Computer-Aided Design and Applications

http://www.cadanda.com

A New Approach to Accelerate NURBS Surface Rendering on GPU

Hong-Tzong Yau[1], Yen-Kun Lin[2] and Chien-Ting Yeh[3]

[1]National Chung-Cheng University, imehty@ccu.edu.tw
[2]National Chung-Cheng University, arb1130@cad.me.ccu.edu.tw
[3]Pou-Yuen Technology, fortears@pcc-server.ccu.edu.tw

ABSTRACT

The aim of this paper is to develop a novel algorithm for the real-time rendering of NURBS surfaces using Graphics Processing Unit (GPU). The GPU is used to compute the basis function and the derivatives of basis function of a NURBS surface in real time using the parallel computing power of the GPU. The results are used to combine with the control points to obtain the exact positions and corresponding normal vectors on NURBS surfaces. Thus, the shading effect also can be calculated by using the exact positions and normal vectors. In the proposed algorithm, both trimmed and untrimmed surfaces can be rendered. According to our experimental results, NURBS surface rendering with less meshes can still produce the evaluation of the exact surfaces.

Keywords: NURBS surfaces, Graphics Processing Unit.
DOI: 10.3722/cadaps.2009.529-538

1. INTRODUCTION

NURBS surfaces provide a convenient and compact representation of the parametric free-form surfaces that have become the standard of choice in mechanical CAD systems. Hence, the focus of this paper is the real-time interaction with NURBS surfaces. Over the last few years, several articles have been devoted to the study of NURBS curve and surface. L. Piegl and W. Tiller [1-2] give the basic definition in NURBS curve and surface whose properties such as the controlled point, knot vector and weight value are introduced. These properties are particularly powerful to modify the shape of the model. NURBS surface rendering can be done by using many different approaches such as direct rendering, tessellated surface rendering and so on. A great deal of effort has been made on the direct rendering such as ray tracing or ray casting [3-6] which can calculate the correct color for an arbitrary point on the surface. What seems to be lacking, however, is that there is no specific hardware to support the direct rendering so far. Hence, the computation for the direct rendering is time-consuming. On the other hand, the tessellated surface rendering scheme is approximated and efficient rendering. The color within the polygon is interpolated and computed according to the normal vectors of vertices in each polygon. Due to the efficiency, the tessellated NURBS surface rendering has become the most commonly-used technique in general CAD systems. The positions and the normal vectors within each polygon, however, are not exact at all. Recently, GPU is applied in general purpose computational tasks by using its programmable parallel processors [7]. The vertex and fragment shaders are the main programmable units where we can execute a user-defined set of instructions. Furthermore, shaders can output directly to a floating-point texture using off-screen render targets called Frame-Buffer Objects (FBOs) [8]. Because multiple vertices and pixels are processed in parallel, and operands are four-component vectors, GPU can achieve much higher computational speeds than conventional CPU on

arithmetically intensive operations. Utilizing the parallel computing power of GPU with the exact representation of NURBS, there is a potential of realizing real-time and interactive NURBS rendering without sacrificing the fidelity.
In this paper, a GPU-based algorithm is proposed to efficiently perform the modeling operation and rendering of NURBS surfaces. The proposed system has the following advantages:

- Present the evaluation of the exact surfaces and use less memory even with a rough tessellation.
- Calculate the exact derivatives and exact normals by using parallel fragment shaders on GPU.
- Render trimmed surfaces in real-time.

In the proposed system, a NURBS surface is read and tessellated. The surface also can be deformed by adjusting the control points and reloaded in the proposed system. At tessellation, the surface is also determined whether the surface is trimmed or untrimmed. After the above steps, some information such as control points, knot vectors and trimmed curve should be stored in textures of a GPU. Finally, the NURBS surface is rendered by using the programmable fragments on the GPU. The computational intension for the normal evaluation and Phong shading is more serious than operations of the tessellation and surface trimming in the proposed method. As a result, the the normal evaluation and Phong shading are computed on GPU and operations of the tessellation and surface trimming are done on CPU.

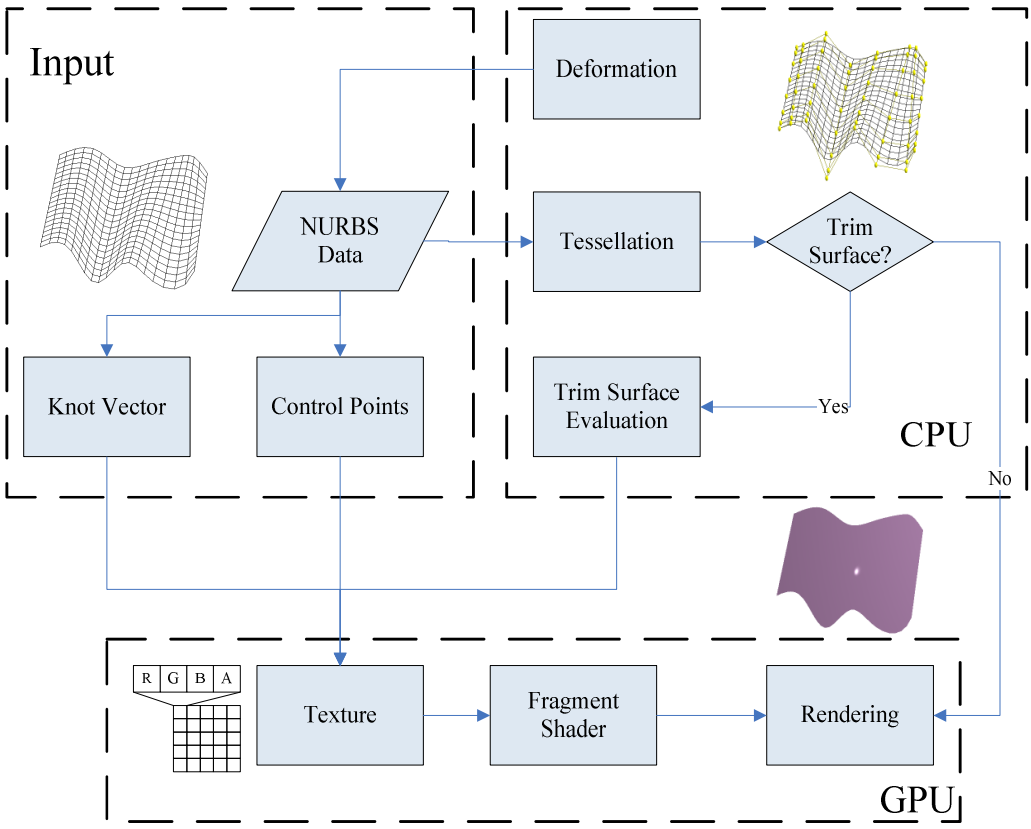


Fig. 1: The flowchart of the proposed system.

The methods of the tessellation and evaluation of a NURBS surface are summarized in section 2. We then explain the rough tessellation and evaluation of the NURBS surface in section 3. The results and discussion are then described in section 4. Finally, we conclude the paper in section 5.

2. LITERATURE REVIEW

2.1 Trimmed Surface Rendering

Rockwood et al. [9] proposed a modular approach to render trimmed surfaces by handling the multiple patches. Based on previous work [10], their approach converts all surfaces into Bezier patches bounded by the trimming curves that require additional calculations in the triangulation workflow. Kumar et al. [11] recommended using a view dependent algorithm for rendering the trimmed surface interactively. They proposed an optimization algorithm that converts the NURBS surfaces/curves into Bezier surfaces/curves and enhances the triangulation process.

2.2 Evaluation

The process of finding the surface coordinates (x, y, z) for given parameter value (u, v) is called evaluation. Recently, programmable GPU has become a new accelerator for general purpose computation. Therefore, a method [12] suggested using either Catmull-Clark concept to evaluate surface with only few control points to be stored into the texture on GPU. Y. Yasui and T. Kanai [13-14] further extended the above method to compute the accurate position and normal of subdivision surfaces for each fragment on GPU. Then, T. Kanai [15] developed a fragment-based algorithm to determine the specific degree in advance and accelerate the non-uniform B-Spline surfaces evaluation on GPU.

2.3 Trimmed Surface on GPU

A trimmed texture for trimmed region is defined by M. Guthe et al. [16-17]. The texture can be dynamically adapted to the required resolution and allow for an efficient trimmed surface evaluation on GPU. The bicubic Bezier patches are used to approximate each NURBS surface in the vertex shader. Then, S. McMains et al. [18] presented a novel method for evaluating any-order trimmed NURBS surfaces on GPU. The surface evaluation is computed by using multi-pass in fragment program to obtain the surface point coordinates.

3. GPU BASED NURBS RENDERING

After converting the NURBS data into a suitable polygonal representation, the polygons can be divided into untrimmed polygons and trimmed polygons. The process of converting the NURBS data is called tessellation which can be quite efficient. Due to the effective memory bandwidth on the modern GPU, multi-million triangles can theoretically be generated per frame. The basis function and its derivative of each patch vertex can be obtained by using corresponding knot vectors. The first derivative with respect to the two parametric directions u and v on the vertex of the polygon is computed. Then, the corresponding control points, basis function and derivative of basis function are used to compute the surface point and surface derivatives. After getting the above information, the normal of the vertex can be calculated by taking cross product of the u and v first derivatives. Therefore, lighting and shading are done by using a normal vector of each vertex and by a linear interpolation of vertex colors in each triangle. If the NURBS surface is a trimmed surface, the trimmed region of the surface should be rendered on the off-screen buffer. After rendering the trimmed region, the trimmed texture can be built by using FBO. Finally, the base polygon and trimmed texture can be blended and the trimmed surface can be rendered on screen. The flowchart is indicated in Fig. 2.

3.1 Tessellation

Although finer surface tessellation produces better surface quality, higher computation cost needs to be paid. Therefore, rough tessellation is used to lessen the computational cost and save the memory. With the proposed method, rough tessellation can still produce fine surface quality at low computation cost, except at (trimmed) boundary curves. After the NURBS data is read, the parametric domain (u, v) of the data can be uniformly partitioned into $n_u \times n_v$ patch size $\frac{1}{n_u} \times \frac{1}{n_v}$ [19], as illustrated in Fig. 3.

3.2 Surface Derivatives

The NURBS surface of degree p, q with knot intervals i, j determined in both u and v directions is represented by [1]:

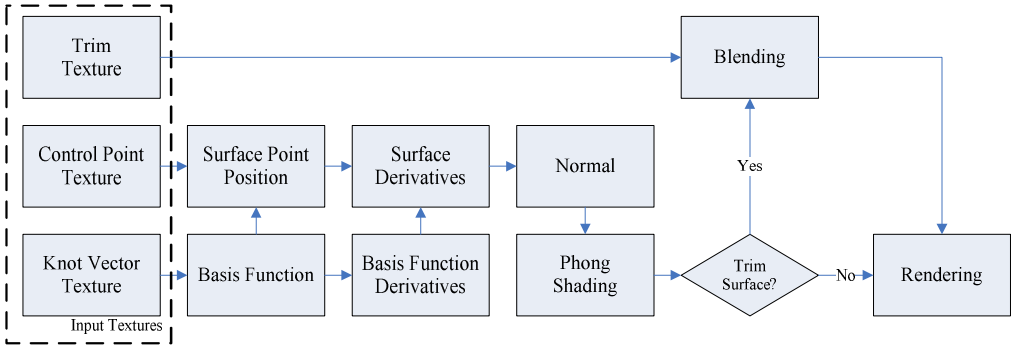


Fig. 2: The flowchart in the proposed GPU based algorithm.

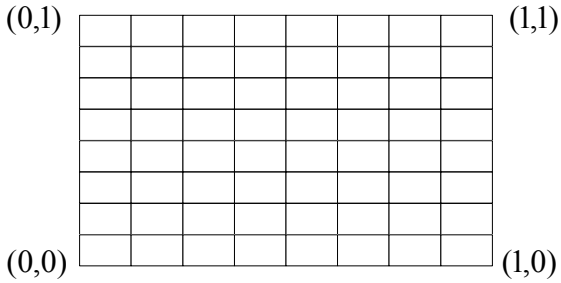


Fig. 3: Uniform tessellation in parametric space.

$$S(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad 0\le u,v\le 1 \tag{3.1}$$

The $N_{i,p}(u)$ and $N_{j,q}(v)$ are the non-rational B-Spline basis functions of degree p and q respectively. The $w_{i,j}$ are the weights. The basis function is defined by equation (3.2) and (3.3). The $P_{i,j}$ are the NURBS control points and the u_i are the knots.

$$N_{i,p}(u)=\frac{u-u_i}{u_{i+p}-u_i}N_{i,p-1}(u)+\frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u) \tag{3.2}$$

$$N_{i,0}(u)=\begin{cases}1 & if \quad u_i\le u\le u_{i+1}\\ 0 & otherwise\end{cases} \tag{3.3}$$

The derivative of the basis function of degree p with respect to u is given by equation (3.4). The basis function of degree p needs to be computed to evaluate the derivative of a basis function of degree p. The N^u denotes the derivative with respect to u. Note that the p in the numerator of equation (3.4)

arises due to the fact that the B-Spline basis function of degree p that we are differentiating is a piecewise polynomial of degree p in u.

$$N_{i,p}^{u}(u) = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{3.4}$$

The derivatives of the B-Spline basis function, N^u and N^v, are then multiplied by the control $P_{i,j}$ to get the derivative along the u or v parametric direction on the surface as given by equation (3.5) and (3.6) respectively. The normal $n(u,v)$ of the NURBS surface can be calculated by taking the cross product of u and v derivatives as given by equation (3.7).

$$S^u(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}^{u}(u) N_{j,q}(v) w_{i,j} P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}^{u}(u) N_{j,q}(v) w_{i,j}} \tag{3.5}$$

$$S^v(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}^{v}(v) w_{i,j} P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}^{v}(v) w_{i,j}} \tag{3.6}$$

$$n(u,v) = S_{p,q}^{u}(u,v) \times S_{p,q}^{v}(u,v) \tag{3.7}$$

3.3 Phong Shading Model

After the normal vectors of the NURBS surface are computed, the Phong shading model is used to render the surface on the screen. $I(u,v)$ is the color on the NURBS surface. $L(u,v)$ is the direction vector from the point on the NURBS surface toward each light source. $n(u,v)$ is the normal of the NURBS surface. $R(u,v)$ is the direction for a perfectly reflected ray of light. $V(u,v)$ is the direction towards the viewer such as a virtual camera. i_a, i_d and i_s are the intensities of the ambient, diffuse and specular components that include RGB values. k_a, k_d and k_s are the ambient, diffuse and specular reflection constant. α is a shininess constant. The Phong shading model is then denoted by:

$$I(u,v) = k_a i_a + \sum_{light\ \ numbers} \{k_d [L(u,v) \cdot N(u,v)] i_d + k_s [R(u,v) \cdot V(u,v)]^{\alpha} i_s\} \tag{3.8}$$

3.4 GPU Implementation

The related algorithms have been described in the last section. Let us now look at the implementation on GPU in detail. Initially, the control point and knot vector from NURBS data should be stored in memory on GPU. The memory on GPU is called the texture. Fig. 4 shows the storage of control points and knot vectors to 2D textures.

3.4.1 Untrimmed Surface Rendering

In the proposed algorithm, the basis function and its derivatives are prepared in the fragment shader. The NUBRS surface can be divided into multi-patch Bezier surfaces during the tessellation. Therefore, these Bezier surfaces can be converted into the individual polygon called base polygon. The corresponding u v coordinates of the NURBS surface should be mapping to each vertex of the base

polygon. The computation for the corresponding u v coordinates is related to the knot vector and control point textures. During rendering, the part within each polygon of the tessellated surface is rasterized into a set of fragments. Then, the corresponding u v coordinates can be interpolated in the fragment shader according to the neighbor exact u v coordinates of the polygon vertexes. Thus, in the fragment shader, the exact positions and normal vectors can be computed and be used to calculate the shading effect for each fragment.

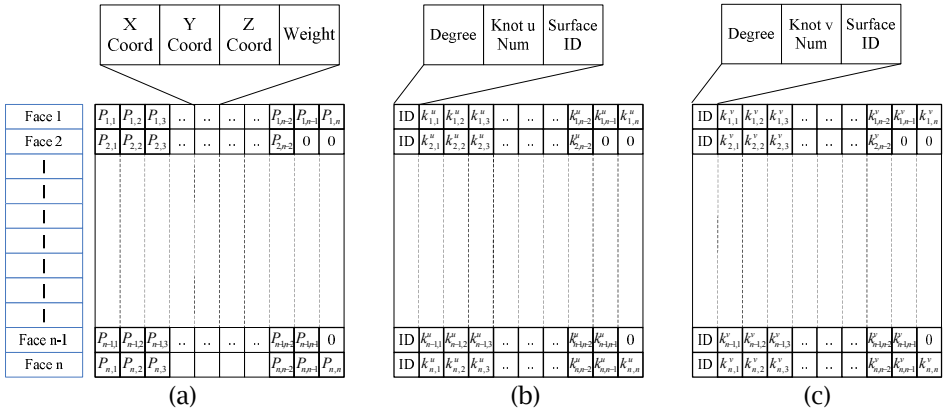


Fig. 4: 2D texture for computing NURBS surface: (a) control point texture (b) knot u vector texture (c) knot v vector texture.

3.4.2 Trimmed Surface Rendering

In view of the untrimmed surface, let us then consider the trimmed surface. The trimmed curve from NURBS data is a closed and oriented curve which lies on a NURBS surface. Therefore, the closed trimmed curve should be transformed into a suitable polygonal representation during converting the NURBS surface. The trimmed texture should be generated by these polygons and be used for a mask to avoid rendering the trimmed region [16]. First, these polygons should be rendered to texture. In this paper, the FBO extension on GPU is adopted. The rendering to texture in FBO is more efficient than conventional PBO (Pixel Buffer Object). In FBO, the texture can be repeatedly read and written without reloading data from CPU or switching buffers in PBO. Thus, an off-screen buffer, a trimmed texture, is created and its alpha value is set to one. The individual base polygon of NURBS data can be divided into the outer loop and the inner loop. The trimmed region should be evaluated and be spanned a triangle fan from the first vertex of each trimmed loop. After rendering the outer loop, the alpha value within the outer loop is set to zero by using the subtraction blending. The alpha within the inner loop is set to one and the alpha between the outer loop and inner loop is set to zero. Then, the trimmed texture is accomplished. Secondly, when the base polygon is rendered on screen, the trimmed texture is also read at the same pass. During the pass, only fragments whose alpha value of the trimmed texture is zero can be rendered using the Phong shading model. If the NURBS surface has multiple patches, the steps described above should repeat many times dependent on the numbers of the patch. The trimmed surface rendering is illustrated in Fig.5.

4. DISUSSION AND RESULTS

The proposed method was implemented in C++ and some cases were run on an AMD 2.31 GHz CPU with 2GB memory and a NVidia GeForce 8600GT graphic card with 256MB memory. After the tessellation, the NURBS data is converted into the uniform polygons. Therefore, the polygons such as triangles are decomposed into a set of fragments by rendering on screen. Then, the Phong shading model is computed for each fragment. The tessellation in different partitions causes different shading effects, as depicted in Fig. 6. What has to be noticed is the shading results of the boundary. The shading of most parts in the model is almost the same with the different patch size except for the boundary of the model. Obviously the quality of the boundary is mainly affected by the patch size.

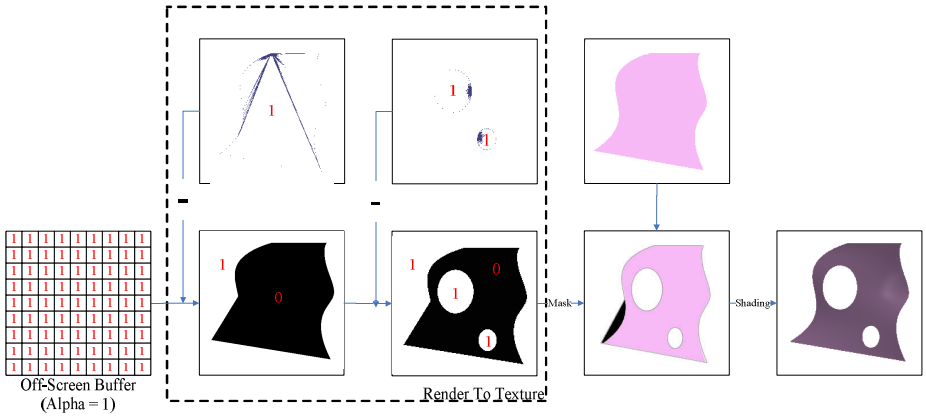
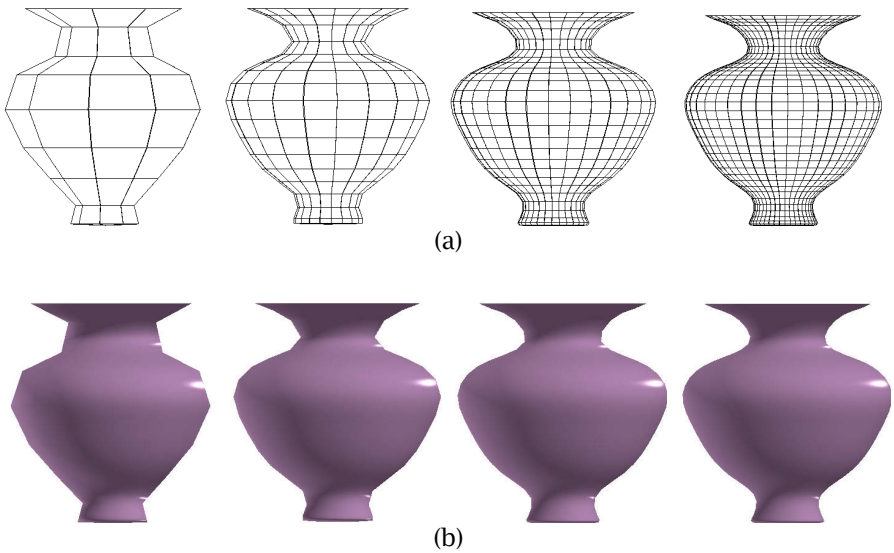


Fig. 5: Trimmed surface rendering.

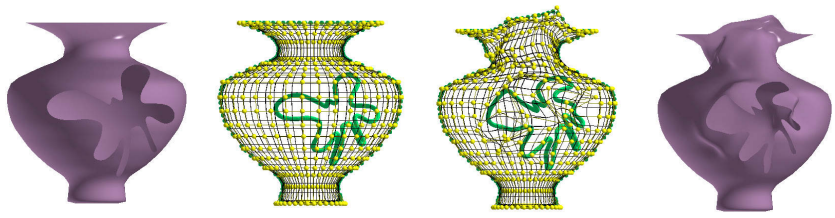


(a)

(b)

Fig. 6: Form left to right: 10x10, 20x20, 30x30 and 40x40 patch sizes. (a) wire-frame display (b) shading.

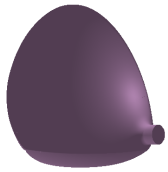
In this paper, the modeling operation is real time deformation. Both the control points of the surface (yellow) and the trimmed curve (green) can be arbitrarily deformed by using a mouse, as indicated in Fig. 7.



(a) (b) (c) (d)

Fig. 7: Modeling operation for deformation: (a) original model (b) control point (yellow) and trimmed curve (green) (c) deformation (d) result.

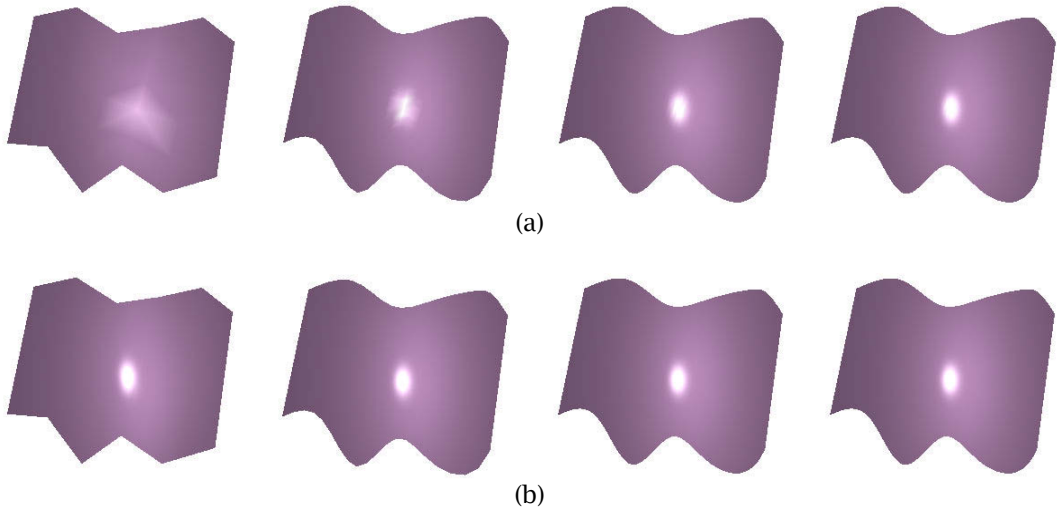
Tab. 1 gives a comparison of efficiency in three different cases. The patch numbers and resolution of the model are tested in the experiment. There are two lights adopted in this shading environment. When the more lights are used for shading environment, the more computations for Phong shading are calculated in fragment shader. However, the efficiency doesn't be seriously affected by numbers of the light in our experiment. As the result, the most likely explanation is that the efficiency is view-dependant resolution. As a result, the real-time rendering can be obtained in the reasonable resolution.

Model	*Patch numbers*	*Resolution*	*Display (FPS)*
	53 (30x30)	256x256	75
		512x512	38
		768x768	24
		1024x1024	20
	57 (30x30)	256x256	75
		512x512	37
		768x768	24
		1024x1024	18
	67 (30x30)	256x256	55
		512x512	31
		768x768	21
		1024x1024	14

Tab. 1: The efficiency in proposed system.

The rendering results in different patch sizes are generated from OpenGL API and proposed algorithm, as illustrated in Fig. 8. The proposed method can keep the better quality than OpenGL API.

The comparison for display rate between OpenGL API and proposed algorithm is then described in Tab. 2. The method for OpenGL API is based on [9],[20] and is entirely software implement for tessellation and ground rendering. The Bezier surfaces are used to extend OpenGL with other smooth primitives. The model is rendered at a resolution of 780x400. The shading model in OpenGL API is ground shading but the proposed method adopts Phong shading. As the patch size becomes larger, the difference in ratio becomes more obvious. The reason is that OpenGL API can make the fine tessellation to render, but the same result can be done by using the rough tessellation in the proposed method. The time for the GPU memory transfer and setup overhead of the rough tessellation is less than the rendering of the fine tessellation. In [16][17], this increases the number of Bezier patches by up to two orders of magnitude compared to the number of original NURBS patches. The numbers of surfaces are dramatically increased to be calculated, stored and render. Furthermore, there are limited parameters in the vertex shader to evaluate the Bezier patch. In the proposed method, the NURBS surface is directly evaluated and can save some computation and memory.



(a)

(b)

Fig. 8: Form left to right : 5x5, 20x20, 40x40 and 60x60 patch size: (a) OpenGL API (Ground shading) (b) proposed method (Phong shading).

Model	*Cells*	*OpenGL (FPS1)*	*Proposed algorithm (FPS2)*	*Ratio (FPS1/FPS2)*
	5x5	8.09	75	9.27
	10x10	6.35	60	9.45
	20x20	3.92	50	12.76
	30x30	2.87	39	13.59

Tab. 2: The comparison for display rate.

In addition, we compare the proposed algorithm to [15] improving their algorithm by integrating the trimmed texture concept [16] into proposed method. Moreover, the trimmed surface can be rendered by using the trimmed texture and alpha blending. Furthermore, the proposed method can manually adjust partition for tessellation to obtain the suitable surface quality. All trimmed surfaces, with the single exception of the trimmed region within the overlap in the single patch can be rendered in proposed method. In this case, the back of this patch can not be seen. Moreover, the proposed algorithm not only can efficiently evaluate the NURBS surface but also render the trimmed surface. Therefore, the balance between the patch size and quality can be obtained. Finally, real-time and the evaluation of the exact NURBS surfaces can be achieved on a normal PC.

5. CONCLUSION

An interactive NURBS surface rendering algorithm is proposed in this paper. A rough tessellation is used to save the memory. And the control points and knot vectors of each patch are stored in texture memory on GPU. Then, the Phong shading effect for each fragment can be obtained by using exact positions and normal vectors that are also calculated on GPU. In the proposed algorithm, NURBS surface rendering is not only efficient but also good quality. Moreover, both untrimmed and trimmed surfaces can be rendered in the proposed method. In the future, the patch size in tessellation can be adaptively controlled by the required local of detail. Furthermore, it is possible to extend the algorithm to add force feedback system such as a Haptic device to intuitively simulate the real-time modeling or deformation operations.

6. REFERENCES

[1] Piegl, L. A.; Tiller, W.: The NURBS Book, Springer-Verlag, New York, NY, 1997.

[2] Piegl, L.; Tiller, W.: Curve and Surface Constructions Using Rational B-Splines, Computer-Aided Design, 19(9), 1987, 485-498.
[3] Kajiya, J. T.: Ray Tracing Parametric Patches, SIGGRAPH'82: Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques, 16(3), 1982, 245-254.
[4] Toth, D. L.: On Ray Tracing Parametric Surfaces, SIGGRAPH'85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, 19(3), 1985, 171-179.
[5] Martin, W.; Cohen, E.; Fish, R.; Shirley, P.: Practical Ray Tracing of Trimmed NURBS Surfaces, Journal of Graphical Tools, 5(1), 2000, 27-52.
[6] Nishita, T.; Sederberg, T. W.; Kakimoto, M.: Ray tracing trimmed rational surface patches, ACM SIGGRAPH Computer Graphics, 24(4), 1990, 337-345.
[7] Rost, R. J.: OpenGL® Shading Language 2nd Edition, Addison, Wesley Professional, 2006.
[8] Owens, J.; Luebke, D., Govindaraju, N.; Harris M.; Kruger, J.; Lefohn, A. E.; Purcell, T.: A Survey of General-Purpose Computation on Graphics Hardware, Eurographics Computer Graphics Forum, 26(1), 2006, 21-51.
[9] Rockwood, A.; Heaton, K.; Davis, T.: Real-time rendering of trimmed surfaces, Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques, 23(3), 1989, 107-116.
[10] Abi-Ezzi, S. S.; Subramaniam, S., Fast dynamic tessellation of trimmed NURBS surfaces, Computer Graphics Forum, 13(3), 1994, 107-126.
[11] Kumar S.; Manocha, D.; Lastra, A.: Interactive display of large NURBS Models, IEEE Transactions on Visualization and Computer Graphics, 2(4), 1996, 323-336.
[12] Bolz, J.; Schröder, P.: Evaluation of subdivision surfaces on programmable graphics hardware, TR, California Institute of Technology Computer Science Department, Pasadena, CA, 2003, http://www.multires.caltech.edu/pubs/GPUSubD.pdf.
[13] Yasui, Y.; Kanai, T.: Surface Quality Assessment of Subdivision Surfaces on Programmable Graphics Hardware, International Conference on Shape Modeling and Applications, 2004, 129-138
[14] Kanai, T.; Yasui, Y.: Per-pixel Evaluation of Parametric Surfaces on GPU, ACM Workshop on General Purpose Computing Using Graphics Processors, 2004, C22.
[15] Kanai, T.: Fragment-based Evaluation of Non-Uniform B-Spline Surfaces on GPUs, Computer-Aided Design & Applications, 4(1-4) 2007, 287-294.
[16] Guthe, M.; Balázs, A.; Klein, R.: GPU-based Trimming and Tessellation of NURBS and T-Spline Surfaces, ACM Transactions on Graphics, 24(3), 2005, 1016-1023.
[17] Guthe, M.; Balázs, A.; Klein R.: GPU-based Appearance Preserving Trimmed NURBS Rendering, Journal of WSCG, 14(1), 2006, 1-8.
[18] McMains, S.; Khardekar, R.; Krishnamurthy, A.: Direct Evaluation of NURBS Curves and Surfaces on the GPU, ACM Symposium on Solid and Physical Modeling, 2007, 329-334.
[19] Espino, F. J.; Bóo, M.; Amor, M., Bruguera, J. D.: Adaptive Tessellation of NURBS Surfaces, Journal of WSCG, 11(1), 2003, 133-140.
[20] Mason, W.; Jackie, N.; Tom, D.; Dave, S.: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2, Addison-Wesley, Boston, MA, 1999.