Accurate Conversion of Earth-Fixed Earth-Centered Coordinates to Geodetic Coordinates

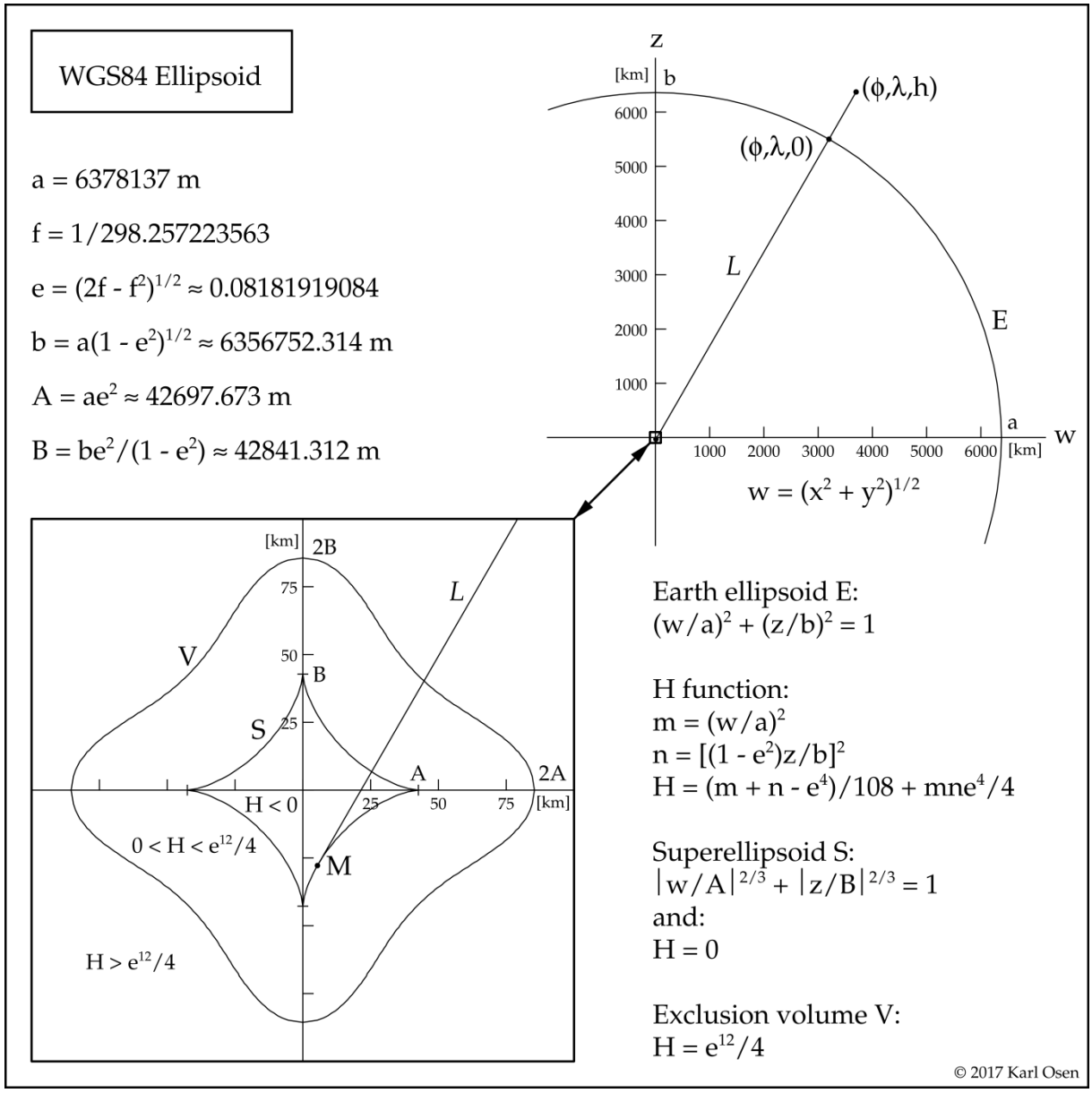
Karl Osen

Accurate Conversion of Earth-Fixed Earth-Centered Coordinates to Geodetic Coordinates

Karl Osen
31 October 2019

Abstract

A closed form algorithm for the exact transformation of Earth-Centered Earth-Fixed (ECEF) coordinates (x, y, z) to geodetic coordinates (ϕ, λ, h) is presented that is computationally fast, safe and accurate. Boosting the computational robustness of Jijie Zhu's algorithm, it reduces the worst case transformation error by up to 500 million times.



Zhu's Algorithm

In 1993 Jijie Zhu published [1] the following closed form algorithm for the exact transformation of Earth-Centered Earth-Fixed (ECEF) coordinates (x, y, z) to geodetic coordinates (ϕ, λ, h):

$$w = \sqrt{x^2 + y^2} \tag{1}$$

$$l = e^2/2 \tag{2}$$

$$m = (w/a)^2 \tag{3}$$

$$n = [(1 - e^2)\, z/b]^2 \tag{4}$$

$$i = -(2l^2 + m + n)/2 \tag{5}$$

$$k = l^2(l^2 - m - n) \tag{6}$$

$$q = (m + n - 4l^2)^3/216 + mnl^2 \tag{7}$$

$$D = \sqrt{(2q - mnl^2)mnl^2} \tag{8}$$

$$\beta = i/3 - \sqrt[3]{q + D} - \sqrt[3]{q - D} \tag{9}$$

$$t = \sqrt{\sqrt{\beta^2 - k} - (\beta + i)/2} - \text{sign}(m - n)\sqrt{(\beta - i)/2} \tag{10}$$

$$w_1 = w/(t + l) \tag{11}$$

$$z_1 = (1 - e^2)z/(t - l) \tag{12}$$

$$\phi = \arctan\left[z_1/\left((1 - e^2)w_1\right)\right] \tag{13}$$

$$\lambda = 2\arctan[(w - x)/y] \tag{14}$$

$$h = \text{sign}(t - 1 + l)\sqrt{(w - w_1)^2 + (z - z_1)^2} \tag{15}$$

Special case when $w = 0$:

$$h = \text{sign}(z)(z - b) \tag{16}$$

$$\phi = \text{sign}(z)(\pi/2) \tag{17}$$

In 1994 Zhu showed [2] that equation (9) can be rewritten to have only one cube root operation, thereby speeding up the transformation:

$$\beta = i/3 - \sqrt[3]{q + D} - P/\sqrt[3]{q + D} \tag{18}$$

Where:

$$P = ((m + n - 4l^2)/6)^2 \tag{19}$$

Operating Range

Zhu states [1] that the transformation of Earth-Centered Earth-Fixed (ECEF) coordinates (x, y, z) to geodetic coordinates (ϕ, λ, h) is feasible provided the point does not fall within 43 km of the Earth's center, and that this minimum distance warrants that the following condition is met:

$$(m + n - 4l^2) > 0 \tag{20}$$

Provided condition (20) is met we find in (7) that $q > 0$ since $mnl^2 \geq 0$. Consequently the radicand in (8) is always positive. We may furthermore show that $q > D \geq 0$ which implies that we won't take the roots of negative numbers in (9) nor divide by zero in (18).

The feasibility analysis of Zhu's algorithm is essentially a two-dimensional problem, and we therefore introduce the two-dimensional coordinate notations $\{w, z\}$ and $\langle \phi, h \rangle$.

Replacing m and n in (20) we find:

$$e^4((w/A)^2 + (z/B)^2 - 1) > 0 \tag{21}$$

Where:

$$A = ae^2 = 42697.7\ m \tag{22}$$

$$B = be^2/(1 - e^2) = ae^2/\sqrt{1 - e^2} = 42841.3\ m \tag{23}$$

Inequality (21) implies that the point $\{w, z\}$ should be outside the ellipsoid with equatorial radius A and polar radius B. Hence, Zhu's 43 km rule has a safety margin of 158.7 meters. The ellipsoid equation is:

$$w^2/A^2 + z^2/B^2 = 1 \tag{24}$$

Ellipsoid (24) has the same eccentricity as the WGS84 ellipsoid, but the polar radius is the larger radius.

Whereas Zhu's 43 km rule can be used for most applications, the fundamental operating range of Zhu's algorithm is related to the *radius of curvature in the meridian* [3]. This radius is the curvature in the north-south direction of the surface of the WGS84 ellipsoid at $\langle \phi, 0 \rangle$, and is given by:

$$R_M = a(1 - e^2)/(1 - e^2\sin^2\phi)^{3/2} \tag{25}$$

R_M has its minimum at equator and its maximum at the poles:

$$R_{Mmin} = a - A \tag{26}$$

$$R_{Mmax} = b + B \tag{27}$$

The center of the circle corresponding to R_M has the following coordinates:

$$w_M = ae^2\cos^3\phi/(1 - e^2\sin^2\phi)^{3/2} \tag{28}$$

$$z_M = -ae^2(1 - e^2)\sin^3\phi/(1 - e^2\sin^2\phi)^{3/2} \tag{29}$$

We have $|w_M| \leq A$ and $|z_M| \leq B$.

The line L starts in $\{w_M, z_M\}$ and ends in $\langle \phi, 0 \rangle$. The length of L is R_M. L is orthogonal to the surface of the WGS84 ellipsoid in $\langle \phi, 0 \rangle$ and its slope relative to the equatorial plane is $\tan \phi$. The points $\{w_M, z_M\}$ and $\langle \phi, 0 \rangle$ are located on opposite sides of the equatorial plane (unless $\phi = 0$).

If $h \leq 0$ the point $\langle \phi, h \rangle$ is located on the line L. If $h > 0$ the point $\langle \phi, h \rangle$ is located on the outwards extension of L. The distance from $\{w_M, z_M\}$ to $\langle \phi, h \rangle$ is $R_M + h$.

Consider a superellipsoid with the same equatorial and polar radii as ellipsoid (24):

$$|w/A|^{2/3} + |z/B|^{2/3} = 1 \tag{30}$$

Any point $\{w_M, z_M\}$ satisfy equation (30), which implies that the starting point of any L is located on the surface of this superellipsoid.

From the derivatives of (28) and (29) we find:

$$(dz_M/dw_M) = (dz_M/d\phi)/(dw_M/d\phi) = \tan \phi \tag{31}$$

Equation (31) shows that L is tangent to the surface of the superellipsoid in $\{w_M, z_M\}$. The line L consequently passes through the superellipsoid (unless $\phi = 0$).

Zhu's algorithm transforms ECEF coordinates $\{w, z\}$ to geodetic coordinates $\langle \phi, h \rangle$ so that $\langle \phi, h \rangle$ and $\langle \phi, 0 \rangle$ are located on the same side of the equatorial plane. Zhu's algorithm does not work if $\{w, z\}$ is located on two lines L (i.e. one in the northern hemisphere and one in the southern hemisphere), which is the case for all points inside the superellipsoid.

The points $\{\pm A, 0\}$ and $\{0, \pm B\}$ are on the surface of the superellipsoid, but generate divide-by-zero operations in equations (12) and (18). A point $\{w, z\}$ should therefore be outside the superellipsoid for successful conversion:

$$|w/A|^{2/3} + |z/B|^{2/3} > 1 \tag{32}$$

Equation (8) has a component that tells us directly if the transformation of a point $\{w, z\}$ is feasible:

$$H = 2q - mnl^2 \tag{33}$$

Inequality (32) is linked to equation (33) as follows:

$$\text{sign}\big(|w/A|^{2/3} + |z/B|^{2/3} - 1\big) = \text{sign}(H) \tag{34}$$

Replacing l and q in (33) we find:

$$H = [(\mathrm{m} + \mathrm{n} - \mathrm{e}^4)^3 + 27\mathrm{mne}^4]/108 \tag{35}$$

Replacing m and n in (35) gives:

$$H = e^{12}([(w/A)^2 + (z/B)^2 - 1]^3 + 27(w/A)^2(z/B)^2)/108 \tag{36}$$

The derivatives of H with respect to w and z are:

$$dH/dw = we^{12}([(w/A)^2 + (z/B)^2 - 1]^2 + 9(z/B)^2)/(18A^2) \tag{37}$$

$$dH/dz = ze^{12}([(w/A)^2 + (z/B)^2 - 1]^2 + 9(w/A)^2)/(18B^2) \quad (38)$$

Combining (30) and (36) shows that $H = 0$ for all points on the superellipsoid. Equations (37) and (38) show that $dH/d|w| > 0$ and $dH/d|z| > 0$ for $\{\neq 0, \neq 0\}$. For $z = 0$ we have $dH/d|w| > 0$ except for $w = 0$ and $|w| = \mathrm{A}$ when $dH/d|w| = 0$. For $w = 0$ we have $dH/d|z| > 0$ except for $z = 0$ and $|z| = \mathrm{B}$ when $dH/d|z| = 0$. This implies that equation (34) holds.

The following test can therefore be used to check if an ECEF point mathematically can be converted to geodetic coordinates using Zhu's algorithm:

$$\mathrm{H} > 0 \quad (39)$$

Unfortunately condition (39) is insufficient to ensure proper computational behavior of Zhu's algorithm close to the singularities at $\{\pm A, 0\}$ and $\{0, \pm B\}$. To stay sufficiently far away from these singularities we can advantageously introduce an exclusion volume fully containing the superellipsoid (30). Tests show that any point outside a volume V defined by $H = H_{min} = e^{12}/4$, having equatorial radius $2A = 85395.3$ meters and polar radius $2B = 85682.6$ meters, avoids all numerical problems and accuracy degradations related to the proximity of the superellipsoid (30).

$$H_{min} = e^{12}/4 \quad (40)$$

Consequently the following test can be used in to check if an ECEF point can accurately be converted to geodetic coordinates using Zhu's algorithm:

$$\mathrm{H} > H_{min} \quad (41)$$

V looks like a deformed ellipsoid squeezed in at mid-latitudes, having its smallest distance from the center of the Earth of 59973.3 meters at $\{\pm 42415.0, \pm 42400.1\}$. The volume of V is 57.5 % of an ellipsoid with equatorial radius $2A$ and polar radius $2B$. The maximum negative convertible altitude due to the exclusion volume V is -6305868.3 meters at $\phi = \pm 53.3$ degrees.

Avoiding Negative Radicands

Numerical noise in implementations of Zhu's algorithm may result in (extremely small) negative radicands. Taking the root of a negative number will cause a computer program to fail or abort. To avoid such failures one may consider using absolute value functions before a root operations.

In equation (8) noise induced negative radicands may occur at very small latitudes, and the equation should therefore be modified as follows:

$$D = \sqrt{|(2q - mnl^2)mnl^2|} \quad (42)$$

In equation (10) noise induced negative radicands may occur at latitudes close to ±45.3 degrees, and the equation should therefore be modified as follows:

$$t = \sqrt{\sqrt{\beta^2 - k} - (\beta + i)/2} - \mathrm{sign}(m - n)\sqrt{|(\beta - i)/2|} \quad (43)$$

Fixing the Accuracy Problem

The accuracy of Zhu's equation is generally quite good, but degrades considerably at latitudes close to ±45.3 degrees [2]. This accuracy degradation occurs when m is close to n. Combining equations (3) and (4) we find:

$$z^2/w^2 = 1/(1-e^2) \tag{44}$$

Equation (44) represents two cones originating in the Earth's center. One cone extends northwards and the other southwards, and they are symmetric around the z axis. The angle between the cone surfaces and the equatorial plane is $\arctan\left(1/\sqrt{1-e^2}\right) = 45.096$ degrees. Points that are very far from the Earth have $m = n$ for $\phi = \pm 45.096$ degrees.

The intersections between the cones and the surface of the WGS84 ellipsoid occurs at $w = \pm a/\sqrt{1+1/(1-e^2)^2}$, which corresponds to $\phi = \pm 45.289$ degrees.

The intersections between the cones and Zhu's 43 km operating limit sphere occurs at $w = \pm 43000/\sqrt{1+1/(1-e^2)^2}$, which corresponds to $\phi = \pm 66.493$ degrees.

By combining equation (44) with the superellipsoid (30) we find that the intersections between the cones and the superellipsoid (30) occurs at $w = \pm\, ae^2/2\sqrt{2}$, which corresponds to $\phi = \pm 75.048$ degrees.

The loss of accuracy is due to the accumulation of errors when calculating t, which is the real root of this 4th degree equation:

$$F = t^4 + 2it^2 + 2l(m-n)t + k \tag{45}$$

If t is computed with unlimited accuracy then F is obviously zero, but unfortunately this is rarely the case. However, since both F and its derivative can be calculated one may use Newton-Raphson's method to find a correction Δt that can be added to t:

$$dF/dt = 4t^3 + 4it + 2l(m-n) \tag{46}$$

$$\Delta t = -F/(dF/dt) \tag{47}$$

The value of dF/dt is well positive and varies very slowly provided condition (41) is met, thereby ensuring risk free division in equation (47) and excellent error reduction (the overall conversion error may in some cases be reduced more than 100 million times).

To fix the accuracy problem of Zhu's algorithm we may now rewrite equations (11) and (12) as follows:

$$w_1 = w/(t + \Delta t + l) \tag{48}$$

$$z_1 = (1-e^2)z/(t + \Delta t - l) \tag{49}$$

WGS84 Constants

To obtain optimal accuracy for coordinate conversions using 32-bit, 64-bit or 80-bit floating point arithmetic, the following constants can advantageously be used:

$$1/a = 1.56785594288739799723 \times 10^{-7} \quad (50)$$

$$1/a^2 = 2.45817225764733181057 \times 10^{-14} \quad (51)$$

$$l = 3.34718999507065852867 \times 10^{-3} \quad (52)$$

$$l^2 = 1.12036808631011150655 \times 10^{-5} \quad (53)$$

$$1 - e^2 = 9.93305620009858682943 \times 10^{-1} \quad (54)$$

$$(1 - e^2)/a^2 = 2.44171631847341700642 \times 10^{-14} \quad (55)$$

$$(1 - e^2)/b = 1.56259921876129741211 \times 10^{-7} \quad (56)$$

$$H_{min} = 2.25010182030430273673 \times 10^{-14} \quad (57)$$

$$1/\sqrt[3]{2} = 7.93700525984099737380 \times 10^{-1} \quad (58)$$

$$a^2/c = 7.79540464078689228919 \times 10^{7} \quad (59)$$

$$b^2/c^2 = 1.48379031586596594555 \times 10^{2} \quad (60)$$

The last two constants are useful for transforming geodetic coordinates to ECEF coordinates:

$$N = (a^2/c)/\sqrt{\cos^2\phi + b^2/c^2} \quad (61)$$

$$d = (N + h)\cos\phi \quad (62)$$

$$x = d\cos\lambda \quad (63)$$

$$y = d\sin\lambda \quad (64)$$

$$z = (N(1 - e^2) + h)\sin\phi \quad (65)$$

Enhanced Algorithm

The following closed form algorithm for the exact transformation of Earth-Centered Earth-Fixed (ECEF) coordinates (x, y, z) to geodetic coordinates (ϕ, λ, h) is computationally faster, safer and more accurate than Zhu's original algorithm [1]:

$$w^2 = x^2 + y^2 \tag{66}$$

$$l = e^2/2 \tag{67}$$

$$m = w^2/a^2 = (w/a)^2 \tag{68}$$

$$n = z^2(1 - e^2)/a^2 = [(1 - e^2)\, z/b]^2 \tag{69}$$

$$p = (m + n - 4l^2)/6 \tag{70}$$

$$G = mnl^2 \tag{71}$$

$$H = 2p^3 + G \tag{72}$$

$$if\ (H < H_{min})\ then\ abort \tag{73}$$

$$C = \sqrt[3]{H + G + 2\sqrt{HG}}/\sqrt[3]{2} \tag{74}$$

$$i = -(2l^2 + m + n)/2 \tag{75}$$

$$P = p^2 \tag{76}$$

$$\beta = i/3 - C - P/C \tag{77}$$

$$k = l^2(l^2 - m - n) \tag{78}$$

$$t = \sqrt{\sqrt{\beta^2 - k} - (\beta + i)/2} - \text{sign}(m - n)\sqrt{|(\beta - i)/2|} \tag{79}$$

$$F = t^4 + 2it^2 + 2l(m - n)t + k \tag{80}$$

$$dF/dt = 4t^3 + 4it + 2l(m - n) \tag{81}$$

$$\Delta t = -F/(dF/dt) \tag{82}$$

$$u = t + \Delta t + l \tag{83}$$

$$v = t + \Delta t - l \tag{84}$$

$$w = \sqrt{w^2} \tag{85}$$

$$\phi = \arctan(zu, wv) \tag{86}$$

$$\Delta w = w(1 - 1/u) \tag{87}$$

$$\Delta z = z(1 - (1 - e^2)/v) \tag{88}$$

$$h = \text{sign}(u - 1)\sqrt{(\Delta w)^2 + (\Delta z)^2} \tag{89}$$

$$\lambda = \arctan(y, x) \tag{90}$$

Notes:

1. For optimal performance pre-computed constants should be used, replacing divisions by multiplications whenever possible (e.g. divide-by-6 should be replaced by multiply-by-0.166666666…).
2. Feasibility test (73) can be reached using only multiplications, additions and subtractions.
3. Feasibility test (73) uses $H_{min} = e^{12}/4$. This value is required for accurate 32-bit floating point operation. H_{min} can be smaller for 64-bit and 80-bit floating point arithmetic.
4. The particular formulation in (74) ensures that $C > 0$ if $H > 0$.
5. The arctan operators in (86) and (90) take two parameters to avoid divide-by-zero situations.

Test Results

This chapter presents the numerical performance of Zhu's algorithm and the Enhanced algorithm, and compares accuracies for different floating point formats.

Conversion accuracy is measured by transforming Earth-Centered Earth-Fixed (ECEF) coordinates (x_0, y_0, z_0) to geodetic coordinates (ϕ, λ, h), and back to ECEF coordinates (x_1, y_1, z_1). The conversion error is given by $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$.

Measurements were made on an Intel Core i5-4200U CPU, with all coordinate transformation calculations done on the x87 FPU. Trigonometric operations used FSINCOS and FPATAN instructions, whereas root operations were handled with FSQRT, FYL2X and F2XM1 instructions. The same transformation code featuring the sequence (66) to (90) was used for 32-bit, 64-bit and 80-bit floating point formats, but with different FPU configurations. The performance of Zhu's algorithm was measured by setting $\Delta t = 0$ in (83) and (84). The inverse transformation function used is described by the sequence (61) to (65).

Measurements were made with 10 billion randomly generated positions for the different altitude ranges and floating point formats, with latitudes up to ±90 degrees and longitudes up to ±180 degrees.

Alt. min:	Alt. max:	Range Description:
-6378 km	-1 km	Subterrestrial
-1 km	15 km	Terrestrial and Aviation
15 km	100 km	Stratosphere and Mesosphere
100 km	2000 km	Low Earth Orbit
2000 km	35000 km	Medium Earth Orbit
35000 km	37000 km	Geostationary Orbit
0.35 Mkm	0.41 Mkm	Moon
146 Mkm	153 Mkm	Sun

32-bit floating point		Zhu's algorithm		Enhanced algorithm		Zhu/Enhanced ratios	
Alt. min:	Alt. max:	Max. err:	Avg. err:	Max. err:	Avg. err:	Max. err:	Avg. err:
-6378 km	-1 km	1501 m	18.0 m	3.25 m	0.46 m	462	39
-1 km	15 km	1104 m	31.2 m	2.96 m	0.47 m	373	66
15 km	100 km	1176 m	35.4 m	3.41 m	0.52 m	345	68
100 km	2000 km	1264 m	36.2 m	3.24 m	0.51 m	390	71
2000 km	35000 km	1265 m	95.2 m	13.4 m	1.37 m	94	69
35000 km	37000 km	966 m	133 m	13.7 m	2.46 m	71	54
0.35 Mkm	0.41 Mkm	1314 m	332 m	120 m	21.2 m	11	16
146 Mkm	153 Mkm	56.6 km	9.17 km	56.1 km	9.16 km	1	1

64-bit floating point		Zhu's algorithm		Enhanced algorithm		Zhu/Enhanced ratios	
Alt. min:	Alt. max:	Max. err:	Avg. err:	Max. err:	Avg. err:	Max. err:	Avg. err:
-6378 km	-1 km	50.8 mm	85.3 nm	5.84 nm	0.78 nm	8.70 M	109
-1 km	15 km	46.4 mm	139 nm	5.97 nm	0.85 nm	7.77 M	164
15 km	100 km	50.9 mm	192 nm	6.72 nm	1.10 nm	7.57 M	175
100 km	2000 km	52.9 mm	192 nm	6.55 nm	1.07 nm	8.08 M	179
2000 km	35000 km	58.9 mm	612 nm	25.1 nm	2.07 nm	2.35 M	296
35000 km	37000 km	42.0 mm	807 nm	25.6 nm	3.65 nm	1.64 M	221
0.35 Mkm	0.41 Mkm	47.4 mm	7.75 um	217 nm	31.5 nm	218 k	246
146 Mkm	153 Mkm	48.1 mm	1.48 mm	99.5 um	13.2 um	483	112

80-bit floating point		Zhu's algorithm		Enhanced algorithm		Zhu/Enhanced ratios	
Alt.min:	Alt.max:	Max. err:	Avg. err:	Max. err:	Avg. err:	Max. err:	Avg. err:
-6378 km	-1 km	1.14 mm	79 pm	3.18 pm	0.45 pm	359 M	176
-1 km	15 km	1.07 mm	145 pm	3.19 pm	0.52 pm	335 M	279
15 km	100 km	1.09 mm	113 pm	3.38 pm	0.58 pm	322 M	195
100 km	2000 km	1.17 mm	111 pm	3.16 pm	0.58 pm	370 M	191
2000 km	35000 km	1.16 mm	480 pm	13.1 pm	1.55 pm	89 M	310
35000 km	37000 km	1.06 mm	686 pm	13.6 pm	2.70 pm	78 M	254
0.35 Mkm	0.41 Mkm	1.45 mm	8.23 nm	123 pm	23.1 pm	12 M	356
146 Mkm	153 Mkm	1.09 mm	1.19 um	53.9 nm	10.1 nm	20 k	118

Conversion errors are due to accumulation of computational errors during transformations, but also of rounding errors when storing (ϕ, λ, h) into floating point destination variables. Each number must be rounded to the nearest representation offered by the floating point format. The worst case rounding error is half of the floating point resolution for a given number. The floating point resolution of a number is the absolute value change caused by flipping the least significant bit of the number's floating point significand.

The *expected error* at (ϕ, λ, h) is the combined worst case rounding errors for latitude, longitude and altitude, expressed as a geometric distance. Suppose the resolution at (ϕ, λ, h) is $(d\phi, d\lambda, dh)$ and (ϕ, λ, h) transforms to (x, y, z) and $(\phi + d\phi, \lambda + d\lambda, h + dh)$ transforms to $(x + dx, y + dy, z + dz)$, then the expected error is given by $\sqrt{dx^2 + dy^2 + dz^2}/2$. The *expected maximum error* permits distinguishing the performance of the transformation algorithms from the limitations of floating point resolution. The expected maximum error is found at latitude zero, longitude ±180 degrees, and at the highest applicable altitude.

Highest Altitude	Application Area	Expected maximum error		
		32-bit	64-bit	80-bit
10 km	Terrestrial	0.85 m	1.58 nm	0.77 pm
500 km	Orbital	0.92 m	1.71 nm	0.83 pm
35000 km	Geostationary	5.86 m	10.9 nm	5.33 pm
400000 km	Moon	56.4 m	105 nm	51.3 pm
150 Mkm	Sun	20.0 km	37.2 um	18.2 nm

The most commonly used floating point format for applications featuring ECEF and geodetic coordinates is the 64-bit format. Even when using 64-bit floating point numbers for coordinate representation it is nevertheless advantageous to use 80-bit floating point numbers during coordinate transformation to maximize accuracy. This puts the maximum error of the Enhanced algorithm just above the expected maximum error at (ϕ, λ, h).

References

[1] Zhu, J., *Exact Conversion of Earth-Centered Earth-Fixed Coordinates to Geodetic Coordinates.* Journal of Guidance, Control, and Dynamics. Vol. 16, No. 2, March-April 1993.

[2] Zhu, J., *Conversion of Earth-Centered Earth-Fixed Coordinates to Geodetic Coordinates*. IEEE Transactions on Aerospace and Electronic Systems Vol. 30, July 1994.

[3] Clynch, James R., *Radius of the Earth – Radii used in Geodesy*, Naval Postgraduate School, 2002

Revision History

30 October 2019:

The exponent signs are corrected in (59) and (60). Both exponents shall be positive.

Appendix A: Proof Related to Equations (30) and (36)

This appendix proves the following statement:

Combining (30) and (36) shows that $H = 0$ for all points on the superellipsoid.

We have:

$$|w/A|^{2/3} + |z/B|^{2/3} = 1 \quad (30)$$

$$H = e^{12}([(w/A)^2 + (z/B)^2 - 1]^3 + 27(w/A)^2(z/B)^2)/108 \quad (36)$$

We define:

$$W = |w/A|^{1/3} \quad (A1)$$

$$Z = |z/B|^{1/3} \quad (A2)$$

We now have:

$$W^2 + Z^2 = 1 \quad (A3)$$

$$H = e^{12}([W^6 + Z^6 - 1]^3 + 27W^6Z^6)/108 \quad (A4)$$

From (A3) we find:

$$Z^2 = 1 - W^2 \quad (A5)$$

$$Z^6 = (1 - W^2)^3 \quad (A6)$$

$$Z^6 = 1 - 3W^2 + 3W^4 - W^6 \quad (A7)$$

Inserting (A6) and (A7) in (A4):

$$H = e^{12}([-3W^2 + 3W^4]^3 + 27W^6(1 - W^2)^3)/108 \quad (A8)$$

Simplifying:

$$H = e^{12}(-27W^6[1 - W^2]^3 + 27W^6(1 - W^2)^3)/108 \quad (A9)$$

$$H = e^{12}W^6(-[1 - W^2]^3 + (1 - W^2)^3)/4 \quad (A10)$$

Result:

$$H = 0 \quad (A11)$$

Appendix B: Zhu Algorithm Measurements

This appendix contains measurements of Zhu's algorithm.

```
Floating point bits: 32
Tests per altitude range: 10000000000
   alt.min:    alt.max:     avg.err:     max.err:     lat:     lon:         alt:
    -6378km        -1km  1.80299e+01  1.50052e+03  -61.509  -88.567 -6.29931e+06
       -1km        15km  3.12276e+01  1.10439e+03  -45.288  165.253 -5.20297e+02
       15km       100km  3.54009e+01  1.17596e+03   46.870  -87.797  2.14284e+04
      100km      2000km  3.62352e+01  1.26410e+03   45.252   66.613  1.49476e+06
     2000km     35000km  9.52355e+01  1.26458e+03   45.206  105.135  4.73515e+06
    35000km     37000km  1.33357e+02  9.65813e+02   45.119  -32.253  3.50062e+07
   350000km    410000km  3.31897e+02  1.31419e+03   44.982 -153.143  3.51136e+08
146000000km 153000000km  9.17251e+03  5.66051e+04  -22.181  169.920  1.51852e+11


Floating point bits: 64
Tests per altitude range: 10000000000
   alt.min:    alt.max:     avg.err:     max.err:     lat:     lon:         alt:
    -6378km        -1km  8.53082e-08  5.08360e-02  -45.391 -118.404 -2.20681e+06
       -1km        15km  1.38848e-07  4.64195e-02   45.289   88.278  8.62067e+02
       15km       100km  1.92025e-07  5.08651e-02   45.288 -122.376  2.32512e+04
      100km      2000km  1.91761e-07  5.29014e-02   45.251   60.060  1.54804e+06
     2000km     35000km  6.12341e-07  5.88981e-02   45.205    8.664  4.87398e+06
    35000km     37000km  8.07273e-07  4.20093e-02  -45.126  179.204  3.50849e+07
   350000km    410000km  7.74543e-06  4.74019e-02   45.099 -173.572  4.02580e+08
146000000km 153000000km  1.47686e-03  4.81116e-02  -45.096 -151.798  1.46003e+11


Floating point bits: 80
Tests per altitude range: 10000000000
   alt.min:    alt.max:     avg.err:     max.err:     lat:     lon:         alt:
    -6378km        -1km  7.93752e-11  1.13871e-03   45.706  -87.313 -4.35704e+06
       -1km        15km  1.45402e-10  1.07178e-03  -45.288  160.829  9.78769e+03
       15km       100km  1.13273e-10  1.09477e-03   45.288 -107.013  3.55589e+04
      100km      2000km  1.11157e-10  1.16615e-03   45.249  123.885  1.66634e+06
     2000km     35000km  4.80216e-10  1.16183e-03   45.129   81.077  3.12577e+07
    35000km     37000km  6.86386e-10  1.05901e-03  -45.126  -33.259  3.51525e+07
   350000km    410000km  8.22997e-09  1.45377e-03  -45.099  161.388  4.03252e+08
146000000km 153000000km  1.18945e-06  1.08962e-03  -45.130 -146.470  1.46191e+11
```

Appendix C: Enhanced Algorithm Measurements

This appendix contains measurements of Zhu's algorithm enhanced with Newton-Raphson error reduction.

```
Floating point bits: 32
Tests per altitude range: 10000000000
   alt.min:    alt.max:     avg.err:     max.err:     lat:     lon:         alt:
    -6378km        -1km  4.58289e-01  3.25068e+00   14.184  -55.013 -6.26362e+06
       -1km        15km  4.69226e-01  2.95557e+00    1.732 -148.096  4.10438e+03
       15km       100km  5.24492e-01  3.41211e+00    4.943  137.110  9.22772e+04
      100km      2000km  5.14966e-01  3.23596e+00  -84.845  143.421  1.44261e+06
     2000km     35000km  1.37119e+00  1.33836e+01    5.850  149.547  3.47264e+07
    35000km     37000km  2.46794e+00  1.37000e+01  -16.648   64.881  3.68389e+07
   350000km    410000km  2.11506e+01  1.20085e+02  -11.032  135.449  3.91684e+08
146000000km 153000000km  9.16434e+03  5.61039e+04   21.030    8.867  1.49380e+11


Floating point bits: 64
Tests per altitude range: 10000000000
   alt.min:    alt.max:     avg.err:     max.err:     lat:     lon:         alt:
    -6378km        -1km  7.80569e-10  5.84315e-09  -84.909   20.059 -5.21512e+06
       -1km        15km  8.53637e-10  5.96515e-09    1.351  139.599  2.63956e+03
       15km       100km  1.10086e-09  6.71813e-09   -1.644  146.186  5.70166e+04
      100km      2000km  1.07308e-09  6.54519e-09    3.216  175.110  3.37770e+05
     2000km     35000km  2.07138e-09  2.51457e-08   11.375 -153.468  3.47446e+07
    35000km     37000km  3.65036e-09  2.56071e-08  -16.869 -153.367  3.60552e+07
   350000km    410000km  3.14502e-08  2.16964e-07   43.478  -80.559  4.02887e+08
146000000km 153000000km  1.31535e-05  9.94751e-05   -8.960  113.231  1.51844e+11


Floating point bits: 80
Tests per altitude range: 10000000000
   alt.min:    alt.max:     avg.err:     max.err:     lat:     lon:         alt:
    -6378km        -1km  4.49966e-13  3.18313e-12   -1.091  -17.575 -6.27794e+06
       -1km        15km  5.20708e-13  3.19134e-12  -31.764 -156.767  1.23675e+04
       15km       100km  5.79222e-13  3.38063e-12   -1.112 -125.856  5.10078e+04
      100km      2000km  5.83478e-13  3.15958e-12    1.689  161.752  1.37304e+06
     2000km     35000km  1.55298e-12  1.31248e-11   -9.203   57.473  3.41968e+07
    35000km     37000km  2.70145e-12  1.36121e-11   19.423   31.678  3.66195e+07
   350000km    410000km  2.30795e-11  1.23490e-10   -3.595  -45.037  3.88163e+08
146000000km 153000000km  1.00831e-08  5.38559e-08   11.873  160.439  1.51738e+11
```

Appendix D: Reference Code

```
/*************************************************************************\
|* wgs84data.h
\*************************************************************************/

#define WGS84_A        +6.3781370000000000000e+0006  /* a */
#define WGS84_INVF     +2.9825722356300000000e+0002  /* 1/f */
#define WGS84_F        +3.3528106647474807199e-0003  /* f */
#define WGS84_INVA     +1.5678559428873979972e-0007  /* 1/a */
#define WGS84_INVAA    +2.4581722576473318105e-0014  /* 1/(a^2) */
#define WGS84_B        +6.3567523142451794974e+0006  /* b */
#define WGS84_C        +5.2185400842338533240e+0005  /* c */
#define WGS84_E        +8.1819190842621494708e-0002  /* e */
#define WGS84_EE       +6.6943799901413170573e-0003  /* e^2 */
#define WGS84_EED2     +3.3471899950706585287e-0003  /* (e^2)/2 */
#define WGS84_EEEE     +4.4814723452404460262e-0005  /* e^4 */
#define WGS84_EEEED4   +1.2036808631011150655e-0005  /* (e^4)/4 */
#define WGS84_AADC     +7.7954046407868922891e+0007  /* (a^2)/c */
#define WGS84_BBDCC    +1.4837903158659659456e+0002  /* (b^2)/(c^2) */
#define WGS84_P1MEE    +9.9330562000985868294e-0001  /* 1-(e^2) */
#define WGS84_P1MEEDAA +2.4417163184734170064e-0014  /* (1-(e^2))/(a^2) */
#define WGS84_P1MEEDB  +1.5625992187612974121e-0007  /* (1-(e^2))/b */
#define WGS84_HMIN     +2.2501018203043027367e-0014  /* (e^12)/4 */
#define WGS84_INVCBRT2 +7.9370052598409973738e-0001  /* 1/(2^(1/3)) */
#define WGS84_INV3     +3.3333333333333333333e-0001  /* 1/3 */
#define WGS84_INV6     +1.6666666666666666667e-0001  /* 1/6 */
#define WGS84_D2R      +1.7453292519943295769e-0002  /* pi/180 */
#define WGS84_R2D      +5.7295779513082320876e+0001  /* 180/pi */

/*** end of file ***/



/*************************************************************************\
|* wgs84def.h
\*************************************************************************/

#pragma once

// Set to 1 to use degrees in WGS84GEO structure
#define DEGREES 1

// constants to convert between degrees and radians
#define D2R (M_PI/180.0)
#define R2D (180.0/M_PI)

// Design limits
#define MIN_LAT -90
#define MAX_LAT  90

#define MIN_LON -360
#define MAX_LON  360

typedef struct
{
    double x, y, z;
} WGS84ECEF;

typedef struct
{
    double lat, lon, alt;
} WGS84GEO;

/*** END OF FILE ***/
```

```
/*****************************************************************************\
|* wgs84conv.h
\*****************************************************************************/

#pragma once

#include "wgs84def.h"

int wgs84GeoToEcef(WGS84ECEF * ecef, WGS84GEO * geo);
int wgs84EcefToGeo(WGS84GEO * geo, WGS84ECEF * ecef);

/*** END OF FILE ***/



/*****************************************************************************\
|* wgs84conv.c
\*****************************************************************************/

#include <windows.h>
#include <math.h>

#include "wgs84def.h"
#include "wgs84data.h"
#include "wgs84conv.h"


const static double invaa = WGS84_INVAA;            // 1/(a^2)
const static double aadc = WGS84_AADC;              // (a^2)/c
const static double bbdcc = WGS84_BBDCC;            // (b^2)/(c^2)
const static double l = WGS84_EED2;                 // (e^2)/2
const static double p1mee = WGS84_P1MEE;            // 1-(e^2)
const static double p1meedaa = WGS84_P1MEEDAA;      // (1-(e^2))/(a^2)
const static double Hmin = WGS84_HMIN;              // (e^12)/4
const static double ll4 = WGS84_EEEE;               // 4*(l^2) = e^4
const static double ll = WGS84_EEEED4;              // l^2 = (e^4)/4
const static double invcbrt2 = WGS84_INVCBRT2;      // 1/(2^(1/3))
const static double inv3 = WGS84_INV3;              // 1/3
const static double inv6 = WGS84_INV6;              // 1/6
const static double d2r = WGS84_D2R;                // PI/180
const static double r2d = WGS84_R2D;                // 180/PI


/*****************************************************************************\
|* Convert geodetic coordinates to ECEF coordinates
\*****************************************************************************/
int wgs84GeoToEcef(WGS84ECEF * ecef, WGS84GEO * geo)
{
    double lat, lon, alt;
    double N;
    double coslon, sinlon;
    double coslat, sinlat;
    double d;

#if (DEGREES)
    lat = d2r * geo->lat;
    lon = d2r * geo->lon;
#else
    lat = geo->lat;
    lon = geo->lon;
#endif
    alt = geo->alt;

    coslat = cos(lat);
    sinlat = sin(lat);
    coslon = cos(lon);
    sinlon = sin(lon);
```

```
    N = aadc / sqrt(coslat * coslat + bbdcc);
    d = (N + alt) * coslat;
    ecef->x = d * coslon;
    ecef->y = d * sinlon;
    ecef->z = (p1mee * N + alt) * sinlat;

    return 0;
}


/*****************************************************************************\
|* Convert ECEF coordinates to geodetic coordinates
\*****************************************************************************/
int wgs84EcefToGeo(WGS84GEO * geo, WGS84ECEF * ecef)
{
    double x, y, z;
    double lat, lon, alt;

    // The variables below correspond to symbols used in the paper
    // "Accurate Conversion of Earth-Centered, Earth-Fixed Coordinates
    // to Geodetic Coordinates"
    double beta;
    double C;
    double dFdt;
    double dt;
    double dw;
    double dz;
    double F;
    double G;
    double H;
    double i;
    double k;
    double m;
    double n;
    double p;
    double P;
    double t;
    double u;
    double v;
    double w;

    // Intermediate variables
    double j;
    double ww;                  // w^2
    double mpn;                 // m+n
    double g;
    double tt;                  // t^2
    double ttt;                 // t^3
    double tttt;                // t^4
    double zu;                  // z * u
    double wv;                  // w * v
    double invuv;               // 1 / (u * v)
    double da;
    double t1, t2, t3, t4, t5, t6, t7;

    x = ecef->x;
    y = ecef->y;
    z = ecef->z;

    ww = x * x + y * y;
    m = ww * invaa;
    n = z * z * p1meedaa;
    mpn = m + n;
    p = inv6 * (mpn - ll4);
    G = m * n * ll;
    H = 2 * p * p * p + G;
```

```
    if (H < Hmin)
    {
        return -1;
    }

    C = pow(H + G + 2 * sqrt(H * G), inv3) * invcbrt2;
    i = -ll - 0.5 * mpn;
    P = p * p;
    beta = inv3 * i - C - P / C;
    k = ll * (ll - mpn);

    // Compute left part of t
    t1 = beta * beta - k;
    t2 = sqrt(t1);
    t3 = t2 - 0.5 * (beta + i);
    t4 = sqrt(t3);
    // Compute right part of t
    t5 = 0.5 * (beta - i);
    // t5 may accidentally drop just below zero due to numeric turbulence
    // This only occurs at latitudes close to +- 45.3 degrees
    t5 = fabs(t5);
    t6 = sqrt(t5);
    t7 = (m < n) ? t6 : -t6;
    // Add left and right parts
    t = t4 + t7;

    // Use Newton-Raphson's method to compute t correction
    j = l * (m - n);
    g = 2 * j;
    tt = t * t;
    ttt = tt * t;
    tttt = tt * tt;
    F = tttt + 2 * i * tt + g * t + k;
    dFdt = 4 * ttt + 4 * i * t + g;
    dt = -F / dFdt;

    // compute latitude (range -PI/2..PI/2)
    u = t + dt + l;
    v = t + dt - l;
    w = sqrt(ww);
    zu = z * u;
    wv = w * v;
    lat = atan2(zu, wv);

    // compute altitude
    invuv = 1 / (u * v);
    dw = w - wv * invuv;
    dz = z - zu * p1mee * invuv;
    da = sqrt(dw * dw + dz * dz);
    alt = (u < 1) ? -da : da;

    // compute longitude (range -PI..PI)
    lon = atan2(y, x);

#if (DEGREES)
    geo->lat = r2d * lat;
    geo->lon = r2d * lon;
#else
    geo->lat = lat;
    geo->lon = lon;
#endif
    geo->alt = alt;

    return 0;
}


/*** END OF FILE ***/
```