Computer-Aided Design 73 (2016) 26–35

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Vectorizing NURBS surface evaluation with basis functions in power basis☆

Stepan Yu. Gatilov*

A. P. Ershov Institute of Informatics Systems, Acad. Lavrentjev pr. 6, Novosibirsk 630090, Russia
Ledas Ltd., Acad. Lavrentjev pr. 6, Novosibirsk 630090, Russia

ARTICLE INFO

Article history:
Received 6 January 2015
Accepted 26 October 2015

Keywords:
NURBS
Evaluation
Vectorization
SSE

ABSTRACT

Several known methods for direct evaluation of NURBS curves and surfaces are described. Runtime performance and simplicity of vectorization are discussed. An evaluation method, which uses basis functions precomputed in shifted power basis, is shown to be promising. This method for surfaces is vectorized with SSE2 intrinsics, yielding 3 times performance improvement over the non-vectorized version. Branchless vectorized linear search is proposed for span search, being most efficient for small number of knots. Binary search ending with a small linear search is shown to be most efficient for large number of knots, and good for general case. Performance comparison of the evaluation method and its equivalents from available geometric kernels is included.

1. Introduction

NURBS curves and surfaces are considered a standard for geometry representation in CAD models. Direct evaluation of NURBS is a basic routine that is required to work with NURBS.

Many papers are devoted to direct evaluation of NURBS curves and surfaces. Generally speaking, it is desirable to make NURBS evaluation as fast as possible without losing numerical stability. Papers [1,2] are dedicated to reducing amount of operations for evaluation. The latter one uses Horner-like evaluation of Bernstein polynomials. A comparison of NURBS evaluation in several representations was done in [3]. Nowadays, GPU computing is a modern trend. [4] is dedicated to NURBS evaluation completely on GPU.

Current paper contains a brief survey of several known NURBS evaluation methods. A major performance improvement is achieved by vectorizing NURBS surface evaluation. The related decisions and results are presented. It is worth noting that high-performance NURBS evaluation on CPU is very important in CAD because it can be used anywhere in a geometric kernel, unlike a GPU version which can be beneficial only for algorithms that use massive evaluations of NURBS (more than 10^4 points at once). Also, geometric kernels use double precision to cope with great stability problems, and most of current consumer video cards still do not have proper double precision hardware, resulting in 10–30 times difference in peak single precision and double precision performance.

The algorithms studied in the current paper were implemented and used during the development of Russian Geometric Kernel.[1]

The paper is organized as follows. Section 2 briefly summarizes the notations and assumptions used throughout the whole paper. Then a survey of several known techniques is given. An algorithm for point evaluation based on basis functions is described in Section 4. Cox–de Boor formula and power basis representation are compared, the shifted power basis form is introduced. In Section 5 ideas of alternative methods are presented, which evaluate NURBS directly in 3D space. Table 2 summarizes the outlined methods at the end of the section. Section 6 is dedicated to extending the basis functions approach for derivatives evaluation. The chosen method is described in Section 7. Most of that section is dedicated to vectorized implementation of NURBS surface direct evaluation. Linear search and binary search for getting span index are compared in Section 7.2. Finally, a comparison of NURBS evaluation methods is done in Section 8.

☆ This paper has been recommended for acceptance by George Allen.

* Correspondence to: A. P. Ershov Institute of Informatics Systems, Acad. Lavrentjev pr. 6, Novosibirsk 630090, Russia.
E-mail address: stgatilov@gmail.com.

[1] Russian Geometric Kernel is a geometric modeling kernel being developed in Russia according to the governmental program "National Technological Base": "Developing Russian Licensable Software — a Mathematical Kernel for 3D-Modeling as a Basis of Computer Systems for Computer-Aided Design of Complex Engineering Products".

http://dx.doi.org/10.1016/j.cad.2015.10.006
0010-4485/

2. Preliminaries

The following NURBS-related notations are used throughout the paper. Spline degree is denoted by d, and order is denoted by p (by definition $p = d + 1$). In case of surface subscripts u and v are sometimes added to distinguish between quantities for U and V parameters. The number of control points is denoted by n (by n_u and n_v for surfaces). The knots are denoted by k_i for $i = 0 \dots n+p-1$. The sequence of knots must be non-decreasing. NURBS must be clamped, i.e. multiplicities of the first and the last knots are equal to p. Multiplicities of all the other (internal) knots must not exceed d. A span is an interval between two subsequent knots. There are $n - d$ spans in total, though some of them may be degenerate if internal knots with multiplicity are present. Degree and knots sequence together define a family of basis functions $N_{i,d}(t)$ for $i = 0 \dots n - 1$. Control points and weights are denoted by P_i and w_i for curves ($i = 0 \dots n - 1$), and $P_{i,j}$ and $w_{i,j}$ for surfaces. These basis functions are used as coefficients to blend control points into a point on NURBS. For each span, there are p control points that may contribute to the point position (called control points of the span). A NURBS is called polynomial (or non-rational) if all the weights are units, and it is called rational otherwise. Both NURBS curves and surfaces are in three-dimensional space.

Most of NURBS present in CAD models have degrees not greater than three, though there are exceptions. Splines of degree one are usually present in extrusion and ruled surfaces. Quadric surfaces, tori, and surfaces of revolution yield degree-two NURBS. A lot of geometric shapes are approximated by cubic splines, so NURBS of degree three are often used too.

High-performance computing on modern x86 CPUs cannot be done without vectorization. Although AVX instruction set is widely used already, processors without it are still common. An older instruction set for double precision calculations is SSE2, which was introduced somewhere around 2001. Nowadays SSE2 is often a prerequisite for a lot of different software, including CAD software. The current paper contains a major section dedicated to SSE2 optimization (see Section 7.1).

For the sake of performance analysis, subtraction and addition are not distinguished, since they take the same amount of time and are usually performed on the same execution units. Also, paired multiplication and addition is often be counted as MAD operation. This reflects the fact that most of the modern processors have separate add and multiply execution units that can work simultaneously. Moreover, some architectures (for instance Intel Haswell) have support for fused multiply–add SIMD instructions. So the throughput of paired multiply and add together is usually equal to that of only multiply or only add operations. The other fact worth noting is that division operations are incredibly slow. For instance, [5, Table C-15] states that division is 35 times slower than both multiplication and addition. Time of addition or subtraction is denoted by T_{add}, multiplication by T_{mul}, division by T_{div}, and multiplication with addition by T_{mad}.

All the performance measurements were done on a single computer with Intel Core i7-3770 3.4 GHz processor. Despite the fact that the processor has 4 physical cores, only a single core was always used. Intel LINPACK benchmark was used to estimate its computational performance in double precision. The benchmark achieves 23.2 GFlops on a single core of the processor. Note that the benchmark is based on AVX instructions, and its peak performance on SSE2 instructions is likely to be lower. All the programs were compiled by msvc2010 (Visual Studio 2010) for 32-bit architecture, including all the binaries of all the geometric kernels used.

3. Span search

The evaluation of a NURBS curve starts with determination of a span which contains a given curve parameter. Let k_b be the last knot not exceeding value t of the parameter. The index $b \in [d \dots n-1]$ is a result of the span search procedure. In case of a surface it is necessary to perform two span searches: along U and V surface parameters.

Since the array of knots is sorted, the span can be found by binary search in logarithmic time. However, binary search necessarily consists of scalar instructions and branches. On the other hand, linear search can be implemented in vectorized way and without branches. Despite the bad asymptotic time, linear search is actually faster unless the number of knots is large. Detailed discussion on linear search will be given in Section 7.2.

4. Basis functions approach

One of the ways to compute NURBS is to evaluate all the required basis functions first. Then the convex combination of all the influencing control points is calculated, with coefficients equal to basis functions values.

4.1. De Boor formula

The recurrent definition of the basis functions is

$$\begin{cases} N_{i,d}(t) = \dfrac{t - k_i}{k_{i+d} - k_i} N_{i,d-1}(t) + \dfrac{k_{i+d+1} - t}{k_{i+d+1} - k_{i+1}} N_{i+1,d-1}(t) \\ N_{i,0}(t) = I(k_i \le t < k_{i+1}), \end{cases} \tag{1}$$

where $I(condition)$ stands for indicator function: it is equal to one when *condition* is true, and to zero otherwise. Also, this formula is known as Cox–de Boor recursion formula.

Since most of the basis functions are zero, inverted triangular scheme is used to compute only the required basis functions [6]. For each degree d only $i \in [b - d; b]$ can yield nonzero values of $N_{i,d}(t)$, where b is the index determined by span search. The overall number of nonzero values is $\frac{1}{2}d(d+1)$. An optimized algorithm for (1) is given in [7, Algorithm A2.2]. The amount of operations is

$$T_{dbf} \approx \frac{p^2}{2} \times (2T_{add} + 2T_{mul} + 1T_{div}).$$

For a cubic case, it is $18T_{add} + 12T_{mul} + 6T_{div}$.

Clearly, most of the computational time here is spent on division. One way to overcome this problem is to precompute $(k_{i+d} - k_i)^{-1}$ for all i and d. However, given the associated memory cost, it is more efficient to precompute other representations of basis functions instead. Another problem is the complex structure of computation which is hard to vectorize. The major advantages of using de Boor formula are: no precomputation required, numerical stability.

4.2. Power basis

Representation in power basis allows perhaps the fastest way to evaluate a polynomial. However, there is a major drawback: evaluation in power basis is unstable for high degree polynomials. More information can be found in [8].

All the nonzero basis functions can be stored as polynomials in power basis. More precisely, basis functions $N_{b-d,d}(t), \dots, N_{b,d}(t)$ are stored for each nondegenerate span $[k_b; k_{b+1})$. They require np^2 real numbers of storage and $O(np^3)$ time to precompute.

Unfortunately, storing basis functions in the form $\sum a_j t^j$ is not possible in practice. Computational error is unacceptable for cubic curves with many control points. This instability can be easily demonstrated on a cubic Bézier curve defined on a span [0.999; 1.000]. In such case the basis function for the last control point is $N_{3,3}(t) = (\frac{x-0.999}{0.001})^3$. Clearly, $N_{3,3}(1) = 1$ (i.e. the endpoint is located at the last control point). However, if we compute it in