Robust Adaptive Floating-Point Geometric Predicates

Jonathan Richard Shewchuk
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
jrs@cs.cmu.edu

Expansion

- **Sorted** sequence of non-overlapping machine native numbers (float, double)
- Sorted by **absolute values**
- Signum of the highest FP number is the signum of the expansion
- Estimate the expansion by summing from the least significant to the most significant member
- Zero members of the expansion will be deleted.

Expansions not unique

$1100 + -10.1$

$(= 1100.0 - 10.1)$

$= 1001 + 0.1$

$= 1000 + 1 + 0.1$

Operations on expansions

IEEE 754 standard on floating point format and computing rules.

Operations on expansions require rounding of each operation to 32 / 64bit.

Fast-Two-Sum: (a>=b) -> (x, y), a+b=x+y

Two-Sum (a, b) -> (x, y)

Linear-Expansion-Sum (exp_a interleaved with exp_b) -> expansion

Split (a) -> (a_hi, a_lo), a=a_hi+a_lo

Two-Product (a,b) -> (x, y)

Theorem 1 (Dekker [4]) *Let a and b be p-bit floating-point numbers such that $\boxed{|a| \geq |b|}$. Then the following algorithm will produce a nonoverlapping expansion $x + y$ such that $a + b = x + y$, where $\boxed{x}$ is an approximation to $a + b$ and $\boxed{y}$ represents the roundoff error in the calculation of x.* $\blacksquare$

<u>FAST-TWO-SUM(a, b)</u>

$x \Leftarrow a \oplus b$ // Rounded sum = approximation

$b_{\text{virtual}} \Leftarrow x \ominus a$ // What was truly added - Rounded

$y \Leftarrow b \ominus b_{\text{virtual}}$ // round-off error

return (x, y)

Theorem 2 (Knuth [10]) *Let a and b be p-bit floating-point numbers, where $p \geq 3$. Then the following algorithm will produce a nonoverlapping expansion $x + y$ such that $a + b = x + y$.* ■

```
TWO-SUM(a, b)
x ⇐ a ⊕ b                                  // Rounded sum = approximation
b_virtual ⇐ x ⊖ a                           // What b was truly added - Rounded
a_virtual ⇐ x ⊖ b_virtual                   // What a was truly added - Rounded
b_roundoff ⇐ b ⊖ b_virtual                  // round-off error of b
a_roundoff ⇐ a ⊖ a_virtual                  // round-off error of a
y ⇐ a_roundoff ⊕ b_roundoff
return (x, y)
```

Sum of two expansions (4-bit arithmetic)

Input: 1111+0.1001 and 1100 + 0.1

Output: 11100 + 0 + 0.0001

Zeroes slow down the computation

Merge both input expansions into a single sequence *g* respecting the order of magnitudes

1111+ 1100 + 0.1 + 0.1001

Use LINEAR-EXPANSION-SUM (*g*)

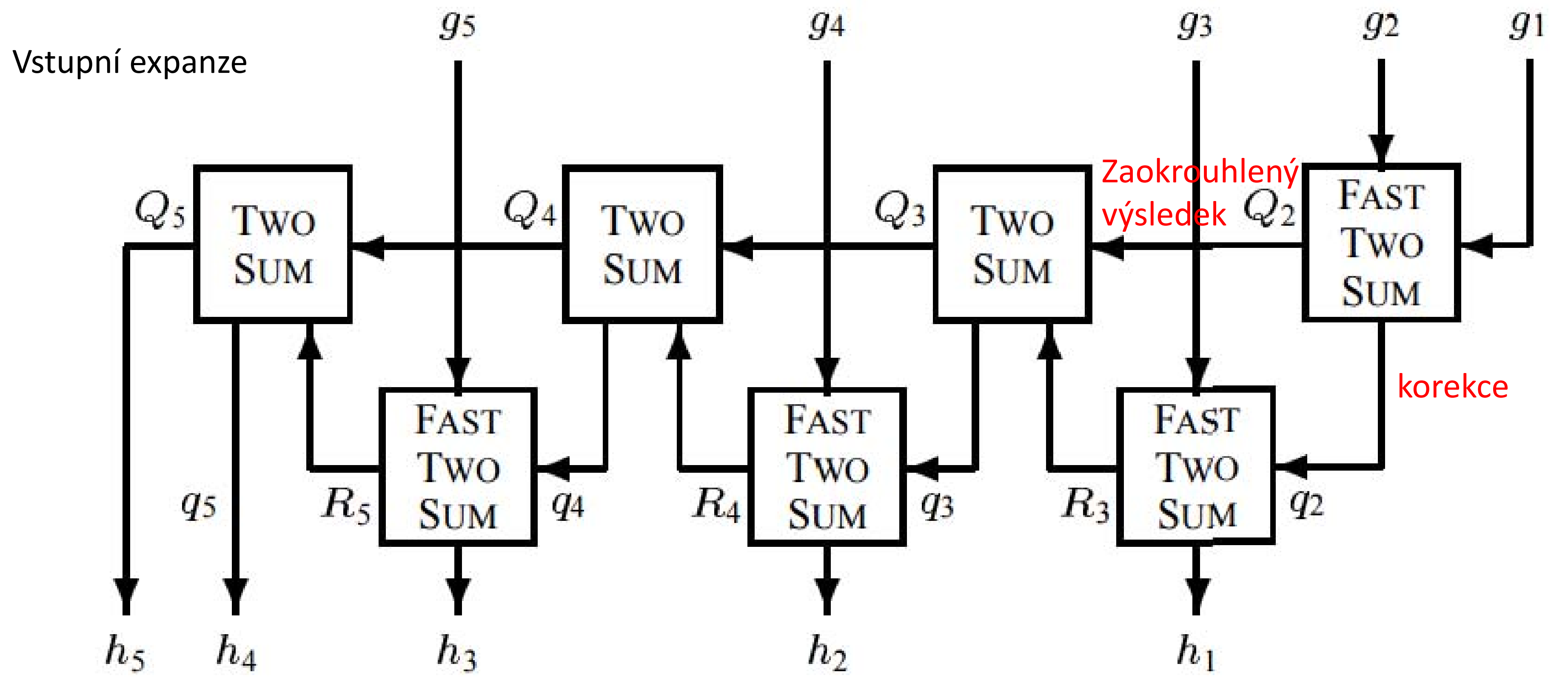


Figure 1: Operation of LINEAR-EXPANSION-SUM. The expansions g and h are illustrated with their most significant components on the left. $Q_i + q_i$ maintains an approximate running total. The FAST-TWO-SUM operations in the bottom row exist to clip a high-order bit off each q_i term, if necessary, before outputting it.

Theorem 4 (Dekker [4]) *Let a be a p-bit floating-point number, where $p \geq 3$. The following algorithm will produce a $\lfloor \frac{p}{2} \rfloor$-bit value a_{hi} and a nonoverlapping $(\lceil \frac{p}{2} \rceil - 1)$-bit value a_{lo} such that $|a_{\mathrm{hi}}| \geq |a_{\mathrm{lo}}|$ and $a = a_{\mathrm{hi}} + a_{\mathrm{lo}}$.* ∎

$\boxed{\text{SPLIT}(a)}$

$c \Leftarrow (2^{\lceil p/2 \rceil} + 1) \otimes a$

$a_{\mathrm{big}} \Leftarrow c \ominus a$

$a_{\mathrm{hi}} \Leftarrow c \ominus a_{\mathrm{big}}$

$a_{\mathrm{lo}} \Leftarrow a \ominus a_{\mathrm{hi}}$

return $(a_{\mathrm{hi}}, a_{\mathrm{lo}})$

Theorem 5 (Veltkamp) *Let a and b be p-bit floating-point numbers, where $p \geq 4$. The following algorithm will produce a nonoverlapping expansion $x + y$ such that $ab = x + y$.* ■

$\textsc{Two-Product}(a, b)$

$x \Leftarrow a \otimes b$

$(a_{\mathrm{hi}}, a_{\mathrm{lo}}) = \textsc{Split}(a)$

$(b_{\mathrm{hi}}, b_{\mathrm{lo}}) = \textsc{Split}(b)$

$err_1 \Leftarrow x \ominus (a_{\mathrm{hi}} \otimes b_{\mathrm{hi}})$

$err_2 \Leftarrow err_1 \ominus (a_{\mathrm{lo}} \otimes b_{\mathrm{hi}})$

$err_3 \Leftarrow err_2 \ominus (a_{\mathrm{hi}} \otimes b_{\mathrm{lo}})$

$y \Leftarrow (a_{\mathrm{lo}} \otimes b_{\mathrm{lo}}) \ominus err_3$

return (x, y)

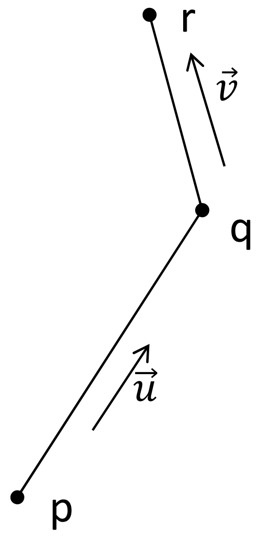
Orientation predicate - definition

$$\text{orientation}(p,q,r) = \text{sign}\left(\det\begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix}\right) =$$

$$= \text{sign}\Big((q_x - p_x)\big(r_y - p_y\big) - \big(q_y - p_y\big)(r_x - p_x)\Big),$$

$$\text{where point } p = \big(p_x, p_y\big), \ldots$$

$$= \text{third coordinate of} = (\vec{u} \times \vec{v}),$$



Three points — $\text{orientation}(p,q,r) =$

- lie on common line — = 0
- form a left turn — = +1 (positive)
- form a right turn — = –1 (negative)

Experiment with orientation predicate

- orientation(p,q,r) = sign($(p_x-r_x)(q_y-r_y)-(p_y-r_y)(q_x-r_x)$)

Ideal return values