S. Krajči (ed.): ITAT 2018 Proceedings, pp. 3–9
CEUR Workshop Proceedings Vol. 2203, ISSN 1613-0073, © 2018 Viliam Kačala, Lukáš Miňo, and Csaba Török

Speedup of Uniform Bicubic Spline Interpolation

Viliam Kačala, Lukáš Miňo, and Csaba Török

Institute of Computer Science, Faculty of Science, Pavol Jozef Šafárik University in Košice
Jesenná 5, 040 01 Košice, Slovakia
viliam.kacala@student.upjs.sk, lukas.mino@upjs.sk, csaba.torok@upjs.sk

Abstract: The goal of the paper is to introduce an efficient algorithm for computation of derivatives of bicubic spline surfaces over equispaced grids with C^2 class continuity. The algorithm is based on a recently proposed approach using a special approximation property between quartic and cubic polynomials. The proposed solution replaces the classical de Boor's systems of equations with systems of reduced size and simple remainder explicit formulas. We will show that the proposed new algorithm is numerically equivalent to de Boor's algorithm and the former is more than 50% faster.

1 Introduction and problem statement

Modeling of surfaces plays a key role in a wide variety of computer science fields such as graphics or CAD applications. The paper proposes an efficient computational algorithm for interpolating uniform bicubic spline surfaces of class C^2. The standard way to accomplish that is an almost 60 years old classic algorithm designed by Carl de Boor [2] that uses tridiagonal systems of equations. We will refer to it as the *full* algorithm.

Even though the evaluation of such systems is quite straightforward in general, running in linear time, it can be still viewed as slow especially in real time computing scenarios. For this reason we present a new *reduced* algorithm for uniform bicubic spline surfaces that, while being in the same complexity class, is faster and needs lower memory requirements.

The structure of the paper is as follows. The remaining part of this section is devoted to the problem statement and introducing terminology. Section 2 presents the new reduced algorithm along with proof of its equality to the full algorithm. The third section provides experimental measurements of actual time savings of the new approach along with some words about its efficient implementation. To be self contained, we provide the equations of de Boor's full algorithm for surfaces and the reduced algorithm for curves in Appendix, for easier comparison with ours.

Now let's jump into the problem statement. Consider a uniform grid

$$[x_0, x_1, x_2, \ldots, x_{I-1}] \times [y_0, y_1, y_2, \ldots, y_{J-1}], \qquad (1)$$

where

$$x_i = x_0 + ih_x, \quad i = 1,2,3,\ldots,I-1, \quad I > 1, h_x \in \mathbb{R}^+,$$
$$y_j = y_0 + ih_y, \quad j = 1,2,3,\ldots,J-1, \quad J > 1, h_y \in \mathbb{R}^+.$$

According to [2], a spline surface is defined by given values

$$z_{i,j}, \quad i = 0,1,2,\ldots,I-1, \quad j = 0,1,2,\ldots,J-1 \qquad (2)$$

at the grid-points, and given first directional derivatives

$$d^x_{i,j}, \quad i = 0, I-1, \quad j = 0,1,2,\ldots,J-1 \qquad (3)$$

at the boundary verticals,

$$d^y_{i,j}, \quad i = 0,1,2,\ldots,I-1, \quad j = 0, J-1 \qquad (4)$$

at the boundary horizontals and cross derivatives

$$d^{x,y}_{i,j}, \quad i = 0, I-1, \quad j = 0, J-1 \qquad (5)$$

at the four corners of the grid.

The task is to define a quadruple $[z_{i,j}, d^x_{i,j}, d^y_{i,j}, d^{x,y}_{i,j}]$ at every grid-point $[x_i, y_j]$, based on which a bicubic clamped spline surface S of class C^2 is constructed with properties

$$S(x_i,y_j) = z_{i,j}, \qquad \frac{\partial S(x_i,y_j)}{\partial y} = d^y_{i,j},$$
$$\frac{\partial S(x_i,y_j)}{\partial x} = d^x_{i,j}, \qquad \frac{\partial^2 S(x_i,y_j)}{\partial x \partial y} = d^{x,y}_{i,j}.$$

For $I = 7$ and $J = 5$, we provide visual illustration of the input situation in Figure 1, where bold values represent (2) – (5) while the remaining non-bold values represent the unknown derivatives to compute.

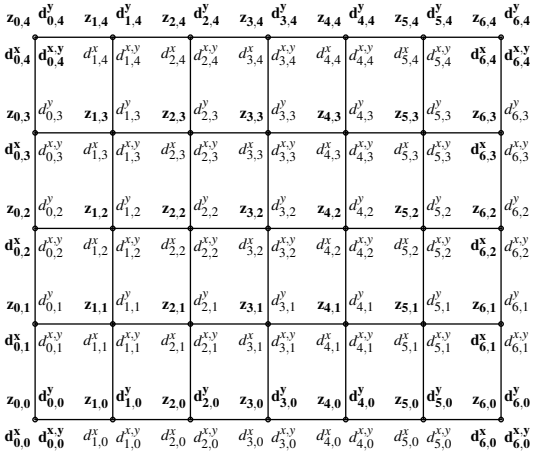

Figure 1: Input situation for a 7×5 uniform grid.

2 Reduced algorithm

The reduced algorithm for uniform cubic spline curves was proposed in [11], where the model equations were obtained using a special approximation property between cubic splines and quartic polynomials [10]. This property was extended for spline surfaces proving that a biquartic polynomial fully determines a 2×2-component uniform bicubic spline surface of class C^2 [9], [5]. This approach was generalized in [3], [6], [4], where a new algorithm for solving the unknown derivatives was proposed and new model equations and explicit formulas were derived. The proposed algorithm introduced equation systems with reduced dimensionality while the remaining unknown derivatives were computed from simple explicit formulas. Measured results showed, that the proposed algorithm is faster than the full one [4]. Thorough analysis has shown that we can further increase the performance of the reduced approach. In the next subsection every model equation will be expressed by simple expressions that correspond to the model equations of [11].

This section consists of two parts. Firstly we show that one of the four systems of de Boor's algorithm can be equivalently solved by a reduced system and additional explicit formulas. In the second part a reduced algorithm is proposed for the solution of the unknown derivatives of a spline surface of class C^2 that is equivalent to de Boor's full algorithm.

2.1 Model equations

Consider the uniform grid (1) with given input values (2) – (5). Any of the remaining unknown values can be computed using the systems of equations (20) – (23) of the full algorithm, see Appendix.

Consider a system of linear equations for the j-th horizontal ($j=0,1,2,\ldots,J-1$), see (20) in Appendix, that takes a form

$$\begin{bmatrix} 4 & 1 & 0 & & & \\ 1 & 4 & 1 & & & \\ 0 & 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & 0 & 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} d^x_{1,j} \\ d^x_{2,j} \\ d^x_{3,j} \\ \vdots \\ d^x_{I-3,j} \\ d^x_{I-2,j} \end{bmatrix} = \begin{bmatrix} \frac{3}{h_x}(z_{2,j}-z_{0,j}) - d^x_{0,j} \\ \frac{3}{h_x}(z_{3,j}-z_{1,j}) \\ \frac{3}{h_x}(z_{4,j}-z_{2,j}) \\ \vdots \\ \frac{3}{h_x}(z_{I-2,j}-z_{I-4,j}) \\ \frac{3}{h_x}(z_{I-1,j}-z_{I-3,j}) - d^x_{I-1,j} \end{bmatrix}. \tag{6}$$

The following lemma shows that any of the systems (6) can be solved by a reduced system.

Lemma 1. *Let $h_x \in \mathbb{R}^+$, $I \geq 4$, $\{z_{0,j}, z_{1,j}, z_{2,j}, \ldots z_{I-1,j}, d^x_{0,j}, d^x_{I-1,j}\} \subset \mathbb{R}$ be some known values and $\{d^x_{1,j}, d^x_{2,j}, d^x_{3,j}, \ldots, d^x_{I-2,j}\} \subset \mathbb{R}$ be unknown values. Then the full equation system (6) and the reduced one*

$$\begin{bmatrix} -14 & 1 & 0 & & & \\ 1 & -14 & 1 & & & \\ 0 & 1 & -14 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -14 & 1 \\ & & & 0 & 1 & \mu \end{bmatrix} \cdot \begin{bmatrix} d^x_{2,j} \\ d^x_{4,j} \\ d^x_{6,j} \\ \vdots \\ d^x_{\nu-2,j} \\ d^x_{\nu,j} \end{bmatrix} = \begin{bmatrix} \frac{3}{h_x}(z_{4,j}-z_{0,j}) - \frac{3}{h_x}(z_{3,j}-z_{1,j}) - d^x_{0,j} \\ \frac{3}{h_x}(z_{6,j}-z_{2,j}) - \frac{3}{h_x}(z_{5,j}-z_{3,j}) \\ \frac{3}{h_x}(z_{8,j}-z_{4,j}) - \frac{3}{h_x}(z_{7,j}-z_{5,j}) \\ \vdots \\ \frac{3}{h_x}(z_{\nu,j}-z_{\nu-4,j}) - \frac{12}{h_x}(z_{\nu-1,j}-z_{\nu-3,j}) \\ \frac{3}{h_x}(z_{\nu+\tau,j}-z_{\nu-2,j}) - \frac{12}{h_x}(z_{\nu+1,j}-z_{\nu-1,j}) - \eta d_{I-1,j} \end{bmatrix}, \tag{7}$$

where

$$\begin{aligned} &\mu=-14,\ \tau=2,\ \eta=1,\ \nu=I-3, && \text{if } I \text{ is odd},\\ &\mu=-15,\ \tau=0,\ \eta=-4,\ \nu=I-2, && \text{if } I \text{ is even}, \end{aligned} \tag{8}$$

with explicit formulas

$$\begin{aligned} &d^x_{i,j} = \frac{3}{4h_x}(z_{i+1,j}-z_{i-1,j}) - \frac{1}{4}(d^x_{i+1,j}+d^x_{i-1,j}),\\ &i=1,3,5,\ldots,\nu+\tau-1, \end{aligned} \tag{9}$$

are equivalent.

Proof. It follows from Lemma 3 of Appendix that the solution of the reduced algorithm is equivalent to the solution of the full system, see e.g. [8],

$$\begin{bmatrix} 4 & 1 & 0 & & & \\ 1 & 4 & 1 & & & \\ 0 & 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & 0 & 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{N-3} \\ d_{N-2} \end{bmatrix} = \begin{bmatrix} \frac{3}{h}(y_2-y_0) - d_0 \\ \frac{3}{h}(y_3-y_1) \\ \frac{3}{h}(y_4-y_2) \\ \vdots \\ \frac{3}{h}(y_{N-2}-y_{N-4}) \\ \frac{3}{h}(y_{N-1}-y_{N-3}) - d_{N-1} \end{bmatrix}. \tag{10}$$

As we see the systems (6) and (10) use the same tridiagonal matrix. The systems (10) and (6) differ in the unknowns and the right-hand side vectors. However, using a substitution

$$h=h_x,\ d_i=d^x_{i,j},\ y_i=z_{i,j},\ i=0,1,2,\ldots,N-1,\ N=I, \tag{11}$$

we get (6) from (10), (7) from (24) and the remainder formulas (9) from (26) that alongside with Lemma 3 finishes the proof. □

2.2 Algorithm

Now we have sufficient theoretical foundations to propose an efficient algorithm to compute spline derivatives over a uniform grid of points. The reduced algorithm consists of four main steps, each evaluating systems of equations derived from (7) and remainder formulas (9). The algorithm as a whole takes the same input values and provides identical results as the original full algorithm in Appendix. Therefore it is intended as a drop-in replacement for the full algorithm.

Based on Lemma 1 the full systems (21) – (23) can be solved using analogical reduced systems and explicit formulas. Hence we get immediately the reduced algorithm for solving the unknown derivatives of a C^2 spline surface that appears to provide faster computations than de Boor's algorithm.

Theorem 1 (Reduced algorithm for surfaces)**.** *If the values (2) – (5) over the uniform grid (1) are given, then the unknown values*

$$\begin{aligned}
&d^x_{i,j}, && i=1,2,3,\ldots,I-2, && j=0,1,2,\ldots,J-1,\\
&d^y_{i,j}, && i=0,1,2,\ldots,I-1, && j=1,2,3,\ldots,J-2,\\
&d^{x,y}_{i,j}, && i=1,2,3,\ldots,I-2, && j=0,J-1,\\
&d^{x,y}_{i,j}, && i=0,1,2,\ldots,I-1, && j=1,2,3,\ldots,J-2
\end{aligned}$$

for spline surface of class C^2 are uniquely determined by the following linear systems and formulas in four main steps:

Step 1a. Computation of d^x along the horizontals from equation systems for inner even-indexed grid-points.
For each $j=0,1,2,\ldots,J-1$,

$$\begin{aligned}
&solve_system(\\
&\quad d^x_{i+2,j}-14d^x_{i,j}+d^x_{i-2,j}=\\
&\qquad=\frac{3}{h_x}(z_{i+2,j}-z_{i-2,j})-\frac{12}{h_x}(z_{i+1,j}-z_{i-1,j}), \quad (12)\\
&\quad where\ i=2,4,6,\ldots,I-3\\
&).
\end{aligned}$$

Step 1b. Computation of d^x along the horizontals from explicit formulas for inner odd-indexed grid-points.
For each $i=1,3,5,\ldots,I-2$ and $j=0,1,2,\ldots,J-1$,

$$d^x_{i,j}=\frac{3}{4h_x}(z_{i+1,j}-z_{i-1,j})-\frac{1}{4}(d^x_{i+1,j}+d^x_{i-1,j}). \quad (13)$$

Step 2a. Computation of d^y along the verticals from equation systems for inner even-indexed grid-points.
For each $i=0,1,2,\ldots,I-1$,

$$\begin{aligned}
&solve_system(\\
&\quad d^y_{i,j+2}-14d^y_{i,j}+d^y_{i,j-2}=\\
&\qquad=\frac{3}{h_y}(z_{i,j+2}-z_{i,j-2})-\frac{12}{h_y}(z_{i,j+1}-z_{i,j-1}), \quad (14)\\
&\quad where\ j=2,4,6,\ldots,J-3\\
&).
\end{aligned}$$

Step 2b. Computation of d^y along the verticals from explicit formulas for inner odd-indexed grid-points.
For each $i=0,1,2,\ldots,I-1$ and $j=1,3,5,\ldots,J-2$,

$$d^y_{i,j}=\frac{3}{4h_y}(z_{i,j+1}-z_{i,j-1})-\frac{1}{4}(d^y_{i,j+1}+d^y_{i,j-1}). \quad (15)$$

Step 3a. Computation of $d^{x,y}$ along the horizontals from equation systems for inner even-indexed grid-points.
For each $j=0,J-1$,

$$\begin{aligned}
&solve_system(\\
&\quad d^{x,y}_{i+2,j}-14d^{x,y}_{i,j}+d^{x,y}_{i-2,j}=\\
&\qquad=\frac{3}{h_x}(d^x_{i+2,j}-d^x_{i-2,j})-\frac{12}{h_x}(d^x_{i+1,j}-d^x_{i-1,j}), \quad (16)\\
&\quad where\ i=2,4,6,\ldots,I-3\\
&).
\end{aligned}$$

Step 3b. Computation of $d^{x,y}$ along the horizontals from explicit formulas for inner odd-indexed grid-points.
For each $i=1,3,5,\ldots,I-2$ and $j=0,J-1$,

$$d^{x,y}_{i,j}=\frac{3}{4h_x}(d^x_{i+1,j}-d^x_{i-1,j})-\frac{1}{4}(d^{x,y}_{i+1,j}+d^{x,y}_{i-1,j}). \quad (17)$$

Step 4a. Computation of $d^{x,y}$ along the verticals from equation systems for inner even-indexed grid-points.
For each $i=0,1,2,\ldots,I-1$,

$$\begin{aligned}
&solve_system(\\
&\quad d^{x,y}_{i,j+2}-14d^{x,y}_{i,j}+d^{x,y}_{i,j-2}=\\
&\qquad=\frac{3}{h_y}(d^y_{i,j+2}-d^y_{i,j-2})-\frac{12}{h_y}(d^y_{i,j+1}-d^y_{i,j-1}), \quad (18)\\
&\quad where\ j=2,4,6,\ldots,J-3\\
&).
\end{aligned}$$

Step 4b. Computation of $d^{x,y}$ along the verticals from explicit formulas for inner odd-indexed grid-points.
for each $i=0,1,2,\ldots,I-1$ and $i=1,3,5,\ldots,I-2$,

$$d^{x,y}_{i,j}=\frac{3}{4h_y}(d^y_{i,j+1}-d^y_{i,j-1})-\frac{1}{4}(d^{x,y}_{i,j+1}+d^{x,y}_{i,j-1}). \quad (19)$$

If I is even, then the last model equation in steps (12) and (16) needs to be accordingly replaced by the model equation derived according to (8). Analogically, if J is even, the same applies to steps (14) and (18).

We underline that the systems (12), (14), (16) and (18) are diagonally dominant and use the same tridiagonal matrix with constant main diagonal and upper, lower diagonals.

In the next section it will be shown that the reduced algorithm is faster than the full one and additionally it needs lower memory requirements.

The main difference between original full algorithm and the reduced one is that the latter computes only half of the unknown values using LU factorization where the remaining half of the unknowns are solved by simple remainder formulas. Because of this, the reduced algorithm is supposed to be more efficient in terms of execution speed and memory requirements as well. We will discuss and measure it's speed in Section 3.

3 Performance evaluation

To be worth of actual implementation, the reduced algorithm should be significantly faster than full one. The elementary task in both algorithms is computation of tridiagonal systems of equations. As the full and the reduced systems are diagonally dominant we use a modified LU factorization method as the internal solver for such systems [4]. Our optimizations of the LU factorization relies on a fact, that the left-hand side matrices comprise only constant values.

While the LU factorization for tridiagonal linear systems is a quite efficient way to solve the equations which are interdependent, it has some drawbacks in the sense of modern CPU's capabilities like usage of vectorized SIMD instructions especially due to the structure of tridiagonal matrices. By breaking down the computational task of the full approach into reduced systems and simple explicit formulas, half of the equations are independent and therefore more versatile to manual or automatic compiler optimizations.

Memory requirements For the sake of completeness some words about applied data structures and memory requirements should be given.

The input grid (1) of size $I \times J$ needs $I+J$ memory to store x and y coordinates of the total IJ grid-points. Each grid-point however requires $4IJ$ space for $z_{i,j}$, $d^x_{i,j}$, $d^y_{i,j}$, $d^{x,y}_{i,j}$ values, where most of them will be computed by the full or reduced algorithm. This gives us $4IJ+I+J$ space requirement to store input/output values.

The needs of the full and reduced algorithms are quite low regarding the size of the grid. The full tridiagonal systems in Lemma 2 require $2 \cdot max(I,J)$ space to store the right-hand side vector and an auxiliary buffer vector used for the LU factorization. In case of the reduced algorithm, only half number of the equations form the tridiagonal system, therefore they require only $max(I,J)$ space for the LU part.

Data structures Since the grid may contain tens of thousands or more grid-points, the most effective representation is the jagged array structure for each of the z, d^x, d^y, $d^{x,y}$ values. Each equation system from any of the two algorithms always depends on one row of a jagged array, therefore entire rows of the jagged structure can be effectively cached under the assumption, that the size of the row is not very large. Notice that the iterations for computing the $d^y_{i,j}$ and most of the $d^{x,y}_{i,j}$ values in both algorithms have interchanged indices compared to the iteration throughout the $d^x_{i,j}$ values. We mention that an efficient implementation needs to setup the jagged arrays in accordance with how we want to iterate the data [7].

3.1 Measurements

Now let us compare the implementations of both algorithms. We implemented a benchmark in C++ 17 compiled with MSVC 2017 using *-O2* optimization level and individual code generation for each tested CPU, i.e. CPUs with AVX2 support were running binaries compiled to AVX2 instruction set whereas older CPUs received for instance only SSE2 compiled binaries. Testing environments comprised several computers with various CPUs ranging from rather obsolete Nehalem to recent Skylake microarchitecture. All systems had 8 – 32 GB of RAM, SSD and Windows 10 installed. The tests were conducted on freshly booted PCs after 2 minutes of idle time without running any non-essential services or processes like web browsers, database engines, etc.

On all testing computers the tests were conducted on two datasets, a small one on a grid of size $I,J = 100$, and a large one, where grid dimensions were $I,J = 1000$. Both datasets comprised the grid $[x_0,x_1,\ldots,x_{I-1}] \times [y_0,y_1,\ldots,y_{J-1}]$, where $x_0 = -20$, $x_{I-1} = 20$, $y_0 = -20$, $y_{J-1} = 20$ and the values $z_{i,j}$, $d^x_{i,j}$, $d^y_{i,j}$, $d^{x,y}_{i,j}$, see (2) – (5), were computed from function $sin\sqrt{x^2+y^2}$ at each grid-point. These datasets were chosen arbitrarily and the behaviour of the algorithms was correct for any input values. The speedup values were gained averaging 50 measurements of each algorithm.

Tables 1 and 2 contain results for both datasets and consist of four columns. The first column contains the tested CPUs ordered by their release date. Columns two through four contain measured execution times in microseconds for both algorithms and their speed ratios.

To provide better comparison for more datasets, Table 3 provides measurements on more different input sizes, however for the sake of readability it contains measurements only from Intel i7 6700K as the fastest testing CPU.

Let us review the measured performance improvement of the reduced algorithm. Results of Table 1 say that the reduced algorithm is more than fifty percent faster than the full one as long as the grid dimensions remains relatively small to enable the entire rows of the grid as well as the auxiliary LU vectors to be cached.

	$I,J = 100$		
CPU	Full	Reduced	Speedup
Intel i5 M430	783	438	1.79
AMD A6 3650M	922	567	1.64
Intel i7 4790	482	250	1.93
Intel i7 6700K	355	190	1.86
AMD X4 845	648	347	1.86

Table 1: Comparison of full and reduced algorithms on small dataset. Times are in microseconds.

	$I,J = 1000$		
CPU	Full	Reduced	Speedup
Intel i5 M430	116 856	90 541	1.29
AMD A6 3650M	116 390	78 892	1.48
Intel i7 4790	54 719	33 584	1.63
Intel i7 6700K	37 029	22 571	1.64
AMD X4 845	78 860	48 955	1.61

Table 2: Comparison of full and reduced algorithms on large dataset. Times are in microseconds.

The speedup of the reduced algorithm in Table 2 with a larger dataset is not so high due to difficulties of keeping data in fast but small L1 cache, resulting in higher ratio of cache misses. In the case of even larger dataset, let's say in the order of billions of grid-points, the performances of both algorithms are similar with the reduced one gaining only small advantage.

In our experiments we observed that the reduced algorithm was always faster than the full one, however exact speedup depends on the size of the input grid. The general rule is: the larger the grid the smaller the speedup.

The highest speedup does not depend on the grid size if it is from range of 50 to 200.

CPU	Full	Reduced	Speedup
$I,J = 50$	92	50	1.84
$I,J = 100$	355	190	1.86
$I,J = 200$	1 417	778	1.82
$I,J = 300$	3 203	1 797	1.78
$I,J = 400$	5 791	3 234	1.79
$I,J = 1000$	37 029	22 571	1.64
$I,J = 1500$	88 236	54 141	1.63
$I,J = 2000$	178 144	115 674	1.54

Table 3: Multiple dataset comparison of full and reduced algorithms tested on i7 6700K. Times are in microseconds.

4 Discussion

Let us briefly discuss the new algorithm from the numerical and experimental point of view. While the classic full algorithm is composed of four series of tridiagonal systems of equations, the reduced algorithm breaks down each equation system into a reduced one approximately half the size of the original one and simple mutually independent explicit formulas.

In addition, from the numerical point of view, the reduced tridiagonal subsystems are diagonally dominant and therefore computationally stable [1], similarly to the full systems. The remainder explicit formulas (9) are simple and thus do not present an issue.

The maximal numerical difference in our C++ implementation was in the order of 10^{-16} on several different datasets so the reduced algorithm yields numerically accurate results.

Since the algorithm consists of many independent systems of linear equations, it can be also effectively parallelized for both CPU and GPU architectures.

There is also a future work for further reduction of the number of equations in the tridiagonal systems.

5 Conclusion

The paper introduced a new algorithm to compute the unknown derivatives for uniform bicubic spline surfaces of class C^2. The algorithm reduces the size of the tridiagonal systems of equations by half and computes the remaining unknown derivatives using simple explicit formulas. A substantial decrease of execution time of derivatives at grid-points has been achieved with lower memory space requirements at the cost of a slightly more complex implementation where the measured speedup ranges from 1.3 to 1.9 depending on the grid size and CPU architecture.

Acknowledgements

This work was partially supported by projects Technicom ITMS 26220220182 and APVV-15-0091 Effective algorithms, automata and data structures.

Appendix

Carl de Boor in [2] proposed an algorithm for computation of the unknown derivatives of spline surface of class C^2 over any grid with four types of full systems of linear equations. The following lemma reformulates this algorithm for uniform grid.

Lemma 2 (Full algorithm for surfaces)**.** *If the values (2) – (5) over the uniform grid (1) are given, then the unknown*

values

$$\begin{aligned} &d^x_{i,j}, && i=1,2,3,\ldots,I-2, && j=0,1,2,\ldots,J-1,\\ &d^y_{i,j}, && i=0,1,2,\ldots,I-1, && j=1,2,3,\ldots,J-2,\\ &d^{x,y}_{i,j}, && i=1,2,3,\ldots,I-2, && j=0,J-1,\\ &d^{x,y}_{i,j}, && i=0,1,2,\ldots,I-1, && j=1,2,3,\ldots,J-2 \end{aligned}$$

for spline surface of class C^2 are uniquely determined by the following linear systems of equations:

Step 1. Computation of d^x along the horizontals from equation systems.

For each $j=0,1,2,\ldots,J-1$,

solve_system(

$$d^x_{i+1,j}+4\cdot d^x_{i,j}+d^x_{i-1,j}=\frac{3}{h_x}\cdot(z_{i+1,j}-z_{i-1,j}), \quad \text{where } i=1,2,3,\ldots,I-2 \tag{20}$$

).

Step 2. Computation of d^y along the verticals from equation systems.

For each $i=0,1,2,\ldots,I-1$,

solve_system(

$$d^y_{i,j+1}+4\cdot d^y_{i,j}+d^x_{i,j-1}=\frac{3}{h_y}\cdot(z_{i,j+1}-z_{i,j-1}), \quad \text{where } j=1,2,3,\ldots,J-2 \tag{21}$$

).

Step 3. Computation of $d^{x,y}$ along the horizontals from equation systems.

For each $j=0,J-1$,

solve_system(

$$d^{x,y}_{i+1,j}+4\cdot d^{x,y}_{i,j}+d^{x,y}_{i-1,j}=\frac{3}{h_x}\cdot(d^x_{i+1,j}-d^x_{i-1,j}), \quad \text{where } i=1,2,3,\ldots,I-2 \tag{22}$$

).

Step 4. Computation of $d^{x,y}$ along the verticals from equation systems.

For each $i=0,1,2,\ldots,I-1$,

solve_system(

$$d^{z,y}_{i,j+1}+4\cdot d^{z,y}_{i,j}+d^{z,y}_{i,j-1}=\frac{3}{h_y}\cdot(d^y_{i,j+1}-d^y_{i,j-1}), \quad \text{where } j=1,2,3,\ldots,J-2 \tag{23}$$

).

A new algorithm was proposed in [11] for computation of the unknown derivatives of a spline curve of class C^2 over uniform grids with a reduced system of linear equations instead of the well known full algorithm with parameters 1,4,1.

Lemma 3 (Reduced algorithm for curves)**.** *Consider a cubic clamped spline of class C^2 over a uniform grid $[u_0,u_1,\ldots,u_{N-1}]$, where $u_i=u_0+ih$, $i=1,2,3,\ldots,N-1$, defined by given values $y_0,\ldots,y_{N-1}$ and d_0, d_{N-1} at grid-points. The tridiagonal system*

$$\begin{bmatrix} -14 & 1 & 0 & & & & \\ 1 & -14 & 1 & & & & \\ 0 & 1 & -14 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -14 & 1 & \\ & & & 0 & 1 & -14 \end{bmatrix} \cdot \begin{bmatrix} d_2 \\ d_4 \\ d_6 \\ \vdots \\ d_{\nu-2} \\ d_\nu \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{3}{h}(y_4-y_0)-\frac{12}{h}(y_3-y_1)-d_0 \\ \frac{3}{h}(y_6-y_2)-\frac{12}{h}(y_5-y_3) \\ \frac{3}{h}(y_8-y_4)-\frac{12}{h}(y_7-y_5) \\ \vdots \\ \frac{3}{h}(y_\nu-y_{\nu-4})-\frac{12}{h}(y_{\nu-1}-y_{\nu-3}) \\ \frac{3}{h}(y_{\nu+\tau}-y_{\nu-2})-\frac{12}{h}(y_{\nu+1}-y_{\nu-1})-\eta d_{N-1} \end{bmatrix}, \tag{24}$$

where

$$\begin{aligned} &\mu=-14,\ \tau=2,\ \eta=1,\ \nu=N-3, && \text{if } N \text{ is odd},\\ &\mu=-15,\ \tau=0,\ \eta=-4,\ \nu=N-2, && \text{if } N \text{ is even}, \end{aligned} \tag{25}$$

and the formula

$$\begin{aligned} &d_i=\frac{3}{4h}(y_{i+1}-y_{i-1})-\frac{1}{4}(d_{i+1}+d_{i-1}),\\ &i=1,3,5,\ldots,\nu+\tau-1, \end{aligned} \tag{26}$$

imply that the second derivatives of spline components at the inner grid-points are equal.

References

[1] Björck A: Numerical Methods in Matrix Computations. Springer (2015)

[2] de Boor, C.: Bicubic Spline Interpolation. Journal of Mathematics and Physics, 41(3), (1962), 212–218

[3] Miňo L.: Efficient Computational Algorithm for Spline Surfaces. ITAT 2015: Information Technologies – Applications and Theory, ISSN 1613–0073, (2015), 30–37

[4] Kačala, V., Miňo, L.: Speeding up the Computation of Uniform Bicubic Spline Surfaces. WSCG 2017: 25. Conference on Computer Graphics, Visualization and Computer Vision 2017, ISSN 2464–4617, (2017), 73–80

[5] Miňo L., Szabó I., Török Cs.: Bicubic Splines and Biquartic Polynomials. Open Computer Science, Volume 6, Issue 1, ISSN (Online) 2299–1093, (2016), 1–7

[6] Miňo L., Török Cs.: Fast Algorithm for Spline Surfaces. Communication of the Joint Institute for Nuclear Research, Dubna, Russian Federation, E11-2015-77, (2015), 1–19

[7] Patterson, J. R. C.: Modern Microprocessors – A 90-Minute Guide!, Lighterra (2015)

[8] Salomon, D.: Curves and Surfaces for Computer Graphics. Springer (2006)

[9] Szabó I.: Approximation Algorithms for 3D Data Analysis. PhD Thesis, P. J. Šafárik University in Košice, Slovakia (2016)

[10] Török Cs.: On reduction of equations' number for cubic splines. Matematicheskoe modelirovanie, Vol. 26 , No. 11, ISSN 0234–0879, (2014), 33–36

[11] Török, Cs.: Speed-up of Interpolating Spline Construction, (to appear)