TIME-EFFICIENT NURBS CURVE EVALUATION ALGORITHMS

Kestutis Jankauskas

Kaunas University of Technology, Department of Multimedia Engineering,
Studentu st. 50, LT-51368 Kaunas, Lithuania, kestutis.jankauskas@ktu.lt

Abstract: This paper analyses time-efficiency of existing NURBS evaluation algorithms. The most competitive computation methods are modified to achieve even better performance. Performance tests indicate that NURBS curve evaluation time-efficiency can be improved in uniform and non-rational B-spline cases. Suggested optimizations are very effective in the evaluation of higher degree splines with a larger number of control points.

Keywords: NURBS, curve evaluation, inverted triangular scheme

1 Introduction

NURBS stands for Non-Uniform Rational B-Spline. It is the most popular spline representation in today's commercial CAD packages [1, 4, 5, 8, 10]. NURBS is able to represent large variety of shapes, like circles, hyperbolas, parabolas, and still preserves mathematical exactness [5].

Generally, a spline is a smooth curve interpolated among given control points. Unfortunately, a spline cannot be constructed in the model space directly. Each point on NURBS curve or surface must be calculated from the set of control points, knot vector, and basis function of specific degree. This process is called NURBS evaluation [1, 5, 8].

In the following sections we will discuss theoretical aspects of NURBS as well as existing evaluation algorithms. Moreover, we will introduce certain modified evaluation algorithms and strategies for uniform and non-rational cases that improve evaluation performance. Finally we will compare actual time-efficiency of suggested method implementations.

2 NURBS in Theory

Acronym NURBS defines special properties of this particular spline: (1) Non-Uniform, (2) Rational, (3) B-Spline. Let us clarify those properties by starting from the last one. It has the basis function of B-spline, which ensures smooth blending [2] of control point influence over the curve. All theory regarding a regular B-spline is covered in Section 2.1. Rational property gives more flexibility to a spline [4, 5, 10], but also increases complexity. It is achieved by adding weights to control points. Rational B-spline is presented in Section 2.2. Finally, introducing the editable knot vector to the spline concept, allows usage of non-uniform piece-wise features [1, 10]. They are covered in Section 2.3.

2.1 B-Spline

Regular B-spline is defined by a set of control points P_i, a knot vector $U = \{u_j\}$, and degree p, where $i = 0..n-1$, $j = 0,1..m$ and $m = n + p + 1$ [1, 4, 6, 9]. Control points are located in the multi-dimensional space we refer to as the model space. The spline interpolates between control points with the help of the basis function:

$$C(u) = \sum_{i=0}^{i<n} N_{i,p}(u) P_i \,, \tag{1}$$

where p is the degree of the basis function, u is a coordinate in the parametric spline space and $C(u)$ is point on curve in the model space. Accordingly, any point on the curve is obtained by summing multiplications of control points P_i and basis functions $N_{i,p}(u)$. The basis function is calculated from the expression:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} , \tag{2}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \,, \tag{3}$$

where u is the coordinate in the parametric spline space and u_j are values from the knot vector U. The last expression is referred to as Cox-de Boor recursion formula [1, 3]. It denotes that basis function domains are divided by elements of the knot vector, i.e. knots [1, 9]. It is also known that the sum of all basis functions $N_{i,p}(u)$ equals one [7]. The sum of $N_{i,p}(u)$ in the interval $k - p \le i \le k$ equals one as well, where $u_k \le u < u_{k+1}$ (the partition of unity [2, 6]):

$$\sum_{i=k-p}^{i\le k} N_{i,p}(u) = 1. \tag{4}$$

As the basis function is non-negative, it means that all basis functions $N_{i,p}(u)$ outside the interval $k-p \le i \le k$ are zero. Consequently, all multiplications $N_{i,p}(u)P_i$ outside the same interval are zero. Thus, such control points has no effect on the portion of the curve within $u_k \le u < u_{k+1}$. So any point on the curve $C(u)$ of the uniform B-spline is affected by $p+1$ control points, with the exception of $C(0)$ and $C(1)$. These special cases can be explained through analysis of the knot vector.

The knot vector is a set of non-decreasing values $u_j \le u_{j+1}$, where $j = 0,1..m-1$ and $m = n + p + 1$. In this paper we use a normalized knot vector form, so the parametric space of the B-spline and knot vector values are bounded by 0 and 1. Also, it is a common practice to use the clamped knot vector, where the first $p+1$ values equal 0 and the last $p+1$ values equal 1 [1, 3, 10]:

$$U = \{u_0 = u_1 = ... = u_p = 0 \le u_{p+1} \le u_{p+2} \le ... \le u_{n-1} \le u_n = .. = u_{n+p} = u_{n+p+1} = 1\}. \tag{5}$$

Let us examine the example of a cubic B-spline ($p=3$), defined by eight control points ($n=8$) and $m=12$ uniform knots: $U = \{0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1\}$. Basis functions are given in Figure 2. For $u = 0.3$ functions $N_{1,3}(0.3)$, $N_{2,3}(0.3)$, $N_{3,3}(0.3)$ and $N_{4,3}(0.3)$ are greater than zero, other functions are zero. This means that a point on the curve $C(0.3)$ is affected by the position of four control points: P_1, P_2, P_3, and P_4. Also function $N_{2,3}(0.3)$ and $N_{3,3}(0.3)$ values are significantly greater than values of $N_{1,3}(0.3)$ and $N_{4,3}(0.3)$. This suggests that the point $C(0.3)$ is closer to P_2 and P_3 than to P_1 and P_4.

In the case when u is in the position of the knot $u = u_5 = 0.4$ we have only three non-zero functions: $N_{2,3}(0.3) \ne 0$, $N_{3,3}(0.3) \ne 0$ and $N_{4,3}(0.3) \ne 0$. Consequently $C(0.4)$ is affected by three control points: P_2, P_3, and P_4. So, a point on the curve $C(u)$ is affected by $p - s + 1$ control points, where s is knot multiplicity at u. Therefore the curve becomes C^{p-s} continuous at this point (here the symbol C refers to spline continuity and has a different meaning than $C(u)$, see Section 2.3 for more details) [1, 9].

Because of the clamped knot vector the first curve point $C(0)$ is affected only by one control point P_0. The last curve point $C(1)$ is affected by the control point P_{n-1} respectively:

$$C(0) = P_0, \tag{6}$$

$$C(1) = P_{n-1}. \tag{7}$$

2.2 Rational B-Spline

Regular B-spline is quite powerful interpolation tool, but it lacks flexibility. B-spline can not represent conic sections, like circles [4, 5, 9]. Therefore a rational form is used to cover these cases [4, 10]:

$$C(u) = \frac{1}{\sum_{i=0}^{i<n} N_{i,p}(u) w_i} \sum_{i=0}^{i<n} N_{i,p}(u) w_i P_i, \tag{8}$$

where the weight $w_i > 0$ is attached to every control point.

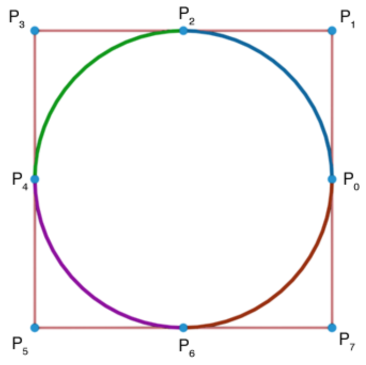


Figure 1. A circle represented by four rational B-splines

Let us take a look at the example of a circle represented as four rational B-splines in Figure 1. Each quarter of the circle is constructed from separate rational quadratic B-spline defined by control point sequences: $\{P_0, P_1, P_2\}$, $\{P_2, P_3, P_4\}$, $\{P_4, P_5, P_6\}$, $\{P_6, P_7, P_0\}$. The weights of the first and the last control points in each sequence are 1. The weight of the middle control point is $\sqrt{2}/2$ [4]. Greater weights pull the curve towards the control point and lesser weights push the curve away [1, 4, 10]. Naturally, regular B-spline is a special case of rational B-spline when all weights are equal to 1.

2.3 Non-uniform Rational B-Spline

The term of uniformity is used to define a relation between the sequence of control points and the parametric spline space. As mentioned in Section 2.1, control point influence over the curve is defined by basis functions and function domains are divided by knots [9]. This means that property of uniformity is embedded into the knot vector [1]. Until now we considered a knot vector to be clamped and uniform:

$$U = \{u_0 = ... = u_p = 0 ... u_i = \frac{i-p}{n-p} ... u_n = ... = u_{n+p+1} = 1\}, \tag{9}$$

where $p+1 \le i < n$. Such a knot sequence divides whole parametric space into uniform intervals. Each of intervals contains $p+1$ non-zero basis functions, thus the curve is affected by $p+1$ control points in this interval (see Section 2.1). In general case, knots can be distributed in non-uniform manner. However a knot sequence must be non-decreasing, as shown in the expression (5).

Let us take the example of the knot vector $U = \{0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1\}$ and modify it by setting $u_6 = u_5 = u_4 = 0.2$: $U = \{0, 0, 0, 0, 0.2, 0.2, 0.2, 0.8, 1, 1, 1, 1\}$. Knot multiplicity of $s = p$ at $u = 0.2$ leaves only one non-zero function at this point (see Figure 3), which suggest that $C(0.2)$ is affected by single control point. Therefore, the curve goes through this control point: $C(0.2) = P_3$. In other words, the knot of multiplicity s reduces curve continuity at that knot by s [3, 10]. In this example the curve becomes $C^{p-s} = C^0$ continuous at $u = 0.2$. Further increment of multiplicity is pointless, because it excludes control points from affecting the curve.

NURBS is powerful enough to compose any shape. Recall the example of the circle in Figure 1. It was represented by four uniform rational B-splines. Knot multiplication in the knot vector allows the construction of such a shape from single quadratic NURBS curve. The same control points with weights are used in the sequence $\{P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_0\}$. The last control point is the same as the first to close the curve. Instead of multiple control points, multiple knots are employed: $U = \{0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1\}$ [4]. Behavior of basis functions is depicted in Figure 4. Notice that every quarter of the circle is represented by single non-zero knot interval and each quarter is independent.

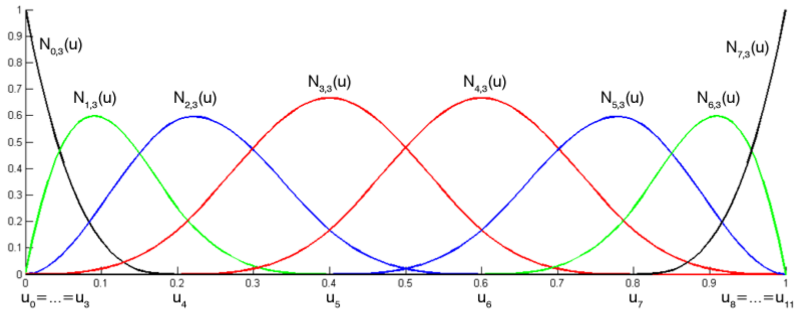


Figure 2. Basis functions of uniform cubic B-spline defined by eight control points

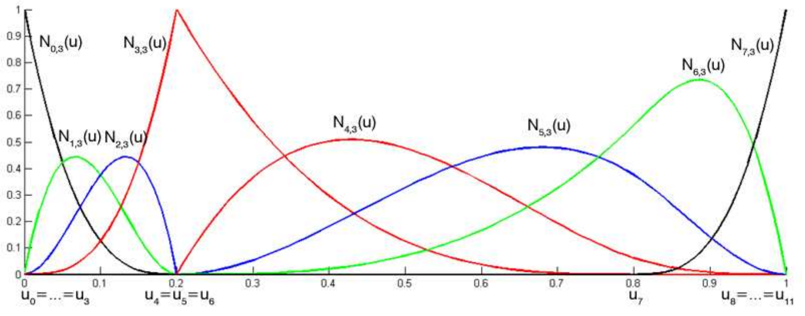


Figure 3. Basis functions of cubic B-spline with knot multiplicity of three at 0.2

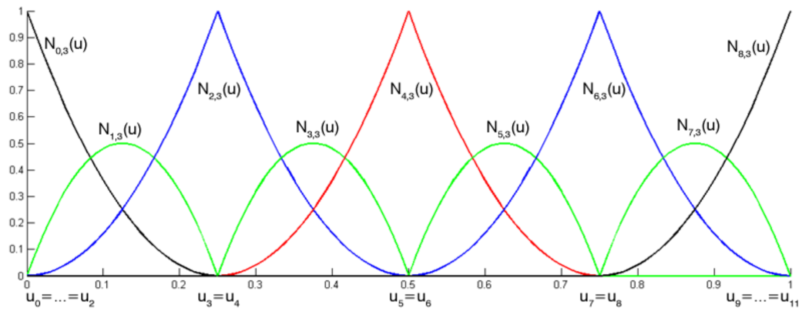


Figure 4. Basis functions of quadratic NURBS defined by the knot vector U={0, 0, 0, 1/4, 1/4, 2/4, 2/4, 3/4, 3/4, 1, 1, 1}

3 NURBS evaluation algorithms

To represent NURBS in the model space (Cartesian multi-dimensional space) as a curve, the spline must be evaluated at multiple u, where $0 \le u \le 1$. According to the expression (8) basis functions are necessary in order to do so. Several basis function calculation methods are covered in Section 3.1. Once basis functions are known, they can be used to determine a point on the curve. The description of single point evaluation algorithms can be found in Section 3.2. Finally, entire NURBS curve evaluation strategies are presented in Section 3.3.

3.1 Basis function

As we already discussed in Section 2.1, calculation of all basis functions is not necessary. There are only $p-s+1$ non-zero basis functions at any u, where p is the degree of the basis function and s is knot multiplicity at u. So we are to obtain all basis functions from $N_{k-p,p}(u)$ to $N_{k,p}(u)$, where $u_k \le u < u_{k+1}$.

3.1.1 Cox-de Boor recursion

The most obvious solution is to use a standard Cox-de Boor recursion formula, given in the expression (2) and (3). Although this formula is simple to understand and easy to implement, [6] and [9] sources state that it involves many unnecessary calculations. Figure 5 illustrates how $N_{k,p}(u)$ is obtained.

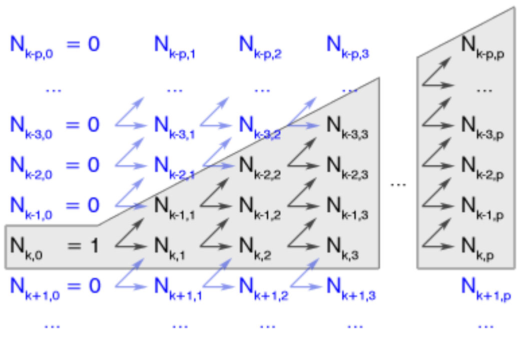


Figure 5. Computation of non-zero basis functions

Zero functions are marked in blue. They have no effect on higher degree functions in successive iterations, because multiplication by zero is zero (blue arrows). In the example of $U=\{0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1\}$, where $p=3$ and $k=4$ (u is in the interval $u_4 \le u < u_5$), the recursive formula returns non-zero values of $N_{1,3}(u)$, $N_{2,3}(u)$, $N_{3,3}(u)$, and $N_{4,3}(u)$. Accordingly to the expression (3), to obtain $N_{1,3}(u)$ the algorithm calculates $N_{1,2}(u)$ and $N_{2,2}(u)$. To acquire second degree functions, the recursion must obtain first degree functions $N_{1,1}(u)$, $N_{2,1}(u)$ and $N_{2,1}(u)$, $N_{3,1}(u)$. Finally, first degree functions is calculated from zero degree functions: $N_{1,1}(u)$ is acquired from $N_{1,0}(u)$ and $N_{2,0}(u)$, $N_{2,1}(u)$ is acquired from $N_{2,0}(u)$ and $N_{3,0}(u)$, $N_{3,1}(u)$ is acquired from $N_{3,0}(u)$ and $N_{4,0}(u)$. Notice that $N_{2,1}(u)$ is calculated twice, so $N_{2,0}(u)$ as well as $N_{3,0}(u)$ is actually calculated three times. Moreover, only $N_{4,0}(u)$ is non-zero among all zero degree functions.

This example illustrates how Cox-de Boor recursion formula is overloaded with unnecessary calculations. Naturally, the evaluation of higher degree B-spline basis functions yields even more unnecessary iterations. Also the expression (3) is numerically unstable, because of $0/0$ cases [5]. Another drawback is noted in [2]. The recursion formula gives an incorrect result when $u = 1$. The last point on the curve is always $C(1) = \{0, 0, 0\}$. To overcome this problem, we simply use expressions (6) and (7) as special cases, so $C(0)$ and $C(1)$ can be found without the calculation of basis functions.

3.1.2 Inverted Triangular Scheme

To avoid unnecessary calculations, authors in [6] present the algorithm based ITS (inverted triangular scheme). It is given as **Basis_ITS0** function in pseudo code. It calculates functions from lower to higher degree in contrast to the recursive algorithm. Also, they suggest rearrangement of the expression (3) to remove operation duplications:

$$N_{k-j,p}(u) = \frac{L_{j+1}}{R_j + L_{j+1}} N_{k-j,p-1}(u) + \frac{R_{j+1}}{R_{j+1} + L_j} N_{k-j+1,p-1}(u), \qquad (10)$$

$$\text{where } L_j = u - u_{k+1-j} \text{ and } R_j = u_{k+j} - u. \qquad (11)$$

```
Basis_ITS0(k, p, u)
1.  N[0] = 1
2.  for (j = 1; j <= p; j++)
    2.1. saved = 0
    2.2. L[j] = u - knots[k + 1 - j]
    2.3. R[j] = knots[k + j] - u
    2.4. for (r = 0; r < j; r++)
        2.4.1.  tmp = N[r] / (R[r + 1] + L[j - r])
        2.4.2.  N[r] = saved + R[r + 1] * tmp
        2.4.3.  saved = L[j - r] * tmp
    2.5. N[j] = saved
3.  return N
```

Note that k should already be known, where k defines the knot interval in which u resides. Therefore, the method **FindKnotSpan** (available in [6]) must be applied to determine k before the implementation of **Basis_ITS0**.

3.1.3 Modified Inverted Triangular Scheme

We noticed another relation. Let the right part of the sum in $N_{i,p}(u)$ be equal A, then the left part of the sum in $N_{i-1,p}(u)$ is always $1 - A$. Based on this observation, we propose another modification of the expression (3):

$$N_{i,p}(u) = A_{i,p}(u) \cdot N_{i,p-1}(u) + (1 - A_{i+1,p}) \cdot N_{i+1,p-1}(u), \qquad (12)$$

$$\text{where } A_{i,p}(u) = (u - u_i)/(u_{i+p} - u_i) \text{ and } k - p \le i \le k. \qquad (13)$$

The example of non-zero cubic basis function calculation is given in Table 1, followed by modified ITS algorithms. As u value is fixed we omit the notation of (u).

Table 1. Non-zero basis function calculations for cubic B-spline, using a modified ITS

i	$p = 0$	$p = 1$	$p = 2$	$p = 3$
$k-3$				$N_{k-3,3} = (1 - A_{k-2,3})N_{k-2,2}$
$k-2$			$N_{k-2,2} = (1 - A_{k-1,2})N_{k-1,1}$	$N_{k-2,3} = A_{k-2,3}N_{k-2,2} + (1 - A_{k-1,3})N_{k-1,2}$
$k-1$		$N_{k-1,1} = (1 - A_{k,1})N_{k,0}$	$N_{k-1,2} = A_{k-1,2}N_{k-1,1} + (1 - A_{k,2})N_{k,1}$	$N_{k-1,3} = A_{k-1,3}N_{k-1,2} + (1 - A_{k,3})N_{k,2}$
k	$N_{k,0} = 1$	$N_{k,1} = A_{k,1}N_{k,0}$	$N_{k,2} = A_{k,2}N_{k,1}$	$N_{k,3} = A_{k,3}N_{k,2}$

```
Basis_ITS1(k, p, u)
1.  N[0] = 1
2.  for (i = 1; i <= p; i++)
    2.1. for (j = i – 1; j >= 0; j--)
        2.1.1.  A = (u - knots[k - j]) /
                (knots[k + i - j] - knots[k - j])
        2.1.2.  tmp = N[j] * A
        2.1.3.  N[j + 1] += N[j] - tmp
        2.1.4.  N[j] = tmp
3.  return N
```

```
Basis_ITSU(k, p, u)
1.  N[0] = 1
2.  M = (u - knots[k])/(knots[k+1]-knots[k])
3.  for (i = 1; i <= p;  i++)
    3.1. for (j = i – 1; j >= 0; j--)
        3.1.1.  tmp = N[j] * (M + j)/i
        3.1.2.  N[j + 1] += N[j] - tmp
        3.1.3.  N[j] = tmp
4.  return N
```

These algorithms return basis functions in reversed order: from $N_{k,p}(u)$ to $N_{k-p,p}(u)$. **Basis_ITS0** and **Basis_ITS1** algorithms are suitable for any NURBS. Only few CAD and CAM applications allow editing the knot vector, because such modification is not intuitive [10]. Hence, in many cases NURBS stays uniform. From the expression (9) it is obvious that every non-zero interval in the knot vector equals $1/(n-p)$. Let us presume that $M = A_{k,1} = (u-u_k)/(u_{k+1}-u_k)$. It is easy to calculate that $A_{k,p} = M/p$, $A_{k-1,p} = (M+1)/p$, $A_{k-2,p} = (M+2)/p$. So, in case of the uniform knot vector, the expression (13) can be simplified:

$$A_{k-j,p} = \frac{M+j}{p}. \tag{14}$$

Plugging the expression (13) into the last row of Table 1 indicates that calculation of a non-zero function set uses knots from u_{k-p+1} to u_{k+p}. So the equation (14) is valid when all knot intervals from u_{k-p+1} to u_{k+p} are equal. In the case of the clamped knot vector, the first p and last p knot intervals are zero. As the first uniform interval begins at u_p and the last uniform interval ends at u_n, the expression (14) can be used for all intervals from u_{p+p-1} to u_{n-p}. This means that the ITS algorithm can be written as **Basis_ITSU** for all $u_k \le u < u_{k+1}$, where:

$$2p-1 \le k \le n-p. \tag{15}$$

3.2 Single point on curve

Each of non-zero functions defines how strongly a certain control point affects a curve (see Section 2.1). According to the expression (8), the strength of the effect is also modified by weights of control points (see Section 2.2). In order to calculate $C(u)$, we require a sum of all $N_{i,p}(u)w_iP_i$ divided by the sum of $N_{i,p}(u)w_i$, where $k-p \le i \le k$ and $u_k \le u < u_{k+1}$. Following algorithms calculate a point on the curve, when basis functions are known. Thus **GetPoint0** should be used after **Basis_ITS0**. Because of the inverted function order in **Basis_ITS1** and in **Basis_ITSU**, those algorithms should be followed by **GetPoint1**.

```
GetPoint0(N, k)
1.  Nsum = 0
2.  Cu = {0, 0, 0}
3.  for (i = 0; i <= p; i++)
    3.1. Nsum += N[i] *= P[k - p + i].Weigth
    3.2. Cu += N[i] * P[k - p + i].To3D()
4.  return Cu/Nsum
```

```
GetPoint1(N, k)
1.  Nsum = 0
2.  Cu = {0, 0, 0}
3.  for (i = 0; i <= p; i++)
    3.1. Nsum += N[i] *= P[k - i].Weigth
    3.2. Cu += N[i] * P[k - i].To3D()
4.  return Cu/Nsum
```

The method **To3D()** returns $\{x, y, z\}$ coordinates and ignores the control point's weight. If a spline is regular B-spline and all weights equal 1, we can use the expression (1) instead of the expression (8) to find a certain point on the curve. In such case **GetPoint1** algorithm can be simplified to **GetPoint_NR1**:

```
GetPoint_NR1(N, k)
1.  Cu = {0, 0, 0}
2.  for (i = 0; i <= p; i++)
    2.1. result += N[i] * P[k - i].To3D()
3.  return Cu
```

3.2.1 De Boor's algorithm

There are several B-spline evaluation techniques that do not need basis functions to determine a point on the curve, like de Boor's algorithm [9]. De Boor's algorithm is based on observation that $C(u)$ is positioned at the location of the control point P_{k-p}, when $u = u_k$ and knot multiplicity at u equals p (see section 2.3). How do we make desired knot multiplicity at any u? The author in [9] suggests a multiple insertion of a knot at u. The insertion of an additional knot also means the insertion of a new control point, thus after p iterations the last control point is exactly at the position of $C(u)$. In case when u is already at the position of the knot u_k with multiplicity s, only $p-s$ iterations of the insertion are required. The position of every new control point can be found from expressions [3, 9]:

$$Q_i^w = (1-a_i)P_{i-1}^w + a_iP_i^w, \tag{16}$$

$$\text{where } a_i = \frac{u-u_i}{u_{i+p}-u_i} \text{ for all } k-p+1 \le i \le k. \tag{17}$$

However, the actual insertion of knots is not performed, because this would lead to the modification of the control point sequence during the evaluation. Thus the sequence of new control points is processed in a temporary array. The expression (16) requires control points to be converted to a homogenous 4D coordinate system by multiplying coordinates by weight: $P_i^w = \{w_i \cdot x_i, w_i \cdot y_i, w_i \cdot z_i, w_i\}$. This task is performed by **ConvertTo4D()** method. The conversion back to Cartesian 3D coordinate system is performed by dividing coordinates by weight $P_i = \{x_i / w_i, y_i / w_i, z_i / w_i, w_i\}$ in **ConvertTo3D()** method.

```
GetPoint_DeBoor(k, u)
1.  s = 0
2.  while (k >= s && knots[k - s] == u)
    2.1. s++
3.  Q = new ControlPoint[p - s + 1]
4.  for (i = k - p; i <= k - s; i++)
    4.1. Q[i - k + p] = P[i].ConverTo4D()
5.  for (r = 1; r <= p - s; r++)
    5.1. for (i = k - s; i >= k - p + r; i--)
        5.1.1.  a = (u - knots[i]) / (knots[i + p - r + 1] - knots[i])
        5.1.2.  j = i - k + p
        5.1.3.  Q[j] = (1 - a) * Q[j - 1] + a * Q[j]
6.  return Q[p-s].ConvertTo3D().To3D()
```

3.3 Multiple Points on Curve

Generally, evaluation of multiple points can be done using single point evaluation several times. But several optimizations can be made. To evaluate entire NURBS curve, we must obtain multiple points $C(u)$, where $u = 0, \Delta u, 2\Delta u \ldots 1 - \Delta u, 1$ and $\Delta u = 1/(steps - 1)$ is the step in the parametric spline space. Under these conditions the initial knot interval is $u_p \le u < u_{p+1}$, thus initial $k = p$. Successive k values can be traced easily, so the procedure **FindKnotSpan** in not needed. Also $u = 0$ and $u = 1$ are handled as special cases (see Section 3.1) and calculated from expressions (6) and (7). The following algorithm evaluates the number of points equal to $steps$ on any NURBS curve.

```
NURBS_ITS0(steps)
1.  step = 1 / (steps - 1)
2.  Cu = new Point[steps]
3.  Cu[0] = P[0].To3D()
4.  iter = 1
5.  u = knots[p] + step
6.  for (k = p; k < n; k++)
    6.1. while (knots[k] == knots[k + 1] && knots[k] < 1)
        6.1.1.  k++
    6.2. while (u < knots[k + 1])
        6.2.1.  N = Basis_ITS0(k, p, u)
        6.2.2.  Cu[iter] = GetPoint0(N, k)
        6.2.3.  iter++
        6.2.4.  u += step
7.  C[steps - 1] = P[n - 1].To3D()
8.  return Cu
```

Algorithms in steps 6.2.1 and 6.2.2 can be replaced by modified **Basis_ITS1** and **GetPoint1** respectively. If a spline is known to be non-rational then **GetPoint_NR1** can be used in step 6.2.2. If a spline is uniform it is possible to optimize this algorithm even further.

3.3.1 Evaluation of Uniform B-spline Curve

Recall Section 3.1.3 and expressions (15), which states that **Basis_ITSU** can be used instead of **Basis_ITS1** within bounds of $2p - 1 \le k \le n - p$. Figure 6 illustrates the basis functions of the cubic uniform B-spline defined by $m = 17$ knots. Notice that $N_{0,3}(0) = N_{12,3}(1)$, $N_{1,3}(0.05) = N_{11,3}(0.95)$, $N_{2,3}(0.1) = N_{10,3}(0.9)$ and so on. Clearly, certain basis functions of the uniform B-spline are symmetrical to each other. Actually, any function $N_{i,p}(u)$ can be reflected to $N_{n-i-1,p}(1-u)$ at the middle point of the parametric space. We refer to this operation as to ref :

$$ref : N_{i,p}(u) \to N_{n-i-1,p}(1-u)\,, \qquad (18)$$

$$\text{where } k - p \le i \le k\,. \qquad (19)$$

The set of non-zero functions $N_{k-p,p}(u)\dots N_{k,p}(u)$ at $0 \le u_k \le u < u_{k+1} < 0.5$ can be cloned to $N_{n-k-1,p}(1-u)\dots$ $N_{n-k+p-1,p}(1-u)$. In other words, there is no need to calculate non-zero functions for the second half of the parametric space, because they can be obtained from the first one.

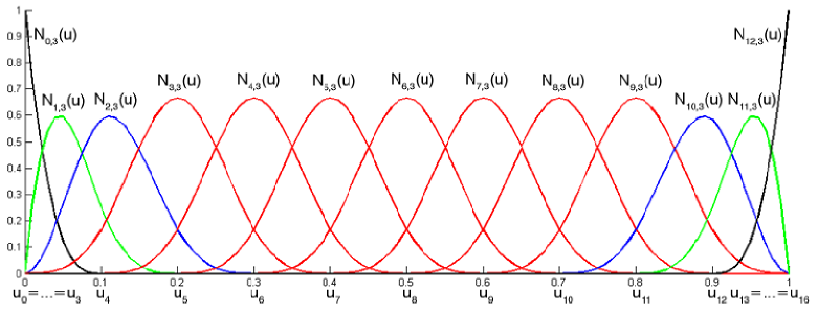


Figure 6. Basis functions of cubic uniform B-spline defined by 13 control points (17 knots)

Figure 6 also depicts another important property of uniform B-spline. Pay attention to functions marked as red, they are identical: $N_{3,3}(0.2) = N_{4,3}(0.3) = N_{5,3}(0.4) = \dots = N_{9,3}(0.8)$. The set of non-zero functions at $u = 0.22$ consists of four functions: $N_{2,3}(0.22)$, $N_{3,3}(0.22)$, $N_{4,3}(0.22)$, and $N_{5,3}(0.22)$. There is a set of functions with the same values at each interval u_k, where $5 \le k \le 10$: $N_{2,3}(0.22) = N_{3,3}(0.32) = \dots = N_{7,3}(0.72)$, $N_{3,3}(0.22) = N_{4,3}(0.32) = \dots = N_{8,3}(0.72)$, $N_{4,3}(0.22) = N_{5,3}(0.32) = \dots = N_{9,3}(0.72)$, and $N_{5,3}(0.22) = N_{6,3}(0.32) = \dots = N_{10,3}(0.72)$. Obviously, non-zero functions at arbitrary $u_{2p-1} \le u < u_{2p}$ can be repeated at $u + j/(n-p)$, where $1 \le j \le (n-p)-(2p-1)$. In this paper we refer to this operation as to rep:

$$rep : N_{i,p}(u) \to N_{i+j,p}(u + j/(n-p)), \tag{20}$$

$$\text{where } 1 \le j \le n-3p+1 \text{ for all } p-1 \le i < 2p. \tag{21}$$

However, u values must be distributed in specific manner, in order to hit a required $1-u$ or $u + j/(n-p)$. This means that the chosen step Δu must divide each non-zero knot interval into the same number of equal subintervals. If we consider that $\kappa \in \mathrm{N}$ is a natural number, then step Δu must satisfy:

$$\Delta u = \frac{1}{(n-p)\kappa}. \tag{22}$$

3.3.2 Multiple Point on Curve Evaluation Strategies

We suggest several NURBS evaluations strategies regarding given observations in Table 2. The strategy name corresponds to the case of the spline and to the basis function algorithm. The interval row indicates the bounds of the parametric space for basis clone operations that are given in the last table row (see expressions (18), (19), (20), and (21)).

Table 2. NURBS curve evaluation strategies

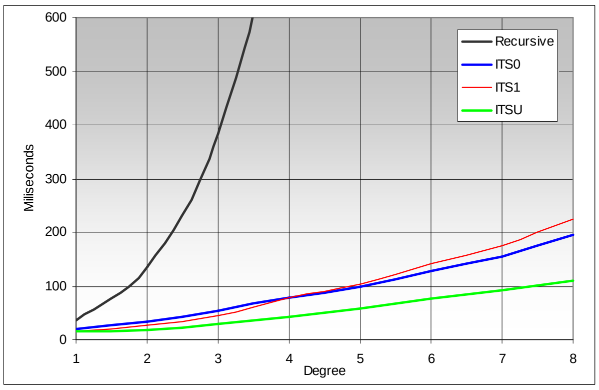
Strategy	Case	Interval	Basis method	Get point method	Basis clone operation
NURBS_ITS0	General	$0 \le u \le 1$	**Basis_ITS0**	**GetPoint0**	
NURBS_ITS1	General	$0 \le u \le 1$	**Basis_ITS1**	**GetPoint1**	
URBS_ITS1	Uniform	$0 \le u < 0.5$	**Basis_ITS1**	**GetPoint1**	$N_{n-i-1,p}(1-u) = ref(N_{i,p}(u))$
		$u = 0.5$	**Basis_ITS1**	**GetPoint1**	
URBS_ITS1+U	Uniform $n \ge 3p-1$	$0 \le u < u_{2p-1}$	**Basis_ITS1**	**GetPoint1**	$N_{n-i-1,p}(1-u) = ref(N_{i,p}(u))$
		$u_{2p-1} \le u < u_{2p}$	**Basis_ITSU**	**GetPoint1**	$N_{i+j,p}(u + j/(n-p)) = rep(N_{i,p}(u))$
UBS_ITS1+U	Uniform Non-rational $n \ge 3p-1$	$0 \le u < u_{2p-1}$	**Basis_ITS1**	**GetPoint_NR1**	$N_{n-i-1,p}(1-u) = ref(N_{i,p}(u))$
		$u_{2p-1} \le u < u_{2p}$	**Basis_ITSU**	**GetPoint_NR1**	$N_{i+j,p}(u + j/(n-p)) = rep(N_{i,p}(u))$

Note that in order to apply uniform case optimizations, the step size Δu must be set accordingly to the expression (22) before evaluation. An uniform non-rational spline is evaluated using the simplified method **GetPoint_NR1** instead of **GetPoint1** (expression (1) instead of (8)).

4 Results

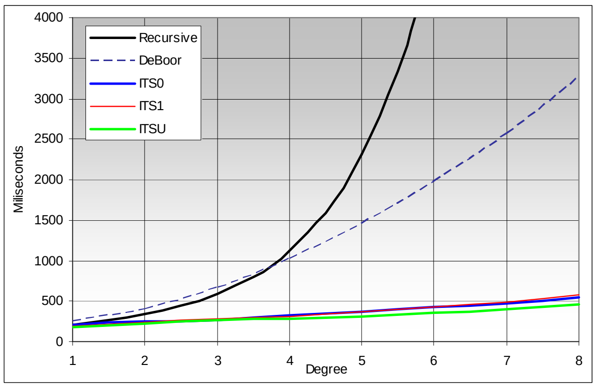
Algorithms given in Section 3 were implemented using C# programming language and .NET framework. The performance tests were acquired on Intel Core2 Duo 1.86 GHz x2 CPU, 3.GB RAM machine. The evaluation of single point or function takes only few nanoseconds. This makes the comparison of evaluation time-effectiveness hardly possible. Therefore all evaluation algorithms were applied 10^5 times at different u. This procedure was performed several times and average calculation times were recorded.

Recursive Cox-de Boor, **Basis_ITS0**, **Basis_ITS1**, and **Basis_ITSU** basis function evaluation algorithms were tested on the uniform 27 control point B-spline. The same algorithms and de Boor's knot insertion method were employed to determine a single point on the curve. Calculation times are given in Figure 7 and Figure 8 respectively.



n = 27				
p	Recursive	ITS0	ITS1	ITSU
1	37	20	15	15
2	135	34	26	19
3	385	55	46	30
4	940	79	78	42
5	2105	100	103	58
6	4806	129	142	77
7	10470	156	176	92
8	24751	195	224	111

Figure 7. NURBS basis function calculation times in milliseconds



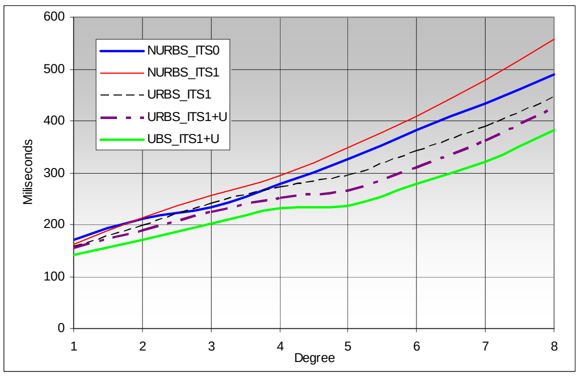
n = 27					
p	Recursive	DeBoor	ITS0	ITS1	ITSU
1	206	251	206	187	179
2	340	404	247	233	225
3	593	667	268	277	266
4	1128	1022	320	311	279
5	2311	1458	370	366	310
6	4927	1968	429	431	360
7	10976	2566	481	496	402
8	25855	3253	543	574	460

Figure 8. NURBS single point on curve evaluation times in milliseconds

Calculation time of the recursive Cox-de Boor algorithm grows rapidly for every successive degree of B-spline. However, the ITS is noticeably less affected by the degree increment. De Boor's knot insertion should be fast, because it has no basis function calculation phase. According to Figure 8, **GetPoint_DeBoor** overtakes recursive algorithm only when $p \geq 4$, but is left far behind by the ITS. This happens because of a large number of scalar multiplications and conversions from 3D to 4D and back.

The ITS takes less time in the basis function determination phase than in the position acquisition phase even when $p = 8$. Therefore performances of single point evaluation using **Basis_ITS0**, **Basis_ITS1** or **Basis_ITSU** are very similar. Due to poor performance of recursive Cox-de Boor and de Boor's knot insertion algorithms they were not included in multiple point evaluation. The evaluation of multiple points over entire 27 control point NURBS curve was carried out using strategies given in Section 3.3.2. Results are given in Figure 9.

The same strategies were applied to $n = 18$ control point and $n = 9$ control point curves. Performance patterns remain the same as in Figure 9, but **URBS_ITS1+U** and **UBS_ITS1+U** provided less time economy. In these cases, less control points mean fewer intervals where the operation *rep* can be applied (see expressions (20) and (21)).



n = 27					
p	NURBS_ ITS0	NURBS_ ITS1	URBS_ ITS1	URBS_ ITS1+U	UBS_ ITS1+U
1	170	162	157	156	142
2	211	213	197	189	170
3	234	256	240	225	203
4	279	295	271	252	231
5	326	348	295	265	237
6	382	409	342	310	278
7	433	479	389	361	322
8	491	557	444	424	381

Figure 9. NURBS multiple points on curve evaluation times in milliseconds

The implementation of *ref* and *rep* operations in evaluation of uniform rational B-spline saved from 3.7% to 24.6% of calculation time (compare **URBS_ITS1+U** and **NURBS_ITS1**). The algorithm designed for uniform non-rational B-spline saved from 12.4% to 32.8% of calculation time. Also, **URBS_ITS1+U** and **UBS_ITS1+U** were respectively up to 18.7% and 27.3% more time-efficient in comparison to **NURBS_ITS0**.

The evaluation of higher degree basis functions takes longer. In these cases *ref* and *rep* operations can save more time (compare **URBS_ITS1+U** and **NURBS_ITS1** in Figure 9). There is one more fact to be taken into consideration. The percentage of saved calculation time depends on the number of NURBS control points. Accordingly to the expression (21), there are $n-3p+1$ knot intervals where *rep* operation can be applied. If $n<3p-1$, this optimization can not be implemented even if B-spline is uniform.

5 Conclusions

In this paper we analyzed three already known NURBS evaluation algorithms. Test results showed that the recursive Cox-de Boor formula is highly ineffective especially in the evaluation of higher degree splines. Although de Boor's knot insertion method performed better while evaluating splines of the fourth and higher degree, it was significantly overtaken by inverted triangular scheme in all cases.

Due to this discovery we composed several modifications of the inverted triangular scheme and few evaluation strategies designed for special cases of NURBS. Accordingly to the test results, the presented strategies saved up to 24.6% of evaluation time in the case of uniform B-spline, and up to 32.8% in the case of uniform non-rational B-spline. A significant gain of performance was observed during NURBS evaluation of the degree $p>4$ with the number of control points greater or equal to $3p-1$.

The stated facts lead to a conclusion that time-efficiency of NURBS curve evaluation based on the inverted triangular scheme can be improved. This is achieved by recognizing uniform and non-rational cases and implementing evaluation strategies presented in this paper. Optimizations are especially effective in the evaluation of higher degree splines with a larger number of control points.

References

[1] **Andersson F., Berit K.** Bezier and B-spline Technology. Master of Science Thesis. 2003.

[2] **Deng J. J.** Theory of a B-Spline Basis Function. *International Journal of Computer Mathematics*. 2003, volume 80, issue 3, pages 649 - 664.

[3] **Farin G.** Curves and Surfaces for CAGD: A Practical Guide 5th Edition. *Morgan Kaufman Publishers*. 2002, pages 135 - 146.

[4] **Fisher J., Lowther J., Shene C. K.** If You Know B-Splines Well, You Also Know NURBS!. *ACM SIGCSE Bulletin*. 2004, volume 36, issue 1, pages 343 - 347.

[5] **Krishnamurthy A., Khardekar R., McMains S.** Direct Evaluation of NURBS Curves and Surfaces on the GPU. *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling*. 2007, pages 329 - 334.

[6] **Piegl L., Tiller W.** The NURBS Book 2nd Edition. *Springer*. 1999, pages 67 - 78.

[7] **Plukas K.** Skaitiniai metodai ir algoritmai, *Naujas Lankas*. 2001, pages 319 - 326.

[8] **Sánchez H., Moreno A., Oyarzun D., García-Alonso A.** Evaluation of NURBS surfaces: an overview based on runtime efficiency. *WSCG SHORT Communication Papers Proceedings, Science Press*. 2004, pages 235 - 242.

[9] **Shene C. K.** CS3621 Introduction to Computing with Geometry Notes. 1997 - 2008. Online access: http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/

[10] **Xie H., Qin H.** Automatic Knot Determination of NURBS for Interactive Geometric Design. *Proceedings of the International Conference on Shape Modeling & Applications*. 2001, pages 267 - 277.