Accurate, Validated and Fast Evaluation of Bézier Tensor Product Surfaces[*]

H. Jiang[†] H. S. Li, L. Z. Cheng [‡]
School of Science and The State Key Laboratory for High Performance Computation, National University of Defense Technology, China
jhnudt@yahoo.cn

R. Barrio[§]
Dpto. de Matemática Aplicada and IUMA, Universidad de Zaragoza, E-50009 Zaragoza, Spain
rbarrio@unizar.es

C. B. Hu [¶]
College of Electronic Science and Engineering, National University of Defense Technology, China

X. K. Liao
School of Computer, National University of Defense Technology, China

Abstract

This paper proposes a compensated algorithm to evaluate Bézier tensor product surfaces with floating-point coefficients and coordinates. This algorithm is based on the application of error-free transformations to improve the traditional de Casteljau tensor product algorithm. This compensated algorithm extends the compensated de Casteljau algorithm for the evaluation of a Bézier curve to the case of tensor product surfaces. Forward error analysis and numerical experiments illustrate the accuracy and efficiency of the proposed algorithm.

Keywords: Bézier tensor product surfaces, Bernstein polynomial, Compensated algorithm, Error-free transformation, Round-off error, Floating point arithmetic.
AMS subject classifications: 65G99, 65-04, 15-04, 65D18

[*]Submitted: September 7, 2012; Accepted: May 9, 2013; Posted: May 25, 2013.

[†]H, Jiang was supported by China scholarship council 2011611057.

[‡]L. Z. Cheng was supported by Science Research Project of National University of Defense Technology JC 12-02-01 and the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant No.60626003)

[§]R. Barrio was supported by the Spanish Research project MTM 2009-10767.

[¶]C.B. Hu was supported by China scholarship council 2011611070.

1 Introduction

In Computer Aided Geometric Design (CAGD) tensor product polynomial surfaces [5] are usually expressed in the following Bézier form

$$F(x,y)=\sum_{i=0}^{m}\sum_{j=0}^{n}b_{i,j}B_i^m(x)B_i^n(y),\quad (x,y)\in[0,1]\times[0,1], \tag{1}$$

where $B_i^k(t)$ is the Bernstein polynomial of degree k given by

$$B_i^k(t)=\binom{k}{i}(1-t)^{k-i}t^i,\quad t\in[0,1]\quad i=0,1,\cdots,k, \tag{2}$$

and the points $b_{i,j}$ are called control points of the surface F. The corresponding surface is called Bézier tensor product surface (see [5]).

Since the most widely used algorithm for the evaluation of a Bézier curve is the de Casteljau algorithm (for brevity we denote it by `DC` algorithm in this paper), the algorithms to evaluate Bézier tensor product surfaces are often direct extensions of this univariate algorithm. They most well known are: the standard bilinear algorithm using conventional repeated bilinear interpolation (see [5]), an algorithm evaluating both the value and the derivatives simultaneously (see [22]) and the de Casteljau tensor product algorithm as a corner cutting algorithm running the `DC` algorithm successively in each parametric direction(see [4]). These algorithms are coherent in essence.

In this paper we only focus on the third one, the de Casteljau tensor product algorithm, for simplicity we denote it by `DCTP` algorithm. This algorithm has better stability and higher accuracy than corresponding Horner algorithm in bivariate case [4].The relative accuracy bound of the computed value $\widehat{F}(x,y)$ verifies the following inequality

$$\frac{|F(x,y)-\widehat{F}(x,y)|}{|F(x,y)|}\le \operatorname{cond}(F,x,y)\times\mathcal{O}(u), \tag{3}$$

where, u is the unit roundoff and $\operatorname{cond}(F,x,y)$ is the condition number of $F(x,y)$ (the expression will be given further). This algorithm is stable, however, when performed in floating point arithmetic, the computed result by the `DCTP` algorithm may be still less exact than expected owing to cancelations in ill-conditioned cases, then a high accurate algorithm is required.

Recently, a compensated Horner algorithm to evaluate the univariate polynomial in monomial basis has been proposed by Graillat, Langlois and Louvet in [9, 8, 16] and used to accurately solve a simple zero of a polynomial in [6]. Graillat also proposed the compensated algorithms for accurate floating-point product and exponentiation in [7]. Motivated by their work, we presented the compensated de Casteljau algorithms to evaluate the univariate polynomial and its first order derivative in Bernstein form in [14], bivariate polynomial in Bernstein-Bézier form in [13], and the compensated Clenshaw algorithm for the evaluation of Chebyshev series in [12]. All algorithms above can yield a full precision accuracy for not too ill-conditioned polynomial. The core technology is to apply error-free transformations which is exhaustively studied by Rump, Ogita and Oishi [24, 25, 23]. In this paper we extend the method proposed in [14] into the case of Bézier tensor product surfaces and present a compensated `DCTP` algorithm. The relative accuracy of the computed value $\overline{F}(x,y)$ by our algorithm satisfies

$$\frac{|F(x,y)-\overline{F}(x,y)|}{|F(x,y)|}\le u+\operatorname{cond}(F,x,y)\times\mathcal{O}(u^2). \tag{4}$$

This bound tells us that the computed values can be relative accurate up to the unit roundoff as long as the problem has a condition number lower than $1/u$.

Partly results of this paper have been presented in the Minisymposia "Accurate algorithms and applications" of 2012 SIAM Conference on Applied Linear Algebra, this paper is an extended version with the detailed proof.

The paper is organized as follows. In section 2 we introduce some basic notations and results about floating-point arithmetic, error-free transformations and the `DCTP` algorithm. In section 3, the compensated `DCTP` algorithm for the evaluation of Bézier tensor product surfaces is provided. In section 4 an *a priori* error analysis is carried out. Finally, in section 5 several numerical tests illustrate the efficiency and accuracy of the proposed algorithm.

2 Basic Notation and Results

2.1 Floating-point Arithmetic and Error-free Transformations

In this paper we assume all the floating-point computation is performed in double precision, with the "round to the nearest" rounding mode and no underflow occurring. We also assume that the computation in floating point arithmetic obeys the model

$$a \text{ op } b = \mathrm{fl}(a \circ b) = (a \circ b)(1 + \varepsilon_1) = (a \circ b)/(1 + \varepsilon_2), \tag{5}$$

where op $\in \{\oplus, \ominus, \otimes, \oslash\}$, $\circ \in \{+, -, \times, \div\}$ and $|\varepsilon_1|, |\varepsilon_2| \le u$. The symbol u is the unit round-off and $'\text{op}'$ represents the floating-point computation, e.g. $a \oplus b = \mathrm{fl}(a + b)$. We also assume that the computed result of $\alpha \in \mathbb{R}$ in floating-point arithmetic is denoted by $\hat{a}$ or $\mathrm{fl}(a)$ and $\mathbb{F}$ denotes the set of all floating-point numbers (see [11] for more details). Following [11], we will use the following classic properties in error analysis (we always assume that $nu < 1$).

(i) if $|\delta_i| \le u$, $\rho_i = \pm 1$, then $\prod_{i=1}^{n}(1 + \delta_i)^{\rho_i} = 1 + \theta_n$,

(ii) $(1 + \theta_k)(1 + \theta_j) \le (1 + \theta_{k+j})$,

(iii) $|\theta_n| \le \gamma_n := nu/(1 - nu)$ and $nu(1 + \gamma_n) = \gamma_n$,

(iv) $(1 + \gamma_k)\gamma_j \le \gamma_k + \gamma_j + \gamma_k\gamma_j \le \gamma_{k+j}$ and $\gamma_k < \gamma_{k+1}$,

(v) $u \le u(1 + u) \le \gamma_1 \le \gamma_3/3$,

(vi) $\gamma_i\gamma_j \le \gamma_{i+t}\gamma_{j-t}$, if $j - i > t > 0$.

Then let us introduce some results concerning error-free transformations (EFT). For a pair of floating-point numbers $a, b \in \mathbb{F}$, when no underflow occurs, there exists a floating-point number y satisfying $a \circ b = x + y$, where $x = \mathrm{fl}(a \circ b)$ and $\circ \in \{+, -, \times\}$. The transformation $(a, b) \longrightarrow (x, y)$ is regarded as an error-free transformation. The error-free transformation algorithms of the sum and product of two floating-point numbers used later in this paper are the `TwoSum` algorithm by Knuth [15] and the `TwoProd` algorithm by Dekker [3], respectively (see Appendix B). The following theorem exhibits the properties of the `TwoSum` and `TwoProd` algorithms (see [23]).

Theorem 2.1 [23] *For $a, b \in \mathbb{F}$ and $x, y \in \mathbb{F}$,* `TwoSum` *and* `TwoProd` *verify*

$$[x, y] = \texttt{TwoSum}(a, b), x = \mathrm{fl}(a + b), x + y = a + b, |y| \le u|x|, |y| \le u|a + b|,$$
$$[x, y] = \texttt{TwoProd}(a, b), x = \mathrm{fl}(a \times b), x + y = a \times b, |y| \le u|x|, |y| \le u|a \times b|.$$

2.2 Compensated Algorithm for the Evaluation of a Bézier Curve

Let us recall the DC algorithm for the evaluation of a Bézier curve $p(t) = \sum_{i=0}^{n} b_i B_i^n(t)$ on $t \in [0, 1]$.

***Algorithm* 1** *de Casteljau algorithm for Bézier curve evaluation*

```
function DC($p, t$)
$\widehat{b}_i^{(0)} = b_i,\ i = 0, \cdots, n$
$r = 1 \ominus t$
for $j = 1 : 1 : n$
    for $i = 0 : 1 : n - j$
        $\widehat{b}_i^{(j)} = \widehat{b}_i^{(j-1)} \otimes r \oplus \widehat{b}_{i+1}^{(j-1)} \otimes t$
    end
end
DC($p, t$) $\equiv \widehat{b}_0^{(n)}$
```

The following result states the forward error bound of the DC algorithm.

Theorem 2.2 *If* $p(t) = \sum_{i=0}^{n} b_i B_i^n(t)$ *and* $\widehat{p}(t)$ *is the computed value of the de Casteljau algorithm then*

$$|p(t) - \widehat{p}(t)| \leq \gamma_{3n} \sum_{i=0}^{n} |b_i| B_i^n(t). \tag{6}$$

Proof: Considering that there will exist roundoff error in the process of $1 - a_{j+1}^{i-1}(x)$ in Proposition 3.1 of [21], we can deduce this theorem from Corollary 3.2 of [21] by substituting γ_{2n} with γ_{3n} directly. □

Note that the difference of Theorem 2.2 and the classical result of [21] is that we consider the error of the evaluation of the term $(1 - t)$, and so we have the term γ_{3n} instead of γ_{2n}.

In [14] the authors proposed the following compensated algorithm to accurately evaluate a Bézier curve.

***Algorithm* 2** *Compensated algorithm for Bézier curve evaluation*

```
function $[res_{org}, err, res_{fin}]$=CompDC($p, t$)
$\widehat{b}_i^{(0)} = b_i,\ \widehat{\epsilon b}_i^{(0)} = 0,\quad i = 0, \cdots, n$
$[\widehat{r}, \rho]$=TwoSum($1, -t$)
for $j = 1 : 1 : n$
    for $i = 0 : 1 : n - j$
        $[s, \pi_i^{(j)}]$=TwoProd($\widehat{b}_i^{(j-1)}, \widehat{r}$)
        $[v, \sigma_i^{(j)}]$=TwoProd($\widehat{b}_{i+1}^{(j-1)}, t$)
        $[\widehat{b}_i^{(j)}, \beta_i^{(j)}]$=TwoSum($s, v$)
        $\widehat{w}_i^{(j)} = \pi_i^{(j)} \oplus \sigma_i^{(j)} \oplus \beta_i^{(j)} \oplus \widehat{b}_i^{(j-1)} \otimes \rho$
        $\widehat{\epsilon b}_i^{(j)} = \widehat{\epsilon b}_i^{(j-1)} \otimes \widehat{r} \oplus (\widehat{\epsilon b}_{i+1}^{(j-1)} \otimes t \oplus \widehat{w}_i^{(j)})$
    end
end
$res_{org} = \widehat{b}_0^{(n)}$ and $err = \widehat{\epsilon b}_0^{(n)}$
CompDC($p, t$) $\equiv res_{fin} = \widehat{b}_0^{(n)} \oplus \widehat{\epsilon b}_0^{(n)}$
```

This algorithm has improved stability properties. After correcting a small missprint in Theorem 4 of [14] we have

Theorem 2.3 [14] *Consider the computed result $\widehat{\epsilon b}_0^{(n)}$ of Algorithm 2 and its corresponding theoretical result $\epsilon b_0^{(n)}$, if no underflow occurs and $n \geq 2$, then*

$$|\epsilon b_0^{(n)} - \widehat{\epsilon b}_0^{(n)}| \leq 2\gamma_{3n+2}\gamma_{3(n-1)} \sum_{i=0}^{n} |b_i| B_i^n(t). \tag{7}$$

Note that

$$p(t) = \widehat{p}(t) + \epsilon b_0^{(n)}, \tag{8}$$

and so the principle of Algorithm 2 is just to find an approximate value $\widehat{\epsilon b}_0^{(n)}$ of $\epsilon b_0^{(n)}$ and to correct the result by the `DC` algorithm. Then the final error bound (see Theorem 5 in [14]) verifies

$$|\texttt{CompDC}(p,t) - p(t)| \leq u|p(t)| + 2\gamma_{3n}^2 \sum_{i=0}^{n} |b_i| B_i^n(t). \tag{9}$$

This bound illustrates that Algorithm 2 is more accurate than Algorithm 1 and it provides a relative accurate result up to the unit roundoff as long as the condition number $\sum_{i=0}^{n} |b_i| B_i^n(t)/|p(t)|$ is lower than $u/2\gamma_{3n}^2 \approx 1/18n^2u$.

3 CompDCTP Algorithm and its Error Analysis

In this section we propose a compensated de Casteljau tensor product algorithm for the evaluation of a Bézier tensor product surface. Besides, the forward error analysis of the algorithm is performed.

3.1 Compensated De Casteljau Tensor Product Algorithm

Since a Bézier tensor product surface can be obtained by moving the control points of a curve along other Bézier curves, the `DCTP` algorithm is based on the `DC` algorithm for the Bézier curve evaluation. For the evaluation of the surface (1), the `DCTP` algorithm can be written explicitly in the following pseudo-Matlab code:

***Algorithm* 3** *de Casteljau tensor product algorithm for Bézier surface evaluation*

```
function DCTP($F, x, y$)
$f_{i,j}^{(0)} = b_{i,j}$ for $0 \leq i \leq m$ and $0 \leq j \leq n$
for $i = 0 : 1 : m$
    $\widehat{f}_{i,0}^{(1)} = DC(f_{i,:}^{(0)}, y)$
end
$\widehat{f}_{0,0}^{(2)} = DC(\widehat{f}_{:,0}^{(1)}, x)$
DCTP($F, x, y$) $\equiv \widehat{f}_{0,0}^{(2)}$
```

Here $f_{i,:}^{(0)}$ is an $1 \times (n+1)$ vector and $f_{:,0}^{(1)}$ is an $(m+1) \times 1$ vector.

Theorem 3.1 *Let $F(x,y)$ be a Bézier tensor product surface given by (1) and let us suppose that $2(m+n)u < 1$, where u is the unit roundoff. Then the value $\widehat{F}(x,y) = fl(F(x,y))$ computed in floating-point arithmetic through Algorithm 3 satisfies*

$$|F(x,y) - \widehat{F}(x,y)| \leq \gamma_{3(m+n)} \sum_{i=0}^{m} \sum_{j=0}^{n} |b_{i,j}| B_i^m(x) B_j^n(y). \tag{10}$$

Proof: Using the forward error bound of Theorem 5 of [4], we just apply it to the particular case of a Bézier tensor product surface. If we take into account the roundoff error generated in the evaluation of $1-y$ then the term $\gamma_{2(m+n)}$ in the original error bound of [4] should be changed to $\gamma_{3(m+n)}$. □

The term $\sum_{i=0}^{m} \sum_{j=0}^{n} |b_{i,j}| B_i^m(x) B_j^n(y)$ is an absolute condition number of the evaluation of $F(x,y)$ (see (16) in [4]).

Motivated by subsection 2.2 we propose to use `CompDC` algorithm instead `DC` algorithm in Algorithm 3 and therefore, to improve `DCTP` algorithm in order to obtain the Compensated `DCTP` algorithm. According to (8), we have

$$f_{i,0}^{(1)} = \widehat{f}_{i,0}^{(1)} + e1_{i,0}, \quad 0 \leq i \leq m, \tag{11}$$

where $e1_{i,0}$ is the theoretical error generated by the process $\widehat{f}_{i,0}^{(1)} = \mathtt{DC}(f_{i,:}^{(0)}, y)$ and

$$f_{i,0}^{(1)} = \sum_{j=0}^{n} b_{i,j} B_j^n(y), \tag{12}$$

is the exact result for each i. In the same way, we also have

$$\widetilde{f}_{0,0}^{(2)} = \widehat{f}_{0,0}^{(2)} + e2, \tag{13}$$

where $e2$ is the theoretical error generated by the process $\widehat{f}_{0,0}^{(2)} = \mathtt{DC}(\widehat{f}_{:,0}^{(1)}, x)$ and

$$\widetilde{f}_{0,0}^{(2)} = \sum_{i=0}^{m} \widehat{f}_{i,0}^{(1)} B_i^m(x), \tag{14}$$

is the exact result. From (11)-(14), we can deduce

$$\sum_{i=0}^{m} \sum_{j=0}^{n} b_{i,j} B_i^m(x) B_i^n(y) = \widehat{f}_{0,0}^{(2)} + e2 + \sum_{i=0}^{m} e1_{i,0} B_i^m(x), \tag{15}$$

that is

$$F(x,y) = \widehat{F}(x,y) + e, \tag{16}$$

where $e = e2 + e3$ and

$$e3 = \sum_{i=0}^{m} e1_{i,0} B_i^m(x). \tag{17}$$

Obviously, the corrected result $\overline{F}(x,y) = \widehat{F}(x,y) \oplus \widehat{e}$ is expected to be more accurate than the floating-point result $\widehat{F}(x,y)$ of Algorithm 3, where $\widehat{e}$ is an approximation of e. Fortunately, `CompDC` algorithm can give us the approximate value of $\widehat{e2}$ and $\widehat{e1}_{i,0}$, and therefore it is easy to get $\widehat{e}$. The previous discussion leads to the following compensated de Casteljau tensor product algorithm.

***Algorithm* 4** *Compensated de Casteljau tensor product algorithm for Bézier surface evaluation*

```
function $[res_{org}, err, res_{fin}]$=CompDCTP$(F, x, y)$
$f^{(0)}_{i,j} = b_{i,j}$ for $0 \le i \le m$ and $0 \le j \le n$
for $i = 0 : 1 : m$
    $[\widehat{f}^{(1)}_{i,0}, \widehat{e1}_{i,0}]$ = CompDC$(f^{(0)}_{i,:}, y)$
end
$[\widehat{f}^{(2)}_{0,0}, \widehat{e2}]$ = CompDC$(\widehat{f}^{(1)}_{:,0}, x)$
$err \equiv \widehat{e} = \widehat{e2} \oplus$ DC$(\widehat{e1}_{:,0}, x)$
$res_{org} \equiv \widehat{F}(x, y) = \widehat{f}^{(2)}_{0,0}$
CompDCTP$(F, u, v) \equiv res_{fin} = \overline{F}(x, y) = res_{org} \oplus err$
```

3.2 Error Bound of Bézier Tensor Product Surface Evaluation

Theorem 3.2 *Let $F(x, y)$ be a Bézier tensor product surface in the form* (1) *and $0 \le x, y \le 1$. If $6mu \le 1$ and $6nu \le 1$, then the forward error bound of the compensated de Casteljau tensor product algorithm* (Algorithm 4) *is such that*

$$|\texttt{CompDCTP}(F, u, v) - F(x, y)| \le u|F(x, y)| + 5(\gamma^2_{3m+1} + \gamma^2_{3n+1})\widetilde{F}(x, y), \tag{18}$$

where

$$\widetilde{F}(x, y) = \sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}|B^m_i(x)B^n_j(y). \tag{19}$$

Proof: From Algorithm 4 we have

$$\overline{F}(x, y) = \widehat{F}(x, y) \oplus \widehat{e} = (\widehat{F}(x, y) + \widehat{e})(1 + \delta),$$

then by (16)

$$\overline{F}(x, y) = F(x, y)(1 + \delta) + (\widehat{e} - e)(1 + \delta).$$

Hence,

$$|\overline{F}(x, y) - F(x, y)| \le u|F(x, y)| + (1 + u)|\widehat{e} - e|. \tag{20}$$

Now, the problem is to find a bound of the term $|\widehat{e} - e|$. Since $\widehat{e} = \widehat{e2} \oplus \widehat{e3} = (\widehat{e2} + \widehat{e3})(1 + \delta)$, $|\delta| < u$, and $e = e2 + e3 = (e2 + e3)(1 + \delta) - \delta(e2 + e3)$, we have

$$|\widehat{e} - e| \le u|e2 + e3| + (1 + u)(|e2 - \widehat{e2}| + |e3 - \widehat{e3}|). \tag{21}$$

First, let us give a bound of the term $e2 + e3$ in (21). From (16) and Theorem 3.1, we obtain

$$|e2 + e3| \le \gamma_{3(m+n)} \sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}|B^m_i(x)B^n_j(y). \tag{22}$$

Now, we will present the bound of the term $|e2 - \widehat{e2}|$ in (21). Seeing that $e2$ and $\widehat{e2}$ are just the theoretical error and the computed one while evaluating the Bézier curve with the control points $\widehat{f}^{(1)}_{i,0}$, from Theorem 2.3 we have

$$|e2 - \widehat{e2}| \le 2\gamma_{3m+2}\gamma_{3(m-1)} \sum_{i=0}^{m} |\widehat{f}^{(1)}_{i,0}|B^m_i(x). \tag{23}$$

By (20) of Theorem 5 in [4], and taking into account the proof of Theorem 3.1, we can deduce

$$|\widehat{f}_{i,0}^{(1)}| \le \sum_{j=0}^{n} |f_{i,j}^{(0)}|(1+\gamma_{3n})B_j^n(y). \tag{24}$$

Hence, from (23), (24) and $f_{i,j}^{(0)} = b_{i,j}$, we have

$$|e2 - \widehat{e2}| \le 2\gamma_{3m+2}\gamma_{3(m-1)}(1+\gamma_{3n}) \sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}|B_j^n(y)B_i^m(x). \tag{25}$$

Finally, the bound of $|e3 - \widehat{e3}|$ in (21) will be provided. We denote the following equations just like (17),

$$e3_{mid} = \sum_{i=0}^{m} \widehat{e1}_{i,0}B_i^m(x), \tag{26}$$

$$\widehat{e3} = \mathtt{DC}(\widehat{e1}_{:,0}, x). \tag{27}$$

Hence, we have

$$|e3 - \widehat{e3}| \le |e3 - e3_{mid}| + |e3_{mid} - \widehat{e3}|. \tag{28}$$

By (17) and (26) we deduce

$$|e3 - e3_{mid}| = \sum_{i=0}^{m} |e1_{i,0} - \widehat{e1}_{i,0}|B_i^m(x). \tag{29}$$

From `DCTP` algorithm (Algorithm 4), we found that $e1_{i,0}$ and $\widehat{e1}_{i,0}$ are the theoretical errors and computed ones by `CompDC` Algorithm (Algorithm 2) when evaluating the Bézier curve $\sum_{j=0}^{n} f_{i,j}^{(0)} B_i^n(y)$, respectively. From Theorem 2.3 and $f_{i,j}^{(0)} = b_{i,j}$ we have

$$|e1_{i,0} - \widehat{e1}_{i,0}| \le 2\gamma_{3n+2}\gamma_{3(n-1)} \sum_{j=0}^{n} |b_{i,j}|B_i^n(y). \tag{30}$$

By (29) and (30), we get

$$|e3 - e3_{mid}| \le 2\gamma_{3n+2}\gamma_{3(n-1)} \sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}|B_i^n(y)B_i^m(x). \tag{31}$$

Meanwhile, since $e3_{mid}$ represents a Bézier curve polynomial of degree m with $\widehat{e1}_{i,0}$ as coefficients and $\widehat{e3}$ is the corresponding computed result by `DC` algorithm, from Theorem 2.2 we obtain

$$|e3_{mid} - \widehat{e3}| \le \gamma_{3m} \sum_{i=0}^{m} |\widehat{e1}_{i,0}|B_i^m(x). \tag{32}$$

We have that

$$|\widehat{e1}_{i,0}| \le |e1_{i,0}| + |e1_{i,0} - \widehat{e1}_{i,0}|, \tag{33}$$

where $e1_{i,0}$ is the theoretical error for the evaluation of the Bézier curve polynomial $\sum_{j=0}^{n} f_{i,j}^{(0)} B_i^n(y)$ and satisfies (8). Then from Theorem 2.2, we obtain

$$|e1_{i,0}| \le \gamma_{3n} \sum_{j=0}^{n} |f_{i,j}^{(0)}|B_j^n(y) = \gamma_{3n} \sum_{j=0}^{n} |b_{i,j}|B_j^n(y). \tag{34}$$

By (30), (33) and (34) we can deduce

$$|\widehat{e1}_{i,0}| \le (\gamma_{3n} + 2\gamma_{3n+2}\gamma_{3(n-1)}) \sum_{j=0}^{n} |b_{i,j}| B_j^n(y), \tag{35}$$

and then by (32)

$$|e3_{mid} - \widehat{e3}| \le (\gamma_{3m}\gamma_{3n} + 2\gamma_{3m}\gamma_{3n+2}\gamma_{3(n-1)}) \sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}| B_j^n(y) B_i^m(x). \tag{36}$$

Hence, by (28), (31) and (36) we have

$$|e3 - \widehat{e3}| \le (2\gamma_{3n+2}\gamma_{3(n-1)}(1+\gamma_{3m}) + \gamma_{3m}\gamma_{3n}) \sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}| B_j^n(y) B_i^m(x). \tag{37}$$

Now by (21), (22), (25) and (37), we obtain

$$|\widehat{e} - e| \le \alpha(m,n) \sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}| B_j^n(y) B_i^m(x), \tag{38}$$

where

$$\begin{aligned}\alpha(m,n) =&(1+u)(2\gamma_{3n+2}\gamma_{3(n-1)}(1+\gamma_{3m}) + 2\gamma_{3m+2}\gamma_{3(m-1)}(1+\gamma_{3n}) + \gamma_{3m}\gamma_{3n}) \\ &+ u\gamma_{3(m+n)}\end{aligned}$$

Then, by the properties (iv) and (v) of floating-point arithmetic, we have $u(1+u)\gamma_{3(m+n)} \le \gamma_1\gamma_{3(m+n)+1} \le \gamma_{3m+1}\gamma_{3n+1} \le \frac{1}{2}(\gamma_{3m+1}^2 + \gamma_{3n+1}^2)$. And since $6mu \le 1$ and $6nu \le 1$, we obtain that $\gamma_{3m}, \gamma_{3n} \le 1$. Then $(1+u)^2(2\gamma_{3n+2}\gamma_{3(n-1)}(1+\gamma_{3m})) \le 4\gamma_{3n+1}^2$ and $(1+u)^2(2\gamma_{3m+2}\gamma_{3(m-1)}(1+\gamma_{3n})) \le 4\gamma_{3m+1}^2$. With $(1+u)^2\gamma_{3m}\gamma_{3n} \le \gamma_{3m+1}\gamma_{3n+1} \le \frac{1}{2}(\gamma_{3m+1}^2 + \gamma_{3n+1}^2)$, we can finally deduce

$$(1+u)\alpha(m,n) \le 5(\gamma_{3m+1}^2 + \gamma_{3n+1}^2) \tag{39}$$

Therefore, by (20), (38) and (39) we can obtain (18) and (19). □

The relative condition number for the evaluation of $F(x,y)$ at entry x, y can be defined by

$$\mathtt{cond}(F,x,y) = \frac{\widetilde{F}(x,y)}{|F(x,y)|} = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} |b_{i,j}| B_i^m(x) B_i^n(y)}{|\sum_{i=0}^{m}\sum_{j=0}^{n} b_{i,j} B_i^m(x) B_i^n(y)|}, \tag{40}$$

Then from Theorem 3.2, we can obtain

$$\frac{|\mathtt{CompDCTP}(F,u,v) - F(x,y)|}{|F(x,y)|} \le u + 5(\gamma_{3m+1}^2 + \gamma_{3n+1}^2)\mathtt{cond}(F,x,y). \tag{41}$$

It is observed that if $5(\gamma_{3m+1}^2 + \gamma_{3n+1}^2)\mathtt{cond}(F,x,y) < u$, the relative error of the result computed by Algorithm 4 is bounded by the constant value u. Thus, formula (4) can be easily deduced from (41) and it illustrates that the computed value is as accurate as the result computed by the `DCTP` algorithm with twice working precision and then rounded to the working precision.

4 Numerical Experiments

All our experiments are performed in IEEE-754 double precision with unit roundoff $u \simeq 1.16 \times 10^{-16}$, and the programs have been written in Matlab 7.0. We consider Bézier tensor product surfaces with floating-point coefficients and the floating-point entry (x, y) defined on $[0, 1] \times [0, 1]$ in the form (1).

Generally, since the Bernstein tensor product basis is well conditioned, the de Casteljau tensor product algorithm presents great accuracy [4]. However, there still exists excessively ill-conditioned problems, and in this case `DCTP` algorithm can not yield enough accuracy digits. A typical example in 1D is a polynomial with multiple roots like in [14], $P(t) = (t - 0.75)^3(t - 0.2)^3$. We consider now a 2D extension on the unit square $[0, 1] \times [0, 1]$

$$P(x, y) = (x - 0.75)^3(x - 0.2)^3(y - 0.75)^3(y - 0.2)^3 \tag{42}$$

and we evaluate its approximate Bézier tensor product form.

The change of basis algorithm from a bivariate polynomial in power form into Bézier tensor product form is taken from [2]. We use the symbolic toolbox in Matlab 7.0 to obtain the converted polynomial (see Appendix D). However, it must be noticed that with IEEE-754 double precision, the coefficients of the polynomial in Bézier tensor product form are the rational fraction, and they have to be rounded to the nearest floating-point number. Therefore, the evaluated bivariate polynomial in Bernstein tensor product basis, denoted by $F(x, y)$, is different from the bivariate original polynomial $P(x, y)$ in power basis (42) .

We evaluate the approximate Bézier tensor product surface $F(x, y)$ at 400 points uniformly distributed near the point $(0.75, 0.2)$, using `DCTP`, `CompDCTP` and `DCTP` algorithms with quad-double arithmetic [10] (`QDDCTP`). The results are reported in Figure 1. It is clear that our compensated method can recover the expected smooth surface, just like the original `DCTP` algorithm with quad-double arithmetic, when the results are rounded to the working precision. Meanwhile, if we evaluate the approximated polynomial in Bézier tensor product form at the point $(0.75, 0.2)$ using the symbolic computation, we will obtain the value $F(0.75, 0.2) = -2.853943049292987 \cdot 10^{-22}$, which is nearly the same value as the result obtained by using the QD library [10]. This fact is also illustrated in Figure 1 by comparing the pictures on the left. The surface of the transformed polynomial with floating point coefficients in Bézier tensor product is very similar to the surface of the original polynomial in power basis after a little translation (left insets of Figure 1). Notice that picture by `DCTP` algorithm is a fold in surface, with the evaluations at the direction x varying slightly. This shape is due to the fact that the `DCTP` algorithm itself first compute in the direction y, and then the roundoff errors generated by the computation on y have a stronger impact on the final evaluation than those in x. Therefore, from Figure 1 we observe that the `CompDCTP` algorithm works perfectly and it has reduced the effect of the rounding errors. What is maintained, of course, is the error in the expression of the polynomial in the new basis.

Figure 2 shows the logarithms of the relative errors on the base 10 by the evaluation algorithms `DCTP` and `CompDCTP` for 2500 points around the point $(0.75, 0.2)$. It is obvious that `CompDCTP` algorithm has much better precision than `DCTP` algorithm. Notice that the relative errors of `CompDCTP` algorithm are equal to or smaller than u even in the neighborhood of the point $(0.75, 0.2)$.

Next, we focus on the relative forward error bounds for general ill-conditioned polynomials. So, we have generated and tested polynomials in tensor product form of

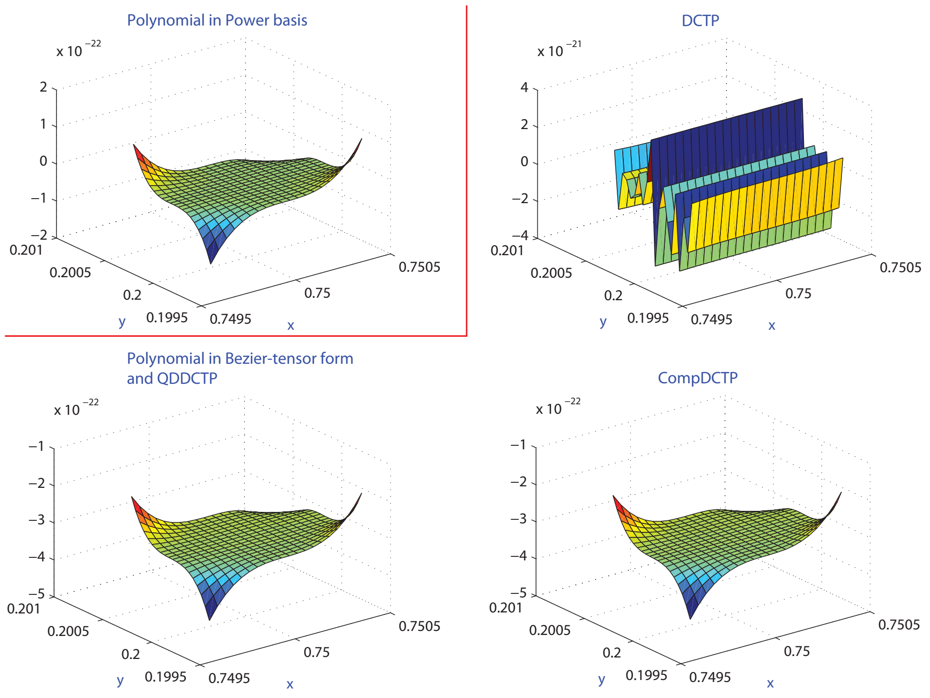

Figure 1: Evaluation of the polynomial (42) in the neighborhood of the point $(0.75, 0.2)$, both using the original power basis expression using the symbolic expression (upper left) and the approximate Bézier tensor product surface by means of the `DCTP` (upper right), `CompDCTP` (lower right) and `QDDCTP`—symbolic expression (lower left).

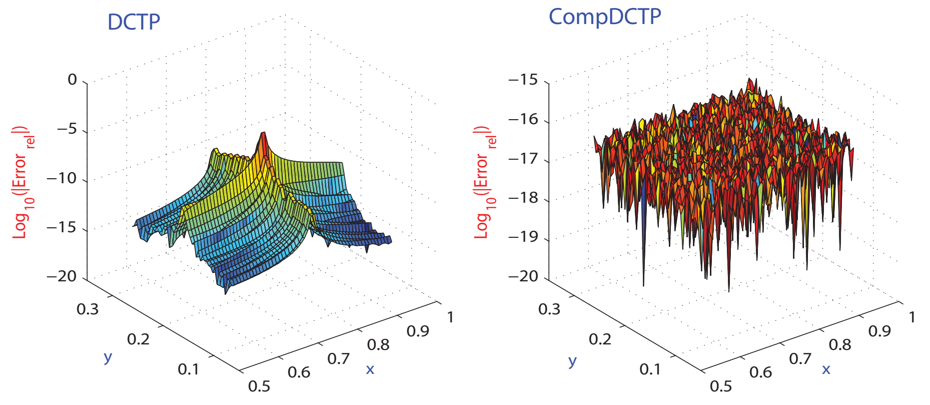

Figure 2: Logarithms of the relative errors of the `DCTP` (left), `CompDCTP` (right) algorithms.

degree $m \times n = 6 \times 7$ with condition numbers varying from 10^4 to 10^{35}. This generation algorithm is shown in Appendix A, which is similar to the algorithm `GenPoly` on page 52 in [20]. The results are reported on Figure 3. In this numerical tests, the exact evaluation of the polynomial is obtained by the `DCTP` algorithm in quad-double format (`QDDCTP`). As we can see, `CompDCTP` algorithm exhibits the expected behavior. When the condition number is smaller than $1/u$, the relative error by `CompDCTP` is equal to or smaller than u. This relative error increases linearly for the condition number between $1/u$ and $1/u^2$. Meanwhile, we observe that in some cases (5 points) the `DCTP` algorithm has high accuracy. This phenomena is due to the good stability of the algorithm.

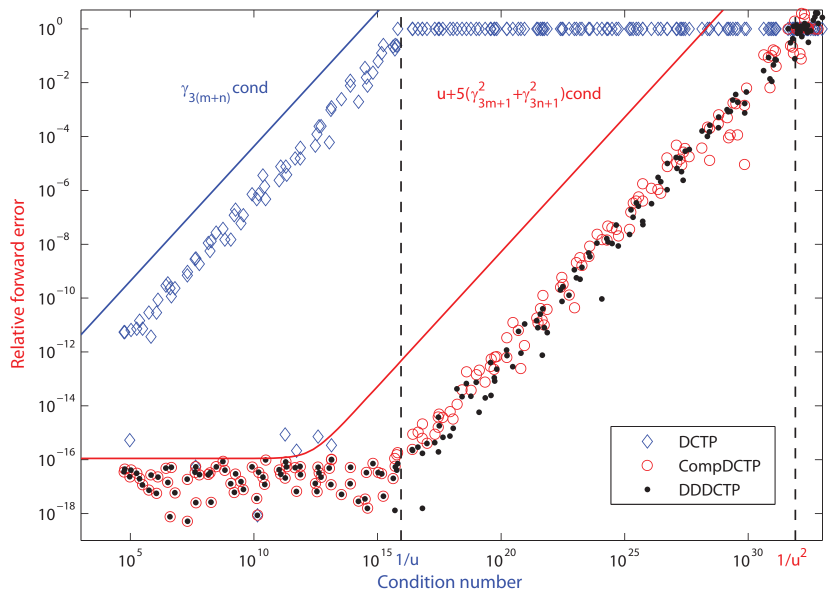

Figure 3: Accuracy of evaluation of polynomials in Bézier tensor product form with respect to the condition number

Increasing the working precision is another direct possible way to improve the accuracy apart from compensated methods, such as using the Bailey's *double-double* [19, 1] (double-double numbers are represented as an unevaluated sum of a leading double and a trailing double). In Appendix B, we present the algorithms to compute the product and addition of two double-double or a double times a double-double. For comparison we propose an algorithm that evaluates the Bézier tensor product surface with floating-point coefficients and entries using internally double-double number, which is abbreviated to `DDDCTP` algorithm. But first we need the de Casteljau algorithm in double-double format (abbreviated to `DDDC`) for the evaluation of the Bézier curve $p(t) = \sum_{i=0}^{n} b_i B_i^n(t)$, with the coefficients expressed in terms of $b_i = b1_i + b2_i$ in double-double precision, where $|b2_i| \leq u|b1_i|$. Especially, when $b2_i = 0$, the precision of b_i degenerates into double. These two algorithms with double-double arithmetic are shown in Appendix C. Here the result by the `DDDCTP` algorithm should be rounded to the working precision (double precision). As we see in Figure 3, the `CompDCTP` algorithm has nearly the same accuracy as the `DDDCTP` algorithm.

Finally, we pay attention to the computational complexity of all the algorithms.

- `DC`: $1.5n^2 + 1.5n + 1$
- `DCTP`: $(1.5n^2 + 1.5n + 1)(m + 1) + 1.5m^2 + 1.5m + 1$
- `CompDC`: $24n^2 + 24n + 7$
- `CompDCTP`: $(24n^2 + 24n + 7)(m + 1) + 24m^2 + 24m + 7 + 1$
- `DDDC`: $33n^2 + 33n + 6$
- `DDDCTP`: $(33n^2 + 33n + 6)(m + 1) + 33m^2 + 33m + 6$

As we know, `TwoSum`, `TwoProd` and `FastTwoSum` algorithms require 6, 17 and 3 flops respectively. In Appendix B, since $th \geq tl$, we modify `Prod_dd_d` and `Prod_dd_dd` in [7, 18] by using `FastTwoSum` as a substitute for `TwoSum` to decrease computation complexity. The improved `Prod_dd_d` is similar to that in [20]. Taking into account the previous comparison of the accuracy, we can affirm that `CompDCTP` algorithm is as accurate as `DDDCTP` algorithm but only requires on the average about 72.7% of flop count of that one.

We have tested the running time of the above six algorithms in a C code with the four environments listed as follows:

I) Intel Celeron 2.66Ghz, 512MB, Microsoft Visual C++ 6.0.

II) Intel Pentium 4 3.06Ghz, 512MB, Microsoft Visual C++ 9.0.

III) Intel Pentium Dual CPU each 1.8Ghz, Microsoft Visual C++ 6.0.

IV) Intel Pentium Dual CPU each 1.8Ghz, Microsoft Visual C++ 9.0.

We optimize the C codes of the algorithms `CompDC`, `DDDC`, `CompDCTP`, `DDDCTP` by taking the procedure `Split`$(2 \times x)$ (which is used in `TwoProd`) out of recurrence. This optimization technique is taken from [20]. The numerical tests are performed with bivariate Bézier tensor product surfaces whose degree $m \times n$ vary from 25×25 to 200×200. The entries and coefficients of the polynomials tested are random floating-point numbers uniformly distributed in the interval $(-1, 1)$. The average measured computing time ratios of `CompDC` over `DDDC` and `CompDCTP` over `DDDCTP` are reported in Table 1. We observe that the running time ratio is better than the flop count one. Thanks to the analysis in terms of instruction level parallelism (ILP) (see details in [17] and [20]), this phenomenon is surprising, but reasonable. As a consequence, `CompDCTP` runs much faster than `DDDCTP` but with the same accuracy, and similar results are obtained on the comparison of `CompDC` and `DDDC`.

Table 1: Running time ratios of compensated algorithms versus the algorithms with double-double arithmetic

	I)	II)	III)	IV)
`CompDC/DDDC`	$40\% \sim 48\%$	$39\% \sim 43\%$	$57\% \sim 66\%$	$54\% \sim 59\%$
`CompDCTP/DDDCTP`	$29\% \sim 33\%$	$47\% \sim 51\%$	$36\% \sim 38\%$	$64\% \sim 68\%$

5 Conclusion

In this paper we have provided an accurate algorithm for computing Bézier tensor product surfaces with floating-point coefficients. This algorithm is based on error-free transformations and it gives a relative accurate algorithm. The forward error analysis of the new algorithm is performed. Finally, we compare this algorithm with the classical de Casteljau tensor product algorithm using internally a double-double library. The results show that our algorithm is much more accurate than the classical algorithm in double precision but faster and as accurate as the classical algorithm using high-precision (double-double library).

A The Generation Algorithm of Ill-conditioned Polynomials

***Algorithm* 5** *Generate an ill-conditioned polynomial* $F(x,y)=\sum_{i=0}^{m}\sum_{j=0}^{n} b_{i,j}B_i^m(x)B_i^n(y)$ *in the form of a Bézier tensor product surface*

```
    $[p, vact, cact] = \texttt{GenPolyDCTP}(m, n, x, y, v, cexp)$
    % v——-the expected evaluation of the polynomial;
    % cexp—the expected condition number;
    % x, y—the coordinates;
    % vact—-the actual evaluation of the polynomial;
    % cact—-the actual condition number;
    $b_{i,j} = 0$, for $i = 0 : m$, $j = 0 : n$;
    $num = (m+1)*(n+1)$;
    $d2 = ceil(num/2)$;
    $vb = log2(cexp * abs(v))$;
    $I = J = zeros(1, num)$;
    $perm = randperm(num)$;
    Change perm to coordinates $(I, J)$, where $(I(i), J(i))$ is a random coefficient
        of the surface in accord with $perm(i)$ in some sort order;
    $b_{I(1),J(1)} = (2*rand-1)/B^m_{I(1)}(x)B^n_{J(1))}(y)$;
    $b_{I(2),J(2)} = (2*rand-1)*2^{vb}/B^m_{I(2)}(x)B^n_{J(2))}(y)$;
    for $i = 3 : d2$
       $b_{I(i),J(i)} = (2*rand-1)*2^{vb*rand}/B^m_{I(i)}(x)B^n_{J(i))}(y)$;
    end
    $log2v = log2(abs(v))$;
    $m = [(2*rand(1, num-d2)-1).*2^{linspace(vb,log2v,num-d2)}]$;
    for $i = d2+1 : num-1$
       $b_{I(i),J(i)} = (m(i-d2) - \texttt{QDDCTP}(F,x,y))/B^m_{I(i)}(x)B^n_{J(i))}(y)$;
    end
    $b_{I(num),J(num)} = (v - \texttt{QDDCTP}(F,x,y))/B^m_{I(num)}(x)B^n_{J(num))}(y)$;
    $vact = \texttt{QDDCTP}(F,x,y)$;
    $cact = \texttt{QDDCTP}(abs(F),x,y)/abs(vact)$;
```

B Error Free Transformations and Double-Double Library

***Algorithm* 6** *[15] Error-free transformation of the sum of two floating-point numbers*

```
function $[x, y] = \texttt{TwoSum}(a, b)$
    $x = a \oplus b$
    $z = x \ominus a$
    $y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$
```

***Algorithm* 7** *[3] Error-free split of a floating-point numbers into two parts*

```
function $[x, y] = \texttt{Split}(a)$
    $c = \text{factor} \otimes a$    (in double precision factor $= 2^{27} + 1$)
    $x = c \ominus (c \ominus a)$
    $y = a \ominus x$
```

***Algorithm* 8** *[3] Error-free transformation of the product of two floating-point numbers*

```
function $[x, y] = \texttt{TwoProd}(a, b)$
    $x = a \otimes b$
    $[a1, a2] = \texttt{Split}(a)$
    $[b1, b2] = \texttt{Split}(b)$
    $y = a2 \otimes b2 \ominus (((x \ominus a1 \otimes b1) \ominus a2 \otimes b1) \ominus a1 \otimes b2)$
```

***Algorithm* 9** *[3] Error-free transformation of the sum of two floating-point numbers ($|a| \geq |b|$)*

```
function $[x, y] = \texttt{FastTwoSum}(a, b)$
    $x = a \oplus b$
    $y = (a \ominus x) \oplus b$
```

***Algorithm* 10** *[20] Addition of double-double number and a double number*

```
$[rh, rl] = \texttt{add\_dd\_d}(ah, al, b)$;
$[th, tl] = \texttt{TwoSum}(ah, b)$;
$tl = al \oplus tl$;
$[rh, rl] = \texttt{FastTwoSum}(th, tl)$.
```

Algorithm 10 requires 10 flops.

***Algorithm* 11** *[19] Addition of double-double number and double-double number*

```
$[rh, rl] = \texttt{add\_dd\_dd}(ah, al, bh, bl)$;
$[sh, sl] = \texttt{TwoSum}(ah, bh)$;
$[th, tl] = \texttt{TwoSum}(al, bl)$;
$sl = sl \oplus th$;
$th = sh \oplus sl$;
$sl = sl \ominus (th \ominus sh)$;
$tl = tl \oplus sl$;
$[rh, rl] = \texttt{FastTwoSum}(th, tl)$.
```

Algorithm 11 requires 20 flops.

***Algorithm* 12** *[20] Multiplication of double-double number by a double number*

```
$[rh, rl] = \texttt{prod\_dd\_d}(ah, al, b)$;
```

```
    $[th, tl] = \texttt{TwoProd}(ah, b);$
    $tl = al \otimes b \oplus tl;$
    $[rh, rl] = \texttt{FastTwoSum}(th, tl).$
```

Algorithm 12 requires 22 flops.

***Algorithm* 13** *[7, 18] Multiplication of two double-double numbers*

```
    $[rh, rl] = \texttt{prod\_dd\_dd}(ah, al, bh, bl);$
    $[th, tl] = \texttt{TwoProd}(ah, bh);$
    $tl = (ah \otimes bl) \oplus (al \otimes bh) \oplus tl$
    $[rh, rl] = \texttt{FastTwoSum}(th, tl).$
```

Algorithm 13 requires 24 flops.

C De Casteljau Tensor Product Algorithm with Double-Double Library

***Algorithm* 14** *De Casteljau algorithm in double-double format*

```
    $[rh, rl] = \texttt{DDDC}(p1, p2, t);$
    $[\widehat{b1}_i^{(0)}, \widehat{b2}_i^{(0)}] = [b1_i, b2_i],\ i = 0, \cdots, n;$
    $[ch, cl] = \texttt{TwoSum}(1, -t);$
    for $j = 1 : 1 : n$
        for $i = 0 : 1 : n - j$
            $[sh, sl] = \texttt{prod\_dd\_dd}(\widehat{b1}_i^{(j-1)}, \widehat{b2}_i^{(j-1)}, ch, cl);$
            $[xh, xl] = \texttt{prod\_dd\_d}(\widehat{b1}_{i+1}^{(j-1)}, \widehat{b2}_{i+1}^{(j-1)}, t);$
            $[\widehat{b1}_i^{(j)}, \widehat{b2}_i^{(j)}] = \texttt{add\_dd\_dd}(sh, sl, xh, xl);$
        end
    end
    $[rh, rl] = [\widehat{b1}_0^{(n)}, \widehat{b2}_0^{(n)}].$
```

***Algorithm* 15** *De Casteljau tensor product algorithm in double-double format*

```
  function $[rh, rl]$=$\texttt{DDDCTP}(F, x, y)$
  $f_{i,j}^{(0)} = b_{i,j}$ for $0 \le i \le m$ and $0 \le j \le n$
  for $i = 0 : 1 : m$
      $[\widehat{p1}_i, \widehat{p2}_i] = \texttt{DDDC}(f_{i,:}^{(0)}, p0, y)$ % here p0=zeros(n+1,1);
  end
  $[rh, rl] = \texttt{DDDC}(\widehat{p1}, \widehat{p2}, x)$
```

D Exact Coefficients of the Test Polynomial (Eq. (42)) in Bézier tensor product form

$$\{b_{i,j}\} = \begin{pmatrix}
\frac{729}{64000000} & \frac{-3159}{128000000} & \frac{14661}{320000000} & \frac{-84591}{1280000000} & \frac{4887}{80000000} & \frac{-351}{8000000} & \frac{27}{1000000} \\
\frac{-3159}{128000000} & \frac{13689}{256000000} & \frac{-63531}{640000000} & \frac{366561}{2560000000} & \frac{-21177}{160000000} & \frac{1521}{16000000} & \frac{-117}{2000000} \\
\frac{14661}{320000000} & \frac{-63531}{640000000} & \frac{294849}{1600000000} & \frac{-1701219}{6400000000} & \frac{98283}{400000000} & \frac{-7059}{40000000} & \frac{543}{5000000} \\
\frac{-84591}{1280000000} & \frac{366561}{2560000000} & \frac{-1701219}{6400000000} & \frac{9815689}{25600000000} & \frac{-567073}{1600000000} & \frac{40729}{160000000} & \frac{-3133}{20000000} \\
\frac{4887}{80000000} & \frac{-21177}{160000000} & \frac{98283}{400000000} & \frac{-567073}{1600000000} & \frac{32761}{100000000} & \frac{-2353}{10000000} & \frac{181}{1250000} \\
\frac{-351}{8000000} & \frac{1521}{16000000} & \frac{-7059}{40000000} & \frac{40729}{160000000} & \frac{-2353}{10000000} & \frac{169}{1000000} & \frac{-13}{125000} \\
\frac{27}{1000000} & \frac{-117}{2000000} & \frac{543}{5000000} & \frac{-3133}{20000000} & \frac{181}{1250000} & \frac{-13}{125000} & \frac{1}{15625}
\end{pmatrix}$$

References

[1] D. H. Bailey. High-precision software directory: QD library, double-double library. *Lawrence Berkeley National Laboratory*, http://www. nersc. gov/ dhbailey/mpdist/ mpdist. html. , 2008.

[2] J. Berchtold, I. Voiculescu, and A. Bowyer. Multivariate Bernstein form polynomials. Technical report, University of Bath, UK, 2007.

[3] T. J Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18(3):224–242, 1971.

[4] J. Delgado and J. M Peña. Error analysis of efficient evaluation algorithms for tensor product surfaces. *J. Comput. Appl. Math.*, 219(1):156–169, 2008.

[5] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, San Diego, CA, fourth edition, 1997.

[6] S. Graillat. Accurate simple zeros of polynomials in floating point arithmetic. *Comput. Math. Appl.*, 56(4):1114–1120, 2008.

[7] S. Graillat. Accurate floating-point product and exponentiation. *IEEE Trans. Comput.*, 58(7):994–1000, 2009.

[8] S. Graillat, P. Langlois, and N. Louvet. Compensated Horner scheme. Technical report, University of Perpignan, France, 2005.

[9] S. Graillat, P. Langlois, and N. Louvet. Algorithms for accurate, validated and fast polynomial evaluation. *Japan J. Indust. Appl. Math.*, 26(2):191–214, 2009.

[10] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating point arithmetic. In *Proceedings of the 15th IEEE Symposium onComputer Arithmetic*, pages 155–162. IEEE Computer Society, 2001.

[11] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, second edition, 2002.

[12] H. Jiang, R. Barrio, H. S. Li, X. K. Liao, L. Z. Cheng, and F. Su. Accurate evaluation of a polynomial in Chebyshev form. *Appl. Math. Comput.*, 217(23):9702–9716, 2011.

[13] H. Jiang, R. Barrio, X. K. Liao, and L. Z. Cheng. Accurate evaluation algorithm for bivariate polynomial in Bernstein-Bézier form. *Appl. Numer. Math.*, 61:1147–1160, 2011.

[14] H. Jiang, S. G. Li, L. Z Cheng, and F. Su. Accurate evaluation of a polynomial and its derivative in Bernstein form. *Comput. Math. Appl.*, 60(3):744–755, 2010.

[15] D. E. Knuth. *The art of computer programming: Seminumerical algorithms*, volume 2. Addison-Wesley, third edition, 1998.

[16] P. Langlois and N. Louvet. How to ensure a faithful polynomial evaluation with the compensated Horner algorithm. In *Proceedings 18th IEEE Symposium on Computer Arithmetic*, pages 141–149. IEEE Computer Society, 2007.

[17] P. Langlois and N. Louvet. More instruction level parallelism explains the actual efficiency of compensated algorithms. Technical report, DALI Research Team, University of Perpignan, France, 2007.

[18] C. Q. Lauter. Basic building blocks for a triple-double intermediate format. Technical report, INRIA, France, 2005.

[19] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, et al. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Software*, 28(2):152–205, 2002.

[20] N. Louvet. *Compensated algorithms in floating point arithmetic: accuracy, validation, performances.* PhD thesis, Université de Perpignan Via Domitia, 2007.

[21] E. Mainar and J. M Peña. Error analysis of corner cutting algorithms. *Numer. Algorithms.*, 22(1):41–52, 1999.

[22] S. Mann and T. DeRose. Computing values and derivatives of bézier and b-spline tensor products. *Comput. Aided Geom. Des.*, 12(1):107–110, 1995.

[23] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

[24] S. M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part I: Faithful rounding. *SIAM J. Sci. Comput.*, 31(1):189–224, 2008.

[25] S. M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part II: Sign, K-fold faithful and rounding to nearest. *SIAM J. Sci. Comput.*, 31(2):1269–1302, 2008.