A Border-Stable Approach to NURBS Surface Rendering for Ray Tracing

Aleksands Sisojevs and Aleksandrs Glazs
Riga Technical University,
Latvia

1. Introduction

Ray tracing has become a popular method for generating high quality images. Most of the modern ray tracing based applications only deals with triangles as basic primitives. NURBS surface representation is common for most of 3D modelling tools because of its compactness and the useful geometric properties of NURBS surfaces. Using the direct ray tracing of NURBS surfaces, one can achieve better quality of rendered images [1]. There are many approaches to solving this problem.

In 1982 Kajiya [5] used ideas from algebraic geometry to obtain a numerical procedure for intersecting a ray with a bicubic surface patch. His method is robust, not requiring preliminary subdivisions to satisfy some a priori approximation. It proceeds more quickly for patches of lower degree. The algorithm is simply structured and does not require memory overhead. But unfortunately the algorithm has many disadvantages. It does not significantly utilize coherence. The algorithm computes all intersections of the given ray with a surface patch, even if just closest intersection need to be found. And finally the algorithm performs enormous amounts of floating point operations. Kajiya estimates that 6000 floating point operations may have to be performed in order to find all of the intersections between one ray and one bicubic patch. In the modern ray tracing applications global illumination algorithms are commonly used, and million of rays can be tested against one parametric patch. It makes the proposed algorithm unpractical [2].

T.Nishita, T.W.Sederberg, and M.Kakimoto [7] described methods crossing for problem solution in star – rational Bezier surface. This method was called Bezier clipping. This method can be classified as an algorithm partly based on division and partly as calculation method. After ray representation as crossing of two planes, ray – surface crossing problem can be projected from 4D space to 2D space. This method reduces the number of arithmetical operations which is necessary to perform de Casteljau subdivision with 50% in every subdivision iteration. Nishita highlighted that Bezier cutting idea can be successfully applied in cases when we need to resolve problem of cropped region estimation. At the same time Nishita noted that described method does not resolve some tasks, one of them being the frequent point search problem and the instability of the method in some surface special cases.

W. Martin et al. [6] proposed a method for reverse NURBS surface visualization in ray tracing. Using node vector processing for generating hierarchic structure of limited space, as a result tree depth declines in comparison with other subdivision methods. The idea is to use handling of node vector so that after NURBS surface transformation Bezier surface set is developed. This surface is wide enough and has narrow limiting box. Bezier surfaces do not use large volumes of memory and are used only for limiting box hierarchic structure construction. The advantage of this method is that achieved plane surface is a good starting condition for Newton's iterative method. NURBS surface calculation scheme that is proposed in the work is based on node vector handling. Unfortunately, algorithm of NURBS surface calculation works slower than Bezier surface calculation. Newton method requires to calculate surface point and two partial derivations for every iteration to get quadric convergence. It is better to divide the initial NURBS surface on Bezier surface during initial processing. This requires extra memory volume to save every surface separately, but it allows to speed-up the calculation significantly.

S.W. Wang, Z.C. Shih and R.C. Chang [11] proposed an algorithm that combines Bezier cutting algorithm and Newton iterative algorithm in order to create effective method for ray coherence application. The first intersection point of the running ray with the Bezier surface is calculated using Bezier iterative algorithm. All following intersection points in the same pixel row are calculated using Newton iterative algorithm. The last calculated intersection point is used as previous result for the following intersection point. The device for barrier detection is used to check-up whether the intersection point that is calculated using Newton iterative algorithm is the last point. When Newton's method is not achieving convergences, then Bezier cutting is used as a replacement for calculating the intersection point.

A. Efremov, V. Havran and H.P. Seidel [1] and [2] proposed the method for NURBS surface visualization in ray tracing using following method: object's every NURBS surface is transformed into equivalent rational set of Bezier surface and exactly this set is mapped. To solve rational Bezier surface problem Bezier cutting method, that is described in [7], is used. And, also [1] and [2] proposes some modifications that improve the activity and effectiveness of Bezier cutting method.

Schollmeyer and Froehlich [9] describe an approach for NURBS surface ray tracing, where surface trimming is used to set of monotonic Bézier curves. For finding of intersection point the bisection method is used. But in this case, the number of calculations increases too.

In particular and NURBS surface is extensively used in computer graphics and computer aided design. Unfortunately, most of the algorithms for intersecting rays with parametric surfaces are expensive or have problems in some special cases. Therefore, most of modern ray tracing applications tessellate parametric surfaces into triangles during the preprocessing step of image generation. Such approach significantly increases computation speed, but can compute wrong images (if tessellation was not good enough) and requires additional memory for storage of generated triangles. Therefore, the problem of finding fast and robust algorithms for ray tracing parametric surfaces is still opened research issue [2].

This paper presents an effective approach for finding ray – NURBS surface intersection points, which are used for high-quality visualization of NURBS surfaces.

2. Ray-Surface Intersection problem

The mathematical task of finding an intersection point between the ray and a parametric surface can be described as a nonlinear equations system [6]:

$$\begin{cases} S'_X(u,v) = C_X(t) \\ S'_Y(u,v) = C_Y(t) \\ S'_Z(u,v) = C_Z(t) \end{cases} \tag{1}$$

where: S'_X, S'_Y, S'_Z – are the surface equations,
$C_X(t), C_Y(t), C_Z(t)$ – are the ray equations.

A NURBS surface patch in Cartesian 3D space can be formulated as [3, 8]:

$$S'(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} P'_{i,j} \cdot w_{i,j} \cdot N_{i,p}(u) \cdot N_{j,q}(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} \cdot N_{i,p}(u) \cdot N_{j,q}(v)} \tag{2}$$

where: $P'_{i,j}$ – are the control points;
$w_{i,j}$ – are the weights;
$N_{i,p}(u), N_{j,q}(v)$ – are the B-spline polynomials;
$n+1, m+1$ – are the number of control points in each parametric direction;
p, q – are the B-spline polynomials degree;
u, v – are the parameters.

2.1 Projection to R^2

Typically calculation is performed in $\mathbf{R}^3$ for non-rational patches and in $\mathbf{R}^4$ for rational [7].

Transforming the computation from 3D space to 2D space is important technique to reduce the comparison cost of finding ray-surface intersection point. Woodward [12] (also alluded to by [6]) shows how the problem can be projected to $\mathbf{R}^2$. This means that the number of arithmetic operations to calculate a rational patch is reduced by 25%. This approach is used in [7] for rational Bezier patch subdivision. But this approach is good for other parametrical surface patch calculation by ray tracing too. In case of NURBS surfaces, the task of ray-surface intersection point search is transformed to the problem of non-linear equations system solving:

$$\begin{cases} S_X(u,v) - x_R = 0 \\ S_Y(u,v) - y_R = 0 \end{cases} \tag{3}$$

where: $S_X(u,v), S_Y(u,v)$ – are the surface equations on the projection plane,
x_R, y_R – is the ray projection.

The system (3) solving task is divided into two parts: preprocessing with root preliminary search and iterative root finding.

2.2 Preprocessing

Other color values in gradient texture interpolate evenly and are put on the surface. The next task is to read data from the color map. Hence, it is proposed to develop preliminary value map in order to find preliminary parameters a *u* and *v* value in each pixel. The map is composed of surface data that is coded in RGB channels. Red channel includes surface number. Mathematical relation can be described in following way:

$$R = Nr + 1 \tag{4}$$

where: R is value of red channel, which is changing in diapason [1; 255];
Nr is surface number, which is changing in diapason [0; 254].

In case if R=0, we can say that in this pixel ray does not intersect any point. Taking into account 24 bits image coding in RGB color system we can say that surface number is changing in diapason [0; 254];

Green and blue channels consist of congener gradient texture that is on peace surface. Let's say, that color value in every channel is a whole number in diapason [0; 225]. Color value corner surface points can be given in the way it is described in the first table.

Surface point	Color value
$S(u_{\min}, v_{\min})$	$\begin{pmatrix} R & 0 & 0 \end{pmatrix}$
$S(u_{\max}, v_{\min})$	$\begin{pmatrix} R & 255 & 0 \end{pmatrix}$
$S(u_{\min}, v_{\max})$	$\begin{pmatrix} R & 0 & 255 \end{pmatrix}$
$S(u_{\max}, v_{\max})$	$\begin{pmatrix} R & 255 & 255 \end{pmatrix}$

Table 1. Corner control points color value

Other color values in gradient texture interpolate evenly and are put on the surface. Green and blue channels consist of congener procedure in gradient texture that is put on peace surface. Color value depends on *u* and *v* parameters and this can be calculated in following way:

$$\begin{cases} G = Round\left[255 \cdot \dfrac{(u - u_{\min})}{(u_{\max} - u_{\min})}\right] \\ B = Round\left[255 \cdot \dfrac{(v - v_{\min})}{(v_{\max} - v_{\min})}\right] \end{cases} \tag{5}$$

The map is coded using the OpenGL graphics library. The example of preliminary values map is shown in Fig. 1.

The next task is to read data from the color map. Input data in this case is R, G and B color value in every separate pixel. In this case we can find the preliminary value of parameters in following way:

$$\begin{cases} u_0 = \dfrac{1}{255} \cdot (u_{\max} - u_{\min}) \cdot G + u_{\min} \\ v_0 = \dfrac{1}{255} \cdot (v_{\max} - v_{\min}) \cdot B + v_{\min} \end{cases} \tag{6}$$

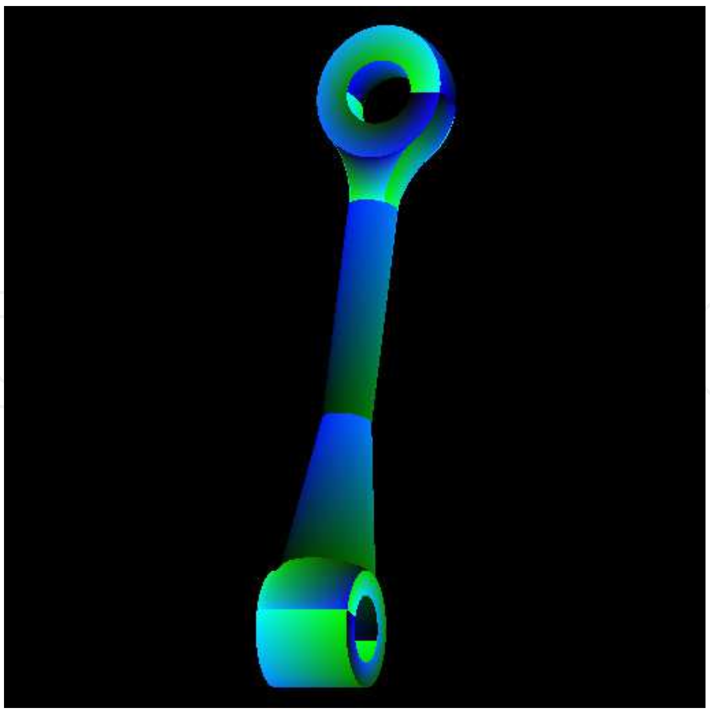
Fig. 1. The example of preliminary values map.

where: u_0, v_0 – are the parameters preliminary value in pixel;
$u_{\min}, u_{\max}$ – are the parameter u minimum and maximum;
$v_{\min}, v_{\max}$ – are the parameter v minimum and maximum;
G – is the color value in green channel;
B – is the color value in blue channel.

The patch number can be calculating from (4) as follows:

$$Nr = R - 1 \tag{7}$$

where: Nr is surface number, which is changing in diapason [0; 254];
R is value of red channel, which is changing in diapason [1; 255].

2.3 Intersection test

The Newton iteration [10] can be used for solving system (3) solving. In this case, an iteration step takes the form:

$$\begin{bmatrix} u_{i+1} \\ v_{i+1} \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \left[J(u_i, v_i) \right]^{-1} \cdot \left[F(u_i, v_i) \right] \tag{8}$$

where matrix for inversion can be calculated as follows:

$$\left[J(u,v) \right] = \begin{bmatrix} \dfrac{\partial S_X}{\partial u} & \dfrac{\partial S_X}{\partial v} \\ \dfrac{\partial S_Y}{\partial u} & \dfrac{\partial S_Y}{\partial v} \end{bmatrix} \tag{9}$$

is Jacobian matrix and

$$\left[F(u,v)\right] = \begin{bmatrix} S_X - x_R \\ S_Y - y_R \end{bmatrix} \tag{10}$$

The equation (8) can be described as follows:

$$\begin{bmatrix} u_{i+1} \\ v_{i+1} \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \tag{11}$$

The increment matrix can be described as follows:

$$\begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = -\left[J(u_i, v_i)\right]^{-1} \cdot \left[F(u_i, v_i)\right] \tag{12}$$

2.4 Proposed approach

The NURBS patch surface equation on the projection plane can be described as follows:

$$\begin{cases} S_X(u,v) = \dfrac{x(u,v)}{w(u,v)} \\ S_Y(u,v) = \dfrac{y(u,v)}{w(u,v)} \end{cases} \tag{13}$$

where:

$$x(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j}^X \cdot w_{i,j} \cdot N_{i,p}(u) \cdot N_{j,q}(v) \tag{14}$$

and

$$y(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j}^Y \cdot w_{i,j} \cdot N_{i,p}(u) \cdot N_{j,q}(v) \tag{15}$$

and

$$w(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} \cdot N_{i,p}(u) \cdot N_{j,q}(v) \tag{16}$$

where: $P_{i,j}^X, P_{i,j}^Y$, – are the control points x and y coordinates on the projection plane.

In next parts of the paper for simplification equations $x(u,v)$, $y(u,v)$ and $w(u,v)$ is described as x, y and w. NURBS surface partial derivatives, which are elements of Jacobian's matrix, can be calculated as follows [8]:

$$\frac{\partial S_X}{\partial u} = S_X \cdot \left(\frac{\partial x}{\partial u} \cdot \frac{1}{x} - \frac{\partial w}{\partial u} \cdot \frac{1}{w} \right) \tag{17}$$

and

$$\frac{\partial S_X}{\partial v} = S_X \cdot \left(\frac{\partial x}{\partial v} \cdot \frac{1}{x} - \frac{\partial w}{\partial v} \cdot \frac{1}{w} \right) \tag{18}$$

For S_Y:

$$\frac{\partial S_Y}{\partial u} = S_Y \cdot \left(\frac{\partial y}{\partial u} \cdot \frac{1}{y} - \frac{\partial w}{\partial u} \cdot \frac{1}{w} \right) \tag{19}$$

and

$$\frac{\partial S_Y}{\partial v} = S_Y \cdot \left(\frac{\partial y}{\partial v} \cdot \frac{1}{y} - \frac{\partial w}{\partial v} \cdot \frac{1}{w} \right) \tag{20}$$

Using equations (17)-(20) after transformation, the increment matrix (12) takes the form:

$$\begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \frac{w}{M} \cdot \begin{bmatrix} M_U \\ M_V \end{bmatrix} \tag{21}$$

where:

$$M = \begin{vmatrix} x & \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ y & \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \\ w & \frac{\partial w}{\partial u} & \frac{\partial w}{\partial v} \end{vmatrix} \tag{22}$$

and

$$M_U = \begin{vmatrix} x & x_R & \frac{\partial x}{\partial v} \\ y & y_R & \frac{\partial y}{\partial v} \\ w & 1 & \frac{\partial w}{\partial v} \end{vmatrix} \tag{23}$$

and

$$M_V = \begin{vmatrix} x & \frac{\partial x}{\partial u} & x_R \\ y & \frac{\partial y}{\partial u} & y_R \\ w & \frac{\partial w}{\partial u} & 1 \end{vmatrix} \tag{24}$$

The equation (11) can be described is as follows:

$$\begin{bmatrix} u_{i+1} \\ v_{i+1} \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} + \frac{w}{M} \cdot \begin{bmatrix} M_U \\ M_V \end{bmatrix} \tag{25}$$

How is can to see from (21) the increment matrix can describe the solving of next system of linear equations:

$$\begin{bmatrix} x & \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ y & \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \\ w & \frac{\partial w}{\partial u} & \frac{\partial w}{\partial v} \end{bmatrix} \cdot \begin{bmatrix} \Delta a^* \\ \Delta u^* \\ \Delta v^* \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \\ 1 \end{bmatrix} \tag{26}$$

where: Δa^* – redundancy root.

In this case the increment matrix (21) is partial solving of system of linear equations by Cramer rule and can be found as follows:

$$\begin{bmatrix} \Delta u^* \\ \Delta v^* \end{bmatrix} = w \cdot \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \tag{27}$$

For practical implementation can be better the Gaussian elimination use. In this case the system of linear equations can be described as extended matrix:

$$\left[\begin{array}{ccc|c} x & \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} & x_R \\ y & \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} & y_R \\ w & \frac{\partial w}{\partial u} & \frac{\partial w}{\partial v} & 1 \end{array}\right] \tag{28}$$

In this case the restriction is big number of division operator using.

2.5 Border criteria

As known the NURBS surface parameters is defended in diapason, what can be described as follows:

$$u \in [u_{\min};\quad u_{\max}] \,\&\, v \in [v_{\min};\quad v_{\max}] \tag{29}$$

In the process of the iterative procedure there can be a situation, that new parameters values is outside of diapason from (23). This case is showed in Fig. 2.

In this case the correction of the result is necessary. This task is divided into two parts: parameter *v* correction using parameter *u* border and the next step is parameter *u* correction using parameter *v* border.

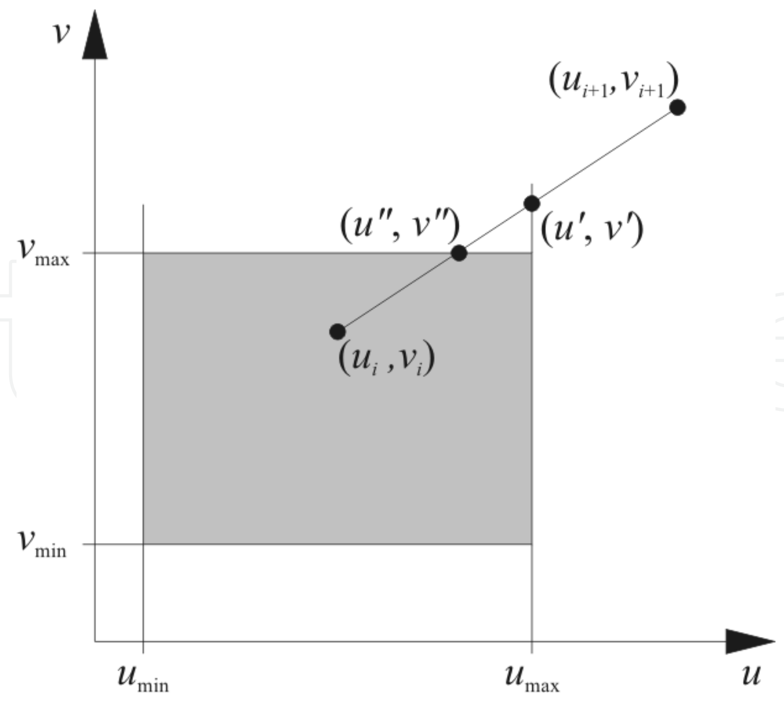

Fig. 2. Iteration step result outside parameters diapason.

First step can be described as follows: if equation $u_{i+1} \in [u_{\min};\ \ u_{\max}]$ is correct then we can go to the second step. Otherwise the correction can be described as follows:

$$u' = \begin{cases} u_{\min} & if \quad u_{i+1} < u_{\min} \\ u_{\max} & if \quad u_{i+1} >_{\max} \end{cases} \tag{30}$$

And the next step:

$$v' = \frac{v_{i+1} - v_i}{u_{i+1} - u_i} \cdot u' + \frac{u_{i+1} \cdot v_i - u_i \cdot v_{i+1}}{u_{i+1} - u_i} \tag{31}$$

If equation $v' \in [v_{\min};\ \ v_{\max}]$ is correct then the correction can be finished with the result of iteration step in point $(u'; v')$. Otherwise, the second correction step is necessary.

Second step can be described by analogy with the first step. The parameter *v* correction can be described as follows:

$$v'' = \begin{cases} v_{\min} & if \quad v' < v_{\min} \\ v_{\max} & if \quad v' > v_{\max} \end{cases} \tag{32}$$

And the next step, parameter *u* correction, as follows:

$$u'' = \frac{u' - u_i}{v' - v_i} \cdot v'' + \frac{u_i \cdot v' - u' \cdot v_i}{v' - v_i} \tag{33}$$

After second correction step it is possible to guarantee, that $u'' \in [u_{\min};\ \ u_{\max}]$ and $v'' \in [v_{\min};\ \ v_{\max}]$, and the result of iteration step is the point $(u''; v'')$.

2.6 Termination criteria

In this work, five criteria are used to decide when to terminate the Newton iteration. This criterion is analogical to termination criteria in work [6] and [13].

The first condition is the success criterion: if we are closer to the root than some predetermined ε_1 then we report an intersection:

$$\left\| \left[F(u_i, v_i) \right] \right\|_2 < \varepsilon_1 \tag{34}$$

where $\|\ldots\|_2$ - is norm of vector $[F]$ in 2D Euclidian space.

But in (25) is necessary to use next success criterion: if we are closer to the root values increment then some predetermined ε_2 and ε_3:

$$\begin{cases} |M_U| < \varepsilon_2 \\ |M_V| < \varepsilon_3 \end{cases} \tag{35}$$

It possible report an intersection point determination if is correct first or second termination criteria. Otherwise, we continue the iteration. The other three criteria are failure criteria, meaning that if they are met, we terminate the iteration and report a miss.

We do not allow the new (u_{i+1}, v_{i+1}) estimate to take us farther from the root then the previous one:

$$\left\| \left[F(u_{i+1}, v_{i+1}) \right] \right\|_2 > \left\| \left[F(u_i, v_i) \right] \right\|_2 \tag{36}$$

A maximum number of iteration steps has been performed, also indicating divergence:

$$i < i_{\max} \tag{37}$$

A final check is made to assure that the matrix $[M]$ is not singular. In the situation where $[M]$ is singular, either the surface is not regular or the ray is parallel to a silhouette ray at the point $S(u_i, v_i)$. In either situation, to determine singularity, we test:

$$\left| \det(M) \right| < \varepsilon_4 \tag{38}$$

3. Experimental results

In this work the proposed method, as well as the methods suggested by Martin et al. were implemented. In order to visualize a scene the 1 ray/pixel approach was used. The size of the obtained image is 512x512 pixels. 4 scenes were visualized during the experiment: the first scene – duck that what is taken from VRML programming language standard examples, the second scene visualized experimental object from the first scene, in total 27 VRML ducks, the third scene visualized experimental object – mobile phone and the fourth scene visualized practical object – modelled machine component.

All surfaces of experimental scenes were described with the help of NURBS surface. Achieved images were shown in Fig. 4-8.

As it is possible to see from these figures the proposed method gives an advantage on quality of the images (there's no distortion on the borders of patches). Image rendering time is shown in Table 2 and in Fig 3.

Objects	Proposed method, sec.	Martin et al. Method, sec.
"Duck"	5,578	6,437
"27 Ducks"	9,0	11,86
"Mobile phone"	4,297	5,422
"Machine component"	1,890	2,203

Table 2. Images rendering time in seconds

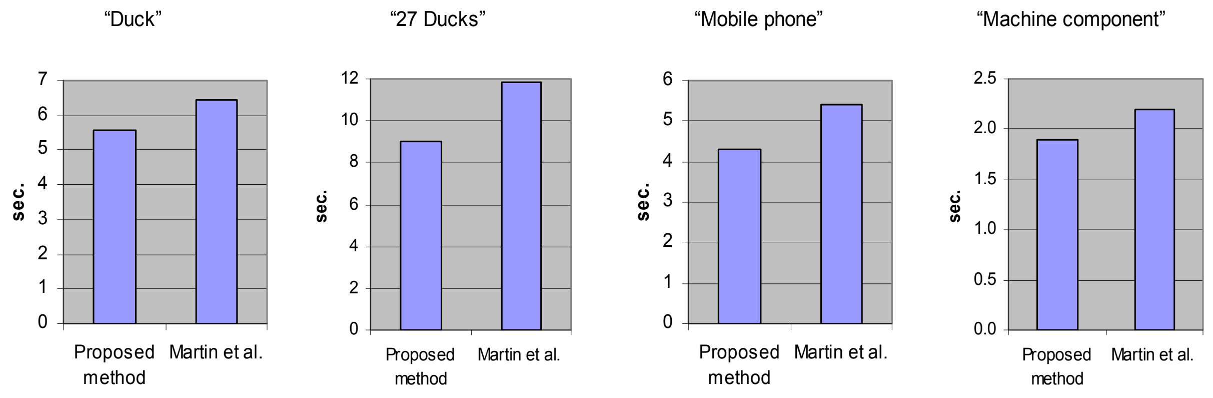

Fig. 3. Images rendering time

As seen from the table, the proposed method gives stable results in the fastest rendering time in our experiments (compared to method Martin et al.).

For comparison we shall consider the time of visualization in percentage, by taking earlier known method (Martin et al.) for 100%. The results are shown in Table 3.

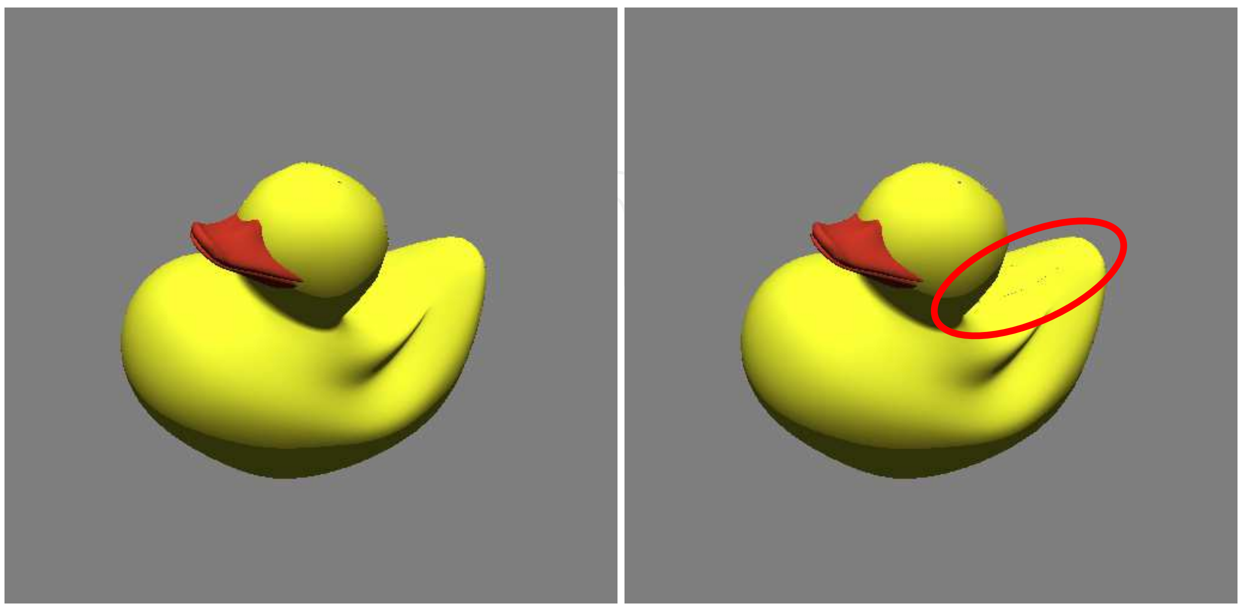

Fig. 4. The image of a scene "Duck" obtained using the proposed method (left) and the method Martin et al.(right).

Objects	Proposed method, %	Speed-up of rendering, %
"Duck"	86,66	13,34
"27 Ducks"	75,89	24,11
"Mobile phone"	79,251	20,749
"Machine component"	85,792	14,208

Table 3. Image rendering time in percentage (Martin et al. – 100%)

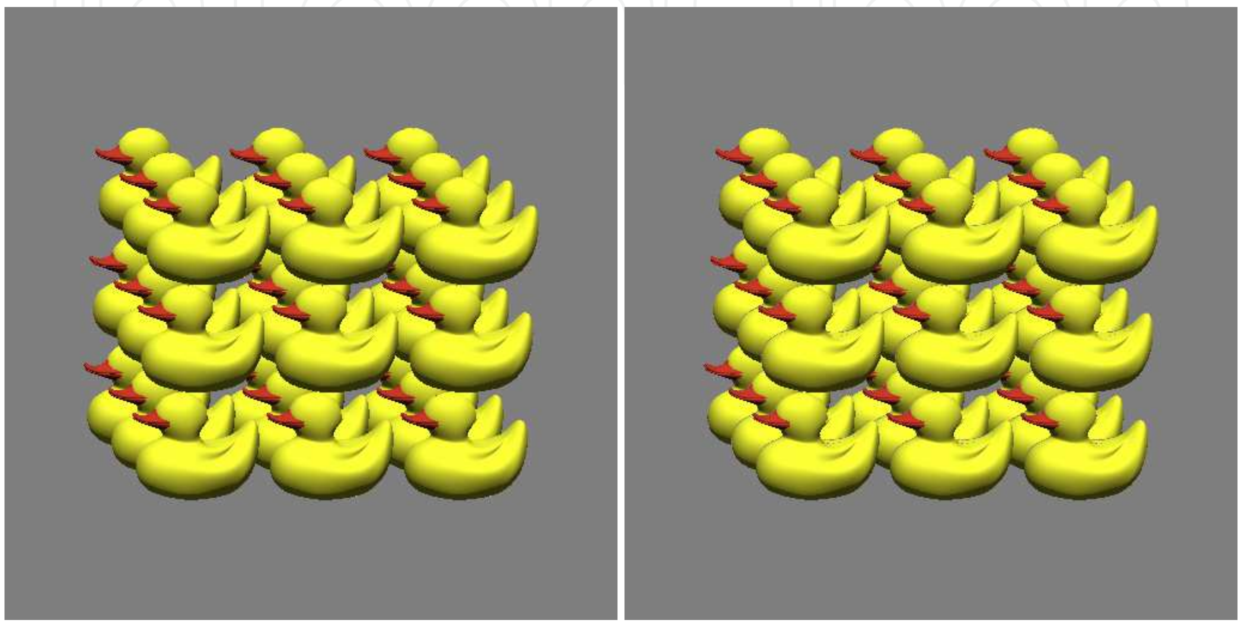
Fig. 5. The image of a scene "27 Ducks" obtained using the proposed method (left) and the method Martin et al.(right).

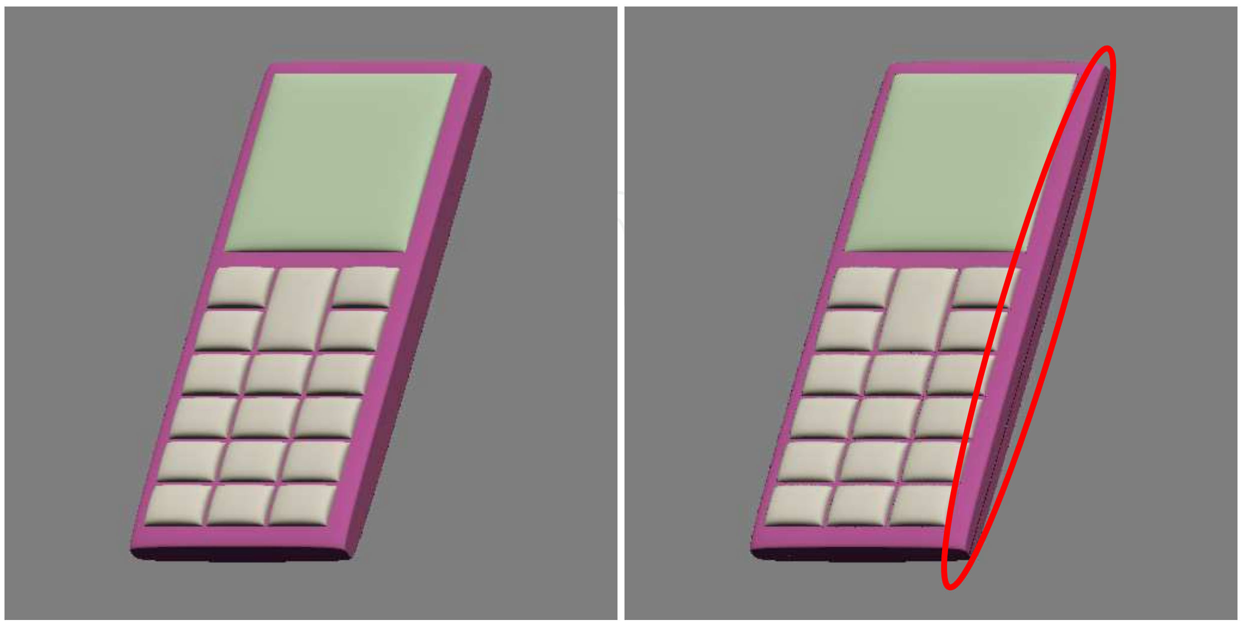
Fig. 6. The image of an object "Mobile phone" obtained using the proposed method (left) and the method Martin et al.(right).

As we can see from Table 3 data the proposed method gives stable visualization time reduction (compare with existing methods) in experiment. Table 3 proves that proposed method gives time reduction 13,3 – 24,1% in experiment in comparison with algorithm Martin et al.

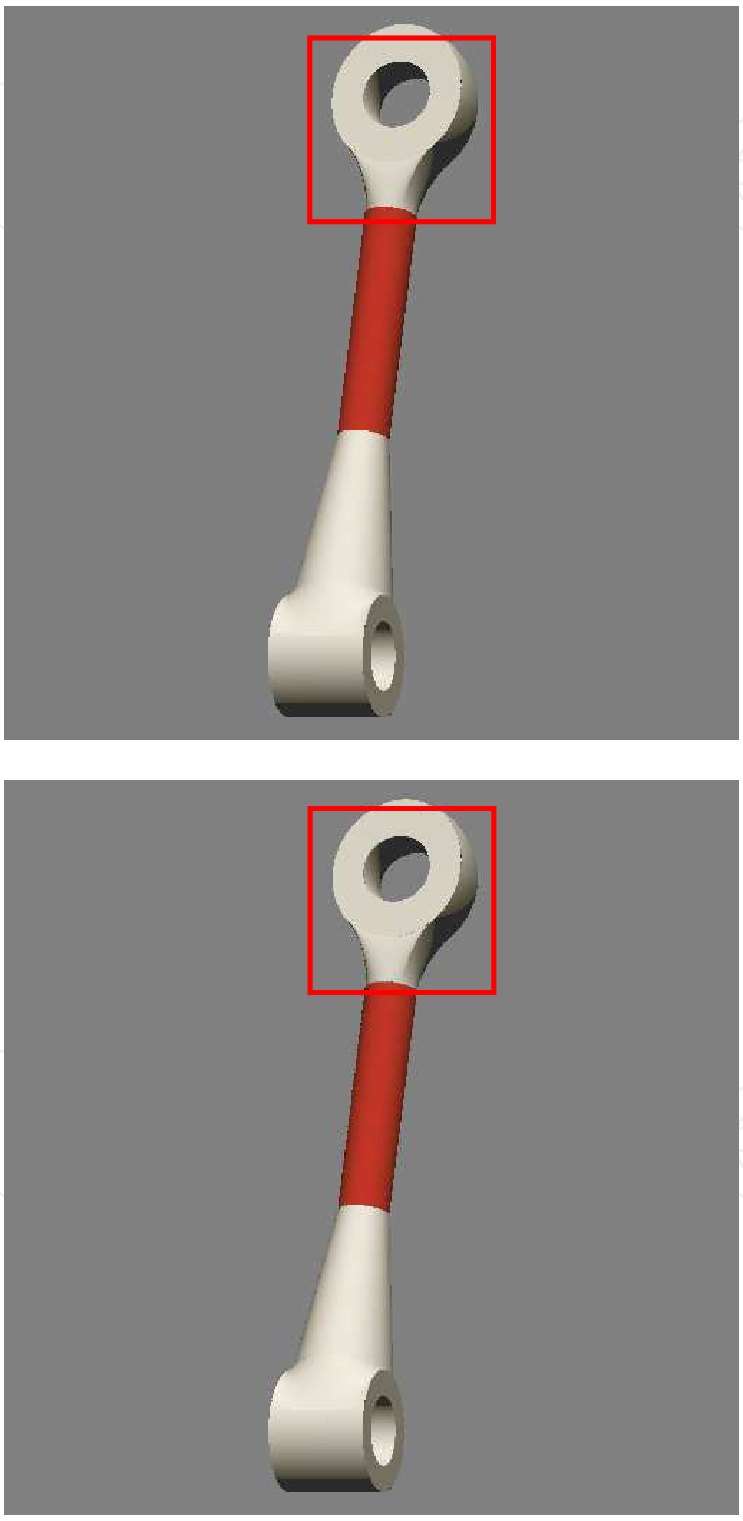

Fig. 7. The image of an object "Machine component" obtained using the proposed method (left) and the method Martin et al.(right).

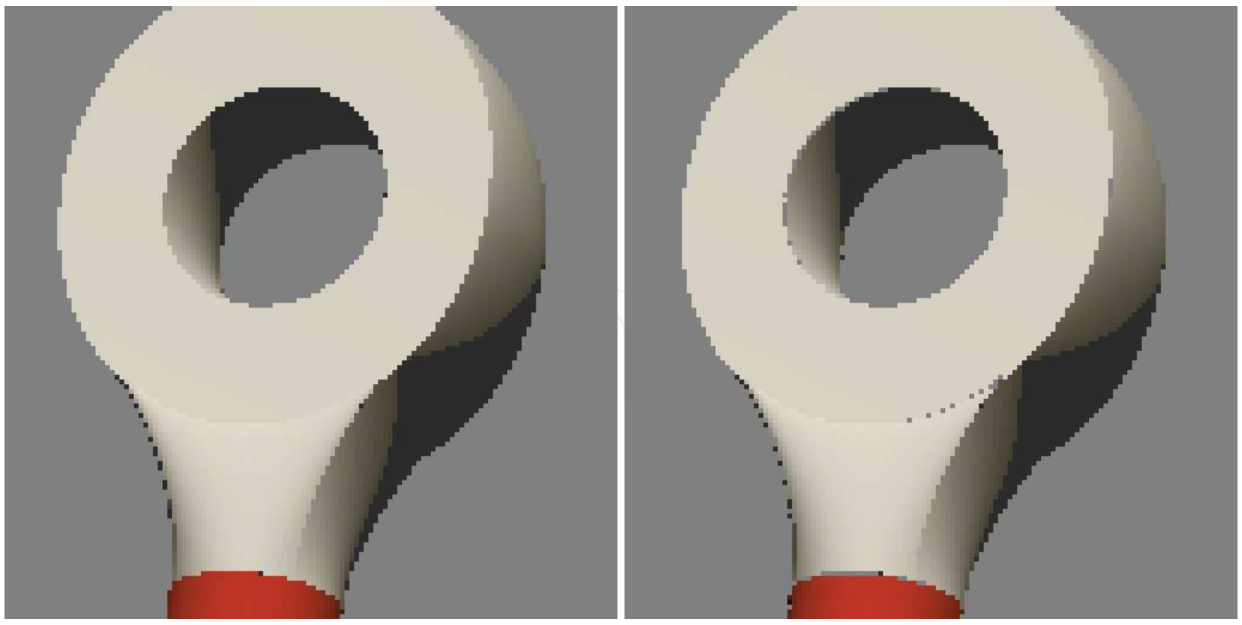

Fig. 8. The 4x enlarged fragment of image of an object “Machine component” obtained using the proposed method (left) and the method Martin et al.(right).

The next experiment that was conducted in order to check-up the described experiment is the comparison with existing CAD system. A comparison with Autodesk AutoCAD 2010 system was conducted. “Machine element” was chosen an object's example. To conduct object's visualization in CAD system Autodesk AutoCAD 2010 this object was made with NURBS surfaces in visualization programs and was imported to Autodesk AutoCAD 2010. To ensure correct comparison the object was colored in one color and equivalent lightening settings were adjusted. Fragments that were made larger in the visualization result, are shown in Fig. 9

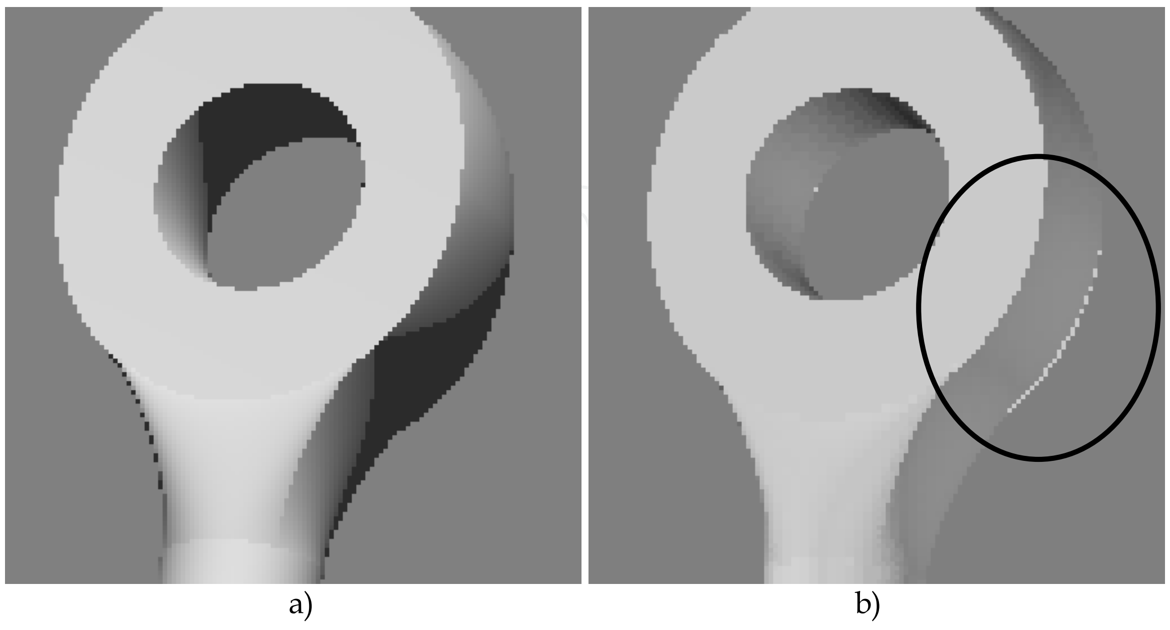

Fig. 9. Objects “Machine element” visualization, using:
a) proposed method, b) Autodesk AutoCAD 2010

From Fig. 9 we can see that the proposed method gives better result regarding image quality, because the object has not any defects on surface borders. Image visualization times in the experiment are following: 1, 89 seconds using proposed method; 10, 24 seconds using Autodesk AutoCAD 2010 visualization. As we can see from given data, proposed method give visualization time reduction in experiment (the difference is 5, 4 times).

4. Conclusion

In this work an efficient approach to direct NURBS surface rendering for ray tracing is proposed. The proposed approach based on Newton method and Cramer rule combination. The proposed approach, as well as the methods suggested by Martin et al. was implemented.

The results (in Figures 4 – 8) shows, that:

- As seen from comparison of Fig.4 – Fig.8. the use of the proposed method results in better quality of the image than Martin et al. method. The proposed method has no distortions on the image, while method Martin et al. has some faults (like distortion on border of patches).
- The modified proposed method results in faster image rendering time. This method works on 14% – 24% faster than the method of Martin et al.

5. References

[1] Efremov, A.; Havran, V.& Seidel, H.-P. (2005). Robust and numerically stable Bézier clipping method for ray tracing NURBS surfaces, *Proceedings of the 21st spring conference on Computer graphics*, pp. 127-135, ISBN 1-59593-204-6, New York, USA

[2] Efremov A., 2005, Efficient *Ray Tracing of Trimmed NURBS Surfaces*, Master's thesis, Computer Graphics Group, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 162 p.

[3] Hearn, D. & Baker, M.P. (2003) *Computer Graphics with OpenGL, 3rd Ed*. ISBN 978-0130153906, Prentice Hall, USA.

[4] Himmelblau, D., 1972. Applied Nonlinear Programming. McGraw-Hill Book Company, Boston, USA.

[5] Kajiya J.T., (1982) Ray Tracing Parametric Patches. *Proceedings of the 9th annual conference on Computer graphics and interactive techniques.* – Boston: SIGRAPH, 1982. – pp. 245 – 254.

[6] Martin, W. et al, 2000. *Practical Ray Tracing of Trimmed NURBS Surfaces*. Journal of Graphics Tools, Vol. 5, No. 1, pp. 27-52.

[7] Nishita, T. et al, 1990. Ray tracing trimmed rational surface patches. Journal ACM SIGGRAPH Computer Graphics, Vol. 24, No. 4, pp. 337–345.

[8] Rogers, D.F. and Adams, J.A., 1990. Mathematical Elements for Computer Graphic, 2nd Ed. McGraw-Hill Book Company, Boston, USA.

[9] Schollmeyer A., and Froehlich, B., 2009. *Direct Trimming of NURBS Surfaces on the GPU*, Jounal ACM Transactions on Graphics, Vol. 28, No. 3, Article 47.

[10] Taha H.A., 2003. Operations Research: an Introduction, 7th Ed. Prentice Hall, New Jersey, USA.

[11] Wang S.-W., Shih Z.-C., Chang R.-C. (2001). An Efficient and Stable Ray Tracing Algorithm for Parametric Surfaces, Journal of Information Science and Engineering, Nr.18. – pp. 541 – 561.
[12] Woodward C. 1989. Ray tracing parametric surfaces by subdivision in viewing plane. Theory and practice of geometric modeling, Springer-Verlag, New York, USA, pp. 273 – 287.
[13] Yang C.-G. 1987. On speeding up ray tracing of B-spline surfaces. Journal Computer Aided Design, Vol. 19, No. 3, pp. 122-130.