computation

MDPI

Article

A Modification of the Fast Inverse Square Root Algorithm

Cezary J. Walczyk [1], Leonid V. Moroz [2] and Jan L. Cieśliński [1,*]

[1] Wydział Fizyki, Uniwersytet w Białymstoku, ul. Ciołkowskiego 1L, 15-245 Białystok, Poland
[2] Department of Security Information and Technology, Lviv Polytechnic National University, st. Kn. Romana 1/3, 79000 Lviv, Ukraine
* Correspondence: j.cieslinski@uwb.edu.pl

Received: 28 June 2019; Accepted: 14 August 2019; Published: 18 August 2019

Abstract: We present a new algorithm for the approximate evaluation of the inverse square root for single-precision floating-point numbers. This is a modification of the famous fast inverse square root code. We use the same "magic constant" to compute the seed solution, but then, we apply Newton–Raphson corrections with modified coefficients. As compared to the original fast inverse square root code, the new algorithm is two-times more accurate in the case of one Newton–Raphson correction and almost seven-times more accurate in the case of two corrections. We discuss relative errors within our analytical approach and perform numerical tests of our algorithm for all numbers of the type **float**.

Keywords: floating-point arithmetic; inverse square root; magic constant; Newton–Raphson method

1. Introduction

Floating-point arithmetic has became widely used in many applications such as 3D graphics, scientific computing and signal processing [1–5], implemented both in hardware and software [6–10]. Many algorithms can be used to approximate elementary functions [1,2,10–18]. The inverse square root function ($x \to 1/\sqrt{x}$) is of particular importance because it is widely used in 3D computer graphics, especially in lightning reflections [19–21], and has many other applications; see [22–36]. All of these algorithms require an initial seed to start the approximation. The more accurate is the initial seed, the fewer iterations are needed. Usually, the initial seed is obtained from a look-up table (LUT), which is memory consuming.

In this paper, we consider an algorithm for computing the inverse square root using the so-called magic constant instead of an LUT [37–40]. The zeroth approximation (initial seed) for the inverse square root of a given floating-point number is obtained by a logical right shift by one bit and subtracting this result from an specially-chosen integer ("magic constant"). Both operations are performed on bits of the floating-point number interpreted as an integer. Then, a more accurate value is produced by a certain number (usually one or two) of standard Newton–Raphson iterations. The following code realizes the fast inverse square root algorithm in the case of single-precision IEEE Standard 754 floating-point numbers (type **float**).

The code *InvSqrt* (see Algorithm 1) consists of two main parts. Lines *4* and *5* produce in a very inexpensive way a quite good zeroth approximation of the inverse square root of a given positive floating-point number x. Lines *6* and *7* apply the Newton–Raphson corrections twice (often, a version with just one iteration is used, as well). Originally, R was proposed as $0x5F3759DF$; see [37,38]. More details, together with a derivation of a better magic constant, are given in Section 2.

Algorithm 1: *InvSqrt*.

```
1.  float InvSqrt(float x){
2.      float halfnumber = 0.5f*x;
3.      int i = *(int*) &x;
4.      i = R - (i>>1);
5.      y = *(float*) &i;
6.      y = y*(1.5f - halfnumber*y*y);
7.      y = y*(1.5f - halfnumber*y*y);
8.      return y ;
9.  }
```

InvSqrt is characterized by a high speed, more that three-times higher than computing the inverse square root using library functions. This property was discussed in detail in [41]. The errors of the fast inverse square root algorithm depend on the choice of the "magic constant" R. In several theoretical papers [38,41–44] (see also Eberly's monograph [19]), attempts were made to determine analytically the optimal value of the magic constant (i.e., to minimize errors). In general, this optimal value can depend on the number of iterations, which is a general phenomenon [45]. The derivation and comprehensive mathematical description of all the steps of the fast inverse square root algorithm were given in our recent paper [46]. We found the optimum value of the magic constant by minimizing the final maximum relative error.

In the present paper, we develop our analytical approach to construct an improved algorithm (*InvSqrt1*) for fast computing of the inverse square root; see Algorithm 2 in Section 4. The proposed modification does not increase the speed of data processing, but increases, in a significant way, the accuracy of the output. In both codes, *InvSqrt* and *InvSqrt1*, magic constants serve as a low-cost way of generating a reasonably accurate first approximation of the inverse square root. These magic constants turn out to be the same. The main novelty of the new algorithm is in the second part of the code, which is changed significantly. In fact, we propose a modification of the Newton–Raphson formulae, which has a similar computational cost, but improve the accuracy by several fold.

2. Analytical Approach to the Algorithm *InvSqrt*

In this paper, we confine ourselves to positive single-precision floating-point numbers (type float). Normal floating-point numbers can be represented as:

$$x = (1 + m_x)2^{e_x} \tag{1}$$

where $m_x \in [0, 1)$ and e_x is an integer (note that this formula does not hold for subnormal numbers). In the case of the IEEE-754 standard, a floating-point number is encoded by 32 bits. The first bit corresponds to a sign (in our case, this bit is simply equal to zero); the next eight bits correspond to an exponent e_x; and the last 23 bits encode a mantissa m_x. The integer encoded by these 32 bits, denoted by I_x, is given by:

$$I_x = N_m(B + e_x + m_x) \tag{2}$$

where $N_m = 2^{23}$ and $B = 127$ (thus $B + e_x = 1, 2, \ldots, 254$). Lines *3* and *5* of the *InvSqrt* code interpret a number as an integer (2) or float (1), respectively. Lines *4*, *6*, and *7* of the code can be written as:

$$I_{y_0} = R - \lfloor I_x/2 \rfloor, \qquad y_1 = \tfrac{1}{2}y_0(3 - y_0^2 x), \qquad y_2 = \tfrac{1}{2}y_1(3 - y_1^2 x). \tag{3}$$

The first equation produces, in a surprisingly simple way, a good zeroth approximation y_0 of the inverse square root $y = 1/\sqrt{x}$. Of course, this needs a very special form of R. In particular, in the single precision case, we have $e_R = 63$; see [46]. The next equations can be easily recognized as the

Newton–Raphson corrections. We point out that the code *InvSqrt* is invariant with respect to the scaling:

$$x \to \tilde{x} = 2^{-2n}x, \qquad y_k \to \tilde{y}_k = 2^n y_k \qquad (k = 0, 1, 2), \tag{4}$$

like the equality $y = 1/\sqrt{x}$ itself. Therefore, without loss of the generality, we can confine our analysis to the interval:

$$\tilde{A} := [1, 4). \tag{5}$$

The tilde will denote quantities defined on this interval. In [46], we showed that the function $\tilde{y}_0$ defined by the first equation of (3) can be approximated with a very good accuracy by the piece-wise linear function $\tilde{y}_{00}$ given by:

$$\tilde{y}_{00}(\tilde{x}, t) = \begin{cases} -\frac{1}{4}\tilde{x} + \frac{3}{4} + \frac{1}{8}t & \text{for } \tilde{x} \in [1, 2) \\ -\frac{1}{8}\tilde{x} + \frac{1}{2} + \frac{1}{8}t & \text{for } \tilde{x} \in [2, t) \\ -\frac{1}{16}\tilde{x} + \frac{1}{2} + \frac{1}{16}t & \text{for } \tilde{x} \in [t, 4) \end{cases} \tag{6}$$

where:

$$t = 2 + 4m_R + 2N_m^{-1}, \tag{7}$$

and $m_R := N_m^{-1}R - \lfloor N_m^{-1}R \rfloor$ (m_R is the mantissa of the floating-point number corresponding to R). Note that the parameter t, defined by (7), is uniquely determined by R.

The only difference between y_0 produced by the code *InvSqrt* and y_{00} given by (6) is the definition of t, because t related to the code depends (although in a negligible way) on x. Namely,

$$|\tilde{y}_{00} - \tilde{y}_0| \leqslant \frac{1}{4}N_m^{-1} = 2^{-25} \approx 2.98 \times 10^{-8}. \tag{8}$$

Taking into account the invariance (4), we obtain:

$$\left|\frac{y_{00} - y_0}{y_0}\right| \leqslant 2^{-24} \approx 5.96 \times 10^{-8}. \tag{9}$$

These estimates do not depend on t (in other words, they do not depend on R). The relative error of the zeroth approximation (6) is given by:

$$\tilde{\delta}_0(\tilde{x}, t) = \sqrt{\tilde{x}}\,\tilde{y}_{00}(\tilde{x}, t) - 1 \tag{10}$$

This is a continuous function with local maxima at:

$$\tilde{x}_0^I = (6 + t)/6, \qquad \tilde{x}_0^{II} = (4 + t)/3, \qquad \tilde{x}_0^{III} = (8 + t)/3, \tag{11}$$

given respectively by:

$$\begin{aligned} \tilde{\delta}_0(\tilde{x}_0^I, t) &= -1 + \frac{1}{2}\left(1 + \frac{t}{6}\right)^{3/2}, \\ \tilde{\delta}_0(\tilde{x}_0^{II}, t) &= -1 + 2\left(\frac{1}{3}\left(1 + \frac{t}{4}\right)\right)^{3/2}, \\ \tilde{\delta}_0(\tilde{x}_0^{III}, t) &= -1 + \left(\frac{2}{3}\left(1 + \frac{t}{8}\right)\right)^{3/2}. \end{aligned} \tag{12}$$

In order to study the global extrema of $\tilde{\delta}_0(\tilde{x},t)$, we need also boundary values:

$$\tilde{\delta}_0(1,t)=\tilde{\delta}_0(4,t)=\frac{1}{8}\,(t-4)\,,\qquad \tilde{\delta}_0(2,t)=\frac{\sqrt{2}}{4}\left(1+\frac{t}{2}\right)-1,\qquad \tilde{\delta}_0(t,t)=\frac{\sqrt{t}}{2}-1, \tag{13}$$

which are, in fact, local minima. Taking into account:

$$\tilde{\delta}_0(1,t)-\tilde{\delta}_0(t,t)=\frac{1}{8}\left(\sqrt{t}-2\right)^2\geqslant 0\,,\qquad \tilde{\delta}_0(2,t)-\tilde{\delta}_0(t,t)=\frac{\sqrt{2}}{8}\left(\sqrt{t}-\sqrt{2}\right)^2\geqslant 0, \tag{14}$$

we conclude that:

$$\min_{\tilde{x}\in\tilde{A}}\tilde{\delta}_0(\tilde{x},t)=\tilde{\delta}_0(t,t)<0. \tag{15}$$

Because $\tilde{\delta}_0(\tilde{x}_0^{III},t)<0$ for $t\in(2,4)$, the global maximum is one of the remaining local maxima:

$$\max_{\tilde{x}\in\tilde{A}}\tilde{\delta}_0(\tilde{x},t)=\max\{\tilde{\delta}_0(\tilde{x}_0^{I},t),\tilde{\delta}_0(\tilde{x}_0^{II},t)\}. \tag{16}$$

Therefore,

$$\max_{x\in\tilde{A}}|\tilde{\delta}_0(\tilde{x},t)|=\max\{|\tilde{\delta}_0(t,t)|,\tilde{\delta}_0(\tilde{x}_0^{I},t),\tilde{\delta}_0(\tilde{x}_0^{II},t)\}. \tag{17}$$

In order to minimize this value with respect to t, i.e., to find t_0^r such that:

$$\max_{x\in\tilde{A}}|\tilde{\delta}_0(\tilde{x},t_0^r)|<\max_{x\in\tilde{A}}|\tilde{\delta}_0(\tilde{x},t)|\qquad\text{for }\ t\neq t_0^r, \tag{18}$$

we observe that $|\tilde{\delta}_0(t,t)|$ is a decreasing function of t, while both maxima ($\tilde{\delta}_0(\tilde{x}_0^{I},t)$ and $\tilde{\delta}_0(\tilde{x}_0^{II},t)$) are increasing functions. Therefore, it is sufficient to find $t=t_0^{I}$ and $t=t_0^{II}$ such that:

$$|\tilde{\delta}_0(t_0^{I},t_0^{I})|=\tilde{\delta}_0(\tilde{x}_0^{I},t_0^{I})\,,\qquad |\tilde{\delta}_0(t_0^{II},t_0^{II})|=\tilde{\delta}_0(\tilde{x}_0^{II},t_0^{II}), \tag{19}$$

and to choose the greater of these two values. In [46], we showed that:

$$|\tilde{\delta}_0(t_0^{I},t_0^{I})|<|\tilde{\delta}_0(t_0^{II},t_0^{II})|. \tag{20}$$

Therefore, $t_0^r=t_0^{II}$, and:

$$\tilde{\delta}_{0\,\max}:=\min_{t\in(2,4)}\left(\max_{x\in\tilde{A}}|\tilde{\delta}_0(\tilde{x},t)|\right)=|\tilde{\delta}_0(t_0^r,t_0^r)|. \tag{21}$$

The following numerical values result from these calculations [46]:

$$t_0^r\approx 3.7309796,\qquad R_0=0x5F37642F,\qquad \tilde{\delta}_{0\,\max}\approx 0.03421281. \tag{22}$$

Newton–Raphson corrections for the zeroth approximation ($\tilde{y}_{00}$) will be denoted by $\tilde{y}_{0k}$ ($k=1,2,\ldots$). In particular, we have:

$$\begin{aligned}\tilde{y}_{01}(\tilde{x},t)&=\tfrac{1}{2}\tilde{y}_{00}(\tilde{x},t)(3-\tilde{y}_{00}^2(\tilde{x},t)\,\tilde{x}),\\ \tilde{y}_{02}(\tilde{x},t)&=\tfrac{1}{2}\tilde{y}_{01}(\tilde{x},t)(3-\tilde{y}_{01}^2(\tilde{x},t)\,\tilde{x}).\end{aligned} \tag{23}$$

and the corresponding relative error functions will be denoted by $\tilde{\delta}_k(\tilde{x},t)$:

$$\tilde{\delta}_k(\tilde{x},t):=\frac{\tilde{y}_{0k}(\tilde{x},t)-\tilde{y}}{\tilde{y}}=\sqrt{\tilde{x}}\tilde{y}_{0k}(\tilde{x},t)-1,\qquad (k=0,1,2,\ldots), \tag{24}$$

where we included also the case $k=0$; see (10). The obtained approximations of the inverse square root depend on the parameter t directly related to the magic constant R. The value of this parameter can be

estimated by analyzing the relative error of $\tilde{y}_{0k}(\tilde{x},t)$ with respect to $1/\sqrt{\tilde{x}}$. As the best estimation, we consider $t = t_k^{(r)}$ minimizing the relative error $\tilde{\delta}_k(\tilde{x},t)$:

$$\forall_{t\neq t_k^{(r)}} \left(\tilde{\delta}_{k\,\max} \equiv \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_k(\tilde{x},t_k^{(r)})| < \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_k(\tilde{x},t)| \right). \tag{25}$$

We point out that in general, the optimum value of the magic constant can depend on the number of Newton–Raphson corrections. Calculations carried out in [46] gave the following results:

$$\begin{aligned} &t_1^r = t_2^r = 3.7298003, \qquad R_1^r = R_2^r = 0x5F375A86, \\ &\tilde{\delta}_{1\,\max} \approx 1.75118 \times 10^{-3}, \quad \tilde{\delta}_{2\,\max} \approx 4.60 \times 10^{-6}. \end{aligned} \tag{26}$$

We omit the details of the computations except one important point. Using (24) to express $\tilde{y}_{0k}$ by $\tilde{\delta}_k$ and $\sqrt{\tilde{x}}$, we can rewrite (23) as:

$$\tilde{\delta}_k(\tilde{x},t) = -\frac{1}{2}\tilde{\delta}_{k-1}^2(\tilde{x},t)(3+\tilde{\delta}_{k-1}(\tilde{x},t)), \quad (k=1,2,\ldots). \tag{27}$$

The quadratic dependence on $\tilde{\delta}_{k-1}$ means that every Newton–Raphson correction improves the accuracy by several orders of magnitude (until the machine precision is reached); compare (26).

The Formula (27) suggests another way of improving the accuracy because the functions $\tilde{\delta}_k$ are always non-positive for any $k \geqslant 1$. Roughly speaking, we are going to shift the graph of $\tilde{\delta}_k$ upwards by an appropriate modification of the Newton–Raphson formula. In the next section, we describe the general idea of this modification.

3. Modified Newton–Raphson Formulas

The Formula (27) shows that errors introduced by Newton–Raphson corrections are nonpositive, i.e., they take values in intervals $[-\tilde{\delta}_{k\,\max}, 0]$, where $k = 1,2,\ldots$. Therefore, it is natural to introduce a correction term into the Newton–Raphson formulas (23). We expect that the corrections will be roughly half of the maximal relative error. Instead of the maximal error, we introduce two parameters, d_1 and d_2. Thus, we get modified Newton–Raphson formulas:

$$\begin{aligned} \tilde{y}_{11}(\tilde{x},t,d_1) &= 2^{-1}\tilde{y}_{00}(\tilde{x},t)(3-\tilde{y}_{00}^2(\tilde{x},t)\,\tilde{x}) + \frac{d_1}{2\sqrt{\tilde{x}}}, \\ \tilde{y}_{12}(\tilde{x},t,d_1,d_2) &= 2^{-1}\tilde{y}_{11}(\tilde{x},t,d_1)(3-\tilde{y}_{11}^2(\tilde{x},t,d_1)\,\tilde{x}) + \frac{d_2}{2\sqrt{\tilde{x}}}, \end{aligned} \tag{28}$$

where zeroth approximation is assumed in the form (6). In the following section, the term $1/\sqrt{\tilde{x}}$ will be replaced by some approximations of $\tilde{y}$, transforming (28) into a computer code. In order to estimate a possible gain in accuracy, in this section, we temporarily assume that $\tilde{y}$ is the exact value of the inverse square root. The corresponding error functions,

$$\tilde{\delta}_k''(\tilde{x},t,d_1,\ldots,d_k) = \sqrt{\tilde{x}}\,\tilde{y}_{1k}(\tilde{x},t,d_1,\ldots,d_k) - 1, \qquad k \in \{0,1,2,\ldots\}, \tag{29}$$

(where $\tilde{y}_{10}(\tilde{x},t) := \tilde{y}_{00}(\tilde{x},t)$), satisfy:

$$\tilde{\delta}_k'' = -\frac{1}{2}\tilde{\delta}_{k-1}''^2(3+\tilde{\delta}_{k-1}'') + \frac{d_k}{2}, \tag{30}$$

where: $\tilde{\delta}_0''(\tilde{x},t) = \tilde{\delta}_0(\tilde{x},t)$. Note that:

$$\tilde{\delta}_1''(\tilde{x},t,d_1) = \tilde{\delta}_1(\tilde{x},t) + \frac{1}{2}d_1. \tag{31}$$

In order to simplify notation, we usually will suppress the explicit dependence on d_j. We will write, for instance, $\tilde{\delta}_2''(\tilde{x},t)$ instead of $\tilde{\delta}_2''(\tilde{x},t,d_1,d_2)$.

The corrections of the form (28) will decrease relative errors in comparison with the results of earlier papers [38,46]. We have three free parameters (d_1, d_2, and t) to be determined by minimizing the maximal error (in principle, the new parameterization can give a new estimation of the parameter t). By analogy to (25), we are going to find $t = t^{(0)}$ minimizing the error of the first correction (25):

$$\forall_{t\neq t^{(0)}} \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1''(\tilde{x},t^{(0)})| < \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1''(\tilde{x},t)|, \tag{32}$$

where, as usual, $\tilde{A} = [1,4]$.

The first of Equation (30) implies that for any t, the maximal value of $\tilde{\delta}_1''(\tilde{x},t)$ equals $\frac{1}{2}d_1$ and is attained at zeros of $\tilde{\delta}_0''(\tilde{x},t)$. Using the results of Section 2, including (15), (16), (20), and (21), we conclude that the minimum value of $\tilde{\delta}_1''(\tilde{x},t)$ is attained either for $\tilde{x} = t$ or for $\tilde{x} = x_0^{II}$ (where there is the second maximum of $\tilde{\delta}_0''(\tilde{x},t)$), i.e.,

$$\min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t) = \min\left\{\tilde{\delta}_1''(t,t), \tilde{\delta}_1''(x_0^{II},t)\right\} \tag{33}$$

Minimization of $|\tilde{\delta}_1''(\tilde{x},t)|$ can be done with respect to t and with respect to d_1 (these operations obviously commute), which corresponds to:

$$\underbrace{\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)})}_{\tilde{\delta}_{1\,\max}''} = -\min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)}). \tag{34}$$

Taking into account:

$$\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)}) = \frac{d_1}{2}, \quad \min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)}) = \tilde{\delta}_1''(t^{(0)},t^{(0)}) = -\tilde{\delta}_{1\,\max} + \frac{d_1}{2}, \tag{35}$$

we get from (34):

$$\tilde{\delta}_{1\,\max}'' = \frac{1}{2}d_1 = \frac{1}{2}\tilde{\delta}_{1\,\max} \simeq 8.7559\times 10^{-4}, \tag{36}$$

where:

$$\tilde{\delta}_{1\,\max} := \min_{t\in(2,4)} \left(\max_{x\in\tilde{A}} |\tilde{\delta}_1(\tilde{x},t)|\right). \tag{37}$$

and the numerical value of $\tilde{\delta}_{1\,\max}$ is given by (26). These conditions are satisfied for:

$$t^{(0)} = t_1^{(r)} \simeq 3.7298003. \tag{38}$$

In order to minimize the relative error of the second correction, we use equation analogous to (34):

$$\underbrace{\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)})}_{\tilde{\delta}_{2\,\max}''} = -\min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)}), \tag{39}$$

where from (30), we have:

$$\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)}) = \frac{d_2}{2}, \quad \min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)}) = -\frac{1}{2}\tilde{\delta}_{1\,\max}''^2\left(3+\tilde{\delta}_{1\,\max}''\right) + \frac{d_2}{2}. \tag{40}$$

Hence:

$$\tilde{\delta}_{2\,\max}'' = \frac{1}{4}\tilde{\delta}_{1\,\max}''^2\left(3+\tilde{\delta}_{1\,\max}''\right). \tag{41}$$

Expressing this result in terms of formerly computed $\tilde{\delta}_{1\,\max}$ and $\tilde{\delta}_{2\,\max}$, we obtain:

$$\tilde{\delta}''_{2\,\max} = \frac{1}{8}\tilde{\delta}_{2\,\max} + \frac{3}{32}\tilde{\delta}^3_{1\,\max} \simeq 5.75164 \times 10^{-7} \simeq \frac{\tilde{\delta}_{2\,\max}}{7.99}, \tag{42}$$

where:

$$\tilde{\delta}_{2\,\max} = \frac{1}{2}\tilde{\delta}^2_{1\,\max}(3 - \tilde{\delta}_{1\,\max}).$$

Therefore, the above modification of Newton–Raphson formulas decreases the relative error two times after one iteration and almost eight times after two iterations as compared to the standard *InvSqrt* algorithm.

In order to implement this idea in the form of a computer code, we have to replace the unknown $1/\sqrt{\tilde{x}}$ (i.e., $\tilde{y}$) on the right-hand sides of (28) by some numerical approximations.

4. New Algorithm with Higher Accuracy

Approximating $1/\sqrt{\tilde{x}}$ in Formulas (28) by values on the left-hand sides, we transform (28) into:

$$\begin{aligned} \tilde{y}_{21} &= \frac{1}{2}\tilde{y}_{20}(3 - \tilde{y}^2_{20}\,\tilde{x}) + \frac{1}{2}d_1\tilde{y}_{21}, \\ \tilde{y}_{22} &= \frac{1}{2}\tilde{y}_{21}(3 - \tilde{y}^2_{21}\,\tilde{x}) + \frac{1}{2}d_2\tilde{y}_{22}, \end{aligned} \tag{43}$$

where $\tilde{y}_{2k}$ ($k = 1, 2, \ldots$) depend on $\tilde{x}, t$ and d_j (for $1 \leqslant j \leqslant k$). We assume $\tilde{y}_{20} \equiv \tilde{y}_{00}$, i.e., the zeroth approximation is still given by (6). We can see that $\tilde{y}_{21}$ and $\tilde{y}_{22}$ can be explicitly expressed by $\tilde{y}_{20}$ and $\tilde{y}_{21}$, respectively.

Parameters d_1 and d_2 have to be determined by minimization of the maximum error. We define error functions in the usual way:

$$\Delta^{(1)}_k = \frac{\tilde{y}_{2k} - \tilde{y}}{\tilde{y}} = \sqrt{\tilde{x}}\,\tilde{y}_{2k} - 1\,. \tag{44}$$

Substituting (44) into (43), we get:

$$\Delta^{(1)}_1(\tilde{x}, t, d_1) = \frac{d_1}{2 - d_1} - \frac{1}{2 - d_1}\tilde{\delta}^2_0(\tilde{x}, t)(3 + \tilde{\delta}_0(\tilde{x}, t)) = \frac{d_1 + 2\tilde{\delta}_1(\tilde{x}, t)}{2 - d_1}, \tag{45}$$

$$\Delta^{(1)}_2(\tilde{x}, t, d, d_2) = \frac{d_2}{2 - d_2} - \frac{1}{2 - d_2}\left(\Delta^{(1)}_1(\tilde{x}, t, d_1)\right)^2\left(3 + \Delta^{(1)}_1(\tilde{x}, t, d_1)\right). \tag{46}$$

The equation (45) expresses $\Delta^{(1)}_1(\tilde{x}, t, d_1)$ as a linear function of the nonpositive function $\tilde{\delta}_1(\tilde{x}, t)$ with coefficients depending on the parameter d_1. The optimum parameters t and d_1 will be estimated by the procedure described in Section 3. First, we minimize the amplitude of the relative error function, i.e., we find $t^{(1)}$ such that:

$$\max_{\tilde{x}\in\tilde{A}} \Delta^{(1)}_1(\tilde{x}, t^{(1)}) - \min_{\tilde{x}\in\tilde{A}} \Delta^{(1)}_1(\tilde{x}, t^{(1)}) \leqslant \max_{\tilde{x}\in\tilde{A}} \Delta^{(1)}_1(\tilde{x}, t) - \min_{\tilde{x}\in\tilde{A}} \Delta^{(1)}_1(\tilde{x}, t) \tag{47}$$

for all $t \neq t^{(1)}$. Second, we determine $d^{(1)}_1$ such that:

$$\max_{\tilde{x}\in\tilde{A}} \Delta^{(1)}_1(\tilde{x}, t^{(1)}, d^{(1)}_1) = -\min_{\tilde{x}\in\tilde{A}} \Delta^{(1)}_1(\tilde{x}, t^{(1)}, d^{(1)}_1)\,. \tag{48}$$

Thus, we have:

$$\max_{\tilde{x}\in\tilde{A}} |\Delta^{(1)}_1(\tilde{x}, t^{(1)}, d^{(1)}_1)| \leqslant \max_{\tilde{x}\in\tilde{A}} |\Delta^{(1)}_1(\tilde{x}, t, d_1)| \tag{49}$$

for all real d_1 and $t \in (2,4)$. $\Delta_1^{(1)}(\tilde{x},t)$ is an increasing function of $\tilde{\delta}_1(\tilde{x},t)$; hence:

$$-\frac{d_1^{(1)} - 2\max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_1(\tilde{x},t_1^{(1)})|}{2-d_1^{(1)}} = \frac{d_1^{(1)}}{2-d_1^{(1)}}, \tag{50}$$

which is satisfied for:

$$d_1^{(1)} = \max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_1(\tilde{x},t_1^{(1)})| = \tilde{\delta}_{1\,\max}. \tag{51}$$

Thus, we can find the maximum error of the first correction $\Delta_1^{(1)}(\tilde{x},t_1^{(1)})$ (presented in Figure 1):

$$\max_{\tilde{x}\in\tilde{A}}|\Delta_1^{(1)}(\tilde{x},t^{(1)})| = \frac{\max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_1(\tilde{x},t^{(1)})|}{2-\max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_1(\tilde{x},t^{(1)})|}, \tag{52}$$

which assumes the minimum value for $t^{(1)} = t_1^{(r)}$:

$$\Delta_{1\,\max}^{(1)} = \frac{\max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_1(\tilde{x},t_1^{(r)})|}{2-\max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_1(\tilde{x},t_1^{(r)})|} = \frac{\tilde{\delta}_{1\,\max}}{2-\tilde{\delta}_{1\,\max}} \simeq 8.7636\times10^{-4} \simeq \frac{\tilde{\delta}_{1\,\max}}{2.00}. \tag{53}$$

This result practically coincides with $\tilde{\delta}''_{1\,\max}$ given by (36).

Analogously, we can determine the value of $d_2^{(1)}$ (assuming that $t = t^{(1)}$ is fixed):

$$-\frac{d_2^{(1)} - \max_{\tilde{x}\in\tilde{A}}|\Delta_1^{(1)2}(\tilde{x},t^{(1)})(3+\Delta_1^{(1)}(\tilde{x},t^{(1)}))|}{2-d_2^{(1)}} = \frac{d_2^{(1)}}{2-d_2^{(1)}}. \tag{54}$$

Now, the deepest minimum comes from the global maximum:

$$\max_{\tilde{x}\in\tilde{A}}|\Delta_1^{(1)2}(\tilde{x},t^{(1)})(3+\Delta_1^{(1)}(\tilde{x},t^{(1)}))| = \frac{2\tilde{\delta}_{1\,\max}^2(3-\tilde{\delta}_{1\,\max})}{(2-\tilde{\delta}_{1\,\max})^3}. \tag{55}$$

Therefore, we get:

$$d_2^{(1)} = \frac{\tilde{\delta}_{1\,\max}^2(3-\tilde{\delta}_{1\,\max})}{(2-\tilde{\delta}_{1\,\max})^3} \simeq 1.15234\times10^{-6}, \tag{56}$$

and the maximum error of the second correction is given by:

$$\Delta_{2\,\max}^{(1)} = \frac{d_2^{(1)}}{2-d_2^{(1)}} \simeq 5.76173\times10^{-7} \simeq \frac{\tilde{\delta}_{2\,\max}}{7.98}, \tag{57}$$

which is very close to the value of $\tilde{\delta}''_{2\,\max}$ given by (42).

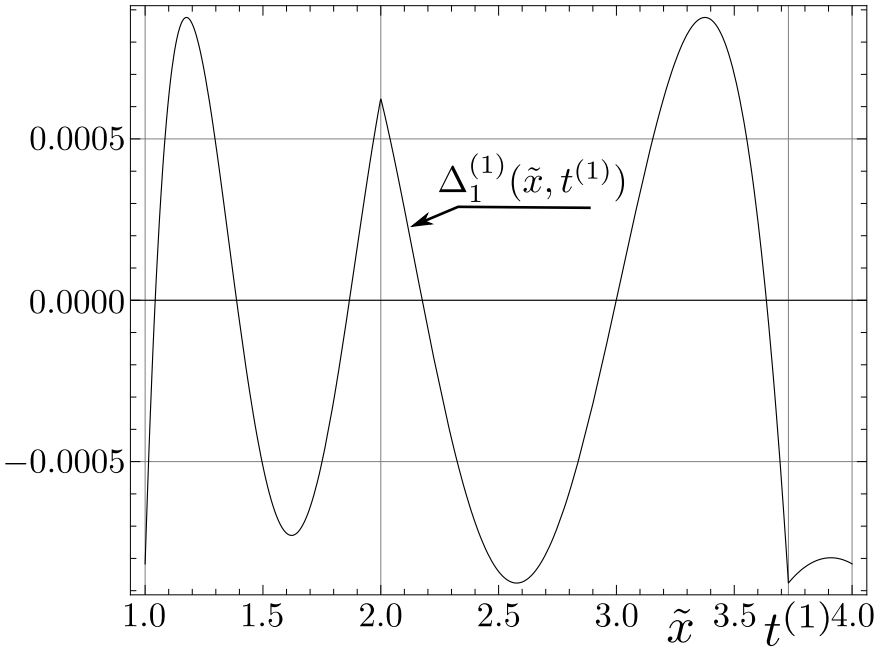

Figure 1. Graph of the function $\Delta_1^{(1)}(\tilde{x}, t^{(1)})$.

Thus, we have obtained the algorithm *InvSqrt1*, see Algorithm 2, which looks like *InvSqrt* with modified values of the numerical coefficients.

Algorithm 2: *InvSqrt1*.

```
1.  float InvSqrt1(float x){
2.        float simhalfnumber = 0.500438180f*x;
3.        int i = *(int*) &x;
4.        i = 0x5F375A86 - (i>>1);
5.        y = *(float*) &i;
6.        y = y*(1.50131454f - simhalfnumber*y*y);
7.        y = y*(1.50000086f - 0.999124984f*simhalfnumber*y*y);
8.        return y ;
9.  }
```

Comparing *InvSqrt*1 with *InvSqrt*, we easily see that the number of algebraic operations in *InvSqrt*1 is greater by one (an additional multiplication in Line 7, corresponding to the second iteration of the modified Newton–Raphson procedure). We point out that the magic constants for *InvSqrt* and *InvSqrt1* are the same.

5. Numerical Experiments

The new algorithm *InvSqrt*1 was tested on the processor Intel Core i5-3470 using the compiler TDM-GCC 4.9.2 32-bit. Using the same hardware for testing the code *InvSqrt*, we obtained practically the same values of errors as those obtained by Lomont [38]. The same results were obtained also on Intel i7-5700. In this section, we analyze the rounding errors for the code *InvSqrt1*.

Applying algorithm *InvSqrt*1, we obtain relative errors InvSqrt1$(\tilde{x})$ characterized by "oscillations" with a center slightly shifted with respect to the analytical approximate solution $\tilde{y}_{22}(\tilde{x}, t^{(1)})$; see

Figure 2. The observed blur can be expressed by a relative deviation of the numerical result from $\tilde{y}_{22}(\tilde{x})$:

$$\varepsilon^{(1)}(\tilde{x}) = \frac{\mathrm{InvSqrt1}(\tilde{\mathrm{x}}) - \tilde{y}_{22}(\tilde{x}, t^{(1)})}{\tilde{y}_{22}(\tilde{x}, t^{(1)})}. \tag{58}$$

The values of this error are distributed symmetrically around the mean value $\langle \varepsilon^{(1)} \rangle$:

$$\langle \varepsilon^{(1)} \rangle = 2^{-1} N_m^{-1} \sum_{x \in [1,4)} \varepsilon^{(1)}(\tilde{x}) = -1.398 \times 10^{-8} \tag{59}$$

enclosing the range:

$$\varepsilon^{(1)}(\tilde{x}) \in [-9.676 \times 10^{-8}, 6.805 \times 10^{-8}], \tag{60}$$

see Figure 3. The blur parameters of the function $\varepsilon^{(1)}(\tilde{x}, t)$ show that the main source of the difference between analytical and numerical results is the use of precision **float** and, in particular, rounding of constant parameters of the function InvSqrt1. We point out that in this case, the amplitude of the error oscillations is about 40% greater than the amplitude of oscillations of $(\tilde{y}_{00} - \tilde{y}_0)/\tilde{y}_0$ (i.e., in the case of *InvSqrt*); see the right part of Figure 2 in [46]. It is worth noting that for the first Newton–Raphson correction, the blur is of the same order, but due to a much higher error value in this case, its effect is negligible (i.e., Figure 1 would be practically the same with or without the blur). The maximum numerical errors practically coincide with the analytical result (53), i.e.,

$$\Delta^{(1)}_{1,N\,\min} \approx -8.76 \times 10^{-4}\,, \qquad \Delta^{(1)}_{1,N\,\max} \approx 8.76 \times 10^{-4}\,. \tag{61}$$

In the case of the second Newton–Raphson correction, we compared results produced by *InvSqrt*1 with exact values of the inverse square root for all numbers x of the type **float** such that $e_x \in [-126, 128)$. The range of errors was the same for all these intervals (except $e_x = -126$):

$$\Delta^{(1)}_{2;N}(x) = \mathrm{sqrt}(x) * \mathrm{InvSqrt1}(x) - 1. \;\in\; (-6.62 \times 10^{-7}, 6.35 \times 10^{-7})\,. \tag{62}$$

For $e_x = -126$, the interval of errors was slightly wider: $(-6.72 \times 10^{-7}, 6.49 \times 10^{-7})$, which can be explained by the fact that the analysis presented in this paper is not applicable to subnormal numbers; see (1). Therefore, our tests showed that relative errors for all numbers of the type float belong to the interval $(\Delta^{(1)}_{2,N\,\min}, \Delta^{(1)}_{2,N\,\max})$, where:

$$\Delta^{(1)}_{2,N\,\min} \approx -6.72 \times 10^{-7}, \qquad \Delta^{(1)}_{2,N\,\max} \approx 6.49 \times 10^{-7}\,. \tag{63}$$

These values are significantly higher than the analytical result (5.76×10^{-7}) (see (57)), but are still much lower than the analogous error for the algorithm *InvSqrt* (4.60×10^{-6}; see [46]).

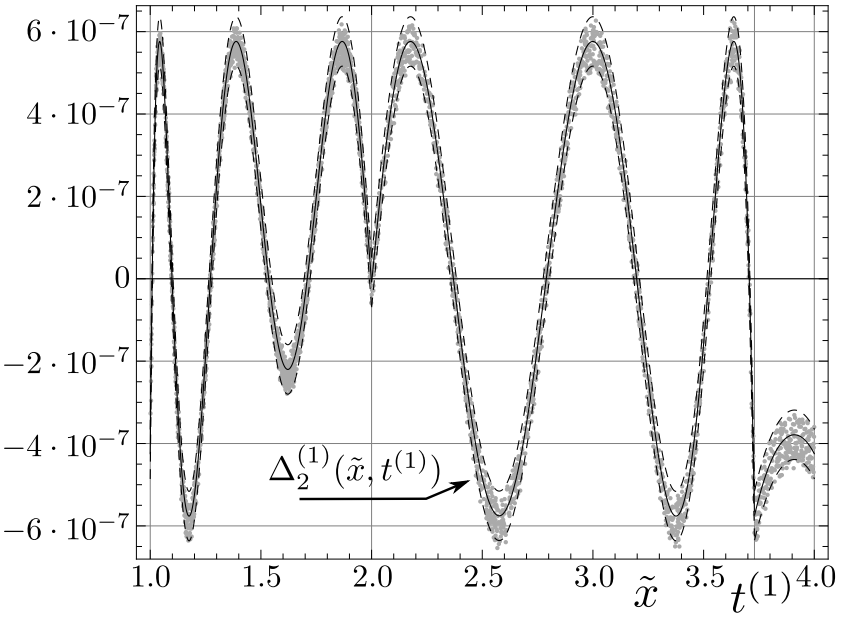

Figure 2. Solid lines represent function $\Delta_2^{(1)}(\tilde{x}, t^{(1)})$. Its vertical shifts by $\pm 6 \times 10^{-8}$ are denoted by dashed lines. Finally, dots represent relative errors for 4000 random values $x \in (2^{-126}, 2^{128})$ produced by algorithm *InvSqrt*1.

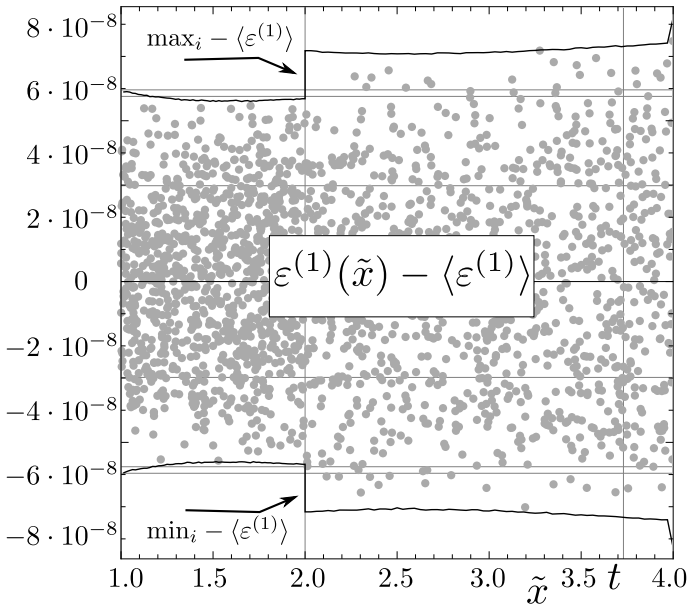

Figure 3. Relative error $\varepsilon^{(1)}$ arising during the **float** approximation of corrections $\tilde{y}_{22}(\tilde{x}, t)$. Dots represent errors determined for 2000 random values $\tilde{x} \in [1, 4)$. Solid lines represent maximum ($\max_i$) and minimum ($\min_i$) values of relative errors (intervals $[1, 2)$ and $[2, 4)$ were divided into 64 equal intervals, and then, extremum values were determined in all these intervals).

6. Conclusions

In this paper, we presented a modification of the famous code *InvSqrt* for fast computation of the inverse square root of single-precision floating-point numbers. The new code had the same magic constant, but the second part (which consisted of Newton–Raphson iterations) was modified. In the case of one Newton–Raphson iteration, the new code *InvSqrt1* had the same computational cost as *InvSqrt* and was two-times more accurate. In the case of two iterations, the computational cost of the new code was only slightly higher, but its accuracy was higher by almost seven times.

The main idea of our work consisted of modifying coefficients in the Newton–Raphson method and demanding that the maximal error be as small as possible. Such modifications can be constructed if the distribution of errors for Newton–Raphson corrections is not symmetric (like in the case of the inverse square root, when they are non-positive functions).

Author Contributions: Conceptualization, L.V.M.; formal analysis, C.J.W.; investigation, C.J.W., L.V.M., and J.L.C.; methodology, C.J.W. and L.V.M.; visualization, C.J.W.; writing, original draft, J.L.C.; writing, review and editing, J.L.C.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ercegovac, M.D.; Lang, T. *Digital Arithmetic*; Morgan Kaufmann: Burlington, MA, USA, 2003.
2. Parhami, B. *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd ed.; Oxford University Press: New York, NY, USA, 2010
3. Diefendorff, K.; Dubey, P.K.; Hochsprung, R.; Scales, H. AltiVec extension to PowerPC accelerates media processing. *IEEE Micro* **2000**, *20*, 85–95.
4. Harris, D. A Powering Unit for an OpenGL Lighting Engine. In Proceedings of the 35th Asilomar Conference on Singals, Systems, and Computers, Pacific Grove, CA, USA, 4–7 November 2001; pp. 1641–1645.
5. Sadeghian, M.; Stine, J. Optimized Low-Power Elementary Function Approximation for Chybyshev series Approximation. In Proceedings of the 46th Asilomar Conference on Signal Systems and Computers, Pacific Grove, CA, USA, 4–7 November 2012.
6. Russinoff, D.M. A Mechanically Checked Proof of Correctness of the AMD K5 Floating Point Square Root Microcode. *Form. Methods Syst. Des.* **1999**, *14*, 75–125.
7. Cornea, M.; Anderson, C.; Tsen, C. Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic. In *Software and Data Technologies* (*Communications in Computer and Information Science*); Springer: Berlin/Heidelberg, Germany, 2008; Volume 10, pp. 97–109.
8. Muller, J.-M.; Brisebarre, N.; Dinechin, F.; Jeannerod, C.-P.; Lefèvre, V.; Melquiond, G.; Revol, N.; Stehlé, D.; Torres, S. Hardware Implementation of Floating-Point Arithmetic. In *Handbook of Floating-Point Arithmetic*; Birkhäuser: Basel, Switzerland, 2010; pp. 269–320.
9. Muller, J.-M.; Brisebarre, N.; Dinechin, F.; Jeannerod, C.-P.; Lefèvre, V.; Melquiond, G.; Revol, N.; Stehlé, D.; Torres, S. Software Implementation of Floating-Point Arithmetic. In *Handbook of Floating-Point Arithmetic*; Birkhäuser: Basel, Switzerland, 2010; pp. 321–372.
10. Viitanen, T.; Jääskeläinen, P.; Esko, O.; Takala, J. Simplified floating-point division and square root. In Proceedings of the IEEE International Conference on Acoustics Speech and Signal Process, Vancouver, BC, Canada, 26–31 May 2013; pp. 2707–2711.
11. Ercegovac, M.D.; Lang, T. *Division and Square Root: Digit Recurrence Algorithms and Implementations*; Kluwer Academic Publishers: Boston, MA, USA, 1994.
12. Liu, W.; Nannarelli, A. Power Efficient Division and Square root Unit. *IEEE Trans. Comp.* **2012**, *61*, 1059–1070.
13. Deng, L.X.; An, J.S. A low latency High-throughput Elementary Function Generator based on Enhanced double rotation CORDIC. In Proceedings of the IEEE Symposium on Computer Applications and Communications (SCAC), Weihai, China, 26–27 July 2014.
14. Nguyen, M.X.; Dinh-Duc, A. Hardware-Based Algorithm for Sine and Cosine Computations using Fixed Point Processor. In Proceedings of the 11th International Conference on Electrical Engineering/Electronics Computer, Telecommuncations and Information Technology, Nakhon Ratchasima, Thailand, 14–17 May 2014.
15. Cornea, M. *Intel® AVX-512 Instructions and Their Use in the Implementation of Math Functions*; Intel Corporation: 2015. Available online: http://arith22.gforge.inria.fr/slides/s1-cornea.pdf (accessed on 27 June 2019).
16. Jiang, H.; Graillat, S.; Barrio, R.; Yang, C. Accurate, validated and fast evaluation of elementary symmetric functions and its application. *Appl. Math. Comput.* **2016**, *273*, 1160–1178.
17. Fog, A. Software Optimization Resources, Instruction Tables: Lists of Instruction Latencies, Throughputs and Micro-Operation Breakdowns for Intel, AMD and VIA CPUs. Available online: http://www.agner.org/optimize/ (accessed on 27 June 2019).

18. Moroz, L.; Samotyy, W. Efficient floating-point division for digital signal processing application. *IEEE Signal Process. Mag.* **2019**, *36*, 159–163.
19. Eberly, D.H. *GPGPU Programming for Games and Science*; CRC Press: Boca Raton, FL, USA, 2015.
20. Ide, N.; Hirano, M.; Endo, Y.; Yoshioka, S.; Murakami, H.; Kunimatsu, A.; Sato, T.; Kamei, T.; Okada, T.; Suzuoki, M. 2.44-GFLOPS 300-MHz Floating-Point Vector-Processing Unit for High-Performance 3D Graphics Computing. *IEEE J. Solid-State Circuits* **2000**, *35*, 1025–1033.
21. Oberman, S.; Favor, G.; Weber, F. AMD 3DNow! technology: Architecture and implementations. *IEEE Micro* **1999**, *19*, 37–48.
22. Kwon, T.J.; Draper, J. Floating-point Division and Square root Implementation using a Taylor-Series Expansion Algorithm with Reduced Look-Up Table. In Proceedings of the 51st Midwest Symposium on Circuits and Systems, Knoxville, TN, USA, 10–13 August 2008.
23. Hands, T.O.; Griffiths, I.; Marshall, D.A.; Douglas, G. The fast inverse square root in scientific computing. *J. Phys. Spec. Top.* **2011**, *10*, A2-1.
24. Blinn, J. Floating-point tricks. *IEEE Comput. Graph. Appl.* **1997**, *17*, 80–84.
25. Janhunen, J. Programmable MIMO Detectors. Ph.D. Thesis, University of Oulu, Tampere, Finland, 2011.
26. Stanislaus, J.L.V.M.; Mohsenin, T. High Performance Compressive Sensing Reconstruction Hardware with QRD Process. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'12), Seoul, Korea, 20–23 May 2012.
27. Avril, Q.; Gouranton, V.; Arnaldi, B. *Fast Collision Culling in Large-Scale Environments Using GPU Mapping Function*; ACM Eurographics Parallel Graphics and Visualization: Cagliari, Italy, 2012.
28. Schattschneider, R. Accurate High-resolution 3D Surface Reconstruction and Localisation Using a Wide-Angle Flat Port Underwater Stereo Camera. Ph.D. Thesis, University of Canterbury, Christchurch, New Zealand, 2014.
29. Zafar, S.; Adapa, R. Hardware architecture design and mapping of "Fast Inverse Square Root's algorithm". In Proceedings of the International Conference on Advances in Electrical Engineering (ICAEE), Tamilnadu, India, 9–11 January 2014; pp. 1–4.
30. Hänninen, T.; Janhunen, J.; Juntti, M. Novel detector implementations for 3G LTE downlink and uplink. *Analog. Integr. Circ. Sig. Process.* **2014**, *78*, 645–655.
31. Li, Z.Q.; Chen, Y.; Zeng, X.Y. OFDM Synchronization implementation based on Chisel platform for 5G research. In Proceedings of the 2015 IEEE 11th International Conference on ASIC (ASICON), Chengdu, China, 3–6 November 2015.
32. Hsu, C.J.; Chen, J.L.; Chen, L.G. An Efficient Hardware Implementation of HON4D Feature Extraction for Real-time Action Recognition. In Proceedings of the 2015 IEEE International Symposium on Consumer Electronics (ISCE), Madrid, Spain, 24–26 June 2015.
33. Hsieh, C.H.; Chiu, Y.F.; Shen, Y.H.; Chu, T.S.; Huang, Y.H. A UWB Radar Signal Processing Platform for Real-Time Human Respiratory Feature Extraction Based on Four-Segment Linear Waveform Model. *IEEE Trans. Biomed. Circ. Syst.* **2016**, *10*, 219–230.
34. Lv, J.D.; Wang, F.; Ma, Z.H. Peach Fruit Recognition Method under Natural Environment. In *Eighth International Conference on Digital Image Processing (ICDIP 2016), Chengdu, China, 20–22 May 2016*; Falco, C.M., Jaing, X.D., Eds.; SPIE: Bellingham, WA, USA 2016; Volume 10033, p. 1003317.
35. Sangeetha, D.; Deepa, P. Efficient Scale Invariant Human Detection using Histogram of Oriented Gradients for IoT Services. In Proceedings of the 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems, Hyderabad, India, 7–11 January 2017; pp. 61–66.
36. Lin, J.; Xu, Z.G.; Nukada, A.; Maruyama, N.; Matsuoka, S. Optimizations of Two Compute-bound Scientific Kernels on the SW26010 Many-core Processor. In Proceedings of the 46th International Conference on Parallel Processing, Bristol, UK, 14–17 August 2017; pp. 432–441.
37. Quake III Arena, id Software 1999. Available online: https://github.com/id-Software/Quake-III-Arena/blob/master/code/game/q_math.c#L552 (accessed on 27 June 2019).
38. Lomont, C. Fast Inverse Square Root, Purdue University, Tech. Rep., 2003. Available online: http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf (accessed on 27 June 2019).
39. Warren, H.S. *Hacker's Delight*, 2nd ed.; Pearson Education: London, UK, 2013.
40. Martin, P. Eight Rooty Pieces. *Overload J.* **2016**, *135*, 8–12.

41. Robertson, M. A Brief History of InvSqrt. Bachelor's Thesis, University of New Brunswick, Fredericton, NB, Canada, 2012.
42. Self, B. Efficiently Computing the Inverse Square Root Using Integer Operations. Available online: https://sites.math.washington.edu/~morrow/336_12/papers/ben.pdf (accessed on 27 June 2019).
43. McEniry, C. *The Mathematics Behind the Fast Inverse Square Root Function Code*, Tech. Rep.; August 2007. https://web.archive.org/web/20150511044204/http://www.daxia.com/bibis/upload/406Fast_Inverse_Square_Root.pdf (accessed on 27 June 2019).
44. Eberly, D. An Approximation for the Inverse Square Root Function, 2015. Available online: http://www.geometrictools.com/Documentation/ApproxInvSqrt.pdf (accessed on 27 June 2019).
45. Kornerup, P.; Muller, J.-M. Choosing starting values for certain Newton–Raphson iterations. *Theor. Comp. Sci.* **2006**, *351*, 101–110.
46. Moroz, L.; Walczyk, C.J.; Hrynchyshyn, A.; Holimath, V.; Cieśliński, J.L. Fast calculation of inverse square root with the use of magic constant—Analytical approach. *Appl. Math. Comput.* **2018**, *316*, 245–255.