Computer Assisted Methods in Engineering and Science, **20**: 271–278, 2013.

TWENTY YEARS OF THE CAMES

A shift-add algorithm for generating B-spline

Feng GU
Zhejiang Technical Institute of Economics, Hangzhou 310018, China
66 Xuezheng St., Xiasha, Hangzhou, Zhejiang, China
e-mail: coolfun630@hotmail.com, coolfun630@163.com

A CORDIC- based shift-add algorithm for generating B-spline curves is presented in this paper. This algorithm can be realized by hardware without multiplier, or coded with assembly language and run in the basic computing system which exists in many application systems. Convergence of the algorithm was proved. Errors were estimated and well controlled in the algorithm. A numerical experiment was carried out to validate algorithm. This algorithm can be used for adding complex curve plotting functions in embedded systems.

Keywords: B-spline, CORDIC, shift-add algorithm, basic computing system.

1. Introduction

In embedded systems there were only simple graphics plotting functions (like straight line drawing function) in their graphics device interfaces. Complex curve plotting functions are helpful for some embedded systems like GIS/GPS handheld device, electronic engraving machines, and electronic game machines, etc. In this paper a coordinate rotation digital computer (CORDIC) – based shift-add algorithm for generating B-spline curves is presented. With this algorithm complex graphics can be plotted in an embedded device.

B-spline curves are very popular in CAD/CAM and other curve fitting systems. For control points $\{P_i\}_0^n$, B-spline function $B(t)$ of order k (degree k-1) can be expressed by the de Boor-Cox recursive procedure,

$$\begin{cases} N_{i,1}(t) = \begin{cases} 1 & t_i \le t < t_{i+1} \\ 0 & \text{Otherwise} \end{cases} \\ N_{i,k}(t) = \dfrac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \dfrac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1} \qquad i=0,1,...,n; \quad k>1, \\ B(t) = \displaystyle\sum_{i=0}^{n} P_i N_{i,k}(t), \\ \text{Here we define } \dfrac{0}{0} = 0, \end{cases} \tag{1}$$

$N_{i,k}(t)$ is a B-spline basis function of order k (degree k-1).

B-spline curves were easily generated in an advanced computing system [1–3, 9]. In this paper we discuss how to generate B-spline curves in a basic computing system. A basic computing system deals only with shift, add and logical operations. Basic computing systems exist in many application systems such as industrial control systems, military application systems, medical application systems, etc.

CORDIC algorithms are well-known shift-add algorithms for computing a wide range of elementary functions including trigonometric, hyperbolic, linear and logarithmic functions [5–7]. Generalization, convergence and error estimation of CORDIC algorithms have been discussed [4, 10]. These fast united shift-add algorithms can be implemented in hardware system without multipliers [8], or be coded with assembly language.

2. Description of the algorithm

A sign function and a positive number series [4] are defined as

$$sg\,(x) = \begin{cases} 1 & x \geq 0, \\ -1 & x < 0, \end{cases}$$

$$\{\delta_i\}_0^\infty = \left\{2^{-i}\right\}_0^\infty .$$

Let $\{P_i = (P_i^x, P_i^y)\}_0^n$ be control points, $N_i^k(t)(i = 0, 1, \ldots, n)$ be B-spline basis functions, ε be the error limit. The shift-add algorithm for generating a B-spline curve consists of one main program and three subprograms. Pseudo codes of the algorithm are shown below.

Subprogram1 MulDiv$(u, v, \varepsilon,$ *flag*$)$.

1° if $v = 0$ then
if flag=0 then result:=0 else result:=false. Exit.
if $u = 0$ then result:=0. Exit.

2° $s := 1$.
if $u < 0$ then $\{s := -s;\ \ u := -u.\}$
if $v < 0$ then $\{s := -s;\ \ v := -v.\}$.

3° $m := 0$;
while $u > 2$ do $\{u := 2^{-1} \times u;\ \ m := m + 1.\}$.

4° $N := 1$; $\varepsilon := 2^{-m-1} \times \varepsilon$;
if flag=0 then $v_0 := v$;
while $v_0 > 1$ do $\{\varepsilon := 2^{-1} \times \varepsilon;\ \ v_0 := 2^{-1} \times v_0;\}$
while $\delta_{N-1} > \varepsilon$ do $N := N + 1$.

5° if flag=0 then $\{i := 1; x_1 := u, y_1 := v, z_1 := 0.\}$
else $\{i := 1; x_1 := 0, y_1 := v, z_1 := u.\}$.

6° while $i < N$ do
{if flag=0 then $s_i := sg(x_i)$ else $s_i := -sg(z_i)$;
$x_{i+1} := x_i + s_i \times \delta_i$; $z_{i+1} := z_i + s_i \times \delta_i \times y_1$; $i := i + 1$;}.

7° if flag=0 then $\{z_N := s \times 2^m \times z_N$; result:=$z_N$.$\}$
else $\{x_N := s \times 2^m \times x_N$; result:=$x_N$.$\}$
Stop.

Subprogram2 B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$.

1° if $(t < t_0$ or $t > t_n)$ then result:=false; Stop.
$i := 0$;
while $t > t_i$ do $i := i + 1$; $i := i - 1$.

2° $N_i^1 := 1$;
for $l := 1$ to $k-1$ do $\{N_{i-l}^l := 0;\ N_{i+1}^l := 0;\}$.

3° if $k = 2$ then $\varepsilon_2 := \varepsilon$;
if $k = 3$ then {if $\varepsilon > 1$ then $\varepsilon := 1$; $\varepsilon_2 := \varepsilon \times 2^{-2}$;}
if $k = 4$ then { if $\varepsilon > 0.24$ then $\varepsilon := 0.24$; $\varepsilon_2 := \varepsilon \times 2^{-3}$;}
$\varepsilon_1 := \varepsilon_2 \times 2^{-2}$.

4° for $m := 2$ to k do
for $l := i - m + 1$ to i do
if ($t_{l+m-1} = t_l$ and $t_{l+m} = t_{l+1}$) then $N_l^m := 0$
else if $t_{l+m-1} = t_l$ then
{$r_2 = \mathrm{MulDiv}(t_{l+m} - t, t_{l+m} - t_{l+1}, \varepsilon_1, 1)$ $N_l^m := \mathrm{MulDiv}(r_2, N_{l+1}^{m-1}, \varepsilon_1, 0)$;}
else if $t_{l+m} = t_{l+1}$ then
{$r_1 = \mathrm{MulDiv}(t - t_l, t_{l+m-1} - t_l, \varepsilon_1, 1)$; $N_l^m := \mathrm{MulDiv}(r_1, N_l^{m-1}, \varepsilon_1, 0)$;}
else {$r_1 = \mathrm{MulDiv}(t - t_l, t_{l+m-1} - t_l, \varepsilon_1, 1)$; $r_2 = \mathrm{MulDiv}(t_{l+m} - t, t_{l+m} - t_{l+1}, \varepsilon_1, 1)$;
$N_l^m := \mathrm{MulDiv}(r_1, N_l^{m-1}, \varepsilon_1, 0) + \mathrm{MulDiv}(r_2, N_{l+1}^{m-1}, \varepsilon_1, 0)$;}.

5° Output $N_j^k, j = i - k + 1, \ldots, i$. Stop.

Subprogram3 B_S($k, t, \{P_j^u\}_0^n, \varepsilon$).

1° $\varepsilon_2 = \varepsilon$;
if $P_{i-k+1}^u < 0$ then $S = -P_{i-k+1}^u$ else $S := P_{i-k+1}^u$;
for $j := i - k + 2$ to i do { if $P_j^u < 0$ then $S = S - P_j^u$ else $S := S + P_j^u$;}
$S := S + k \times 2^{-2}$;
while $S > 1$ do { $S := S \times 2^{-1}$; $\varepsilon_2 := \varepsilon_2 \times 2^{-1}$;}
$\varepsilon_1 := \varepsilon_2 \times 2^{-2}$.

2° B_Basis($k, t, \{P_j^u\}_0^n, \varepsilon_2$);
$B^u := \mathrm{MulDiv}(P_{i-k+1}^u, N_{i-k+1}^k, \varepsilon_1)$;
for $j := i - k + 2$ to i do $B^u := B^u + \mathrm{MulDiv}(P_j^u, N_j^k, \varepsilon_1)$.

3° Output P^u. Stop.

Main Program. B_Spline($k, t, (P_j^x, P_j^y)_0^n, \varepsilon$)

1° $B^x := B\_S(k, t, \{P_j^x\}_0^n, \varepsilon)$.

2° $B^y := B\_S(k, t, \{P_j^y\}_0^n, \varepsilon)$.

3° Output B^x, B^y. Stop.

3. Notes on the algorithm

3.1 Only operations shift (i.e., $2^{-i} \times t$) and add were concerned in the algorithm.
3.2 When flag=0, Subprogram1 MulDiv(u, v, ε, *flag*) multiplies u by v with error limit ε. When flag=1, divides u by v with error limit ε.

$\{\delta_i\}_0^\infty = \{2^{-i}\}_0^\infty$ is a normal series with measurement radius $R(\delta) = \sum_{i=0}^{\infty} \delta_i = 2$ [4]. U decomposes into $2^m \times U_0$ (U_0 is stored in U variable in algorithm) with $U_0 < 2$ in 3° of MulDiv(u, v, ε, *flag*). This ensures $U_0 < R(\delta) = \sum_{i=0}^{\infty} \delta_i = 2$ and so it can be expressed as $U_0 \approx F_N(U_0, \delta) = \sum_{i=0}^{N} sg(u_i)\delta_i$.

4° of MulDiv(u, v, ε, *flag*) adjusts error limit according to 3° and determines the number of iterations N based on Theorem 1 in 4.

5°, 6° and 7° of MulDiv(u, v, ε, *flag*) are from CORDIC algorithm [4, 5].

3.3 Subprogram2 B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$ is for calculating values of B-spline basis functions of order k for t with error limit ε.

1° of B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$ is to find the interval $[t_i, t_{i+1})$ where t is in.

2° of B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$ assigns values for B-spline basis functions $\left\{N_j^1(t)\right\}_{i-1}^{i+1}$, $\left\{N_{i-j}^j(t)\right\}_1^{k-1}$, $\left\{N_{i+1}^j(t)\right\}_1^{k-1}$.

3° of B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$ re-assigns value of error limit (like 0.24 in B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$) based on Theorem 2 in Sec. 4.

4° of B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$ calculates values of B-spline basis functions $N_j^l(t)$ ($l = 2$ to k, $j = i - l + 1$ to i). Other $N_j^l(t)$ are 0. $\dfrac{0}{0}$ is defined as 0.

3.4 Subprogram3 B_S$(k, t, \{P_j^u\}_0^n, \varepsilon)$ is for calculating coordinates of B(t) on the B-spline curve with n+1 control points. Based on Theorem 3, calculation error limit is ε.

3.5. The main operation of the algorithm is run in MulDiv$(u, v, \varepsilon, \mathit{flag})$. 4° of MulDiv$(u, v, \varepsilon, \mathit{flag})$ shows that when calculating error of MulDiv$(u, v, \varepsilon, \mathit{flag})$ is halved, the number of iterations N increases by one. Experience shows that MulDiv$(u, v, \varepsilon, \mathit{flag})$ usually runs not more than 40 steps.

4. Convergence and error estimation of the algorithm

$\{\delta_i\}_0^\infty = \{2^{-i}\}_0^\infty$ is a normal series with measurement radius $R(\delta) = \sum\limits_{i=0}^{\infty} \delta_i = 2$ [4].

5° and 6° of Subprogram1 MulDiv(u, v, ε) in 2 is based on iterative process (1).

Theorem 1. *Let* $\{\delta_i\}_0^{+\infty} = \{2^{-i}\}_0^{+\infty}$, $x \in (-R(\delta), R(\delta)) = (-2, 2)$. *For Subprogram1 MulDiv(u, v, ε, flag) there are,*

(a) in the case of flag=0, $\{z_i\}$ *converges to* $u \times v$, $|z_N - u \times v| < \varepsilon$;

(b) when flag=1, $\{x_i\}$ *converges to* x/y, $|x_N - u/v| < \varepsilon$.

Proof:
(a) From 2° ∼ 4° of MulDiv$(u, v, \varepsilon, \mathit{flag})$, for $u \neq 0$ and $v \neq 0$, there is

$$\delta_{N-1} < \min(|u|^{-1}, 1) \min(|v|^{-1}, 1)\varepsilon.$$

Conclusion can be deduced from theorem 7 of [4].
(b) Same as above, by pre-process in 4° of Subprogram1 MulDiv$(u, v, \varepsilon, \mathit{flag})$ and based on Theorem 8 of [4], result can be obtained.

Theorem 2. *Theorem 2.* N_j^k $(j = i - k + 1, ..., i)$ *in Subprogram2 B_Basis*$(k, t, \{t_j\}_0^n, \varepsilon)$ *are calculated values of B-spline basis functions of order* k *with* $t \in [t_i, t_{i+1}]$. *The calculation error limit is* ε.

Proof:
Let real value of B-spline basis function be $\overline{N}_i^k(t)$, $\varepsilon^{(k)} = \max_{i,t} \left| N_i^k - \overline{N}_i^k(t) \right|$, ε_1 be error limit of MulDiv$(u, v, \varepsilon_1, \mathit{flag})$. From iterative process (1), Theorem 1, and $0 \leq \overline{N}_i^k(t) \leq 1$, there is

$$\begin{aligned}
\left|N_i^k - \overline{N}_i^k(t)\right| &= \Big|\Big[\mathrm{MulDiv}\left(\mathrm{MulDiv}\left(t-t_i, t_{i+k-1}-t_i, \varepsilon_1, 1\right), N_i^{k-1}, \varepsilon_1\right) \\
&\quad + \mathrm{MulDiv}\left(\mathrm{MulDiv}\left(t_{i+k}-t, t_{i+k}-t_{i+1}, \varepsilon_1, 1\right), N_{i+1}^{k-1}, \varepsilon_1\right)\Big] \\
&\quad - \left[\frac{t-t_i}{t_{i+k-1}-t_i}\overline{N}_i^{k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}\overline{N}_{i+1}^{k-1}(t)\right]\Big| \\
&\le \Big|\Big[\mathrm{MulDiv}\left(t-t_i, t_{i+k-1}-t_i, \varepsilon_1, 1\right) N_i^{k-1} + \mathrm{MulDiv}\left(t_{i+k}-t, t_{i+k}-t_{i+1}, \varepsilon_1, 1\right) N_{i+1}^{k-1}\Big] \\
&\quad - \left[\frac{t-t_i}{t_{i+k-1}-t_i}\overline{N}_i^{k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}\overline{N}_{i+1}^{k-1}(t)\right]\Big| + 2\varepsilon_1 \\
&\le \left|\mathrm{MulDiv}\left(t-t_i, t_{i+k-1}-t_i, \varepsilon_1, 1\right)\right| \left|N_i^{k-1} - \overline{N}_i^{k-1}(t)\right| \\
&\quad + \left|\frac{t-t_i}{t_{i+k-1}-t_i} - \mathrm{MulDiv}\left(t-t_i, t_{i+k-1}-t_i, \varepsilon_1, 1\right)\right| \overline{N}_i^{k-1}(t) \\
&\quad + \left|\mathrm{MulDiv}\left(t_{i+k}-t, t_{i+k}-t_{i+1}, \varepsilon_1, 1\right)\right| \left|N_{i+1}^{k-1}\Big] - \overline{N}_{i+1}^{k-1}(t)\right| \\
&\quad + \left|\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} - UV\left(t_{i+k}-t, t_{i+k}-t_{i+1}, \varepsilon_1, 1\right)\right| \left|\overline{N}_{i+1}^{k-1}(t)\right| + 2\varepsilon_1 \\
&\le \left(\left|\frac{t-t_i}{t_{i+k-1}-t_i}\right| + \varepsilon_1\right)\varepsilon^{(k-1)} + \left(\left|\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}\right| + \varepsilon_1\right)\varepsilon^{(k-1)} + 4\varepsilon_1 \le 2(1+\varepsilon_1)\varepsilon^{(k-1)} + 4\varepsilon_1,
\end{aligned}$$

or $\varepsilon^{(k)} \le 2(1+\varepsilon_1)\varepsilon^{(k-1)} + 4\varepsilon_1$.

Note that $\varepsilon^{(1)} = 0$, $\varepsilon_1 \le \dfrac{\varepsilon}{4}$, $\varepsilon^{(2)} \le 4\varepsilon_1 \le \varepsilon$ and

$$\begin{aligned}
\varepsilon^{(k)} &\le \left(2+\frac{\varepsilon}{2}\right)\varepsilon^{(k-1)} + \varepsilon \le \ldots \le \left(2+\frac{\varepsilon}{2}\right)^{k-2}\varepsilon^{(2)} + \left[\left(2+\frac{\varepsilon}{2}\right)^{k-2} - 1\right]\varepsilon \\
&\le \left(2+\frac{\varepsilon}{2}\right)^{k-2}\varepsilon + \left[\left(2+\frac{\varepsilon}{2}\right)^{k-2} - 1\right]\varepsilon = \left[2\left(2+\frac{\varepsilon}{2}\right)^{k-2} - 1\right]\varepsilon.
\end{aligned}$$

When $k = 3$ and $\varepsilon \le 2\left(\dfrac{5}{2} - 2\right) = 1$, $\varepsilon^{(3)} \le 4\varepsilon$.

When $k = 4$ and $\varepsilon \le 2\left(\sqrt{\dfrac{9}{2}} - 2\right) = 0.2426$, $\varepsilon^{(4)} \le 8\varepsilon$.

3° of B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$ pre-controls error according to discussion above. $k > 4$ are seldom used. Result is proved.

Theorem 3. *The calculation error limit of B^u in B_S$(k, t, \{P^u(t_j)\}_0^n, \varepsilon)$ is ε.*

Proof:
Let true value of B-spline function be $\overline{B}^u(t)$, true value of B-spline basis function be $\overline{N}_i^k(t)$, ε_1 be error limit of MulDiv(u, v, ε_1, *flag*), ε_2 be error limit of B_Basis$(k, t, \{t_j\}_0^n, \varepsilon_2)$. There is

$$\begin{aligned}
\left|B^u - \overline{B}^u(t)\right| &= \left|\sum_{j=i-k+1}^{i}\left[\mathrm{MulDiv}(P^u(t_j), N_j^k, \varepsilon_1) - P^u(t_j) \times \overline{N}_j^k(t)\right]\right| \\
&\le \left|\sum_{j=i-k+1}^{i}\left[P^u(t_j)\left(N_j^k - \overline{N}_j^k(t)\right)\right]\right| + k\varepsilon_1 \le \left(\sum_{j=i-k+1}^{i}\left|P^u(t_j)\right|\right)\varepsilon_2 + k\varepsilon_1.
\end{aligned}$$

Referring to Subprogram2 B_Basis$(k, t, \{t_j\}_0^n, \varepsilon)$, we can set $\varepsilon_1 = 2^{-2}\varepsilon_2$. There is

$$\left|B^u - \overline{B}^u(t)\right| \leq \left(\sum_{j=i-k+1}^{i} |P^u(t_j)| + 2^{-2}k\right)\varepsilon_2.$$

1° of B_S$(k, t, \{P^u(t_j)\}_0^n, \varepsilon)$ guaranteed precision.

5. The numerical experiment

For 10 control points (0, 0.5), (1, 1), (2, 1.5), (3, 2), (4, 2.5), (5, 2.5), (6, 2), (7, 1.5), (8, 1), (9, 0.5), where $t_i = x_i$, and error limit $\varepsilon = 5 \times 10^{-8}$, $t = 2.8$, the algorithm was used to generate a B-spline curve. Results are shown below. The number of iterations in Subprogram1 MulDiv(u, v, ε) was not more than 40 steps.

Table 1. Calculation of B-basis functions.

B-spline basis function	Calculated value for $t = 2.8$ with the algorithm	True value for $t = 2.8$	Error
$N_2^1(t)$	1	1	0
$N_1^2(t)$	0.199999999953	0.2	-4.7×10^{-11}
$N_2^2(t)$	0.800000000047	0.8	4.7×10^{-11}
$N_0^3(t)$	0.019999999967	0.02	-3.3×10^{-11}
$N_1^3(t)$	0.659999999902	0.66	-9.8×10^{-11}
$N_2^3(t)$	0.32000000013	0.32	1.3×10^{-10}
$N_j^i(t)$ for other $i \in \{1, 2, 3\}$ & $j \in \{0, ..., 9\}$	0	0	0

Coordinates of 8 points on the B-spline curve were calculated and shown below.

Table 2. 8 points on the B-spline curve.

t	Calculated value of $B_3^x(t)$	True value of $\overline{B}_3^x(t)$	Error
2.2	0.7000000032	0.7	3.2×10^{-9}
2.8	1.3000000011	1.3	1.1×10^{-9}
3.4	1.90000000095	1.9	9.5×10^{-10}
4.0	2.49999999959	2.5	4.1×10^{-10}
4.6	3.10000000038	3.1	3.8×10^{-10}
5.2	3.70000000027	3.7	2.7×10^{-10}
5.8	4.30000000019	4.3	1.9×10^{-10}
6.4	4.90000000023	4.9	2.3×10^{-10}
t	Calculated value of $B_3^y(t)$	True value of $\overline{B}_3^y(t)$	Error
2.2	0.85000000085	0.85	8.5×10^{-10}
2.8	1.1500000010	1.15	1.0×10^{-9}
3.4	1.33000000093	1.33	9.3×10^{-10}
4.0	1.00000000023	1	2.3×10^{-10}
4.6	0.40000000062	0.4	6.2×10^{-10}
5.2	-0.20000000017	-0.2	-1.7×10^{-10}
5.8	-0.8000000011	-0.8	-1.1×10^{-9}
6.4	-1.32000000095	-1.32	-9.5×10^{-10}

It showed that calculation errors were under control. Data points of the B-spline curve are shown below. Black points are control points. White points are points on B-spline curve generated by the algorithm with $t = 2.2, 2.8, 3.4, 4.0, 4.6, 5.2, 5.8, 6.4$.

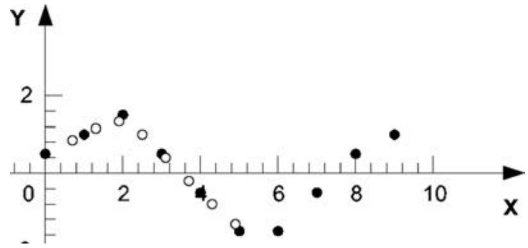

Fig. 1. Data points of the B-spline curve.

6. Conclusion

In this paper a CORDIC based shift-add algorithm for generating B-spline curves is presented. It can be realized by hardware without multiplier, or coded with assembly language and run in the basic computing system which exists in many application systems.

The algorithm is composed of three subprograms and a main program. Convergence of the algorithm was proved. Errors of every subprogram and the main program were estimated. The error of the algorithm is well controlled. Main iteration process in the algorithm usually runs no more than 40 steps. A numerical experiment was presented to validate the algorithm. This is an effective and efficient algorithm and can be used for complex curve plotting in an embedded system.

Appendix

Definition in [4]. A normal series $\{\delta_i\}_0^\infty$ is a positive number series which decreases monotonically and has a finite sum $R(\delta) = \sum\limits_{i=0}^{\infty} \delta_i$. For all $x \in \{x : |x| \le R(\delta)\}$ there is $|x - F_n(x,\delta)| \le \delta_n$, where $F_n(x,\delta)$ is defined as

$$x_0 = x, \qquad x_{i+1} = x_i - \mathrm{sgn}(x_i) \times \delta_i, \qquad i = 0, 1, \ldots$$

$$F_n(x,\delta) = \sum_{i=0}^{n} \mathrm{sgn}(x_i) \times \delta_i.$$

$\left\{2^{-i}\right\}_0^\infty$ is a normal series.

Lemma 1 of [4]. For a normal series $\{\delta_i\}_0^\infty$ there is $\delta_{n+1} < \sum\limits_{i=n+1}^{\infty} \delta_i \le \delta_n$.

Proof. Let $x = \sum\limits_{i=0}^{\infty} \delta_i$. There is $F_n(x,\delta) = \sum\limits_{i=0}^{n} \delta_i$. From the definition of the normal series there is

$$|x - F_n(x,\delta)| = \sum_{i=n+1}^{\infty} \delta_i \le \delta_n.$$

Left in equation is apparent. Lemma is proved.

Define an iterative process as

$$\begin{cases} x_1 = 0, \quad y_1 = y, \quad z_1 = 0, \\ s_i = \mathrm{sgn}\left(x_i\right), \\ x_{i+1} = x_i - s_i \times 2^{-i}, \\ z_{i+1} = z_i + s_i \times 2^{-i} \times y_1, \quad i = 1, 2, \ldots \end{cases} \tag{A1}$$

$$x_{n+1} \approx 0, \qquad z_{n+1} \approx x \times y.$$

Theorem 7 of [4]. $\{z_i\}$ defined in iterative process (A1) converges to $x \times y$, and there is $|z_{n+1} - x \times y| \leq 2^{-n} |y|$.

Proof. From iterative process (A1) there is $z_{i+k} - z_i \leq y \sum_{j=i}^{i+k-1} 2^{-j} s_j$.

From Lemma 1 there is $|z_{i+k} - z_i| \leq |y| \sum_{j=i}^{i+k-1} 2^{-j} \leq |y| \sum_{j=i}^{\infty} 2^{-j} \leq |y| 2^{-i+1}$. $\{z_i\}$ is a Cauchy series. The design of iterative process (A1) makes $z_i \to x \times y$. Let $i = n+1$ and $k \to \infty$ there is $|z_{n+1} - x \times y| \leq 2^{-n} |y|$.

Define another iterative process as

$$\begin{cases} x_1 = 0, \quad y_1 = x, \quad z_1 = -y, \\ s_i = -\operatorname{sgn}(z_i), \\ x_{i+1} = x_i + s_i \times 2^{-i}, \\ z_{i+1} = z_i + s_i \times 2^{-i} \times y_1, \quad i = 1, 2, \ldots \end{cases} \tag{A2}$$

$$x_{n+1} \approx y/x, \qquad z_{n+1} \approx 0.$$

Theorem 8 of [4]. $\{x_i\}$ defined in iterative process (A2) converges to y/x, and there is

$$|x_{n+1} - y/x| \leq 2^{-n}.$$

Proof. Similar as proof of Theorem 7.

Acknowledgements

I would like to acknowledge with appreciation the support received. This work was supported by Scientific Research Fund of Zhejiang Provincial Education Department [Grant Number Y201223296], China.

References

[1] G. Aumann. Corner cutting curves and a new characterization of Bézier and B-spline curves. *Computer Aided Geometric Design – CAGD*, **14**(5): 449–474, 1997.

[2] C. de Boor. *Splines as linear combinations of B-splines, Approximation Theory II*, G.G. Lorentz, C.K. Chui and L.L. Schumaker [Eds.], pp. 1–47, Academic Press, New York, 1976.

[3] C. de Boor, K. Höllig. *B-splines without divided differences, in Geometric Modeling*, G. Farin [Ed.], SIAM, pp. 21–27, 1987.

[4] GU Feng. Convergence and Error Estimation of Coordinate Rotating Algorithm and Its Expansion. *Chinese Journal of Numerical Mathematics and Applications*, **28**(2): 1–9, 1987.

[5] J.E. Volder. The CORDIC Computing Technique, *IRE Transactions on Electronic Computers*, EC-**8**(9): 330–334, 1959.

[6] J.M. Muller. *Elementary Functions, Algorithms and Implementation*, Birkhauser Boston, 1st edition, 1997. 2nd edition, 2006, pp. 133–156.

[7] N. Eklund. CORDIC: Elementary Function Computation Using Recursive Sequences. *International Conference on Technology in Collegiate Mathematics (ICTCM)*, **11**, 1998.

[8] R. Andraka. A Survey of CORDIC Algorithms for FPGA Based Computers. *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 191–200, 1998.

[9] W. Bohm, G. Farin, J. Kahmann. A survey of curve and surface methods in CAGD, *Computer Aided Geometric Design*, **1**: 1–60, 1985.

[10] X. Hu, R. Harber, S. Bass. Expanding the Range of Convergence of the CORDIC Algorithm. *IEEE Transactions on Computers*, **40**: 13–21, 1991.