Annex B
(informative)

Implementation notes

B.1 Introduction

This informative annex provides advisories relative to the implementation of the spatial operations contained in this International Standard. It also contains illustrations of two time-dependent embedding transformations.

B.2 General observations

B.2.1 Finite precision

It is generally not possible to exactly implement theoretical formulations on a digital computer due to limitations in representing real numbers on a finite word length computer. If x is a real number, its representation on a digital computer can be viewed as x_c. The difference between x and x_c is called *digitization error*. There are some real numbers that can be exactly represented but generally the digital representation is only good to a prescribed number of bits depending on the precision of the floating-point representation of the computer system used. Implementation of spatial operations can involve relatively large numbers. Differences of numbers with large absolute values can occur along with products of relatively small numbers with large numbers. Loss of significance can occur due to these conditions.

EXAMPLE Using single precision arithmetic for SRFs associated with the Earth may lead to a loss of precision on the order of half a metre even when the application is for the near Earth region.

NOTE To mitigate loss of precision it is advisable to employ double precision [IEC 60559] arithmetic for floating-point operations.

B.2.2 Computational efficiency

In many application domains computational efficiency is very important. Some examples of such applications include: embedded systems with real time control feed-back, the processing of large numbers of very large environmental data files, real time graphics display of geographic data and large scale simulations involving hundreds of thousands of interacting objects. Historically, computational assets were much less capable than those currently available. As a result, much research over the last century has been devoted to reducing the computational complexity for the type of spatial operations contained in this International Standard. Many of the techniques currently used were developed for hand computation or in the context of more rudimentary computational systems. Implementers have been slow to adapt to the capabilities provided by computational systems that currently exist. Concomitant with the increased computational capabilities there have been significant technical advances in the field of computational mathematics. New methods have emerged along with better strategies for exploiting the current computational capabilities. These advances in computational mathematics have generally not been exploited for the types of spatial operations within the scope of this International Standard.

The strategy for selecting algorithms for implementation is dependent on the intended application. For a general service system, where an interactive user needs a few spatial operations computed, efficiency is becoming much less important. Current machines are so fast that humans cannot perceive the difference between very fast machines and very slow ones. For such application domains the choice of algorithm is not critical as long as it is accurate, reliable and covers the domain of interest.

For computationally intense applications most of the mathematical formulations contained in this International Standard are not appropriate for implementation. Some of the closed-form solutions may be unacceptably inefficient and may be replaced by various approximate methods.

EXAMPLE Most implementations of the inverses for map projections are implemented with finite series methods in order to avoid using potentially inefficient iterative methods.

B.2.3 Approximation

The replacement of theoretical formulations with approximations made to increase computational efficiency introduces an error. The difference between the true value x and the approximation value x_a is the *approximation error*. The implementation of an approximation using a double precision representation includes both the digitization and approximation errors. The combination of these errors is called the *computational error.* However, the magnitude of the approximation error usually dominates that of the digitization error and therefore the digitization error may generally be ignored.

The acceptable computational error is application dependent. Increased capabilities of real measurement systems and improved SRF models have led to increased requirements for more stringent error tolerances. In high-resolution simulation applications the requirement is to keep the computational error in position as small as 1 millimetre. Increased system accuracy requirements coupled with efficiency requirements place a considerable premium on development and use of efficient algorithms. Given the variability in computer system characteristics and application domain accuracy requirements there is no single solution that fits all cases.

Subsequent clauses provide a set of general guidelines for algorithm designers and software developers that are intended to broaden their conceptual approach to implementations. These guidelines are specific to Earth-related spatial operations but most of them are applicable to the more general case.

B.3 Guidelines for algorithm development for spatial operations

B.3.1 Introduction

Many computational algorithms have been developed for spatial operations processing for a wide range of applications. Many of these are not appropriate for efficient processing using current computer system environments. If an application domain does not require efficient processing, any accurate algorithm for computing spatial operations may be employed. In such cases, it is recommended that closed-form solutions be employed when available, and iterative procedures otherwise.

This clause includes a set of guidelines or advisories for use in designing *efficient* algorithms. While the target environment is generally a computer system with a super-scalar architecture, many of these advisories are applicable to legacy computer systems and specialized systems used for embedded processing. Most of the advisories are applicable to spatial operations processing for celestial bodies other than the Earth.

B.3.2 The computational environment

The properties of the computational environment should be taken into account. In recent decades a significant improvement in computational capabilities has occurred. Yet, in some application domains, algorithms that were developed for hand calculation are still being implemented. In addition, many traditional computational methods developed for legacy computers are inappropriate for the new environment but continue to be used.

The principal characteristics of the new computational environments include:

a) readily-available low-cost Dynamic Random Access Memory (DRAM),

b) development of very high-speed cache memory that permits the dynamic allocation of blocks of critical data to the processor cache memory,

c) super-scalar architectures that permit pipelined (parallel) processing,

d) integrated processors that permit very high-speed processing for some critical mathematical functions (e. g. some transcendental functions and square root),

e) development of compilers that exploit the advantages of super-scalar architectures,

f) development of optimizing operating systems that re-order computations and memory accesses in real time to increase efficiency, and

g) integrated support for Institute of Electrical and Electronics Engineers (IEEE) double precision floating-point representation in which the mantissa is 52 bits (this is equivalent to 15 plus decimal digits of accuracy, (see also [IEC 60559])).

An example of the impact of these changes is in the computation of trigonometric functions. In the legacy computational environment, trigonometric functions were evaluated as system-level subroutines written in software. Such routines were very slow sometimes taking between 30 and 45 floating-point operations to complete. To meet system timeline specifications, software developers often replaced trigonometric subroutine calls by in-line procedures that used piecewise linear or quadratic trigonometric calculations (colloquially called “table lookup”). This required a considerable portion of the relatively small memory available on legacy computers. Because memory was scarce at the time, this technique could only be used sparingly. Accuracy was often degraded so that the array of fitting coefficients did not get too large.

In the current computational environment the trigonometric functions are computed in special processors using high-speed memory and parallel processing. As a result, these functions produce double precision results very quickly relative to the basic central processor cycle time of the computer. In particular, a sine function call executes in a processing time equivalent to 8 to 10 floating-point operations. Square root calls are even faster. On some computers, implementing a general in-line sine sub-routine (by table lookup) may actually be slower than calling the system sine function. This can happen because the access time required to fetch the appropriate fitting coefficients from dynamic random access memory may take longer than the entire system routine computation. On the other hand, for modern machines where memory is virtually unlimited, it is possible to develop in-line algorithms for the standard transcendental functions with accuracies approaching that of double precision. Carefully designed procedures based on piecewise continuous approximations can be developed for this purpose.

The development of in-line code for general-purpose calculation of standard mathematical routines is also useful for reducing the execution time of compound functions or mathematical functions that are not in the system library. In particular, it may be more efficient to evaluate $\sin(f(x))$ in-line rather than computing $f(x)$ and then calling the sine function. The efficacy of the in-line approach in such a case depends on the nature of $f(x)$.

B.3.3 Domain of application

The domain of applicability should be defined *before* developing or selecting an algorithm. Algorithm designers and software developers may expend valuable development and computational time forcing their methods to work in non-practical regions such as near the centre of an ERM or at ellipsoidal heights of 100 million kilometres from the surface. The number of applications for such regions is small (or zero) and the interest in these regions is primarily academic. For an Earth referenced application there are several regions of practical interest

EXAMPLE 1 For aircraft, an appropriate region in terms of ellipsoidal height is –200 metres to 35 000 metres for all latitudes and longitudes. This covers the region where air-breathing vehicles can operate.

EXAMPLE 2 For sub-surface operations an appropriate region is –12 000 metres to +200 metres for all latitudes and longitudes. This region covers the lowest bathymetric point of the Earth (Marianas Trench) to slightly above the ocean’s

surface. It is convenient to combine the two regions of these examples and to call the composite region, the near Earth region.

EXAMPLE 3 Space operations may require a region extending above 35 kilometres to beyond the orbit of the moon. This region will be called the space operations region.

All regions may be further divided into sub-regions for particular applications in order to simplify formulations or for computational efficiency. Usually the latitude domain in this application is $[-\pi/2, \pi/2]$ and the longitude domain is $(-\pi, \pi]$. On occasion, a particular application may be restricted to a smaller latitude/longitude region in order to simplify formulations and in the case of map projections to reduce distortions.

B.3.4 Define a meaningful error measure

In many situations involving spatial operations computation, the resulting variables are not exact due to approximations made in the computational formulations. An error measure is needed to determine the approximation error. If the target variables are in terms of distance in a Euclidean coordinate system, a Euclidean metric can be used to measure the approximation error. Such an error measure is called *position error*. Often the maximum error in the absolute value of the difference between the true values and the approximate values of each component is used.

The average value of the magnitude of the differences of each component of position error has also been used as an error measure. This practice makes the approximation errors appear to be much smaller than the maximum errors, and depends on where the samples for the average are collected. This approach is misleading and should not be used.

Sometimes the target variables contain angular errors along with distance errors. In this case the angular error should be converted to distance so that a Euclidean error measure can be applied. Some variables, such as point distortion are unit-less and the resulting computational error is unit-less.

B.3.5 Avoid excessive computational accuracy

The literature on methods for spatial operations processing contains many algorithms that are excessively accurate. One paper on geocentric to geodetic coordinate conversion develops a procedure where the approximation error is 10^{-20} metres, an accuracy far exceeding any practical use. Many iterative procedures can achieve such accuracies provided that a computational environment is available with sufficiently high precision arithmetic. However, it is important not to waste computer cycles to attain superfluous accuracy.

EXAMPLE A method, A, with maximum error 10^{-8} m is sometimes declared superior to a method, B, which has a maximal error of 10^{-6} m. If method A takes more processing time than method B, it is not superior. In fact it is quite likely that both methods are too accurate. Suppose there is a method C with maximum error less than 10^{-4} metres but takes less computer time than A or B. Then method C would likely be preferable for most (if not all) applications.

B.3.6 Determine the acceptable error before starting

The maximum allowable position error should be determined before starting an algorithm development. Note that, when position error is small, its component errors are also very small. When a position error criterion is used, the nominal position error is usually much smaller than the maximum error. In some applications, such as in simulations, it is important to keep the computational error very small, as small as 1mm. It is difficult to conceive of an application domain that requires, or would find useful, errors smaller than this for spatial operations. This is particularly the case if scarce computational resources are used in order to achieve superfluous accuracy.

B.3.7 Mathematical approaches

Mathematical formulations for spatial operations processing can be relatively complex. This complexity is driven by the fact that many SRFs of interest are based on oblate ellipsoid ORMs. This results in formulations

in which the principal equations involved are non-linear and sometimes not solvable in closed form (e.g., geodesic distance). For most spatial operation formulations it is also necessary to have an inverse spatial operation. Many spatial operation formulations have closed-form solutions in one direction but do not have closed-form solutions for the inverse. This situation leads to a requirement to solve multivariate non-linear equations where no closed solution is readily available.

Traditionally, either truncated power series, or iterative solutions have been used for solving spatial operation computation problems. Power series solutions are almost always inferior to well-designed iterative solutions from an efficiency point of view. Both of these methods have an interesting property that is often not recognized. Almost all existing implementations of truncated power series solutions use all the terms in the series no matter how close the independent variables are to the expansion point. In fact, when the independent variables are close to the expansion point only a few terms are needed and the effect of higher-order terms is vanishingly small. It is often easy to develop simple tests on the independent variables to determine how many terms to use in a particular formulation. A similar situation exists in determining how many iterations to perform in an iterative approach. The maximal number of iterations required to achieve a required accuracy over some domain can be determined when testing an implementation. The implementation of the iterative procedure can then use a fixed number of iterations. This avoids excessive iteration and avoids the need for a termination test (which is usually computationally expensive). Legacy software designs often use the maximum number of terms or iterations, regardless. This is often a significant waste in computation time.

Another approach for solving multivariate non-linear equations is much more appropriate in the new computational environment. This is the use of curve fitting or approximation of a function or the inverse of a function. In its simplest form, this amounts to piecewise approximation or “table lookup”. Historically, the perceived penalty associated with this approach is that it takes too much memory to store the coefficients of the piecewise-defined functions to achieve usable accuracy. This penalty has been virtually eliminated by the availability of large capacity low-cost dynamic random access memory. The trend in computational mathematics is to use low-order local approximations for efficiency.

B.3.8 Good programming and formulation practices

Experienced programmers usually employ good programming practices. They move the computation of global constants out of embedded loops to start up procedures, move locally computed constants to the highest possible level, nest polynomials, avoid using power functions and leverage many other good practices. Unfortunately, some universities now teach that these practices are not important because modern computers are so fast that they are not needed, or that optimising compilers will invoke such good practices automatically. In complex algorithms it may not be easy or possible for a compiler to clean up poor practices, so it is advisable to always use good programming practices in situations where performance is important.

In many cases the formulation implemented is the published form of the mathematical formulation. Often the author of the formulation is not familiar with computational mathematics or has chosen a mathematical formulation for publication that is unambiguous and convenient for exposition. Often the efficiency of the formulation can be greatly improved by eliminating redundant transcendental functions. In particular, trigonometric functions can sometimes be eliminated or simplified through the use of identities or just simple observations.

One simple illustration is a test such as sqrt(x) < a, used as a branch point test. If a is a constant, this can be re-written in the equivalent form, $x < a \cdot a$. This observation generalizes to tests in the form $f(x) < a$ that may become $x < f^{-1}(a)$ (a constant) in cases where this makes mathematical sense.

More complex examples have lead to publications whose conclusions about convergence rate and processing time requirements are incorrect. A classic case is in the conversion of geocentric coordinates to geodetic 3D coordinates using the originally published form of Bowring’s method [BOWR]. If the algorithm is formulated as published, it appears that several intermediate trigonometric function evaluations are needed for each iteration. In fact, only a square root is needed for each iterate [TOMS], [FUKU]. Fukushima has coined an appropriate term for the direct implementation of the published form of Bowring’s method. He calls this a

"naive implementation". This appellation can be applied to many spatial operation formulations where the direct implementation of the published form of the formulation is often naive.

B.3.9 Design in context

Spatial operation computations are usually not performed in isolation. More often they are used as part of a sequence of spatial operations. In particular, a coordinate in the geocentric SRF is converted to a geodetic 3D SRF and this is immediately followed by a conversion to a map projection SRF. Such a sequence is sometimes referred to as a chain of spatial operations. By anticipating a chain, some of the early calculations may be saved for use later in the chain. Such chains often occur in simulations and embedded systems.

EXAMPLE When the above chain includes Transverse Mercator, the curvature in the prime vertical, the sine of latitude, the cosine of latitude and the trigonometric functions of longitude will be needed to support the conversion.

Often when simulating dynamics models the values of latitude and longitude are not needed at all, only the trigonometric functions of these are used. Some procedures for converting geocentric coordinates to geodetic coordinates accurately compute these variables as part of the process and they can be saved for future use in the chain. Other techniques do not compute these accurately and only supply accurate angular values for latitude and longitude. If the trigonometric functions of these are needed, they have to be computed by calling at least a cosine function followed by a square root (or a sine function call). When selecting algorithms, preference should be given to those approaches that are useful for efficient chaining.

B.3.10 Software verification and computational error testing

Verification testing involves determining if an implementation properly reflects the mathematical formulation of the problem. Computational error evaluation is the determination of computed position error (or another appropriate error measure) over the predetermined region of interest. Much of verification testing is done by inspection. Rapid changes in computational error or unexpected results on some subset of the area of interest often indicate a formulation or implementation error. As a result, verification testing is aided by computational error testing if the domain of test points is dense enough.

Computational error testing often uses results obtained from external sources, usually from authoritative agencies. In fact it is always helpful to compare results to other implementations developed for spatial operations. Unfortunately, such authoritative data may be sparse or not cover the full range of the area of interest. Fortunately, many spatial operation formulations have closed-form solutions in at least one direction. Closed-form solutions may not be very efficient and approximation algorithms may be needed to provide high-speed solutions. However, closed-form solutions are very useful for constructing reliable data sets for testing.

In most cases it is very difficult to exactly determine the maximum computation error. In some cases, mathematical analysis can provide analytic bounds to the error. However, these may not relate well to what is actually implemented and the vagaries of computation with a finite word length. In the end, the error analysis should be accomplished with respect to the procedures as implemented. Consequently, it is desirable to test an algorithm on a very large set of points uniformly distributed over the region of interest. This is commonly referred to as dense testing. The set of test data points themselves should be automatically generated over a region or sub-region of interest. A convenient way to do this is to define a lattice of exact reference data points. Such a lattice can be in one or more dimensions and is often referred to as a gridded data set. This lattice is used in conjunction with a closed-form solution to develop a corresponding reference data set for the processed spatial operation. This set is taken to be exact if double precision arithmetic is used and assumes that no implementation or formulation errors are made. A procedure then can be developed to test the implementation of an approximate algorithm at each grid point. This procedure determines the absolute approximation error at each point and the maximum of all these errors is computed for the whole grid. As the grid size is made smaller the maximum approximation error should converge to a fixed value. This fixed value will closely approximate the maximum approximation error over the region of interest.

Occasionally a test is developed where a spatial operation is computed for a given point and this is followed by the computation of the inverse of the operation. The spatial operation is then re-computed for the resulting point and the inverse is applied to this new point. Continuing this process a large number of times will

generally lead to divergence of the results. This is to be expected. Both round-off error and approximation error are generally present in each direction. These normally do not cancel each other so that the approximation error will grow. Such tests are meaningless for validating a spatial operation.

The lack of symmetry of the error between an approximate spatial operation process and its inverse can also cause difficulties when points are on or very near a validity boundary. Application developers need to be aware of this type of problem and to provide guards against this type of behaviour.

EXAMPLE When a surface geodetic coordinate represents a point near or on a zone boundary for a UTM SRF set member, the UTM coordinate is usually computed using a truncated power series. Due to the combination of approximation and digitization error the resulting UTM coordinate may be in an adjacent UTM zone. Applying the inverse spatial operation may yield a surface geodetic coordinate in that adjacent UTM zone and not in the original UTM zone. This is due to the lack of symmetry of the forward and inverse computational errors.

B.3.11 Singularities and near singularities

Certain points may represent points of singularity in a formulation. Values at the singularity are usually determined analytically and the mathematical formulation accounts for the singularity. In computing values close to a singularity, numerical sensitivities can occur. In the neighbourhood of such a point, dense testing should be performed to ensure that proper results are obtained throughout the neighbourhood.

B.3.12 Performance testing

Since most of these guidelines are addressing efficiency, it is important to consider performance testing. As the computational environment has evolved with time to its current state, performance testing has become increasingly difficult to conduct. The vast majority of legacy and current literature on algorithm development for spatial operation processing has relied on counts of both arithmetic operations and system-level mathematical function calls to estimate performance. This policy generally ignores the system-to-system variation of the performance of arithmetic operations and mathematical functions. At best this policy is only valid for making relative comparisons for the same computational environment. Even then, operation counts are only reliable when the difference in operation counts is large. Obviously, if one method requires many more operations, transcendental functions, and square roots than another it probably will be slower than an alternative that requires less of these. However, determining a percent difference in performance with this approach is apt to be imprecise.

Another approach is often taken to performance testing. This consists of developing a simple program in which the algorithm is embedded in a loop. The loop execution time is determined by executing an empty loop a large number of times using a system-level timer. The algorithm to be tested is then embedded in the loop and execution time is determined again. The time difference of the two processes is divided by the number of loop cycles and the result is used as an estimate of the execution time of the algorithm. This allows comparisons between alternative algorithms to be made on the same computer. Even on legacy systems some care should be taken with this technique. If the algorithm is very simple and if an optimising compiler is used, sometimes setting the optimiser to a high level can produce erroneous results. Thus, if the procedure is $x = \sin(a)$, the compiler may recognize that the result is the same for every pass through the loop and move the computation outside the loop.

Simple loop tests like the one discussed in the previous paragraph are likely to be unreliable for predicting performance when implemented on a machine with a super-scalar architecture. An algorithm's performance may be quite different when it is embedded in a large application program.

The use of a simple loop test is called *in vitro* testing while testing an algorithm embedded in a larger application program is called *in vivo* testing. This is in analogy with *in vitro* testing in biological medicine where an experiment is generally small and controlled, such as being limited to a Petri dish and conducted within a controlled environment. In the biological medicine context, *in vivo* testing is done within its natural environment, perhaps in a living body.

In vitro tests are much less reliable in the new computer environment because an operating system will be able to concentrate all of its capabilities on a relatively small code to make optimal use of cache memory and parallelism. The same algorithm, tested *in vivo* will have competition from other processes for computational resources. When comparing the performance of two algorithm options even their relative performance may not be preserved when transitioning from *in vitro* to *in vivo* testing. Obviously, *in vivo* testing may not be possible in a development program. The code in which the algorithm is to be embedded may not even exist until late in the program. This suggests that initial tests be done by inspection (operation counts) and by *in vitro* testing with the understanding that the results are not precise. When enough of the product software is available, re-testing *in vivo* is recommended.

B.4 Spherical reference models

All of the mathematical formulations for spatial operations with respect to SRFs based on an oblate ellipsoid ORM will be valid when the eccentricity of the ellipsoid is set to zero. That is, they are valid for spherical reference models. However, the majority of spatial operations for the spherical case have formulations that are available in closed form. It may be more convenient for some application domains to use the closed-form solutions in this case.

B.5 Practical considerations

B.5.1 Distortion considerations

For map projections, distortion effects generally increase with distance from the origin. Distortions may become unacceptably large for a particular application domain. In this case the distortion error dominates the computational approximation error. As a consequence, it is not useful to develop algorithms that have minimal computation error in such regions. In practice an implementation may be designed to prevent processing the projection in this case or it may do so but issue a warning that distortions are large.

B.5.2 Validity checking

An implementation should verify that both input and output data for a spatial operation are in the proper domain and range.

In some spatial operations the domain of the implementation may be restricted to avoid computations near singular points.

When using even a convergent iterative routine, a computation near a pole may result in a latitude slightly exceeding 90 degrees. The developer needs to test for this case and set the result to 90 degrees.

B.6 The Bursa-Wolfe equation

The seven-parameter transformation Equation (5) involves many multiples of sines and cosines. In many ERMs of interest, the rotations are very small. When a rotation ω is very small, both $\sin(\omega) = \omega$ and $\cos(\omega) = 1$ hold to a very close approximation (when expressed in radians)[32]. With this simplification, Equation (5) applied to transforming a source normal embedding S to a target normal embedding T can be approximated without any trigonometric functions (in the position vector rotation convention) by the *Bursa-Wolfe equation*:

[32] When $\omega = 1''$, the error multiplied by the radius of the Earth is less than 75 μm. In general, $|\omega - \sin(\omega)| < \frac{1}{2}\omega^2$, when ω is expressed in radians.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{T} \simeq \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}_{S,T} + (1+\Delta s)\begin{pmatrix} 1 & -\omega_3 & \omega_2 \\ \omega_3 & 1 & -\omega_1 \\ -\omega_2 & \omega_1 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{S} \quad \text{(B.1)}$$

When the rotations are given in the coordinate frame convention, the rotation terms in the matrix in Equation (B.1) reverse their sign.

NOTE In the case of ERMs the published values for Δs, when non-zero, are typically one part per million in magnitude (10^{-6}) or smaller. Combined with the assumption of very small rotations and dropping second-order terms of the form $\omega \cdot \Delta s$, Equation (B.1) in the coordinate frame convention is in the following form[33]:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{T} \simeq \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}_{S,T} + \begin{pmatrix} 1+\Delta s & \omega_3 & -\omega_2 \\ -\omega_3 & 1+\Delta s & \omega_1 \\ \omega_2 & -\omega_1 & 1+\Delta s \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{S}$$

B.7 Time dependent reference transformation examples

B.7.1 Introduction

Two reference transformations are provided here to illustrate the reference transformation $\boldsymbol{H}_{SR}(t)$ that appears in 10.3.2. The two provided are ERMs based on OBRSs.

B.7.2 Notation and terminology

The *Greenwich sidereal hour angle* $\theta_{GSH}(t)$ is the angle in radians from the first point of Aries to the direction of the x-axis of ORM WGS_1984 Earth reference ORM. The Greenwich sidereal hour angle depends on the epoch of definition of the first point Aries and is a function of UTC time t elapsed from a given epoch.

The *obliquity of the ecliptic* $\varepsilon(t)$ is the angle in radians from the equatorial plane to ecliptic. The obliquity of the ecliptic is a function of UTC time t elapsed from a given epoch.

The *ecliptic longitude of the Sun* $\lambda_{\odot}(t)$ is the angle in radians from the first point of Aries and the line from the centre of the Earth to the centre of the Sun. The direction to the Sun is represented by $\lambda_{\odot}(t)$. The ecliptic longitude of the Sun depends on the epoch of definition of the first of point Aries and is a function of UTC time t elapsed from a given epoch.

NOTE Approximations of $\theta_{GSH}(t)$, $\varepsilon(t)$, and $\lambda_{\odot}(t)$ are published in astronomical almanacs and other documents (see [SEID] or [USNOA]).

B.7.3 Earth equatorial inertial reference transformation

Given an ERM in the equatorial inertial OBRS, if it is assumed that the ERM z-axis and ORM WGS_1984 z-axis are coincident, then the transformation $\boldsymbol{H}_{EI,WGS84}(t, \boldsymbol{p})$ from the ERM embedding to the ORM WGS_1984 reference ORM embedding is given at time t by:

[33] In [ISO 19111] this approximation is called the generic seven parameter formula for the similarity transformation.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{WGS84}} = \boldsymbol{H}_{\text{EI,WGS84}}\left(t, \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{EI}}\right) = \begin{pmatrix} \cos\left(\theta_{\text{GSH}}(t)\right) & \sin\left(\theta_{\text{GSH}}(t)\right) & 0 \\ -\sin\left(\theta_{\text{GSH}}(t)\right) & \cos\left(\theta_{\text{GSH}}(t)\right) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{EI}}. \quad \text{(B.2)}$$

B.7.4 Solar ecliptic reference transformation

In the case of Earth (see [HAPG]), the transformation $\boldsymbol{H}_{\text{SE,WGS84}}(t, \boldsymbol{p})$ from solar ecliptic ORM embedding coordinates to ORM WGS_1984 reference embedding coordinates at time t is:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{WGS84}} = \boldsymbol{H}_{\text{SE,WGS84}}\left(t, \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{SE}}\right) = \boldsymbol{R}_Z \boldsymbol{R}_X \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\text{SE}} \quad \text{(B.3)}$$

where

$$\boldsymbol{R}_X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\left(\varepsilon(t)\right) & -\sin\left(\varepsilon(t)\right) \\ 0 & \sin\left(\varepsilon(t)\right) & \cos\left(\varepsilon(t)\right) \end{pmatrix}, \text{ and}$$

$$\boldsymbol{R}_Z = \begin{pmatrix} \cos\left(\theta_{\text{GSH}}(t) - \lambda_{\odot}(t)\right) & \sin\left(\theta_{\text{GSH}}(t) - \lambda_{\odot}(t)\right) & 0 \\ -\sin\left(\theta_{\text{GSH}}(t) - \lambda_{\odot}(t)\right) & \cos\left(\theta_{\text{GSH}}(t) - \lambda_{\odot}(t)\right) & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$