20 February 2006

# GUIDELINES FOR THE DEVELOPMENT OF EFFICIENT ALGORITHMS FOR SPATIAL OPERATIONS

Prepared by:

Ralph Toms, Senior Technical Advisor
Instrumentation and Simulation Program

and

Cameron Kellough, Research Engineer
Intelligence and Information Systems

Approved:

Don C. Arns, Jr.
Program Manager, Engineering & Systems Division

# I INTRODUCTION

## 1.1 Background

This report contains guidelines for the development of computationally efficient algorithms for computing spatial operations. A spatial operation is a coordinate transformation, a coordinate conversion, an azimuth determination, a distance calculation or other computations associated with elliptical trigonometry and map projections. These guidelines have evolved as the result of several projects that have emphasized efficient processing. One of these was the SEDRIS program, covering the period 1996 to the present [SEDR]. The SEDRIS program involves standardized representations of environmental data for Live, Virtual and Constructive (LVC) simulations.

LVC simulations must execute in near real time or faster. The archived environmental data is saved in very large files in a multitude of spatial reference frames (coordinate systems and Earth reference models). Such data must be converted to spatial reference frames consistent with the simulation nodes of a distributed federation. Changes to the simulation scenario require rapid turnaround of these conversions. Conversion from one spatial reference frame to another also must be done efficiently when developing or receiving protocols in a distributed federation. Spatial operation computations are also required internal to a simulation node or in embedded systems for geometry modeling and dynamics. As a consequence of such applications, a premium is placed on processing efficiency, while at the same time, maintaining challenging accuracy requirements.

A number of ISO/IEC International Standards have been developed as part of the SEDRIS program. In addition, software has been implemented for computing spatial operations that conform to these standards [SEDR]. This software is open and freely available to users from at the SEDRIS web site [SEDR]. For the SEDRIS program, the authors have focused on the development of a Spatial Reference Model (SRM), algorithms for the efficient computation of spatial operations and code for use in the SEDRIS implementation.

The SRM has evolved over the life of the program into an ISO/IEC International Standard, which is currently at the Final Draft International Standard (FDIS) state [18026]. One of the authors of this report has served as a co-editor for ISO/IEC 18026. The FDIS was released to the international community for ballot in January of 2006. The SEDRIS implementation conforms to several ISO/IEC Standards developed for the SEDRIS program. In particular, the implementation meets the computational accuracy requirement, in position, of 1mm for the computation of a set of spatial operations as specified in the conformance clause of ISO/IEC 18026 (FDIS). There is no performance requirement specified in ISO/IEC 18026. However, for the reasons stated above a premium is placed on processing efficiency, while at the same time, maintaining the challenging accuracy requirements of ISO/IEC 18026.

One of the goals of the SEDRIS program has been to educate users in the complexities of spatial referencing across numerous spatial reference frameworks associated with real and conceptual objects occurring in the solar system. Special emphasis is given to near Earth applications associated with various Earth reference models and geodesy. A principal media for the educational process has been in the form of tutorials given at annual SEDRIS Technology Conferences [SEDR]. A popular part of the SRM tutorial has been the development of guidelines for the design of efficient algorithms for spatial operations. As part of an SRI sponsored Internal

Research and Development program, the guidelines developed for the tutorial have been revised and enhanced with time. A version of the guidelines was incorporated into ISO/IEC 18026 FDIS as a major part of a non-normative annex (Annex B. Implementation Notes).

This report provides an enhanced and up to-date-version of the guidelines. Use of these guidelines is not restricted, but it should be understood that the utility of these guidelines is application dependent. Selection of the appropriate design approach is the developer's responsibility. The authors welcome comments and suggestions for improvement that can be used for future enhancements of this report.

## 1.2 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. Using single precision arithmetic for spatial operation computations associated with the Earth may lead to a loss of precision on the order of half a meter, even when the application is for the near Earth region. To mitigate loss of precision it is advisable to employ double-precision arithmetic for floating-point operations [IEEE-754]. Through the use of double precision arithmetic it is assumed that the digitization error is so small that it can be ignored in the following discussions.

## 1.3 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 many 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 computational complexity for the type of spatial operations addressed in this report. 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 addressed in this report.

As noted, a spatial operation computation is a coordinate transformation, a coordinate conversion, an azimuth determination, a distance calculation or other computations associated with elliptical trigonometry and map projections. Distance may be Euclidean or on the Earth reference surface (geodesics). Over the years, a large number of computational algorithms have

been developed for a wide range of applications. Many of these are not appropriate for efficient processing using current computer system environments circa the year 2000 and beyond.

If a target application does not require efficient processing, any accurate algorithm for computing spatial operations may be employed. The purpose of this report is to develop a set of guidelines or advisories that will support a developer in the design of efficient algorithms that maintain a specified accuracy when overall throughput is important. While the target environment is generally a computer system with a superscalar architecture, many of these advisories are applicable to legacy computer systems and specialized systems used for embedded processing.

## 1.4 Error

Historically, measurement errors for locations on the Earth have been so large that sometimes little attention has been given to approximation errors made in procedures for computing spatial operations. These errors are present because the mathematical formulation involves a series or an iterative process that has been terminated at some finite number of computations. Computational error also includes the deliberate replacement of some functions by approximations that are employed for the purpose of increasing efficiency. Computational error, as used here, does not include measurement or implementation errors. With the increasingly accurate measurement systems now available it has become important to develop procedures with very small computational errors. At the same time it is important for many applications that such procedures be efficient.

# 2.0 GUIDELINES

## 2.1 The Computational Environment:

It is important to consider the properties of the current computational environment. In the last two decades an astonishing 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 machine environments include:

1. A surfeit of low cost Dynamic Random Access Memory (DRAM),
2. the development of very high-speed cache memory that permits the dynamic allocation of blocks of critical data to the cache memory. Several levels of cache memory, often denoted L1, L2, ... exist and afford differing access speed.
3. superscalar architectures that permit pipelined (parallel) processing,
4. integrated chips that permit very high speed processing for some critical mathematical functions (e. g. some transcendental functions and square root),
5. the development of compilers that exploit the advantages of super scalar architectures,
6. the development of optimizing operating systems that re-order computations and memory accesses in real time to maximize throughput.

Generally such machines are designed to nominally operate in IEEE double precision in which the mantissa is 52 bits. This is equivalent to 15 plus decimal digits of accuracy. This level of precision is generally adequate for Earth-specific applications.

An example of the impact of the new computing environment is the computation of trigonometric functions. In the legacy 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 in 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 new environment, the trigonometric functions are computed in special processor units using high-speed memory and parallel processing. As a result, these functions produce double precision results that are very fast relative to the basic operating speed of the computer. Circa 2006, a rule of thumb is that a double precision multiplication typically takes 1 processor cycle, a double precision divide takes about 32 cycles, a double precision square root about 40 cycles and a double precision sine 100 -200 processor cycles depending on the argument. Development of a faster approximation seems reasonable until one realizes it can take between 2 and 200 processor cycles to retrieve each coefficient from memory. One popular software mathematics library initializes over 20 variables when a sine call is made and can perform over 50 double precision multiplication operations per call. On computers with effective hardware implementation of transcendentals, implementing a general in-line sine sub-routine (by table look up) can be slower than calling the system sine function. This can happen because the access time required to fetch the appropriate fitting coefficients from DRAM may take longer than the entire system routine computation.

The previous paragraph advises against attempting to develop in-line code for general-purpose calculation of standard mathematical routines. However, this advisory does not apply to compound functions or mathematical functions that are not implemented in special processor units. For example, it may be more efficient to evaluate $\sin\left(f\left(x\right)\right)$ in-line using an approximation 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 complexity of $f(x)$ and tends to favor the in-line approximation approach when more than two transcendental functions are composed.

## 2.2 Domain Of Applicability

The domain of applicability should be defined *before* developing or selecting a procedure. Many authors waste development time and processing time by forcing their methods to work in non-practical regions such as near the center of the Earth reference model or at ellipsoidal heights of 100 million kilometers from the surface. The number of applications for such regions is so small that any interest in these regions is primarily academic

For Earth-referenced applications there are several regions of practical interest. For aircraft, an appropriate region in terms of ellipsoidal height is –200 meters to 35,000 meters for all latitudes and longitudes. This covers the region where air-breathing assets can operate. For sub-surface operations an appropriate region is –12,000 meters to +200 meters 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 and to call this composite region the near-Earth region. Of course, space operations may require a region extending above 35 kilometers 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 is $[-\pi/2, \pi/2]$ and the longitude domain is $(-\pi, \pi]$. Often a particular application may be restricted to a smaller latitude/longitude region to simplify formulations and in the case of map projections, to reduce distortions.

## 2.3 Defining a Meaningful Measure of Computational Error

In many situations involving spatial operation computations, the resulting variables are not exact, due to approximations made in their computational formulations. An error measure is needed to determine the approximation error. If the target variables are in terms of distance in some Euclidean coordinate system then a Euclidean norm can be used to measure the approximation error. Such an error measure is called position error. Some authors use the maximum error in the absolute value of the difference between the true values and the approximate values of each component (just another mathematical norm).

Some authors have also used the average of the absolute value of the differences of each component of position error 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.

## 2.4 Avoiding Excessive Accuracy

The literature is replete with algorithms that are excessively accurate. One journal paper on geocentric to geodetic coordinate conversion develops a procedure whose approximation error is $10^{-20}$ meters, surely 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. A method, A, with maximum error $10^{-8}$ meters is some times declared superior to a method, B, which has a maximal error of $10^{-6}$ meters. If method A takes more processing time than B it is not superior. In fact it is quite likely that both methods are too accurate. For example, suppose there is a method C with maximum error less than $10^{-4}$ meters (tenth of a mm) but takes less computer time than A or B. Then method C would likely be preferable for most (if not all) applications.

## 2.5 Determining the Acceptable Error Before Starting

The maximum allowable position error should be determined before starting an algorithm development. The SEDRIS program has a goal of keeping the computational error (position error) to less than 1 mm for all procedures involving spatial operations [SEDR]. Note that if the position error is less than 0.9mm then 0.9mm = .0009m = .02286in <<0.03125in=1/32in which, from a practical point of view, is very small. When position error is small its component errors also very small. When a position error criterion is used, the nominal position error is usually

much smaller than the maximum error, i. e. much less than 1 mm. 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.

## 2.6 Mathematical Approaches

Mathematical formulations for solving spatial operation computational problems can be relatively complex. The complexity is driven by the fact that Earth reference models are usually oblate ellipsoids. This results in formulations in which the principal equations involved are non-linear and often not solvable in closed form. For most 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 non-linear equations in more than one dimension where no closed solution is available.

Traditionally, either power series or iterative solutions have been used for solving inverse spatial operation problems. The choice of which approach to use depends on the problem, the skill of the algorithm designer and the skill of the software developer. Note that both of these methods have an interesting property that is often not recognized. Almost all publications and subsequent implementations of finite 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 nil. 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 use in an iterative approach. Almost all publications on iterative methods suggest iterating until the difference between successive iterates is sufficiently small. This is not a good policy when efficiency is important. Legacy software implementations always seem to use the maximum number of terms or iterations no matter what. This is generally a huge waste in computation time.

It is advisable, if not mandatory, to perform an error analysis to determine the maximum error in an algorithm over an applicable domain. This permits the *a priori* determination of how many terms in a series are needed to achieve a specified accuracy and the corresponding region(s) surrounding the expansion point where this happens. This will save calculation of terms in regions where the number of terms is so small as to not affect the results.

Similarly, the maximum number of iterations required for an iterative scheme can be determined along with the region(s) where further iterations do not affect the results. This will save at least one iteration (often more than one) and eliminates the need for in line comparison testing.

Another approach for solving multivariable non-linear equations exists that is much more appropriate to the new computational environment. This is the use of curve fitting or approximation of a function or the inverse of a function. In one dimension this amounts to piecewise approximation or “table look up”. The penalty, associated with this approach, is that it takes additional 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 DRAM. The trend in computational mathematics is to use low order local approximations for efficiency.

The choice of a method to employ is dependent on the application and in particular on the computer system being used.

## 2.6 Good Programming and Formulation Practices

Good programmers usually employ good programming practices. They move the computation of global constants out of 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 colleges and universities now teach that these practices are not important because modern computers are so fast that they are not needed. Or it is taught that optimizing compilers will invoke such good practices automatically. In complex codes it may not be easy or possible for a compiler to clean up poor practices. It is advisable to always use good programming practices in situations where throughput is important as many popular compilers with excellent integer optimization have virtually no capacity for floating point optimization. In some rudimentary systems, there may be neither a compiler nor a higher order language.

In many cases the formulation coded 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. Often the efficiency of the formulation can be greatly improved by eliminating redundant transcendental functions. For example, trigonometric functions can sometimes be eliminated or simplified through the use of identities or often just simple observations.

One simple example is a test like $\sqrt{x} \le c$ used as a branch point test. If *c* is a constant this can be re-written in the equivalent form, $x \le c^2$ which does not require a square root. This observation generalizes to branch tests like $f(x) \le c$ which become $x \le f^{-1}(c)$ in cases where this makes mathematical sense. If *c* is a constant, the right hand side is now a constant and can be computed at start up. Sometimes *c* is a variable but $f^{-1}$ is easier to compute than *f*.

More complex examples of use of the publication form of a formulation have led to publications whose conclusions about convergence rate and processing time requirements are wrong. A classic example is in the conversion of geocentric coordinates to geodetic coordinates using Bowring's method. 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 [FUKA], [TOMS]. Fukashima has coined an appropriate term for the implementation of the published form of Bowring's method. He calls this a *naïve implementation*. This appellation can be applied to many spatial operation formulations where the direct implementation of the published form of the formulation is often naïve.

## 2.7 Design For Context

Spatial operation computations are usually not performed in isolation, but are often used as part of a sequence of operation computations. For example, a point in geocentric coordinates is converted to geodetic coordinates and this is immediately followed by a conversion to a map projection. Such a sequence is sometimes referred to as a chain of operations. By anticipating a chain, some of the early calculations may be saved for use later in the chain.

For example, when the projection in the above chain is Transverse Mercator, the curvature in the prime vertical, the sine of latitude and the cosine of latitude will be needed to support the conversion. These values are computed as part of the first conversion from geocentric to geodetic

coordinates, and should be saved for use in the ensuing conversion to Transverse Mercator. Similar chains occur in simulations and embedded systems. When simulating dynamics models, the values of latitude and longitude are often not needed at all (only the trigonometric functions of these are used).

Efficiency gains from treating a spatial operation computation as a chain may depend on the individual procedures used. For example, some procedures for converting geocentric coordinates to geodetic coordinates do not compute the trigonometric values and only supply angular values for latitude and longitude. If the trigonometric functions of these are later 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. For example, in the non-naïve implementation of Bowring's method, the sine and cosine of latitude can be computed from basic principles as part of the process [FUKA], [TOMS]. No trigonometric functions are needed. As a consequence, the sine and cosine of latitude should be saved for later use.

## 2.8 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 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 coding error.

As a result, verification testing should be aided by computational error testing whereby the results of the developed software are tested against properly vetted results from the literature. Computational error testing is generally resource intensive because it involves archiving and utilizing a wide range of test data sets with their pedigrees while each case and result are tracked individually. Such testing is hampered by the fact that there are frequent errors in published results and caution must be used when selecting test data. It is wise to assume at the beginning of an algorithm development project that any line of code that was not executed during computational error testing may contain one or more software defects.

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 codes developed for spatial operation testing. Unfortunately, such authoritative data may be sparse or may not cover the full range of the area of interest. Fortunately, many spatial operation formulations have closed form solutions in at least in one direction. Even though closed form solutions may not be very efficient and approximation algorithms may be needed to provide high-speed solutions, 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 absolute computation error. In some cases, mathematical analysis can provide analytic bounds to the error [KENI]. 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 must be accomplished with respect to the procedures as implemented in software.

It is important 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 points

themselves should be automatically generated over a region or sub-region of interest. A convenient way to do this is to set up a lattice of reference points. Such a lattice can be in one or more dimensions and is often referred to as a *gridded data set*. A procedure is developed to test an implemented algorithm at each grid point. This procedure records the absolute 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 error should converge to a fixed value. This fixed value will closely approximate the maximum error over the region of interest.

As an example, consider the testing of the conversion of geocentric coordinates to geodetic coordinates for a region with ellipsoidal height bounds defined by

$0 \le \lambda \le \frac{\pi}{2}$, $0 \le \phi \le \frac{\pi}{2}$ and $\mathrm{h}_{\min} \le h \le h_{\max}$, where longitude is $\lambda$, latitude is $\phi$, $h$ is

ellipsoidal height, and $h_{\min}$ and $h_{\max}$ are respectively the lower and upper bounds on the

ellipsoidal heights of interest. A three dimensional lattice or grid can be formed on $\left[0, \frac{\pi}{2}\right] \times \left[0, \frac{\pi}{2}\right] \times \left[h_{\min}, h_{\max}\right]$ with equal spacing along each grid axis. The coordinate $(\lambda, \phi, h)$ of each lattice point is exact by construction[1]. Each coordinate $(\lambda, \phi, h)$ in geodetic coordinates can be exactly converted to the corresponding point $(x, y, z)$ in geocentric coordinates using well-known closed form formulas. Of course, "exact" here means to implement this process in double precision, that is, to more than 15 decimal digits. This process yields two test data sets that are exact by construction. One is a set of points in geodetic coordinates $(\lambda, \phi, h)$ and the other is a corresponding set of points in geocentric coordinates $(x, y, z)$. If an approximate coordinate conversion method is used in either direction on these exact sets it is pro forma to determine the error at each test point. The maximum error is then easy to compute on the test set. When the lattice spacing is made smaller on each axis the maximum error in the process and its location(s) can be estimated. If the error is excessively large at some test point, either the algorithm is defective or there is a coding error.

Certain points may represent points of singularity in a formulation. Values at the singularity are usually determined analytically. In computing values close to a singularity numerical sensitivities can occur. In the neighborhood of such a point dense testing should be performed to insure that proper results are obtained throughout the neighborhood. This is the dense testing on a sub-region referred to in the previous paragraph.

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. This is referred to as round-trip testing and is recommended in some publications [HOOI]. Other publications warn of the dangers in round trip testing (see page 76 of [HDPT]). In general, round trip testing cannot yield precise error values. Both round-off error and approximation error are generally present in each direction. These normally not symmetric and do not cancel each other. At best, round trip testing

[1] Points in geodetic coordinates have their components written in the order longitude, latitude and ellipsoidal height, $(\lambda, \phi, h)$, in order to comply with ISO/IEC 18026 (FDIS). This ordering is a consequence of the use of right-handed coordinate systems [18026].

is useful for rough order of magnitude testing and may expose some gross coding errors. Round trip testing cannot be counted upon to show that a routine has no software defects.

Example

This example is intended to illustrate one of the inadequacies of round trip testing. The SEDRIS implementation of the transverse Mercator (TM) map projection uses finite Taylor series representations for computing both the forward and inverse conversions. This formulation is used in this example.

The formula for the Easting value, u, contains a multiplicative factor $k_o$ called the scale factor. Suppose that the designer or implementer forgets to put this factor in the formulation and in the inverse formulation. Effectively, this means that the scale factor is always 1 in both the forward and inverse conversions. Suppose that a round trip test is used to try to verify the accuracy of the formulation. The developer, designer and tester are all unaware of error.

The test point is on the equator at 3 degrees longitude. In SRM notation, in radians, this is the input point (0.0523598775598299 , 0.0). The forward conversion yields the approximate point $u_a$=834112.201701349 and $v_a$=0.0 m, This point is used to provide input to the inverse procedure to complete the round trip. The resulting approximate point in radians is (0.0523598775451542, 0.0) and in degrees (2.99999999915914, 0.0). The degrees representation is encouragingly close to 3 degrees, but is it good enough? To assess this question, the error is converted to an equivalent distance error.

The angular longitude error is converted to an approximate position error by differential error analysis which uses the semi-major axis of the ellipsoid in the formula, error = $a$(true longitude-approximate longitude).= $a$(0.0523598775598299-0.0523598775451542)= 0.936037019286309D-04 ≤ .00094 mm. As a result of this round trip test one might be tempted to conclude that the result is very accurate and meets a 1mm maximum position error criterion. This conclusion might be reached even though the conversion formulas have an unknown error in them.

The exact value of u is known, that is u=833978.556918999995105. This value was supplied by Craig Rollins of the U.S. National GeoSpatial Intelligence Agency [ROLL]. After the forward approximation, the approximate Easting value $u_s$ can be directly compared with the true value to determine the error in the forward formulation. That is, the error=u- $u_a$ =(833978.556918999995105-834112.201701349) m = -133.644782348536 m. This error is way outside the acceptable limit of 1mm. This means that using the exact results for the error analysis shows unequivocally that there is an error, either in the input or the formulation or the code. Using exact values for comparison may be the only way to be sure that something is wrong.

Continuing the round trip process a large number of times for a fixed test point will generally lead to divergence of the results. This is to be expected due to the lack of symmetry of the errors in each direction. Repeated round trip tests like this are meaningless for validating an algorithm for a spatial operation.

The lack of symmetry of the error between the computation of a spatial operation and its inverse can also cause difficulties when points are on or very near a validity boundary. For example, suppose that a geodetic point lies on a zone boundary for a UTM projection. When the point in UTM is computed a truncated power series is used. Due to the combination of truncation and round-off errors the resulting point may be in an adjacent zone. Applying the inverse spatial operation may not yield a point in the adjacent zone and not in the zone that it started in. This is due to the lack of symmetry of the forward and inverse errors. Application developers need to be aware of this type of problem.

## 2.9 Performance Testing

As the computational environment has evolved 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 computing 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 operation or 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 common approach to performance testing 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 timing routine. 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 must be taken with this technique. Sometimes the algorithm is very simple and if a compiler is used and has its optimizing level set to a high level, erroneous results can result. For example, if the procedure is $y = \sin(x)$, the compiler may recognize that the result is the same for every pass through the loop and move the computation outside the loop. This will of course lead to erroneous results.

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 superscalar architecture. An algorithm's performance may be quite different when it is embedded in a large application program. One of the authors (Kellough) has coined some nomenclature to distinguish between such cases. 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 the early stages of 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), followed 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.

## 2.10 Spherical reference models

All of the mathematical formulations for spatial operations with respect to spatial reference frames based on an oblate ellipsoid reference model 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. For efficiency purposes it may be important to approximate the exact formulations for the spherical cases.

## 2.11 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.

## 2.12 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 as appropriate. Another frequently encountered problem has to do with the computation of longitude for inverse cases. The result $\lambda$ (in radians) may lie outside the interval $-\pi < \lambda \leq \pi$. Some publications are not very specific about this but what should be done is to add or subtract $2\pi$ as appropriate. In Clause 5, Table 5.6 of ISO/IEC 18026 a longitude centering function is introduced for this purpose of the form

$$\Lambda_C(\lambda, \lambda_C) = \begin{cases} \lambda - \lambda_C & \text{if } -\pi < \lambda - \lambda_C \leq \pi \\ \lambda - \lambda_C - 2\pi & \text{if } \pi < \lambda - \lambda_C \\ \lambda - \lambda_C + 2\pi & \text{if } \lambda - \lambda_C \leq -\pi \end{cases},$$

where $\lambda_c$ is a specified origin or reference value [18026].

## 3.0 REFERENCES

[SEDR] SEDRIS Program, http/www.sedris.org

[18026] ISO/IEC 18026 Spatial Reference Model (FDIS), January 2006.

[IEEE-754] Standard for Binary Floating Point Arithmetic, 1985

[TOMS ] Toms, R. M., "Efficient Procedures for Geodetic Coordinate Transformations,"*Military Application Society of INFORMS, Proceedings of the First National Meeting,* University of Alabama at Huntsville, Huntsville, AL, 19-21 May 1998. See also <http://www.sedris.org/download/documentation/CompositeSR53.pdf.

[FUKA] Fukashima, T. *Fast transform from geocentric to geodetic coordinates*. Journal of Geodesy, vol. 73, no. 11, p. 603-610. Heidelberg (Germany): Springer-Verlag Heidelberg, 1999.

[KENI] Keeler, Steven P. and Nievergelt, Yves, *Computing Geodetic Coordinates,* SIAM Review, Vol. 40, No. 2, pp. 300-309, June 1998.

[HOOI] Hooijberg, Marten, Practical Geodesy Using Computers, Springer-Verlag, Berlin Heidlberg, 1997, ISBN3-540-61826-0.

[HDPT] US Department of Defense, US Army Corps of Engineers, Topographic Engineering Center (TEC). *Handbook for transformation of datums, projections, grids and common coordinate systems*. Alexandria (Virginia): TEC, 1996. TEC handbook TEC-SR-7.

[ROLL] Private Communication from Craig Rollins of the U.S. National GeoSpatial Intelligence Agency, Coordinate Systems Analysis Team, Arnold, MO.