Chapter 2
On Valid and Invalid Three-Dimensional Geometries

Baris M. Kazar[1], Ravi Kothuri[1], Peter van Oosterom[2] and Siva Ravada[1]

Abstract

Advances in storage management and visualization tools have expanded the frontiers of traditional 2D domains like GIS to 3Dimensions. Recent proposals such as CityGML and associated gateways bridge a long-standing gap between the terrestrial models from the GIS and the CAD/CAM worlds and shift the focus from 2D to 3D. As a result, efficient and scalable techniques for storage, validation and query of 3D models will become a key to terrestrial data management. In this paper, we focus on the problem of validation of 3D geometries. First we present Oracle's data model for storing 3D geometries (following the general OGC/ISO GML3 specifications). Then, we define more specific and refined rules for valid geometries in this model. We show that the solid representation is simpler and easier to validate than the GML model but still retains the representative power. Finally, we present explicit examples of valid and invalid geometries. This work should make it to easy to conceptualize valid and invalid 3D geometries.

2.1 Introduction

The combination of civil engineering surveying equipment, GPS, laser scanning and aerial and satellite remote sensing coupled with strong spatial research has enabled the growth and success of the GIS industry. These industries typically cater to a wide variety of application-specific databases includ-

[1]Oracle USA, Inc.
One Oracle Drive, Nashua, NH 03062, USA
[2]Delft University of Technology, OTB, section GIS Technology,
Jaffalaan 9, 2628 BX the Netherlands
Baris.Kazar, Ravi.Kothuri, Siva.Ravada@Oracle.com, oosterom@tudelft.nl

ing cadastral databases for property/land management, and utility management in addition to popular location-based searching [G84, BKSS90, BDA04, MTP04, MA05] and services [J04, YS05]. Cadastral and Census databases typically maintain exact geometry boundaries for personal properties or administrative boundaries. For example, most countries in the European Union including the Netherlands maintain exact boundaries of properties and use that information for better visualization, property tax calculation etc. The United States Census Bureau maintains the database of exact geometrical shapes for various entities in different levels of the administrative hierarchy. The hierarchy consists of census blocks, block-groups, cities, counties, states etc. In most of these GIS applications, the data is represented using 2Dimensional geometries and stored inside a spatial database. The databases can be quite huge storing up to tens and hundreds of millions of spatial geometries. These geometries are typically stored, managed and queried using efficient processing techniques developed in the spatial and database research community [A94, BKSS90, BKS94, DE04, G84, S89].

Till recently, 3D research has been primarily confined to the graphics, CAD/CAM, gaming, virtual reality and computational geometry communities [FD97, SE02]. The rapid strides in scanner equipment and the declining costs of the same in recent years has generated renewed enthusiasm in 3D data management in various fields like GIS, CAD/CAM, medical and city modeling. Recent proposals such as CityGML [KGP05, OGC06C] bridge a long-standing gap between the terrestrial models from the GIS and the CAD/CAM worlds and shift the focus from 2D to 3D. As a consequence, more and more city departments are planning to utilize 3D models for storing building footprint information for city records databases. Figure 1 shows one such example of a 3D model for the city of Berlin.

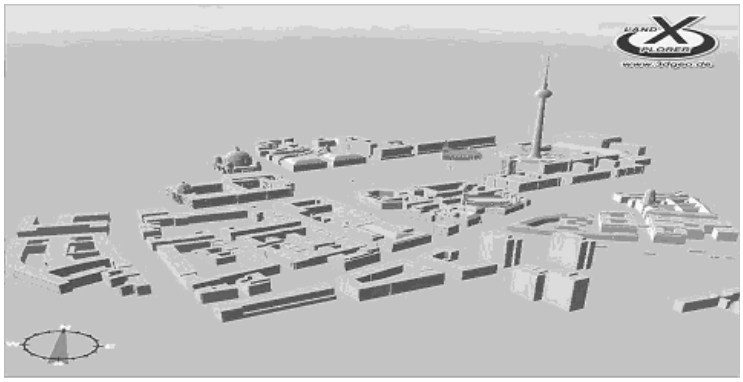


Fig. 2.1 3D Model for the Buildings part in the City of Berlin (courtesy data from the www.citygml.org and LandXplorer visualization tool)

The renewed interest on the acquisition and storage of 3D data in the GIS industry and city departments is likely to result in large amounts of 3D information in the near future. As a result, efficient and scalable techniques for storage and querying of 3D models will be essential to scalable terrestrial data management. To address this need, Oracle Spatial [KAE04] enhanced its SDO_GEOMETRY data type to store 3D data and created additional functionality for the efficient storage, query and management of such 3D data inside the database. One important piece of successful data models is "validation" which verifies whether or not the data generated by third parties, say, a city department, conforms to the data model supported. Validation is an essential and important component of 3D data modeling and enables subsequent operations on valid data to run correctly and efficiently (without the need to check for validity). Standardization organizations such as ISO and OGC have tried to give unambiguous and complete definitions of valid geometric primitives. But as it was already pointed out in [OT03] and [OT04] it turns out that the standards are not unambiguous and complete even in the 2D case of polygons. It should further be noted that ISO [ISO03] provide abstract descriptions, but not a detailed implementation specification enabling system interoperability. This aspect has been provided by OGC/ISO in [OGC99, OGC06a, OGC06b]. It is interesting to note that OGC did improve their definition [OGC06a] of a valid polygon in 2D by adding the condition that the outer boundary must have a counter clockwise orientation compared to the previous definition [OGC99]. What is still missing is a good treatment of tolerance values (also noted in [OT04]), which somehow is implied in the used terms such as spikes and also needed in a final digital machine to decide if two rings of a polygon do touch. Further different vendors of geo-DBMSs did also have different interpretations of a valid polygon, both compared to each other and to the ISO and OGC standards (again see [OT04]). However it must be stated that in the meantime also the producers did take notice of this issue and significant improvements have been noticed (though a systematic benchmark has not been repeated). For 3D geometric primitives there is an abstract ISO specification (again [ISO03]), but this is not the needed implementation specification. In [ASO05] a proposal was made to extend Oracle Spatial with a polyhedron primitive. Despite the fact that much attention was paid to the validation of this primitive, this paper did have some limitations. For example, it was stated that every edge in a valid polyhedron must exactly be used two times, but in this paper we will show that also valid cases exits where an edge is used four (or higher even number) times. Also the paper did focus on a single polyhedron (called in ISO terms 'simple solid'), and did not discuss composite and multi solids. In this paper a more complete set of validation rules are given for the complete set of 3D geometric primitives. Also this paper gives more illustrations of valid and invalid polyhedrons (simple solids), illustrating that quite complicated configurations are possible for which it is not easy to decide whether the primitive is valid.

One could wonder if the subject of valid or invalid solids has not been treated within CAD, with its long-standing 3D basis. There one has to distinguish between the different type of CAD models: voxels, CSG (constructive solid geometry) and boundary representation using (curved) surfaces [M97]. Only the boundary representations are comparable to the discussion of valid solids. However, within CAD the focus is more on the shape of the (curved) boundary, than on the validness of the solid object. Most CAD systems work with CSG and form complex objects from primitive objects. Validity of the resulting objects is typically assumed (due to the underlying nature of the primitives). In practical GIS applications, Boundary representation is more natural. The OGC's GML defines solids in a boundary-representation format [GML03]. GML also allows composite solids to mimic CAD world's convenience of making complex man-made objects. Oracle's model closely follows OGC's specifications in both aspects. In this paper, we illustrate how the Oracle 3D model compares with the GML definitions and present explicit validation rules for checking for validity in a practical implementation. This paper will also help in shedding more light on the valid geometry definitions of GML by providing explicit examples of valid and invalid geometries.

Section 2 defines the data model for storing 3D geometries in Oracle. Section 3 describes rules for determining 'structurally-valid' polygons and surfaces. Note that the definition of polygons is quite similar to those in GML. However, the contribution of that section is Lemma 1 (which can be converted to an algorithm) to prove that polygons with inner rings can be converted to composite surfaces without inner rings. Besides, this section also forms the basis for solid modeling. Section 4 describes solid modeling in Oracle using either 'simple solids' or 'composite solids' and how these differ from the standard definitions in GML. This section also establishes that simpler representations using simple solids without inner rings in faces (or the equivalent composite solids) are sufficient to model 3D geometries (as long as they don't have arcs and parametric curves). By adopting this paradigm (of simple solids) for storing solids, much complexity is avoided and validation algorithms are simplified. Section 5 describes validation rules and examples for collection geometries. The final section discusses the implementation details of these rules (with pointers to the full report) and concludes the paper with pointers for future research.

2.2 3D Geometry in Oracle Spatial

The SDO_Geometry data type for storing 3D data in Oracle has the class structure depicted in Figure 2. The SDO_GEOMETRY is represented by an array of one or more elements. The element array can be used to represent a point, linestring, surface, solid, or collection (heterogeneous collection or homogenous collection such as multi-point, multi-curve, multi-surface, or

multi-solid). Two or more points form a line-string. A closed line-string is called a ring. One or more rings (1 outer and 0 or more inner rings) within the same plane form a polygon [OT04]. One or more polygons form a surface (called a composite if consisting of more than 1 polygon and the surface is connected via shared parts of edges). Two or more outer rings of a composite surface can be on the same or different planes. One outer surface and 0 or more inner surfaces (represented as 1 or more in Figure 2) surfaces form a simple solid (Ssolid in Figure 2) if all the involved surfaces bound a closed volume. One or more adjacent (sharing at least a 2D part of a face) simple solids form a composite solid (referred to as CSolid in Figure 2). As we will show in the next sections, any composite solid can be represented as simple solid by removing the shared polygons. Composite solid is however very convenient from the user's perspective and part of the ISO standard [ISO03]. The collection types are formed as one or more elements of the appropriate type (e.g., one or more points form a multi-point, one or more solids form a multi-solid etc.). Note that the elements of the multi-X (X=surface, or solid) are either disjoint or touching via lower dimensional shared part of their boundary.

Buildings and other architectural elements of city models will likely be represented using simple solids (Ssolids in Figure 2), composite solids (CSolids in Figure 2), multi-solids or collection-type geometries. In this paper, we mainly focus on these types of 'complex' geometries (and do not discuss the rest of the simpler types as they are simple extensions of the 2Dimensional counterparts). The above SDO_GEOMETRY is realized as an implementation object type called SDO_GEOMETRY in the Oracle database. This object has attributes such as SDO_GTYPE, SDO_ELEM_INFO and SDO_ORDINATES. The SDO_GTYPE specifies the type for the geometry. The SDO_ORDINATES stores the ordinates of the vertices of the geometry. The SDO_ELEM_INFO is an array of element descriptors each describing how to connect the ordinates stored in the SDO_ORDINATES field. The following SQL shows the constructor for a Composite surface geometry composed of two (axis-aligned) rectangle polygons. More examples of the SDO_GEOMETRY type with additional examples can be found in Appendix B.

```
SDO_GEOMETRY (
   3003,   -- SDO_GTYPE: 3Dimensional surface type geometry
   NULL, NULL, -- SDO_SRID for coordinate system ID
SDO_ELEM_INFO_ARRAY( -- SDO_ELEM_INFO constructor
  1,     -- starting offset for element
  1006,  -- Element type is a COMPOSITE SURFACE
  2,     -- Number of primitive polygon elements making up
         -- the composite surface
  1,     -- starting offset of first polygon of Composite
         -- surface
  1003,  -- Polygon element type
  3,     -- Axis-aligned rectangle specified by two corners
  7,     -- starting offset of second polygon of Composite
```

```
        -- surface
    1003,  -- Polygon element type
    3      -- Axis-aligned rectangle specified by two corners,
    )
    SDO_ORDINATE_ARRAY ( -- Constructor for SDO_ORDINATES:
       -- Store the actual ordinates of the vertices
       2,0,2, -- 1st corner or 1st axis-aligned rect. polygon
       4,2,2, -- 2nd corner of 1st axis-aligned rect. polygon
       2,0,2, -- 1st corner of 2nd axis-aligned rect. polygon
       4,0,4  -- 2nd corner of 2nd axis-aligned rect. polygon
    )
)
```

Note that the class structure of Figure 2 closely follows the class structure for 3D geometries in GML 3.1.1 [GML03] but with a notable restriction: arcs and parametric curves are not currently supported in the SDO_Geometry. The validation rules and algorithms described in this paper work even when arcs are incorporated into the data model.

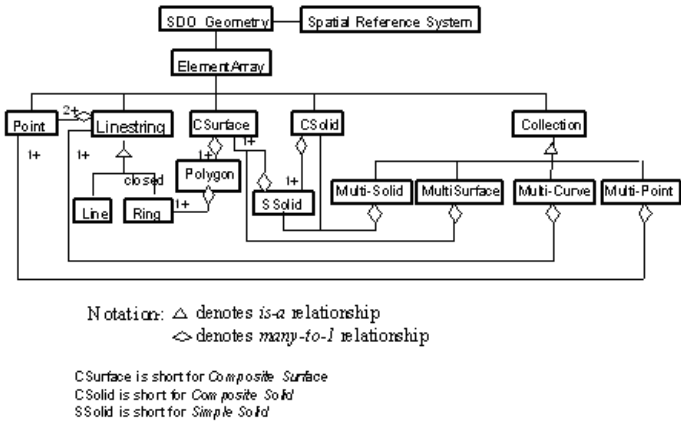


Fig. 2.2 The Class structure for the SDO_GEOMETRY data type

In the next sections, we describe how surfaces, solids and collections are defined in Oracle and present specific validation rules with appropriate examples. We skip points and lines as for these types the validation rules are no different than their 2D counterparts, they are relatively trivial, and there is not much ambiguity in the OGC/ISO standards (no repeated notes, no

self intersection of edges). The only tricky part is the treatment of tolerance values (needed to decide if there is indeed an intersection or equal node), but this will be discussed for surfaces.

2.3 Surfaces

In this section surfaces and their validation rules are discussed, as the closed surfaces are the building block for defining solids (outer and possible inner boundaries). First the ring is discussed, the simple surfaces (polygon) and finally the composite surface. This section does not present new validation rules but it presents the concepts and proves an important lemma needed for the next section on solid modeling.

2.3.1 Rings

Based on our interpretation of the ring definition in OGC/ISO GML3, a ring can be defined as a closed string of connected non-intersecting lines that lie in the same plane. A ring R is typically specified as the sequence of n+1 vertices $R =< V_1, V_2, \ldots, V_n, V_1 >$ where the first and the $(n+1)^{th}$ vertex are the same (to make the ring closed). All other vertices are distinct. Each pair $< V_i, V_{i+1} >$ represents a directed edge connecting the vertex V_i to V_{i+1}. Also note that no two edges of the ring can intersect (no self intersection). The only exception is the first edge $< V_1, V_2 >$ and the last edge $< V_n, V_1 >$ can touch at the vertex V_1.

Validation Rules for a Ring:

- **Closedness Test**: The first and last vertices of the ring are identical.
- **Planarity Test**: All vertices of the ring are on the same plane (within a planarity-tolerance error).
- **Non-intersection of edges**: If edge e_i connects vertex $< V_i, V_{i+1} >$ and edge e_j connects $< e_j, e_{j+1} >$, e_i and e_j have the following properties:
  - If $(j = i+1 \bmod n)$, then e_i and e_j only touch at vertex V_j
  - Otherwise, e_i and e_j do not intersect.
- **Distinct Vertex Test**: Adjacent vertices Vi, Vi+1 should not represent the same point in space. Vi, Vi+1 are considered to duplicates of the same point if the distance between Vi, and Vi+1 is less than a tolerance error.

Note that the planarity tolerance discussed in bullet 2 and tolerance in bullet 4 are different. These tolerance values ensure that spikes and other degenerate cases are invalidated. Note that due to the tolerance also spikes to the inside and outside are not allowed (in [OT04] spikes to the inside were allowed, but not to the outside, which was a bit asymmetric).

2.3.2 Polygon in GML

In GML [GML03], a Polygon is a planar Surface defined by one exterior boundary and zero or more interior boundaries. Each interior boundary defines a hole in the Polygon. GML has the following assertions for Polygons (the rules that define valid Polygons):

a) Polygons are topologically closed.
b) The boundary of a Polygon consists of a set of LinearRings that make up its exterior and interior boundaries.
c) No two Rings in the boundary cross and the Rings in the boundary of a Polygon may intersect at a Point but only as a tangent.
d) Polygon may not have cut lines, spikes or punctures.
e) The interior of every Polygon is a connected point set.

The exterior of a Polygon with one or more holes is not connected. Each hole defines a connected component of the exterior.

Simple Surface, Polygon in Oracle

A polygon P in Oracle strictly adheres to the definition in GML (nothing new here). It is defined as a single contiguous area in a planar space bounded by one outer (or exterior) ring PR_e as the exterior boundary and zero or more (interior) rings $PR_{i1}, \ldots, PR_{ik}$ which are interior to the outer ring and non-overlapping with one another. The inner rings should be oriented in opposite direction as the outer ring. The outer ring itself can always be oriented in any manner. In addition to the mentioned assertions for polygons in GML, we also add the implicitly mentioned co-planarity of points.

Validation Rules for a Polygon:

The rules for polygons can be listed as follows.

- **Validity of Rings**: The rings in a polygon are valid (satisfy closedness, planarity, No Self-intersection tests, and distinct vertex).
- **Co-planarity of Rings**: Since the polygon defines an area in a plane, all rings are on the same plane (within tolerance).

- **Proper orientation**: The inner rings (if any) must have the opposite orientation compared to the outer ring.
- **Single Contiguous Area**: Together the outer ring and the interior rings define a single area. This means the inner rings cannot partition the polygon into disjoint areas.
- **Non-overlapping inner rings**: Inner rings cannot overlap (tolerance) with each other but may touch at a point (under the single contiguous area condition).
- **Inner-outer disjointedness**: Every inner-ring must be inside outer-ring and can only touch (tolerance) at a single point (under the single contiguous area condition).

Note that for 2Dimensional data, Oracle required that the vertices of the outer ring be specified in counterclockwise direction and those of the interior rings in clockwise direction. Such orientation restrictions are not needed for polygons and surface geometries (only the fact that the inner boundaries have opposite orientation compared to the outer boundary). Orientation becomes more important when these polygons/surfaces become components in a solid geometry. Modeling the polygon on the 3D ellipsoid is difficult (co-planarity may not be enforced as points on ellipsoidal surface are not on the same plane) and is not discussed here.

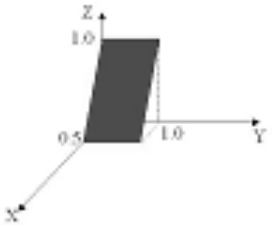


Fig. 2.3 Example of a 3D polygon

2.3.3 Composite Surface

A composite surface is a contiguous area formed as a composition of M non-overlapping polygons. Note that the polygons may or may not be in the same plane. GML does not give any explicit rules here.

Validation Rules for Composite Surfaces:

The validation rules that we propose for composite surface are as follows.

- **Validity of Polygons**: Each of the M polygons has to be a valid polygon.
- **Non-overlapping but edge-sharing nature**: Any two polygons P_i and P_j should not overlap, i.e. if P_i and P_j are in the same plane, the area of intersection of the two polygons has to be zero. However, two polygons may touch (tolerance) in a (part of a) line/edge.
- **Contiguous area**: Every polygon in the composite should be reachable from any other polygon by appropriate tracing of the shared (parts of) edges.

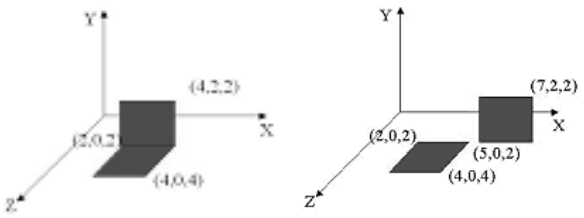


Fig. 2.4 Left: Valid composite-surface. Right: Invalid composite-surface: Not a single contiguous area. Right can be modeled as a homogenous (multi-surface) or heterogeneous collection

Decomposition of a Polygon with inner rings into a Composite Surface

Lemma 1: Any polygon P with an outer ring P_o and (non-overlapping) inner rings $P_{i1}, \ldots, P_{in}$ can always be decomposed into a composite surface S where each polygon has no inner rings with the following characteristics:

- Every edge/vertex in P is an edge in one of the polygons of S.
- Area(P) = Union of Area(Q) for all Q in S.
- No polygon of S has an inner ring
- Every edge in S
  - Either belongs to P and is traversed only once in S
  - Or is an edge inside the outer ring P_o and is traversed twice.

Proof: see Appendix A.

2.4 Solids

GML and ISO define specific representations for solids. Oracle's definition of the solid is quite equivalent except that it does not allow arcs and parametric curves in the solid specification. The GML definition of a solid is as follows [GML03]:

The extent of a solid is defined by the boundary surfaces (shells). A shell is represented by a composite surface, where every shell is used to represent a single connected component of the boundary of a solid. It consists of a composite surface (a list of orientable surfaces) connected in a topological cycle (an object whose boundary is empty). Unlike a Ring, a shell's elements have no natural sort order. Like Rings, shells are simple. The element 'exterior' specifies the outer boundary of the solid. Boundaries of solids are similar to surface boundaries. In normal 3Dimensional Euclidean space, one (composite) surface is distinguished as the exterior. In the more general case, this is not always possible. The element 'interior' specifies the inner boundary of the solid. Boundaries of solids are similar to surface boundaries.

In this paper, we will only focus on solid modeling in Euclidean spaces and restrict our attention to solids without arcs and parametric curves. Oracle supports two types of solids: a simple solid, and a composite solid. Both representations are equivalent but a composite solid may be more convenient to form for the user. Further composite solids are also required by the OGC/ISO standards.

One problem with this definition is: it allows inner rings in a polygon of a composite surface. Consider a solid such as that in the following figure that has an inner ring on its top surface. Is this solid, valid or invalid? The answer depends on whether the inner ring in the top surface is complemented with inner walls too as shown in the subsequent figure.

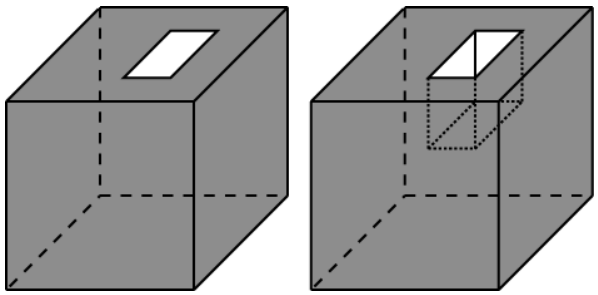


Fig. 2.5 Is the white inner ring an empty surface-patch on the solid, not matched with other faces as shown on the left side? (In that case, the solid is not valid.) Or does it connect to other 'inner' faces in which case it could become valid as shown on the right side

Oracle offers a simpler representation: all polygons only have an outer ring but no inner ring (thus avoiding unnecessary computation). It turns out that this simpler representation does not lose any power (i.e. can represent any solid with polygons that have inner rings). Oracle supports two variants, simple solids and composite solids, that both exhibit this property.

2.4.1 Simple Solids in Oracle

In Oracle, a simple solid is defined as a 'Single Volume' bounded on the exterior by one exterior composite surface and on the interior by zero or more interior composite surfaces. To demarcate the interior of the solid from the exterior, the polygons of the boundary are oriented such that their normal vector always point 'outward' from the solid. In addition, each polygon of the composite surfaces has only an outer ring but no inner ring. (This is a restriction compared to the GML definitions, but without loosing any expression power).

Validation Rules for Simple Solids:

Based on these above definitions, we can define the rules/tests for validation of solids (again all operations are using tolerance values):

- **Single Volume check**: The volume should be contiguous.
  - **Closedness test**: The boundary has to be closed. [Z00] show that the vector sum of the edges in the boundary traversal should be zero (i.e. every edge on the boundary needs to be traversed even number of times: note that some implementations check for just 2 times but that may disallow some valid solids as shown in Figure 9). Necessary condition but not sufficient (Figure 11 left, Figure 12 left, Figure 13 left are invalid)
  - **Connectedness test**: For sufficiency, volume has to be connected. (Figure 11 right, Figure 12 right, Figure 13 right are valid). This means each component (surface, solid) of the solid should be reachable from any other component.
- **Inner-outer check**:
  - Every surface marked as an inner boundary should be 'inside' the solid defined by the exterior boundary.
  - Inner boundaries may never intersect, but only touch under the condition that the solid remains connected (see above)
- **Orientation check**: The polygons in the surfaces are always oriented such that the normals of the polygons point outward from the solid that

they bound. Normal of a planar surface is defined by the right-hand thumb rule (if the fingers of the right hand curl in the direction of the sequence of the vertices, the thumb points in the direction of the normal). The volume bounded by exterior boundary is computed as positive value if every face is oriented such that each normal is pointing away from the solid due to the Green's Theorem. Similarly, the volume bounded by interior boundary is computed as negative value. If each exterior and interior boundary obeys this rule and they pass connectedness test as well, then this check is passed.

- **Element-check**: Every specified surface is a valid surface.
- **No-inner-ring in polygons**: In the composite surfaces of a solid, no inner rings are allowed.

A solid cannot have polygons that overlap. Due to the use of tolerances some very thin volume parts could collapse to spikes (dangling faces or edges). However, it is not possible to have spikes (either linear or areal shaped) as it is not allowed to have the vector sum of edges unequal to 0.

Theorem 1: Any valid solid S where polygons have inner rings can also be represented as a simple solid without inner rings in the faces.

Proof: Consider a polygon P that has interior rings in S. During a traversal of P, every edge of P is traversed just once. Each of these edges is traversed a second time in the traversal of the rest of the polygons that close the solid. Replace polygon P (that has interior rings) in S by its equivalent composite surface consisting of the no-interior-polygons $P_1, \ldots, P_k$ as in Lemma 1. Since every edge in P is traversed only once during a traversal of $P_1, \ldots, P_k$, the boundary is preserved. All edges that are in $P_1, \ldots, P_k$ but not in P are traversed exactly twice in opposite directions and are cancelled out in the traversal. Thus preserving the solid-closedness properties. Other properties are also likewise preserved.

In Figure 6, simple solid has an outer boundary represented by a closed composite surface and a hole (Hole) represented by an inner composite surface. These two solids share a face. This solid is invalid since it violates the inner-outer disjointedness rule of a simple solid. However, this solid can be modeled as a valid simple solid represented by a single outer composite surface without any inner surfaces (solid with dent).

The composite-solid in Figure 7 is composed of Solid 1 and Solid 2. Even though solids have common (shared) area, Solid 2 is inside Solid 1, which violates the No-volume-intersection of composite-solids and hence invalid.

The examples in Figures 8-13 give a number of valid and invalid simple solids. The captions of the figures do explain why. From these figures it becomes clear that the validation includes a non-trivial topological analysis of the inner and outer boundary elements. Let us consider another example. The geometry in Figure 15 is designated as a composite-solid geometry consisting of simple solids: Solid 1 and Solid 2. Solid 2 has an outer boundary and an additional surface patch intersecting (overlapping) one of its faces. This violates the No-surface-patch rule of simple solids and hence is an in-

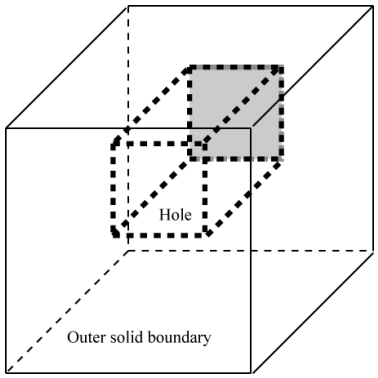


Fig. 2.6 Simple Solid: invalid if modeled as outer, inner surfaces. Note that the back face (i.e. shaded area) of the inner solid boundary is partly shared with the back face of the outer solid boundary

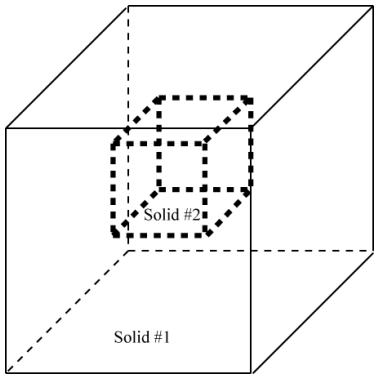


Fig. 2.7 Invalid composite solid because solid elements cannot be inside the other. If modeled as a simple solid, the object will be invalid as it has two outer boundaries (which are disconnected)

valid geometry. The geometry, however, can be modeled as a (heterogeneous) collection.

2.4.2 Composite Solid in Oracle

In addition to a simple solid, Oracle (and GML too) also allows the specification of a 'composite solid'. In Oracle, a composite solid is a combination of n simple solids. Compared to the simple solid definition in Section 4.1, what this allows is the overlap of the polygons of different simple-solids but the

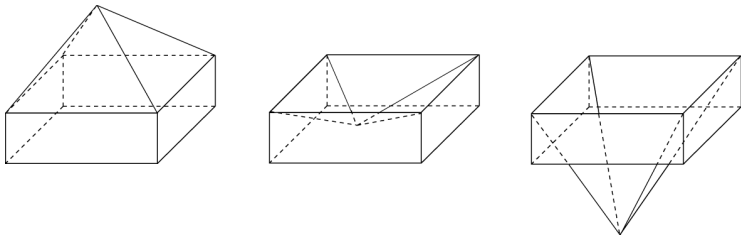


Fig. 2.8 The importance of checking the intersection between faces: all 3 simple solids have the same node-edge connectivity, but the last (rightmost) one has intersecting faces and is therefore invalid

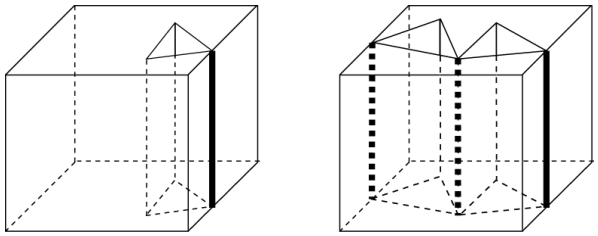


Fig. 2.9 Simple solid with 'inner' boundary touching the outer boundary in one line (fat edges are used 4 times). The left solid is valid, while the right simple solid is invalid (together the two 'inner' boundaries separate the solid into two parts). Note that both solids do not really have inner boundaries (it is a more complex single outer boundary causing through holes touching the outer boundary in other places)

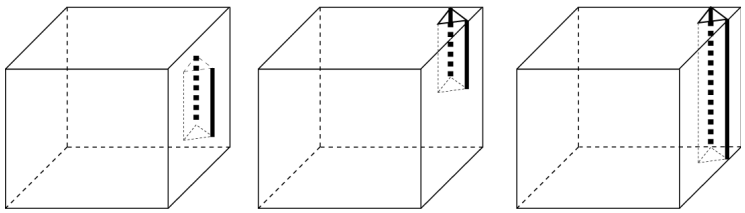


Fig. 2.10 Simple solid with inner boundary touching the outer boundary in two lines (fat edges are used 4 times). The left solid is the only solid with a true inner boundary (touching the outer boundary in two lines: right outer boundary in the fat line and the back outer boundary in the fat dashed line). The left is a valid simple solid. The middle solid is also valid (because inner boundary does not continue through the whole), while the right simple solid is invalid (as inner boundary does separate the solid into two parts). Again note that the middle and right solids do not really have inner boundaries (it is a more complex single outer boundary)

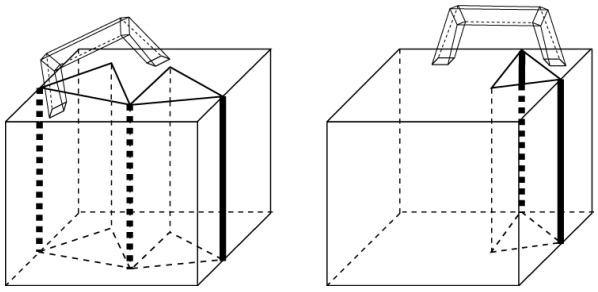


Fig. 2.11 Invalid simple solids of previous figures becoming valid via adding an additional handle making it possible to travel from one part to another part of the object (completely via the interior). Note: where handle touches the face, a part of the faces is removed (that is an interior ring is added within the exiting face to create the open connection). So, all faces have always (and everywhere) on one side the object and on the other side something else (outside, where the normal is pointing to)

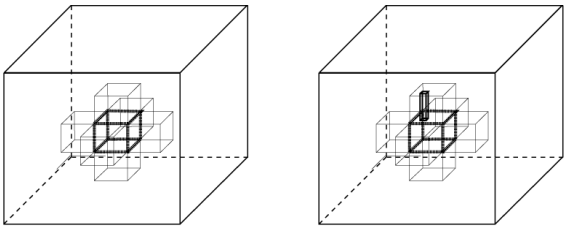


Fig. 2.12 Left: simple solid with 6 internal (cube-shaped) boundaries separating the big cube into two parts (the internal one draw with fat lines is implied by the 6 boundaries of the 6 smaller cube-shaped holes). Therefore the left simple solid is invalid (note that removing one of the 6 holes, makes it valid again). Right: Invalid simple solids of previous figures becoming valid via adding an additional handle making it possible to travel from one part to another part of the object (completely via the interior). Right: the two parts are connected via a 'pipe' making it a valid simple solid again

boundary is still closed. Note that this does not allow overlapping polygons in the same simple solid but only across multiple simple solids.

Following theorem shows the equivalence of a 'simple solid' and a 'composite solid'. Composite solids are defined for convenience: it is easier to combine two or more simple solids and make a composite. For the same reason, they are also included in GML specification too. However, the composite solids in GML do allow inner rings in polygons whereas the Oracle model does not but still equivalent (from Theorem 1 and Theorem 2).

Theorem 2: Every valid composite solid can also be represented as a simple solid.

Proof: Composite solid always has solids attached to each other via partially or fully shared faces. Having detected these shared faces, one can get rid

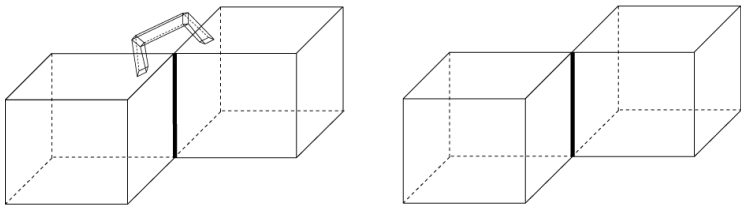


Fig. 2.13 Left: valid simple solid (fat edge still used 4 times), but handle is added through which it is possible to travel from one part to the other part via the interior only, Right: invalid simple solid with one edge being used four times (fat line)

of these faces and redefine the solid without these shared areas. Figure 14 illustrates an example.

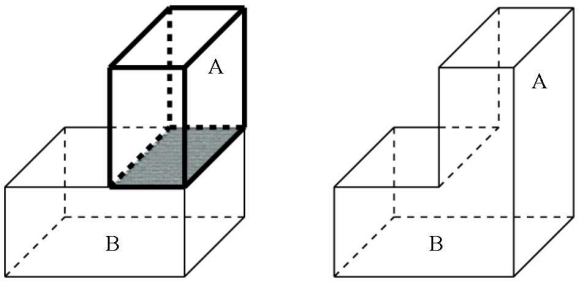


Fig. 2.14 (a) Composite Solid consisting of two simple solids A and B. Shared portion of bottom face of A (i.e. shaded area) in both A and B. (b) Equivalent Simple Solid. Shared portion of bottom face of A should not be included in the boundary.

Validation Rules for Composite Solids

- **Component Validity**: Each component simple solid of a composite is valid.
- **Shared-face but no-volume intersection**: Intersection of two simple solid components of a composite solid has to be a zero volume (can be non-zero area).

- **Connectedness**: The volume of the composite is contiguous, i.e. we can go from any point in one component to any other component without going out of the composite solid.

Since composite solids are equivalent to a simple solid, an alternate way is simply convert the composite to the equivalent simple solid and then validate the resulting solid.

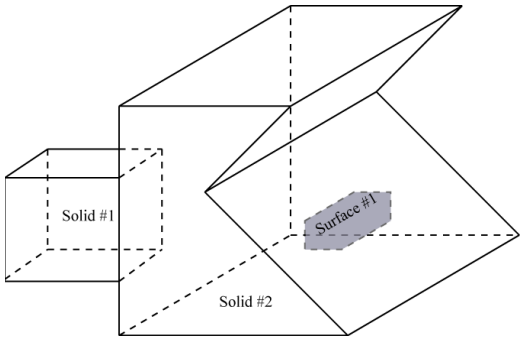


Fig. 2.15 Invalid Composite Solid: Cannot have surface patches on a solid

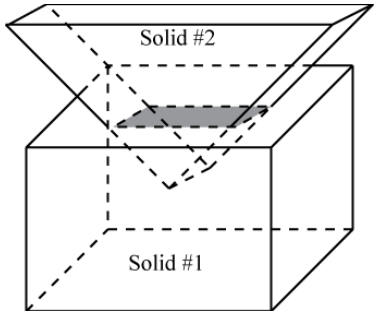


Fig. 2.16 If modeled as a Composite Solid, the object is invalid due to intersection of solid elements. If modeled as a simple solid, the object is invalid due to 'overlapping polygons' of composite surface

Consider a third example on composite-solids in Figure 16, which is composed of a cube and a triangular prism where a prism goes into the cube through its top face. This is an invalid geometry because it violates the No-volume intersection rule of composite-solids.

2.5 Collections

Following GML [GML03] specifications, collections in Oracle can be either homogenous or heterogeneous. Let us look at each of these in turn.

2.5.1 Homogenous Collections

A homogenous collection is a collection of elements, where all elements are of the same type. A homogenous collection can be either a multi-point, multi-line, multi-surface, or multi-solid corresponding to the element type point, line, surface and solid. For example, in a multi-solid, all elements have to be (valid) solids.

Validation Rules for Homogenous Collections:

- All elements of same type and conform to the homogenous collection type (multi-point, multi-line, multi-surface, multi-solid).
- Each element should be valid.

In addition to the above rules, Oracle also adds a specific rule for determining disjointedness of elements in a homogeneous collection in validation procedures (this is a deviation from the GML specification). The reason to add the disjointedness is to distinguish between composite solids and multi-solids. If the user does not want this restriction, he can model it as a heterogeneous collection.

The geometry in Figure 16 is also invalid as a multi-solid because it violates the disjointedness rule for multi-solids. Note that the disjointedness in multi-solids not only implies no-volume intersection but also no-area intersection. The latter is represented by the geometry in Figure 17, which is invalid due to the disjointedness (no-area intersection) rule of multi-solids.

2.5.2 Heterogeneous Collections

In a heterogeneous collection[1], the elements can be a mixture of different types. For example, a building (simple solid) with the windows/doors (surfaces) can be modeled as a heterogeneous collection.

[1] In Oracle documentation, a heterogeneous collection is referred to simply as a 'collection'. All homogeneous collections are referred to by their names (e.g. multi-line, multi-point, multi-surface, multi-solid).

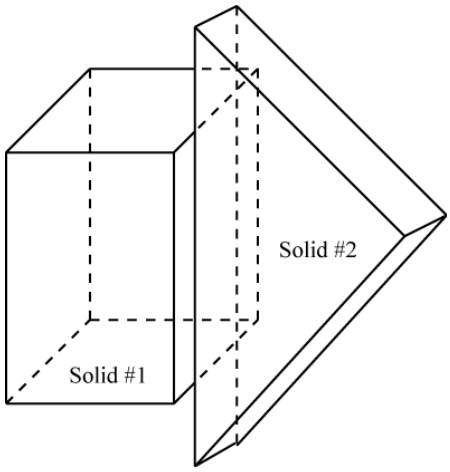


Fig. 2.17 Invalid multi-solid because solid elements are non-disjoint (but a valid composite solid

Validation Rules for Heterogeneous Collections:

- Each element of the collection should be valid.

Figure 18 shows an example of a building modeled as a heterogeneous collection: The building shape is modeled as a solid whereas the windows and doors are modeled as surfaces.

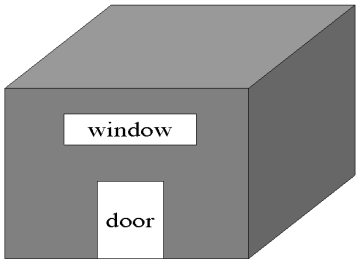


Fig. 2.18 Building modeled as a Heterogeneous collection

2.6 Discussions and future work

In this paper, we presented a data model for storing 3D objects that occur in the rapidly evolving area of city modeling. We examined different types and described rules for validating these types of geometries based on OGC/ISO standards [ISO03, OGC6a, OGC6b, OGC6c]. We then developed and presented in this paper the more detailed, unambiguous validation rules and applied them to different examples to determine validity or otherwise. Such a data model and validation rules can form the backbone of 3D data models needed for proper interoperability (having exactly the same definition of the geometric primitive during the exchange of 3D data). Our specific contributions include: simpler representation for solids (by eliminating the need for inner rings in polygons) thereby simplifying the validation of solids, explicit rules for validation of each geometry type, and concrete examples for valid and invalid 3D geometries that bring out the concepts of the validation model. This model and validation rules have been implemented in the Oracle 11g database (the implementation details have not been included due to space restrictions). The rules that are more involved to implement are the Closedness test and the Reachability/Connectedness tests. Full details on our implementation of these tests are available in [Oracle07]. Future work could concentrate on performance evaluation of the 3D data model, validation and query on large-scale city models. Specific optimizations based on 3D query window sizes [KR01] could be investigated. Besides, the 3D data modeling using SDO_GEOMETRY can be compared with other models such as Tetrahedron-based models [POK06] in terms of functionality and performance. Since the area of 3D modeling in databases is a fledgling topic, we need to investigate on easy approaches for deriving 3D data by extruding 2D footprints. By first converting arbitrary 2D polygons to a composite surface, via Lemma1, and then extruding them to 3D may be a good approach. Fast algorithms for doing this conversion may also be investigated (there exist more efficient algorithms than the recursive construction in Lemma 1). Another important aspect in 3D modeling is visualization. Popular tools like Google Sketchup work well with 3D surfaces but do not understand 3D solids. Other tools such as Landxplorer and Aristoteles visualize all types of 3D GML geometries. Commercial products from Autodesk work with their own proprietary 3D formats and are planning to publish to GML geometries. The Xj3D tools that are popular in the gaming community are also interoperating with the GML forums. Using appropriate functions, the 3D data stored as SDO_GEOMETRY can be converted to GML thereby opening it further to other interoperable formats. Future work can be devoted to developing fast rendering of the native 3D geometries.

Acknowledgements

We would like to thank Friso Penninga for proof reading an earlier version of this paper. The contribution of Peter van Oosterom to this publication is the result of the research programme 'Sustainable Urban Areas' (SUA) carried out by Delft University of Technology and the Bsik RGI project 3D Topography.

References

[Oracle07]. 'Validation Rules and Algorithms for 3D Geoemtries in Oracle Spatial', 2007.

[ASO05] Calin Arens, Jantien Stoter and Peter van Oosterom, Modelling 3D spatial objects in a geo-DBMS using a 3D primitive, In: Computers & Geosciences, Volume 31, 2, 2005, pp. 165-177.

[A94] Lars Arge, Mark de Berg, Herman J. Haverkort, Ke Yi: The Priority R-Tree: A Practically Efficient and Worst-Case Optimal R-Tree. SIGMOD Conference 2004: 347-358.

[BKSS90] Beckmann, N., Kriegel, H., Schneider, R. and Seeger, B., The R* tree: An efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 322-331, 1990.

[BKS94] Brinkhof, T., Kriegel, H., and Seeger, B., Efficient processing of spatial joins using R-trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 237-246, 1994.

[BDA04] Nagender Bandi, Chengyu Sun, Amr El Abbadi, Divyakant Agrawal: Hardware Acceleration in Commercial Databases: A Case Study of Spatial Operations. VLDB 2004: 1021-1032.

[DE94] Dimitris Papadias, Yannis Theodoridis, Timos K. Sellis, Max J. Egenhofer: Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees. SIGMOD Conference 1995: 92-103.

[FD97] Foley, van Dam, Feiner, Hughes. Computer Graphics: Principles and Practice, The Systems Programming Series, 1997.

[G84] A. Guttman. R-trees: A dynamic index structure for spatial searching. Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 4757, 1984.

[GML03] The Geographic Markup Language Specification. Version 3.1.1, http://www.opengeospatial.org.

[GSAM04] Thanaa M. Ghanem, Rahul Shah, Mohamed F. Mokbel, Walid G. Aref, Jeffrey Scott Vitter: Bulk Operations for Space-Partitioning Trees. ICDE 2004.

[HKV02] Marios Hadjieleftheriou, George Kollios, Vassilis J. Tsotras, Dimitrios Gunopulos: Efficient Indexing of Spatiotemporal Objects. EDBT 2002: 251-268.

[ISO03] ISO/TC 211/WG 2, ISO/CD 19107, Geographic information - Spatial schema, 2003.

[J04] Christian S. Jensen: Database Aspects of Location-Based Services. Location-Based Services 2004: 115-148.

[KGP05] Th. H. Kolbe, G. Gröger, L. Plümer. CityGML: Interoperable Access to 3D City Models, In: Oosterom, P, Zlatanova, S., Fendel E. M. (editors) Geo-information for Disaster Management, Springer, pages 883-899, 2005.

[KAE04] Kothuri, R. Godfrind A, Beinat E. 'Pro Oracle Spatial', Apress, 2004.

[KSSB99] Kothuri R., Ravada S., Sharma J., and Banerjee J., Indexing medium dimentionality data, in Proc. ACM SIGMOD Int. Conf. On Management of Data, 1999.

[KR01] Kothuri, R., Ravada, S., Efficient processing of large spatial queries using interior approximations, Symposium on Spatio-Temporal Databases, SSTD, 2001.

[L07]. Jung-Rae Hwang, Ji-Hyeon Oh, Ki-Joune Li: Query Transformation Method by Dalaunay Triangulation for Multi-Source Distributed Spatial Database Systems. ACM-GIS 2001: 41-46.

[MP01] Nick Mamoulis, Dimitris Papadias: Multi-way Spatial Joins, ACM TODS, Vol 26, No. 4, pp 424-475, 2001.

[MTP05] Nikos Mamoulis, Yannis Theodoridis, Dimitris Papadias: Spatial Joins: Algorithms, Cost Models and Optimization Techniques. Spatial Databases 2005: 155-184

[MA04] Mohamed F. Mokbel, Ming Lu, Walid G. Aref: Hash-Merge Join: A Non-blocking Join Algorithm for Producing Fast and Early Join Results.

ICDE 2004: 251-263.

[M97] Mortenson, M. Geometric Modelling, second ed. Wiley, New York 523pp., 1997.

[OGC99] Open GIS Consortium, Inc., OpenGIS Simple Features Specification For SQL, Revision 1.1, OpenGIS Project Document 99-049, 5 May 1999.

[OGC06a] Open Geospatial Consortium Inc., OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture, Version: 1.2.0, Reference number of this document: OGC OGC 06-103r3, 2006.

[OGC06b] Open Geospatial Consortium Inc., OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option, Version: 1.2.0, Reference number of this document: OGC 06-104r3, 2006.

[OGC06c] Open Geospatial Consortium Inc., Candidate OpenGIS CityGML Implementation Specification , Reference number of this document: OGC 06-057r1, 2006.

[POK06] Penninga, F., van Oosterom, P. & Kazar, B. M., A TEN-based DBMS approach for 3D Topographic Data Modelling, in Spatial Data Handling 2006.

[S89] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1989.

[SE02] Schneider J. P., and Eberly, D.H. Geometric Tools for Computer Graphics, Morgan Kaufman, 2002.

[TS96] Y. Theodoridis and T. K. Sellis, A model for the prediction of r-tree performance, In Proc. of ACM Symp. on Principles of Databases, 1996.

[OT03] P.J.M. van Oosterom, C.W. Quak and T.P.M. Tijssen, Polygons: the unstable foundation of spatial modeling, ISPRS Joint Workshop on 'Spatial, Temporal and Multi-Dimensional Data Modelling and Analysis', Québec, October 2003.

[OT04] P.J.M. van Oosterom, C.W. Quak and T.P.M. Tijssen, About Invalid, Valid and Clean Polygons. In: Peter F. Fisher(Ed.); Developments in Spatial Data Handling, 11th International Symposium on Spatial Data Handling, 2004, pp. 1-16

[YE06] Yohei Kurata, Max J. Egenhofer: The Head-Body-Tail Intersection for Spatial Relations Between Directed Line Segments. GIScience 2006: 269-286 2005.

[YS05] Jin Soung Yoo, Shashi Shekhar: In-Route Nearest Neighbor Queries. GeoInformatica 9(2): 117-137 (2005).

[Z00] Zlatanova, S. On 3D Topological Relationships, DEXA Workshop, 2000.

Appendix A

Lemma 1: Any polygon P with an outer ring P_o and (non-overlapping) interior rings $P_{i1}, \ldots, P_{in}$ can always be decomposed into a composite surface S where each polygon has no inner rings with the following characteristics:

- Every edge/vertex in P is an edge in one of the polygons of S.
- Area(P) = Union of Area(Q) for all Q in S.
- No polygon of S has an inner ring.
- Every edge in S
  - either belongs to P and is traversed only once in S
  - or is an edge inside the outer ring P_o and is traversed twice.

Proof: By induction on the number of inner rings. Let n=1. The polygon P has one inner ring. On the inner ring find the two extreme (min, max) points I_{min}, I_{max} along a specific axis of the plane of the polygon. Extend the min, max points along the dimension to meet the outer ring O of the polygon at O_{min}, O_{max}. The polygon P is now P_1 and P_2: the new edges $< O_{min}, I_{min} >$ and $< O_{max}, I_{max} >$ are traversed twice: once in P_1 and another time in reverse direction in P_2. All other edges belong to P and are in either P_1 or P_2. The new edges do not cross any other edges, as O_{min} and I_{min} are the extreme vertices. Hence P_1 and P_2 are valid polygons.

Hypothesis: Let the lemma be true for n=m-1.

Consider a polygon P with m inner rings. Again, sort the rings along a specific axis of the plane and find the ring that has a vertex with the min value on this axis (i.e. closest to the outer ring). Connect the vertex to either side of the outer ring along this axis. This 'cutting line' cuts some inner rings (at least 1): For each such ring identify the first and the last vertex along the cutting line for the ring. These vertices are the endpoints for that inner ring and are connected to other rings that are cut or to the outer ring boundary. The cutting line cuts the outer ring and some inner rings into two parts. Without loss of generality, let us assume it cuts these rings into top, and bottom halves. To identify these halves precisely for each ring r, first determine the extreme (leftmost and rightmost) points $v_L(r)$, $v_R(r)$ on

the ring r that are also on the cutting line. The line string $v_L(r)$, $v_R(r)$ in counterclockwise direction determines the top half for ring r. The line string $v_R(r)$ to $v_L(r)$ (in counterclockwise direction) determines the bottom half. Figure 19 shows an example. Ring EJHGF is cut at two extreme points E (leftmost) and G (rightmost on ring and cutting line). The ring is split into a linestring EFG (top half) and GHJE (bottom half). The new edges will be AE, EA, GK, KG, MC, CM.

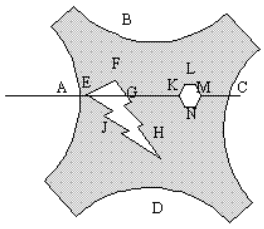


Fig. 2.19 Example of a cutting line and a Polygon with 1 outer and 2 inner rings.

The linestrings of the top halves along with the new connecting edges between the rings form one polygon. In Figure 19, AE, EFG, GK, KLM, MC, CBA becomes the outer ring of the first polygon P_1. Likewise, the bottom half forms along with the new edges in reverse direction form another polygon. In Figure 19, CM, MNK, KG, GHJE, EA, ADC form the outer ring of the second polygon P_2.

Note that for a specific ring s, the points $v_L(s)$ and $v_R(s)$ may be the same, i.e. the cutting line intersects at a point of the ring s. Then only one of the two halves can include the ring, the other half will just include the point. Figure 20 shows an example. The inner ring, KNK, on the right touches the cutting line. The linestring KNK is added to the bottom half and the point K is added to the top half.

If the cutting line cuts at multiple points in a ring q, then $v_L(q)$ will be the leftmost and $v_R(q)$ will be the rightmost. Figure 20 shows an example. Even though the cutting line cuts the first ring at E, H, and G, the end points will be E and G. So, the top half will have a linestring EHFG, and the bottom half will have a line string GJE.

The remaining inner rings that are not cut by the cutting line are split between P_1 and P_2 depending on which outer ring (of P_1 or P_2) that they are inside. Since there is at least 1 inner ring that is split, P_1 and P_2 have at most m-1 rings. The inner rings were non-overlapping in P, so they will still be

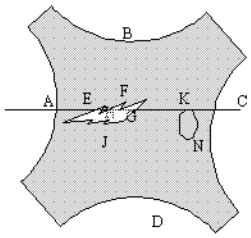


Fig. 2.20 Example of a Polygon with two inner rings: One ring just touches the cutting line; another inner ring intersects at multiple points

non-overlapping in P_1 and P_2. Besides the cutting line does not cut the inner rings of P_1 and P_2: so they still do not touch the exterior rings of P_1 or P_2.

The only new edges in P_1 or P_2 are the 'new edges'. Each such new edge is traversed once in one direction in P_1. All other edges also existed in P and are traversed only once (in the same direction as in P).

Since there is at least one inner ring that is split in this process, the polygons that are formed have at most m-1 inner rings. By induction hypothesis, each of these polygons satisfies Lemma 1 resulting in two composite surfaces one for top half of the cutting line and another for the lower half. A combination of these two composites, is also a composite surface and obeys the above properties thus proving the induction.

For example, in Figure 19, the polygon connecting vertices A, D, B, C is decomposed into a composite surface consisting of two polygons: AEFGKLMCBA and CMNKGHJEADC. Note that each of them has at most m-1 inner rings, and each of the two polygons is a valid polygon, and the resulting surface combining these two polygons is a valid composite and each new edge is traversed twice and all existing edges are traversed only once, thus proving the lemma.

Appendix B

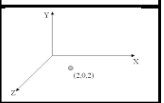
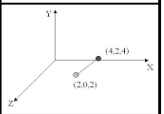
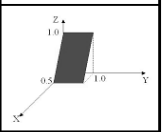
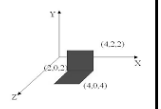
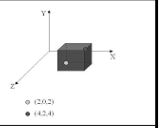
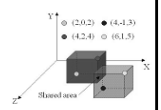
SDO_GTYPE	Name/Type of Element, Value of Etype	Interpretation (has one or more numbers)	SDO_GEOMETRY Representation	Picture
3001 (3005 if N>1)	Point,1	1	SDO_GEOMETRY(3001,NULL,SDO_POINT_TYPE(2,0,2), NULL,NULL)	
3002 (linestring)	Line,2	1: Connected by Line	SDO_GEOMETRY(3002,NULL, NULL,SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(2,0,2,4,2,4))	
3003 (surface)	1 Outer (1003) and [0,n] Interior (2003)	1: *Planar polygon surface 3: Planar rectangle surface 4: Planar circle surface	SDO_GEOMETRY(3003,NULL,NULL, SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0.5,0.0,0.0,0.5,1.0,0.0,0.0,0.0,1.0,1.0, 0.0,0.0,1.0,0.5,0.0,0.0))	
3003 (surface)	Composite Surface: Simple Outer (1006) and Inner (2006) Planar Polygons	N: number of planar polygon surfaces	SDO_GEOMETRY(3003,0,NULL,SDO_ELEM_INFO_ARRAY(1,1006,2,1,1003,3,7,1003,3),SDO_ORDINATE_ARRAY(2,0,2,4,2,2,2,0,2,4,0,4))	
3008 (solid)	Simple Solid (1007) w/ Inners (2006s)	1: Explicit format 3: *Box	SDO_GEOMETRY(3008,NULL,NULL, SDO_ELEM_INFO_ARRAY(1,1007,3),SDO_ORDINATE_ARRAY(2,0,2,4,2,4))	
3008 (solid)	Composite Solid, 1008	N: the # of solid elements (etype=1007)	SDO_GEOMETRY(3008,NULL,NULL, SDO_ELEM_INFO_ARRAY(1,1007,3,7,1007,3),SDO_ORDINATE_ARRAY(2,0,2,4,2,4,4,-1,3,6,1,5))	

Fig. 2.21 Examples of different types of 3D geometries