1. Introduction

Mainstream computers base integer and floating point arithmetic on fixed word lengths. As a consequence, only values with a limited number of significant digits can be represented directly, so that the results of arithmetic operations may have to be rounded off or truncated. Such errors can be avoided or, at least, mitigated, by implementing special algorithms for the execution of arithmetic operations. A fully “exact arithmetic”, however, would have to be based on quotients of integers for representing numerical values. In any case, a final limitation is due to finite memory. The need for exact arithmetic became apparent during the development of software for generating triangular and tetrahedral nets from very large point sets. Typically, this need is not due to high accuracy requirements for results—the input data are often noisy or given up to only a few significant digits—but is rather due to the need to maintain the consistency of a combinatorial structure. Building or manipulating such geometry-based combinatorial structures requires the calculation of indicators such as determinants in order to evaluate their sign and to check for zero values: round-off may lead to a false sign or zero value. An example considered later in this paper is to decide whether four given spatial points are coplanar. The approach of using exact computations for implementing computational geometry algorithms in a robust manner has been addressed in [2], [3], [4], [5], [6], [7]. Some computational geometry implementations ([2], [3], [4], [5]) reduce computational effort by utilizing exact arithmetic selectively whenever a decision might be affected by round-off.

In this paper, we document software for exact integer arithmetic, accommodating an indeterminate number of
digits, for multiplication, addition, subtraction, but excluding division. We found that, in many computational geometry applications, decision variables such as determinants can be calculated without division. Also the sign of a decision variable stated as a quotient but not evaluated is readily derived from the signs of the numerator and denominator.

We also describe a preprocessing step, called “two-integer decomposition”, which leads from floating point input to one composed of integers only. At the root of this step lies the concept of space as an integer grid of points, all of which have integer coordinates in some shared unit. After completing the transition from floating point numbers to intermediate representations as pairs of integers—prompted by the fact that Fortran 77 does not provide a double precision integer format—a polynomial decomposition creates the number representations to be used in the exact arithmetic calculations. Software for this preprocessing step together with software for exact integer arithmetic has been successfully incorporated into several computational geometry related programs such as REGTET [1].

In what follows, a “standard computer” is a computer that uses 64 bits of storage for a double precision number and 32 bits for an integer. Given a standard computer, even though it may not store exactly an input decimal number as a double precision floating point number, it is safe to assume that the number will be represented as accurately as possible by a double precision floating point number up to its fourteenth significant digit.

2. Two-Integer Decomposition

Let \( x(i), i = 1, \ldots, n \), be a double precision array into which input numbers \( x_i, i = 1, \ldots, n \), have been read. The two-integer decomposition process is a preprocessing step that takes place before any computations based on the input data are carried out. It rounds off the numbers in the array at a prespecified location of their decimal fractions and decomposes each rounded off number into two integers that are saved in integer arrays, say \( ix(i), i = 1, \ldots, n \). The rounded off numbers are then saved in array \( x(i), i = 1, \ldots, n \).

Given integers \( k, l, 1 \leq k \leq 9, 0 \leq l \leq 9, k + l \leq 14 \), and assuming each input number \( x_i, i = 1, \ldots, n \), has no more that \( k \) digits to the left of the decimal point, each number \( x(i), i = 1, \ldots, n \), is rounded off at the \( l \)th digit of its decimal fraction and decomposed into two integers in one of two ways according to its size. If the absolute value of \( x(i) \) times \((10.0d0)^l\) is less than \(2^{30}(=1073741824)\), \( x(i) \) is multiplied by \((10.0d0)^l\) and rounded off at the decimal point. The resulting integer is then placed in \( ix(i) \) while \( ix2(i) \) is set to zero. Finally, \( x(i) \) is redefined to be the double precision value of integer \( ix(i) \) divided by \((10.0d0)^l\). On the other hand, if the absolute value of \( x(i) \) times \((10.0d0)^l\) exceeds or equals \(2^{30} \), \( x(i) \) is truncated at the decimal point. The resulting integer (absolute value less than \(2^{30} \) since \( k \leq 9 \)) is placed in \( ix(i) \). In addition, the signed decimal fraction obtained by subtracting the double precision value of this integer from the initial value of \( x(i) \) is multiplied by \((10.0d0)^l\) and rounded off at the decimal point. The resulting integer (absolute value less than \(2^{30} \) since \( l \leq 9 \)) is placed in \( ix2(i) \). Next, \( x(i) \) is redefined to be the double precision value of integer \( ix(i) \) plus the value obtained by dividing the double precision value of integer \( ix2(i) \) by \((10.0d0)^l\). Finally, if the integer \( ix2(i) \) is zero then the integer \( ix2(i) \) is set to \(2^{30} \) so that \( ix2(i) \) is zero if and only if the initial absolute value of \( x(i) \) (before the two-integer decomposition process) times \((10.0d0)^l\) is less than \(2^{30} \).

The following is Fortran code for carrying out the two-integer decomposition process. Variables are either integer or double precision following convention. It is noted that while some loss in precision may occur at the time the input numbers are read and transformed into double precision floating point numbers, some additional loss in precision may occur here as well when the decimal point in a number is shifted by dividing or multiplying it by a multiple of \(10.0d0\), when the signed decimal fraction of a number is obtained by truncating the number at its decimal point and subtracting the result from the initial value of the number, and when a number is rounded off with the two-integer decomposition process. However once the two-integer decomposition process is completed all computations that follow, exact and otherwise, are carried out in terms of the arrays \( x(i), ix(i), ix2(i), i = 1, \ldots, n \), under the assumption that for the purposes of the user for each \( i, i = 1, \ldots, n \), \( x(i) \) represents closely enough the input number \( x_i \) rounded off at the \( l \)th digit of its decimal fraction, and an integer (not necessarily stored by the computer) in terms of \( ix(i) \) and \( ix2(i) \) represents closely enough the input number \( x_i \), times \(10^l \) rounded off at the decimal point.

```fortran
mfull=1073741824
if(1.Lt.0.or. Lgt.9) stop 10
isclu = 1
dscle = 1.0d0
if(l.Leq.0) go to 200
do 100 i = 1, 1
```
isclu = 10*isclu
dscle = 10.0d0*dscle
100 continue
200 continue
dfull = dble(mfull)
dfill = dfull/dscle
do 300 i = 1, n
ix2(i) = 0
if(dabs(x(i)).lt.dfill) then
  ix(i) = idnint(dscle*x(i))
going to 300
endif
endif
dabs(x(i)).gt.dfull) stop 20
ix(i) = x(i) = dble(ix(i))/dscle
else
  x(i) = dble(ix(i) + (dble(ix2(i))/dscle)
endif
300 continue

3. Polynomial Decomposition

Given an integer \( l, 0 \leq l \leq 9 \), let \( x_i, i = 1, ..., n \), be input numbers whose double precision floating point representations have been rounded off at the \( l \)th digit of their decimal fractions through the two-integer decomposition process. Let \( x(i), ix(i), isx2(i), i = 1, ..., n \) be the arrays produced by the two-integer decomposition process that contain the rounded off numbers and the two-integer decompositions. For each \( i, i = 1, ..., n \), an integer \( J(i, l) \) is symbolically defined as follows (its actual value is not necessarily computed or stored by the computer). If \( ix2(i) \) equals zero then \( J(i, l) \) is set equal to \( ix(i) \). If \( ix2(i) \) equals \( 2^n \) then \( J(i, l) \) is set equal to \( ix(i) \cdot 10^{l'} \). Finally, if \( ix2(i) \) is neither zero nor \( 2^n \) then \( J(i, l) \) is set to \( ix(i) \cdot 10^{l'} + ix2(i) \). In all cases for each \( i, i = 1, ..., n \), \( J(i, l) \) is considered to approximate closely enough (for the purposes of the user) the input number \( x_i \) times \( 10^{l'} \) rounded off at the decimal point.

Set \( M \) to \( 2^{15} \). Given \( J(i, l), 1 \leq i \leq n \), the polynomial decomposition process is a procedure (presented below in the form of Fortran subroutine decmp2) that decomposes the integer \( J(i, l) \) into a unique collection of integers \( isga, isga \) in \{−1, 0, 1\}, \( ika, ika > 0, a_k, 0 \leq a_k < M, k = 1, ..., ika \), such that \( J(i, l) \) equals \( isga(\Sigma_{k=1}^{ika} a_k \cdot M^{k-1}) \), \( isga \) the sign of \( J(i, l) \), Integers \( a_k, k = 1, ..., ika \), are saved in an integer array, say \( ia(k), k = 1, ..., ika \), and the collection of integers \( isga, ika, ia(k), k = 1, ..., ika \), and the symbolic expression \( isga(\Sigma_{k=1}^{ika} ia(k) \cdot M^{k-1}) \) are then called, respectively, the polynomial decomposition and the symbolic polynomial representation of \( J(i, l) \), with \( isga \) as the sign of the representation. For each \( i, i = 1, ..., n \), the polynomial decomposition of the integer \( J(i, l) \) is identified each time an exact computation involving additions, subtractions, or multiplications is required that references the input number \( x_i \). During one such computation, for each \( i, 1 \leq i \leq n \), if the number \( x_i \) is referenced in the computation, once the polynomial decomposition of the corresponding integer \( J(i, l) \) is identified, each reference of \( x_i \) in the computation is replaced by the symbolic polynomial representation of \( J(i, l) \). The computation then takes effect as a sequence of additions, subtractions, or multiplications of symbolic polynomial representations with the final result being itself the symbolic polynomial representation of some integer. This final result can usually be used in only one of two ways. If it is known that for some positive integer \( m \) the integer that is equal to the final symbolic polynomial representation is approximately equal to the product of \((10^m)\) and the true value of the computation, then this integer is computed approximately as a double precision floating point number from its symbolic polynomial representation and the true value of the computation is then approximately obtained by dividing it by \( ((10.0d0)^m) \). On the other hand, if the purpose of the computation is simply that of obtaining the sign of the true result then the sign of the final symbolic polynomial representation is a satisfactory answer.

The concepts of polynomial decomposition and symbolic polynomial representation defined above for \( J(i, l), 1 \leq i \leq n \), can also be defined for any integer \( K \) (not necessarily stored by the computer) in the same manner. Accordingly, the following is a Fortran subroutine called decomp for finding the polynomial decomposition \( isga, ia(k), k = 1, 2, (ika \) is already known to equal 2) of an integer \( iwi \) (stored by the computer) with absolute value less than \( 2^{32} \). Here \( mhalf \) equals \( 2^{32} (=32768) \).

```
subroutine decomp(isa, isga, iwi, mhalf)
  integer ia(*), isa, iwi, mhalf, iwi
  if(iwi.gt.0) then
    isga = 1
    iwi = iwi
  endif
```

elseif(iwi.lt.0) then
    isga = -1
    ivi = -iwi
else
    isga = 0
    ia(1) = 0
    ia(2) = 0
    return
endif
ia(2) = ivi/mhalf
ia(1) = ivi - ia(2)*mhalf
return
end

In particular if isclu is set to 10 then isclu is less than 2^30 (since l ≤ 9) so that the polynomial decomposition isgu (equal to 1), iu(i), i = 1, 2, (iku is already known to equal 2) of isclu can be obtained by calling subroutine decomp with a Fortran instruction as follows.

call decomp(iu, isgu, isclu, mhalf)

Finally, the following is a Fortran subroutine called decmp2 for finding the polynomial decomposition isga, ika, ia(k), k = 1, ..., ika, of the integer J(i, l), 1 ≤ i ≤ n. Here iwi equals ix(i), iwi2 equals ix2(i), mhalf equals 2^15, mfull equals 2^30, and iu(k), k = 1, 2, is an array such that the polynomial decomposition of 10^l is iu(1), iu(2) (isgu and iku are already known to equal 1 and 2, respectively). In addition, it is assumed that subroutines mulmul and mutdif (presented below) exist for multiplying and subtracting, respectively, two symbolic polynomial representations.

subroutine decmp2(ia, isga, ika, iwi, iwi2, mhalf, mfull, iu)
integer nkmax
parameter (nkmax=5)
integer ia(*), iwi(*), mfull, iu(*)
intrinsic abs
integer ie(nkmax), io(nkmax), isge, isgo, ike, iko, isgu, iku
call decomp(ia, isga, iwi, mhalf)
* iko = ika + iku
if(iwi2.ne.0) then
    isgu = 1
    iku = 2
    call mulmul(ia, iu, isga, iku, isge, isgo, ike, iko, iku
    * iko1 = 2
    if(iwi2.eq.mfull) iwi2 = 0
    call decmp(io, isgo, iwi2, mhalf)
else
    isgu = -isgo
    iko = 2
    call mutdiff(ie, iu, isga, iku, ike, iko, iku
    * iku1 = 2
endif
return
end

4. Multiplying Symbolic Polynomial Representations

Given the polynomial decompositions isga, ika, ia(k), k = 1, ..., ika, and iu(k), k = 1, ..., iku, of two integers K_1 and K_2, respectively, the following is a Fortran subroutine called mulmul that produces the polynomial decomposition isgo, iko, iu(k), k = 1, ..., iko, of the integer K_1 · K_2 by multiplying the symbolic polynomial representation of K_1 by that of K_2 (as polynomials) to produce a symbolic polynomial representation of K_1 · K_2 from which the polynomial decomposition of K_1 · K_2 can be obtained. Here nkmax is the dimension of the arrays ia, ib, io in the calling routine and mhalf equals 2^15. It is noted that the value of mhalf is of importance here since given integers i, j, 1 ≤ i ≤ ika, 1 ≤ j ≤ iku, then 0 ≤ ia(i) < 2^15, 0 ≤ ib(j) < 2^15, so that the product ia(i) · ib(j) is less than 2^30 and therefore can be stored in a 32 bit integer word.

subroutine mulmul(ia, ib, io, isga, isgb, isgo, ika, ikb, iko, * nkmax,mhalf)
integer ia(*), ib(*), io(*)
integer isga, isgb, isgo, ika, ikb, iko, nkmax, mhalf
integer i, ipt, ipr, iko1, k, j
if(isga.eq.0.or.isgb.eq.0)then
    isgo=0
    iko = 2
    io(1) = 0
    io(2) = 0
    return
else
    if(isga.gt.0)then
        if(isgb.gt.0)then
            isgo = 1
            else
                isgo = -1
            endif
        else
            if(isgb.gt.0)then
                isgo = 1
            else
                isgo = -1
            endif
        endif
    endif
endif
return
end
5. Subtracting Symbolic Polynomial Representations

Given the polynomial decompositions \( isga, ika, io(k), k = 1, \ldots, ika \), and \( isgb, ikb, ib(k), k = 1, \ldots, ikb \), of two integers \( K_1 \) and \( K_2 \), respectively, the following is a Fortran subroutine called muldif that produces the polynomial decomposition \( isgo, iko, io(k), k = 1, \ldots, iko \), of the integer \( K_1 - K_2 \) by subtracting the symbolic polynomial representation of \( K_2 \) from that of \( K_1 \) (as polynomials) to produce a symbolic polynomial representation of \( K_1 - K_2 \) from which the polynomial decomposition of \( K_1 - K_2 \) can be obtained. Here \( nkmax \) is the dimension of the arrays \( ia, ib, io \) in the calling routine and \( mhalf \) equals 2\(^{15}\). It is noted that by setting \( isgb \) equal to \(-isgb\) the polynomial decomposition of \( K_1 + K_2 \) can also be obtained with this subroutine.

```fortran
subroutine muldif(ia, ib, io, isga, isgb, isgo, ika, ikb, iko, * nkmax,mhalf)
    integer ia(*), ib(*), io(*)
    integer isga, isgb, isgo, ika, ikb, iko, nkmax, mhalf
    integer i, iko1, irel
    if(isgb.eq.0)
        then
            isgo=0
            iko = 2
            io(1) = 0
            io(2) = 0
            return
        endif
        isgo = isga
        iko = ika
        do 100 i=1,iko
            io(i) = ia(i)
        100 continue
        io(iko) = ipr
        if(ipr.ge.mhalf) stop 120
        irel = 0
        do 500 i = iko, 1, -1
            if(ia(i).gt.ib(i)) then
                irel = 1
                go to 600
            elseif(ia(i).lt.ib(i)) then
                irel = -1
                go to 600
            endif
        500 continue
        600 continue
        if(irel.eq.0)
            then
                isgo = 0
                do 700 i=1,iko
                    io(i) = 0
                700 continue
            else
                isgo=isga*irel
                io(1) = (ia(1)-ib(1))*irel
                do 800 i=2,iko
                    io(i) = (ia(i)-ib(i))*irel
                800 continue
            endif
        return
    endif
end
```
Given an integer \( n \geq 4 \), let \( S \) be a set of \( n \) points in 3-dimensional space. Given an integer \( l \), \( 0 \leq l \leq 9 \), let \( x_l, y_l, z_l, i = 1, ..., n \), be the input decimal coordinates of the points in \( S \), and assume that their double precision floating point representations have been rounded off at the \( l \)th digit of their decimal fractions through applications, one per coordinate, of the two-integer decomposition process. Accordingly, let \( x(i), ix(i), ix2(i), y(i), iy(i), iy2(i), z(i), iz(i), iz2(i), i = 1, ..., n \), be the arrays produced by the three applications of the two-integer decomposition process that contain the rounded off \( x \), \( y \), \( z \)-coordinates and their two-integer decompositions.

Given points \( p_i, p_2, p_3 \) in \( S \) that are vertices of a non-degenerate triangle, a fundamental problem in computational geometry is that of finding the location of a point \( p_4 \) in \( S \) relative to the plane \( H \) that contains the triangle. Let \( H' \) be the open half-space defined by \( H \) for which \( p_1, p_2, p_3 \) appear in a counterclockwise direction around the boundary of the triangle when looking at the triangle from \( H' \). Let \( H' \) be the other half-space defined by \( H \). Determining in which of \( H, H', H' \), the point \( p_4 \) is located may not on occasion be satisfactorily done using floating point arithmetic. Accordingly, the following is a Fortran subroutine called ccrsinn for doing this using polynomial decompositions. On output the sign \( isgo \) (\(-1, 0, 1\)) of some polynomial decomposition determines the location of \( p_4 \) in \( (H, H, H') \).

This routine actually does more. It produces polynomial decompositions \( isgox, ikox, iox(k), k = 1, ..., ikox, isgoy, ikoy, ioy(k), k = 1, ..., ikoy, isgoz, ikoz, izo(k), k = 1, ..., ikoz, \) of integers that are the coordinates of a vector \( \nu \) pointing into \( H' \) and perpendicular to \( H \). It also produces the polynomial decomposition \( isgo, iko, io(k) \), \( k = 1, ..., iko, \) of an integer whose sign \( isgo \) determines the location of \( p_4 \) and whose value when divided by both \( 10 \) and the length of \( \nu \) is the perpendicular distance from \( p_4 \) to \( H \). Here \( mhalf \) equals \( 2^{15} \), \( mfull \) equals \( 2^{30} \), and \( ifir, isec, ithi, ifou \) are locations in the arrays \( ix, iy, iz \), etc. corresponding to the points \( p_1, p_2, p_3, p_4 \), respectively. In addition, \( isclp(k), k = 1, 2, \) is an array such that the polynomial decomposition of \( 10^k \) is \( isclp(1), isclp(2) \) (the sign of \( 10^k \) and the dimension of array \( isclp \) are already known to be \( 1 \) and \( 2 \), respectively).

```
subroutine ccrsinn(ix, iy, iz, ix2, iy2, iz2, ifir, isec, ithi,
    *       ifou, mhalf, mfull, isclp, io, isgo, iko, iox,
    *       isgox, ikox, iox(k), isgoy, ikoy, ioy(k),
    *       isgoz, ikoz, izo(k), isclp(*), io(*),
    *       isgo, iko, io(k), mhalf, mfull, nkmax
integer nkmax = 30
integer ix(nkmax), iy(nkmax), iz(nkmax)
integer ix3(nkmax), iy3(nkmax), iz3(nkmax)
integer ix4(nkmax), iy4(nkmax), iz4(nkmax)
integer ixf(nkmax), iyf(nkmax), izf(nkmax)
integer ifir, isec, ithi, ifou
integer isclp(*), mhalf, mfull, nkmax
```

6. Application: Locating a Point Relative to a Plane

Given an integer \( n \), \( n \geq 4 \), let \( S \) be a set of \( n \) points in 3-dimensional space. Given an integer \( l \), \( 0 \leq l \leq 9 \), let \( x_l, y_l, z_l, i = 1, ..., n \), be the input decimal coordinates of the points in \( S \), and assume that their double precision floating point representations have been rounded off at the \( l \)th digit of their decimal fractions through applications, one per coordinate, of the two-integer decomposition process. Accordingly, let \( x(i), ix(i), ix2(i), y(i), iy(i), iy2(i), z(i), iz(i), iz2(i), i = 1, ..., n \), be the arrays...
Sometimes besides knowing the location of the point \( p_4 \) relative to the plane \( H \) it may be desirable to know the perpendicular distance from \( p_4 \) to \( H \). The following is Fortran code for this purpose. It uses the polynomial decompositions that are part of the output of subroutine \( 	ext{crsinn} \). Variables here are either integer or double precision following convention. Here \( r_{215} \) equals \((2.0 \times 10^{15})\) and \( \text{dist} \) is the resulting signed perpendicular distance. In addition, it is assumed that subroutine \( \text{doubnm} \) (presented below)
exists for transforming the polynomial decomposition of an integer into the double precision floating point value of the integer.

call crsinn(ix, iy, iz, ix2, iy2, iz2, ifir, isec, ithi, ifou,
  * mhalf, mfull, isclp, io, isgo, iko, iox, 
  * isgox, ikox, ioy, isgoy, ikoy, ioz, 
  * isgoz, ikoz)
call doubnm(io, isgo, iko, r215, dnum)
call doubnm(iox, isgox, ikox, r215, xnum)
call doubnm(ioy, isgoy, ikoy, r215, ynum)
call doubnm(ioz, isgoz, ikoz, r215, znum)

\[ dnux = \text{dmax1}(\text{dabs}(xnum), \text{dabs}(ynum), \text{dabs}(znum)) \]
\[ xnum = \frac{xnum}{dnux} \]
\[ ynum = \frac{ynum}{dnux} \]
\[ znum = \frac{znum}{dnux} \]

\[ \text{dnom} = \text{dsqrt}(xnum^2 + ynum^2 + znum^2) \]
\[ \text{dist} = \left( \frac{dnum}{dnux} / \text{dnom} \right) / \text{dscle} \]

The following is subroutine doubnm that was called above.

subroutine doubnm(io, isgo, iko, r215, dnum)
integer io(*)
double precision dnum, r215, rpwr
integer isgo, iko, i

if(isgo.eq.0) then
  dnum = 0.0d0
  go to 900
else
  if(iko .lt. 2) stop 310
  if(iko .gt. 68) stop 320
  rpwr = 1.0d0
  dnum = dble(io(1))
  do 100 i = 2, iko
    rpwr = rpwr*r215
    dnum = dnum + dble(io(i))*rpwr
  100 continue
  if(isgo.lt.0) dnum = -dnum
  go to 900
endif
900 continue
return
end

7. Numerical Examples

Twelve lines follow, each line containing three numbers. Each line corresponds to a point in 3-dimensional space, and the three numbers in the line correspond to the \( x \)-, \( y \)-, \( z \)-coordinates of the point, in that order. Given \( i \), \( 1 \leq i \leq 12 \), point \( i \) is the point corresponding to the \( i \)th line. It is assumed that the coordinates of the

called above.

\[ -13.729277089 \ 14.530621914 \ 97.981467003 \]
\[ 38.000000000 \ 7.049967880 \ -92.123710427 \]
\[ 41.736468803 \ 68.831641719 \ -59.331882431 \]
\[ 85.557213025 \ -53.594476834 \ -13.994897166 \]
\[ 33.675274550 \ -77.937397763 \ 52.741164465 \]
\[ -2.277195916 \ -53.594476834 \ -84.424190762 \]
\[ -33.285508962 \ -14.545102894 \ 93.175307798 \]
\[ 15.161728368 \ 3.186043237 \ 98.792566086 \]

Given \( l \) equal to 8, through the two-integer decomposition process, the numbers above are rounded off at the \( \ell \)th = 8th digit of their decimal fractions and saved in \( x(i) \), \( y(i) \), \( z(i) \), \( i = 1, \ldots, 12 \), so that then they appear as follows.

\[ -13.729277090 \ 14.530621910 \ 97.981467000 \]
\[ 38.000000000 \ 7.049967880 \ -92.123710430 \]
\[ 41.736468800 \ 68.831641720 \ -59.331882430 \]
\[ 85.557213020 \ -53.594476830 \ -13.994897170 \]
\[ 33.675274550 \ -77.937397760 \ 52.741164460 \]
\[ 1.724283840 \ -53.594476830 \ -84.424190760 \]
\[ -2.277195920 \ -58.886394970 \ 80.791131020 \]
\[ 70.061142980 \ 9.068097310 \ -70.800333280 \]
\[ -33.285508960 \ -14.545102890 \ 93.175307800 \]
\[ -2.277195920 \ -58.886394970 \ 80.791131020 \]
\[ 70.061142980 \ 9.068097310 \ -70.800333280 \]

Each rounded off coordinate is also decomposed into two integers. Twelve lines follow, each line containing six integers. For each \( i \), \( 1 \leq i \leq 12 \), the first two integers in the \( i \)th line are the two integers into which the \( x \)-coordinate of point \( i \) is decomposed. Similarly the next two integers correspond to the \( y \)-coordinate, and the last two to the \( z \)-coordinate. The two-integer decompositions of the twelve points are then saved into \( ix(i), iy(i), iz(i), i = 1, \ldots, 12 \), in the obvious manner. It is noted that when \( mfull = 2^{30} = 1073741824 \) appears as the second integer corresponding to a coordinate it is to be interpreted as a zero.

\[ -13 \ -72927709 \ 14 \ 53062191 \ 97 \ 98146700 \]
\[ 38 \ 1073741824 \ 704996788 \ 0 \ -92 -12371043 \]
\[ 41 \ 73646880 \ 68 \ 83164172 \ -59 -33188243 \]
the polynomial decompositions associated with the 

\[ 24832 + 3051 \cdot 2^{15} \].

As obtained above. The resulting polynomial decomposition of the coordinate, i.e. the integers 85 and 70, as given above, the computation that is the product of the integers may be too big to be saved into a 32 bit integer word is obtained instead. Thus, the polynomial decomposition of the integer can be approximated by a double precision number which when divided by \( 10^8 \) rounded off at the decimal point equals \( 24832 + 3051 \cdot 2^{15} \). By calling subroutine doubnm the integer can be approximated by

\[
\text{(38.000000000)} - \text{(–49.840807040)} - \text{(7.049976880)} - \text{(85.57213020)}. \]

On output the integer is approximated by a double precision number which when divided by \( 10^8 \) is approximately \( -2997.1262712133 \), an approximation of the desired result.

Finally, an example is given for the purpose of describing the process of locating a point relative to a plane that contains three other points. Here the point whose location is desired is point 12 as given above, and the three other points are point 1, point 2, point 8, and point 118 as given above. With \( r \) as the triangle with vertices point 1, point 2, point 8, and \( H \) as the plane that contains \( t \), \( H' \) is taken to be the open half-space defined by \( H \) for which point 1, point 2, point 8 appear in a counterclockwise direction around the boundary of \( t \) when looking at \( t \) from \( H' \). \( H' \) is taken to be the other half-space defined by \( H \) with \( t \) set to 2, \( t \) set to 8, ifou set to 12, which of \( H, H' \) contains point 12 can be determined by calling subroutine csrnn as follows.

\[
\text{call csrnn(x, iy, iz, ic2, iy2, iz2, ifir, isec, ith, ifou,}
\]

\[
* \text{mhalf, mfull, isclp, io, isgo, iko, iox, isgoy,}
\]

\[
* \text{iko, isgoy, ikoy, ioz, isgo, ikox)} \]

On output isgo equals –1 so that point 12 must be in \( H \).

In addition iko equals 7, isgo = 1, ..., 7, equals 213, 287, 380, 5372, 13630, 9887, 213, isgoy equals –1, ikox equals 5, isgo = 1, ..., 5, equals 11577, 8756, 364, 27887, 63, isgoy equals 1, ikoy equals 5, ioz(k) = 1, ..., 5, equals 11577, 8756, 364, 27887, 63, isgoy equals –1, ikor equals 5, ioz(k) = 1, ...

Given \( l \) as above equal to 8 and setting isclu to 10 \( l \), by calling subroutine decmp the polynomial decomposition of isclu is found to be isgcl, isclpl, isclp(1), isclpl(2), with isgcl equal to 1, isclpl(1) equal to 24832 and isclpl(2) equal to 3051 (10 \( l \) = 24832 + 3051 \cdot mhalf = 24832 + 3051 \cdot 2^{15}).

As an example of how exact computations are carried out that reference coordinates of the input points given above, the computation that is the product of the \( x \)-coordinate of point 2 and the \( y \)-coordinate of point 4 minus the product of the \( y \)-coordinate of point 2 and the \( x \)-coordinate of point 4 is described. Using rounded-off numbers the result of the computation should equal

\[
(38.000000000) - (–49.840807040) - (7.049976880) - (85.57213020). \]

On the other hand, if exact computations are required then each of the four numbers involved must first be converted into an integer which is the number times \( 10^8 \) rounded off at the decimal point. Since the resulting integer may be too big to be saved into a 32 bit integer word, its polynomial decomposition in the form of two or more 32 bit integer words is obtained instead. Thus, for example, the polynomial decomposition of the integer which is the \( x \)-coordinate of point 4 times \( 10^8 \) rounded off at the decimal point can be obtained by calling subroutine decmp2 using the two-integer rounded off at the decimal point can be obtained by calling subroutine decmp the polynomial decomposition of

\[
(38.000000000) - (–49.840807040) - (7.049976880) - (85.57213020). \]
The vector \((ix, iy, iz)\), which is perpendicular to the plane \(H\) and points into \(H'\), can then be approximated by the vector \((xnum, ynum, znum)\). Finally, by dividing \(dnum\) by both the length of the vector \((xnum, ynum, znum)\) and \(10^8\) the number \(-25.047402554921\) is approximately obtained, an approximation of the signed perpendicular distance from point 12 to plane \(H\), the negative sign indicating that point 12 is in \(H'\).

8. Summary

A scheme has been presented and software has been documented for transforming into a series of integers input decimal numbers that have been read into a computer as double precision floating point numbers and for carrying out multiplication, addition and subtraction operations based on these numbers using their integer representations. The total number of significant digits of each input number must not be greater than 14, and the number of digits to the left of the decimal point must not exceed 9. Through a preprocessing step the double precision floating point representation of each input decimal number is rounded off at a prespecified location of its decimal fraction, a location no more than 9 digits to the right of the decimal point, and the rounded off number is decomposed into two integers. All operations that follow involving the input number are then carried out in terms of the rounded off double precision floating point number and when this is not satisfactory in terms of the two integers. This scheme has been successfully incorporated into several computational geometry related programs such as REGTET [1]. Other programs that incorporate this scheme can be found at http://math.nist.gov/JBernal/JBernal_Sft.html. Besides being used for locating a point relative to a plane, in these programs the scheme has also been used for locating a point relative to a sphere, for computing the intersection of a line and a plane, for computing the center of a sphere, etc.

9. References

[1] J. Bernal. REGTET: A Program for Computing Regular Tetrahedralizations. NISTIR 6786 (2001).
[2] S. J. Fortune and C. J. van Wyk. Efficient exact arithmetic for computational geometry. Proc. of the 9th Symp. on Computational Geometry (1993) pp. 163-172.
[3] S. J. Fortune and C. J. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry, ACM Transactions on Graphics 15 (3), 223-248 (1996).
[4] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation, Proc. of the 15th Symp. on Computational Geometry pages (1999) pp. 351-359.
[5] K. Mehlhorn and S. Näher. LEDA: A platform for combinatorial and geometric computing. Cambridge University Press, Cambridge, UK (1999).
[6] C. K. Yap. Towards exact geometric computation, Computational Geometry: Theory and Applications 7, 3-23 (1997).
[7] C. K. Yap and T. Dubé. The exact computation paradigm, in Computing in Euclidean Geometry, World Scientific Press, Singapore (1995) pp. 452-492.

About the authors: Javier Bernal is a mathematician in the Mathematical and Computational Sciences Division of the NIST Information Technology Laboratory in Gaithersburg, MD. He received his Ph.D. in Mathematics in 1980 from Catholic University in Washington, DC in the same year he joined NIST. His research interests include the development, analysis and implementation of Computational Geometry algorithms. Christoph Witzgall holds the designation of Scientist Emeritus from the NIST Information Technology Laboratory (ITL). Although he retired from government service in 2003, he continues to serve as a guest researcher in the ITL Mathematical and Computational Sciences Division. Christoph has been associated with NIST since 1962, serving as Acting Chief of its Operations Research Division from 1979 to 1982. He received the Department of Commerce Silver Medal for meritorious Federal service. The National Institute of Standards and Technology is an agency of the Technology Administration, U.S. Department of Commerce.