

## Chapter 12

# Use of Finite Arithmetic in 3D Spatial Databases

Rodney, James and Thompson

**Abstract.** Most of the spatial theory that underpins geographic information repositories is based on the assumption of Euclidean space, which requires the support of the real number system. The actual implementation of this theory on computer hardware then requires the use of finite precision arithmetic, meaning that some approximations must be made during the processing of certain calculations. This paper considers these approximations, and the effect they have when made at different critical points in the processes.

**Keywords.** 3D DBMS; imprecision; spatial logic; topology; computational arithmetic.

### 12.1 Introduction

Most of the spatial theory that underpins geographic information repositories is based on the assumption of Euclidean space, which requires the support of the real number system. The actual implementation of this theory on computer hardware then requires the use of finite precision arithmetic, meaning that some approximations must be made during the processing of certain calculations.

This paper considers these approximations, and the effect they have when made at different critical points in the processes.

### 12.2 Approaches Taken

There is an inherent and unavoidable loss of accuracy when any measurements of real-world phenomena are taken and stored. This obvious fact is common to all approaches discussed in this paper. The issue being discussed here is the small inaccuracies that accumulate in the database representation of spatial features as those representations are processed and manipulated during the life of the data.

In order to compare the different approaches, a comparison will be made using as examples:

---

Department of Natural Resources and Water, Queensland, Australia  
Rod.Thompson@nrw.qld.gov.au  
Delft University of Technology, OTB, GIS Technology, The Netherlands.  
rthompson@tudelft.nl

- The validation of a polygon in 3D as a planar object.
- The validation of polyhedra.
- The operation of forming the union of polyhedra.
- A test for equality of volumetric features in space.

In addition, the important topological operations: union, intersection and complement [4], the predicates intersects and overlaps, and the Region Connection Calculus predicates: connected, disconnected, tangential proper part etc. [12] are considered.

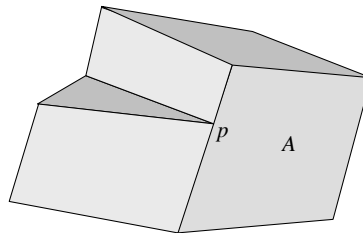
### 12.2.1 Validation of a Polygon as a Planar Object

Any three points may have a plane that passes exactly through them, and if the points are not collinear or coincident, this plane will be uniquely defined. Where four or more points are involved, it may not be possible to find a plane that passes through them. Many 3D objects are represented as having planar faces, where these faces have more than three vertices. It is important in these representations that the faces are truly planar.

This issue can be sidestepped by using a triangulation of the surfaces, where each surface is broken into triangular facets. This does, however lose the definition of the original surfaces, and does not document the intention that they be planar.

### 12.2.2 Validation of Polyhedra.

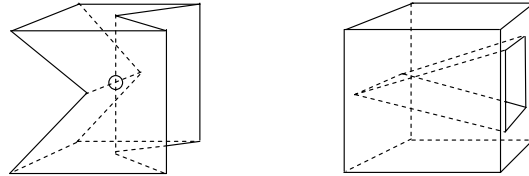
In addition to the requirement for planar surfaces, there are other validation requirements being suggested for 3D objects. One of the more important is that the volume be “watertight”. That is to say, if the volume is defined in terms of the bounding faces, then the faces must meet exactly along all edges. In practice, given the finite precision of point representations, this means that additional vertices must be inserted in some faces. For example in Fig. 12.1, a point at  $p$  has to be inserted into face  $A$ , to prevent mismatch of the approximated position with the edge.



**Fig. 12.1.** A redundant point in face  $A$  of polyhedron

Another requirement is that the outer surface should be simple, while the internal volume should be connected. [9] give a number of specific cases of polyhedra which

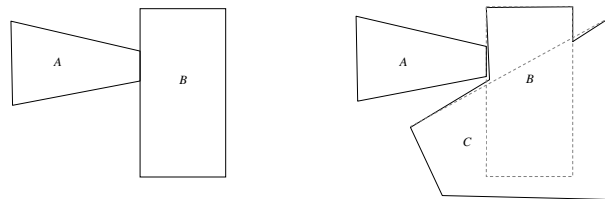
fail this requirement. In the cases depicted in Fig. 12.2, the finite precision calculations may result in their being detected as valid or invalid by different software.



**Fig. 12.2.** Polyhedra with non-simple boundary surfaces

### 12.2.3 Union of Polyhedra

The operation of forming the union of two polyhedra can be problematic, when calculated using finite precision arithmetic. For example, if a region is connected to another region, it should be connected to the union of that region with any other. For example, in Fig. 12.3, since  $A$  is connected to  $B$ , it should be connected to  $B \cup C$ , however, small inaccuracies in the calculation of the points of intersection of the boundaries can cause this to break down. For simplicity, this issue is discussed using 2D cases, but applies in any dimensionality.



**Fig. 12.3.** Connectivity is not necessarily conserved in calculations. Here  $A$  is strongly connected to  $B$ , but due to limited accuracy of calculations (shown exaggerated) is not connected to  $B \cup C$ .

Further, the calculation of the union or intersection of regions can lead to non-associativity of operations. In the same way that floating point arithmetic is not necessarily associative, it is possible that the result of calculation of the union or intersection of spatial regions may differ depending on the order that calculation is carried out. For example in Fig. 12.3, since  $A$  is connected to  $B$ , the union  $A \cup B$  is a simple, connected polygon. Forming  $(A \cup B) \cup C$ , the result would be a simple connected polygon. If, however,  $B \cup C$  is calculated, small inaccuracies can result in  $A$  not being connected to  $B \cup C$ , so that  $A \cup (B \cup C)$  is a multi-polygon.

### 12.2.4 Equality of Volumetric Objects in Space

The simple test of determining if A is equal to B should be the most basic of operations in any system. It is therefore surprising that it is rarely implemented correctly if at all, and that the ISO 19107 definition of equality of spatial objects would not satisfy the requirements of the Object class in Java.

The ISO 19107 definition of equals() (ISO-TC211 2001) uses the phrase "shall return true if this GM\_Object is equal to another GM\_Object", but qualifies this definition with: "Since an infinite set of direct positions cannot be tested, the internal implementation of equal must test for equivalence between two, possibly quite different, representations. This test may be limited to the resolution of the coordinate system or the accuracy of the data. Application schemas may define a tolerance that returns true if the two GM\_Objects have the same dimension and each direct position in this GM\_Object is within a tolerance distance of a direct position in the passed GM\_Object and vice versa" (ISO 19107 Section 12.6.2.2.18.3).

There is also a need for a stronger form of equality. As an example, consider the implementation of geometric objects as subclasses of the Object class of Java. This requires the implementation of "equals" and "hashCode" methods [13]. A hash-keyed structure is used where large numbers of objects are to be stored, with the expectation that the hashCode method generates a key value that can be used to provide fast access to any object in the collection. It is acceptable (and unavoidable) for different objects to generate the same key value ("collisions"), with the "equals" method being used to distinguish between them. The requirements of the hashCode method are that for two objects *a* and *b*:

```
a.equals(b) ⇒ a.hashCode() = b.hashCode()
a.hashCode() = b.hashCode() "nearly always implies"
a.equals(b)
```

The "nearly always implies" determines how useful the algorithm is. If many unequal objects generate the same hash code, the algorithm is inefficient, leading to multiple collisions.

It is difficult to imagine any hashCode routine that is useable in conjunction with an "equals" test which allows a tolerance.

## 12.3 Approximate the Results of Every Step

This is the most common approach taken currently. The problem is that at any stage of an operation it is possible for the logic to break down, because the rounding implied in any stage can invalidate the results of earlier stages.

### 12.3.1 Floating Point Arithmetic

At each step in an operation the arithmetical unit of the computer performs a rounding. This may be an explicit rounding to a selected accuracy, or an implied rounding

that is generated by the floating point arithmetic unit. This is the approach taken by many of the systems that operate in floating point.

The IEEE standard for floating point operations IEEE 754 [5, 6] is a partially successful attempt to ensure that a calculation will give consistent results in differing computational contexts. The remaining inconsistencies seem to be in the handling of overflow, underflow and “divide by zero” [8].

On the other hand, the results of certain arithmetic calculations are exactly specified – down to the mandatory rounding to be used, for any computer hardware that advertises IEEE compliance. These include addition, subtraction, division, multiplication, square root and binary to/from decimal conversions. Significantly, they do not include the trigonometric functions (sin, cos etc).

It might be thought that it would be possible to take a similar approach to the calculation of geometric results, but this is out of scope of this paper. The difficulties of this approach are legion, but include the same types of issues that prevent the exact standardisation of trigonometric functions.

One non-technical reason for the reluctance of standardisation bodies to take such an approach is that the details of actual calculations may be commercial in confidence, and particularly well designed algorithms can be valuable distinguishing features of software, giving a competitive commercial advantage.

In summary, it may be assumed that the calculation of a result, such as the point of intersection of three planes, will result in a good approximation to the true point regardless of the platform used, but that the same result cannot be expected on different platforms.

It should be expected, however, that the re-calculation of a function from identical inputs should have exactly the same result, but even this can be jeopardised by “over-enthusiastic” optimisation in some programming environments [8].

Arithmetic operations between floating point numbers are not necessarily associative. That is to say  $(a+b)+c$  may not give the same answer as  $a+(b+c)$ .

For example,  $10^{10} + (-10^{10} + 10^{-10})$  gives zero, while  $(10^{10} - 10^{10}) + 10^{-10}$  gives  $10^{-10}$ . This can be significant in many calculations, and must be considered where repeatability of results is desired.

### 12.3.2 Validation of a Polygon as a Planar Object

In general, any polygon in 3D with more than 3 vertices cannot be represented using floating point coordinates such that the vertices are exactly coplanar. If the calculation of the vertex coordinates were carried out using infinite precision arithmetic, and the results rounded according to the IEEE 754 rules, the amount of deviation from planarity could be kept to a predictable (and very small) distance.

If the vertices are the result of calculations which are carried out using floating point arithmetic, the deviation can be more extreme. For example, if a vertex is calculated as the point of intersection of three planes, defined by parametric equations:

$$\begin{aligned} a_1x + b_1y + c_1z + d_1 &= 0 \\ a_2x + b_2y + c_2z + d_2 &= 0 \end{aligned} \tag{1}$$

$$a_3x + b_3y + c_3z + d_3 = 0$$

then the point of intersection  $p = (x, y, z)$  can be calculated as  $x = p_x/q, y = p_y/q, z = p_z/q$ , where:

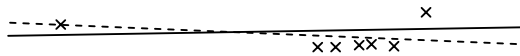
$$q = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} \tag{2}$$

$$p_x = \begin{vmatrix} -d_1 - d_2 - d_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}, p_y = \begin{vmatrix} a_1 & a_2 & a_3 \\ -d_1 - d_2 - d_3 \\ c_1 & c_2 & c_3 \end{vmatrix}, p_z = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ -d_1 - d_2 - d_3 \end{vmatrix} \tag{3}$$

The calculations of these intermediate results require the addition of six terms, each of which are the product of three variables. Thus, the final accuracy of  $x, y,$  and  $z$  will be dependent on the initial values. This can be reduced by normalising the parameters in various ways (e.g. by replacing  $d_1$  by  $d_1/\sqrt{a_1^2 + b_1^2 + c_1^2}$ ,  $a_1$  by  $a_1/\sqrt{a_1^2 + b_1^2 + c_1^2}$  etc.. thus converting to Hessian normal form) [16], but there will still be a deviation of the vertices from the strictly correct position. The non-associativity of floating point arithmetic means that the results may not be repeatable on different platforms.

Typically, the approach taken to the issue of planarity of a polygon is to state that a polygon is considered planar if all points are within some tolerance of the plane.

This is not, however a useful form of statement for standardisation. It is necessary, in deciding that all points are sufficiently close to a plane to determine the plane first. It is quite possible for such a plane to exist and be known to the sender, but if the parameters of the plane are not passed to the receiving system, there is no guarantee that it can be found. (The sender may have used a more sophisticated form of interpolation than was available or known to the recipient).



**Fig. 12.4.** Attempting to find a plane close to vertices

For example, in Fig. 12.4, the vertices marked as x's are all within a reasonable tolerance distance of the plane (drawn from the side as a solid line). However, the plane shown as a dashed line, the result of a least squares fitting does not pass suffi-

ciently close to all points. The plane generated, even by the same algorithm, cannot be guaranteed to be the same on all platforms. It is not common practice for the parametric forms of planar surfaces to be included in interchange specifications.

### 12.3.3 Validation of Polyhedra

In order to ensure that this validation is computable, it is common practice when making the initial data load approximations, to apply a normalisation<sup>1</sup> in the form suggested by [11]. This specifies a tolerance  $\epsilon$ , such that the distance between a point and a line can be calculated with accuracy  $\frac{\epsilon}{10}$ . Note that the polyhedra in Fig. 12.2 highlight the need in 3D for additional rules to those of Milenkovic, that no two lines may be closer than  $\epsilon$  except at their endpoints, no line may be nearer than  $\epsilon$  to a plane, etc.

Given the complexity of the calculation of the minimum distance between two lines, and the non-associativity of floating point arithmetic, even the determination of Milenkovic normalisation cannot be used as a validity criterion, unless we are willing to specify the exact calculation to be used, and the order of individual operations within that calculation. It is not even clear whether the order of the two lines, or the direction of the two lines affect the calculation (e.g. for points  $a,b,c,d$  does the distance calculated between line  $ab$  and line  $cd$  equal the distance calculated between  $cd$  and  $ab$ , and between  $ab$  and  $dc$  etc).

That is to say that even if an object is valid, and Milenkovic normal<sup>2</sup> to tolerance  $\epsilon$  (including the additional criterion), there is no guarantee that it is Milenkovic normal as determined by other software.

### 12.3.4 Union of Polyhedra

Since the calculation of the intersections of lines and surfaces is of limited precision, rounding effects will apply as described in section 12.2.3. Thus it cannot be assumed that many obvious results of spatial theory can apply. For example, using the terminology  $C(A, B)$  to mean  $A$  is connected to  $B$ :

$C(A, B)$  does not imply  $C(A, B \cup C)$ .

$C(A, B \cap C)$  does not imply  $C(A, B)$ .

$A \cup (B \cup C)$  is not necessarily equal to  $(A \cup B) \cup C$ .

$A \cap (B \cap C)$  is not necessarily equal to  $(A \cap B) \cap C$ .

And so on. In such a system, it is difficult to create a toolkit of operations and predicates that can be relied on in all cases.

The situation in topologically-encoded data stores is different. Once a complete polygon coverage of space has been built, the topological and connectivity operations between polygons are clearly rigorous and repeatable. On the other hand, the later ad-

<sup>1</sup> This is usual when building a topologically-encoded data structure.

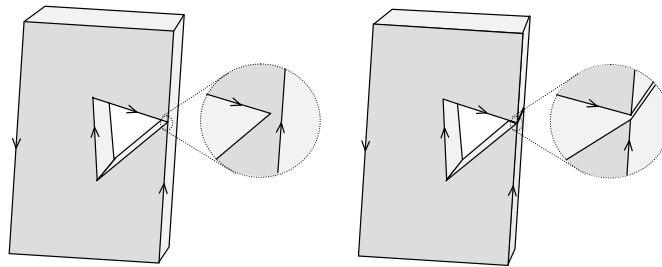
<sup>2</sup> Note also that the “winding number” defined by Milenkovic needs to be extended to 3D in the obvious way.

dition of polygons to an existing coverage is not so clear, and it is possible that the order of adding these polygons may be significant.

### 12.3.5 Equality of Volumetric Objects in Space

The implementation of the ISO 19107 definition of equality in a repeatable and standard form is highly problematic. Note that the standard does not specify the tolerance, but further, it does not specify how the tolerance is to be applied.

Determination of an algorithm is also problematic, since the representations of two objects may, indeed be different, while they are equal by this definition. For example, in Fig. 12.5, two polyhedra are depicted. Due to small differences in positioning, the polyhedron on the left is topologically homeomorphic to a toroid, while the polyhedron on the right is simply connected. Under the ISO 19107 definition, if the gaps are sufficiently small, the two polyhedra can be equal.



**Fig. 12.5.** Equal polyhedra of differing forms. The object on the left has a hole right through it. The one on the right has a simple interior.

In a similar way, a single polyhedron can be equal to a disconnected set of polyhedra. Thus, in a system that does not permit multiple geometries, a feature may be valid, but equal to a feature which is invalid.

## 12.4 Delay Rounding Until End of Operation

Again, when data is loaded, there is an implied or explicit rounding/truncation.

In this approach, the rounding at every step in the process is delayed until the whole operation is completed. For example, in the case depicted in Fig. 12.3, only when the full calculation of  $(A \cup B) \cup C$  or  $A \cup (B \cup C)$  is completed is any rounding permitted.

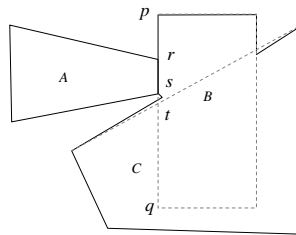
This can be achieved in one of two ways:



- Using rational number arithmetic (unlimited precision) during the calculation, and then rounding back to integer or floating point coordinates at the completion of the calculation, or
- Using floating point or integer arithmetic, with “lazy evaluation” of point positions.

The use of infinite precision rational arithmetic will be discussed in Section 12.5, while the lazy evaluation approach will be discussed here. The essence of this approach is that any calculation which involves rounding is delayed as long as possible. Alternatively, a result is calculated in rounded form, but the original data are retained so that subsequent calculations can be carried out with the unrounded values.

Returning to the example of Fig. 12.3 (see also Fig. 12.6), in calculating  $B \cup C$ , the point  $t$  may be calculated in approximate form, but the original line  $pq$  is retained, so that when the union of  $A$  with  $(B \cup C)$  is calculated it can be used to calculate points  $r$  and  $s$  (in approximate form). Using this approach, it makes no difference to the final result whether the union is formed as  $A \cup (B \cup C)$  or  $(A \cup B) \cup C$ .



**Fig. 12.6.** Forming the union of regions using lazy evaluation of point positions

This is only true for as long as the original points are retained, and if the operation is carried out as two steps, the result will vary – e.g. if a new region is calculated as  $D = (B \cup C)$ , then  $A \cup D$  may not equal  $(A \cup B) \cup C$ .

The essence of this approach is that the imprecision can be seen as applying only at the end of each operation, while the operations themselves are rigorous. Thus it can be modelled as if the output of the calculation is then applied to an imprecision function. Thus in the example above it is clear why  $\text{APPROX}(\text{APPROX}(A \cup B) \cup C)$  is not necessarily equal to  $\text{APPROX}(A \cup \text{APPROX}(B \cup C))$ .

## 12.5 Round on Data Load Only

In this approach, rounding is avoided as far as possible, apart from the rounding implied at the time of data load. Typically, a rational number representation is used, or the points are represented by homogenous coordinates. In this discussion, homogenous integer coordinates are used. Rational coordinates have a similar behaviour.

Some rounding may be necessary, for example in datum shifts or point positional adjustments, and the geometry validity may be compromised in this situation.

### 12.5.1 Homogeneous Coordinates

In section 12.3.1., the formula for the point of intersection of three planes is given in terms of  $p_x, p_y, p_z$  and  $q$ , with the point being represented as  $p = (x, y, z)$ , where  $x = p_x/q, y = p_y/q, z = p_z/q$ . An alternate storage for rational points is the homogeneous coordinate form, as defined in the discipline of Projective Geometry [3]. Here the  $q$  is retained, and the point (in 3D) is represented as a quadruple of numbers  $p = (p_x, p_y, p_z, q)$ . The advantage is that integers can be used throughout, since in calculations such as determining the plane through three points, or the intersection of three planes, no division need be done. (In the discussion below, capital letters will be used for integers, lowercase letters for rational, floating point or real numbers).

The formula for the plane through three points  $(X_1, Y_1, Z_1, Q_1), (X_2, Y_2, Z_2, Q_2), (X_3, Y_3, Z_3, Q_3)$  becomes:

$$\begin{vmatrix} X & Y & Z & Q \\ X_1 & Y_1 & Z_1 & Q_1 \\ X_2 & Y_2 & Z_2 & Q_2 \\ X_3 & Y_3 & Z_3 & Q_3 \end{vmatrix} = 0 \quad (4)$$

This is used, for example in the LEDA library of geometric tools [10]. In this approach, the rational points are stored as a tuple of integers  $(X, Y, Z, Q)$  with  $Q > 0$ . The point is interpreted as having rational coordinates  $(x, y, z)$  where  $x = X/Q, y = Y/Q, z = Z/Q$ .

The point of intersection of three planes defined as

$$\begin{aligned} A_1X + B_1Y + C_1Z + D_1Q &= 0 \\ A_2X + B_2Y + C_2Z + D_2Q &= 0 \\ A_3X + B_3Y + C_3Z + D_3Q &= 0 \end{aligned} \quad (5)$$

can be calculated as  $p = (X, Y, Z, Q)$  where:

$$Q = \begin{vmatrix} A_1 & A_2 & A_3 \\ B_1 & B_2 & B_3 \\ C_1 & C_2 & C_3 \end{vmatrix}, \quad (6)$$

$$X = \begin{vmatrix} -D_1 - D_2 - D_3 \\ B_1 & B_2 & B_3 \\ C_1 & C_2 & C_3 \end{vmatrix}, Y = \begin{vmatrix} A_1 & A_2 & A_3 \\ -D_1 - D_2 - D_3 \\ C_1 & C_2 & C_3 \end{vmatrix}, Z = \begin{vmatrix} A_1 & A_2 & A_3 \\ B_1 & B_2 & B_3 \\ -D_1 - D_2 - D_3 \end{vmatrix} \quad (7)$$

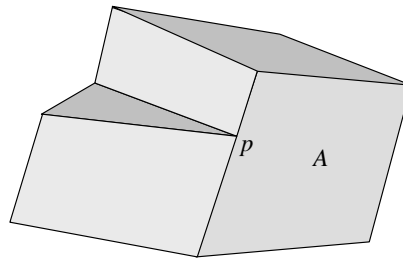
There is a pleasant symmetry to this approach, where a point is stored as a tuple of four integers,  $p = (X, Y, Z, Q)$  and a plane is also stored as four integers  $(A, B, C, D)$ . This parallels the fact that three points define a plane, while three planes define a point.

### 12.5.2 Validation of a Polygon as a Planar Object

Using homogeneous coordinates, the vertices that are calculated as the points of intersection of planes can be exact, so that there is no need to allow a tolerance in the planarity of surfaces. (except at initial data load).

### 12.5.3 Validation of polyhedra.

Returning to the example in section 12.2.2, Fig. 12.1, Fig. 12.7, the point at  $p$  does not need to be inserted into face  $A$ , since it can be calculated to fall exactly on the boundary of  $A$ . On the other hand, if it is made a vertex of  $A$ , it can be guaranteed to fall exactly on the plane of  $A$ .



**Fig. 12.7.** A redundant point in face  $A$  of polyhedron (reprint of Fig. 12.1)

Returning to the second example in Section 12.2.2, Fig. 12.2, the exact calculations of intersections of planes allow repeatable determination of the validity of complicated surfaces.

### 12.5.4 Union of Polyhedra

The difficulties in forming the union of two polyhedra are also solved by using homogeneous coordinates, since all calculations are exact.

### 12.5.5 Equality of Volumetric Objects in Space

The equality of polyhedra can be determined exactly using homogeneous coordinates, but if a tolerance is to be allowed, the complexity of the issues as described for floating point representations still apply. It would be expected that in this type of approach, exact equality would be mandated, or that a clear distinction would be made between equality, and an “approximately equal” condition based on the ISO 19107 definition.

### 12.5.6 Operations Requiring Imprecise Calculations

Although this approach covers many of the operations that make up the toolkit of geographic information systems, there are some that it does not. Any operation that involves rotation, projection or adjustment to a different datum can best be handled by approximated transformations. This is not necessarily a problem, provided it is realised that these do not necessarily preserve the full integrity of the database. Ideally, following such an operation, the validity of all features should be re-checked. For example, it is possible that some contiguous geometric objects may become disconnected.

Frequently operations of this type are not carried out in a rigorous form in any case – for example in 2D it is common for a datum correction to be applied only to the end-points of a line segment, with the assumption that the line remains straight.

Conversely, the application of an approximation sweep through the database can be an advantage, in reducing the space requirements for the storage of points.

### 12.5.7 Storage Requirements

This approach requires arithmetic operations to be available in the computational language that are effectively unlimited in size. That is to say, for integers  $A$  and  $B$ , it must always be possible to calculate  $A+B$  and  $A \times B$  exactly, and with no possibility of overflow. This is possible in several languages, including Java, but with the loss of some programming convenience. (In an language such as Java which does not permitting overloading of operators, the coding  $A = B * C + D * E$ ; has to be replaced by  $A = B.multiply(C).plus(D.multiply(E))$ ).

The disadvantage of this approach is that every operation is likely to increase the size requirements of the result. Thus after a long series of operations, the storage requirements of a representation increase (without bound), and processing time requirements for further operations increase (also without bound). Note that with numbers of the size found in spatial data – where 9 digits resolution is commonplace, the numerators and denominators can become very large indeed.

As a rough calculation, if the coordinates of points in 3D are stored as 32 bit integers, the formula for a plane which passes through them in the form  $Ax + By + Cz + D = 0$ , with  $A$ ,  $B$ ,  $C$  and  $D$  integers will in general require  $A$ ,  $B$ ,  $C$  to be at least 64 bit integers, and  $D$  to be 96 bits. Further estimates show that if three planes are intersected to define a point, and three such points used to define a plane, the storage require-

ments for this plane are ten times those of the original ones. (For example, if the original planes require 64 bits, the next generation of planes require 640 bits per parameter). Thus every operation on a three dimensional object defined by unrestricted rational number points potentially creates a very large (and multiplicative) rise in the precision requirements. Note, however, that this effect is not as extreme in the case of 2D spatial objects, where the multiplier effect is smaller.

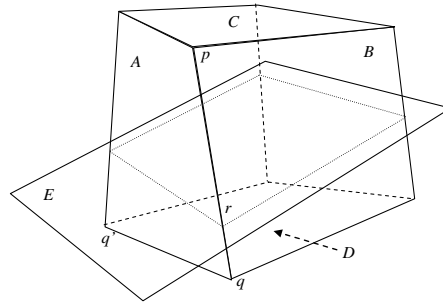
It is clear that if a common factor can be found for the integers that define a point or a plane, the storage requirements can be reduced. Clearly:

Point  $(RX, RY, RZ, RW) = (X, Y, Z, W)$  and

Plane  $(TA, TB, TC, TD) = (A, B, C, D)$ .

If it exists, a common factor can be found using a fairly obvious modification of the Euclidean Algorithm [2], allowing simplification in some cases, but whether this is productive is not necessarily clear.

Given two random integers, the probability that they are relatively prime is given as  $6/\pi^2$  [1]. This means that the probability that a common factor can be found is about 0.4. For four integers at random, the probability of any common factor drops significantly, but this does not really answer the question. There are many cases where the integers are not randomly distributed, and common factors can be found.



**Fig. 12.8.** Point of intersection of a line with a plane

For example, in Fig. 12.8, if  $p$  and  $q$  have each been defined as the intersection of three planes, ( $ABC$  and  $ABD$  respectively). If the planes require 64 and 96 bit integers for their parameters, then the points  $p$ , and  $q$  will require of the order of 224 bits for the  $XYZ$ , and 192 bits for the  $Q$  ordinates.

In general, the intersection of a plane using 64/96 bits (such as  $E$ ) with the line joining two points ( $p$  and  $q$ ) each requiring 224/192 bits would be expected to take a very large number of bits to represent the resultant point  $r$ . In fact,  $r$  is at the point of intersection of three planes, therefore it requires only 224/192 bits – the same as  $p$  and  $q$ . Thus the result of the calculation of  $r$  from  $p$  and  $q$  must be a 4-tuple of integers which have a common factor. If any calculation is made of a point or plane, it may be wise to test for common factors, even though this calculation is expensive.

Despite this case, and “accidental” cases of common factors, typically the precision requirements grow linearly (in terms of number of bits required in the result) with the number of operations executed.

As alluded to above, this type of database may benefit from an occasional sweep through all of the geometry (even if not needed to apply a datum change or other such action), approximating it back to reasonable sized coordinate values. It must be remembered that this action can introduce validation errors, and so must be accompanied by a validation sweep. If such an approximation operation is carried out at the completion of each operation, a result equivalent to the lazy evaluation approach of section 12.4 is obtained.

## 12.6 Round Only on Specific Operations

The regular polytope, based on domain-restricted rational (dr-rational) numbers [15] rounds at data load time, but also when some (rare) operations are carried out. It does, however provide rigorous evaluation of most of the important operations and predicates, including all those of topological and RCC theory.

Operations which round include:

- Datum shift or any point positional adjustment.
- Calculation of derived geometries such as convex hull.

Operations which are rigorous include:

- All RCC predicates
- All basic topological operations (union, intersection, inverse).

A regular polytope representation of spatial objects is defined as the union of a finite set of (possibly overlapping) "convex polytopes", which are in turn defined as the intersection of a finite set of half spaces (in 3D, half planes in 2D). These half spaces (planes) are defined by finite precision integer parameters (3 values in 2D, 4 in 3D etc). Although the definitions of the half spaces use integral coordinates, the points within them (and therefore the point sets defined by regular polytopes) are interpreted as domain-restricted rational points, and expressed in homogeneous coordinates.

### 12.6.1 Half Space Definition

In 3D a half space  $H(A,B,C,D)$  is defined as the set of all points (in homogeneous coordinates)  $p(X,Y,Z,Q)$ :  $Q > 0$ ,  $-MQ \leq X,Y,Z < MQ$  for which computational evaluation of the following inequalities yields these results:

$$\begin{aligned} &(AX + BY + CZ + DQ) > 0 \text{ or} \\ &[(AX + BY + CZ + DQ) = 0 \text{ and } A > 0] \text{ or} \\ &[(BY + CZ + DQ) = 0 \text{ and } A=0 \text{ and } B>0] \text{ or} \\ &[(CZ + DQ) = 0 \text{ and } A=0, B=0 \text{ and } C>0], \end{aligned}$$

where  $M$  is the limit of values allowed for point representations, and  $A, B, C$  and  $D$  are integers. We place the restrictions that:

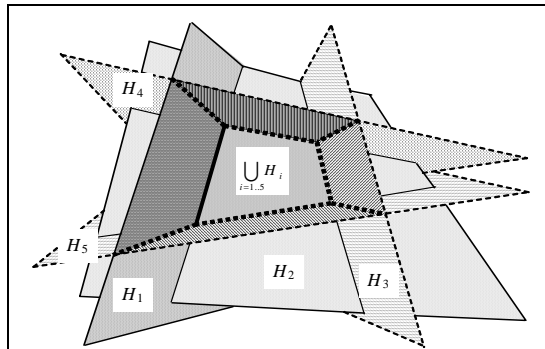
$$\begin{aligned}
 & -M < A, B, C < M, \quad -3M^2 < D < 3M^2 \text{ in 3D applications,} \\
 & -M < A, B < M, \quad -2M^2 < D < 2M^2 \text{ in 2D (C is not required in 2D).}
 \end{aligned}$$

Clearly each half space defines a plane in the same way as described in section 12.5.1 unless  $A=B=C=0$ , with the half space being the set of points to one side of that plane.  $H(0,0,0,0)$  is not a permitted half space. Two special half spaces are defined:

$$\begin{aligned}
 H_\phi &= H(0,0,0,-1) \text{ ('empty' i.e. points for which } -1 > 0\text{).} \\
 H_\infty &= H(0,0,0,1) \text{ ('everything' i.e. points for which } 1 > 0\text{).}
 \end{aligned}$$

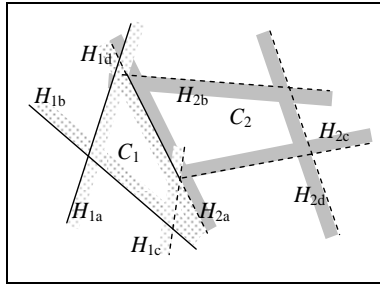
The complement of a half space is defined as:

$$\bar{H} = (-A, -B, -C, -D), \text{ where } H = (A, B, C, D).$$



**Fig. 12.9.** A convex region defined by a set of half spaces (in 3D)

### 12.6.2 Regular Polytope Definition



**Fig. 12.10.** Regular polytope  $O$  defined as the union of two convex polytopes  $C_1$  and  $C_2$ <sup>3</sup>.

A regular polytope  $O$  is defined as the union of a finite set of (possibly overlapping) non empty "convex polytopes" ( $C_1$  and  $C_2$  in Fig. 12.10), which are in turn defined as the intersection of a finite set of half spaces ( $H_{1a}$  to  $H_{1d}$  and  $H_{2a}$  to  $H_{2d}$ ) [14]. This may

be represented as  $O = \bigcup_{i=1..m} C_i$ , where  $C_i = \bigcap_{j=1..n} H_{ij}$ . Note – a regular polytope may consist of disconnected parts, and parts may overlap.

Points coincident with the boundaries, shown as dashed in Fig. 12.9 and Fig. 12.10 do not belong to the regions they define. In Fig. 12.10  $H_{2a} = \overline{H_{1d}}$ , and points that lie along the common boundary are within  $C_2$  but not  $C_1$  (and therefore are within the regular polytope  $O$ ).

The natural definitions of the union, intersection, and complement of regular polytopes is used, so that the meanings are exactly equivalent to the point set interpretations. E.g.:

$$\forall p: p \in O_1 \cup O_2 \Leftrightarrow p \in O_1 \vee p \in O_2$$

It is possible to define two forms of connectivity for the regular polytope, known as  $C_a$  (weak) and  $C_b$  (strong), and to show that for these definitions, the usual functions for a Region Connection Calculus apply. Thus it is guaranteed that all the functions and predicates based on union, intersection, overlap, connectivity, and complement will be rigorously calculated [15].

<sup>3</sup> Note – Fig. 12.2 and many others are drawn as 2D cases for the ease of presentation where there is an obvious extension to 3D. By contrast, the text and mathematics are expressed in 3D.



### 12.6.3 Validation of a Face as a Planar Object

This follows immediately from the fact that all objects are defined from the half space primitive which can only represent planar faces. Vertices are a secondary calculation from the half spaces, and are calculated exactly.

### 12.6.4 Validation of Polytopes

A regular polytope, as defined here, is similar to a polyhedron, but has some significant differences. It is not necessary that a regular polytope is fully bounded, nor does it need to be internally connected. In general, there is no such thing as an invalid regular polytope, but in certain applications, it may be useful to insist that a regular polytope should be:

non-empty – i.e.  $\exists p: p \in O$ .  
 bounded -  $\forall p=(X,Y,Z,Q) \in O: -MQ < X,Y,Z < MQ$ .  
 connected.

Whatever validity requirements are to be enforced, it is assured that, since integer arithmetic is mandated, the same results will be assured on any computing platform.

### 12.6.5 Union of Polytopes

The union of two regular polytopes  $O_1 = \bigcup_{i=1..m_1} C_i$  and  $O_2 = \bigcup_{j=1..m_2} C_j$  is defined as:

$$O_1 \cup O_2 = \left( \bigcup_{i=1..m_1} C_i \right) \cup \left( \bigcup_{j=1..m_2} C_j \right) \quad (8)$$

This is simply the union of a set of convex polytopes, and is therefore a regular polytope. It clearly does not matter in what order this set is specified, and so the union operation is clearly commutative and associative. The cases for the intersection and complement operations, while more complex, are similar.

### 12.6.6 Equality of Volumetric Objects in Space

The functions and predicates, operating on regular polytopes, are defined in the following order, in order to implement them.

The  $\text{Empty}(C)$  predicate is defined first for the convex polytope – where  $\text{Empty}(C)$  is defined as  $\forall p: p \notin C$ . It is fairly simple to implement this while a convex polytope is being constructed. As each half space is added to the definition, the vertices of the

convex polytope are calculated. If all vertices are outside or on the plane of the new half space, the resultant convex polytope is empty.

This is then used to define overlap:  $OV(C_1, C_2)$  as  $\neg \text{Empty}(C_1 \cap C_2)$ .

Which is used to define regular polytope overlap,  $OV(O_1, O_2)$  as  $\exists C_i \in O_1, C_j \in O_2: OV(C_i, C_j)$ .

This is then used to define part of:  $P(O_1, O_2)$  as  $\neg OV(O_1, \overline{O_2})$ .

Which defines equality:  $EQ(O_1, O_2)$  as  $P(O_1, O_2)$  and  $P(O_2, O_1)$ .

Thus the definition is rigorous, and can be applied on any computational platform.

### 12.6.7 Operations Requiring Imprecise Calculations

All the operations that require rounding in the infinite precision rational approach clearly require rounding in this approach, but in addition some other calculations cannot maintain full precision. An example is the calculation of the convex hull. It is not possible to rigorously calculate a convex polytope which is the convex hull of a given regular polytope. The best that can be found is the “convex enclosure”, which is a convex polytope, guaranteed to enclose the regular polytope, and is within one unit of resolution of the true convex hull.

Unlike the infinite precision rational number case, there is no need to run a periodic sweep through the database approximating the objects to reduce storage and processing requirements.

### 12.6.8 Storage Requirements

The storage requirements of this approach are constrained. The half planes are representable using conventional integers – for example, using 32 bit integers for  $A$ ,  $B$  and  $C$ , and 64 bits for  $D$ , as used in the proof-of-concept coding described by [15]. (Note that by the definition of a half space (Section 12.6.1);  $A$ ,  $B$  and  $C$  have a range of  $\pm M$ , but  $D$  has a larger range:  $\pm 3M^2$ ). If it is desired to store vertices, they require extended precision representations, but these are of finite and pre-determined size. (In 3D,  $X$ ,  $Y$ ,  $Z$  require 128 bits, and  $Q$  requires 96 bits).

## 12.7 Conclusions

All spatial data that is represented in a computer is subject to imprecision and rounding errors. These may lead to failures of logic when they are applied, so it is advantageous to be aware of when this occurs. Different approaches have been investigated and compared, with the results tabulated in Table 12.1. No individual approach is optimum in all situations, but the regular polytope, based on dr-rational homogeneous coordinates has been shown to provide rigorous results in all of the most useful operations, without unconstrained growth in storage requirements or the need for periodical approximation sweeps of the database.

**Table 12.1.** Summary of Approaches

	Floating Point	Lazy Calculation	Unrestricted Rational	Domain Restricted Rational
Approximate on data capture	Yes	Yes	Yes	Yes
Approximate on adjustments	Yes	Yes	Yes	Yes
Approximate results to store in database	Yes	Yes	No	No
Approximation sweep through database	No	No	May be useful	No
RCC operations	Not rigorous	Rigorous, then approximated	Rigorously supported	Rigorously supported
Convex Hull	Yes	Yes	Yes	Approximate
Representation storage requirements	Fixed	Unlimited during operations	Unlimited	Fixed

## References

1. Castellanos D (1988) The Ubiquitous pi (Part II). *Mathematics Magazine* Vol 61, 3: 148-161.
2. Courant R, Robbins H (1941) *What is Mathematics?* Oxford University Press, New York.
3. Coxeter HSM (1974) *Projective Geometry*. Springer-Verlag, New York.
4. Gaal SA (1964) *Point Set Topology*. Academic Press, New York.
5. Goldberg D (1991) What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, March 1991 [http://docs.sun.com/source/806-3568/ncg\\_goldberg.html](http://docs.sun.com/source/806-3568/ncg_goldberg.html)

6. IEEE (1985) IEEE Standard for Binary Floating-Point Arithmetic. IEEE Standards Department, New York.
7. ISO-TC211 (2001) Geographic Information - Spatial Schema. IS19107. International Organization for Standards, Geneva.
8. Kahan W (1996) Lecture notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic. From <http://www.cs.berkeley.edu/~wkahan/>.
9. Kazar BM, Kothuri R, van Oosterom P, Ravada S (2008) On Valid and Invalid Three-Dimensional Geometries. In: Advances in 3D Geoinformation Systems. Van Oosterom P, Penninga F, Zlatanova S, Fendel E (eds). Springer, Berlin.
10. Mehlhorn K, Näher S (1999) LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press.
11. Milenkovic VJ (1988) Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence* 37, 377-401.
12. Randell DA, Cui Z, Cohn AG (1992) A spatial logic based on regions and connection. 3rd International Conference on Principles of Knowledge Representation and Reasoning, Cambridge MA, USA, Morgan Kaufmann.
13. Sun (1993) Class Object. In: Java Documentation, from <http://java.sun.com/j2se/1.3/docs/api/java/lang/Object.html>.
14. Thompson RJ (2005) 3D Framework for Robust Digital Spatial Models. In: Large-Scale 3D Data Integration. Zlatanova S, Prospero D (eds). Taylor & Francis, Boca Raton, FL.
15. Thompson RJ (2007) Towards a Rigorous Logic for Spatial Data Representation. *Publications on Geodesy* 65. Netherlands Geodetic Commission, Delft, The Netherlands.
16. Weisstein EW (2002) Hessian Normal Form. In: MathWorld -- A Wolfram Web Resource Retrieved June 2007, from <http://mathworld.wolfram.com/HessianNormalForm.html>.