department of Mechanical and Material Engineering, The University of Western Ontario, London, 2008

[11] Igwe, P. C.; Knopf, G. K.; Canas, R.: Developing alternative design concepts in VR environments using volumetric self organizing feature maps, Intelligent Manufacturing, 19(6), 2008, 661-675.

[12] Jimenez, P.; Thomas, F.; Torras, C.: 3D collision detection: a survey, Computer and Graphics, 25(2), 2001, 269-285.

[13] Knopf, G. K.; Igwe, P. C.: Deformable mesh for virtual shape sculpting, Robotics and Computer Integrated Manufacturing, 21(4), 2005, 302-311.

[14] Lin, M. C.; Manocha, D., Collision and proximity Queries, in Goodman, J. E.; O'Rourke, J., eds., Handbook of Discrete and Computational Geometry, Chapman & Hall, Boca Raton, Florida, USA, 2004.

[15] Patoglu, V.; Gillespie, R. B.: A closest point algorithm for parametric surfaces with global uniform asymptotic stability. In: Proceedings of 1st Joint Eurohaptic Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleporter System (WHCí05), Pisa Italy, 2005, 348-355.

[16] Piegl, L.; Tiller, W.: Algorithm for degree reduction of B-spline curves, Computer Aided Design, 27(2), 1995, 101-110.

[17] Piegl, L.; Tiller, W., The NURBS book, Monographs in visual communications, Springer, New York, 1997.

[18] Piegl, L.; Tiller, W.: Software engineering approach to degree elevation of B-spline curves, Computer Aided Design, 26(1), 1994, 17-28.

[19] Pungotra, H.; Knopf, G. K.; Canas, R.: Efficient algorithm to detect collision between deformable B-spline surface for virtual sculpting, Computer-Aided Design, 40(10-11), 2008, 1055-1066.

[20] Pungotra, H.; Knopf, G. K.; Canas, R.: Framework for modeling and validating concept designs in virtual reality environments. In: Proceedings of the IEEE Virtual Reality 2009 Conference (Symposium on Human Factors and Ergonomics), Toronto, Canada, 2009, 393-398.

[21] Pungotra, H.; Knopf, G. K.; Canas, R.: Merging multiple B-spline surface patches in a virtual reality environment, Computer Aided Design, 42(10), 2010, 847 - 859.

[22] Pungotra, H.; Knopf, G. K.; Canas, R.: Novel collision detection algorithm for physics-based simulation of deformable B-spline shapes, Computer Aided Design & Applications, 6(1), 2009, 43-54.

[23] Sener, B.; Wormald, P.; Campbell, R.: Evaluating a haptic modeling system with industrial designers. In: Proceedings of EuroHaptics International Conference, Edinburgh, Scotland, 2002, 165 -169.

[24] Taia, C. L.; Hub, S. M.; Huangb, Q. X.: Approximate merging of B-spline curves via knot adjustment and constrained optimization, Computer-Aided Design, 35(10), 2003, 893-899.

[25] Ye, J.; Campbell, R.: Supporting conceptual design with multiple VR based interfaces, Virtual and Physical Prototyping, 1(3), 2006, 171-181.

[26] Ye, J.; Campbell, R.; Page, T.; Badni, K.: An investigation into the implementation of virtual reality technologies in support of conceptual design, Design Studies, 27(1), 2006, 77-97.

[27] Yong, J.-H.; Hu, S.-M.; Sun, J.-G.; Tan, X.-Y.: Degree reduction of B-spline curves, Computer Aided Geometric Design, 18(2), 2001, 117-127.

[28] Zheng, J. M.; Chan, K. W.; Gibson, I.: Constrained deformation of freeform surfaces using surface features for interactive design, Advanced Manufacturing Technology, 22(2), 2003, 54-67.