



中华人民共和国国家标准

GB/T 23707—2009/ISO 19107:2003

地理信息 空间模式

Geographic information—Spatial schema

(ISO 19107:2003, IDT)

2009-05-06 发布

2009-10-01 实施



中华人民共和国国家质量监督检验检疫总局
中国国家标准化管理委员会

发布

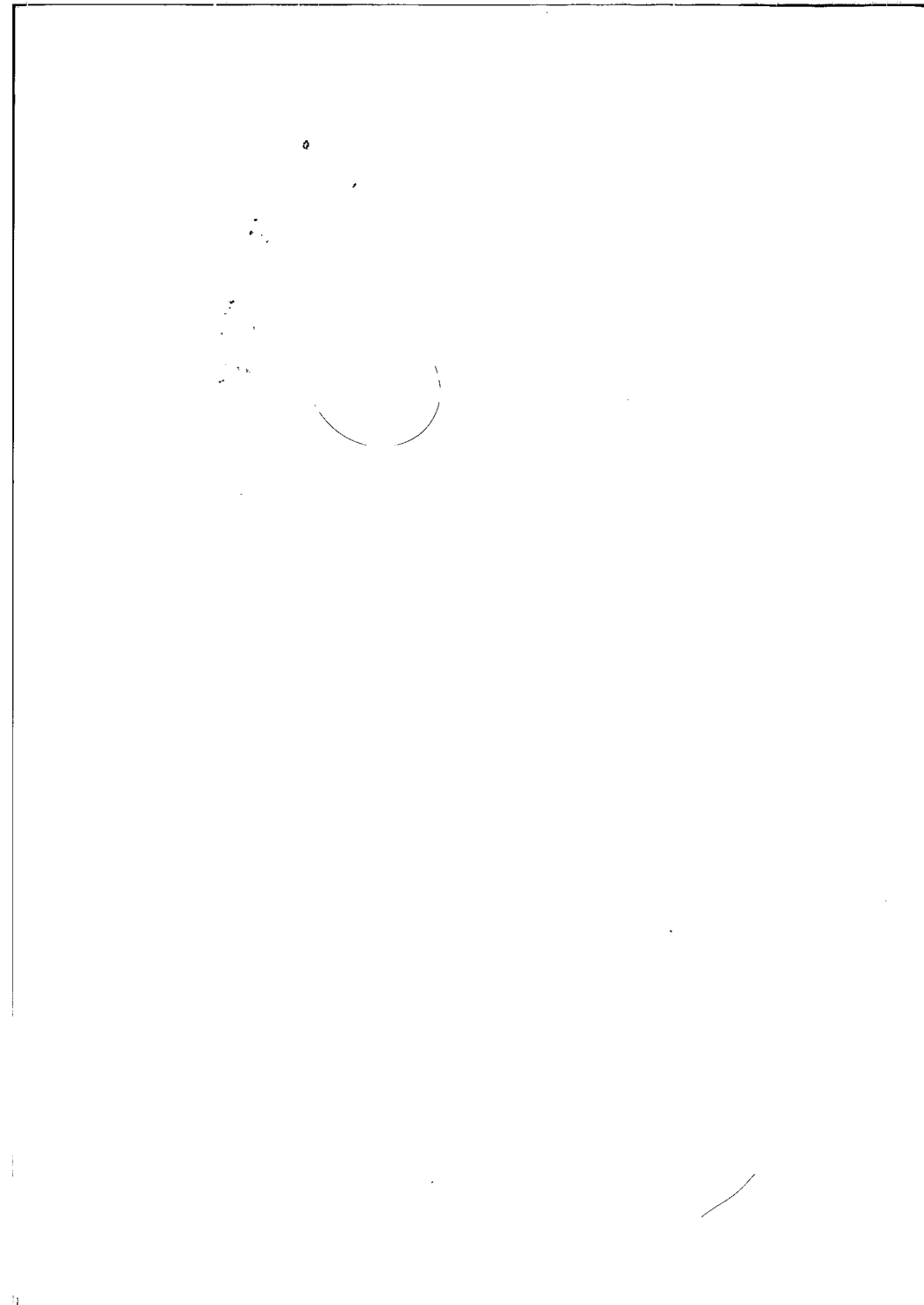


目 次

前言	V
引言	VI
1 范围	1
2 一致性	1
2.1 概述	1
2.2 一致性类	2
3 规范性引用文件	3
4 术语和定义	3
5 符号、注记与缩略语	12
5.1 表达与注记	12
5.2 组织	17
5.3 缩略语	18
6 几何包	18
6.1 语义	18
6.2 几何根包	20
6.3 几何单形包	26
6.4 坐标几何包	37
6.5 几何聚集形包	68
6.6 几何复形包	70
7 拓扑包	74
7.1 语义	74
7.2 拓扑根包	76
7.3 拓扑单形包	79
7.4 拓扑复形包	91
8 派生拓扑关系	93
8.1 概述	93
8.2 布尔或集合算子	93
8.3 Egenhofer 算子	94
8.4 全拓扑算子	95
8.5 组合	95
附录 A (规范性附录) 抽象测试套件	96
附录 B (资料性附录) 按概念分类组织的术语与定义	105
附录 C (资料性附录) 空间概念模式示例	113
附录 D (资料性附录) 应用模式示例	119
附录 NA (资料性附录) 不同维数的几何对象与拓扑对象构成对照	128
参考文献	129
图 1 UML 关联示例	14

图 2 UML 包依赖关系示例	17
图 3 标准条目中 UML 包的依赖关系	18
图 4 几何包:类的内容和内部依赖关系	19
图 5 具有特化关系的几何基本类	20
图 6 GM_对象(GM_Object)	21
图 7 GM_边界(GM_Boundary)	27
图 8 GM_单形(GM_Primitive)	29
图 9 GM_点(GM_Point)	31
图 10 GM_可定向单形(GM_OrientablePrimitive)	32
图 11 GM_曲线(GM_Curve)	34
图 12 GM_曲面(GM_Surface)	35
图 13 GM_体(GM_Solid)	36
图 14 直接位置(DirectPosition)	37
图 15 曲线段类(Curve segment class)	39
图 16 线性、圆弧和大地曲线插值(Linear, arc and geodesic interpolation)	44
图 17 弧(Arcs)	46
图 18 圆锥曲线与配置(Conics and placements)	50
图 19 样条与特殊曲线(Spline and specialty curves)	52
图 20 曲面片(Surface patches)	58
图 21 多边形的曲面(Polygonal surface)	61
图 22 TIN 的构造(TIN construction)	62
图 23 GM_参数曲线族曲面和它的子类(GM_ParametricCurveSurface and its subtypes)	64
图 24 GM_聚集形(GM_Aggregate)	69
图 25 GM_复形(GM_Complex)	71
图 26 GM_组合形(GM_Composite)	72
图 27 GM_组合点(GM_CompositePoint)	73
图 28 GM_组合曲线(GM_CompositeCurve)	73
图 29 GM_组合曲面(GM_CompositeSurface)	74
图 30 GM_组合体(GM_CompositeSolid)	74
图 31 拓扑包:类内容和内部依赖关系	75
图 32 拓扑类概图	75
图 33 几何与拓扑间的关系	76
图 34 TP_对象(TP_Object)	77
图 35 边界和作为关联表达的余边界操作	78
图 36 拓扑中的重要类	78
图 37 边界关系数据类型	80
图 38 TP_单形(TP_Primitive)	81
图 39 TP_有向拓扑子类(TP_DirectedTopo subclasses)	83
图 40 TP_有向拓扑(TP_DirectedTopo)	83
图 41 TP_结点(TP_Node)	85
图 42 TP_边(TP_Edge)	86
图 43 TP_拓扑面(TP_Face)	87
图 44 TP_拓扑体(TP_Solid)	88

图 45 TP_表达式(TP_Expression)	89
图 46 TP_复形(TP_Complex)	92
图 C.1 由 GM_Primitive 组成的数据集	114
图 C.2 对示例数据的简单图形化表达	116
图 C.3 具有所标坐标系统的 3 维几何体	116
图 C.4 曲面示例	117
图 D.1 用于简单拓扑的包与类	119
图 D.2 简单拓扑中的拓扑与几何类	120
图 D.3 简单拓扑中的要素成分	121
图 D.4 基于要素拓扑的专题	123
图 D.5 “最小拓扑”拓扑结构的集合示例	123
图 D.6 最小拓扑	124
图 D.7 典型的最小拓扑记录示意图	126
表 1 几何单形的一致性类	2
表 2 几何复形的一致性类	2
表 3 拓扑复形的一致性类	2
表 4 具有几何实现的拓扑复形的一致性类	3
表 5 派生拓扑算子	3
表 6 包与类	17
表 7 各种类型的参数曲线族曲面	63
表 8 布尔交范型(pattern)矩阵的意义	93
表 9 Egenhofer 9 交范型矩阵的含义	94
表 10 全拓扑交范型矩形的含义	95
表 D.1 原始最小拓扑指针与现行模型之间的对应关系	127
表 NA.1 不同维数的几何对象与拓扑对象构成对照表	128



前 言

本标准等同采用 ISO 19107:2003《地理信息 空间模式》(英文版),并作了下列编辑性修改:

- a) 本标准的编写方法执行 GB/T 1.1—2000《标准化工作导则 第1部分:标准的结构和编写规则》、GB/T 20000.2—2001《标准化工作指南 第2部分:采用国际标准的规则》的要求;
- b) 将“本国际标准”一词改为“本标准”;
- c) 删除了国际标准的前言;
- d) ISO/IEC 11404:1996 已被我国等同采用为国家标准,在本标准中用国家标准的代号(GB/T 18221—2000)和名称取代相应的国际标准的代号和名称;
- e) 由于 ISO 19109、ISO 19111 国际标准已经出版,在本标准规范性引用文件中删去了原国际标准中标识即将出版的角标;
- f) 在本标准规范性引用文件中增加了 ISO 19101:2002《地理信息 参考模型》;
- g) 在参考文献中增加了 ISO 19111:2007《Geographic information—Spatial referencing by coordinates》,对原参考文献顺序按作者(或机构)首字母顺序进行了重排,并对文中引用编号进行了相应调整;
- h) 对原文本中章条编号错误(6.5.4 下直接就是 6.5.4.3)进行了更正;
- i) 对原文本中 7.4.2.7 下的代码语句去掉了多余的“[”符号,
即将 `TP_Primitive::complex [[1..n]:Reference<TP_Complex>`
改为 `TP_Primitive::complex [1..n]:Reference<TP_Complex>`;
- j) 对原文中附录 B 的 B.8 中 `homomorphism(4.52)` 注后面多余的一段话去掉,以与 4.52 一致;
- k) 增加了资料性附录 NA。

本标准的附录 A 为规范性附录,附录 B、附录 C、附录 D、附录 NA 为资料性附录。

本标准由全国地理信息标准化委员会(SAC/TC 230)提出并归口。

本标准起草单位:中国测绘科学研究院、武汉大学。

本标准主要起草人:李青元、王涛、石丽红、刘纪平、邓跃进、杜道生、龚健雅。

引 言

本标准提供的概念模式用于描述与处理地理要素的空间特征,空间特征概念模式的标准化将是其他地理信息标准的基石。

要素是对现实世界现象的抽象,如果要素与地球上的位置相关就称其为地理要素。矢量数据是由几何单形与拓扑单形组成,人们单独或组合使用几何单形与拓扑单形来构建对象,以表达地理要素的空间特征。栅格数据是将覆盖区域划分为规则格网单元,并为每个单元赋一个属性值所形成的数据。本标准仅涉及矢量数据。

在本标准定义的模型中,空间特征通过一个或多个空间属性来描述,而这些空间属性的值通过几何对象(GM_Object)或拓扑对象(TP_Object)来定义。几何以坐标和数学函数为基础,量化地描述要素的维数、位置、尺寸、形状、方向等空间特征。描述对象的几何的数学函数取决于定义空间位置的坐标参照系类型。当地理信息从一个大地参照系或坐标系转换到另一个大地参照系或坐标系时,仅几何特征发生变化。

拓扑用于处理当空间发生弹性、连续变形时(例如当地理数据从一个坐标系转换到另一个坐标系)几何图形上保持不变的那些特征。在地理信息范畴内,拓扑通常用来描述 n 维图的连通性——图(graph)在连续变换中不变的一种性质。计算拓扑提供关于几何单形的连通信息,这些信息可以从基础的几何特征中推导出来。

空间算子是使用、查询、创建、修改、删除空间对象的函数和程序。本标准定义这些算子的分类方法,以便创建一个对这些算子进行定义与实现的标准。其目标是:

- a) 无歧义地定义空间算子,以保证在已知精度和分辨率限定下的不同实现均能生成可比较的结果;
- b) 基于这些定义来制定一套标准操作,以形成兼容系统的基础,并作为实现的测试平台和兼容性确认的基准;
- c) 定义一套允许将这些算子组合起来的算子代数,以用于可预见的地理数据查询与处理。

空间特征概念模式的标准化将提高在不同应用中共享地理信息的能力。这些概念模式将被地理信息的系统与软件开发者以及地理信息用户所使用,以实现空间数据结构的一致理解。

地理信息 空间模式

1 范围

本标准定义了用于描述地理要素的空间特征的概念模式和基于这些模式的一组空间操作。它处理 3 维以内的矢量几何与拓扑,为位于最多 3 个轴的坐标空间中的不超过 3 个拓扑维的空间对象(几何的与拓扑的)的地理信息的存取、查询、管理、处理和数据交换定义标准的空间操作。

2 一致性

2.1 概述

本标准的第 6 章、第 7 章以统一建模语言(UML)来表达用于描述地理要素空间特征的概念模式。这些模式定义可用于应用模式、专用标准和实现规范的概念类。本标准只关注外部可见的接口,而对内部实现没有限制,并且这些接口不同于下列真实环境下需要满足的接口规范:

- 使用如 COM 或 CORBA 技术的软件服务接口;
- 使用如 SQL 技术的数据库接口技术;
- 使用 ISO 19118 定义编码的数据交换技术。

几乎没有哪个应用会需要本标准概念模式所描述的全部内容,因此本章定义了相容类的一个集合,这些相容类可支持从定义数据结构的最小需求到完整对象的实现。其灵活性是由一套可用多种方式实现的 UML 类型来控制的。当定义完整对象功能的应用时,必须实现由所选择的相容类的类型所定义的所有操作,因为这些操作对 UML 设计目标的实现是共同的。对于某些或全部操作都选择基于外部“自由函数”或放弃完全实现的那些应用,不必支持所有的操作,但是应该支持这样一种数据类型,它能通过定义成员变量来记录所选 UML 类型的每一种状态。“语意相同”的通用名称允许不同的技术实现。本标准的 UML 模型定义抽象的类型(type);应用模式定义概念上的类(class);各种软件系统定义类或数据结构的实现;从编码标准(ISO 19118)中来的 XML 定义实体的标签。所有这些定义引用相同的信息内容。在数字实体的实现中,虽然在深度级别上存在明显的技术差别,但在使用相同名称表达相同信息内容方面并没有困难。这就“允许”定义在 UML 模型中的类型直接用在应用模式中。

有 39 个相容的可选项(见表 1~表 4)用于实例化几何或拓扑对象的应用模式。下面按数据复杂度、维数和功能复杂度三个准则对它们进行划分。

前两个准则确定定义在本模式中的类型,而本模式将按照由所给的一致性选项所确定的应用模式来实现。为了定义要实现的对象类型的维数,需要应用模式指定所要实例化的曲线和曲面所使用的插值类型。对于那些包含 1 维对象的应用模式,曲线实例化需要包含一个“线性”插值方法,即包含一个以线串逼近任意曲线的机制,以允许将数据转化为所需要的更简单的模式。对于那些包含 2 维对象的应用模式,曲面实例化需要包含一个“平面插值”方法,即以平面型的面片组合来逼近任何曲面的方法,以允许将数据转化为所需要的更简化的模式。附加的曲线与曲面插值机制是可选的,但如果要实现,则应遵循包含在本标准中的定义。第三个准则(功能复杂度)确定要实例化的类型的成员元素(属性、相关规则和操作)。这些模式的最大限定就是仅定义数据类型,并且可能被用于数据传输或服务提供者传递操作参数。

第一个准则是数据复杂度级别,分为四级:

- 几何单形;

- 几何复形；
- 拓扑复形；
- 具有几何实现的拓扑复形。

注：对于通常被称之为“非结构 (spaghetti)”数据的模式仅使用几何单形的无结构集合。如果需要对于每一个几何成分单独定义，模式中就需要引入几何复形。同一几何复形内的不同单形仅在边界处相连。如果模式需要明确的拓扑信息，则几何复形需要拓展到包含拓扑复形的结构。包含在复形中的对象的类型受该复形的维数控制。通常称为“边-结点”的拓扑结构是 1 维拓扑复形。在 2 维制图环境中的“全拓扑 (full topology)”是一个 2 维拓扑复形，它是由 2 维坐标系的几何对象实现的。

第二个准则是维数，对于简单的几何对象有 4 级：

- 0 维对象；
- 0、1 维对象；
- 0、1、2 维对象；
- 0、1、2、3 维对象。

由于 0 维复形没有提供比 0 维几何单形更多的有用信息，故一致性类仅定义 1、2、3 维复形。

第三个准则是功能的复杂度级别，分为三级。

- 只有数据类型；
- 还有简单操作；
- 具有完整的操作。

本标准的第 8 章定义了三组布尔操作用来导出在几何对象与拓扑对象之间的拓扑关系。本标准为使用这些操作的应用模式定义了四个一致性的类 (见表 5)。

2.2 一致性类

要遵从本标准，一个实现应该满足附录 A 中抽象测试套件对于指定的一致性类的要求，表 1～表 5 列出了用于每一个一致性类的抽象测试套件的章节。

表 1 几何单形的一致性类

功能复杂度维数	数据类型	数据类型+简单操作	数据类型+完整操作
0	A.1.1.1	A.1.2.1	A.1.3.1
1	A.1.1.2	A.1.2.2	A.1.3.2
2	A.1.1.3	A.1.2.3	A.1.3.3
3	A.1.1.4	A.1.2.4	A.1.3.4

表 2 几何复形的一致性类

功能复杂度维数	数据类型	数据类型+简单操作	数据类型+完整操作
1	A.2.1.1	A.2.2.1	A.2.3.1
2	A.2.1.2	A.2.2.2	A.2.3.2
3	A.2.1.3	A.2.2.3	A.2.3.3

表 3 拓扑复形的一致性类

功能复杂度维数	数据类型	数据类型+简单操作	数据类型+完整操作
1	A.3.1.1	A.3.2.1	A.3.3.1
2	A.3.1.2	A.3.2.2	A.3.3.2
3	A.3.1.3	A.3.2.3	A.3.3.3

表 4 具有几何实现的拓扑复形的一致性类

功能复杂度维数	数据类型	数据类型+简单操作	数据类型+完整操作
1	A. 4.1.1	A. 4.2.1	A. 4.3.1
2	A. 4.1.2	A. 4.2.2	A. 4.3.2
3	A. 4.1.3	A. 4.2.3	A. 4.3.3

表 5 派生拓扑算子

布尔算子	A. 5.1
Egenhofer 算子	A. 5.2
全拓扑算子	A. 5.3
所有的派生拓扑算子	A. 5.4

3 规范性引用文件

下列文件中的条款通过本标准的引用而成为本标准的条款。凡是注日期的引用文件,其随后所有的修改单(不包括勘误的内容)或修订版均不适用于本标准,然而,鼓励根据本标准达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件,其最新版本适用于本标准。

GB/T 18221—2000 信息技术 程序设计语言、环境与系统软件接口 独立于语言的数据类型
(idt ISO/IEC 11404:1996)

ISO 19101:2002 地理信息 参考模型

ISO 19109:2005 地理信息 应用模式规则

ISO 19111:2007 地理信息 基于坐标的空间参照

4 术语和定义

本标准使用下列术语与定义。本章中的术语按英文字母顺序列出,在附录 B 中按它们的概念关系组织。

4.1

应用 application

支持用户需求的数据操作和处理。

[ISO 19101:2002]

4.2

应用模式 application schema

一个或多个应用所需数据的概念模式。

[ISO 19101:2002]

4.3

包 bag

可以重复的相关项(对象或值)的有限、无序集合。

注:逻辑上,包是(项,数目)对的集合。

4.4

边界 boundary

表示一个实体界限的集合。

注:边界通常用在几何结构中,这里的集合是点的集合或代表这些点的对象的集合。在其他场合,边界这个术语通常隐晦地描述在实体与论域其余部分之间的一个跃迁。

4.5

缓冲区 buffer

包含距一个指定几何对象的距离都小于或等于一个给定值的所有直接位置的几何对象。

4.6

循环序列 circular sequence

没有逻辑起始,因而等同于任何自身循环移位的序列,因此可认为序列的最后一项位于该序列的第一项之前。

4.7

类 class

具有相同的属性(域)、操作、方法、关系与语义的对象集的描述。

注:类可以用接口的集合来定义它所提供给它的外界(环境)的操作集。该术语过去最先在面向对象的编程中使用,后来被 UML(统一建模语言)所采纳,并具有相同的含义。

[ISO/TS 19103:2005]

4.8

闭包 closure

拓扑对象或几何对象的内部与边界的并集。

4.9

余边界 coboundary

与该拓扑对象相关联(该拓扑对象在其边界上),具有更高维数的拓扑单形的集合。

注:如果一个结点在边的边界上,则该边就在该结点的余边界上。任何定向参数,只要与这些关系中的一个相关联也就与其他的关系相关联。所以,如果一个结点是一条边的终结点(定义为正向向边的终点),那么该结点的正定向(定义为正向向结点)将有边在它的余边界上,见图 35。

4.10

组合曲线 composite curve

曲线的序列。在该序列中,每条曲线(第一条除外)均从前一条曲线的终点开始。

注:组合曲线作为直接位置的集合,具有曲线的所有特性。

4.11

组合体 composite solid

沿共有边界曲面相邻且相连的体的集合。

注:组合体作为直接位置的集合,具有体的所有特性。

4.12

组合曲面 composite surface

沿共有边界曲线相邻且相连的曲面的集合。

注:组合曲面作为直接位置的集合,具有曲面的所有特性。

4.13

计算几何 computational geometry

几何表达式的处理与计算,用于几何操作的实现。

示例:计算几何操作包括几何对象间的几何包含或相交测试、凸包或缓冲区计算和确定最短路径。

4.14

计算拓扑 computational topology

在计算几何中通常用来辅助、增强或定义对拓扑对象进行操作的拓扑概念、拓扑结构和拓扑代数。

4.15

连通 connected

几何对象的一种特性,指该对象内的任何两个直接位置能置于完全在该对象内的一条曲线上。

注:当且仅当一个拓扑对象的所有几何实现是连通的,该拓扑对象才是连通的。由于它源自拓扑定理,所以未写在定义内。

4.16

相连结点 connected node

起始于或终止于一条或多条边的结点。

4.17

凸包 convex hull

包含所给几何对象的最小凸集。

注:“最小”是集合理论上的最小,并不是指尺度上。这个定义也可以重写为“包含所给几何元素的所有凸集的并集”。

4.18

凸集 convex set

一种几何集,在该几何集中连接任意两个直接位置的直线段上的任意直接位置都包含在该几何集中。

注:凸集间是“简单连接”,这意味着它们没有内部孔洞,并且通常可以认为在拓扑上是与适当维数的欧几里德球同构的。所以,球的表面可以被认为几何凸集。

4.19

坐标 coordinate

表示 n 维空间中点位置的 n 个序列数之一。

注:在一个坐标参照系中,坐标数值取决于所选单位。

[ISO 19111:2007]

4.20

坐标维数 coordinate dimension

在一个坐标系中描述一个位置所需的量测或坐标轴个数。

4.21

坐标参照系 coordinate reference system

通过基准与对象相联系的坐标系。

注:对大地基准和垂向基准而言,对象是地球。

[ISO 19111:2007]

4.22

坐标系 coordinate system

给点赋予坐标的数学规则集。

[ISO 19111:2007]

4.23

曲线 curve

1维几何单形,表达一条线的连续映像。

注:曲线的边界是曲线两个端点的集合。如果曲线是一个圆,则这两个端点相同,并且该曲线(如果拓扑上是闭合的)被认为没有边界。这第一个点被称作始点,最后一个点被称为终点。曲线的连通性由“一条线的连续映像”条款所保证,拓扑定理认为一个连通集合的连续映像仍然是连通的。

4.24

曲线段 curve segment

具有相似的插值与定义方法,用来表达一条曲线的连续组分的1维几何对象。

注:用一条单独的曲线段表达的几何集与一条曲线等价。

4.25

圈 cycle

〈几何学〉没有边界的空间对象。

注:圈用来描述边界组成部分(参见壳、环)。圈没有边界,因为它自身闭合,但它是有限的(即它不具有无限的外延)。例如圆环或球,没有边界,但却是有限的。

4.26

直接位置 direct position

用坐标参照系中的一组坐标描述的位置。

4.27

有向边 directed edge

表达边和它的某一定向(orientation)相关联的有向拓扑对象。

注:与边的定向一致的有向边为正向(+),反之,它为反定向(-)。有向边用在拓扑中以区分同一条边的右侧(-)与左侧(+),同一条边的始结点(-)与终结点(+)。在计算拓扑中使用这些概念。

4.28

有向面 directed face

表达拓扑面和它的某一定向相关联的有向拓扑对象。

注:根据有面向定向的矢量方向,组成该面的外边界的有向边的定向为正;约束一个拓扑体的有向面的定向指向离开该拓扑体。共享边界面的相邻体将具有不同的定向,这与相邻面和它们的共享边之间的关联关系相一致。有向面用在余边界关系中,以维持面与边之间的空间关联。

4.29

有向结点 directed node

表达结点和它的某一定向相关联的有向拓扑对象。

注:有向结点用在余边界中以维持边与结点间的空间关联。结点的定向是关于某条边的,终结点为“+”,始结点为“-”,这与“结果=结束一起始”的矢量概念相一致。

4.30

有向体 directed solid

表达拓扑体和它的某一定向相关联的有向拓扑对象。

注:有向拓扑体用在余边界关系中以维持拓扑面与拓扑体之间的空间关联。拓扑体的定向是关于某一拓扑面的,如果拓扑面的外法向指向外,则拓扑体的定向为“+”;如果拓扑面的外法向指向内,则拓扑体定向为“-”。这与约束一个拓扑体的曲面“向上=向外”的概念相一致。

4.31

有向拓扑对象 directed topological object

表达拓扑单形和它的某一定向逻辑关联的拓扑对象。

4.32

域 domain

定义明确的集合。

注:域用于定义域集以及属性、算子和函数的范围集。

4.33

边 edge

1维拓扑单形。

注:边的几何实现是曲线。边的边界是拓扑复形中关联到该边的一个或两个结点的集合。

4.34

边-结点图 edge-node graph

镶嵌在拓扑复形中的图,该拓扑复形由其内的所有边和相连结点构成。

注:边-结点图是镶嵌在复形中的子复形。

4.35

终结点 end node

在边的边界中对应于边的终点的结点。在使用该边的拓扑复形的任何有效几何实现中,边为曲线。

4.36

终点 end point

曲线的最后一个点。

4.37

外部 exterior

全域与闭包之差。

注:“外部”这个概念对拓扑复形和几何复形都适用。

4.38

[拓扑]面 face

2 维拓扑单形。

注:拓扑面的几何实现是曲面。拓扑面的边界是在拓扑复形内通过边界关系关联到该拓扑面的有向边的集合。这些有向边的集合可以组织成若干环。

4.39

要素 feature

现实世界现象的抽象。

注:要素可以类型或实例的形式出现。当仅表达一种含义时,应使用要素类型或要素实例。

[ISO 19101:2002]

4.40

要素属性 feature attribute

要素的特征。

示例 1:一个名为“颜色”的要素属性有一个属性值为“绿色”,其数据类型为“文本”。

示例 2:一个名为“长度”的要素属性有一个属性值为“82.4”,其数据类型为“实型”。

注 1:要素的属性包括名称、数据类型及与其相关的值域。要素实例的要素属性有一个来自于其值域的属性值。

注 2:在要素目录中,要素属性可以包括一个值域,但并不指定要素实例的属性值。

[ISO 19101:2002]

4.41

函数 function

从一个域(源或函数的定义域)中的每一个元素到另一个域(目标域、因变量域、值域)中唯一元素相关联的规则。

4.42

地理信息 geographic information

与地球上的地点直接或间接相关的现象的信息。

[ISO 19101:2002]

4.43

几何聚集形 geometric aggregate

没有内部结构的几何对象的集合。

注:不对这些元素间的空间关系作任何假设。

4.44

几何边界 geometric boundary

由约束几何对象范围的较低几何维数的几何单形的集合所表达的边界。

4.45

几何复形 geometric complex

分离的几何单形的集合,其每一个几何单形的边界都可以表达为该集合内维数更低的几何单形的并集。

注:集合中的几何单形是分离的,意味着没有一个直接位置能同时在两个或两个以上的几何单形的内部。集合在边界操作中闭合,意味着几何复形中的每一个元素,存在一个代表该元素边界的几何单形的组合(即一个几何复形)。点(几何中唯一的0维单形对象)的边界是空。这样,如果最高维的几何单形是体(3维),按照该定义,边界算子的运算最多三步结束。这就是任何对象的边界是一个圈(cycle)的情形。

4.46

几何维数 geometric dimension

在几何集内每一个直接位置都可以与其内部具有该直接位置的子集相关联,且相似(同构)于 n 维欧几里德空间 R^n 的最大数目 n 。

注:曲线的几何维数为1,因为它们实线段的连续映像。虽然曲面不能将它的整体映射到2维空间 R^2 中,但围绕每个点的位置,(在连续函数的情况下)可以找到一个类似于 R^2 单位圆的小邻域,因此曲面是2维的。在本标准中,大多数曲面片(几何曲面片 GM_SurfacePatch 的实例)通过它们所定义的插值机制映射为 R^2 中的一部分。

4.47

几何对象 geometric object

表达一个几何集的空间对象。

注:几何对象由几何单形、几何单形的组合或处理为一个单独实体的几何复形构成。几何对象可以是作为要素或要素的一个有意义部分的对象的空间表达。

4.48

几何单形 geometric primitive

表达空间中单一、连通和同质元素的几何对象。

注:几何单形是在能表达几何结构信息的层次上不可再分的对象。几何单形包括点、曲线、曲面和体。

4.49

几何实现 geometric realization

几何复形,其几何单形与一个拓扑复形中的拓扑单形一一对应,且这两个复形的边界关系一致。

注:在该实现中,拓扑单形表达相应的几何单形的内部。组合几何对象是闭合的。

4.50

几何集 geometric set

直接位置的集合。

注:在大多数情况下,这个集合是无限的。

4.51

图 graph

结点集,有些结点与边相连。

注:在地理信息系统中,一个图可以有多于一条边连在两个结点上,也可以一条边的首尾结点相同。

4.52

同态 homomorphism

两个域(例如两个复形)之间存在一种函数关系,能从一个域保持结构地映射到另一个域。

注:同态区别于同构在于它没有反函数的要求。在同构中,两个同态互为反函数。连续函数是拓扑同态,因为它们保留了“拓扑特征”。拓扑复形到其几何实现的映射保持了边界的概念,因此是同态。

4.53

实例 instance

类的实现对象。

4.54

内部 interior

几何对象除边界外的所有直接位置的集合。

注：拓扑对象的内部是与其相对应的几何实现的内部的同态映像。因为它遵从拓扑定理，所以不再定义。

4.55

孤立结点 isolated node

不与任何边相连的结点。

4.56

同构 isomorphism

在两个域(例如两个复形)之间的一种关系，它们具有从一个域到另一个域——对应地保持结构的函数和相对应的反函数，并且按两种顺序组合这两个函数得到相应的恒等函数。

注：如果几何复形与拓扑复形的元素个数、维数、边界都——对应，则该几何复形与该拓扑复形同构。

4.57

邻域 neighbourhood

几何集，在它的内部包含有一个指定的直接位置和距该直接位置的距离都在一个给定值之内的所有直接位置。

4.58

结点 node

0 维拓扑单形。

注：结点的边界是空集。

4.59

对象 object

〈UML〉具有明确定义的边界与标识，封装了状态与行为的实体。

注：该术语最先是用在面向对象编程的一般理论中，后来以相同的含义被 UML 所采纳。对象是类的实例，属性与关系表达状态。操作、方法和状态机制表达行为。

[ISO/TS 19103:2005]

4.60

平面拓扑复形 planar topological complex

具有一个可以嵌入到欧几里德 2 维空间中的几何实现的拓扑复形。

4.61

点 point

0 维几何单形，表示一个位置。

注：点的边界是空集。

4.62

记录 record

有限的、有名称的相关项(对象或值)的集合。

注：逻辑上，记录是〈名称，项〉对的集合。

4.63

环 ring

为圈的简单曲线。

注：环用来描述在 2 维和 3 维坐标系中的曲面的边界成分。

4.64

序列 sequence

可重复的相关项(对象或值)的有限、有序集合。

注：逻辑上，序列是(项，偏移量)对的集合。它原是 LISP 语言中一种用括号括起来，有次序限定，元素间用逗号隔开的结构，现用在本标准中。

4.65

集[合] set

不重复的相关项(对象或值)的无序汇集。

4.66

壳 shell

为圆的简单曲面。

注：壳用来描述 3 维坐标系中体的边界成分。

4.67

简单 simple

几何对象的一种特性，其内部是等向的(所有点都具有同构的邻域)，因而它处处都局部同构于适当维数欧几里德坐标空间的开子集。

注：这暗示几何对象内部的直接位置不会出现任何形式的自相交。

4.68

体 solid

3 维几何单形，表达欧几里德 3 维空间中一个区域的连续映像。

注：体可局部地实现为直接位置的三个参数的集合。体的边界是构成体的界限的有向闭合曲面的集合。

4.69

空间对象 spatial object

用于表达要素空间特征的对象。

4.70

空间算子 spatial operator

在它的域或范围内至少有一个空间参数的函数或程序。

注：对空间对象的任何 UML 操作都可以归为空间算子，例如本标准中第 8 章的查询算子。

4.71

始结点 start node

在边的边界中对应于边的始点的结点，在使用该边的拓扑复形的任何有效几何实现中，边为曲线。

4.72

始点 start point

起点

曲线的第一个点。

4.73

强可替换性 strong substitutability

继承或实现另一个类、类型或接口的派生类的任何实例在任何相关环境中用于替代其祖先实例的能力。

注：弱可替换类型对于要进行替换的环境有各种限制。

4.74

子复形 subcomplex

所有元素都在一个更大复形里的复形。

注：因为几何复形与拓扑复形的定义只要求它们在边界操作中闭合，所以一个特定维数以及更低维数的任何单形的集合总是原来更大复形的子复形。这样，任何完全的平面拓扑复形包含一个作为子复形的边-结点图。

4.75

曲面 surface

2 维几何单形,局部表达一个平面区域的连续映像。

注:曲面的边界是定义曲面界限的有向、闭合曲线的集合。与一个球面或 n 圆环面(具有 n 个贯通孔洞的拓扑球面)同构的曲面没有边界,这样的曲面被称之为圆。

4.76

曲面片 surface patch

2 维且连通的几何对象,用于表达使用相同插值与定义方法的曲面的连续部分。

4.77

拓扑边界 topological boundary

由约束拓扑对象范围的更低拓扑维的有向拓扑单形的集合所表达的边界。

注:拓扑复形的边界对应于该拓扑复形的几何实现的边界。

4.78

拓扑复形 topological complex

在边界操作中闭合的拓扑单形的集合。

注:在边界操作中闭合,意味着如果一个拓扑单形在该拓扑复形中,则它的边界对象也在该拓扑复形中。

4.79

拓扑维 topological dimension

在几何对象内区分邻近但不同的直接位置所需自由变量的最小个数。

注:上面提到的自由变量通常可以认为是一个局部坐标系。在 3 维坐标空间中,平面可以写为 $P(u, v) = A + uX + vY$, 这里 u 和 v 是实数, A 是平面上的任意点, X 和 Y 是该平面的两个切矢量。因为平面上的地点可以用 u, v 这两个变量来区分(没有例外),所以平面是 2 维的,并且 (u, v) 是该平面上对这些点的一个坐标系。在一般曲面上,并不总能这样。但是,如果我们取该曲面的切平面,并且将曲面上的点投影到该切平面上,我们通常可以在该切平面上得到曲面在该切点的小邻域的局部同构。对于这个潜在曲面,该“局部坐标系”足以建立该曲面为一个 2 维拓扑对象。

由于本标准仅涉及空间坐标,任何 3 维对象可以依靠坐标建立它的拓扑维。在一个四维模型(空间-时间)中,切空间(tangent spaces)在将对象在 3 维上建立拓扑维的过程中也起着重要的作用。

4.80

拓扑表达式 topological expression

有向拓扑单形的集合,这些有向拓扑单形可以像多变量多项式一样进行操作。

注:拓扑表达式用在很多计算拓扑的计算中。

4.81

拓扑对象 topological object

表达在连续变换下不变的空间特性的空间对象。

注:拓扑对象可以是拓扑单形、拓扑单形的一个集合或拓扑复形。

4.82

拓扑单形 topological primitive

单一的、不可再分的拓扑对象。

注:拓扑单形对应于几何实现中同维几何单形的内部。

4.83

拓扑体 topological solid

3 维拓扑单形。

注:拓扑体的边界由有向拓扑面的集合组成。

4.84

全端面 universal face

在 2 维复形中无界的拓扑面。

注：全端面通常不是任何要素的组成部分，它用来表达数据集的无界部分，通常认为它的内边界（它没有外边界）是由数据集所表达的地图的外边界。本标准并不将全端面当特殊情况对待，但应用模式可发现将其特殊处理也很方便。

4.85

全体 universal solid

在 3 维复形中无界的拓扑体。

注：3 维中全体与 2 维中的全端面相似，通常也不是任何要素的组成部分。

4.86

矢量几何 vector geometry

用构造的几何单形表达的几何。

5 符号、注记与缩略语

5.1 表达与注记

5.1.1 统一建模语言(UML)概念

本标准中，用统一建模语言(UML)表达概念模式。

UML 类是对共享相同属性、操作、方法、关系和语义的一组对象的集合的描述。参数化类是对若干类的集合的一个单独描述，这些类具有共同的操作、方法和关系，这些类的区别在于具有不同的参数，而这些参数控制其属性的精细结构和操作的行为。例如，整型参数用来固定在参数化类的实现（称为实例化的类）中属性数组的大小。

不应将本标准中以某种方式对几何或拓扑元素的模型化视为对实现的限制。属性既可以作为数据直接实例化，也可以作为“获取”与“修改”这样的“操作对”来实现对值的获取与设置。在本标准中，大多数的图是“结构示意图(context diagram)”，这种图主要关注某个类的属性、操作与重要关系。而其他图是对这些类间关系的一个概览。UML 并不要求将所有关系都显示在图上，一些琐碎的细节被忽略以保持其简洁性。例如，GM_Object 与集合(GM_Object)具有明显的关系，但是在 GM_Object 结构示意图上并没有明确地显示这种关系。

5.1.2 属性、操作与关联

表现在 UML 图上的属性与操作遵从 UML 符号指南（见参考文献[20]）。UML 的属性符号有下面几种形式：

属性声明：==(<构造型>)可见性 名称 多重性：类型=初值{特性，...}

多重性：==[基数范围，...]

基数范围：==起始值{.. 结束值}

注：基数(Cardinality)指集合中的元素个数。多重性(Multiplicity)指一个集合可以允许的基数范围规定。多重性规定可以赋给关联中的角色、组合中的组成部分、副本以及其他目。本质上，多重性是一个非负整数的子集。见 ISO 19103。

UML 对于操作的符号有下面几种形式：

操作：==(<构造型>)可见性 名称(参数列表)：[返回值]，...{{特性{=值}，...}}，...

参数列表：==[方向] 参数名：类型 [=缺省值]

上面语法的解释如下：

- 构造型(stereotype)：对所定义的属性或操作进行标签，见 5.1.3。
- 可见性(visibility)：从对象的外部来看，对象的属性和方法的可见性包括公有(+)、私有(-)、

保护(#)。如果可见性包括“/”,则属性是从该模型的其他部分派生而来的。

- c) 名称(name):属性或操作的名称。
- d) 多重性(multiplicity):该属性可以取的值的个数,除非有其他定义,都假定数据是按一个集合组织的。这是对 GB/T 18221 的一个扩展,除非使用 UML 的符号“[...]”,否则就与 GB/T 18221 标准中的 size 机理一致。为了保持概念的一致性,当它用在与 GB/T 18221 标准的图表中的“size”相关联的地方时,本标准使用一个单独的(来自 UML 的)多重性语义。
- e) 起始值(begin-value):有效的多项可选之中的任意一个整数;如果没有结束值相随,则只有起始值作为多重性的可能值。
- f) 结束值(end-value):大于前面的起始值的任何整数,或用“n”表达无穷或无界的坐标范围。“a..b”是指大于或等于 a 并且小于或等于 b 的任何整数(例如 j,则 $a \leq j \leq b$);[a..a] 是指与 [a] 一样。
- g) 参数列表(parameter list):用逗号分隔的一个参数列表声明。
- h) 参数名(parameter-name):操作参数的名称,通常指示该参数在操作中被定义的角色。注意,除非操作包含了一个参数列表,否则操作和属性的语义结构是相同的。
- i) 方向(direction):流向的可选指示符,该参数有“in”(该值在操作之前设定,并影响到操作)、“out”(该值是在操作之中赋设定,并且在完成操作之后,调用者可以得到其值)或“inout”(该值在操作之前设定,并影响到操作,在操作中被重新设置为一个在操作完成后可存取的值)。对于任何参数,缺省的方向是“in”。
- j) 类型(type):对象类型或前述参数或属性的值的类型。
- k) 缺省值(default-value):当调用者未设定时,“in”或“inout”参数所取的值;或当构造函数中未设定时,对象的属性所取的值。
- l) 返回类型(return-type):操作的返回值或对象的类型,本质上是操作的类型。
- m) 特性(property):属性或操作的附加信息,例如非空或唯一。可以作为附加名随值一同构造,例如“{size=[0..n]}”。(见 GB/T 18221 对特性在图表类型的有关使用解释)
- n) ...:前述内容可以以任意次数重复。
- o) 初值(initial-value):属性的缺省值,当构造新对象时使用,除非被构造参数列表专门重载。

在本标准中,从对象约束语言(OCL)中来的符号在使用中稍微有一点修改。“ocl”前缀在很多操作中都被去掉了,因为它是不必要的,而且会引起混淆,尤其是当这些操作符作为基本类型的一部分已经出现在 ISO/TS 19103 中而没有使用前缀时。“::”是一个分辨操作符,分辨它所跟随的名称空间。在 OCL 中,大多数情况下,名称空间定义该操作的类。但是它也可以包含定义该类的包(package)的名称。在本文档中,所有的名称空间是类的标识符,并且只能是下面两种形式中的一种:

类标识:==类名|包名::类标识

类型:==类标识

除非存在潜在的混淆,或需要强调,一般不用包括包的名称。在本标准中,所有的类名都包含一个双字符的包名前缀,后跟一个下划线“_”,并且在该模型中是唯一的。这避免了在类名和类型名中使用包名鉴别。本标准中的专用标准也鼓励尽可能地保持这个约定。对于属性、角色名和操作的文字描述如下:

属性名{多重性}:属性类型

{关联名::}角色名{多重性}:属性类型{类型 1::}

操作名(名称 2:类型 2,名称 3:类型 3,...):返回类型

对于角色,如果多重性不是[1],则假定角色值是以集合的方式组织的。如果角色值需要以某种其他方式组织,属性类型就应当使用参数化类的适当组合,其多重性 size 的给出与 GB/T 18221 一样:

boundary: CircularSequence<GM_OrientableCurve> {size=[1..n]}

o

面向对象的操作符(例如在 C++ 中使用的那样)在操作之前设定第一个参数,声明方法如下:

返回类型 类型 1::操作(类型 2,类型 3...)

这些方法被限定在名称空间中,这意味着只对“类型 1”名称空间可用。另外,在赋值过程中,暗示参数类型“类型 1”是已知的。在 OCL 中,该对象被标识为“self”。在 C++ 中,该对象被标识为“this”。在非对象语言或对象语言中的自由函数,对于一个操作的函数标识不以任何方式区分第一个参数,并且书写为:

操作(名称 1:类型 1,名称 2:类型 2,名称 3:类型 3...):返回类型,

这些符号是等价的(除非强调)并且两者都可以用在本标准的专用标准中。

这些操作定义被称作“操作署名(operation signatures)”或“协议(protocols)”。这将操作与调用机制区分分开。在 UML 中,形式化的符号定义协议,与协议相关联的操作只是非正式地定义在相关的文档中,文档中可以包括 OCL 约束。

根据属性可以被认为是一种操作(更改与存取对)的观点,该术语可以被扩展到包括属性“署名”。署名的定义包括操作名、参数名和类型,以及返回值。方法或属性可以被重载,这通过提供一个与原名相同但有些类型不同(通常是被原始类型的子类型替换)的新方法来实现。署名的重新使用被称为多态性。从类的遗传而来的多态性被称为“结构性”的。从语义类似而产生的多态性被称为“自然的”或“普通的”。例如,在几何与拓扑类中,对于“边界”的常见协议是自然多态性,因为它是源自于基于拓扑定义的操作约束。这不是一个结构性多态,因为两个包没有共享一个共同的超类祖先。假定这个类继承的层次是基于这些对象的语义,则结构性多态就是自然的。不依赖于语义类似的多态是“特定”多态。例如,“+”的使用,用在数值时表示加法,用在字符串类时表示连接操作,这就是一个“特定”多态。特定多态是语义混乱的,本标准中不使用,并且也应当尽量避免在本标准的专用标准中使用。

大多数的操作是以函数形式定义的,也就是说,它的所有参数是以“只读”(方向=“in”)方式传递的,并且只有通过指派的操作符的返回类型来实现对对象的修改或创建。在描述接口时,“this”指示对象接口正在被调用的这个实体。在 OCL 中,该对象被叫做“self”。如果一个对象是作为一个参数传递到另一个对象的方法中,则这个对象被称作“被传递的”对象。

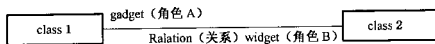


图 1 UML 关联示例

在本模型中,每个关联都给出一个关联名,并且参加到该关联中的每一个类都被赋予一个角色名。在这种情况下,“class 1”与“class 2”之间有一个“关系(Relation)”关联,如图 1 所示,这两个类在实现中通常包含对其他类的引用作为“属性”,而这个被包含的类被称为该关系的角色,例如:

Class1::gadget;Set(Class2) //这是多对多的强连接

Class2::widget;Set(Class1)

或

Class1::gadget[1..n];Class2 //这是多对多的强连接

Class2::widget[1..n];Class1

注:对于 class2,角色名是被作为变量用在 class1 中,而 class1 又指向 class2。这种组合的类型取决于该角色的分类标准的类型。

Class1::gadget;Sequence(Class2) //这是多对多、有序的强连接,

Class2::widget;Set(Class1)

大多数的角色或者与集合 Set(无序关系)对应,或者与序列 Sequence(有序关系)对应。当角色的次序没有一个自然的起始位置时,就是循环序列。这种符号将出现在下面的文本中,但这并不意味着暗指关联的一个特别实现。对于弱连接类型,这种情况下,目标类并不依赖于这种关联的存在,参数化的类

引用 Reference <> 被恰当地用在这种场合。例如：

```
Class1::gadget [1..n];Reference(Class2)    //弱连接
Class2::widget [1..n];Class1              //强连接
```

5.1.3 构造型

在 UML 模型中,大多数的实体可以被描述为“构造型”,它包含了相近的对象名和包含在“<<”和“>>”中的内容。构造型允许模型扩展 UML 以包括对模型的元素的描述。在本标准中使用下列的构造型。

- a) <<接口>>(<<Interface>>):这种类不可直接实例化,但可用在操作署名的抽象组合中。<<接口>>类的目的是对于操作的结构型多态定义一个参考类。在各种程序语言中,这种类被称之为“虚类”或“抽象类”。在 UML 标准中,<<接口>>类不得有属性或者来自接口的可见关联。接口可以参与一个关联,而提供该关联的接口看不见这个关联,即,一个类(而不是一个接口)可以有对于一个接口的关联,该(接口)关联对于这个类是可见的,但对于该接口则是不可见的。就像用在本标准中那样,在应用模式中,来自本标准的接口类与具有相同名称的非接口类逻辑上相互矛盾,不能并存在一个应用模式中。
- b) <<类型>>(<<Type>>):这种类不可直接实例化,但可用在操作、属性和关系署名的抽象组合中。<<类型>>类的目的是对于包含属性和关联的结构性多态定义一个引用类,该引用类包含属性和关联。实际属性和关联的内部组织依赖于实现。因为这些元素的组织是未知的,UML 不允许<<type>>类有任何方法(操作的实现)。就像用在本标准中那样,在应用模式中,只要 type 不是抽象的(在 UML 模型中用斜体字表示),就可以创建具有与从本标准中来的 type 名称相同的实现类。
- c) <<抽象>>(<<Abstract>>):(在 UML 中通过斜体字的类名表示)这种 class 除了它被允许有方法外,类似于<<type>>。这种 class 用来定义包括这些附加元素的结构性多态的根类。就像用在本标准中那样,在应用模式中,来自本标准的抽象 class 与具有相同名称的具体 class 逻辑上相互矛盾,不能并存在一个应用模式中。
- d) <<数据类型>>(<<Data Type>>):这种类可以直接实例化,并且它的最初目的是封装数据,这与分类学和行为描述相反。<<Data Type>>没有它自己的标识,而必须强制约束到某种类型的容器中,作为另一个 class 的属性,或者作为一个强聚集的目标 class。<<Data Type>>类型不能作为弱聚集的目标,它们也不能被用在引用<>的参数化类中。
- e) <<联合>>(<<Union>>):由几个可选项(作为成员属性列出)的一个且仅一个值组成的一种类型。它很类似于很多程序语言中的有区别的联合。在一些使用指针的语言中,它要求一个“void”指针,使得它可以根据识别器的属性决定转换为适当的类型。
- f) <<叶>>(<<Leaf>>):不含有子包,只含有对象类和接口定义的可包。虽然不是一个技术要求,本标准将所有的对象定义放在 Leaf 包中,然后在更大的包中组织成非 Leaf 的包。
- g) <<代码列表>>(<<Code List>>):类似于可枚举数据类型,在这里一系列值中的任何一个都是可能的,但是在设计目的上不同于枚举,代码列表可以随时间膨胀。大多数的代码列表是以数值型值来存储的,也有一些用来存字符串代码,但这只是实现的细节,纯数值代码也是可接受的。

5.1.4 数据类型与集合类型

为了本标准的相容性,需要多种类型组合,但是并没有根据它们的接口定义这些类型。当在 UML 中没有包括这些类型时,通常通过使用对象约束语言(OCL)来暗示,见 ISO/TS 19103。

这些接口通常是有限的集合。如果用类型“T”表示新的实例化类的类型,则称“Set(T)”由对象类为“T”的所有对象的有限、无序集合构成。实现环境通常提供几种常见的组合类型,例如数组,并且我们没有打算提供这些类型的通用的接口。ISO/TS 19103 包含了对这些类型定义的一个示范接口。本标准并不限制对于一些应用环境使用一些逻辑上等价的本地化。一些基本 class 类型和参数化类型,

例如用在本标准中的这些组合类型,包括下面的内容:

- a) 超限集(T)(TransfiniteSet(T));可能为无穷集(infinite set),只对值有限制。例如,整数与实数是超限集。这实际上是数学上对集合的通常定义,但是程序语言限定该术语的含义为有限集(finite set)。
- b) 集合(T)(Set(T));有限集合,用在对象类型上。每一个对象在集合中只出现一次。通常不是一个强聚集,因为每一个对象可以是很多集合中的元素。除非另外注明,本标准将用集合来表达弱聚集类型(集合(引用(T))形式的强聚集类型的等价形式)。
- c) 包(T)(Bag(T));一种有限无序的集合,这里每一个对象都可以在集合中出现多次。在逻辑上可以认为这是一种成对的集合(对象 T,出现次数 整数)。
- d) 序列(T)(Sequence(T));对象的有序、有限集合,通常具有重复的值。在逻辑上可以认为这是一种成对的集合(对象 T,偏移量;整数),这里偏移量给出对象出现在序列中的位置。偏移量可以从 0、1 或任何数开始计数,这取决于具体实现。投影到每一个成对的对象上时形成一个 Bag。去掉重复的元素就是一个集合。
- e) 循环序列(T)(CircularSequence);自身重复使用的序列,被认为与循环迁移一样。可以认为它与序列的类是等价的,只是以一个偏移量循环出现。
- f) 引用(T)(Reference(T));对于类 T 的一个可确认对象的引用,等价于 C++ 的指针,SQL99 (以前叫 SQL3) 的 REF,或 Java 类中的变量。在本标准中对类的文字声明里,弱聚集被表述为某个类的引用。强聚集以相同方式表述为属性。
- g) 数、浮点数、整数、实数;在这个环境中,各种简单数值类型通常被实例化为程序语言中的简单数。见 ISO/TS 19103。
- h) 长度、面积、距离;各种标量,与度量的特定单位(例如米或平方千米)相联系。见 ISO/TS 19103。

注:在 UML 的代码构造函数中,常用“代码列表”表达关联。关联角色被用作引用(T)类型的成员名称,这里 T 是关联中目标类的角色名称。在可能产生混淆的情况下,关联名用作角色的名称空间,即<关联_名>::<角色_名>;引用<目标_类>。逻辑上,在使用关联的源类时已经这样做了,例如<源_类>::<角色_名>;引用<目标_类>。如果是强聚集,在逻辑上可以去掉引用,即为<源_类>::<角色_名>;目标_类。强聚集、单向关联的语义在逻辑上被认为与一个成员属性等价。在 UML 设计中,使用“源”的一种可供选择的方案是使用“关联”来代替使用“角色属性”。一旦使用代码生成器,产生逆向 UML 关联(缺少任何其他信息)可能需要工程师们回去配对角色属性。

一些数据类型是定义在 ISO/TS 19103 中的记录类型的简单实例,与 GB/T 18221 稍有不同。因为后一个标准有一个在参数列表上可能会引起混淆的说明,本标准使用了略加修改的语义(用“<.”代替“<.”)。

记录类型:==<字段名称;类型 [=缺省值],...>

记录实例:==<字段名称;字段值,...>

注意除了括号被省略以外,一个多种返回类型的语义与记录(Record)的语义相一致。当单独使用时,本标准使用上面的记录语义,但当定义返回结构为匿名类型的“似记录”(record-like)操作时,使用标准 UML 的多重返回类型语义。

定义在本标准中的几个操作中可以使用 NULL 和 EMPTY。NULL 指要求的值没有定义。本标准假定所有的 NULL 的值是等价的。当请求一个对象,而没有与所请求相匹配的对象存在时,就返回一个 NULL。EMPTY 指这种对象,它们可以被解释为一种形式或另一种形式的集合,并且指所提到的这些集合中没有元素存在。不像程序设计中具有强类型聚集(strongly typed aggregates),本标准使用数学同义语,即有一个且只有一个空集,代表这个空集的任何对象与使用任何其他集合所起的作用是等价的。不同于空(empty),这种 EMPTY 集合缺乏任何内在信息,因此在适当的相关环境中,一个 NULL 值假定等价于一个 EMPTY 集合。

5.1.5 强可替换性

本标准假定专用标准和传输模式将使用一个强可替换版本来构建。这意味着,在设计应用模式的个别地方,专用标准制定者可以使用一个类来替换定义在本模式中的一个类,只要这个类支持基类所要求的数据、操作和关联。这种替换实现的方法并不是标准的,可以有各种方式实现,这取决于所使用的实现环境的特征。尤其是转换标准取决于数据类型。转换集中的实体与它在本标准中的基类可能仅有很微弱的联系,在基类中它们可能仅是数据的表象形式。可替换性最大的用处是在与该类关联的子项检测中,这最能发挥该技术的优势。

本标准假定子类的语义需要严格遵从一种“is type of(是.....类型)”的层次体系。类的每一个实例必须是由该类的父类的语义定义的集合中的一个成员。这样,我们可以定义圆是椭圆的子类型,而不是其他的形式,虽然这有违于子类型比超类型更复杂的这样一种观点。

5.2 组织

本标准中的章节按照 UML 包的方式组织。包(Package)是相关类型与接口的一个集合,它们形成一个软件系统设计的相容成分。包通常并不形成一个完整的系统,因为它们通常还要调用系统中其他包提供的服务。当包作为一个客户,使用其他服务包来提供所需服务,我们说该客户包依赖于服务包。当一个包中类的对象存取定义在服务包中的另一个对象时,就出现这种依赖关系。因为在地理信息中,几何体很少只是纯粹的客户或服务器,这些形式在本标准中很少使用。因为所依赖的类是通过一种可以携带请求的关联来关联到它们的服务器,所以由大多数的对象类之间的依赖关系派生出对象类之间的关联关系。不同包的对象间的每一个依赖关系必须由包的依赖关系来反映。这种包的依赖关系用图 2 所示的图形符号来指示。



图 2 UML 包依赖关系示例

由于这个客户—服务器关系,包交叉的依赖关系定义了可行的应用模式准则。包含对任何根据本标准定义的包的实现的应用模式也应该包含所有这些依赖关系的实现。

表 6 总结了本标准中的包。在第 6 章、第 7 章的包是标准的。它们为应用模式提供几何与拓扑成分,应用模式形成相容系统的外部接口的基础。附加的包是从其他标准引用来的,例如从 ISO 19111 中来的对于坐标的空间参照系包以及从 ISO/TS 19103 来的基本类型包。只是把它们复制到这里,为所需要的范围提供一个完整、易读的空间模式的描述。

表 6 包与类

条目号	英文包名	中文包名	所包含的主要类
6	Geometry	几何	几何类
6.2	Geometry root	几何根	几何根类
6.3	Geometric primitive	几何单形	几何单形类
6.4	Coordinate geometry	坐标几何	坐标几何类
6.5	Geometric aggregates	几何聚集形	几何聚集形类
6.6	Geometric complex	几何复形	几何复形类与组合形类
7	Topology	拓扑	拓扑类
7.2	Topology root	拓扑根	拓扑根类
7.3	Topological primitive	拓扑单形	拓扑单形类
7.4	Topological complex	拓扑复形	拓扑复形类

图 3 显示了对于本标准的上述标准条目的 Leaf 包的相互依赖关系。

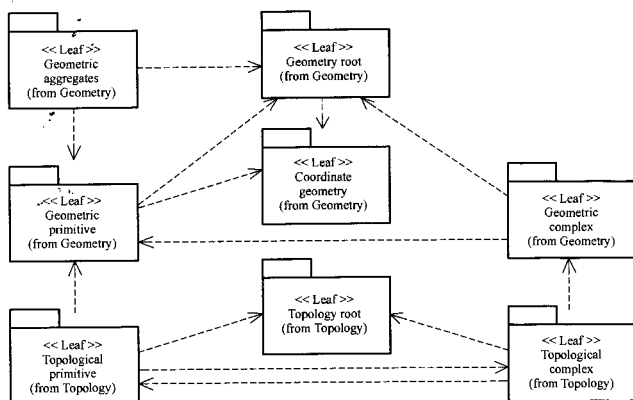


图 3 标准条目中 UML 包的依赖关系

注：在这里文中所给的例子最适合于理解本标准中的这些条目，以及经常隐含地提前引用在后面才讨论的其他条目。例如，GM_包络(GM_Envelope)的讨论包含对 GM_线串(GM_LineString)的提前引用。在大多数情况下，这不会引起混淆，因为某个提前引用的条目(类型、操作或属性)经常在语义上丰富并且相当接近于通常使用的名词。如果读者发现有混淆，建议跳过示例，先通读全文建立起总体框架后再细读所包含的例子。

5.3 缩略语

本标准采用以下缩略语：

- ATS 抽象试验套件(Abstract Test Suite)
- C++ 基于 C 的面向对象扩展的程序语言
- LISP 基于列表处理的程序语言，该标准称为 Common LISP
- MBR 最小包含范围(Minimum Bounding Region)
- OCL 对象约束语言(Object Constraint Language)
- SQL3 开发 SQL 99 期间采用的通用名称
- SQL99 1999 年接受的 SQL 语言规范，包含面向对象数据类型扩展机制
- TIN 不规则三角网(Triangulated Irregular Network)
- UML 统一建模语言(Unified Modelling Language)
- 2D 2 维(2-dimensional)
- 3D 3 维(3-dimensional)

6 几何包

6.1 语义

几何包(图 4)包含了坐标几何的各个类。所有这些类都从根 GM_Object(GM_对象)继承一个对于坐标参照系的可选关联。所有通过定义在本标准中的接口对外公开的接口位置将处在与几何对象相关联的坐标参照系中。所有几何复形、组合形或聚集形的元素都与相同的坐标参照系相关联。当 GM_Object 的实例聚集在另一个已经定义有坐标参照系的 GM_Object(例如 GM_Aggregate 或 GM_Complex)上时，除非单独说明，否则假定这些元素具有相同的坐标参照系。

几何包有几个内部包,即分离的单形、聚集形和复形,其中复形具有比简单的聚集形更精巧的内部结构。图4显示了几何包之间的依赖关系以及对于每一个包的类的列表。

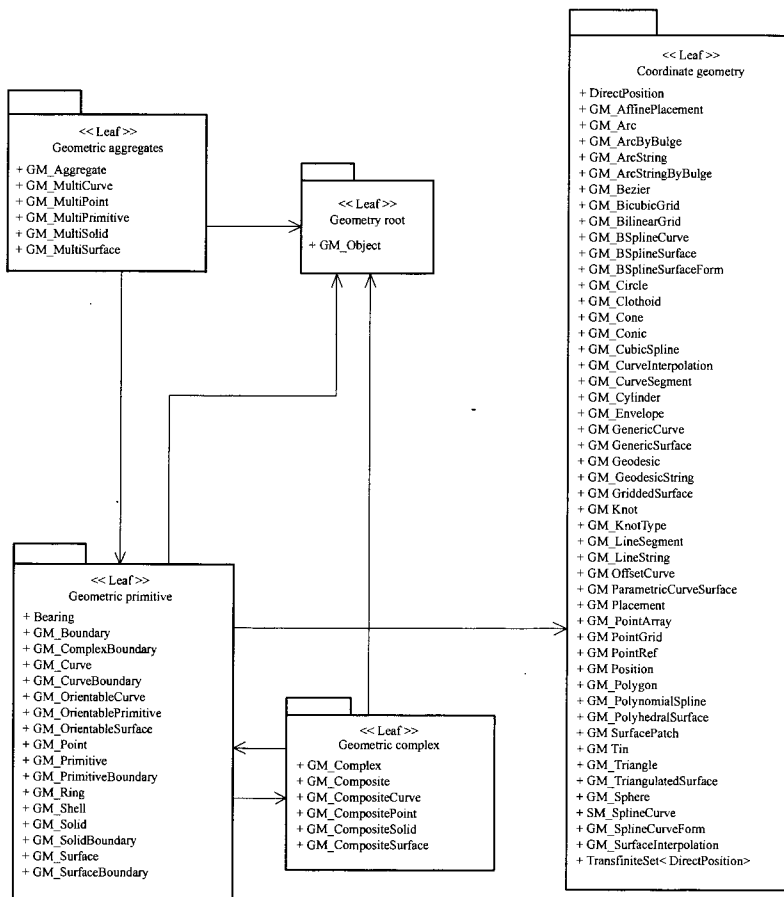


图4 几何包:类的内容和内部依赖关系

图5所示为定义在几何包中的基本类。任何继承 GM_Object 语义的对象都作为直接位置(direct position)的集合。它的行为由它所包含的直接位置决定。GM_Primitive(GM_单形)中的对象是开放的,即它们不包含它们的边界点;曲线不包含它们的端点,曲面不包含它们的边界曲线,体不包含它们的边界曲面。GM_Complex(GM_复形)中的对象是闭合的,即它们包含它们的边界点。这可能导致一些明显的含混。一条线作为一个单形表述时必然引用它的端点,但是在直接位置的集合中却不包含这些端点。一条线作为一个复形表述时,也将引用它的端点,并且将把端点包含在直接位置的集合中。这意味着,同样的数值表述将有轻微的语义差别,这取决于是否把它们处理为单形还是复形。

最明显的语义差别是在 GM_CompositeCurve(GM_组合曲线)。组合曲线是用来表达在几何上也可以被表述为曲线单形的要素。从制图学的观点,这两个表述是没有差别的。从拓扑学的观点,它们是不同的。这个差别出现在继承图(图 5)上,即作为 GM_CompositeCurve 和 GM_OrientableCurve(GM_可定向曲线)之间的继承关系上。GM_CompositeCurve(见 6.6.5)的原始语义是作为一个闭合的 GM_Object,但是在 GM_Primitive 中,它也可以作为一个开放的 GM_Object 来操作(见 6.3.10)。建立在该对象上的拓扑细节上的接口协议,必须区分它们是从 GM_Primitive 继承来的还是从 GM_Complex 继承来的。尽管这些协议继承了定义在 GM_Object 中相同的操作,它们将根据各自进化过程中所经历的不同途径而具有不同的行为。专用标准实现的创建者可以把这个因素考虑进去,采用代理机制来实现导致语义分歧的这些关系。在那些不允许多重继承或对于方法绑定假设作了限制的系统,这个过程在面向对象的程序和数据库中将是必要的。

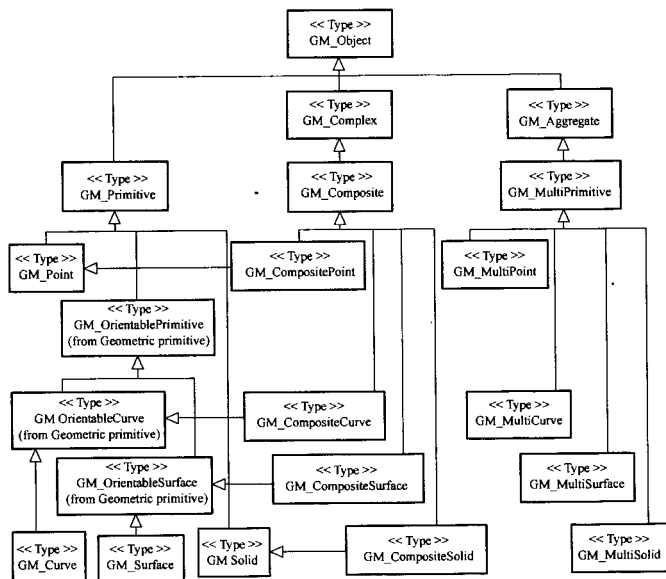


图 5 具有特化关系的几何基本类

6.2 几何根包

6.2.1 语义

几何对象是坐标几何与坐标参照系的结合。在所有的操作中,所有的几何计算都应以最先使用的几何对象的坐标系为准,因为它们通常为操作的调用者。返回的对象应该是在进行计算的坐标参照系中,如果返回的对象不在该坐标系下则必须明确说明。本条定义的接口协议基本上是来自于集合理论。通常,几何对象是由直接位置(见 6.4.1)描述的几何点的集合。几何对象的对象实例化是 GM_Object,用数值对几何点进行对象实例化就是直接位置。定义在 GM_Object 中的通用集合理论操作随着层次向下遗传,根据边界点的直接位置是否包含在集合元素中而各不相同。GM_Primitive 的子类型不包含边界点,而 GM_Complex 的子类型包含边界点。

GM_Object 和 GM_Primitive 是纯粹的抽象类,即在应用模式中不能用对象(或数据结构)直接实例化它们。这些类的实例必须是它们非抽象化的子类型(如 GM_Point、GM_Curve 或 GM_Surface)的

实例。而 GM_Complex 就不是这样,它可以通过应用模式直接实例化,而不必是 GM_Composite 的非抽象子类的实例。虽然在本标准中 GM_Complex 没有显式地实例化,但应用模式可以在与本标准一致的类库中包含一个称作“GM_Complex”的具体的类。应用模式的名称空间是不同于本标准所采用的名称空间,因此看上去不合逻辑的名称实际上是可用的。在本标准中的抽象类可不是这样,这些类在逻辑上不支持直接实例化。这些抽象类型在逻辑上并不直接实例化,而是通过它们的具体子类的构造函数进行实例化。

与 UML 中的通常用法相比,这是对“抽象”的更严谨解释,并且适合作为对应用模式开发者的一个向导。

6.2.2 GM_对象(GM_Object)

6.2.2.1 语义

GM_Object(见图 6)是几何对象分类学上的根类,它支持通常地理上引用的所有几何对象的接口。GM_Object 的实例是在一个特定坐标参照系下直接位置的集合。GM_Object 可以被看作是点的无穷集(infinite set),这些点满足对于直接位置的超限集 TransfiniteSet(DirectPosition)的集合操作接口。因为无限组合的类不能直接实现,所以“包含”的布尔测试是由 GM_Object 接口提供的。本标准主要关心矢量几何类,但对于栅格几何类的进一步工作可以直接使用 GM_Object 作为根类而无需修改。

注:作为一个类型,GM_Object 没有一个明确定义的缺省状态或值来作为一种数据类型的表达。GM_Object 只能在其子类中实例化。

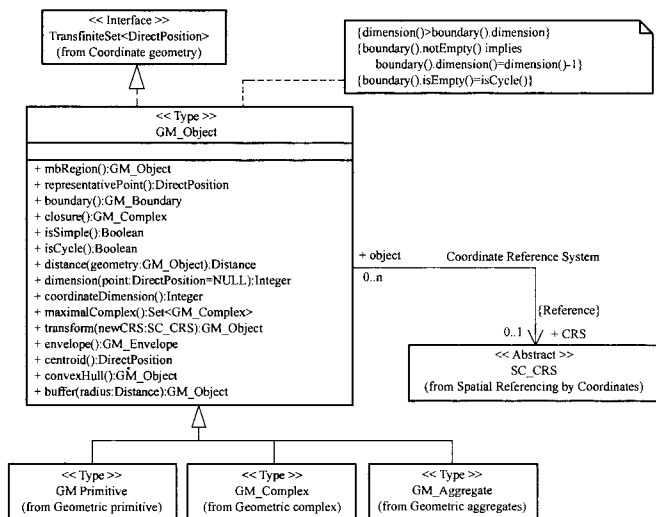


图 6 GM_对象(GM_Object)

6.2.2.2 mbRegion(最小限定区域)

“mbRegion”操作只是作为接口包含在这里,不同应用可以选择不同的方式进行实现。它将返回在坐标系下包含该 GM_Object 对象的一个区域。缺省状态将返回一个 GM_Object 子类的实例,该子类表达从 GM_Object::envelope(包络)操作返回的相同的空间集合。mbRegion 最通常的用法是用来支持使用范围(extents)而不是它的最小边界矩形(MBR 或 envelope)的索引方法。

GM_Object::mbRegion():GM_Object

返回的 GM_Object 为一个非矢量的几何表达也是可以的,虽然这些类型在本标准中没有定义。

6.2.2.3 representativePoint (代表点)

“representativePoint”操作只是作为一个接口包含在这里,它可以用不同的方式实现。它将返回保证在该 GM_Object 上的一个点的值(DirectPosition)。缺省逻辑使用由“GM_Object::centroid”操作返回的点(如果该点在这个对象上)的直接位置。

GM_Object::representativePoint();DirectPosition

RepresentativePoint 操作还可用于符号化表达系统中的注记配置。本标准不讨论符号化和注记配置。

6.2.2.4 boundary (边界)

“boundary”操作应该返回由若干 GM_Object 组成的一个有限集合,这些 GM_Object 包含在该 GM_Object 边界上的所有直接位置。这些对象的组合在适当的场合具有更多的内部结构,并且应该作为 GM_Boundary(GM_边界)数据类型的子类来表达,而 GM_Boundary 本身是 GM_Complex 的子类。返回的这些 GM_Objects 的有限集合应该与该 GM_Object 在同一个坐标参照系中。如果该 GM_Object 是在一个 GM_Complex 内,那么返回的这些边界 GM_Objects 应该在这同一个 GM_Complex 中。如果该 GM_Object 不在任何的 GM_Complex 上,那么返回的这些边界 GM_Objects 可能是为了响应这个操作而构造的。

GM_Object::boundary();GM_Boundary

返回的该集合的组织方式取决于该 GM_Object 的类型。下面描述的 GM_Object 的每个子类都非常完整地定义了它的边界集合的组织方式。

边界元素的空间维数应该比原始元素的空间维数低。

//边界上的所有对象的空间维数都比原始 GM_Object 的空间维数至少 1

boundary→select(dimension)(<=self, dimension-1

6.2.2.5 closure (闭包)

“closure”操作应该返回由若干 GM_Object 组成的一个有限集合,这些 GM_Object 包含在该 GM_Object 边界和对象上的所有的点(即该对象与它的边界的联合)。这些对象组合将在适当的场合具有更多的内部结构。返回的这些 GM_Objects 的有限集合应该与这个 GM_Object 在同一个坐标参照系中。如果该 GM_Object 是一个 GM_Complex,则返回的这些边界 GM_Objects 也应该是在该 GM_Complex 上。如果该 GM_Object 不在任何的 GM_Complex 上,那么返回的这些边界 GM_Objects 可能是为了响应这个操作而构造的。

GM_Object::closure();GM_Complex

6.2.2.6 isSimple (简单性判断)

如果该 GM_Object 没有自交或自切的内部点,“isSimple”操作将返回 TRUE。在数学形式上,这意味着在对象的内部的每一个点都必须有一个可度量的邻域,该邻域与该对象的交同构于一个 n 维球,这里 n 是该 GM_Object 的空间维数。

GM_Object::isSimple();Boolean

因为大多数的坐标几何都是直接或间接由它们的拓扑维的欧几里德空间区域的函数来表达,对于简单性最容易的测试就是需要有一个函数存在,该函数是一对一并且是双向连续的。这样一个函数是拓扑同构的。这个测试对于“闭合的”对象(即对于那些 isCycle 操作中返回 TRUE 的对象)不起作用。

当 GM_Complex 只包含简单 GM_Object 时,非简单的 GM_Object 通常被用在“非结构”的数据集里。

注:“非结构”是一个贬义的词语,通常是暗指数据中在应用前应该清除掉的不受欢迎的几何异常或矛盾。这些矛盾可能包括(但不限于)下列的一些异常类型:

1) 未及线(undershot);一条线本来应该与另一条线相交,但是由于有一小点误差没有交上。这与真正的“邻

近未交”常常难以区分(例如,某处一条道路与另一条道路被一堵仅仅一砖厚的墙所隔开)。当一张曲面或多边形边界的悬挂线未能闭合时,这个问题尤其难以处理。这通常是由于数值化仪在太小的比例尺上工作,而未能抓住线的端点所致。

- 2) 过伸线(overshot):一条线本来应该相交并终止于另一条线,但它走过了,在交点的另一边留下一小段过剩的线头。这通常是由于数值化仪在太小的比例尺上工作,而试图在视觉上抓住线的端点所致。
- 3) 端点打圈(end loop):一条线本来应该相交并终止于另一条线,但它走过了然后又折回来,留下一个小的多余的环在线交点的端点的另一边。这通常是由于数值化仪在太小的比例尺上工作,在它已经过伸后还试图抓住这根线所致。
- 4) 碎条与裂缝(splinters and gaps):很多线,本来是表达同一个几何对象,但是相互之间不一致,在两个曲面的边界处留下一些细碎长条状的重叠区域和裂缝。在辫状河流区域,这种问题尤其难于处理,这里自然要素的真实的几何形状本来就类似于由于糟糕的数值化作业造成的重叠区与裂缝。这通常是由于对于同一区域的多重数据来源,当它们被合并到同一数据库中,但没有很好地进行合并处理所致。

“非结构”数据的真正问题经常是来自于为了改正上述问题而使用的纠正方法(既可能是手工的也可能是自动的)所导致的附加但又各不相同的实际误差。这可能是几何数据中严重的质量问题。

6.2.2.7 isCycle (圈判断)

在拓扑简化(去掉了在非构造性聚集,例如 GM_Aggregate 的子类的成分之间的重叠)之后,如果 GM_Object 有一个空的边界,则“isCycle”操作返回 TRUE。可以说一条“闭合曲线(closed curve)”是“闭合的(closed)”。这造成一些混淆,因为对于“闭合的”这个词有两个截然不同并且不相容的定义。圈(cycle)这个词的使用是非常少的(通常限于代数拓扑领域),这就减小了混淆。尤其在某一欧几里德空间内如果一个对象同构于一个是某个区域的边界的几何对象,则该对象就是一个圈。这样,点是圈,因为它的边界是空。如果一条曲线同构于一个圆(circle)(具有相同的始点和终点),这样的曲线就是一个圈。如果一张曲面同构于一个球面或某个圆环面(torus),该曲面就是一个圈。一个大小有限的体在3维空间中不是一个圈。

GM_Object::isCycle(); Boolean

范例:下列的 OCL 使用边界操作来产生一个 GM_Object,然后使用 TransfiniteSet<DirectPosition>::isEmpty()算子测试一个空集。

```
GM_Object:
    isCycle()=boundary().isEmpty()
```

6.2.2.8 distance (距离)

“distance”操作将返回该 GM_Object 与另一个 GM_Object 之间的距离。这个距离定义为两个 GM_Object 的点集间的最小距离。“Distance”值是与距离单位(例如米或厘米)相联系的正数。在计算距离之前,如有必要,第二个几何对象应该转换到与第一个几何对象相同的坐标参照系中。

GM_Object::distance(geometry:GM_Object); Distance

如果几何对象重叠或接触,则它们相距的距离是 0。目前一些实现中对这种情况使用“负”距离,但这种方法在不同的实现之间既不相容,在理论上也不站不住脚。

“Distance”是量测数据类型的一种单位,定义在 ISO/TS 19103 中。

注:在距离计算中,坐标参照系的作用是重要的。通常,在点之间(因而在几何对象间)至少可以定义三种距离:地图距离、大地距离和地面距离。

地图距离是两点在图上的距离,点的位置是定义在一个投影坐标系下(例如某一比例尺的一张地图上)。通常,地图距离在与比例尺相适应的一个适当小的范围内是精确的。

大地距离(也称测地距离)是沿着被作为坐标参照系的地球模型表面的最短曲线长度。大地距离用于覆盖面积广大的范围,并且要考虑地球的曲率。大地距离常用在大气与海洋航行中,有时还需区别罗盘方位线(方位固定的曲线)距离和大地曲线距离。

地面距离考虑局部垂直位移(地形起伏)。地面距离可能是基于大地距离或地图距离。

6.2.2.9 dimension (维数)

“dimension”操作将返回该 GM_Object 固有的空间维数,它应该小于或等于坐标系的维数。几何对象的组合的空间维数应该是这些几何对象中维数最大的对象的维数。点是 0 维,曲线是 1 维,曲面是 2 维,体是 3 维。局部地,在一个点上,几何对象的维数是该点的局部邻域的维数,即该点的任何坐标邻域的维数。只有给出该 GM_Object 内部的 DirectPosition,维数的定义才是明白无误的。如果传递的 DirectPosition 是 NULL,则该操作将返回对于该 GM_Object 中的所有 DirectPosition 的可能维数的最大值。

GM_Object::dimension(point; DirectPosition=NULL); Integer

6.2.2.10 coordinate Dimension (坐标维数)

操作“coordinateDimension”返回定义该 GM_Object 的坐标的维数,它必然等于该 GM_Object 所在的坐标参照系的维数。

GM_Object::coordinateDimension(); Integer

6.2.2.11 maximalComplex (最大复形)

作为单形的集合,复形可能作为一个集合被包含在另一个更大的复形里,这个更大的复形被称之为原来复形的“超复形(super complex)”。如果没有更大的超复形,该 GM_Complex 就是最大复形。操作“maximalComplex”将返回包含该几何对象的最大 GM_Complexes 的集合。

GM_Object::maximalComplex(); Set<GM_Complex>

如果所采用的应用模式不包含 GM_Complex,则这个操作将返回 NULL 值。

注:作为最大复形的一般语义,不允许任何 GM_Primitive 在多于一个的最大复形中构成强聚集类型。但这也不是绝对的,这取决于实现的语义,在 GM_Primitives 与最大 GM_Complexes 间的关联可能是多对多的。从编程的观点,要动态地构造并维护它们是困难的(虽然不是不可能)。但是作为一个静态的仅供查询的结构,在减少同样的原始几何对象在两种表达中存在数据冗余方面,这可能非常有用。

6.2.2.12 transform (转换)

操作“transform”将返回一个新的 GM_Object,它在允许的转换精度内将对象的坐标从原来的坐标系转换到传进的新坐标系中。

GM_Object::transform(newCRS; SC_CRS); GM_Object

6.2.2.13 envelope (包络)

操作“envelope”将返回该 GM_Object 最小的边界矩形框。这是横跨该 GM_Object 的 DirectPosition 的每一维坐标的最小值与最大值的一个坐标区域。对于 envelope 最简单的表达是用两个 DirectPosition,这第一个点包含了所有坐标的最小值,第二个点包含所有坐标的最大值。但是,有时这两个点可能在该对象的坐标参照系有效范围之外。该操作包含在这里仅仅是作为一个接口,在应用中可以选择以不同的方法来实现。

GM_Object::envelope(); GM_Envelope

6.2.2.14 centroid (质心)

“centroid”操作将返回该对象的数学质心。并不能保证返回的结果还在该对象上。对于单形的异类集合(例如曲线与曲面集合),该 centroid 操作只考虑这些单形中最大维数。例如,当计算曲面的质心,取面积作为权重的平均数。因为曲线没有面积,它们对于面积的平均数没有贡献。

GM_Object::centroid(); DirectPosition

注:可能出现这种情况,质心的位置位于该对象的坐标参照系有效区域范围之外,虽然未必总是这样,大多数坐标参照系的有效范围是凸的。如果可能出现这种意外情况,由实现来决定采取适当的对策。

6.2.2.15 convexHull (凸包)

“convexHull”操作将返回表达该 GM_Object 的凸集的 GM_Object。

GM_Object::convexHull(); GM_Object

注:可能出现这种情况,该 GM_Object 部分地在该对象的坐标参照系有效域范围之外,虽然并不总是这样,因为大多数坐标参照系的有效范围是凸的。如果可能出现这种意外情况,由实现来决定采取适当的对策。

凸要求使用“直线”或“最短长度的曲线”，并且使用不同的坐标系可能导致一个对象的凸集的不同版本。每一个实现应当对这个含混采取适当的措施。对于两个合理的坐标系，在一个坐标系中一个对象的凸集与该对象的凸集转换到另一个坐标系中的镜像可能非常接近。

6.2.2.16 buffer (缓冲区)

“buffer”操作将返回一个 GM_Object，它包含距原来 GM_Object 的距离都小于或等于作为参数传进来的距离。返回的 GM_Object 与原来的 GM_Object 在相同的坐标参照系中。返回的 GM_Object 的空间维数在通常情况下与坐标的空间维数相同，在 2 维空间中是 GM_Surface 的一个集合，在 3 维空间中是 GM_Solid 的一个集合，但这些都是由应用模式来定义的。

GM_Object::buffer(radius;Distance);GM_Object

注：可能出现这种情况，该 GM_Object 可能部分地在该对象的坐标参照系有效范围之外。如果可能出现这种意外情况，由实现来决定采取适当的对策。

6.2.2.17 Coordinate Reference System association (坐标参照系关联)

关联角色“Coordinate Reference System::CRS”链接该 GM_Object 到表达该对象的直接位置所在的那个坐标参照系。如果该关联是空，这个 GM_Object 使用从另一个包含它的 GM_Object 中来的 SC_CRS(坐标参照系)。

GM_Object::CRS[0,1];SC_CRS

注：这种关联是空的普通例子是一个最大 GM_Complex 元素与子复形之间的坐标参照系关联。这个最大 GM_Complex 可以携带 SC_CRS 为它所有 GM_Primitive 元素和所有 GM_Complex 的子复形所共用。该关联仅仅是从 GM_Object 到 SC_CRS 的索引。这意味着在数据集中，坐标参照系对象并不保持一个使用它的 GM_Object 的列表。

6.2.2.18 从 TransfiniteSet (超限集)继承来的操作

6.2.2.18.1 语义

TM_Object 实现下列从接口 TransfiniteSet(DirectPosition)(见 ISO/TS 19103) 继承来的操作。

6.2.2.18.2 contains (包含)

如果该 GM_Object 包含另一个 GM_Object，或包含一个用 DirectPosition 给出的单独的点，操作“contains”将返回 TRUE。设计该算子的目的是为了实现在 TransfiniteSet(DirectPosition)::contains。

GM_Object::contains(pointSet;GM_Object);Boolean // 子集算子

GM_Object::contains(point;DirectPosition);Boolean // 元素包含算子

如果传进的 GM_Object 是一个 GM_Point，那么该操作等价于该点的 DirectPosition 是否在该 GM_Object 内的集合元素测试。由于点和其他的几何对象共享共同的基类(GM_Object)，对于 GM_Object 来说，通常不需要区分点包含与集合包含。下面的 OCL 重申了基本集合理论的公理。

GM_Object:

contains(that;GM_Object) 并且 that.contains(other;GM_Object)

则隐含有 contains(other)

contains(that;GM_Object) 并且 that.contains(p;DirectPosition)

则隐含有 contains(p)

contains(point;GM_Point) 则隐含有 contains(point, position)

注：“Contains”确切地说是集合理论上的包含，并且没有维数限制。在一个 GM_Complex 中，一个 GM_Primitive 不能包含另一个 GM_Primitive，除非有一个维数的跳跃。见 6.3.11.3。

6.2.2.18.3 intersects (交)

如果该 GM_Object 与另一个 GM_Object 相交，操作“intersects”将返回一个布尔值的 TRUE。该算子的目的是实现在 TransfiniteSet(DirectPosition)::intersects。

4

```
GM_Object::intersects(pointSet; GM_Object); Boolean
```

在一个 GM_Complex 内, GM_Primitive 不会与另一个 GM_Primitive 相交。通常, 拓扑结构数据用它们共享的几何对象来获取相交信息。

注: 这个交实际上是集合理论中通常的 DirectPositions 包含。在 GM_Primitive 下, 如果两个 GM_Curve 共享一个共同的端点, 它们不相交, 因为 GM_Primitive 被认为是开放的 (即不包含它们的边界)。在 GM_Complex 下, 如果两个 GM_组合曲线 (GM_CompositeCurve) 共享一个共同的端点, 它们是相交的, 因为 GM_Complexes 被认为是闭合的 (即包含边界)。

6.2.2.18.4 equals (相等)

如果一个 GM_Object 等于另一个 GM_Object, 那么操作“equals”将返回一个布尔值的 TRUE。该算子的目的是实现 TransfiniteSet<DirectPosition>::equals。

```
GM_Object::equals(pointSet; GM_Object); Boolean
```

两个不同的 GM_Object, 如果在相关联的坐标参照系有效范围内对于每一个 DirectPosition, GM_Object::contains 操作都返回相同的布尔值, 则这两个 GM_Object 是相等的。

注: 因为无法测试直接位置的无穷集 (infinite set), equals 操作的内部实现必须测试两个可能完全不同的表述之间的等价性。该测试可能受限于坐标系的分辨率或数据的精度。应用模式可以定义一个容差, 如果两个 GM_Object 具有相同的维数, 并且该 GM_Object 中每一个直接位置与另一个传进来的 GM_Object 的直接位置的距离小于该容差, 那么该操作返回 TRUE, 反之亦然。

6.2.2.18.5 union (联合)

“union”操作返回该 GM_Object 和传进来的 GM_Object 在集合理论中的联合。union 的设计目的是实现 TransfiniteSet<DirectPosition>::union 操作。

```
GM_Object::union(pointSet; GM_Object); GM_Object
```

6.2.2.18.6 intersection (交)

“intersection”操作将返回该 GM_Object 和传进来的 GM_Object 在集合理论中的交集。intersection 的设计目的是实现 TransfiniteSet<DirectPosition>::intersection。

```
GM_Object::intersection(pointSet; GM_Object); GM_Object
```

6.2.2.18.7 difference (差)

“difference”操作将返回该 GM_Object 和传进来的 GM_Object 在集合理论中的差。difference 的设计目的是实现 TransfiniteSet<DirectPosition>::difference。

```
GM_Object::difference(pointSet; GM_Object); GM_Object
```

注: difference 操作不是对称的, 并且通常 A.difference(B) 不等于 B.difference(A)。

6.2.2.18.8 symmetricDifference (对称差)

“symmetricDifference”操作将返回该 GM_Object 和传进来的 GM_Object 在集合理论中的对称差。symmetricDifference 的目的是实现 TransfiniteSet<DirectPosition>::symmetricDifference。

```
GM_Object::symmetricDifference(pointSet; GM_Object); GM_Object
```

6.3 几何单形包

6.3.1 语义

几何单形包包含了所有几何单形和所支持的用来描述它们的边界的数据类型。

6.3.2 GM_边界 (GM_Boundary)

用来表达几何对象边界的所有数据类型的数据根数据类型就是 GM_Boundary (见图 7)。

任何 GM_Object 的子类都使用 GM_Boundary 的子类通过操作 GM_Object::boundary 来表达它的边界。根据几何性质, 边界对象应该是圈。

```
GM_Boundary;
{isCycle()=TRUE}
```

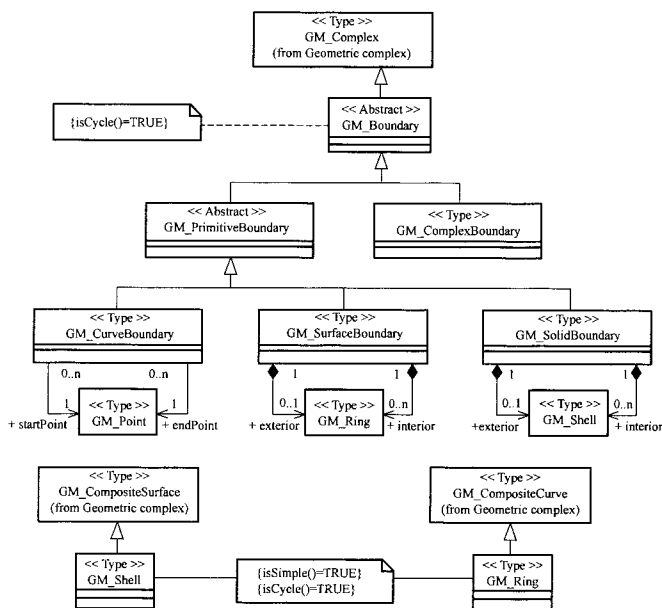


图 7 GM_边界(GM_Boundary)

6.3.3 GM_复形边界(GM_ComplexBoundary)

对于 GM_Complex 对象的边界的操作将返回一个 GM_ComplexBoundary,它是若干单形与一个 GM_Complex 的组合,该 GM_Complex 的维数比原对象维数少 1 维。

6.3.4 GM_单形边界(GM_PrimitiveBoundary)

抽象类 GM_PrimitiveBoundary 是对于 GM_Primitive 的子类的边界算子的各种返回类型的根。因为点没有边界,所以对于点的边界不需要特别的子类。

6.3.5 GM_曲线边界(GM_CurveBoundary)

6.3.5.1 语义

GM_Curve 的边界表达为 GM_CurveBoundary。

6.3.5.2 startPoint, endPoint (始点、终点)

GM_CurveBoundary 包含两个 GM_Point 引用。

GM_CurveBoundary::startPoint; Reference<GM_Point>;

GM_CurveBoundary::endPoint; Reference<GM_Point>;

6.3.6 GM_环(GM_Ring)

GM_Ring 是用来表达作为 GM_SurfaceBoundary(GM_曲面边界)的相互连接的独立成分。它包含若干 GM_OrientableCurve(GM_可定向曲线)的引用,这些 GM_OrientableCurve 连成一个圈(其边界是空)。

GM_Ring 在结构上类似于 GM_CompositeCurve,在这个序列里的每一条 GM_OrientedCurve(GM_有向曲线)的终点是下一条 GM_OrientedCurve 的始点。由于该序列是循环的,对于这条规则没有例外。每一个环象所有的边界一样,都是一个圈,并且每一个环都是简单的。

```
GM_Ring;
{isSimple()=TRUE}
```

虽然每一个 GM_Ring 是简单的,但边界不一定是简单的。最简单的例子是一张曲面的内环相切于它的外环。应用实现中,可以对于相交的边界元素施加更强的限制。

6.3.7 GM_曲面边界(GM_SurfaceBoundary)

6.3.7.1 语义

GM_Surface 的边界表达是 GM_SurfaceBoundary。

6.3.7.2 exterior, interior (外部、内部)

一个 GM_SurfaceBoundary 由若干 GM_Ring 构成,对应于它的边界的各种成分。在通常的 2 维情况下,这些环中有一个是外界边界环。在更一般的流形(manifold)情况下,这并不总成立,在有的情况下,所有的边界都是内边界,外界边界是空。

```
GM_SurfaceBoundary::exterior[0,1];GM_Ring;
GM_SurfaceBoundary::interior[0..n];GM_Ring;
```

注:这里所用的外部和内部并不打算调用几何对象的“interior”和“exterior”的定义。这些术语是其普通的用法,并且反映一个语义上的隐喻,即对于在一个容器内的一个对象内的概念使用相同的语义构造。在通常的数学术语中,外边界是出现在 Jordan(约旦)分隔定理(Jordan 曲线定理在 2 维以外的扩展)中的曲线。外边界是分隔该曲面(或 3 维中的体)与无限外空间的这样一种边界。内边界是分隔该对象与其他有界对象的边界。外部唯一性来自于无界空间的唯一性。本质上,Jordan 分隔定理显示了通常 2 维或 3 维空间分别通过插入环或壳,来分隔有界部分与无界部分。本标准最多到 3 维空间。

范例 1:如果有一个无限的圆筒流形,在圆筒的两个横切面之间定义一个紧致的(compact)曲面,并且这两个横切面分隔开该圆筒的无界部分。在这种场合,这两个切面都有理由被称为外部。对这种容易混乱的情形,本标准选择列出所有“内部”边界的集合。只在 2 维平面空间 E^2 中才能保证外部边界的唯一性。

范例 2:用球的赤道产生一个 1 m 宽的缓冲区,我们得到一个具有两个同构的边界成分的曲面。没有不带偏见的方

法来区分这两个边界哪一个是外部。

6.3.8 GM_壳(GM_Shell)

GM_Shell 是用来表达 GM_SolidBoundary 的单个连接成分。它由若干连在拓扑圈(它的边界是空)上的 GM_OrientableSurface(GM_可定向曲面)的引用构成。不像 GM_Ring,构成 GM_Shell 的元素(曲面片)没有自然的排序方法。与 GM_Ring 一样,GM_Shell 也是简单的。

```
GM_Shell;
{isSimple()=TRUE}
```

6.3.9 GM_体边界(GM_SolidBoundary)

6.3.9.1 语义

GM_Solid 的边界用 GM_SolidBoundary 来表达。

6.3.9.2 exterior, interior (外部、内部)

GM_SolidBoundary 类似于 GM_SurfaceBoundary。在通常的 3 维欧几里德空间中,壳用来区分其外部。在更一般的情况下,这并非总是可行。

```
GM_SolidBoundary::exterior[0,1];GM_Shell;
GM_SolidBoundary::interior[0..n];GM_Shell;
```

注:对于没有外壳的体,一个可供选择的方法是定义有限体的“补(complements)”。所定义的无限体将只有内部边界。如果本标准拓展到 4 维欧几里德空间,或者采用 3 维紧致流形(3D compact manifolds)(可能在地理信息中不存在这种空间),可能就有其他样式的没有外边界的有界体。

6.3.10 GM_单形(GM_Primitive)

6.3.10.1 语义

GM_Primitive(图 8)是几何单形的抽象根类。它的主要目的是定义在各个空间维中将

系起来的基本“boundary”操作。GM_Primitive 是在系统中不能再进一步分解成其他的单形的几何对象。GM_Primitive 包括曲线和曲面(也包括点与体),虽然它们分别是由曲线段和曲面片构成的,但这个构成是强聚集,即曲线段和曲面片不能单独存在于单形的相关环境之外。

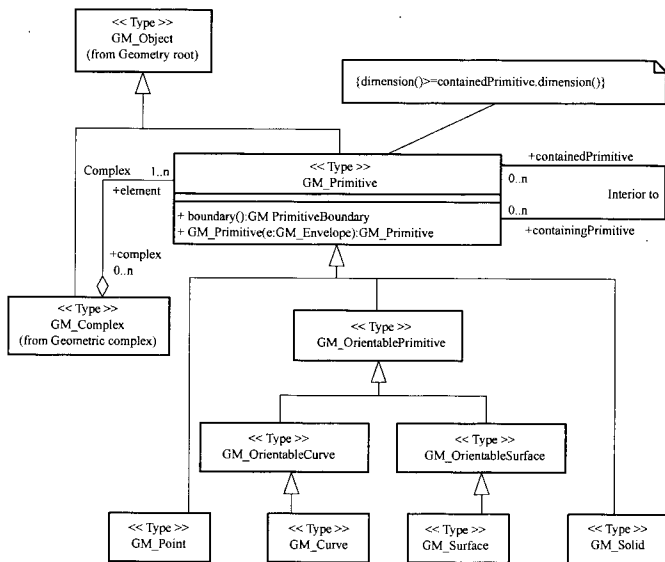


图 8 GM_单形(GM_Primitive)

注: 大多数的几何单形是无限再分解的, 在一条线上加一个中点就可将这条线分为两条独立的线。在曲面上画一条新的曲线就可以把曲面分成两个部分, 这每一个部分都是一张曲面。这就是为什么对于几何单形的定义为“不可分解(non-decomposable)”, 而不是在几何模型上似是而非的定义——在几何上唯一不可再分的对象是点。通常描述为要素的任何几何对象都是若干几何单形的组合。几何单形的组合可以是也可以不是一个几何复形。几何复形有附加的性质, 例如边界操作闭合和组分互斥。

GM_Primitive 与 GM_Complex 在操作意义、属性和关联上共享大多数的语义。不同的地方是 GM_Primitive 没有包含它的边界(除了边界是空的 GM_Point 这种平凡情况之外), 而 GM_Complex 在所有情况下都包含它的边界。这意味着, 如果一个实例化的对象既实现了 GM_Object 对于 GM_Primitive 的操作, 也实现了对于 GM_Complex 的操作, 那么对于每一个集合理论的操作由它的名称来决定。特别要提到的是, 对于特定对象, 例如 GM_CompositeCurve(GM_组合曲线), GM_Primitive::contains(对于端点将返回 FALSE)是不同于 GM_Complex::contains(对于端点将返回 TRUE)。进一步, 如果一个对象被映射为一个 GM_Primitive 值和一个 GM_Complex 值, 这两个作为 GM_Object 的值是不必相等的。

6.3.10.2 boundary (边界)

“boundary”操作将返回由若干 GM_Primitive 构成的一个集合作为 GM_Primitive 的边界。这是对 GM_Object 操作的一个特化, 它并不限制返回类的组合。GM_Primitive 边界集的组织形式取决于单形的类型。

GM_Primitive::boundary(); GM_PrimitiveBoundary

6.3.10.3 GM_Primitive (constructor) (GM_单形 (构造函数))

GM_Envelope 经常用于查询操作中,因此必须有返回一个 GM_Object 的映射操作。这里提供 GM_Primitive 的一个构造函数。

```
GM_Primitive::GM_Primitive(env; GM_Envelope); GM_Primitive
```

注:该操作的实际返回取决于坐标参照系的维数和 envelope 的范围。在 2 维系统中,这个返回的单形将是一个 GM_Surface (如果该 envelope 没有退化为一个点或一条线)。在 3 维系统中,通常返回的是一个 GM_Solid。

范例:在 GM_Envelope 完全包含在它的 SC_CRS 对象的有效区域内的情况下,它相关联的 GM_Primitive 是一个在该区域里的坐标的各种排列的凸壳。例如,假定在 2 维中一个特定的范围定义为(在下面我们忽略 SC_CRS,假定它是一个全局变量):

```
env; GM_Envelope = < lowerCorner = (x1, y1), upperCorner = (x2, y2) >
```

然后我们就可以取坐标值的各种排列来创建一个多边形角点的列表。

```
multi_point; GM_MultiPoint = { (x1, y1), (x1, y2), (x2, y1), (x2, y2) }
```

如果对于 multi_point(多点)使用定义在 GM_Object 的凸壳函数,可得到一个多边形。

```
multi_point.convexHull() → polygon; GM_Surface
```

在 2 维中,多边形的范围完全是由它的边界来定义的(内部曲面片是平坦的,并且不需要内部控制点),该边界给我们提供了在 2 维中表达 GM_Surface 的数据类型:

```
polygon.boundary → ring; GM_Ring = { string; GM_Linestring =  
<<(x1, y1), (x1, y2), (x2, y2), (x2, y1), (x1, y1)>> }
```

所以,该 GM_SurfaceBoundary 记录的是没有内部孔洞的凸集:

```
boundary; GM_SurfaceBoundary = < exterior = ring, interior = { } >
```

细节见这种类型的每一个形式化定义的相关条目。

6.3.10.4 “interior to”(“内在于”)

“Interior to”关联把按定义相容的若干 GM_Primitive 关联起来。允许应用模式重载 Set(Direct-Position)的解释以及与它相关联的计算几何,并声明一个 GM_Primitive 是“interior to”于另一个 GM_Primitive。

当这些 GM_Primitive 在一个 GM_Complex 之内时,这个关联通常是空的,因为在这种情况下边界信息对于多数用例已足够了。

```
GM_Primitive::coincidentSubelement [0..n]; Reference<GM_Primitive>
```

```
GM_Primitive::superElement [0..n]; Reference<GM_Primitive>
```

该关联被定义在 GM_Object 中的集合算子和维数算子所限定。

```
GM_Primitive;
```

```
superElement → includes(p; GM_Primitive) = GM_Object::contains(p)
```

```
dimension() = coincidentSubelement.dimension()
```

注:当两个 GM_Primitive 不是彼此闭合于另一个上时,就不应使用这种关联。其目的是允许应用模式来补偿内在的且不可避免的舍入、切断以及其他由计算机计算所带来的数学问题。

6.3.10.5 complex (复形)

一个 GM_Primitive 可以在几个 GM_Complexes 中,见 6.6.2。该关联在从单形到复形的这个方向上可能是不可导航的,这取决于应用模式。

```
GM_Primitive::complex [0..n]; Reference<GM_Complex>
```

6.3.11 GM_点(GM_Point)

6.3.11.1 语义

GM_Point(图 9)是由一个且仅由一个点组成的几何对象的基本数据类型。

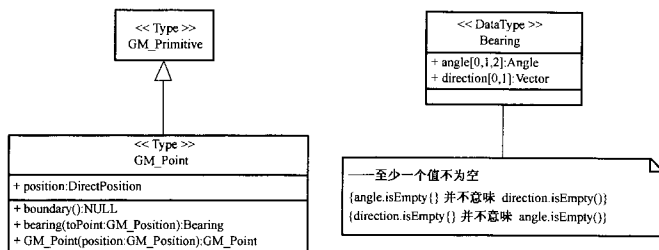


图9 GM_点(GM_Point)

6.3.11.2 position (位置)

属性“position”就是 GM_Point 的 DirectPosition。

GM_Point::position: DirectPosition

GM_Point 是 GM_Primitive 中唯一不能用 GM_Position 来表达它的几何定义的子类。一个 GM_Position 可以是一个 DirectPosition 也可以是一个 GM_Point 的引用(从该引用可以得到一个 DirectPosition)。不允许 GM_Point 使用该方法,以避免无穷循环。应用模式可以采用其他的机制来避免这个逻辑问题。

注:在大多数情况下,GM_Point 的状态是完全由它的 position 属性决定的。唯一的例外是如果该 GM_Point 已经被子类化了以提供一个附加的非几何信息,例如符号化信息。

6.3.11.3 boundary (边界)

“GM_Point::boundary”操作是 GM_Object 边界操作的一个特化,它返回一个 NULL 来指示空集。

GM_Point::boundary(): NULL

6.3.11.4 bearing (方位)

“bearing”操作返回在该 GM_Point 和传进来的 GM_Position 之间的曲线(在该 GM_Point 处)的切线方向。

GM_Point::bearing(toPoint: GM_Position): Bearing

对于所定义的方位的曲线类型的选择依赖于定义 GM_Point 的 SC_CRS。例如,对于墨卡托投影,该曲线是经线方向。在以地球为中心的 3 维坐标系中,该曲线可能是连接这两个点沿着大地水准面或所使用的椭球面的大地曲线。支持这个函数的应用实现应定义所使用的曲线的性质。

注:“Vector(矢量)”类型是一个定义在 ISO/TS 19103 中的公共数据类型。

6.3.11.5 GM_Point (Constructor) (GM_点(构造函数))

构造函数 GM_Point 在给定位置创建一个 GM_Point。

GM_Point::GM_Point(position: GM_Position): GM_Point

6.3.12 方位(Bearing)

6.3.12.1 语义

方位是用来在坐标参照系中表达方向的数据类型。在 2 维坐标参照系中,可以用“从正北方向起算的角度”或在这个方向上的一个 2 维矢量点来表达。在 3 维坐标参照系中,用两个角或任何的 3 维矢量来表达。如果一个角度的集合和矢量都给出了,它们应该是相互一致的。

6.3.12.2 angle (角度)

方位的角度变量通常用在 2 维坐标系中,这第一个角(方位角)是从第一个坐标轴(通常是北)以逆时针方向平行于参考曲面的切平面量测的。如果给出两个角,这第二个角(海拔高度)通常表达在一个平行于参考曲面的切平面的局部平面的上面(作为正角)或下面(作为负角)的这个角度。

Bearing::angle [0,1,2]: Angle

6.3.12.3 direction (方向)

方位变量通常用在 3 维坐标系中, direction 用来表达该坐标系中的一个任意矢量。

Bearing::direction [0,1]:Vector

6.3.13 GM_可定向单形(GM_OrientablePrimitive)

6.3.13.1 语义

可定向单形(图 10)是那些可以根据它们的内部局部坐标系(流形图)(manifold charts)映射到新几何对象上的对象。对于曲线,定向(orientation)反映在曲线在它的参数化过程中经过的方向。当用作边界曲线,曲面被约束在定向曲线的左侧。对于曲面,定向反映在局部坐标系的方向可以看作一个“右手系统”,“顶部”或在 Z 轴方向的曲面将形成一个右手系统。当用作边界曲面,有界体是在曲面的“下面”。在 3 维空间中,点的定向和体的定向没有直接的几何含义。

GM_OrientablePrimitive 对象本质上是携带带有一个“orientation”颠倒符号(“+”或“-”)的几何单形,这个“orientation”颠倒符号确定该几何单形的定向与所引用的对象的定向是否一致。

注:有几个原因支持从可定向单形中分出“positive(正)”单形子类。首先是根据子类的语义。子类被认为是一个层次结构的类型。根据这个观点,“positive”单形仅仅是一个具有正定向的可定向单形。如果采用过去的一种相反的观点,过去曾认为可定向单形是“positive”单形下面的子类,根据子类的逻辑,“negative(负)”单形应该具有和“positive”单形相同类型的几何描述。唯一的可行方案是将“negative”单形从几何根下面分离出,作为“positive”单形的一种相反类型的引用。这大大增加了子类树的复杂程度。为了减少对象的数目并且绕过这个复杂的逻辑,“正”的定向单形作为自引用(是相应的单形子类的实例),而“负”的定向单形则不是。

可定向单形经常用一个符号(对于该定向的记号)和基类几何元素(曲线或曲面)来表达。该符号数据类型定义在 ISO/TS 19103 中。如果“c”是一条曲线,那么“⟨+,c⟩”就是一条正的定向曲线,“⟨-,c⟩”就是一条负的可定向曲线。在大多数情况下,省略语义符号“⟨,⟩”不会导致混淆,所以“⟨+,c⟩”可以写为“+c”或简化为“c”,“⟨-,c⟩”写为“-c”。如果曲线排列适当,可以完成曲线空间算法。所以:

对于 $c, d: \text{GM_OrientableCurves}$ 如果 $c.\text{endPoint} = d.\text{startPoint}$ 那么

$(c+d) == \text{GM_CompositeCurve} = \langle c, d \rangle$

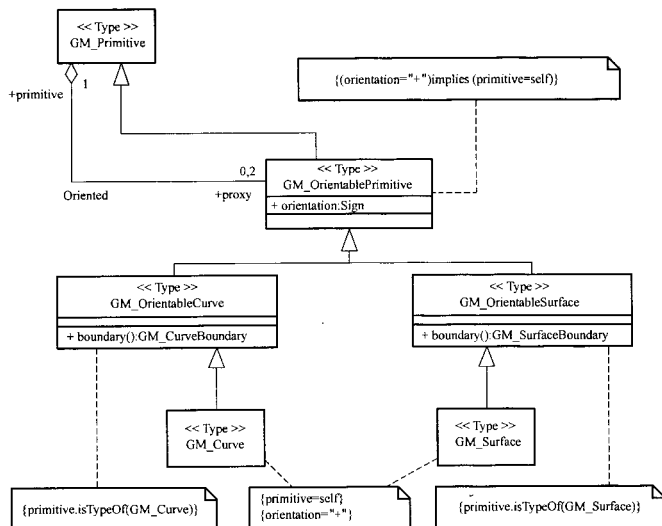


图 10 GM_可定向单形(GM_OrientablePrimitive)

6.3.13.2 orientation (定向)

可定向单形的“orientation”属性确定了该对象为所表达的两个可能的定向(即“+”或“-”)中的哪一个。

```
GM_OrientablePrimitive::orientation; Sign
```

6.3.13.3 Oriented association (定向关联)

每一个1维或2维的GM_Primitive被关联到两个GM_OrientablePrimitive上,每一个可能的定向都有一个GM_OrientablePrimitive。

```
GM_Primitive::proxy [0,2]; Reference<GM_OrientablePrimitive>;
```

```
GM_OrientablePrimitive::primitive [1]; Reference<GM_Primitive>;
```

对于曲线和曲面,每一个几何对象正好有两个可定向单形。

```
GM_Primitive;
```

```
(proxy→notEmpty)=(dimension=1 或 dimension=2);
```

```
GM_OrientablePrimitive;
```

```
a,b;GM_OrientablePrimitive
```

```
((a.primitive=b.primitive)且(a.orientation=b.orientation))指示 a=b;
```

如果定向是“+”(正),则该GM_OrientablePrimitive就是这个对应的GM_Curve或GM_Surface。

```
GM_OrientableCurve;
```

```
orientation=“+” 指示 self.isTypeOf(GM_Curve);
```

```
GM_OrientableSurface;
```

```
orientation=“+” 指示 self.isTypeOf(GM_Surface);
```

6.3.14 GM_可定向曲线(GM_OrientableCurve)

6.3.14.1 语义

GM_OrientableCurve由一条曲线和从GM_OrientablePrimitive继承来的一个定向组成。如果该定向是“+”,则该GM_OrientableCurve就是一条GM_Curve。如果定向是“-”,则该GM_OrientableCurve与该曲线具有相反参数化方向的另一条GM_Curve相关联。

```
GM_OrientableCurve;
```

```
{Orientation=“+” 指示 primitive=self};
```

```
{Orientation=“-” 指示
```

```
primitive.parameterization(length()-s)=parameterization(s)};
```

6.3.14.2 boundary (边界)

“boundary”操作是定义在GM_Object和GM_Primitive中的boundary操作的特殊实现。boundary操作应该返回一个有序的点对,即这条曲线的始点和终点。如果曲线是闭合的,其边界将是空的。GM_CurveBoundary数据类型是定义为这条曲线的边界的简化结构,见6.3.5。

```
GM_OrientableCurve::boundary(); GM_CurveBoundary
```

6.3.15 GM_可定向曲面(GM_OrientableSurface)

6.3.15.1 语义

GM_OrientableSurface由一张曲面和从GM_OrientablePrimitive继承来的一个定向组成。如果该定向是“+”,则该GM_OrientableSurface就是一张GM_Surface。如果定向是“-”,则该GM_OrientableSurface是对于一个GM_Surface的引用。这个被引用的GM_Surface具有与GM_OrientableSurface相反的外法线方向,意思是“该曲面的顶部”(见6.4.33.2)。

```
GM_OrientableSurface;
```

```
{Orientation=“+” 指示 primitive=self};
```

```
{{(Orientation=“-” 并且 TransfiniteSet::contains(p;DirectPosition))
```

```
指示(primitive.upNormal(p)=-self.upNormal(p))};
```

6.3.15.2 boundary (边界)

操作“boundary”专门用于定义在 GM_Object 中的边界操作,具有与 GM_OrientableSurface 相应的返回类型。它将返回 GM_OrientableCurve 循环序列的一个集合,它们限制该 GM_Surface 的外延。对于该 GM_Surface 的每一个边界成分,这些曲线将被组织成曲线的一个循环序列。

GM_OrientableSurface::boundary():GM_SurfaceBoundary;

当“外部”边界没有明确定义时,GM_SurfaceBoundary 所有的环都是作为“内部”列出。

注:曲面外边界的概念仅对 2 维平面有效。一个有界的圆筒具有两个边界组分,但这两个组分在逻辑上都不能被归类为外部。这样,在 3 维空间中就没有涵盖所有情况的“外部”的有效定义。

6.3.16 GM_曲线(GM_Curve)

6.3.16.1 语义

GM_Curve(图 11)是 GM_Primitive 通过 GM_OrientablePrimitive 派生的子类。它是 1 维几何对象的基础。曲线是一个开放区间的连续映像,因而可以被写为一个形如 $c(t):(a,b) \rightarrow E^n$ 的参数函数。这里, t 是一个实型参数, E^n 是一个 n 维空间(通常 2 维或 3 维,由坐标参照系确定)。任何沿着同样的方向,产生同样的映像曲线的其他的参数化方法,例如任何线性平移和缩放,如 $e(t)=c(a+t(b-a)):(0,1) \rightarrow E^n$,都是对同一条曲线的一个等价表达。为了简单起见,GM_Curve 应该以弧长进行参数化,这样参数化操作继承 GM_GenericCurve(见 6.4.7),在 0 到弧长之间的参数将都是有效的。

曲线是连续的、连通的,并且根据坐标系有一个可度量的长度,曲线的定向是由参数化过程决定的,并且与切线函数相一致,该切线函数近似于参数的导数,且始终指向“前进”的方向。由 $c(t):(a,b) \rightarrow E^n$ 定义的曲线相反方向的参数化由从 $s(t)=c(a+b-t):(a,b) \rightarrow E^n$ 函数定义。

一条曲线由一条或多条曲线段构成。在一条曲线上,每一条曲线段可以使用不同的插值方法来定义。每条曲线段在端点处与另一曲线段相连,除非在曲线段列表中上一曲线段指定了下一曲线段的始点。

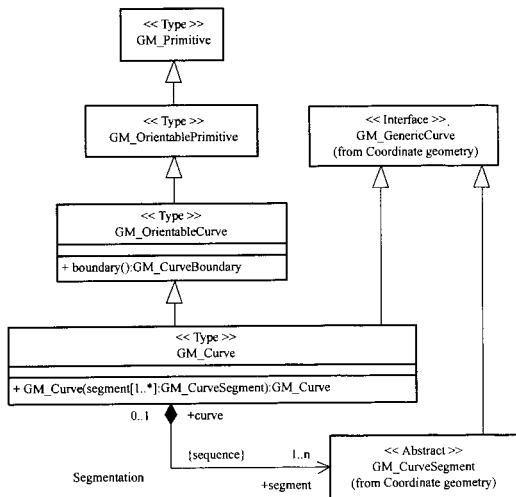


图 11 GM_曲线(GM_Curve)

6.3.16.2 GM_Curve (constructor) (GM_曲线 (构造函数))

GM_Curve 的构造函数采用 GM_CurveSegment(GM_曲线段)的列表来创建 GM_Curve,该列表规定了从头至尾的关系。

GM_Curve::GM_Curve(segment[1..n]; GM_CurveSegment); GM_Curve

6.3.16.3 Segmentation association(分段关联)

该关联“分段”列出 GM_Curve 的 GM_CurveSegment 组分,每一个 GM_CurveSegment 定义沿着曲线的某一部分的点的直接位置。GM_CurveSegment 的次序是用来追踪 GM_Curve 的次序。

GM_Curve::segment[1]; Sequence(GM_CurveSegment)

GM_CurveSegment::curve[0,1]; Reference(GM_Curve)

对于一个特定的间隔参数,GM_Curve 和 GM_CurveSegment 遵从下列规则:

GM_CurveSegment:

{curve.startParam() <= self.startParam()};

{curve.endParam() >= self.endParam()};

{self.startParam() < self.endParam()};

{s:Distance(startParam() <= s <= endParam())

则指示(curve.parameterization(s)=self.parameterization(s));

注:在本标准中,曲线段(curve segment)只出现在一条曲线的相关环境中,因此,作用在该关联中的曲线角色的基数(cardinality)应该是“1”,这可以保证曲线段只能以这种方式被使用。允许基数为“0..1”并不会影响到该标准,这样可以允许其他标准基于这一点以一种更灵活的方式去使用曲线段。

6.3.17 GM_曲面(GM_Surface)

6.3.17.1 语义

作为 GM_Primitive 的子类,GM_Surface(图 12)是 2 维几何对象的基础。不可定向的曲面,例如 Möbius(牟比乌斯)带是不允许的。如果一张曲面不是圈,曲面外边界为逆时针方向的这一面的“外法向(upward normal)”的定向被认为是“向上(up)”的。颠倒每一个边界成分的曲线方向并且交换这个曲面方向“上”“下”的概念,则该曲面的定向颠倒。如果该曲面是体的边界,这个“向上”方向通常是向外的。对于没有边界的闭合曲面,这个“外法向”方向是构成该闭合曲面的曲面片的方向。该 GM_Surface 所包含的这些 GM_SurfacePatches 描述该曲面的内部形状。

注:除了可定向性的限制之外,对于 GM_Surface 没有要求其他的“有效性”条件。

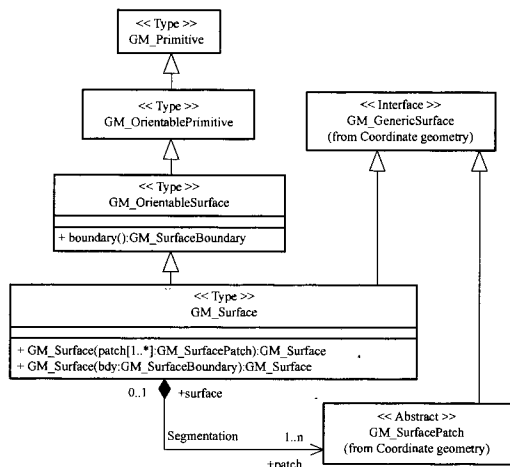


图 12 GM_曲面(GM_Surface)

6.3.17.2 GM_Surface (constructor) (GM_曲面 (构造函数))

对于 GM_Surface, 构造函数的第一个版本采用一个具有适当的边—边关系的 GM_SurfacePatch 列表来创建一个 GM_Surface。

```
GM_Surface::GM_Surface(patch[1..n]; GM_SurfacePatch); GM_Surface
```

第二个版本, 它只保证在 2 维坐标空间中工作, 通过指明它的边界作为 GM_Curve 的一个组合组织成一个 GM_SurfaceBoundary 来构造一个 GM_Surface。在 3 维坐标系空间, 构造函数的第二个版本要求所定义边界的 GM_Curve 实例全部共面(位于同一个平面), 并以此来定义曲面的内部。

```
GM_Surface::GM_Surface(bdy: GM_SurfaceBoundary); GM_Surface
```

6.3.17.3 Segmentation association(分段关联)

“Segmentation”关联指该 GM_Surface 是 GM_SurfacePatche 的一个集合, 这些 GM_SurfacePatche 连在一起形成该 GM_Surface。根据所采用的插值方法, 曲面片的这个集合可能还需要另外的附加结构。通常, 曲面片的形式在应用模式中定义。

```
GM_Surface::patch [0..n]; GM_SurfacePatch
```

```
GM_SurfacePatch::surface [0,1]; Reference(GM_Surface)
```

如果 GM_Surface.coordinateDimension 是 2, 则整个的 GM_Surface 是通过边界的线性插值定义的一个逻辑曲面片。

注: 在本标准中, 曲面片只出现在曲面的相关环境中。因此, 作用在该聚集形中的曲面的基数(cardinality)应该是“1”, 这可以保证曲面片只能以这种方式被使用。允许基数为“0..1”并不会影响到该标准, 这样可以允许其他标准基于这一点, 以一种更灵活的方式去使用曲面片。

6.3.18 GM_体(GM_Solid)

6.3.18.1 语汇

GM_Solid(图 13), 为 GM_Primitive 的子类, 是 3 维几何对象的基础。体的范围由边界曲面定义。

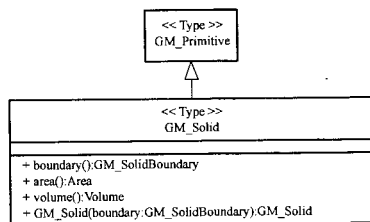


图 13 GM_体(GM_Solid)

6.3.18.2 boundary(边界)

“boundary”操作是对定义在 GM_Object 和 GM_Primitive 中的边界操作的特殊实现, 它具有适当的返回类型。它将返回约束该 GM_Solid 范围的 GM_Surface 集合的序列。这些曲面将被组织成曲面的集合——壳, 来作为 GM_Solid 的边界组分。这些壳的每一个都将是一个圈(没有边界的闭合的组合曲面)。

```
GM_Solid::boundary(); GM_SolidBoundary
```

注: 定义一个体的外壳仅仅因为体所在的坐标空间始终是处在一个 3 维欧几里德空间中。更一般的情况下, 在一个有界 3 维流形中的体没有可辨别的外边界。

在“外部”边界没有明确定义的情况下, 所有 GM_SolidBoundary 的壳将作为“内部”边界列出。

约束一个体的 GM_OrientableSurfaces 被定向向外, 即由它的定向定义的每一个 GM_Surface 的“顶部”将从体的内部指向外。

每一个 GM_Shell 当被视作一个组合曲面时都将是一个圈(见 6.2.2.6)。

6.3.18.3 area (面积)

“area”操作将返回体的所有边界组分的曲面面积的总和。

```
GM_Solid::area():Area
```

Set(GM_Surface)类有一个称作“area”的“column operation(列操作)”，用来累计集合的组分的面积，可用于 GM_Solid:

```
GM_Solid:
    area()=boundary().area()
```

6.3.18.4 volume 体积

“volume”操作返回该 GM_Solid 的体积。它是该体的外边界壳所包含的体积减去所有内边界壳所包含的体积。

```
GM_Solid::volume():Volume
```

6.3.18.5 GM_Solid (constructor) (GM_体 (构造函数))

因为本标准仅限于 3 维坐标参照系,任何体可由它的边界定义。GM_Solid 缺省的构造函数是来自由 GM_Shell 组织成 GM_SolidBoundary 构成的具有适当结构的一个集合。

```
GM_Solid::GM_Solid(boundary:GM_SolidBoundary);GM_Solid
```

6.4 坐标几何包

6.4.1 直接位置(DirectPosition)

6.4.1.1 语义

DirectPosition 对象数据类型(图 14)保存在某个坐标参照系下一个位置的坐标。坐标参照系在 ISO 19111 中描述。由于 DirectPosition 作为数据类型经常被包含在更大的对象(例如 GM_Object),这些更大的对象已经引用了 ISO 19111::SC_CRS,所以,如果这个特别的 DirectPosition 是包含在一个更大的具有对 SC_CRS 引用的对象中,DirectPosition::coordinateReferenceSystem 可能留下一个 NULL。在这种情况下,DirectPosition::coordinateReferenceSystem 隐含地被假定为采用了包含该对象的 SC_CRS 的值。

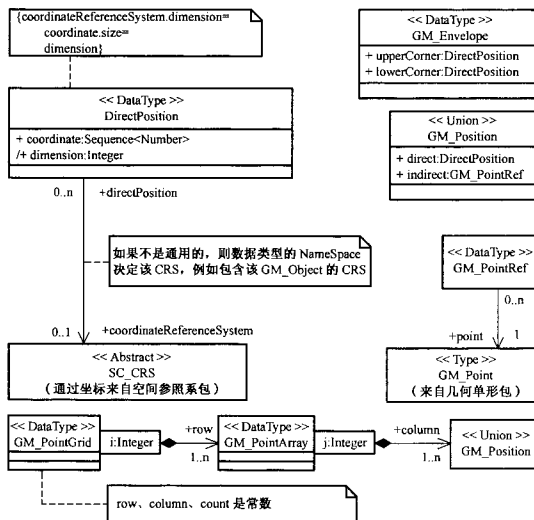


图 14 直接位置(DirectPosition)

9

6.4.1.2 **coordinate** (坐标)

属性“coordinate”是一个数值序列,它包含了在所指定的参照系下该位置的坐标。

DirectPosition::coordinate; Sequence<Number>

6.4.1.3 **dimension** (维数)

属性“dimension”是坐标轴的个数,它由坐标参照系决定。

DirectPosition::dimension; Integer = (coordinate.length)

6.4.1.4 **coordinate ReferenceSystem** (坐标参照系)

关联角色“coordinateReferenceSystem”是所给坐标的坐标系。SC_CRS 定义在 ISO 19111 中。

DirectPosition::coordinateReferenceSystem [0,1]; ISO 19111::SC_CRS

6.4.2 **GM_点引用** (GM_PointRef)

GM_PointRef 是用来引用已经存在的点。它是模板类 Reference(GM_Point) 的一个实例。

GM_PointRef::point; Reference<GM_Point>

6.4.3 **GM_包络** (GM_Envelope)

6.4.3.1 语义

GM_Envelope 经常指一个最小边界盒子或矩形。不管维数,一个 GM_Envelope 可以没有含混地表达为两个坐标点(一个最小点、一个最大点)。要对于一个 GM_Envelope 编码,只要对这两个点进行编码就足够了。它与本标准中的所有数据类型是一致的,它的状态是由它的公共可存取的属性来表达的。

6.4.3.2 **upperCorner** (上角点)

GM_Envelope 的“upperCorner”是由最大坐标值构成的一个坐标位置(coordinate position),该点对于在 GM_Envelope 内所有点的每一维都是最大的。

GM_Envelope::upperCorner; DirectPosition

6.4.3.3 **lowerCorner** (下角点)

GM_Envelope 的“lowerCorner”是由最小坐标值构成的一个坐标位置,该点对于在 GM_Envelope 内所有点的每一维都是最小的。

GM_Envelope::lowerCorner; DirectPosition

6.4.4 **超限集(直接位置)** (TransfiniteSet(DirectPosition))

几何对象的大多数功能都是基于把它们视为 DirectPosition 的无穷集(infinite set)(见图 6)的观点。参数化的类 TransfiniteSet(T) 定义在 ISO/TS 19103 中。

6.4.5 **GM_位置** (GM_Position)

GM_Position 数据类型是一个联合类型,它由一个 DirectPosition 或对 GM_Point 的一个引用构成,从这个对 GM_Point 的引用可以得到一个 DirectPosition。该数据类型的使用允许区分一个位置是直接作为一个坐标(直接变量)还是间接地作为一个对 GM_Point 的引用(间接变量)。

GM_Position::direct [0,1]; DirectPosition;

GM_Position::indirect [0,1]; GM_PointRef;

GM_Position;

{direct, isNull=indirect, isNotNull}

6.4.6 **GM_点数组, GM_点格网** (GM_PointArray, GM_PointGrid)

在本标准中,很多几何构造函数需要使用参考点,这些参考点组织成序列或格网(等长序列的序列)。

GM_PointArray::column[1..n]; GM_Position;

GM_PointGrid::row[1..n]; GM_PointArray;

6.4.7 **GM_一般曲线** (GM_GenericCurve)

6.4.7.1 语义

GM_Curve 和 GM_CurveSegment 两者都代表曲线几何的分段,因此共享若干的操作语义。这些操作语义定义在接口类 GM_GenericCurve 中(见图 15)。

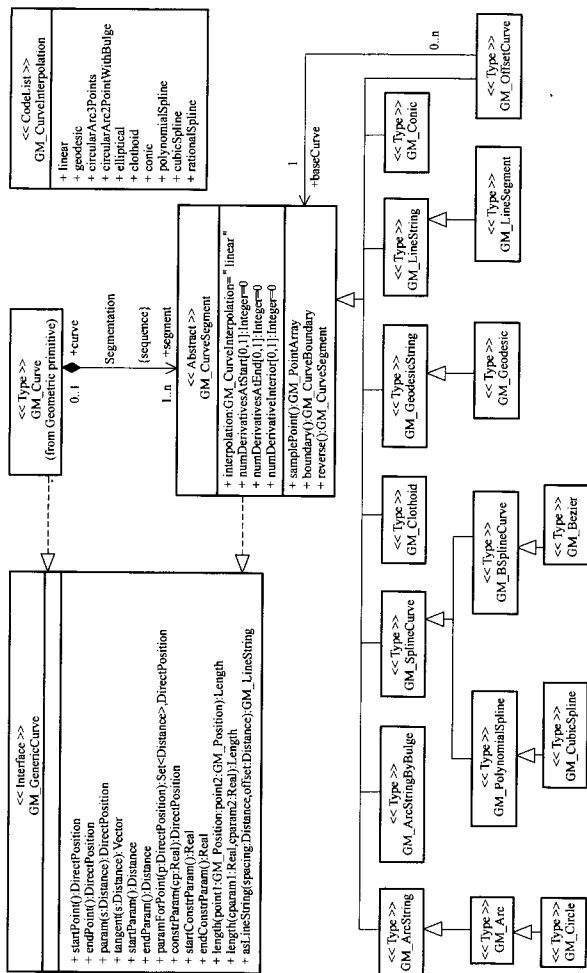


图 15 曲线段类 (Curve segment class)

6.

6.4.7.2 startPoint, endPoint (始点、终点)

“startPoint”和“endPoint”操作将分别返回 GM_GenericCurve 上第一个点和最后一个点的 DirectPositions。这不同于 GM_Primitive 中的边界操作,因为它仅仅返回这两个点的值,而不是所代表的对象。

```
GM_GenericCurve::startPoint():DirectPosition
```

```
GM_GenericCurve::endPoint():DirectPosition
```

6.4.7.3 tangent (正切)

“tangent”操作将返回在所传参数值位置处沿着 GM_GenericCurve 的切线矢量。该矢量近似于曲线的参数导数。该切线矢量是一个单位矢量(长度为 1.0),这与由弧长的参数化相一致。

```
GM_GenericCurve::tangent(s;Distance):Vector
```

6.4.7.4 startParam, endParam (始点参数、终点参数)

startParam 和 endParam 分别指示对于 startPoint 和 endPoint 的参数。

```
GM_GenericCurve::startParam():Distance
```

```
GM_GenericCurve::endParam():Distance
```

```
GM_GenericCurve;
```

```
{parameterization(startParam())=startPoint()};
```

```
{parameterization(endParam())=endPoint()};
```

```
{length()=endParam()-startParam()}
```

一条 GM_Curve 的起始参数和结束参数分别是 0 和曲线弧长。对于在一条 GM_Curve 内的各个 GM_CurveSegment(GM_曲线分段)的起始和结束参数分别等于这些分段在这条 GM_Curve 中的 Segmentation 关联所对应的起始与结束参数(见 6.3.16.3)。所以任何分段(第一段除外)的 startParam 等于上一个线段的 endParam。如果一条 GM_GenericCurve 被用作其他目的,对于这两个参数将有一个限制,即由该 GM_GenericCurve 的弧长所决定的这两个参数(起始参数和结束参数)必须不同。

6.4.7.5 paramForPoint (点的参数)

“paramForPoint”操作将返回传进来的 DirectPosition 所对应的该 GM_GenericCurve 的曲线参数。如果 DirectPosition 不在曲线上,将返回曲线上最近点所对应的参数。

```
GM_GenericCurve::paramForPoint(p;DirectPosition):Set(Distance),
```

```
DirectPosition
```

使用最接近点“p”的 DirectPosition,即它是在 GM_GenericCurve 上最接近所传递的点“p”的点。返回集将只包含一个坐标,除非不是简单曲线。如果在 GM_GenericCurve 上多于一个 DirectPosition 具有距所传递的“p”点相同的最小距离(例如曲线是一段圆弧,所传“p”为圆心),这个返回值可能是所有可选答案中的任意一个。

6.4.7.6 param (参数)

“param”操作将对作为实数间隔的连续映像的曲线作参数化的表达。该操作以所传的距离返回在 GM_GenericCurve 上的 DirectPosition。这个参数化可以是弧长,即从始点沿着 GM_GenericCurve 量测并加到起始参数上去。

```
GM_GenericCurve::param(s;Distance):DirectPosition
```

6.4.7.7 startConstrParam, endConstrParam (始点构造参数、终点构造参数)

“startConstrParam”和“endConstrParam”分别指示用来构造 startPoint 和 endPoint 的参数。

```
GM_GenericCurve::startConstrParam():Real
```

```
GM_GenericCurve::endConstrParam():Real
```

```
GM_GenericCurve;
```

```
constrParam(startConstrParam())=startPoint();
```

```
constrParam(endConstrParam())=endPoint();
```


并没有假定 startConstrParam 比 endConstrParam 小,但是限定参数化是单调的(严格递增或严格递减)。

注:为了方便计算,经常选用构造性参数,而构造性参数很少与作为缺省参数的弧长有任何简单关系。通常,几何构造将使用构造性参数,因为程序员认为这样作合理,当查询的时候计算弧长。

6.4.7.8 constrParam (构造性参数)

“constrParam”操作是将曲线作为实数间隔的连续映像的另一种可替代表达,这里没有限制要求参数一定是代表曲线弧长,对于 GM_Curve 和它的组分 GM_CurveSegments 之间也没有限制。使用该操作通常是为了暴露隐藏在曲线后面的构造函数,当该曲线是用来构造参数曲面时尤其有用。

GM_GenericCurve::constrParam(cp;Real);DirectPosition

6.4.7.9 length (长度)

曲线段的长度是在坐标参照系下对它的长度的数值度量。由于长度是距离的积累,它的返回值将带一个与量测的距离相匹配的量测单位。“length”操作将返回曲线上的两个点沿着曲线的距离。这两个参数的缺省值分别是始点和终点。如果所给的点没有在曲线上,在计算长度之前,该点将被映射到曲线上最近的直接位置。如果曲线是非简单的,并且多次通过这两个所给点的一个或两个,这个返回的距离将是这两个点在 GM_Curve 上的最小距离。

GM_GenericCurve::length(point1;GM_Position=startPoint(),
point2;GM_Position=endPoint());Length

length 操作的第二种形式将直接从构造参数产生,允许用在参数化过程中的变量和构造性参数之间直接转换。

GM_GenericCurve::length(cparam1;Real=startConstrParam(),
cparam2;Real=endConstrParam());Length

在直接位置之间通过缺省参数确定的距离简单地就是参数间的差。length 函数也允许构造性参数与弧长参数间的转换。

如果 $p = \text{length}(\text{startConstrParam}, p2) + \text{startParam}$

则 $\text{parameterization}(p) = \text{constrParam}(p2)$

6.4.7.10 asLineString (作为线串)

“asLineString”函数构造一个线串(线段的序列),这里控制点(线段的端点)位于这条曲线上。如果给出了“maxSpacing(最大间隔)”,即 maxSpacing 非 0,则沿着所产生的曲线在控制点间的距离将小于 maxSpacing。如果给出了“maxOffset(最大偏移量)”,即 maxOffset 非 0,则在所产生的曲线上任何点与原始曲线之间的距离将小于 maxOffset。如果 maxSpacing 和 maxOffset 两个参数都设定了,则这两个准则将同时满足。如果曲线的原始控制点位于曲线之上,则它们将包含在所返回的 GM_LineString 的 controlPoints(控制点)里。如果两个参数都为 0,则所返回的线串将由原始曲线的控制点构造。

GM_GenericCurve::asLineString(maxSpacing;Distance=0,maxOffset;
Distance=0);GM_LineString

注:该函数在创建曲线的线性逼近时有用,即在一些例如显示的简单动作时有用。它经常作为一个“打造曲线(stroked curve)”被引用。为了这个目的,“maxOffset”版本用在作为目标的显示设备上保持这条曲线的最小表达。该函数在准备将一条曲线根据转换控制点从一个坐标参照系转换到另一个坐标参照系时也有用。在这种情况下,通过控制“maxSpacing”的版本是非常合适的。允许这两个参数缺省为 0,并不意味着在地理上或几何上有任何有用的解释,除非知道关于曲线是怎样构造的等更多的信息。

6.4.8 GM_曲线插值(GM_CurveInterpolation)

GM_CurveInterpolation 是由应用模式定义,用来鉴别插值机制的代码列表。作为代码列表,没有打算限制 GM_CurveInterpolation 的可能值。GM_CurveSegment 的子类型可以通过子类直接产生,也可通过定义一个插值方法和相关联的支持它的 controlParameters(控制参数)来间接地定义。用于“插值”的有效方法包括但不仅限于下列类型:

- a) linear(线性),插值机制将返回在每个连贯的控制点对之间的一条直线上的直接位置。

- b) geodesic(大地线):插值机制将返回在每个连贯的控制点对之间的大地曲线(也称测地曲线)上的直接位置。一条大地曲线是一条沿地球大地水准面具有最短长度的曲线。大地曲线由 GM_CurveSegmen 所使用的 GM_Curve 的坐标参照系所决定。
- c) circularArc3Points(3 点圆弧):对于每一个由三个构造性控制点组成的集合,中间一个点是从该控制点序列的首点起算的偶数偏移量,插值机制将返回在圆弧上通过首点、中点到达第三个点的直接位置。控制点的序列将是一个奇数。
注:如果这三个点是共线的,该圆弧将成为一条直线。
- d) circularArc2PointWithBulge(两点带凸圆弧):对于每一个构造性控制点对,该插值机制将返回在圆弧上通过第一个控制点到达第二个控制点的直接位置,相关联的控制参数决定弧的中心点到弦线的中心点间的偏移量,正值往左偏,负值往右偏。因为该项技术定义的限制,这种形式只用在 2 维中。
- e) elliptical(椭圆弧):对于每一个由 4 个构造性控制点组成的集合,插值机制将返回在椭圆弧上从第一个控制点出发通过中间的控制点依次到达第四个控制点的直接位置。
注:如果这 4 个点是共线的,这条弧将成为一条直线。如果这 4 个点是在一个圆上,该弧将成为一条圆弧。
- f) clothoid(回旋曲线):使用一个角度的螺旋或回旋进行的曲线插值。
- g) conic(圆锥曲线弧):类似于椭圆曲线,但使用 5 个构造性参数来决定一个圆锥曲线段。
- h) polynomialSpline(多项式样条):控制点是以一个线串排列,但它们之间是用多项式样条函数进行插值。通常,连续程度由所选的多项式的幂次决定。
- i) cubicSpline(三次样条):在控制点用初始切线方向插值,中间用三次多项式样条插值,它是多项式样条的 3 次幂形式。
- j) rationalSpline(有理样条):控制点以线串形式排列,但它们以有理(多项式的商)的样条函数进行插值。通常连续程度由所选多项式的幂次决定。
- 该列表通过一个代码列表实现,其值根据实际情况可以有所变化。

GM_CurveInterpolation::

```
linear
geodesic
circularArc3Points
circularArc2PointWithBulge
elliptical
clothoid
conic
polynomialSpline
cubicSpline
rationalSpline
```

6.4.9 GM_曲线段(GM_CurveSegment)

6.4.9.1 语义

GM_CurveSegment 定义了一个 GM_Curve 上同类型的线段。每一个 GM_CurveSegment 应该出现在至少一个 GM_Curve 上。

6.4.9.2 interpolation(插值)

“interpolation”属性定义了用在该线段上的曲线插值机制。这个机制使用控制点和控制参数来定义该 GM_CurveSegment 的位置。

GM_CurveSegment::interpolation:GM_CurveInterpolation

6.4.9.3 numDerivatives(可微次数)

属性“numDerivativesAtStar(始点可微次数)”和“numDerivativesAtEnd(终点可微次数)”定义在该

曲线段与它的直接相邻曲线段之间的连续类型,第一个参数用于与它前一个直接相邻的曲线段之间,第二个用于与它的后一个直接相邻的曲线段之间。如果该曲线段是一条曲线的第一条或最后一条曲线段,这两个值一个有用,一个被忽略。“numDerivativesInterior(内部可微次数)”该属性定义了曲线内部连续的类型。缺省值是“0”,意味着简单连接,它是强制性的最低水平的连接,在数学教科书上,这个水平被称作“ C^0 ”。值“1”意味着在它对应的端点处函数的一阶导数连续,即“ C^1 ”连续。对于任何整数的值“n”意味着它的函数和它的第 n 阶导数连续;即“ C^n ”连续。

GM_CurveSegment::numDerivativesAtStart; Integer=0;

GM_CurveSegment::numDerivativesInterior; Integer=0;

GM_CurveSegment::numDerivativesAtEnd; Integer=0;

注:当系统的基本曲线定义未定时,使用这些值是唯一恰当的。例如,线串和线段不支持 C^0 以上的连续,因为没有多余的控制参数来调节线段端点引入的角度。相反,样条函数在线段的端点经常还有富裕的自由度使得它可以调节导数的值以支持 C^1 或更高级别的连续。

6.4.9.4 samplePoint (采样点)

“samplePoint”操作返回位于 GM_CurveSegment 上的点的一个有序数组(GM_PointArray)。在大多数情况下,这些点与用来构造这条线段的控制点有关。

GM_CurveSegment::samplePoint(); GM_PointArray

注:用来控制曲线段形状的 controlPoint(控制点)并不总是在曲线段上。例如,在一个样条曲线上,曲线段是作为 controlPoints 的权重矢量的总和给出。每一个权重函数在构造参数间隔内都有一个最大值,这大致对应于曲线上最接近相应的控制点 controlPoint 的曲线经过点。具有曲线的权重函数最大值的这些点就是该曲线段的采样点。

6.4.9.5 boundary (边界)

GM_CurveSegment 的“boundary”操作的语义相同于 GM_Curve 中“boundary”的工作方式,不同点是 GM_CurveSegment 的端点不一定非得以 GM_Point 的方式存在,在它的 boundary 里可以包含若干的临时 GM_Point。

GM_CurveSegment::boundary(); GM_CurveBoundary

注:上面的 GM_CurveBoundary(GM_曲线边界)几乎总是两个不同的位置,但是像 GM_Curves 一样,GM_CurveSegments 也可以是自圈闭的。最有趣的是,除了与 GM_Curve 关联的 startPoint 和 endPoint 之外,所使用的点都是临时的(被构造并支持返回值)。在 GM_Curve 卷入到一个 GM_Complex 时,这两个位置就像在 GM_Complex 中一样,被表述为 GM_Point。

6.4.9.6 reverse (翻转)

GM_CurveSegment 的“reverse”操作简单地颠倒曲线段参数化的定向。

GM_CurveSegment::reverse(); GM_CurveSegment

6.4.10 GM_LineString (GM_线串)

6.4.10.1 语义

GM_LineString(图 16)由线段的序列构成,每一个线段都有一个像在 GM_LineSegment 中的线段一样的参数构造方法(见 6.4.11)。这个类本质上是将 Sequence<GM_LineSegments> 结合到一个单独的对象中,具有自己的存储空间。

6.4.10.2 controlPoint (控制点)

GM_LineString 中的若干 controlPoint 构成一个位置序列,在这些位置间曲线为线性插值。该序列中的第一个位置是 GM_LineString 的 startPoint,最后一个位置是 GM_LineString 的 endPoint。

GM_LineString::controlPoint; GM_PointArray

6.4.10.3 GM_LineString (constructor) (GM_线串 (构造函数))

GM_LineString 的构造函数采用点的序列,并以这些点作为 controlPoints 来构造一个 GM_LineString。

该 GM_LineString 的构造函数采用两个或更多位置,并且创建适当的线串来连接它们。

GM_LineString::GM_LineString(points[2..n];GM_Position);GM_LineString

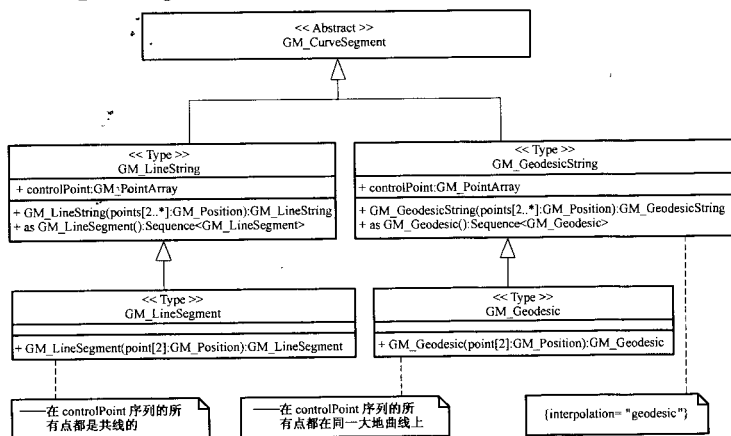


图 16 线性、圆弧和大地曲线插值(Linear, arc and geodesic interpolation)

6.4.10.4 asGM_LineSegment (作为 GM 线段)

asGM_LineSegment 操作分解一个线串为一个等价的线段序列。

GM_LineString::asGM_LineSegment();Sequence<GM_LineSegment>

6.4.11 GM 线段(GM_LineSegment)

6.4.11.1 语义

GM_LineSegment 由一段直线相连的两个不同的 DirectPositions(startPoint 和 endPoint) 构成。这样,插值属性将是线性的。缺省的 GM_GenericCurve::parameterization=c(s) 是:

$(L; \text{Distance}) = \text{endParam} - \text{startParam}$

$c(s) = \text{ControlPoint}[1] + ((s - \text{startParam}) / L) * (\text{ControlPoint}[2] - \text{ControlPoint}[1])$

在该 controlPoint 数组中的任何其他点必然落在这条直线上。这些 GM_LineSegment 的控制点都将位于直线的始点与终点之间。在这些控制点的两两之间的其他位置可以是线性插值的。

注:使用一个构造性参数 t , $0 \leq t \leq 1.0$, 并且 $c(0) = c.\text{startPoint}()$, $c(1) = c.\text{endPoint}()$, 这个线性插值函数是:

$c(t) = c(0)(1-t) + c(1)t$

6.4.11.2 GM_LineSegment (constructor) (GM 直线段 (构造函数))

GM_LineSegment 的构造函数采用两个位置并创建合适的线段连接它们。构造函数在类范围内使用。

GM_LineSegment::GM_LineSegment(point[2];GM_Position);GM_LineSegment

6.4.12 GM 大地曲线串(GM_GeodesicString)

6.4.12.1 语义

GM_GeodesicString 由大地曲线段的一个序列构成。这个类本质上将 Sequence<GM_Geodesic> 结合到一个单独的对象中,并具有自己的存储空间。

6.4.12.2 controlPoint (控制点)

GM_GeodesicString 的 controlPoints 是关于位置的序列,在这些位置之间使用从大地水准面或所使用的坐标参照系椭球体来的大地曲线对该 GM_GeodesicString 进行插值。这些点的组织方式与 GM_LineString(见 6.4.10.2) 相同。

GM_GeodesicString::controlPoint;GM_PointArray

对于一个 GM_GeodesicString 的插值是“geodesic”。

GM_GeodesicString::interpolation; GM_CurveInterpolation = "geodesic"

6.4.12.3 GM_GeodesicString (constructor) (GM_测地曲线串 (构造函数))

GM_GeodesicString 的构造函数采用两个或更多的位置,使用一个定义在所使用的坐标参照系的大地水准面(或椭球体)中的大地曲线进行插值,并且创建一个连接这些位置的大地曲线串。

GM_GeodesicString::GM_GeodesicString(points[2..n]; GM_Position)
: GeodesicString

6.4.12.4 asGM_Geodesic (分解为 GM_测地曲线)

“asGM_Geodesic”操作分解一个大地曲线串为一个等价的大地曲线段序列。

GM_GeodesicString::asGM_Geodesic(); Sequence(GM_Geodesic)

6.4.13 GM_大地曲线(GM_Geodesic)

6.4.13.1 语义

GM_Geodesic 由一条大地曲线连接的两个不同的位置构成。一条 GM_Geodesic 的控制点都将落在大地曲线的始点与终点之间。在这些控制点两两之间,一个由所使用的坐标参照系的椭球体或大地水准面来定义的大地曲线被用来插值产生其他位置。在该 controlPoint 数组中的任何其他点必然落在这条大地曲线上。

6.4.13.2 interpolation (插值)

GM_Geodesic 的 interpolation 是“geodesic”。

GM_Geodesic::interpolation; GM_CurveInterpolation = "geodesic"

6.4.13.3 GM_Geodesic (constructor) (GM_测地曲线 (构造函数))

GM_Geodesic 构造函数使用两个位置,并创建连接它们的适当的大地曲线。构造函数在类中使用。

GM_Geodesic::GM_Geodesic(point[2]; GM_Position); GM_Geodesic

6.4.14 GM_弧串(GM_ArcString)

6.4.14.1 语义

GM_ArcString(图 17)类似于 GM_LineString,不同点在于前者使用圆弧插值。因为需要 3 个点来确定一段圆弧,controlPoints 被处理为重载 3 个 GM_Positions 的集合,每一条弧段的始点、终点和在始点和终点之间的某一点。因为每一条弧段的结束是下一条弧段的开始,在 controlPoint 序列中的 GM_Position 不重复。

6.4.14.2 numArc (弧段数目)

属性“numArc”是该串中圆弧的数目。因为插值方法需要重载 3 点集合,弧段的数目决定了 controlPoints 的数目。

GM_ArcString; numArc; Integer = ((controlPoint.length - 1) / 2)

6.4.14.3 controlPoint (控制点)

属性“controlPoint”是该串中的弧段的点的序列。该序列中的前三个 GM_Positions 确定第一条弧段。以奇数偏移量开始的任意三个连续的 GM_Positions 确定该串中的另一条弧段。

GM_ArcString; controlPoint; GM_PointArray {size = 2 * numArc + 1}

6.4.14.4 interpolation (插值)

GM_ArcString 的 interpolation 是“circularArc3Points(三点圆弧)”。

GM_ArcString::interpolation; GM_CurveInterpolation = "circularArc3Points"

6.4.14.5 GM_ArcString (constructor) (GM_弧串 (构造函数))

GM_ArcString 构造函数使用一个由 GM_Position 定义的点序列,创建一个连接这些点的三点圆弧序列,根据弧串的性质,该位置序列的个数必须是奇数。

GM_ArcString::GM_ArcString(point[3,5,7,...]; GM_Position); GM_ArcString

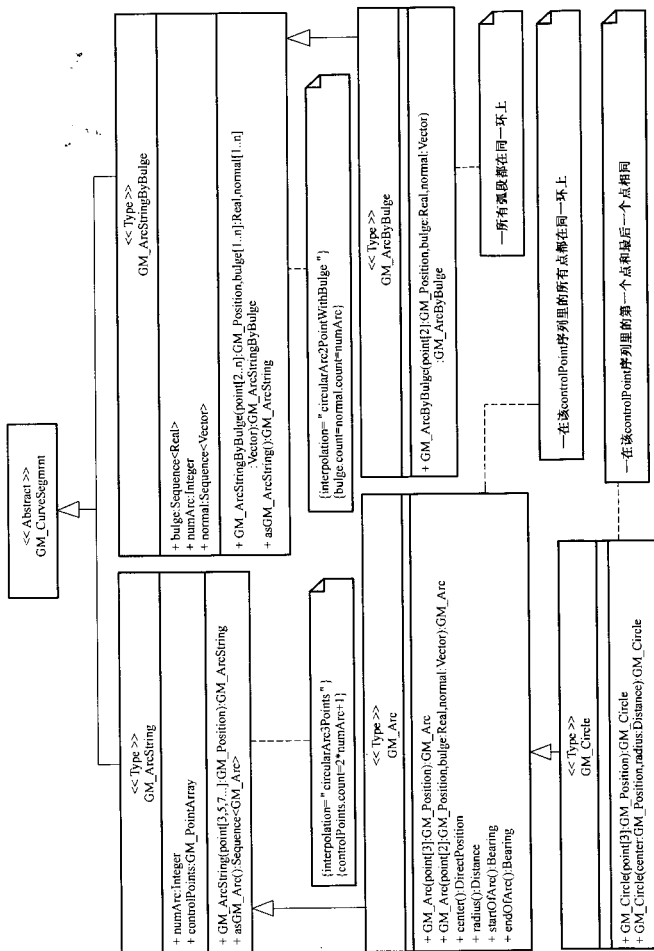


图 17 弧 (Arcs)

6.4.14.6 asGM_Arc (作为 GM_弧)

asGM_Arc 操作构造一个与该弧串几何等价的弧序列。

GM_ArcString::asGM_Arc(); Sequence(GM_Arc)

6.4.15 GM_弧(GM_Arc)

6.4.15.1 语义

一条 GM_Arc 由三个点定义,构成的圆弧由这三点决定,从第一点开始经过第二点,终止于第三点。如果这三点共线,这段弧就是一个三点直线串,并且不能返回圆心、半径、起始角、终止角等参数。

注:在该模型中,GM_Arc 是 GM_ArcString 的子类,仅由一条弧段构成一个平凡的弧串。这可能违反子类比它的父类更复杂的这样一个直觉(因为有附加的方法和属性)。GM_Arc 比 GM_ArcString 更简单在于它具有更少的数据,而其复杂性源于它可以返回圆心、半径、起始角、终止角等几何信息。这些所附加的计算复杂性迫使它成为子类。另外,“...是一种... (is type of)”这个语义在该情况起作用。

在它的最简单的表达中,GM_Arc 的 controlPoint 序列中的 3 点将由这样的次序构成:弧段的始点、弧段中既不在始点也不在终点的另一个点、弧段的终点。

GM_Arc::controlPoint; GM_PointArray = (startPoint; GM_Position,
midPoint; GM_Position,
endPoint; GM_Position)

如果给出了附加点,则所有的点必须在由该控制点数组中任意三个非共线的点定义的圆上。所有的点将位于这同一个圆上,并且控制点数组应该以它们在圆上出现的先后次序给出。

注:对于该 controlPoint 序列的中间 GM_Position 使用“midPoint”这个词并不意味着要求该 GM_Position 是位于弧段的几何中心。从计算稳定性的观点看,该 GM_Position 位于弧段的几何中心是一个最好的选择,但从数学实现上这并不是绝对必要的。

6.4.15.2 GM_Arc (constructor) (GM_弧 (构造函数))

GM_Arc 构造函数使用三个点并且创建相应的弧。

GM_Arc::GM_Arc(point[3]; GM_Position); GM_Arc

GM_Arc 的第二个构造函数使用两个位置和一个弧段的中点与弦线的中点之间的偏移量,由一个距离和方向给出,构造一个相应的弧段。

GM_Arc::GM_Arc(point[2]; GM_Position, bulge; Real, normal; Vector); GM_Arc

所产生的弧的中点这样给出:

midPoint = ((startPoint + endPoint) / 2.0) + bulge * normal

在 2 维坐标参照系下,该 bulge(凸)可以给出一个符号,并且可以假定法线是垂直于弧的始点和终点的直线段(即弧的弦线),并指向左。

例如:如果这两个点是 $P_0 = (x_0, y_0)$ 和 $P_1 = (x_1, y_1)$, bulge 是 b , 从 P_0 到 P_1 的矢量方向是:

$$\vec{u} = (u_0, u_1) = (x_1 - x_0, y_1 - y_0) / \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

为了形成一个右手局部坐标系 $\{\vec{u}, \vec{v}\}$, 这两个矢量的点积为 0, 交积模为 1。可以检验,“向左(leftward)”法向量使得该矢量对为:

$$\vec{v} = (v_0, v_1) = (-u_1, u_0)$$

弧的中点,由弦线的中点加上偏移量,变成:

$$m = \frac{P_0 + P_1}{2} + b\vec{v}$$

如果 $b > 0$ 则向左, $b < 0$ 则向右。

6.4.15.3 center (中心)

center 操作计算该弧所在圆的圆心。返回的 DirectPosition 的坐标参照系与 GM_Arc 所在的坐标参照系相同。在一些极端的情况下,计算所得的 DirectPosition 可能位于 GM_Arc 所使用的坐标参照系的有效范围之外(尤其是如果潜在的弧具有一个非常大的半径)。应用模式可以对这种情况选择一个适当的策略。

0

GM_Arc::center():DirectPosition

6.4.15.4 radius (半径)

radius 操作返回所在圆的半径。

GM_Arc::radius():Distance

6.4.15.5 startOfArc (弧段起始方位)

startOfArc 操作计算从弧所在圆的圆心到弧的始点这条直线的方位。在 2 维情况下,这就是一个起始角。在 3 维情况下,由该弧所暗指的法线方位角平行于参考圆。如果不是这种情况,该方位角必须包含高度信息。

GM_Arc::startAngle():Bearing

6.4.15.6 endOfArc (弧段终止方位)

endOfArc 操作计算从弧所在圆的圆心到弧的终点这条直线的方位。在 2 维情况下,这就是一个终止角。在 3 维情况下,由该弧所暗指的法线方位角平行于参考圆。如果不是这种情况,该方位角必须包含高度信息。

GM_Arc::endAngle():Bearing

6.4.16 GM_圆(GM_Circle)

像 GM_Arc 一样,但 GM_Circle 闭合形成一个整圆。其“start”和“end”方位相等并且是所列出的第一个控制点的方位。

注:这仍然至少需要 3 个单独的非共线的点来没有歧义地定义。该弧简单延伸直到遇到第一个点为止。

6.4.17 GM_凸形弧串(GM_ArcStringByBulge)

6.4.17.1 语义

该弧的变量简单存储 GM_Arcs 类的第二个构造函数的参数,并且重新计算该标准弧的其他属性。该 ControlPoint 序列类似于 GM_ArcString,但是中点 GM_Position 不再需要了,因为它可以被计算出来。控制点序列将由每一条弧的始点和终点构成。

6.4.17.2 bulge (凸)

bulge 控制每一条弧段的中点偏移量。该属性是对于法线的实数放大器,该法线确定每一条弧段中点的偏移方向。bulge 序列的长度是比控制点数组的长度少 1。因为一个 bulge 需要控制点数组中的两个相邻点。

GM_ArcByBulge::bulge:Sequence<Real>

该 bulge 不是由距离给出,因为它是对于其法线的一个简单放大。偏移量距离的单位是由在该坐标参照系中的矢量的 length 函数确定的。在本标准的范例中,该法线经常是由欧几里德单位矢量给出,它可以固定也可以不固定其长度,这取决于坐标参照系所用的 distance 公式。

所产生的弧的中点这样给出:

$$\text{midPoint} = ((\text{startPoint} + \text{endPoint}) / 2, 0) + \text{bulge} * \text{normal}$$

6.4.17.3 numArc (弧段数目)

属性“numArc”是在该串中循环弧段的数目。因为插值方法要求重载两个位置的集合,弧段的数目决定了控制点的数目。

GM_ArcStringByBulge::numArc:Integer=((controlPoint.length-1))

6.4.17.4 normal (法向量)

属性“normal”是一个垂直于弧的弦线的矢量,该弦线连接弧的第一个点和下一个点。在 2 维坐标系中,对于法线只有两个可能的方向,经常是作为一个带符号的实数给出,数的大小指示其长度,“+”指示其由弦线向左拐角,“-”指示其从弦线向右拐角。在 3 维中,沿着弧的始点和终点,该法线确定弧所在的平面。

法向量通常是单位矢量,但这并不是绝对必须的。如果法向量是零矢量,该几何对象成为等价于两个端点间的直线。矢量序列的长度与凸序列的长度严格相等,比控制点序列的长度少 1。

GM_ArcByBulge::normal; Sequence<Vector>

注: 派生属性“midPoint(中点)”可以被定义为由 bulge(凸)和 normal(法矢量)属性确定的弧的中点。

GM_ArcByBulge::midPoint; Sequence<GM_Position>

midpoint(n)=(controlPoint(n)+controlPoint(n+1))/2.0+bulge*normal

如果每一个 controlPoint 对与它相关联的 midpoint 一起给出,其结果是对一条 GM_ArcString (它使用 3 点插值方法)控制点的一个有效集合,该 GM_ArcString 几何上等价于 GM_ArcStringByBulge(GM_凸形弧串)。

6.4.17.5 interpolation (插值)

GM_ArcStringByBulge 的 interpolation 属性始终是一个“circularArc2PointWithBulge(两点凸形圆弧)”。

6.4.17.6 GM_ArcStringByBulge (constructor) (GM_凸形弧串 (构造函数))

该构造函数等价于 GM_Arc 的第二个构造函数,不同点在于 bulge 表达保持在对象内部。

GM_ArcByBulge::GM_ArcByBulge(point[2..n];GM_Position,

bulge[1..n];Real,normal[1..n];Vector);GM_ArcStringByBulge

产生弧的中点这样给出:

midPoint(n)=((point(n)+point(n+1))/2.0)+(bulge*normal)

6.4.17.7 asGM_ArcString (作为 GM_弧串)

用 AsGM_ArcString 操作,每一个 GM_ArcStringByBulge 可以被重新投影为基础 GM_ArcString。

GM_ArcStringByBulge::asGM_ArcString();GM_ArcString;

6.4.18 GM_凸形弧(GM_ArcByBulge)

6.4.18.1 语义

GM_ArcByBulge 是在 GM_ArcStringByBulge(见 6.4.17)基础上增加了一个约束。GM_ArcByBulge 实例要求所有的控制点都在同一条圆弧上。

6.4.18.2 GM_ArcByBulge (constructor) (GM_凸形弧 (构造函数))

该构造函数等价于 GM_Arc 的第二个构造函数,不同点在于它还有 bulge 表达。

GM_ArcByBulge::GM_ArcByBulge(point[2];GM_Position,

bulge;Real,normal;Vector);GM_ArcByBulge

产生的弧的中点这样给出:

Midpoint=((startPoint+endPoint)/2.0)+(bulge*normal)

6.4.19 GM_圆锥曲线(GM_Conic)

6.4.19.1 语义

GM_Conic 类型(见图 18)代表任何一般的圆锥曲线。任何圆锥曲线段的曲线可以以极坐标(ρ, φ)的形式表达为:

$$\rho = \frac{P}{1 + e \cos(\varphi)} \text{ 适合于 } -\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2}$$

这里 P 是半-正焦弦(semi-latus rectum), e 是离心率,这给出一个焦点在极(原点)的圆锥曲线,并且在圆锥曲线上的顶点在极轴方向上最接近于焦点, $\varphi=0$ (极坐标下,在 $(\rho, \varphi) = \left(\frac{P}{1+e}, 0\right)$)。当 $e=0$,就是一个圆。当 $0 < e < 1$,就是一个椭圆。当 $e=1$,就是一条抛物线。当 $e > 1$ 就是一个双曲线的分枝。

这些一般的圆锥曲线可以被看作是 2 维笛卡尔参数空间 (u, v) ,由一般的坐标系通过 $u = \rho \cos(\varphi)$ 和 $v = \rho \sin(\varphi)$ 转化而来。通过仿射变换 $(u, v) \rightarrow (x, y, z)$,我们可以将其转化到 3 维坐标参照系,这个仿射变换的定义如下:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u_x & v_x \\ u_y & v_y \\ u_z & v_z \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

这里 φ 作为构造性参数。由 (x_0, y_0, z_0) 给出的 DirectPosition 是来源于局部坐标空间 (u, v) 的映射。

作为另一种选择, 这个极点(origin)可以平移到圆锥曲线的顶点成为:

$$u' = \rho \cos(\varphi) - (P/1+e) \text{ 和 } v' = \rho \sin(\varphi)$$

并且 v 可以被作为构造性参数(见 6.4.7.7 GM_GenericCurve 的定义)。

在通常情况下, 圆锥曲线具有一个小的偏心率 P , 用第一种或“中心”表达。那些具有大的偏心率或大的 P 的情况, 倾向于使用第二种或“线性”表达。

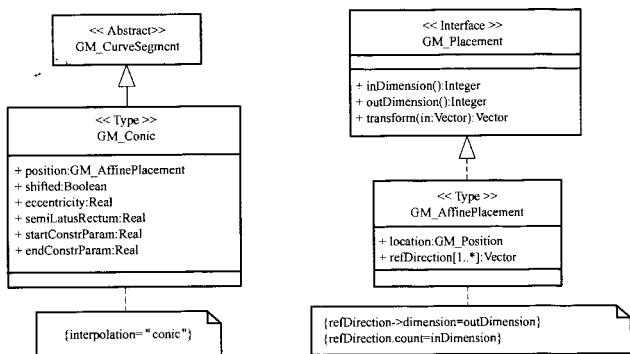


图 18 圆锥曲线与配置(Conics and placements)

6.4.19.2 position (位置)

属性“position”是一个仿射变换对象, 用它将圆锥曲线从参数空间映射到圆锥曲线对应的 GM_Object 所在坐标参照系的坐标空间。这个仿射变换是由 6.4.19.1 的公式给出。

GM_Conic::position: GM_AffinePlacement

6.4.19.3 shifted (已偏移)

当在仿射变换中使用了参数 unshifted (u, v) , 属性“shifted”就是 FALSE; 如果仿射变换中使用了参数 shifted (u', v') , 则属性“shifted”为 TRUE。这用来控制圆锥曲线的焦点或顶点是否还在参数空间的原点。

GM_Conic::shifted: Boolean

6.4.19.4 eccentricity (偏心率)

属性“eccentricity”是用来定义上面方程式中偏心参数“ e ”的值。它控制曲线的形状, 决定这条曲线是圆、椭圆、抛物线还是双曲线。

GM_Conic::eccentricity: Real

6.4.19.5 semiLatusRectum (半正焦弦)

属性“semiLatusRectum”是用在上面方程中的参数“ P ”的值。它控制圆锥曲线焦距的宽度。

GM_Conic::semiLatusRectum: Real

6.4.19.6 startConstrParam, endConstrParam (起始构造参数, 结束构造参数)

“startConstrParam”和“endConstrParam”分别指示在 6.4.19.1 中给出的使用在构造性参数化过程中的参数 startPoint 和 endPoint。

GM_Conic::startConstrParam: Real

GM_Conic::endConstrParam: Real

GM_Conic;

constrParam(startConstrParam)=startPoint();

constrParam(endConstrParam)=endPoint();

并没有假设 startConstrParam 小于 endConstrParam,但是参数化必须是严格单调(严格递增或严格递减)。

6.4.20 GM_配置(GM_Placement)

6.4.20.1 语义

配置操作采用一个标准的几何构造函数,并且把它放置在地理空间中。它定义了从构造性参数空间到所使用的坐标参照系的坐标空间的一个转换。在本标准中,参数空间在 2 维中由 (u, v) 给出,在 3 维中由 (u, v, w) 给出;坐标参照系位置由公式给出,在 2 维为 (x, y) ,3 维为 (x, y, z) 。

6.4.20.2 inDimension (输入维数)

inDimension 操作将返回输入的参数空间的维数。

GM_Placement::inDimension();Integer

6.4.20.3 outDimension (输出维数)

outDimension 操作将返回输出的坐标参照系的维数。

GM_Placement::outDimension();Integer

注:通常, outDimension 大于 inDimension。如果不是这种情况,该配置是有奇异的,可以使用一个更简单维的参数空间的转换来替换它。

6.4.20.4 transform (转换)

“transform”操作映射参数坐标点到输出的笛卡尔坐标空间中的坐标点。

GM_Placement::transform(in:Vector {size=inDimension()});

Vector {size=outDimension()}

6.4.21 GM_仿射配置(GM_AffinePlacement)

6.4.21.1 语义

这些配置是由一个从参数空间到目标坐标空间的线性变换定义的。2 维笛卡尔参数空间 (u, v) , 转换到 3 维坐标参照系空间 (x, y, z) , 使用了一个仿射变换,其定义是:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u_x & v_x \\ u_y & v_y \\ u_z & v_z \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

这里给出这个方程,该 GM_AffinePlacement::location 是原参数空间中的 (u, v) 在目标空间中的直接位置 (x_0, y_0, z_0) 。两个参考方向 (u_x, u_y, u_z) 和 (v_x, v_y, v_z) 是在 (u, v) 原点的单位基础矢量的目标方向。

6.4.21.2 location (定位)

属性“location”给出参数空间原点的目标。这就是上面公式中的矢量 (x_0, y_0, z_0) 。

GM_AffinePlacement::location;GM_Position

6.4.21.3 refDirection (参考方向)

属性“refDirection”给出参数空间的坐标基本矢量的目标方向。它们是上面公式中矩阵的列。所给出的方向的数量是 inDimension, 方向的维数就是 outDimension。

GM_AffinePlacement::refDirection[inDimension];Vector {size=outDimension}

6.4.22 GM_回旋曲线(GM_Clothoid)

6.4.22.1 语义

GM_Clothoid(见图 19)实现了回旋曲线(或角的旋转),它是一个曲率是其长度的固定函数的平面曲线。在适当地选取坐标后,它由 Fresnel(菲涅耳)积分给出:

$$x(t) = \int_0^t \cos\left(\frac{Ar^2}{2}\right) dr, \text{ 和 } y(t) = \int_0^t \sin\left(\frac{Ar^2}{2}\right) dr$$

在参考书[18]中,有关于回旋曲线和回旋曲线分段性质更进一步的描述。

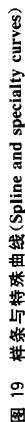


图 19 样条与特殊曲线(Spline and specialty curves)

该几何类主要是用于直线/圆弧或圆弧/圆弧曲线间的相互转换。使用这类曲线类型,可以实现在上文提到的曲线转换中的 C^2 连续的转换。对于回旋曲线的一个公式是 $A^2 = Rt$, 这里 A 是常量, R 是沿着曲线变化的曲率半径, t 是由 Fresnel 积分给出的沿曲线的长度。

6.4.22.2 refLocation (参照定位)

属性“refLocation”是一个仿射映射,它将由 Fresnel(菲涅耳)积分定义的曲线映射到该对象的坐标参照系中。

GM_Clothoid::refLocation; GM_AffinePlacement

6.4.22.3 scaleFactor (比例因子)

属性“scaleFactor”给出上面方程中 A 的值。

GM_Clothoid::scaleFactor; Number

6.4.22.4 startParameter (起始参数)

属性“startParameter”是从作为曲线段始点的拐点(inflection point)开始的弧长距离。在 Fresnel 积分中,它应该比“ t ”小,并且是该曲线段在起始点的构造性参数的值。“startParameter”可以是正值也可以是负值。参数“ t ”作为构造性参数,见 6.4.7.8。

GM_Clothoid::startParameter; Real

注:如果 0.0(零)位于回旋曲线的 startConstrParam(起始控制参数)与 endConstrParam(终止控制参数)之间,当曲线通过回旋曲线的拐点时,由第二个导出矢量给出的曲率半径的方向,随着切矢量的变化,从一侧变到另一侧。对于参数“ t ”的“length”这个词,仅在参数空间中有用,在进行 placement 操作以后,它的长度与坐标参照系中曲线弧长的对应关系是不确定的。

6.4.22.5 endParameter (终止参数)

属性“endParameter”是从作为曲线段终点的拐点开始的弧长距离。在 Fresnel 积分中,它应该比“ t ”大,并且是该曲线段在终止点的构造性参数的值。“startParameter”可以是正值也可以是负值。

GM_Clothoid::endParameter; Real

6.4.23 GM_偏移曲线(GM_OffsetCurve)

6.4.23.1 语义

偏移曲线(offset Curve)是距基曲线(baseCurve)距离恒定的曲线。可用作对于根据定义发生了偏移的曲线而构造的一个简化替换。

6.4.23.2 baseCurve 基曲线

属性“baseCurve”是定义偏移曲线时的一个参考曲线。

GM_OffsetCurve::baseCurve; Reference(GM_Curve)

6.4.23.3 distance (距离)

属性“distance”是偏移曲线距基曲线的距离。在 2 维系统中,正距离是在基曲线的左面,负距离是在基曲线的右面。

GM_OffsetCurve::distance; Length

6.4.23.4 refDirection (参考方向)

属性“refDirection”是用来定义偏移曲线从基曲线偏移的矢量方向。在 2 维的情况下,可以忽略,因为可以用偏移量的正或负来表达。在这种情况下,相对于基曲线的切线,距离定义为左边(正距离)或右边(负距离)。

在 3 维,基曲线在每一点都有一个定义好的切线方向。偏移曲线在 3 维中任何一点,基曲线都有一个定义好的切线方向。偏移曲线在基曲线“ c ”的任何点(参数)都在这个方向上。

$$\vec{s} = \vec{v} \times \vec{t} \quad \text{式中} \quad \vec{v} = c.\text{refDirection}(), \vec{t} = c.\text{tangent}()$$

对于已经定义好的偏移方向,在曲线的任何点上,方向 \vec{v} 与方向 \vec{t} 不是相同就是相反。

GM_OffsetCurve::refDirection; Vector

RefDirection 的缺省值是局部坐标的高程轴方向,在地理意义上指示向上。

注:如果 refDirection 对于局部高程轴(指向上的) tangent 是正值,当从上往下看时,偏移矢量指向曲线的左方。

6.4.24 GM_节点(GM_Knot)

6.4.24.1 语义

GM_Knot 用来控制样条曲线和曲面的构造性空间参数。每一个节点序列用作参数空间的一个维,这样,在样条曲面中,有两个节点序列,一个序列对应于参数 (u, v) 中的 u ,另一个对应 v 。当节点原始序列是 (u_i) 和 (v_j) 时,第 i 行,第 j 列的节点将对应于 (u_i, v_j) 。样条曲线或曲面的每一个节点都是用—个 GM_Knot 来描述的。

6.4.24.2 value(值)

属性“value”是样条节点参数的值。节点序列是一个非描述性的序列,即在该序列中每个节点的值等于或大于原来节点的值。相等连续节点常用在处理多重性中。

GM_Knot::value:Real

6.4.24.3 multiplicity(多重性)

属性“multiplicity”是指用于样条(具有相同的权重)定义中的节点的多重性。

GM_Knot::multiplicity:Integer

6.4.24.4 weight(权重)

属性“weight”是样条上该节点的平均权重。

GM_Knot::weight:Real

6.4.25 GM_节点类型(GM_KnotType)

当且仅当所有节点是单重(multiplicity:1),并且它们与前一个节点相差一个正的常量,B样条(B-spline)才是均匀的(uniform)。当且仅当节点在端点处是多重(multiplicity:degree+1),在别处是单重(multiplicity:1),并且它们与前一个节点相差一个正的常量,这个B样条才是准均匀的(quasi-uniform)。该代码列表用来描述节点在各种样条的参数空间的分布。可能的值是:

——uniform(均匀):节点的形式适合于均匀B样条曲线。

——quasiUniform(准均匀):节点的形式适合于准均匀B样条曲线。

——piecewiseBezier(分段贝塞尔):节点的形式适合于分段贝塞尔曲线。

GM_KnotType::

uniform

quasiUniform

piecewiseBezier

6.4.26 GM_样条曲线(GM_SplineCurve)

6.4.26.1 语义

GM_SplineCurve(见图19)作为使用多种版本的样条(多项式或有理函数)的GM_CurveSegment子类的根。

6.4.26.2 knot(节点)

属性“knot”是用来定义样条基函数的各种节点的序列。回调节点数据类型保持节点的多重性信息。

GM_SplineCurve::knot:Sequence(GM_Knot)

6.4.26.3 degree(次数)

属性“degree”是用于GM_PolynomialSpline插值的多项式的次数。

GM_SplineCurve::degree:Integer

6.4.26.4 controlPoints(控制点)

属性“controlPoints”是用于该GM_SplineCurve插值的点的数组。

GM_SplineCurve::controlPoints:GM_PointArray

6.4.27 GM_多项式样条(GM_PolynomialSpline)

6.4.27.1 语义

用分段的 n 次多项式来定义 n 次多项式样条,定义多项式变化的控制点可以达到 C^{n-1} 连续。连续

的级别是由 numDerivativesInterior(内部可微次数)属性控制的。参数将包括方向,方向为该线段的始点与终点间的多项式“次数-2”阶导数。GM_Linestring 等价于一次多项式样条。在控制点处具有简单的连续性(C^0),但是不需要导数信息(次数-2=-1)。

注:多项式样条和 B 样条(基样条)之间的主要差别是多项式样条通过它们的控制点,使得控制点与采样点一致。

6.4.27.2 Interpolation (插值)

GM_PolynomialSpline 的插值方法是“polynomialSpline(多项式样条)”。

GM_PolynomialSpline::

interpolation:GM_InterpolationMethod=“polynomialSpline”

6.4.27.3 vectorAtStart (初矢)

属性“vectorAtStart”是用在 GM_PolynomialSpline 样条函数插值中的始点的初导数的值(最高到:次数-2)。

GM_PolynomialSpline::vectorAtStart:Sequence<Vector> {size=degree-2}

6.4.27.4 vectorAtEnd (末矢)

属性“vectorAtEnd”是用在 GM_PolynomialSpline 插值样条函数中的终点的末导数的值(最高到:次数-2)。

GM_PolynomialSpline::vectorAtEnd:Sequence<Vector> {size=degree-2}

6.4.28 GM_三次样条(GM_CubicSpline)

三次样条在它们是分段序列这一点上类似于线串,但每一个段有它们自己定义的函数。三次样条使用控制点和一个导数集合来定义一个分段 3 次多项式插值。虽然仍然是多项式,但不像线串插值,并不必须以弧长作参数。在本标准中,样条有两个参数化方式,一个为定义的(构造性参数)和另一个采用弧长的参数化方法,以满足 GM_GenericCurve(GM_一般曲线)的需要。

描述三次样条的方程必须是 C^2 连续的,即在所有点具有连续的一阶与二阶导数,并且依次通过控制点。在控制点间,曲线段定义为三次多项式。在每一个控制点,多项式的一阶与二阶导数在控制点的两侧相等。控制参数记录中必须包含 vectorAtStart 和 vectorAtEnd,它们是在 controlPoint[1] 和 controlPoint[n] 的单位矢量, n 为控制点个数。

对于“vectorAtStart”和“vectorAtEnd”的限制简化这个序列到一个单独的切线矢量序列。

GM_CubicSpline::vectorAtStart:Vector//“degree-2”是 1

GM_CubicSpline::vectorAtEnd:Vector//“degree-2”是 1

注:实际应用的三次多项式是变化的,但是这样产生的曲线保证是唯一的。见参考资料[2]、[10]、[12]、[23]和[24]的插值实例。

对于 GM_CubicSpline 的插值方法是“cubicSpline”。

GM_CubicSpline::interpolation:GM_InterpolationMethod=“cubicSpline”

GM_CubicSpline 的次数是“3”。

GM_CubicSpline::degree:Integer=“3”

6.4.29 GM_样条曲线形式(GM_SplineCurveForm)

这个代码序列用来指示可能用来逼近一个特定的 B 样条的曲线类型。可能的值有:

- polyline form(多线):由一阶 B 样条表达的线段连接序列(线串)。
- circular Arc(圆弧):一段圆弧或一个完整的圆。
- elliptic Arc(椭圆弧):一段椭圆弧或一个完整的椭圆。
- parabolic Arc(抛物线弧):一段有限长度的抛物线。
- hyperbolic Arc(双曲线弧):一段有限长度的双曲线的一枝。

GM_SplineCurveForm::

polylineForm

circularArc

ellipticalArc

parabolicArc

hyperbolicArc

6.4.30 GM_B 样条曲线 (GM_BSplineCurve)

6.4.30.1 语义

B 样条(见图 19)是由控制点和基函数描述的分段参数化多项式或有理曲线。如果在节点处的权重相等,它就是一条多项式样条。如果不等,它就是一条有理函数样条。如果布尔值“isPolynomial”被设为 TRUE,则权重就被设为“1”。一条 B 样条曲线,如果它是准均匀的,除非它内部节点 multiplicity (重数)为“degree”而不是 1,那么这条 B 样条曲线就是一条分段的贝塞尔曲线。在该子类型(subtype)中,节点间隔是 1.0,从 0.0 开始。一条只有两个节点 0.0 和 1.0 的分段的贝塞尔曲线,每一个节点的重数为“degree+1”,等价于一条简单贝塞尔曲线。

6.4.30.2 degree (次数)

属性“degree”是基函数的次数。

GM_BSplineCurve::degree: Integer

6.4.30.3 curveForm (曲线形式)

属性“curveForm”是用来确定曲线的特定类型,即接近于哪种样条曲线。它只用于查询,用以捕获原始用意。如果没有打算使用样条逼近,该属性的值是 NULL。

GM_BSplineCurve::curveForm: GM_SplineCurveForm

6.4.30.4 knotSpec (节点类型)

“knotSpec”属性给出在定义这条样条曲线时节点分布的类型。它只是用于查询,它是根据不同的构造函数设定的。

GM_BSplineCurve::knotSpec[0,1]: GM_KnotType

6.4.30.5 isPolynomial (是多项式)

如果这条样条曲线是多项式样条函数,则“isPolynomial”这个属性为 TRUE。

GM_BSplineCurve::isPolynomial: Boolean

6.4.30.6 GM_BSplineCurve (constructor) (GM_B 样条曲线 (构造函数))

GM_BSplineCurve 类的构造函数使用上面描述的相关属性信息构造一条 B 样条曲线。如果 knotSpec 不存在, knotType(节点类型)就是均匀的,并且节点是均匀间隔,只有第一个点和最后一个点具有 multiplicity=1,在其他节点处, multiplicity=degree+1。如果 knotType 是均匀的,它们不需要专门指定。

GM_BSplineCurve::GM_BSplineCurve(deg: Integer, pts: GM_PointArray,

k [0,1]: Sequence<GM_Knot>, ks [0,1]: GM_KnotType): GM_BSplineCurve

注: 如果 B 样条曲线是均匀的,并且 degree=1,则该 B 样条等价于 polyline(GM_LineString)。如果 knotType 是“piecewise Bézier(分段贝塞尔)”,则节点缺省地为均匀间隔,并且除去首、尾节点外的中间节点的 multiplicity 为 degree,首、尾节点的 multiplicity=degree+1。

6.4.31 GM_贝塞尔曲线 (GM_Bezier)

GM_Bezier 是在插值过程中使用 Bézier 或 Bernstein(伯恩斯坦)多项式的多项式样条。一个有 n 个元素的控制点数组将创建一个 n 段多项式曲线,这里 n 定义了整个曲线的段数。这些曲线由称作 Bézier 或 Bernstein 多项式基函数的集合定义。Bézier 或 Bernstein 基函数的形式如下:

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad \text{这里} \quad \binom{n}{i} = \frac{n!}{i!(n-i)!} \quad \text{对于} \quad i = 0, 1, 2, \dots, n$$

这“ $n+1$ ”个控制点 P_0, P_1, \dots, P_n 的集合将确定一条如下面给出的曲线段:

$$\vec{c}(t) = \sum_{i=0}^n P_i J_{n,i}(t) \quad \text{其} \quad t \in [0, 1].$$

该曲线段的采样点是曲线的值,该曲线是由每一个多项式的最大值(i/n)定义的:

$$S_i = \vec{c}\left(\frac{i}{n}\right), \text{式中} \quad i = 0, 1, 2, \dots, n$$

注: 对于 $n=1$, 这两个权重函数如下:

$$J_{1,0}(t) = \binom{1}{0} t^0 (1-t)^1 = (1-t) \quad \text{和} \quad J_{1,1}(t) = \binom{1}{1} t^1 (1-t)^{1-1} = t$$

当给出 P_0 和 P_1 , 曲线段成为:

$$\vec{c}(t) = (1-t)P_0 + tP_1 \quad \text{其中} \quad t \in [0, 1].$$

换句话说, 对于 $n=1$, Bézier 多项式在几何上等价于一条简单线段。

6.4.32 GM_曲面插值(GM_SurfaceInterpolation)

GM_SurfaceInterpolation(见图 20)是一个代码列表,它可用来标示由应用模式定义的插值机制。

对于“interpolation(插值)”的有效值包括但不限于如下内容:

- none(无): 曲面的内部没有定义。这假设曲面跟随由坐标参照系定义的参考曲面。
- planar(平面): 插值方法将返回的点都在一个平面上。在这种情况下, 边界将被限制在该平面内。
- spherical(球面), Elliptical(椭球面), Conic(圆锥曲面): 该曲面是一个球面、椭球面或圆锥曲面的一部分。
- tin(不规则三角网): 控制点组织成相邻的三角形, 这些三角形形成小的平面片。
- parametricCurve(参数曲面): 控制点组织成 2 维格网, 在格网内的每一个单元是由一个曲线族定义的蒙皮曲面。
- polynomialSpline(多项式样条曲面): 控制点组织成一个不规则的 2 维格网, 格网内的每一个单元都是由多项式样条函数生成。
- rationalSpline(有理样条曲面): 控制点组织成一个不规则的 2 维格网, 格网内的每一个单元是由一个有理样条(多项式商)函数生成。
- triangulatedSpline(三角样条曲面): 控制点组织成相邻的三角形, 每一个三角形是由一个多项式样条函数生成。

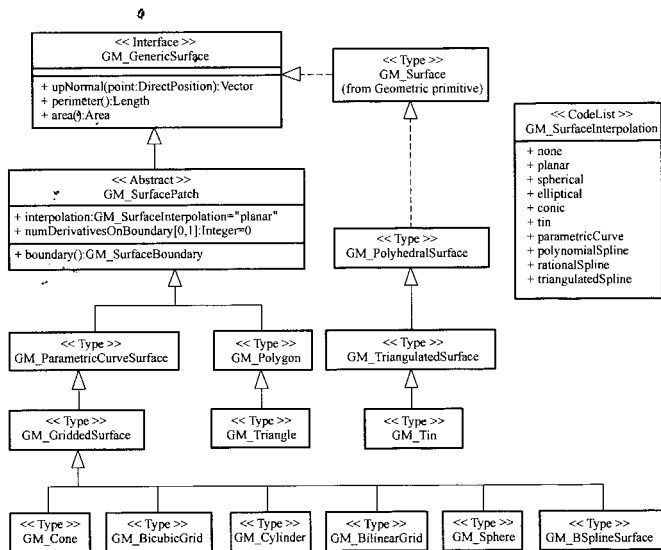
如果这个方法使用了多于一种的插值类型, 将有更多的限制。

GM_SurfaceInterpolation::

```

none
planar
spherical
elliptical
conic
tin
parametricCurve
polynomialSpline
rationalSpline
triangulatedSpline

```



6.4.33 GM_一般曲面 (GM_GenericSurface)

6.4.33.1 语义

GM_Surface 和 GM_SurfacePatch (GM_曲面片) 两者都是描述曲面几何的片段, 因此分享很多操作符号 (operation signature)。它们定义在 GM_GenericSurface 类的接口里 (见图 20)。

6.4.33.2 upNormal (外法向)

“upNormal”操作返回一个在所传递的 DirectPosition 处垂直于 GM_GenericSurface 的矢量, 该 DirectPosition 必然是在 GM_GenericSurface 上。

GM_GenericSurface::upNormal(point:DirectPosition):Vector

外法向始终应与边界一致的方式指向上 (upward)。这意味着当从曲面的外法向这一面来看曲面时 (例如从上往下看 2 维多边形的边界), 曲面的外边界是逆时针方向, 内边界是顺时针方向, 由外法向指示的这一面被称作曲面的“顶 (top)”。外法向函数应该是连续的, 并且法向长度始终是 1.0。

注: 外法向沿着体的边界运动将始终指向离开体的方向。在处理体的空洞时, 在语义上稍微有点问题, 在孔洞处外法向 (为了数学一致性的缘故) 指向空洞的中心, 这在语义上可能被认为是空洞的内部。混淆的地方在于, 在大多数语言中的语义隐喻“体的内部”和“容器的内部”使用“向里 (inward)”与拓扑的观点不一致。“在 (in)”岩石中的孔洞并不在岩石内部, 这里的岩石内部是指岩石的物质以相同方式组成的固体材料。咖啡“在”杯子里, 不同于“在”构成杯子的陶瓷玻璃里。使用这个文化派生出来的“向里”与“在”这两个词隐喻可能在所有语言中都是不一致的, 有的语言可能对于这两种不同的概念使用不同的前置词。在本标准中使用从数学 (拓扑) 中派生的语义中立的“内部 (interior)”概念。

6.4.33.3 perimeter (周长)

“perimeter”操作将返回该 GM_GenericSurface 的边界成分的所有长度的总和。由于周长像长度一样, 是距离的累积 (积分), 它的返回值将是在度量距离所使用的参照系中。

GM_GenericSurface::perimeter():Length

注: 周长定义为所有边界成分的长度的总和。一条曲线或一组曲线的组合的长度始终是正值并且非零 (除非曲线是病态的)。这意味着曲线上的孔洞对于曲面的总周长将有正的贡献。

6.4.33.4 area (面积)

2 维几何对象的面积是对于它的曲面面积(以距离的平方单位)的一个数值度量。因为面积是一个积累(积分),它的返回值将具有一个为度量距离的平方的度量单位,例如平方米(m^2)。“area”这个操作将返回 GM_GenericSurface 的面积。

GM_GenericSurface::area(); Area

该返回值与坐标参照系和曲面的形状有关。

注:为了与曲面定义为 DirectPositions 的集合相一致,表面上的孔洞对于总面积没有贡献。如果使用通常的 Green (格林)定理或更一般的 Stokes(斯托克)积分定理,绕着曲面上洞的积分要从绕着曲面片的外环的积分中减去。

6.4.34 GM_曲面片(GM_SurfacePatch)

6.4.34.1 语义

GM_SurfacePatch(见图 20)定义一个 GM_Surface 的同类部分。相关联的“Segmentation(分片)”的 multiplicity(多重性)属性(见图 12)定义每一个 GM_SurfacePatch 只能属于一个 GM_Surface,而不能为多个 GM_Surface 所拥有。

6.4.34.2 interpolation (插值)

属性“interpolation”确定用在 GM_SurfacePatch 上曲面的插值机制。该机制使用定义在各个子类中的控制点和控制参数来确定该 GM_SurfacePatch 的位置。

GM_SurfacePatch::Interpolation; GM_SurfaceInterpolation

6.4.34.3 numDerivativesOnBoundary (边界可微次数)

属性序列“numDerivativesOnBoundary”定义在曲面以及它与它共享边界曲线的方式直接相邻的曲面片之间的连续类型。对于该面片,值的序列与在 GM_SurfaceBoundary 中由 GM_GenericCurve::boundary 返回的 GM_Rings 相对应。缺省值“0”意味着简单连续,即强制性最低水平的连接。在数学教科书中,这个水平被叫做“C”连续。值 1 意味着函数在端点处连续并可微;即“C¹”连续。对于任何整数“n”意味着 n 次可微,即“Cⁿ”连续。

GM_SurfacePatch::numDerivativesOnBoundary[0..1]; Integer

6.4.34.4 boundary (边界)

“boundary”操作返回该 GM_SurfacePatch 的边界,该边界表达为由 GM_OrientableCurves 组成的 GM_Rings 的一个组合。

GM_SurfacePatch::boundary(); GM_SurfaceBoundary

注:该操作的语义相同于 GM_Surface::boundary 操作的语义,但这里使用的曲线可能是非持久的 GM_OrientableCurve。临时性的 GM_Curve 数据类型值也是有效的。在通常情况下,GM_SurfacePatches 将与聚合的 GM_Surface 以及具有 GM_SurfacePatches(并不明显需要)的其他部分共享它们的部分边界。在附录 C 中, solid(体)示例(C.1.3.1)使用了一个单独的曲面片折回到它自己形成一个拓扑的圆柱面,再加上两个底面形成一个体的边界。在这种情况下,第一个曲面片与另外两个底面中的每一个共享一个边界线段,还有一个与它自己共享的边界(即曲面片折回形成圆筒时的连接带)。

6.4.35 GM_多面体表面(GM_PolyhedralSurface)

6.4.35.1 语义

GM_PolyhedralSurface(见图 21)是一个 GM_Surface,它由若干 GM_Polygon(GM_多边形)构成并沿着它们公共的边界曲线相连接。它仅在可接受的曲面片类型限制上不同于 GM_Surface。

6.4.35.2 GM_PolyhedralSurface (constructor) (GM_多面体表面 (构造函数))

GM_PolyhedralSurface 构造函数用若干 GM_Polygon 小面来构造所需的聚合曲面。

GM_PolyhedralSurface::GM_PolyhedralSurface(tiles[1..n]; GM_Polygon);

GM_PolyhedralSurface

6.4.35.3 patch (面片)

关联角色“patch”将曲面与构成它的各个多边形小面片关联起来。它应该是非空的。

GM_PolyhedralSurface::patch[0..n]; Reference<GM_Polygon>

6.4.36 GM_多边形(GM_Polygon)

6.4.36.1 语义

GM_Polygon(见图 21)是一个曲面片,它由边界曲线的集合和一个将这些曲线粘在一起的一个潜在的曲面所定义。缺省状态这些曲线是共面的,并且在它内部使用平面插值。

6.4.36.2 boundary (边界)

属性“boundary”存储作为 GM_Polygon 边界的 GM_SurfaceBoundary。

GM_Polygon::boundary:GM_SurfaceBoundary

注:曲面片的边界与作为包含 GM_Surface 的 GM_Complex 的边界不必相同。包含在 GM_Surface 内部的曲线(作为两个相邻曲面片的公共边界)不是任何包含 GM_Surface 的 GM_Complex 的一部分,它们纯粹是构造性的,不应该将其作为定义该 GM_Complex 的 GM_Surface 与 GM_Curve 间连接性的拓扑关系。

6.4.36.3 spanningSurface (生成曲面)

可选的 spanningSurface 提供一种生成多边形内部的机制

GM_Polygon::spanningSurface[0,1]:GM_Surface

注:生成曲面应当没有与多边形边界相交的边界成分,并且对于该多边形,不应被混淆为被边界曲线定义的曲面的一部分。虽然 Tin 和 格网曲面也时常用作生成曲面,但最常见的生成曲面还是高程模型,在本标准中不直接描述它。

6.4.36.4 GM_Polygon (constructor) (GM_多边形 (构造函数))

GM_Polygon 构造函数的第一种形式直接由(组织成一个 GM_SurfaceBoundary 的)边界曲线的集合来创建 GM_Polygon,这些边界曲面由这些共面的 GM_Position 作为控制点来定义。

GM_Polygon::GM_Polygon(boundary:GM_SurfaceBoundary):GM_Polygon

注:GM_SurfaceBoundary 的外部与所构造的平坦多边形在同一个平面。

GM_Polygon 构造函数的第二种形式的第二个变量提供用来创建 GM_Polygon 的生成曲面。对于用在 GM_SurfaceBoundary 中的组合曲线所使用的插值类型没有限制,但是在接下来的处理中它们必须依赖于“生成曲面”。

GM_Polygon(boundary:GM_SurfaceBoundary,spanSurf:GM_Surface):GM_Polygon

注:边界成分是有向的这一属性对于完成这项任务很重要。在有界的流形(例如球)内经常存在含混,除非适当地使用定向(orientation)属性。

6.4.37 GM_三角化曲面(GM_TriangulatedSurface)

GM_TriangulatedSurface(见图 21)是仅由 GM_Triangle 构成的 GM_PolyhedralSurface。对于三角形如何派生而来没有限制。

6.4.38 GM_三角形(GM_Triangle)

GM_Triangle 是由三个角点定义的平面 GM_Polygon;GM_Triangle 应当由下面这种形式的构造函数产生。

GM_Polygon(GM_LineString(<P1,P2,P3,P1>>))

这里,P1,P2,P3 是三个 GM_Position。GM_Triangle 没有洞。GM_Triangle 应当用来构造 GM_TriangulatedSurfaces。

注:在三角形中的点可以根据三角形的角点通过定义一套重心坐标来定位,三个非负数值 c_1, c_2, c_3 , 令 $c_1 + c_2 + c_3 = 1.0$, 则三角形中的每一个点可以用这套重心坐标表为:

$$P = c_1 P_1 + c_2 P_2 + c_3 P_3$$

6.4.39 GM_不规则三角网(GM_Tin)

6.4.39.1 语义

GM_Tin(见图 21)是一个使用 Delaunay(德洛内)算法或类似算法构成的 GM_TriangulatedSurface,这里类似的算法是指考虑了阻断线、停止线和三角边的最大长度的算法(见图 22)。这些网络满足没有修正的 Delaunay 准则;网络中的每一个三角形,通过三角形顶点的圆的内部不包括其他三角形的顶点。

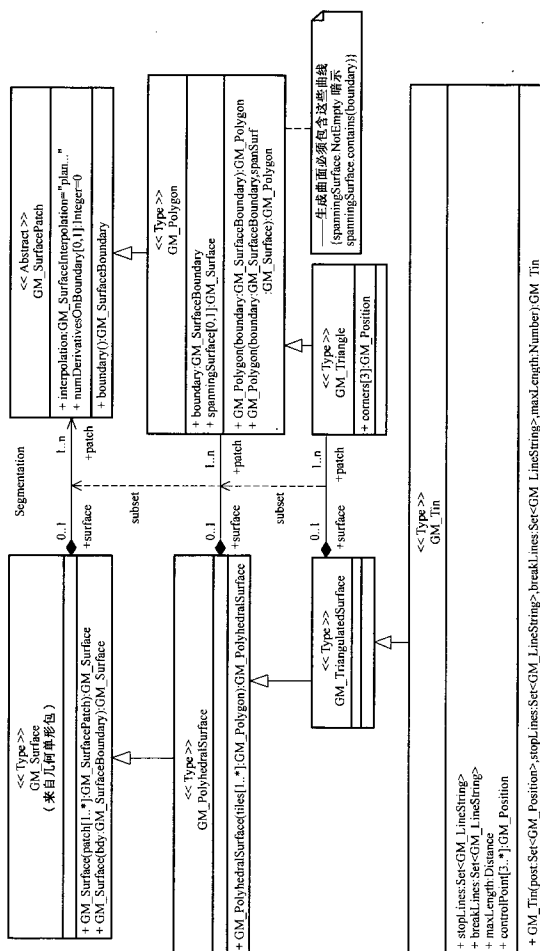


图 21 多边形的曲面(Polygonal surface)

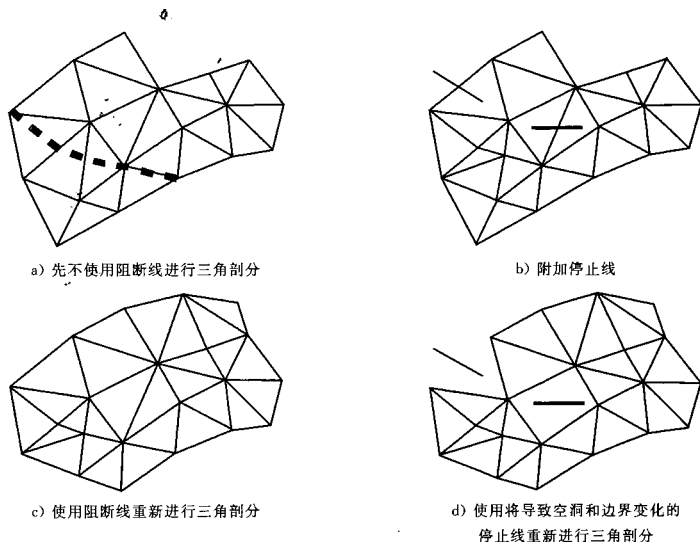


图 22 TIN 的构造(TIN construction)

6.4.39.2 stopLines (停止线)

stoplines 是曲面的局部连续性或正则性有问题的线。在这种有问题的区域,相交于一个 stopline 的三角形应当从 TIN 曲面上去掉,在曲面上留下空洞。如果这种情况出现在曲面边界的三角形上,将导致曲面边界的改变。“stopLines”属性包含所有这些有问题的线段作为一个线串集。

GM_Tin::stopLines:Set(GM_LineString)

6.4.39.3 breakLines (阻断线)

breaklines 是对曲面的形状有控制作用的线,代表表面上的局部山脊或沟谷(如汇水线)。因为这些成分的线段必须包含在 TIN 中,尽管它们有违于 Delaunay 准则。“breakLines”属性包含这些评判线段作为一个线串集。

GM_Tin::breakLines:Set(GM_LineString)

6.4.39.4 maxLength (最大长度)

对于数据密度不能保证合理计算的这些区域,应当通过增加一个基于三角形边长的“保持标准”来消去这些区域。对于超过最大长度的任何三角形边,则应调整三角形的划分,以消除这些长边。

GM_Tin::maxLength:Distance

6.4.39.5 controlPoint (控制点)

在 TIN 中三角形的角点被称为桩(post)。属性“controlPoint”包含用在这个 TIN 中的角点的这些 GM_Position 的一个集合。因为每一个 TIN 都包含若干三角形,至少有三个 post。这些点给出的次序并不影响它们所表达的曲面。应用模式可以基于控制点的次序添加信息,以利于从控制点重构这个 TIN。

GM_Tin::controlPoint[3..n]:GM_Position

注: TIN 的控制点常被称为“桩(post)”。

6.4.39.6 GM_Tin (constructor) (GM_不规则三角网 (构造函数))

对于受限 Delaunay 网的构造函数需要若干三角形角点(桩)、若干阻断线、若干停止线和三角形边的最大长度。

GM_Tin::GM_Tin(post:Set(GM_Position),stopLines:Set(GM_LineString),
breakLines:Set(GM_LineString),maxLength:Number);GM_Tin

6.4.40 GM_参数曲线族曲面(GM_ParametricCurveSurface)

6.4.40.1 语义

构成参数曲线族曲面 GM_ParametricCurveSurface(见图 23)的曲面片是由连续的曲线族参数化而产生的,由下面的构造函数给出。

$$surface(s,t):[a,b]\times[c,d]\rightarrow DirectPosition$$

通过固定这两个参数中一个参数的值,我们就可以得到一个单参数的曲线族。

$$c_i(s)=c_i(t)=surface(s,t)$$

GM_ParametricCurveSurface(见图 23)的功能是对外界暴露两个曲线族。第一个为“horizontal(水平)”横切面 $c_i(s)$,第二个为“vertical(垂直)”纵切面 $c_i(t)$ 。“horizontal”和“vertical”这两个术语分别指参数空间,这并不需要曲线在坐标参照系下是“水平”或“垂直”。表 7 列出了这些曲面的一些可能的类型配对(对这些曲面可能还有其他表达方法)。曲面参数化的两个偏导数 i 和 j 由下面给出:

$$i \equiv \frac{dsurface}{ds} = \frac{d}{ds}c_i(s) = \frac{\partial}{\partial s}surface(s,t)$$

和

$$j \equiv \frac{dsurface}{dt} = \frac{d}{dt}c_i(t) = \frac{\partial}{\partial t}surface(s,t)$$

当曲面的两个偏导数矢量非零时,曲面缺省的外法向是这两个偏导数矢量的叉积:

$$k=i\times j$$

如果坐标参照系是 2 维,则这个矢量 k 根据所提供的“向上”的高程矢量扩展局部坐标系。在这种情况下,基矢量 (i,j) 必须是一个右手系统,这就是说,从 i 到 j 的方位角必须小于 180° 。由 $\langle i,j \rangle$ 给出一个局部坐标轴的右手“移动框架”。移动框架被定义为从该几何对象到该对象所在的局部切空间的一个基的连续函数。对于曲线,这就是曲线的导数,即局部切线。对于曲面,这就是一个局部切矢对。参数化曲面有一个自然的移动框架,在本章被用来定义曲面的外法向。

注:一个可见的移动框架的存在是“可定向”流形的定义。这就是为什么一个连续的外法向的存在意味着该曲面是可定向的。不可定向的曲面,例如牟比乌斯带(Möbius band)和克莱因瓶(Klein bottle),是违反直觉的。

6.3.17.1 禁止将它们用在与本标准一致的应用模式中。克莱因瓶在 3 维中甚至构造不出来,而需要 4 维空间的非奇异表达。

表 7 各种类型的参数曲线族曲面

曲面类型	水平曲线类型	垂直曲线类型
GM_Cylinder(GM_圆柱面)	圆,常数半径	线段
GM_Cone(GM_圆锥曲面)	圆,衰减半径	线段
GM_Sphere(GM_球面)	具有常数纬度的圆	具有常数经度的圆
GM_BilinearGrid(GM_双线性格网曲面)	线串	线串
GM_BicubicGrid(GM_双三次格网曲面)	三次样条	三次样条

6.4.40.2 horizontalCurveType(水平曲线类型)

属性“horizontalCurveType”指示以水平的方式穿越该曲面,与参数“s”相对应。

GM_ParametricCurveSurface::horizontalCurveType;GM_CurveInterpolation

6.4.40.3 verticalCurveType(垂直曲线类型)

属性“verticalCurveType”指示以垂直的方式穿越该曲面,与参数“t”相对应。

GM_ParametricCurveSurface::verticalCurveType;GM_CurveInterpolation

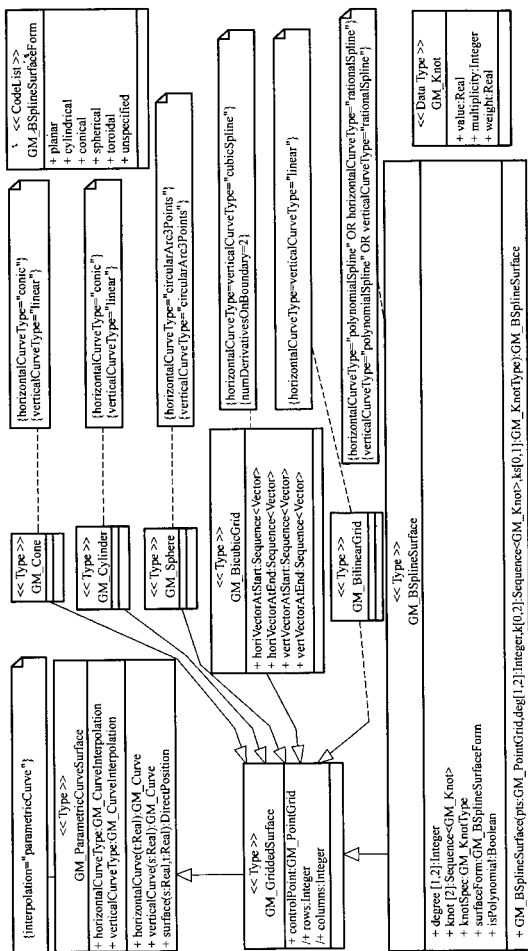


图 23 GM_参数曲线族曲面和它的子类 (GM_ParametricCurveSurface and its subtypes)

6.4.40.4 horizontalCurve (水平曲线)

操作“horizontalCurve”构造一条曲线,这条曲线水平地穿越该曲面,与参数“s”相一致。该曲线的参数“t”是常数。

GM_ParametricCurveSurface::horizontalCurve(t:Real);GM_Curve

注:通过这个操作或相应的垂直曲线函数操作返回的 GM_Curve 通常不是任何包含该曲面的 GM_Complex 的一部分。在极端的参数空间,可能出现例外。支持这种曲面的参数空间通常映射到目标曲面的边界。

6.4.40.5 verticalCurve (垂直曲线)

“verticalCurve”这个操作构造一条曲线,这条曲线垂直地穿越该曲面,与参数“t”相一致。该曲线的参数“s”是常量。

GM_ParametricCurveSurface::verticalCurve(s:Real);GM_Curve

6.4.40.6 surface (曲面化)

操作“surface”在垂直和水平两个方向上穿越曲面。

GM_ParametricCurveSurface::surface(s:Real,t:Real);DirectPosition

6.4.41 GM_格网化曲面(GM_GriddedSurface)

6.4.41.1 语义

GM_GriddedSurface(见图 23)是在参数空间由矩形网格定义的一张 GM_ParametricCurveSurface。该网格的行是曲面的水平曲面线的控制点,列是曲面的垂直曲面线的控制点。这里假设对于每一对参数坐标(s,t),水平曲面线对于每一个整数偏移量是根据“s”来求值的。这就定义了一个控制点序列:

$\langle c_n(s); s=1 \dots \text{columns} \rangle$

从这个序列,垂直曲线计算“s”,并且以“t”求值。在大多数情况下,计算的次序(“水平—垂直”与“垂直—水平”)并没有差别。在使用时,首先使用“水平—垂直”次序。

注:大多数情况下一个格网曲面是一个 2 维样条曲面。在这种情况下,对于每一个参数的权重函数使得计算的次序无关紧要。

$$\text{surface}(s,t) = \sum_{i=0}^{\text{rows}-1} \sum_{j=0}^{\text{columns}-1} w_i^t(s) w_j^t(t) \bar{P}_{i,j}$$

这里 $\bar{P}_{i,j}$ 是在第 i 行 j 列的控制点。

逻辑上,任何成对的曲线插值类型都可以形成一个 GM_GriddedSurface 的子类。下面定义了一些可以以这种方式表达的最常用的曲面。

6.4.41.2 controlPoint (控制点)

这是以行为主要形式给出的控制点双索引序列。

GM_GriddedSurface::controlPoint:GM_PointGrid;

注:对于格网曲面的形态并没有作任何假设。例如,这个位置并不需要作用于一个“2.5 维”的曲面,在任何或它们所有的纵坐标上这些连贯的点可以相等。进一步,曲线在这两个方向中的一个或两个方向上可以是闭合的。

6.4.41.3 rows (行数)

导出属性“rows”给出该参数格网的行的数目。

GM_GriddedSurface::rows:Integer=controlPoint→row.count;Integer

6.4.41.4 columns (列数)

导出属性“columns”给出该参数格网的列的数目。

GM_GriddedSurface::rows:Integer=controlPoint→row→column.count;
Integer

6.4.42 GM_圆锥曲面(GM_Cone)

GM_Cone 是由圆锥曲面片段族给出的 GM_GriddedSurface(GM_格网化曲面),这些圆锥曲面片

段的控制点是线性变化的。

注：所有定义位置都相同的一个五点椭圆是一个点。这样一个被截取的椭圆圆锥曲面可以由一个 2×5 的控制点集合 $\langle (P1, P1, P1, P1, P1), (P2, P3, P4, P5, P6) \rangle$ 给出。P1 是该圆锥曲面的顶点，P2、P3、P4、P5、P6 是在该圆锥曲面的基椭圆上的任何五个不同的点。如果该水平曲线是圆而不是椭圆，则一个圆锥曲面可以由 $\langle (P1, P1, P1), (P2, P3, P4) \rangle$ 来构造。

6.4.43 GM_圆柱面(GM_Cylinder)

GM_Cylinder 是一个由圆族构成的 GM_GriddedSurface，这些圆族的位置沿着一个平行线的集合变化，保持水平横切面的形态不变。

注：这里与前面的假设一样，一个 GM_Cylinder 可以由两个圆给出，由 $\langle (P1, P2, P3), (P4, P5, P6) \rangle$ 给出控制点。

6.4.44 GM_球面(GM_Sphere)

GM_Sphere 是一个由圆族构成的 GM_GriddedSurface，这些圆的位置沿着该球的轴呈线性变化，而圆的半径与中心角的余弦函数成比例。水平圆曲线类似于不变的纬线，垂直弧线类似于不变的经线。

注：如果控制点是根据增加的经度和纬度排序的，球面的外法向是一个向外的法向量。

例如：如果我们采用经纬度 (u, v) 格网集合，例如：

$(-90, -180)$	$(-90, -90)$	$(-90, 0)$	$(-90, 90)$	$(-90, 180)$
$(-45, -180)$	$(-45, -90)$	$(-45, 0)$	$(-45, 90)$	$(-45, 180)$
$(0, -180)$	$(0, -90)$	$(0, 0)$	$(0, 90)$	$(0, 180)$
$(45, -180)$	$(45, -90)$	$(45, 0)$	$(45, 90)$	$(45, 180)$
$(90, -180)$	$(90, -90)$	$(90, 0)$	$(90, 90)$	$(90, 180)$

然后使用普通的方程(这里 R 是球的半径)投影这些点到 3 维空间中去。

$$\begin{aligned} z &= R \sin u \\ x &= (R \cos u) (\sin v) \\ y &= (R \cos u) (\cos v) \end{aligned}$$

我们就得到一个半径 R 、圆心 $(0, 0)$ 的球，其表面为一个格网化的曲面。注意，所有的第一行和最后一行的控制点分别影射到 3 维欧几里德空间中的南北极点，并且每一个水平曲线闭合回到它自身，形成一个几何圈。这给我们一个度量上有界(具有有限的尺寸)，拓扑上无界的曲面(没有边界的圈)。

6.4.45 GM_双线性格网(GM_BilinearGrid)

GM_BilinearGrid 是在水平和垂直曲线上均使用线串的 GM_GriddedSurface。

注：这不是一个多边形曲面，因为每一个方格网是一个规则曲面，不一定是平面。

6.4.46 GM_双三次格网(GM_BicubicGrid)

6.4.46.1 语义

GM_BicubicGrid 是在水平和垂直曲线上都使用三次多项式样条曲线的 GM_GriddedSurface。

注：样条的初矢量经常由一个额外的控制点行、列对来代替。

6.4.46.2 horiVectorAtEnd, horiVectorAtStart, vertVectorAtEnd, vertVectorAtStart (末端水平矢量、始端水平矢量、末端垂直矢量、始端垂直矢量)

对于一个完整的定义，水平与垂直曲线需要始端与末端切矢量，这些值是由这四个属性值来提供的：

```
GM_BicubicSpline::horiVectorAtEnd; Sequence<Vector>;
GM_BicubicSpline::horiVectorAtStart; Sequence<Vector>;
GM_BicubicSpline::vertVectorAtEnd; Sequence<Vector>;
GM_BicubicSpline::vertVectorAtStart; Sequence<Vector>;
```

6.4.47 GM_B 样条曲面形式(GM_BsplineSurfaceForm)

代码列表“GM_BsplineSurfaceForm”用于指示一个由 GM_BsplineSurface 表达的特定几何形式。其可能值有：

- planar(平面)：在每一个参数上都是 1 阶的 B 样条曲面表达的平面的有界部分。
- cylindrical(柱面)：由 B 样条曲面表达的柱面的有界部分。
- conical(圆锥曲面)：由 B 样条曲面表达的圆锥曲面的有界部分。
- spherical(球面)：由 B 样条曲面表达的球面的有界部分。
- toroidal(圆环曲面)：由 B 样条曲面表达的圆环曲面或圆环曲面的一部分。
- unspecified(未定义)：没有特别定义的曲面。

GM_BsplineSurfaceForm::

planar
cylindrical
conical
spherical
toroidal
unspecified

6.4.48 GM_B 样条曲面(GM_BsplineSurface)

6.4.48.1 语义

B 样条曲面是有理或多项式参数曲面，它通过控制点和基函数以及可能的权重来表达。如果权重都相等，样条为一个分片多项式。如果权重不等，则样条为分片有理函数。如果布尔值“isPolynomial”被设为 TRUE，则权重都被设为 1。

6.4.48.2 degree (次数)

属性“degree”是第一个和第二个参数的基函数的代数次数。如果只设定了一个值，则这两个次数是相等的。

GM_BsplineSurface::degree [1,2]:Integer

6.4.48.3 surfaceForm (曲面形式)

“surfaceForm”是用来标识曲面的特定类型，即哪种样条被用来逼近曲面。它仅仅是用于信息查询，用来捕获原始意图。如果不打算使用样条来逼近曲面，则该属性的值为 NULL。

GM_BsplineSurface::surfaceForm:GM_BsplineSurfaceForm

6.4.48.4 knot (节点)

属性“knot”是两个不同的节点序列，用来定义两个参数的 B 样条基函数。调用该节点数据类型来保存节点的多重性(multiplicity)信息。

GM_BsplineSurface::knot [2]:Sequence<GM_Knot>

6.4.48.5 knotSpec (节点类型)

属性“knotSpec”给出用来定义该样条的节点的分布类型。它仅用于查询，并由不同的构造函数设定。

GM_BsplineSurface::knotSpec[0,1]:GM_KnotType

6.4.48.6 isPolynomial (是多项式)

如果这是一个多项式样条，“isPolynomial”该属性就设为 TRUE。

GM_BsplineSurface::isPolynomial:Boolean

6.4.48.7 GM_BsplineSurface (constructor) (GM_B 样条曲面 (构造函数))

“GM_BsplineSurface”类的构造函数采用上面描述的有关信息，构造一个 B 样条曲面。如果 knotSpec 不存在，则 knotType 是一致的，并且除去第一个和最后一个节点外，节点是均匀间隔的，具有 multiplicity=1。在首末端点处 multiplicity=degree+1。如果 knotType 是一致，则不需要专门定义。

```
GM_BSplineSurface::GM_BSplineSurface(
    pts:Sequence<GM_PointArray>,
    deg[2]:Integer,
    k [0,2]:Sequence<GM_Knot>
    ks[0,1]:GM_KnotType,):GM_BSplineSurface
```

6.5 几何聚集形包

6.5.1 语义

几何对象可以以任意方式聚集。没有假定有任何附加的内部结构用来“组合(collect)”这些所定义类型的几何曲面片。在这方面它不同于定义在 6.6 中的“composites(组合)”与“complexes(复形)”。这些聚集形类型的操作是从构成它们的元素的类的操作中派生出来的操作的总和。应用模式在它们的表达中,可以对于使用多种几何对象的要素使用聚集形,例如使用点的组合来表达一个油库或果园。

6.5.2 GM_聚集形(GM_Aggregate)

6.5.2.1 语义

GM_Aggregate(见图 24)将几何对象收集在一起。由于 GM_Aggregate 经常使用定向(orientation)进行调整,因而曲线引用与曲面引用通常不是直接到 GM_Curve 和 GM_Surface,而是定向到(directed to)GM_OrientableCurve 和 GM_OrientableSurface。

大多数的几何对象都是包含在要素里,但不能固持在强聚集的组合里。因为这个原因,在这一条里描述的组合都是弱聚集,因而应该使用引用来包含几何对象。下面给出各种引用对象之间的关系。

注:以这样一种方式来操作 GM_OrientablePrimitive 的子类,将引用对象链接到某个 GM_OrientablePrimitive 子类对象的一个明确的定向(orientation)上。

6.5.2.2 element (元素)

关联角色“element”是包含在该 GM_Aggregate 中的若干 GM_Object 的集合。在 GM_Aggregate 的子类中,element 被限制为所定义类型的 GM_Primitive。

```
GM_Aggregate::element:Set<GM_ObjectRef>
```

6.5.2.3 fromSet (来自集合)

操作“fromSet”是使用若干 GM_Object 的集合来创建 GM_Aggregate 的构造函数。

```
GM_Aggregate::fromSet(set:Set<GM_Object>):GM_Aggregate
```

6.5.3 GM_多单形(GM_MultiPrimitive)

GM_MultiPrimitive 是所有单形聚集的根类。关联角色“element”是包含在该 GM_MultiPrimitive 中的若干 GM_Primitive 的集合。这里,该属性声明指定包含在该 GM_Aggregate 类型中的那一种 GM_Primitive。

```
GM_MultiPrimitive::element:Set<GM_Primitive>
```

6.5.4 GM_多点(GM_MultiPoint)

6.5.4.1 语义

GM_MultiPoint 是指包含点的聚集形类。关联角色“element”是包含在 GM_MultiPoint 中的所有 GM_Point 的集合。

```
GM_MultiPoint::element:Set<GM_Point>
```

6.5.4.2 position (位置)

派生属性“position”是包含在 GM_MultiPoint 中所有 GM_Point 的 DirectPosition 的集合。

```
GM_MultiPoint::position:Set<DirectPosition>
```

6.5.5 GM_多曲线(GM_MultiCurve)

6.5.5.1 语义

GM_MultiCurve 是指包含 GM_OrientableCurve 实例的聚集形类。关联角色“element”应该是包含该 GM_MultiCurve 中的所有 GM_OrientableCurve 的集合。

GM_MultiCurve::element;Set(GM_OrientableCurve)

6.5.5.2 length (长度)

派生属性“length”是包含在该 GM_MultiCurve 中的所有 GM_Curve 的长度的总和。

GM_MultiCurve::length;Length

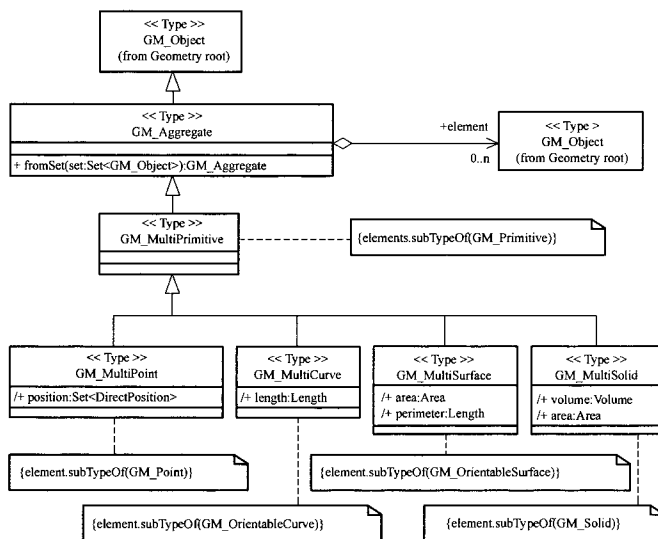


图 24 GM_聚集形(GM_Aggregate)

6.5.6 GM_多曲面(GM_MultiSurface)

6.5.6.1 语义

GM_MultiSurface 是指包含 GM_OrientableSurface 实例的聚集形类。关联角色“element”是包含在该 GM_MultiSurface 中的 GM_OrientableSurface 的集合。

GM_MultiSurface::element;Set(GM_OrientableSurface)

6.5.6.2 area (面积)

派生属性“area”是包含在该 GM_MultiSurface 中的所有 GM_Surface 的面积的和。

GM_MultiSurface::area;Area

6.5.6.3 perimeter (周长)

派生属性“perimeter”是包含在该 GM_MultiSurface 中的所有 GM_Surface 的周长的总和。

GM_MultiSurface::perimeter;Length

6.5.7 GM_多体(GM_MultiSolid)

6.5.7.1 语义

GM_MultiSolid 是指包含 solid 实例的聚集形类。关联角色“element”是包含在该 GM_MultiSolid 中的所有 GM_Solid 的集合。

GM_MultiSolid::element;Set(GM_Solid)

6.5.7.2 volume (体积)

派生属性“volume”是包含在该 GM_MultiSolid 中的所有 GM_Solid 的体积的和。

0.

GM_MultiSolid::volume; Volume

6.5.7.3 area (面积)

派生属性“area”是包含在该 GM_MultiSolid 中的所有 GM_Solid 的边界曲面面积的总和。

GM_MultiSolid::area; Area

6.6 几何复形包

6.6.1 语义

几何复形(GM_Complex)是若干(在一个共同的坐标系下)内部不相交的单形几何对象的集合。进一步,如果一个单形在一个几何复形里,而在该复形内有由若干单形构成的一个单形集合,该单形与其他单形的点态联合(point-wise union)就是这第一个单形的边界。

一个复形的子复形是该复形的单形的子集,即是一个有它们自己的辖区的几何复形。复形的超复形是属于一个更大的复形的单形的超集。这些定义本质上是子集和超集再加上一个它们必须是一个复形的附加限制。一个复形如果不是一个更大复形的子复形,那么它就是一个最大复形。

在几何复形中的几何对象的边界是该复形的子复形。最简单的复形是一个单点。最简单的 1 维复形是具有两个端点的曲线。最简单的 2 维复形是一张曲面具有它的边界曲线和曲线的始点与终点。

复形所在的几何体通常被称为“流形(manifold)”。复形的结构使得构成该流形的几何成分组织成单形元素,类似于很多“图表(chart)”通过“地图集”组织到一个世界地图册中。

有一种方法,但显然不止这一种方法,由单形的集合来生成一个复形,就是由这些单形开始,作如下的操作。

- a) 如果两个单形重叠,就对两个单形细分,去掉重复单元直到没有重叠。
- b) 同样地,如果一个单形不是简单的,细分自交的部分直到没有重叠。
- c) 如果一个单形不是一个点,就计算它的边界作为与其他单形的一个组合,如果可能,使用那些已经存在于产生的集合中的元素,并且将它们插入到复形中去。
- d) 重复步骤 a) 到 c),直到没有新单形。

许多系统都有一个全表面(对于 2 维)和全形体(对 3 维)的概念。这只有当复形所在的空间是一个无界的欧几里德空间时才有效。在这种情况下,对于 2 维,全表面就是复形只有内边界环时所在的曲面(它的外点是在一个“无穷远的点”)。类似的,在 3 维中,全形体是只有内边界壳的复形体。在有界的流形里,例如球,没有点在无穷远处,所有的单形都是有界的。不考虑 Jordan(约旦)分离法则,所有的边界本质上都是内边界。在那些无界的流形中,例如双曲线,可能有多于一个的无界单形。由于本标准不是直接针对这些无界流形,如果本标准用于那些非地理的流形中,有些元素集合的基数(cardinality)可能需要放宽。本标准并不将全表面与全形体当特殊情况对待,因而全表面、全形体与它们的边界之间的关系是以与其他边界关系相同的方式来表达的。

注:在应用模式的内部语义中,一个最大复形可能被合理地考虑为它的单形的一个强聚集类型。由于这个原因,没有定义多个 GM_Primitive 在一个 GM_Complex 中存在方式的控制机制。如果一个强聚集类型用在一个最大复形中,对于子复形的限制关联可能必须把最大复形作为对于单形所在的命名空间来引用。在任何情况下,一旦一个 GM_Primitive 被用在一个 GM_Complex 里,或者一个 GM_Complex 是一个最大复形的子复形,它的边界操作将不再需要构造代表性的 GM_Objects,因为根据一个复形的定义,需要用来表述所包含对象的边界的对象已经存在,而只需要通过 GM_Object::boundary 操作,引用这些所需对象就行了。记住,对于多个 GM_Complexes 包含在另一个 GM_Complex 中是一个“子集-超集”的关联,而当多个 GM_Primitive 包含在一个 GM_Complex 中则是一个“元素-集合”的关联。

6.6.2 GM_复形(GM_Complex)

6.6.2.1 语义

GM_Complex(见图 25)是若干几何分离的简单 GM_Primitive 的一个组合(collection)。如果一个非 GM_Point 的 GM_Primitive 在一个特定的 GM_Complex 中,则在该复形中存在更低维的单形的一

个集合,形成该非 GM_Point 单形的边界。

注:可以有两种截然不同的方式将几何复形看作是一个集合。第一种方式认为几何复形是对象的一个有限集合(通过降维到它的元素成员),第二种方式将几何复形作为点值的无穷集,而点是几何对象的子类。委托(delegate)和子类型(subtyping)这两种用途作为集合接口的两种类型是不相混淆的。要确定一个 GM_Primitive P 是否是一个 GM_Complex C 的元素,调用 C.element(), contains(P)这个方法就行了。

属性“element”允许 GM_Complex 继承有限集合 Set<GM_Primitive> 的行为,而且不会与通过 GM_Object 继承的超限集 TransfiniteSet<DirectPosition> 的同类行为相混淆。

应用模式中,在几何共享很重要的那些场合应该使用复形,例如在计算拓扑的应用中。在复形中,单形可以以“多对多”的形式聚集在若干组合形(composite)中,以便于使用要素的属性。这个例子提供在本标准的附录 D 中。

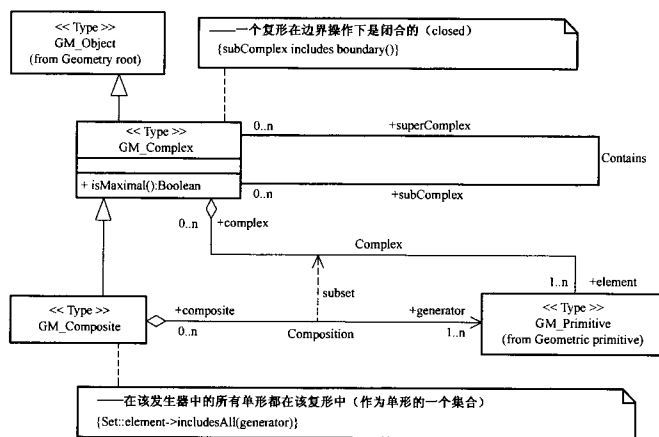


图 25 GM_复形(GM_Complex)

6.6.2.2 isMaximal (是最大的)

当且仅当 GM_Complex 是最大复形时,操作“isMaximal”才返回布尔值 TRUE。

GM_Complex::isMaximal(): Boolean

6.6.2.3 Contains association (包含关联)

关联“Contains”实例化从 Set<GM_Primitive> 中来的 contains 操作作为一个关联。

GM_Complex::subComplex [0..n]: GM_Complex

GM_Complex::superComplex [0..n]: GM_Complex

6.6.2.4 Complex association (复形关联)

关联“Complex”是由 GM_Object 的“contains”操作定义的,而 GM_Object 则是继承自 TransfiniteSet<DirectPosition>。

GM_Complex::element [1..n]: GM_Primitive

如果复形包含一个 GM_Primitive,则它必然包含这些元素在它的边界里。

GM_Complex:

——在边界操作中闭合

self→forAll(self→includesAll(boundary()))

6.6.3 GM_组合形(GM_Composite)

6.6.3.1 语义

一个几何组合形(geometric composite)GM_Composite(见图 26)是一个几何复形,该几何复形具有一个潜在的同构于 $n-1$ 个单形的几何核心。这样,组合曲线(composite curve)是曲线的组合(collection),它的几何接口应当满足单曲线的接口(尽管是非常复杂的单曲线)。组合曲线是打算以数据集的方式使用属性值,虽然它们在几何上已经被分解了。组合形通常用来暴露它们的拓扑属性。

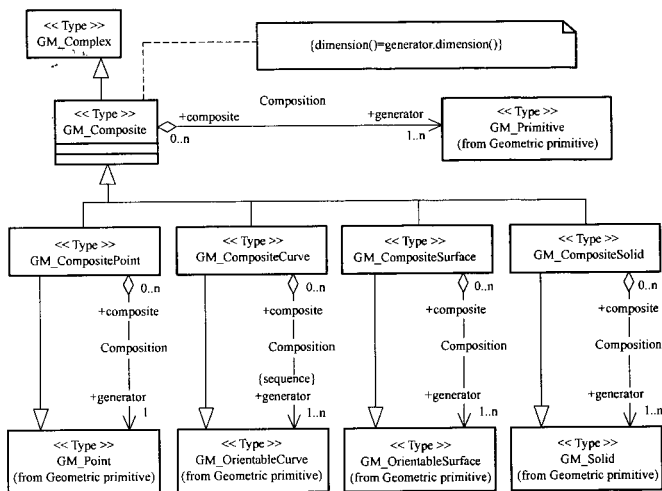


图 26 GM_组合形(GM_Composite)

6.6.3.2 generator (生成器)

关联角色 Composition::generator 是若干 GM_Primitive 的同类组合,它的联合就是该组合形的几何核心。该复形将包括这个发生器中的所有单形,以及在这些单形的边界上的所有单形,继续该过程,直到包括到 GM_Point。这样,在 GM_Composite 上的关联角色 Composition::generator 是 GM_Complex 上的关联角色 Complex::element 的子集。

GM_Composite::generator[1..n]:GM_Primitive

发生器中的几何类型完全由该组合形对象的维数决定。组合的曲线与曲面是面向允许以适当的方式集成到该组合形中。

GM_CompositePoint:

generator.type=GM_Point

GM_CompositeCurve:

generator.type=GM_OrientableCurve

GM_CompositeSurface:

generator.type=GM_OrientableSurface

GM_CompositeSolid:

generator.type=GM_Solid

6.6.4 GM_组合点(GM_CompositePoint)

6.6.4.1 语义

为了完整性,组合点 GM_CompositePoint(见图 27)作为一个单独的类。它是包含一个且仅包含一个 GM_Point 的 GM_Complex。

6.6.4.2 generator (生成器)

关联角色 Composition::generator 关联该 GM_Composite 点到该复形内的一个单独的单形上。

GM_CompositePoint::generator [1];GM_Point

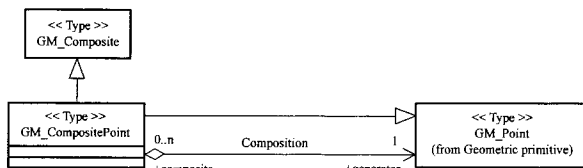


图 27 GM_组合点(GM_CompositePoint)

6.6.5 GM_组合曲线(GM_CompositeCurve)

6.6.5.1 语义

组合曲线 GM_CompositeCurve(见图 28)是具有曲线的所有几何性质的 GM_Composite。这些性质是在“curve”操作中实例化的。本质上,一条组合曲线是可定向曲线(GM_OrientableCurve)的一个列表,并遵从每一条曲线(第一条除外)以前一条曲线的终点为始点的这样一种规则。

6.6.5.2 generator (生成器)

关联角色 Composition::generator 将该 GM_CompositeCurve 与其发生序列中的单形曲线 GM_Curve 和 GM_OrientableCurve 关联起来,这些曲线形成该复形的核心。

GM_CompositeCurve::generator;Sequence<GM_OrientableCurve>

——在该发生器里,每一条有向曲线的始点是前一条有向曲线的终点

GM_CompositeCurve:

```
forAll(1 < j < generator.count-1)→
```

```
generator[j].endPoint=generator[j+1].startPoint;
```

注:要得到在 GM_Complex 里的元素的完整表达,在 GM_Curve 发生器集合的边界上的 GM_Point 应该加到发生器列表的曲线里。

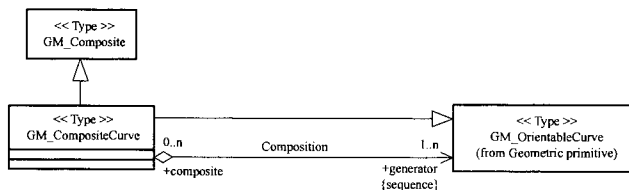


图 28 GM_组合曲线(GM_CompositeCurve)

6.6.6 GM_组合曲面(GM_CompositeSurface)

6.6.6.1 语义

组合曲面 GM_CompositeSurface(见图 29)是具有曲面的所有几何性质的 GM_Composite,因而可以将其当作可定向曲面(GM_OrientableSurface)的一种类型。本质上,组合曲面是有向曲面的组合,这些有向曲面以成对的公共边界曲线连接在一起,并被当作一个整体,形成一个单独的曲面。

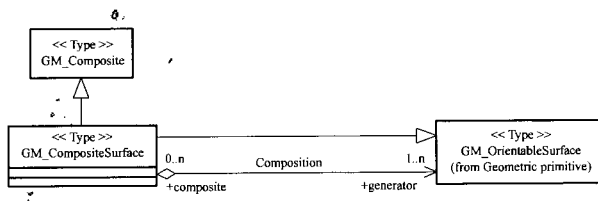


图 29 GM_组合曲面(GM_CompositeSurface)

6.6.6.2 generator (生成器)

关联角色 `Composition::generator` 关联该 `GM_CompositeSurface` 到若干单形 `GM_Surface` 上,并且在它的产生集合(`GM_Surface` 的列表)中的这些 `GM_OrientableSurfaces` 形成该复形的核心。

`GM_CompositeSurface::generator; Set<GM_OrientableSurface>`

注: 要得到 `GM_Complex` 里元素的完整表达,在这些 `GM_Surface` 发生器集合的边界上的 `GM_Curve` 和 `GM_Point` 应该加到发生器列表的曲线里。

6.6.7 GM_组合体(GM_CompositeSolid)

6.6.7.1 语义

组合体 `GM_CompositeSolid` (见图 30)是具有体的所有几何性质的 `GM_Composite`。本质上,组合体是若干体的组合,这些体以成对的公共边界曲面连接在一起形成一个单独的体。

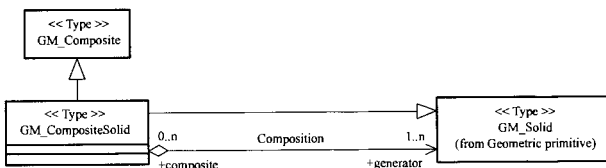


图 30 GM_组合体(GM_CompositeSolid)

6.6.7.2 generator (生成器)

关联角色 `Composition::generator` 将该 `GM_CompositeSolid` 关联到在它的产生集合里的所有单形 `GM_Solid` 上,这些 `GM_Solid` 形成该复形的核心。

`GM_CompositeSolid::generator; Set<GM_Solid>`

注: 为了得到在 `GM_Complex` 里的元素的完整表达,如果要将这些 `GM_Solids` 加到发生器列表里,则还需要将在发生器集合边界上的 `GM_Surface`、`GM_Curve` 和 `GM_Point` 加到发生器列表里。

7 拓扑包

7.1 语义

拓扑的最大用处是加速几何计算。通过拓扑方法将要素实例与几何对象实例清楚地关联起来,这种关联是以相容以及从它们的隐含几何关系派生的方式进行的(见 D.3)。在有些情况下,这些关联要从一个与要素实例的关系不一致的几何概念中派生出来。为了这个目的,有必要定义与第 6 章中的几何包平行的拓扑包。图 31 显示这个包和它的依赖关系。

图 32 给出了基本拓扑包的类结构的一个概览。该图中的根类是 `TP_Object`。在它下面是 `TP_Primitive` 和 `TP_Complex`,这两者的关系类似于 `GM_Primitive` 和 `GM_Complex`,因此 `TP_Complex` 是若干 `TP_Primitive` 的一个有组织的结构。主要的差别在于 `GM_Primitive` 与 `GM_Complex` 的耦合更松散,允许它独立存在;而一个 `TP_Primitive` 必须至少存在于一个 `TP_Complex` 之中。`TP-DirectedTopo`(`TP` 有向拓扑)的实例应该包含对于一个 `TP_Primitive` 的引用和一个定向参数,类似于

6.3.13中的 GM_OrientablePrimitive。不管几维,都只有两个可能的定向,每一个单形被关联到两个有向拓扑实体上,这类似于在 GM_OrientableCurve 和 GM_Curve 的之间的关系以及在 GM_OrientableSurface 和 GM_Surface 之间的关系。为了保存对象的数目和使得单形自然地与它的正定向相一致,每一维的每一个单形在它的相应定向拓扑对象下又被细分,这在 7.3.11.1 中进一步扩展。

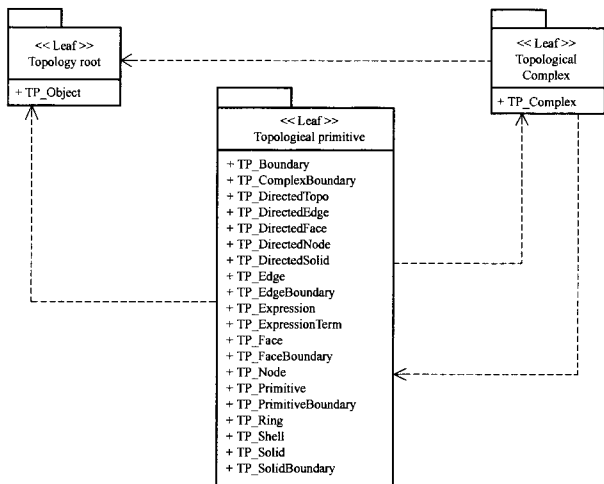


图 31 拓扑包、类内容和内部依赖关系

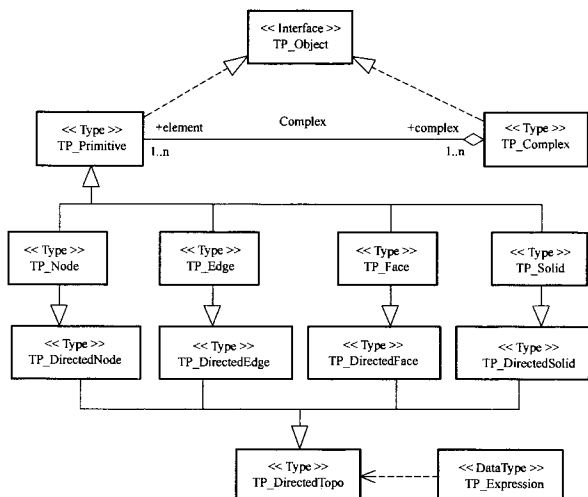


图 32 拓扑类概图

7.2 拓扑根包

7.2.1 语义

几何计算,例如包含(点是否在多边形内)、邻接、边界和网络追踪都是高强度计算。为此,人们构造了被称作拓扑复形的组合结构来将计算几何的运算转换为组合运算,其拓扑定义平行于第6章里的几何结构定义。另外一个目的是在地理信息范围内,使要素实例独立于它的几何表达。为了第一个目的,在这一章里的拓扑定义平行于第6章里的几何结构定义。为了第二个目的,在该包里的类被定义为可以独立于几何类使用。

拓扑复形由若干个维数最高到该复形所在维的不同种类拓扑单形的组合构成。这样,2维复形应包含面、边和结点,而1维复形或图(graph)仅包含边和结点。

注:拓扑单形对应于几何单形,而不是几何单形的子类。这与拓扑复形是被构造出来通过使用组合运算优化几何计算过程的观点相一致。这就允许通过使用拓扑复形来创建结构而忽略几何的限制,而该拓扑复形并不是由几何复形来实现的。

理解使用拓扑计算的关键在于考察这两个系统的相关过程。如图33所示,这两个系统间单形与复形的关系上具有很大的对应性。

拓扑系统是基于多变量多项式的代数运算。在拓扑包中,过程(procedures)、函数(functions)和操作(operations)的定义是以在几何领域里的几何问题可以转化为拓扑领域的代数问题的这样一种思路来解决的,最后再将解转换回几何领域中去。一个拓扑表达式在代数里是一个多变量一阶多项式,这里的变量对应于拓扑单形。

图33的图表总结了拓扑与几何间的相互关系。OCL约束意味着角色由TP_Primitive::complex(拓扑单形取复形)操作跟随TP_Complex.geometry(拓扑复形取几何)操作,在作用上等价于TP_Primitive::geometry(拓扑单形取几何)操作跟随GM_Primitive::complex(几何单形取复形)操作。

注:一个单独的GM_Primitive可以被绑定到很多独立的GM_Complexes上,这些GM_Complexes的每一个都可能是一个不同的TP_Complex的实现。这样,一个GM_Primitive可以是很多不同的TP_Primitive的实现,因为一个TP_Primitive必须出现在一个而且也只能出现在一个最大TP_Complex中(见7.3.10.2)。由于对于可实例化的类,既可以实现为TP_Primitive和TP_Complex,也可以实现为GM_Primitive和GM_Composite,因而TP_Primitive特殊的实例也可以通过一个GM_Composite来实现,实例见附录D.3。

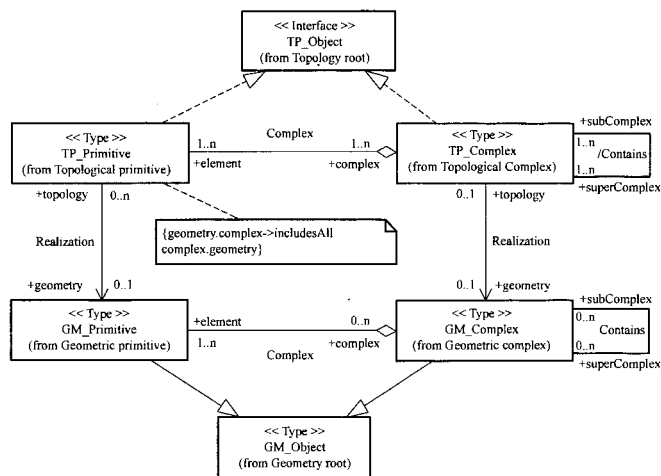


图33 几何与拓扑间的关系

7.2.2 TP_对象(TP_Object)

7.2.2.1 语义

拓扑对象 TP_Object(见图 34)是一个为拓扑复形和拓扑单形提供根类型的抽象类。

在逻辑和结构上,拓扑对象和几何对象可以共享一些子类的结构,但是由于有一个从拓扑到几何的分类同态(categorical homomorphism),这个同态保存边界操作,这种存取可能导致拓扑对象和相对应的几何对象的边界间的混乱。当这两种机制共享很多计算特征时,例如由同态所示例的那样,它们是不同的操作,并且需要清楚地分开。

7.2.2.2 dimension (维数)

“dimension”操作将返回该 TP_Object 的拓扑维数,它是一个整数。它仅依赖于该对象实例化的类,在没有改变类的情况下,它不会随着每一个特别的对象而变化。例如结点类的维数是 0,边是 1、面是 2、体是 3。任何关联到该 TP_Object 上的 GM_Object 也将与该 TP_Object 具有相同的维数。

TP_Object::dimension():Integer

7.2.2.3 boundary (边界)

“boundary”操作将返回 TP_DirectedTopo 的一个集合,这些 TP_DirectedTopo 结构是为了表达该 TP_Object 的边界 TP_Boundary。

TP_Object::boundary():TP_Boundary

如果该 TP_Object 关联到一个 GM_Object 上,它的边界将与在几何包中描述的 GM_Object 具有一致的定向。

作为限制,边界的维数将始终比它原来对象的维数少 1。由此,空集的维数被认为是“-1”。

TP_Object:

boundary, dimension()=dimension()-1

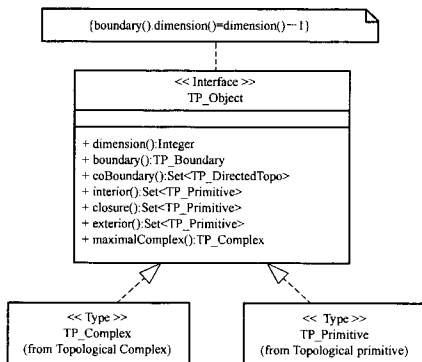


图 34 TP_对象(TP_Object)

图 35 显示了边界函数怎样被视作从每一个对象到比它低一维的对象间的一个关联。

在大多数情况下,返回值将是一个有效的 TP_Expression(TP_表达式)(见 7.3.18)。由于需要最简单的条件,返回的边界也可能不是有效的 TP_Expression。在一个面上摇摆或孤立的边(例如在边的两侧都是同一个面的这种边)将抵消向拓扑表达式的转换。

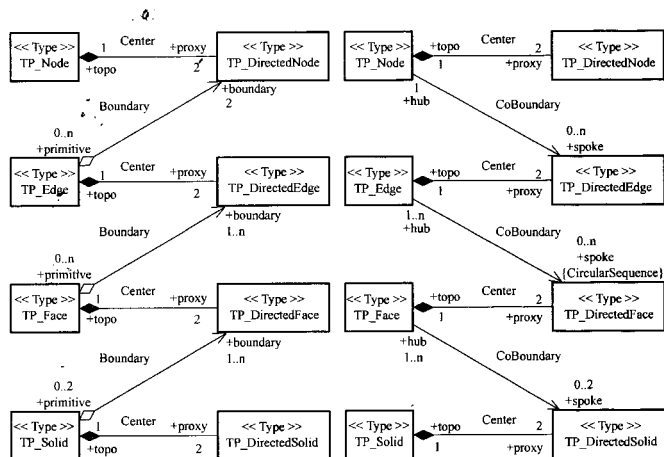


图 35 边界和作为关联表达的余边界操作

7.2.2.4 coBoundary (余边界)

“coBoundary”该操作将返回 TP_DirectedTopo 的集合, 这些 TP_DirectedTopo 代表在边界中包含该 TP_Object 的所有的 TP_Object。

在大多数情况下, 返回值将是一个有效的 TP_Expression (见 7.3.20)。有一个例外是当相应的 GM_Object 在一个闭合对象的边界上 (例如曲线开始与终止于同一个点)。对应于 GM_Object 的 TP_Object 将以相反的方向两次出现在 TP_DirectedTopo 中, 因此当 coBoundary 从 TP_DirectedTopo 的集合映射到 TP_Expression 上时就相互抵消了。

TP_Object::coBoundary(); Set(TP_DirectedTopo)

图 36 显示了这个操作怎样被作为不同维度的 TP_Primitive 之间的关系, 类似于边界操作, 但是以相反的方向增加而不是减少维数。

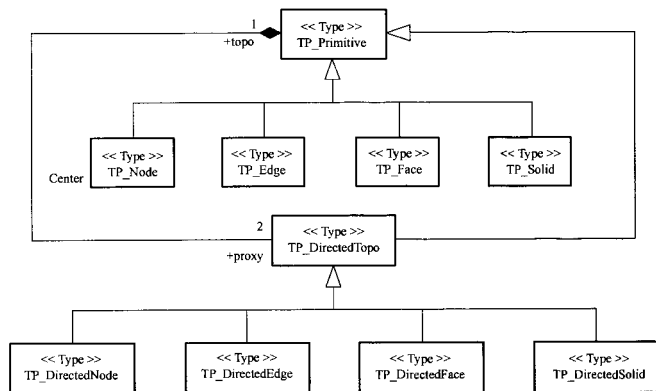


图 36 拓扑中的重要类

7.2.2.5 interior (内部)

“interior”操作将返回 TP_Primitive 的有限集合,该集合在该对象的最大复形里,包含该对象的内部。对于一个 TP_Primitive,这就是一个自引用。对于一个 TP_Complex,这就是在该 TP_Complex 里,但不在边界上的所有 TP_Primitive 元素。它与该 TP_Object 的几何实现的内部是同态等阶(homomorphic equivalent)。

TP_Object::interior():Set<TP_Primitive>

7.2.2.6 exterior (外部)

“exterior”操作将返回 TP_Primitive 的有限集合,这些 TP_Primitive 包含该 TP_Object 的外部,但仍然在该 TP_Object 所在的最大复形内,即它由在该 TP_Object 所在的最大 TP_Complex 内,但不在该 TP_Object 的内部或边界上的所有 TP_Primitive 组成。

TP_Object::exterior():Set<TP_Primitive>

7.2.2.7 closure (闭包)

“closure”操作被定义为对象的内部与边界的联合。该操作经常是有用的,但在基本实现中并不作要求。

TP_Object::closure()=interior().union(boundary())

7.2.2.8 maximalComplex (最大复形)

“maximalComplex()”操作将返回包含该 TP_Object 最大 TP_Complex。

TP_Object::maximalComplex():TP_Complex

一个 TP_Object 应该且只能被包含在一个最大 TP_Complex 里。

注:一个复形如果不是被包含在一个更大的复形里,则它自己就是一个最大复形。由该操作所暗示的对于集合的基数(cardinality)的限制意味着任何拓扑对象是在一个,且只在一个最大复形里。

7.3 拓扑单形包

7.3.1 语义

拓扑单形包包含每一维数的所有单形,并且支持这些类对于它们的结构关系的表达。

7.3.2 TP_边界(TP_Boundary)

这个类是用作拓扑包中的所有边界数据类型的根类。除了要求是一个拓扑表达式(TP_Expression) 和一个圈(cycle)之外,它不需要更多的细节。

TP_Boundary:

IsCycle();

7.3.3 TP_复形边界(TP_ComplexBoundary)

这个类是用作拓扑包中的拓扑复形的所有边界数据类型的根类。除了要求是拓扑表达式外,它不需要更多的细节。

7.3.4 TP_单形边界(TP_PrimitiveBoundary)

每一个拓扑单形都可以返回它的边界。在 TP_PrimitiveBoundary(见图 37)下的数据类型是用来以一种方便的方式构造它们的边界。因为 TP_Node 有一个空边界,对于它的边界没有特别的数据类型。

一个简单的事实是任何几何对象的边界都是一个圈(没有边界)。曲面的边界成分是若干环形组合曲线的集合,这些环形组合曲线的每一条都是自身闭合的。为了拓扑与几何间的一致性,对于所有 TP_Boundary 的子类,也要求其边界是一个圈。如果一个 TP_Complex 代表一个 GM_Complex 的拓扑,那么其几何实现也将遵循这个限制。

7.3.5 TP_边边界(TP_EdgeBoundary)

TP_EdgeBoundary(见图 37)包含作为 TP_DirectedNode 实例的两个 TP_Node 引用。startNode(始结点)为正定向,endNode(终结点)为负定向。作为一个 TP_Expression,TP_EdgeBoundary 就是:

Edge. boundary() = +endNode - startNode

TP_EdgeBoundary 的属性是:

TP_EdgeBoundary::startNode: TP_DirectedNode;

TP_EdgeBoundary::endNode: TP_DirectedNode;

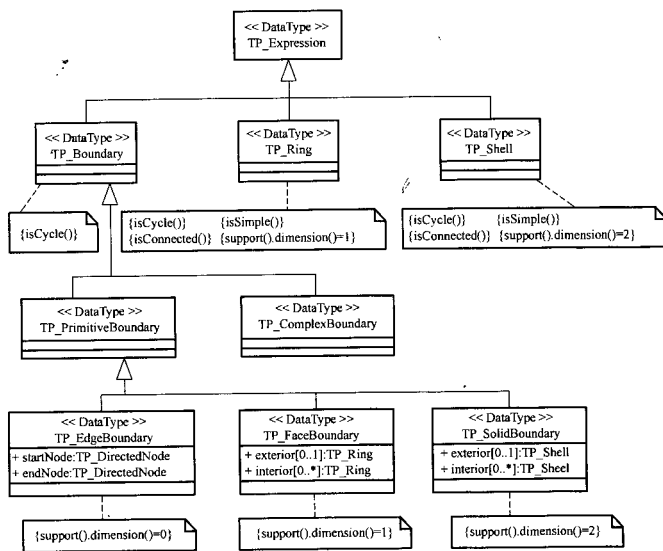


图 37 边界关系数据类型

7.3.6 TP_拓扑面边界(TP_FaceBoundary)

TP_FaceBoundary 由一些 TP_Ring(TP_环)构成,对应于它的边界的不同成分。在一般的 2 维情况下,这些环中的一个被单独称之为外边界。在一个更一般的流形情况下,这并不总是可行的,在有些情况下,所有边界都将是作为内边界列出,外边界是空。

TP_FaceBoundary::exterior[0..1]: TP_Ring;

TP_FaceBoundary::interior[0..n]: TP_Ring;

由于每个环都是定向的,并且拓扑面在环的左边,因而可得到拓扑面边界的表达式为:

Boundary(face) = b; TP_FaceBoundary = b. exterior + b. interior

7.3.7 TP_拓扑体边界(TP_SolidBoundary)

TP_SolidBoundary 类似于 TP_FaceBoundary。在通常的欧几里德空间中,有一个壳被单独称为外边界。在更一般的情况下,这并不总是可行的。

TP_SolidBoundary::exterior[0..1]: TP_Shell;

TP_SolidBoundary::interior[0..n]: TP_Shell;

由于每个壳都是定向的,并且拓扑体在壳的下面,因而可得到拓扑体边界的表达式为:

Boundary(solid) = b; TP_SolidBoundary = b. exterior + b. interior

7.3.8 TP_环(TP_Ring)

TP_Ring 用来表达 TP_FaceBoundary 连在一起的一个单独成分。它由若干 TP_DirectedEdge

(TP_有向边)构成,这些 TP_DirectedEdge 连成一个圈(其边界为空的一个对象)。一个 TP_Ring 是一个结构上类似于 GM_CompositeCurve,即在序列中每一个 TP_DirectedEdge 的 endNode 是下一个 TP_DirectedEdge 的 startNode。由于序列是循环的,该规则没有例外。

作为一个 TP_Expression,TP_Ring 的解释是有向边的一个序列。用在正定向中的每一条边“e”表达为“+e”,用在负定向中的每一条边“d”表达为“-d”。因为 TP_Ring 被用在 TP_FaceBoundary 对象中,这些环应该是定向的,所以在任何几何实现中,面是位于环的“左边”。

7.3.9 TP_壳(TP_Shell)

TP_Shell 用来表达 TP_SolidBoundary 连在一起的一个单独成分。它由连接成一个拓扑圈(其边界为空的对象)的若干 TP_Face 构成。不像 TP_Ring,TP_Shell 没有自然的排序方法。

作为一个 TP_Expression,TP_Shell 的解释是定向面的一个集合。用在正定向中的每一个面“f”表达为“+f”,用在负定向中的每一个面“g”表达为“-g”。因为 TP_Shell 是用在 TP_SolidBoundary 对象中,这些壳应该被定向,所以在任何几何实现中,壳的外法向指向离开该体的方向。

7.3.10 TP_单形(TP_Primitive)

7.3.10.1 语义

拓扑单形 TP_Primitive(见图 38)是一个拓扑复形中无法再分解的元素。这样,它们通常对应于同样维数的几何单形,这些几何单形是一个几何复形的组成部分。如果一个几何复形是一个拓扑复形的几何实现,则这些单形将保持维数并一一对应。

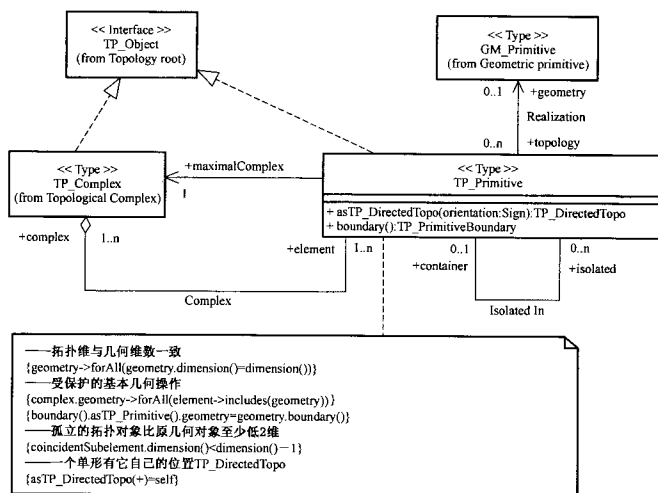


图 38 TP_单形(TP_Primitive)

7.3.10.2 Realization (实现)

关联操作“Realization”链接该 TP_Primitive 到它所在的最大复形中表达它的那个 GM_Primitive 上。如果该 TP_Primitive 描述的是一个没有被 GM_Complex 实现的逻辑拓扑结构,则包含该 TP_Primitive 的最大 TP_Complex 所对应的所有 TP_Primitive 的这种关系都将是空。每一个 GM_Primitive 可以被关联到任何 TP_Complex 的最多一个 TP_Primitive 上。如果该 TP_Primitive 是在任何实现的 TP_Complex 中,则它恰好关联到一个 GM_Primitive 上。一个 GM_Primitive 可以被关联到多个

不同 TP_Complex 的不同 TP_Primitive 上。

```
TP_Primitive::geometry [0,1];GM_Primitive
GM_Primitive::topology [0..n];TP_Primitive
```

注：由于 GM_Composite(GM_组合形)是对应的单形下的子类型，可以定义一个模式使 TP_Primitive 是 GM_Composite 在相同维里的实现。这样，TP_Edge 就可以作为一个 GM_CompositeCurve(GM_组合曲线)来实现。见附录 D.3。

为了保持拓扑对象和它的边界算子间以及对应的几何对象与它的边界算子间的同态，由该关联定义的映射是保持维数不变以及在两个域的元素与复形间的关联保持不变。

```
TP_Primitive:
dimension()=geometry.dimension();
```

7.3.10.3 Complex association (复形关联)

“Complex”该关联操作将该 TP_Primitive 关联到包含它的若干 TP_Complexes 构成的有限集合上。每一个 TP_Primitive 可以在若干个 TP_Complexes 里，这些 TP_Complexes 都是包含该 TP_Primitive 的一个唯一最大 TP_Complex 的子复形。

```
TP_Primitive::complex [1..n];TP_Complex
TP_Complex::element [1..n];TP_Primitive
```

7.3.10.4 IsolatedIn association (孤立式关联)

在拓扑中，在维数相差 1 或 0 的单形间的邻接性操作都是由 boundary 或 coboundary 操作来完成的。这些操作仅涉及位于另一个更高维的单形的边界上的一个单形的实例，或处理共享一个边界元素具有相同维数的单形的实例。这包括在边的两侧具有相同的面的那种“悬挂(dangling)”的边，或在面的两面具有相同的体那种“悬挂”的面。有一个例外，当一个单形完全被至少高两维的另一个单形所包围，而没有中间的过渡单形。这些单形是未关联的。例如，在面当中没有附着在作为面的边界的边上的结点。在 3 维空间中，孤立的结点可能是连在不是曲面边界的边上。在这种情况下，该边是由垂直于该曲面的一条曲线实现的，而该(几何)曲面是该(拓扑)面的实现。在 3 维体内，这包括没有附着在体的边界曲面上的若干结点或边。

```
TP_Primitive::isolated [0..n];TP_Primitive
TP_Primitive::container [0,1];TP_Primitive
TP_Primitive:
isolated.dimension() < self.dimension()-1;
container.count=0 implice
TP_Primitive→exists(boundary(). topo→includes(self))
```

7.3.10.5 boundary (边界)

TP_Primitive 的“boundary”操作将通过向 TP_DirectedTopo 集合中添加更多的结构来重载定义在 TP_Object 中的“boundary”操作。

```
TP_Primitive::boundary();TP_PrimitiveBoundary
```

由于 TP_Primitive 是抽象的，附加结构只是为 TP_Primitive 的每一个子类型所定义。

7.3.11 TP_有向拓扑(TP_DirectedTopo)

7.3.11.1 语义

从计算的观点来看，TP_DirectedTopo 的元素(见图 39、图 40)等价于几何包中的各种定向几何对象 GM_OrientableObject(如 GM_OrientableCurve 和 GM_OrientableSurface)。TP_DirectedNode 和 TP_DirectedSolid 没有单独的几何对象与之等价。

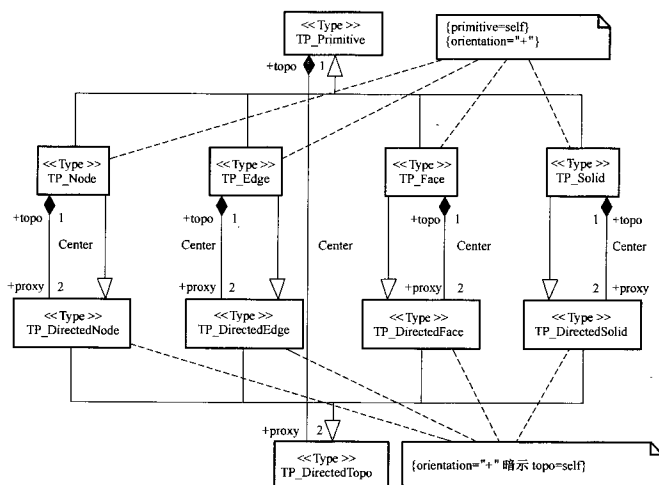


图 39 TP_有向拓扑子类 (TP_DirectedTopo subclasses)

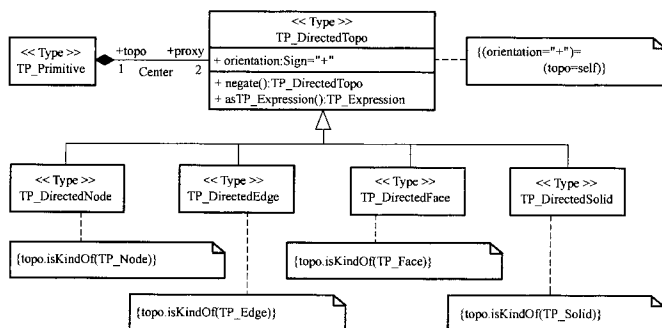


图 40 TP_有向拓扑 (TP_DirectedTopo)

在几何上,每一个拓扑单形是从相应的有向拓扑单形继承而来的,但它满足更多的约束。这意味着 TP_Node 等价于“正”的 TP_DirectedNode, TP_Edge 等价于一个“正”的 TP_DirectedEdge,以此类推。

注:一个可选择的层次类型将 TP_Primitive 和 TP_DirectedTopo 分开,对于每一个单形,它们有 3 个对象,单形自己、等价的正有向拓扑单形和它的翻转(一个负的有向)拓扑单形。该方案是对于本模型中的抽象类型的一个有效实现,但它没有强调一个拓扑单形和它的正有向拓扑单形之间的逻辑等价性。从代数的观点看,子类和 OCL 限制把一个单形当作一个正有向单形看待,这使得它等价于代数中对于“+”的标准解释,就如同“ $x = +x$ ”一样。因为对于拓扑对象最主要的应用是在它们的符号操作中,保持其代数含义是合适的。

在维数邻接的有向拓扑对象之间有一个隐含的关系。boundary 和 coboundary 操作及关系使得它们具有相同的定向含义。这样,如果一个正的有向边是在一个面的边界上,则这个正的有向面是在相关联的边的余边界上。如果一个正的有向结点是在一个边的边界上,则相应的正的有向边是在它相关联的结点的余边界上。

0

7.3.11.2 orientation (定向)

属性“orientation”的意义是该有向拓扑对象以该定向关联到它所在的 TP_Primitive 上。

TP_DirectedTopo::orientation; Sign = “+”

7.3.11.3 negate (负)

“negate”操作返回该单形的相反定向。

TP_DirectedTopo::negate(); TP_DirectedTopo

7.3.11.4 asTP_Expression (作为拓扑表达式)

“asTP_Expression”操作将从它的 TP_DirectedTopo 中创建一个 TP_Expression, 并保持其符号和定向意义。该操作是从 TP_Expression 类来的构造函数。

TP_DirectedTopo::asTP_Expression(); TP_Expression

7.3.11.5 Center association (中心关联)

在“Center”关联中的角色“topo(拓扑)”标识所关联的 TP_Primitive。其相反的角色“proxy(代理)”标识关联到这个特定的 TP_Primitive 上的这两个 TP_DirectedTopo 的实例。

TP_DirectedTopo::topo [1]; TP_Primitive

TP_Primitive::proxy [2]; TP_DirectedTopo

7.3.11.6 约束

遵循有向拓扑对象的语义逻辑, 对于每一个有向拓扑对象所关联的拓扑都应当有一个适当的类型。

TP_DirectedNode;

topo. isKindOf(TP_Node);

TP_DirectedEdge;

topo. isKindOf(TP_Edge);

TP_DirectedFace;

topo. isKindOf(TP_Face);

TP_DirectedSolid;

topo. isKindOf(TP_Solid);

注: 这些约束使用了 OCL 的操作符“isKindOf(是一种)”来指示对应于拓扑单形的有向拓扑单形的类必须是相应的拓扑单形类型的一个实现。

“Center”关联是边界与余边界操作代数的一个重要部分。

TP_DirectedTopo;

[boundary() = (orientation) * topo. boundary()]

TP_Primitive;

[boundary() = (proxy. orientation) * proxy. boundary()]

TP_DirectedTopo;

negate. topo = topo;

negate. orientation <> orientation;

7.3.12 TP_结点(TP_Node)

7.3.12.1 语义

TP_Node(见图 41)继承了来自 TP_Primitive 的所有接口, 在边界和余边界上具有更精巧的结构。

对于 TP_Node, 定义在 TP_Object 上的“coBoundary”操作将返回 TP_DirectedEdge 的引用的一个集合, 该集合指示哪些边进入(正 TP_DirectedEdge)该结点, 哪些边离开(负 TP_DirectedEdge)该结点。该操作重载了 TP_Object 类。同样的信息可以表达为一个关联(即 TP_Node 的余边界表达为对 TP_DirectedEdge 的关联)。

注: 当 2 维最大 TP_Complex 中包含该 TP_Node 时, 余边界在这个最大 TP_Complex 的任何几何实现中可以作为逆时针循环序列来存储。在 3 维复形中, 其次序是任意的。

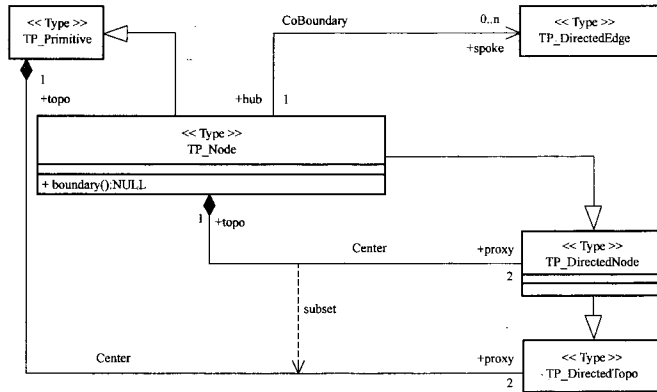


图 41 TP_结点(TP_Node)

TP_Node::coBoundary; Set(TP_DirectedEdge) {size=[0..n]}

TP_Node::coBoundary.spoke; Set(TP_DirectedEdge) {size=[0..n]}

7.3.12.2 Center association (中心关联)

每个 TP_Primitive, 包括 TP_Node, 被关联到两个 TP_DirectedTopo 实例上。

TP_Node::proxy [2]; TP_DirectedNode

TP_DirectedNode [1]; Reference(TP_Node)

7.3.12.3 boundary (边界)

TP_Node 的 boundary 操作通过指定一个空集来重载在 TP_Object 中的定义。

TP_Primitive::boundary(); NULL

7.3.12.4 约束

TP_Node 的维数应该是 0, 其边界是空(NULL)。

TP_Node;

TP_Object::dimension=0;

TP_Object::boundary()=NULL;

注: 在一个拓扑面中, 即使一个结点是在一条边的端点上, 仍然可能是孤立的, 只要这条边不是在包含这个拓扑面的边界上。这条边的几何实现将是游离在空间但终止在它在一个曲面的相交处。

7.3.13 TP_有向结点(TP_DirectedNode)

“TP_DirectedNode”类在一个计算拓扑类的 TP_Expression 中支持 TP_Node。对于 TP_Node, 定义在 TP_Object 上的“boundary”操作将始终返回一个零值表达式, 以对应于空集。该操作重载了来自 TP_Object 的 boundary 操作。

TP_Node::boundary(); NULL

7.3.14 TP_边(TP_Edge)

7.3.14.1 语义

单形 TP_Edge(见图 42)是 1 维拓扑单形。对于 TP_Edge, 定义在 TP_Object 上的“boundary”操作将返回一个结点对, 一个是在边的始点(负 TP_DirectedNode), 一个是在边的终点(正 TP_DirectedNode)。该操作重载了来自 TP_Object 的 boundary 操作。同样的信息可以表达为一个关联(即 TP_Edge 的边界表达为对 TP_DirectedNode 的关联)。

```

TP_Edge::boundary():Set<TP_DirectedNode> {size==2}
TP_Edge::boundary, boundary:Set<TP_DirectedNode> {size==2}

```

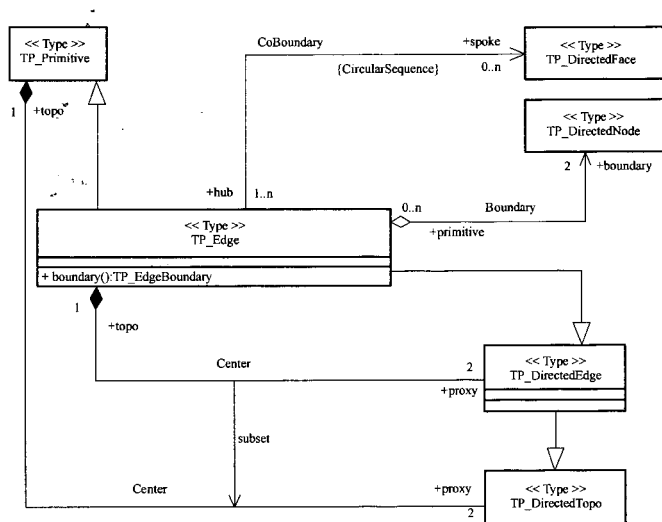


图 42 TP_边(TP_Edge)

7.3.14.2 coBoundary (余边界)

对于 TP_Edge, 定义在 TP_Object 里的“coBoundary”操作将返回有向面(Directed Edge)的一个循环序列, 指示哪些面在它们的边界上使用了这条边(正 TP_DirectedFace)或者是这条边的负代理(负 TP_DirectedFace)。当从包含该 TP_Edge 的最大 TP_Complex 的任何几何实现的关联曲线的端点来观察, 该循环序列表达这些面的一个逆时针枚举。该操作重载了 TP_Object 的操作。同样的信息可以实现为一个关联(即 TP_Edge 的余边界表达为对 TP_DirectedFace 的关联)。

```

TP_Edge::coBoundary():CircularSequence<TP_DirectedFace> {size=[0..n]}
TP_Edge::coBoundary, spoke:CircularSequence<TP_DirectedFace> {size=[0..n]}

```

注: 在 2 维平面的情况下, 边的余边界最多有两个面, 在全拓扑(full topology)的情况下, 正好有两个有向面, 一个有向面以正“+”定向关联到它的左边界上, 另一个有向面以负“-”定向关联到它的右边界上。

7.3.14.3 boundary (边界)

TP_Edge 的边界操作“boundary”以指定一个包含一个始结点和一个终结点的 TP_EdgeBoundary 来重载定义在 TP_Object 中的 boundary 操作。

```
TP_Edge::boundary():TP_EdgeBoundary
```

TP_Edge 也将有一个关联 Boundary, 它所带的关联角色 boundary 定义了相同的信息, 即两个有向边, 正的定向到终结点上, 负的定向到始结点上。

```
TP_Edge::boundary [2];TP_DirectedNode
```

7.3.14.4 Center association (中心关联)

包括 TP_Edge 的每个 TP_Primitive, 被关联到两个 TP_DirectedTopo 实例上。

```
TP_Edge::proxy [2];TP_DirectedEdge
```

TP_DirectedEdge::topo [1]; Reference(TP_Edge)

注：在 2 维平面的情况下，每条有向边最多约束在一张拓扑面上，在完整的平面拓扑中，正好约束在一张拓扑面上。
在 3 维情况下，或者在非平面 2 维复形中，一条有向边可以约束几张拓扑面。

7.3.14.5 约束

TP_Edge 的维数为 1。

TP_Edge:

TP_Object::dimension()=1

7.3.15 TP_有向边(TP_DirectedEdge)

“TP_DirectedEdge”类在计算拓扑类 TP_Expression 中支持 TP_Edge。它类似于 GM_OrientableCurve 的概念，它的作用是当需要的时候，作为基本的曲线/边的代理。

7.3.16 TP_拓扑面(TP_Face)

7.3.16.1 语义

“TP_Face”类(见图 43)为 GM_Surface 提供拓扑单形。

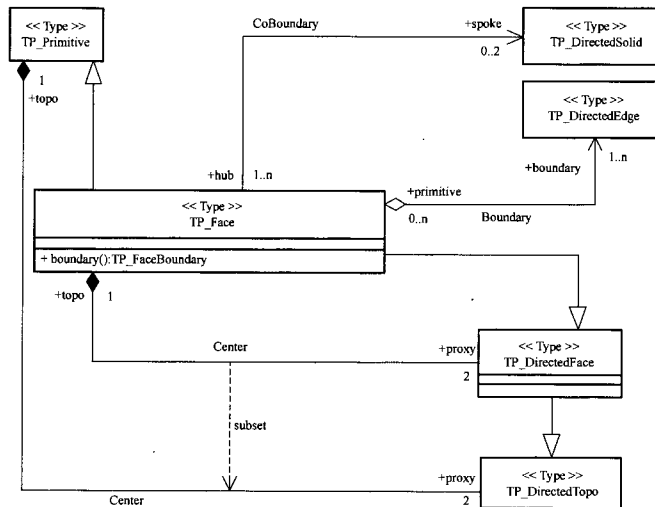


图 43 TP_拓扑面(TP_Face)

7.3.16.2 边界 (boundary)

对于 TP_Face，定义在 TP_Object 中的“boundary”将返回具有适当定向的有向边的一个集合。该操作重载了来自 TP_Object 类的 boundary 操作。同样的信息可以表达为一个关联（即 TP_Face 的边界表达为对 TP_DirectedEdge 的关联）。

TP_Face::boundary():TP_FaceBoundary

注：对于外部的限制，用在拓扑中与用在几何中的含义一样。

TP_Face 也应该有一个带关联角色 boundary 的关联边界，该关联角色 boundary 定义了与有向边相同的信息，正定向为边的左侧，负定向为右侧。

TP_Face::boundary [1..*]:TP_DirectedEdge

由 boundary 操作符返回的附加信息是将 TP_FaceBoundary 组织成若干环 (ring)，并且指示哪一

个环为外环。

7.3.16.3 coBoundary (余边界)

对于 TP_Face, 定义在 TP_Object 里的“coBoundary”将返回对于有向拓扑体的引用的一个集合, 该集合指示哪些拓扑体在边界面上使用该面(正 TP_DirectedSolid)或该拓扑面的负代理(负 TP_DirectedSolid)。该操作重载了来自 TP_Object 的 coBoundary 操作。同样的信息可以实现为一个关联(即 TP_Face 的余边界实现为对 TP_DirectedSolid 的关联)。

```
TP_Face::coBoundary() [0..2]; Reference<TP_DirectedSolid>
```

```
TP_Face::coBoundary.spoke [0..2]; Reference<TP_DirectedSolid>
```

7.3.16.4 Center association (中心关联)

包含 TP_Face 的每个 TP_Primitive 被关联到两个 TP_DirectedTopo 实例上。

```
TP_Face::proxy [2]; TP_DirectedFace
```

```
TP_DirectedFace::topo [1]; Reference<TP_Face>
```

7.3.16.5 约束

TP_Face 的维数是 2。

```
TP_Face:
```

```
TP_Face: TP_Object::dimension = 2
```

7.3.17 TP_有向拓扑面(TP_DirectedFace)

TP_DirectedFace 用来定义 TP_Solid 的边界。它类似于 GM_OrientableSurface 的概念, 它的作用是在需要的时候, 作为基本的曲面/拓扑面的代理。

7.3.18 TP_拓扑体(TP_Solid)

7.3.18.1 语义

TP_Solid 类(见图 44)为 GM_Solid 提供拓扑单形。

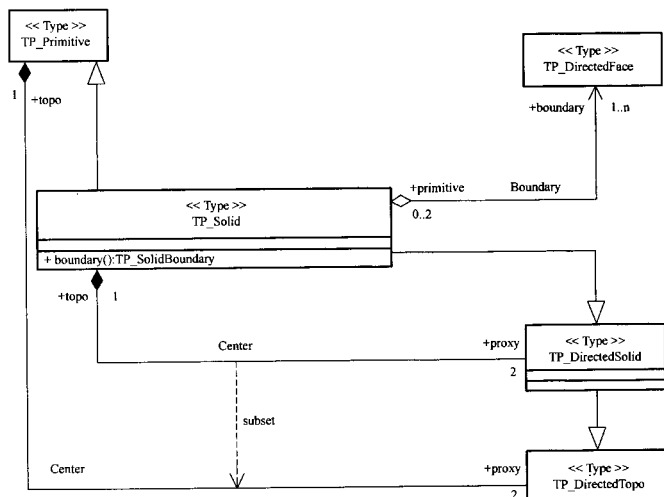


图 44 TP_拓扑体(TP_Solid)

7.3.18.2 boundary (边界)

对于 TP_Solid, 定义在 TP_Object 里的“boundary”操作将返回一个拓扑面或拓扑面的负代理的一

个组合。该操作重载了来自 TP_Object 类的 boundary。同样的信息可以表达为一个关联。

```
TP_Solid::boundary():TP_SolidBoundary
```

由 boundary 操作返回的附加信息是将 TP_SolidBoundary 组织成若干壳(shell),并且指示哪一个壳为外壳。

7.3.18.3 coBoundary (余边界)

对于 TP_Solid,“coBoundary”操作返回 NULL。

```
TP_Solid::coBoundary():NULL
```

7.3.18.4 Center association (中心关联)

每一个包含 TP_Solid 的 TP_Primitive 被关联到两个 TP_DirectedTopo 实例上。

```
TP_Solid::proxy [2]:TP_DirectedSolid
```

```
TP_DirectedSolid::topo [1]:Reference<TP_Solid>
```

7.3.18.5 约束

TP_Solid 的维数是 3。

```
TP_Solid:
```

```
TP_Object::dimension=3
```

7.3.19 TP_有向拓扑体(TP_DirectedSolid)

TP_DirectedSolid 类在计算拓扑类 TP_Expression 中支持 TP_Solid。

7.3.20 TP_表达式(TP_Expression)

7.3.20.1 语义

代数或计算拓扑是最容易被概念化为对多变量 1 阶多项式的操作,这里变量对应于 TP_Primitive。类 TP_DirectedTopo 以 TP_Expression 的代数方式来表达这些项。TP_Expression 类(见图 45)表达为多项式表达式。

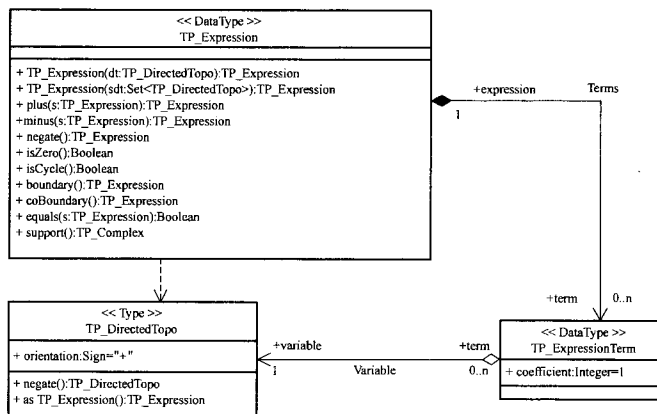


图 45 TP_表达式(TP_Expression)

多项式中项的次序并不影响它的值,所以 TP_Expression 类是由 Set<TP_DirectedTopo>类进行子类化而来的。TP_Expression 类的操作是用来构造、处理和测试这些“多项式”的。

计算拓扑的关键在于能以代数或组合的方式处理拓扑片段。这个代数中的单形是 TP_Primitive。单项式(单变量,单项的多项式)是 TP_Primitive 的实例,每一个项有一个整形的常数,实例化为 TP_ExpressionTerm(TP_表达式项)。

对于 TP_Expression,任何与多变量、一阶多项式代数相一致的约束都将将是有效的,例如:

TP_DirectedTopo:

```
negate(). asTP_Expression()=asTP_Expression(). negate()
asTP_Expression. negate(). Plus(asTP_Expression). isZero()
```

7.3.20.2 TP_ExpressionTerm (TP_表达式项)

像多项式一样,TP_Expression 是由若干项的一个集合构成,这些项由变量和常数构成。

```
TP_ExpressionTerm= $\langle$ coefficient; Integer=1,variable;  
Reference $\langle$ TP_DirectedTopo $\rangle\rangle$ 
```

其算法与一般的多项式操作的规则一致。

7.3.20.3 TP_Expression (constructor) (GM_表达式 (构造函数))

“TP_Expression”构造函数用 TP_DirectedTopo 来创建 TP_Expression。该操作将被其他类(例如 TP_Object)用来创建表达式。

```
TP_Expression(dt; TP_DirectedTopo); TP_Expression={ $\langle$ 1,dt $\rangle$ }  
TP_Expression(sdt; Set $\langle$ TP_DirectedTopo $\rangle$ ); TP_Expression={ $\langle$ 1,dt $\rangle$ |sdt.contains(dt)}
```

7.3.20.4 plus (加)

对于 TP_Expression,“plus”操作的作用如多项式的加法。它将采用加上它们的“orientation”系数的方法合并这些具有 TP_Primitive 相同潜在实例的 TP_DirectedTopo 元素。它将消除系数为 0 的项。

```
TP_Expression::plus(s; TP_Expression); TP_Expression
```

7.3.20.5 minus (减)

对于 TP_Expression,“minus”操作的作用如多项式的减法。它将采用减去它们的“orientation”系数的方法合并这些具有 TP_Primitive 相同潜在实例的 TP_DirectedTopo 元素。它将消除系数为 0 的项。

```
TP_Expression::minus(s; TP_Expression); TP_Expression
```

7.3.20.6 negate (取负)

“negate”操作对 TP_Expression 中的每一个项取负,它是多项式中的一元减去操作。

```
TP_Expression::negate(); TP_Expression
```

7.3.20.7 isZero (是零)

对于零多项式,“isZero”操作将返回 TRUE,它等价于“Set.IsEmpty”操作。

```
TP_Expression::isZero(); Boolean
```

7.3.20.8 isCycle (是圈)

对于边界(由 TP_Expression::boundary() 定义)是零的那些多项式,“isCycle”该操作将返回 TRUE。如果 TP_Expression 表达一个闭合的(closed)几何对象,例如多边形的边界,则它是一个圈(cycle)。在大多数 GIS 中,由“isCycle”返回的 TRUE 值意味着潜在的几何对象是另一个几何对象的边界。它等价于 isZero(boundary())操作。

```
TP_Expression::isCycle(); Boolean
```

注:一个边界的任何镜像操作是一个圈。这意味着 boundary(). boundary(). isZero()=TRUE。

7.3.20.9 boundary (边界)

“boundary”操作将在该 TP_Expression 中对每一个 TP_DirectedTopo 中的每一个 TP_Primitive 用它的边界取代,并且化简最后的表达式。边界始终由低一维的若干 TP_Primitive 构成。如果在该 TP_Expression 中所有的 TP_Primitive 的维数都是零(即所有的 TP_Primitive 都是结点),则该边界操作将返回一个为零的 TP_Expression。

```
TP_Expression::boundary(); TP_Expression
```

7.3.20.10 coBoundary (余边界)

“coBoundary”操作将该 TP_Expression 中每个 TP_DirectedTopo 的每个 TP_Primitive 用它的余边界取代,并且化简最后的表达式。注意余边界始终由高一维的若干 TP_Primitive 构成。如果在这个 TP_Expression 中所有的 TP_Primitive 的维数都相同于对应的最大 TP_Complex 的维数,则该 coBoundary 操作将返回一个为零的 TP_Expression。(例如,2 维中 2 个多边形相加,取余边界操作后变为 2 个体相加,但在 2 维中没有体,故是 2 个 NULL 相加,即为零的 TP_Expression。)

TP_Expression::coBoundary():TP_Expression

7.3.20.11 equals (相等)

对于一个与本拓扑表达式相等的多项式,“equals”操作将返回 TRUE,元素(项)的次序是无关紧要的。

TP_Expression::equals(s:TP_Expression):Boolean

7.3.20.12 support (支持)

“support”操作将映射该 TP_Expression 作为 TP_Primitive 的一个集合,以用于计算几何算子。该操作本质上是“asSet”操作,跟随着一个在 TP_DirectedTopo 和 TP_Primitive 间的 Center 关联的连接。

TP_Expression::support():Set<TP_Primitive>

7.3.20.13 asSet (作为集合)

“asSet”操作将映射该 TP_Expression 为 TP_DirectedTopo 的一个集合,以用于计算几何算子。该映射将包括添加所有的边界元素到该集合中,直到到达 TP_DirectedNode 为止。换句话说,TP_Expression 的 support 操作的返回值将是一个有效的 TP_Complex。

TP_Expression::asSet():Set<TP_DirectedTopo>

7.4 拓扑复形包

7.4.1 语义

“Topological complex”包提供附加的类用于创建 TP_Complex。

7.4.2 TP_复形(TP_Complex)

7.4.2.1 语义

本条包含拓扑复形的定义,它平行于在 6.6 中介绍的几何复形。拓扑复形(见图 46)可以根据它的元素作集合操作,以完成对于潜在的直接位置集的等价的集合操作,这些直接位置是由一个几何实现(即一个 GM_Complex)的若干几何元素所表达的。

7.4.2.2 TP_Complex (constructor) (GM_复形(构造函数))

拓扑复形缺省的构造函数使用一个几何复形。构造之后,该几何复形就是该拓扑复形的几何实现。只有那些由相互不连通的几何单形构成的几何复形才能没有错误地产生一个拓扑复形。

TP_Complex::TP_Complex(GC:GM_Complex):TP_Complex

对每一个几何复形使用缺省的构造函数定义一个缺省的拓扑复形,确保由该 TP_Complex 所表达的拓扑是该 GM_Complex 所表达的一个几何配置(configuration)的拓扑。“Realization”这个关联将跟踪该 TP_Complex 的每一个部分以回到该 GM_Complex 的适当部分。这允许我们可以在一个拓扑复形(TP_Complex)内谈论拓扑操作,就像它们是在一个几何复形(GM_Complex)中一样。

7.4.2.3 maximalComplex (最大复形)

私有属性“maximalComplex”包含对于该 TP_Complex 内唯一的最大拓扑复形的一个引用,本拓扑复形是这个最大拓扑复形的一个成员。这在编码中确定输出数据集的界限时需要。

TP_Complex::maximalComplex:Reference<TP_Complex>

7.4.2.4 isMaximal (是最大的)

如果该 TP_Complex 不再被更大的 TP_Complex 所包含,则布尔操作“isMaximal”返回 TRUE。

TP_Complex::isMaximal():Boolean

7.4.2.5 isConnected (是连通的)

如果该 TP_Complex 是拓扑上相互连通的,则“isConnected”操作返回 TRUE。

TP_Complex::isConnected(); Boolean

注:如果一个 TP_Complex 是连通的,则它的几何实现也是连通的。这并不意味着这是一个组合形(composite)(几何的或拓扑的),因为组合形必须符合与一个单形同构的这样一个很强的约束条件。为了测试一个拓扑复形是否连通,而又不参考其几何实现,就需要转而计算边界、余边界的闭包和孤立式关联(IsolatedIn association)。如果在该复形中的每一个单形都是通过这些关联角色的一个序列连接到该复形中的另一个单形上,这里每一个中间单形也都在该复形中,则该复形是连通的。

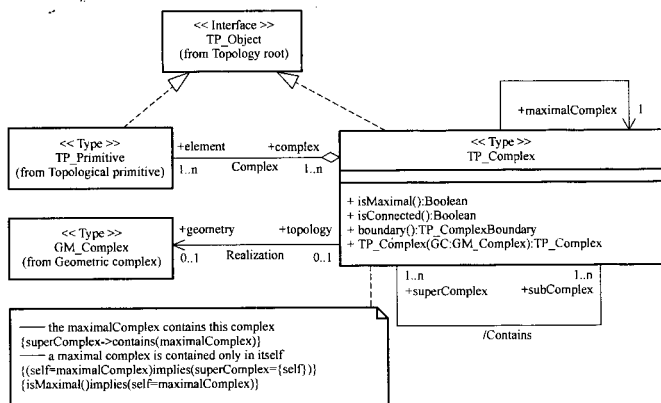


图 46 TP_复形(TP_Complex)

7.4.2.6 Contains association (包含关联)

派生关联“Contains”用来描述哪一个 TP_Complex 是作为 TP_Primitive 集合中的成员包含在该 TP_Complex 中。“superComplex(超复形)”角色是两个复形中较大的那一个,“subComplex(子复形)”角色是两个复形中较小的那一个。这个关系与从 Set(TP_Primitive)中继承来的“contains”操作相一致。

TP_Complex::subComplex [1..n]; Reference(TP_Complex)

TP_Complex::superComplex [1..n]; Reference(TP_Complex)

7.4.2.7 Complex association (复形关联)

“Complex”关联操作将使一些 TP_Primitive 元素与该 TP_Complex 发生关联。正是这个关联使得一个 Set(TP_Primitives) 成为 TP_Complex。通过“Contains”暗示的该集合操作与 TP_Complex 作为单形的一个集合的定义是一致的。

TP_Complex::element [1..n]; Reference(TP_Primitive)

TP_Primitive::complex [1..n]; Reference(TP_Complex)

7.4.2.8 Realization association (实现关联)

“realization”关联链接该 TP_Complex 到它对应的 GM_Complex 上(如果必要的话)。

TP_Complex::geometry [0,1]; GM_Complex

GM_Complex::topology [0,1]; TP_Complex

8 派生拓扑关系

8.1 概述

本章定义将拓扑关系作为查询算子的机制。可以用定义在 GM_Object 和它的子类上的集合理论操作以及定义在 TP_Expression 上的代数操作来计算这些查询算子。这两种机制对于作为对应于拓扑复形的几何实现的几何复形来说是等价的。本章所定义的算子主要用于查询评价,并且其定义确保算子的各种不同实现能从具有等效信息内容的数据集获得等效的结果。

本标准对于特定的空间操作符并不指派专门的名称。假设应用模式将使用下面三种分类技术中的任何一个或全部来定义其操作。在下面的用例中,分类模式基于 TP_Object。这里,对于 GM_Object 也定义相同的算子并给出一些限制,遵循上面的定义,由 GM_Object 的集合来创建 TP_Complex。接下来只是针对点、曲线、曲面和体对象。维数不同的对象聚集理论还不足以作为一个标准的基础。

查询系统如果与本标准这一部分一致,意味着该系统所支持的拓扑查询操作可以根据下面的描述来定义,即可以根据下面几条以及定义在本条的直接可用的操作符来定义,或通过对所支持的操作符的一个易于理解的组合来定义。对于本章最低的一致性要求是:

- 布尔查询操作的定义与所包含的子项相一致。
- 在 8.2、8.3 或 8.4 的一个或多个上下文环境中所有可定义的有效布尔算子都是有效可用的。

完整的一致性要求支持本章整个上下文环境中的所有可定义的有效布尔操作。

8.2 布尔或集合算子

8.2.1 布尔算子的形式

集合理论算子有时也称为布尔算子。因为这种算子不区分集合的内部与边界,所以 closure(闭包)操作可用来合并它们。

$GM_Object::closure() == interior().union(boundary())$

对于两个对象 A 和 B,可以做下面 4 个交操作:

$intersection(closure(A), closure(B)) \quad intersection(closure(A), exterior(B))$
 $intersection(exterior(A), closure(B)) \quad intersection(exterior(A), exterior(B))$

可以测试集合的矩阵,看每一个集合是空还是非空。这里将 A 和 B 的关系分为 2^4 , 即 16 种组合中的一种。

一个操作可以被定义为一个模板(template),用于测试交集矩阵中两个对象间特定的空间关系。该模板是具有 4 个扩展的布尔值的一个矩阵,对扩展的布尔值的解释由表 8 给出。有 3^4 , 即 81 种可能的算子模板。

表 8 布尔交范型(pattern)矩阵的意义

符 号	是否非空	意 义
T	TRUE	矩阵在该位置的交是非空。
F	FALSE	矩阵在该位置的交是空。
N	NULL	该操作无法测试矩阵在该位置的交。

注:值 TRUE 意味着交的集合是非空(见列的顶栏)。

为了测试两个对象是否具有与特定的算子模板一致的关系,根据矩阵中的范型,这个交不与计算与测试的 NULL 值相联系(即两个对象对这些算子的“交”测试不会为 NULL)。如果关系相一致,则对这两个对象的测试返回值是 TRUE,否则是 FALSE。

8.2.2 布尔相关

通过对两个由“交范型矩阵(intersectionPatternMatrix)”中的值所控制的几何对象的闭包与外部的测试,如果这些对象是空间相关的,则“bRelate”算子返回 TRUE。

0

Boolean bRelate(GM_Object, GM_Object, intersectionPatternMatrix)

Boolean bRelate(TP_Object, TP_Object, intersectionPatternMatrix)

“交范型矩阵”由主表的行给出,即由一个由 4 个字符构成的字符串给出,这 4 个字母可能的取值为 T、F、N。对于矩阵中第一行有两个值,接下来第二行有两个值。

8.2.3 与集合操作的关系

“Boolean relate(布尔相关)”可以被用来实现定义在 6.2.2.18 中的 GM_Object 的“contains”、“intersects”和“equals”操作。

例如:

C:GM_Composite, G:GM_Object;

C.contains(G)=bRelate(C,G,"TNFT");

8.3 Egenhofer 算子

8.3.1 Egenhofer 算子的形式

这项技术来自缅因大学的 Egenhofer(埃根霍菲尔)教授(见参考文献[8]、[9])。对于两个对象 A 和 B,可以做下面的 9 交操作。

intersection(boundary(A), boundary(B))	intersection(boundary(A), interior(B))
intersection(boundary(A), exterior(B))	Intersection(interior(A), boundary(B))
intersection(interior(A), interior(B))	intersection(interior(A), exterior(B))
Intersection(exterior(A), boundary(B))	intersection(exterior(A), interior(B))
intersection(exterior(A), exterior(B))	

可以测试这个集合矩阵(叫作 9 交矩阵),检测每一个“交”是“空”还是“非空”。它将 A 和 B 的关系分类为 2^2 , 即 512 种组合中的一种。实际上,并非这所有的 512 种组合在几何上都是可能的,但这并不妨碍它的使用。

一个操作可以被定义为一个用于 9 交矩阵模板,以测试两个对象间特定的空间关系。该模板是一个具有 9 个扩展布尔值的矩阵,其解释由表 9 给出,表的内容与前一个表一致。有 3^9 , 即 19 683 种可能的算子模板。

表 9 Egenhofer 9 交范型矩阵的含义

符 号	是否非空	意 义
T	TRUE	矩阵在该位置的交是非空。
F	FALSE	矩阵在该位置的交是空。
N	NULL	该操作无法测试矩阵在该位置的交。

为了测试两个对象对一个特定算子的一致关系,根据这个矩阵中的范型,计算与测试这些“交”是否“非空”,这些交不与 NULL 值相联系(即两个对象对这些算子的“交”测试不会为 NULL)。如果一致,则算子对于这两个对象的测试返回值是 TRUE,否则是 FALSE。

8.3.2 Egenhofer 相关

通过对两个由“交范型矩阵”中的值所控制的几何对象之间 interior, boundary 和 exterior 关系的测试,如果这两个对象是空间相关的,则“eRelate(Egenhofer 相关)”算子返回 TRUE。

Boolean eRelate(GM_Object, GM_Object, intersectionPatternMatrix)

Boolean eRelate(TP_Object, TP_Object, intersectionPatternMatrix)

“交范型矩阵”是作为一个串在主表的行中列出,这个串有 9 个字母(每一个字母是 T、F 或 N)。

8.3.3 与集合操作的关系

“Egenhofer 相关”可以用来实现定义在 6.2.2.18 中 GM_Object 的“contains(包含)”、“intersects(相交)”和“equals(相等)”等操作。

例如:

```
C:GM_Primitive,G:GM_Primitive;
C.contains(G)=eRelate(C,G,"NFNNTNNFT");
C:GM_Primitive,G:GM_Composite;
C.contains(G)=eRelate(C,G,"FFNTTNFFT");
```

8.4 全拓扑算子

8.4.1 全拓扑算子的形式

全拓扑算子考虑了不同维的差异(见参考文献[4]、[5]和这一部分扩展的分析)在方式上类似于 Egenhofer 算子,但是在可能值上作了更精细的区别。

表 10 全拓扑交范型矩形的含义

符 号	是否非空	含 义
0	TRUE	矩阵这个位置的交只包含点
1	TRUE	矩阵这个位置的交只包含点和曲线
2	TRUE	矩阵这个位置的交只包含点、曲线和曲面
3	TRUE	矩阵这个位置的交只包含点、曲线、曲面和体
F	FALSE	矩阵这个位置的交为空
N	NULL	该操作无法测试矩阵在该位置的交

为了测试两个对象是否与 $6^9=10\ 077\ 696$ 种算子模板中的一种相一致,根据该范型矩阵,对于非空和维数的计算与测试的“交”不与 NULL 值相联系。如果存在一致性,对这两个对象的操作的值是 TRUE,如果不一致则为 FALSE。

8.4.2 全拓扑相关

通过对两个由“交范型矩阵”中的值所控制的几何对象之间内部、边界、外部关系的测试,如果这两个对象是空间相关的,则“cRelate”算子返回 TRUE。

```
Boolean cRelate(GM_Object,GM_Object,intersectionPatternMatrix)
```

```
Boolean cRelate(TP_Object,TP_Object,intersectionPatternMatrix)
```

“交范型矩阵”是作为一个串在主表的行中列出,这个串有 9 个字母(每一个字母是 0、1、2、3、F、N 中的一个)。

8.5 组合

算子可以定义为前面章节的基本操作的一个或多个的任意布尔组合。

附录 A
(规范性附录)
抽象测试套件

A.1 几何单形

A.1.1 几何单形的数据类型

A.1.1.1 0 维几何单形的数据类型

- a) 测试目的:检验应用模式或专用标准的实例 GM_Point 是否具有 position 属性和相关联的从 GM_Object 继承来的坐标参照系。如果应用模式或专用标准也实例化 GM_MultiPoint,检验它是否包含属性 position 和对 element 的关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:6.1、6.2.1、6.2.2.17、6.3.10.1、6.3.11.1、6.3.11.2、6.4.1、6.5.1、6.5.2.1、6.5.2.2、6.5.3、6.5.4。
- d) 测试类型:能力测试。

A.1.1.2 1 维几何单形的数据类型

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.1 的所有要求以及实例化的 GM_Curve 是否具有 orientation 属性和 segmentation 关联以及至少一个具有它的所有属性的可实例化的 GM_CurveSegment 的子类。如果应用模式或专用标准也实例化 GM_MultiCurve,检验它是否包含 element 和 length 属性。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.1、6.3.5、6.3.13、6.3.14.1、6.3.16、6.4.1、6.4.6、6.4.8~6.4.31、6.5.5。
- d) 测试类型:能力测试。

A.1.1.3 2 维几何单形的数据类型

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.2 的所有要求,以及实例化的 GM_Surface 是否具有 orientation 属性和 interiorTo、segmentation 关联,以及至少一个具有它的所有属性的可实例化的 GM_SurfacePatch 子类。如果应用模式或专用标准也实例化 GM_MultiSurface,检验它是否包含的 element、area 和 perimeter 等属性。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.2、6.3.6、6.3.7、6.3.10.4、6.3.15、6.3.17.1、6.3.17.3、6.4.6、6.4.32~6.4.48、6.5.6。
- d) 测试类型:能力测试。

A.1.1.4 3 维几何单形的数据类型

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.3 的所有要求,以及实例化的 GM_Solid 是否具有 interiorTo 关联。如果应用模式或专用标准也实例化 GM_MultiSolid,检验它是否包含 element、area 和 volume 等属性。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.2、6.3.6、6.3.7、6.3.10.4、6.3.15、6.3.17.1、6.3.17.3、6.4.6、6.4.32~6.4.48、6.5.6。
- d) 测试类型:能力测试。

A.1.2 几何单形的简单操作

A.1.2.1 0 维几何单形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.1 的所有要求以及 GM_Point 的实例是否包含 boundary、mbRegion 和 representativePoint 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用: A.1.1.1、6.2.2.2、6.2.2.3、6.2.2.4、6.3.10.2、6.3.11.3。
- d) 测试类型:能力测试。

A.1.2.2 1 维几何单形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.2 和 A.1.2.1 的所有要求以及 GM_Curve 的实例是否包含 boundary、mbRegion 和 representativePoint 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用: A.1.1.2、A.1.2.1、6.2.2.2、6.2.2.3、6.2.2.4、6.3.10.2、6.3.14.2。
- d) 测试类型:能力测试。

A.1.2.3 2 维几何单形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.3 和 A.1.2.2 的所有要求以及 GM_Surface 的实例化,这包含 boundary、mbRegion 和 representativePoint 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用: A.1.1.3、A.1.2.2、6.2.2.2、6.2.2.3、6.2.2.4、6.3.10.2、6.3.15.2。
- d) 测试类型:能力测试。

A.1.2.4 3 维几何单形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.4 和 A.1.2.3 的所有要求以及 GM_Solid 的实例是否包含 boundary、mbRegion 和 representativePoint 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用: A.1.1.4、A.1.2.3、6.2.2.2、6.2.2.3、6.2.2.4、6.3.10.2、6.3.18.2。
- d) 测试类型:能力测试。

A.1.3 几何单形的完整操作

A.1.3.1 0 维几何单形的完整操作

- a) 测试目的:检验应用模式或专用标准实例化的 GM_Point 和 GM_MultiPoint 是否具有由这两个类所定义的以及从 GM_Object 和 GM_Primitive 继承来的除 Complex 关联和 maximalComplex 操作以外的所有属性、操作和关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用: 6.1.6.2、6.3.10.2、6.3.11.2、6.3.12.2、6.4.1~6.4.5、6.5.1~6.5.4。
- d) 测试类型:能力测试。

A.1.3.2 1 维几何单形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.3.1 的所有要求,以及它是否直接或通过一个非抽象的子类实例化 GM_Curve、GM_CurveSegment 和 GM_MultiCurve。检验这些实例是否支持专门为这些类定义的以及从 GM_Object、GM_GenericCurve 和 GM_Primitive 继承来的除了 Complex 关联和 maximalComplex 操作以外的所有的属性、操作和关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用: A.1.3.1、6.3.1.2、6.3.2.2、6.3.4.2、6.3.5.2、6.3.13.2、6.3.14.2、6.3.16.2、6.4.6~6.4.31、6.5.5。
- d) 测试类型:能力测试。

A.1.3.3 2 维几何单形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.3.2 的所有要求,以及它是否直接或通过

④

一个非抽象的子类实例化 GM_Surface、GM_SurfacePatch 和 GM_MultiSurface。检验这些实例是否支持专门为这些类定义的以及从 GM_Object、GM_GenericSurface 和 GM_Primitive 继承来的除了 Complex 关联和 maximalComplex 操作以外的所有的属性、操作和关联。

- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.3.2、6.3.6、6.3.7、6.3.15、6.3.17、6.4.32~6.4.48、6.5.6。
- d) 测试类型:能力测试。

A.1.3.4 3维几何单形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.3.3 的所有要求,以及它是否直接或通过一个非抽象的子类实例化 GM_Solid 和 GM_MultiSolid。检验这些实例是否支持专门为这些类定义的以及从 GM_Object、GM_GenericSolid 和 GM_Primitive 继承来的除了 Complex 关联和 maximalComplex 操作以外的所有的属性、操作和关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.3.3、6.3.8、6.3.18、6.5.7。
- d) 测试类型:能力测试。

A.2 几何复形

A.2.1 几何复形的数据类型

A.2.1.1 1维几何复形的数据类型

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.2 的所有要求,以及它实例化的 GM_Complex、GM_CompositePoint 和 GM_CompositeCurve。检验它是否支持在(GM_Primitive)集合与 GM_Complex 之间的 Contains 关联,在 GM_Primitives(GM_Point 和 GM_Curve)与 GM_Complex 之间的 Complex 关联,在 GM_Point 与 GM_CompositePoint 之间,GM_Curve 和 GM_CompositeCurve 之间的 Contains 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.2、6.3.6、6.1.6、6.2.1、6.6.2.3、6.6.2.4、6.6.3~6.6.5。
- d) 测试类型:能力测试。

A.2.1.2 2维几何复形的数据类型

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.3 和 A.2.1.1 的所有要求,以及它实例化的 GM_CompositeSurface 是否具有在 GM_Surface 和 GM_Complex 之间的 Complex 关联,以及 GM_Surface 和 GM_CompositeSurface 之间的 Composition 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.3、A.2.1.1、6.6.6。
- d) 测试类型:能力测试。

A.2.1.3 3维几何复形的数据类型

- a) 测试目的:检验应用模式或专用标准满足 A.1.1.4 和 A.2.1.2 的所有要求,以及它实例化的 GM_CompositeSolid 是否具有在 GM_Solid 和 GM_Complex 间的 Complex 关联,以及 GM_Solid 和 GM_CompositeSolid 间的 Composition 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.4、A.2.1.2、6.6.7。
- d) 测试类型:能力测试。

A.2.2 几何复形的简单操作

A.2.2.1 1维几何复形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.2.2 和 A.2.1.1 的所有要求,检验它实例

化的 GM_CompositeSolid 和 GM_CompositeCurve 是否支持 boundary、envelope、representative point、maximal、isMaximal 等操作。

- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.2.2、A.2.1.1、6.6.2.2。
- d) 测试类型:能力测试。

A.2.2.2 2 维几何复形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.2.3、A.2.1.2 和 A.2.2.1 的所有要求,检验它实例化的 GM_CompositeSurface 是否支持 boundary、envelope、representative point、isMaximal 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.2.3、A.2.1.2、A.2.2.1。
- d) 测试类型:能力测试。

A.2.2.3 3 维几何复形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.2.4、A.2.1.3 和 A.2.2.1 的所有要求,检验它实例化的 GM_CompositeSolid 是否支持 boundary、envelope、representative point、isMaximal 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.2.4、A.2.1.3、A.2.2.1。
- d) 测试类型:能力测试。

A.2.3 几何复形的完整操作

A.2.3.1 1 维几何复形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.3.2 和 A.2.2.1 的所有要求。检验 GM_CompositePoint、GM_CompositeCurve 的实例是否支持专门为这些类所定义的以及那些从 GM_Object、GM_Complex、GM_Composite 继承来的所有的属性、操作和关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.3.2、A.2.2.1。
- d) 测试类型:能力测试。

A.2.3.2 2 维几何复形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.3.3、A.2.2.2 的所有要求。检验 GM_CompositeSurface 的实例是否支持专门为该类所定义的以及那些从 GM_Object、GM_Complex、GM_Composite 继承来的所有的属性、操作和关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.3.3、A.2.2.2。
- d) 测试类型:能力测试。

A.2.3.3 3 维几何复形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.3.4、A.2.2.3 的所有要求。检验 GM_CompositeSolid 的实例是否支持专门为该类所定义的以及那些从 GM_Object、GM_Complex、GM_Composite 继承来的所有的属性、操作和关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.3.4、A.2.2.3。
- d) 测试类型:能力测试。

A.3 拓扑复形

A.3.1 拓扑复形的数据类型

A.3.1.1 1 维拓扑复形的数据类型

- a) 测试目的: 检验应用模式或专用标准实例化的 TP_Complex、TP_Node、TP_DirectedNode、TP_Edge 和 TP_DirectedEdge。检验 TP_DirectedNode 和 TP_DirectedEdge 的实例是否支持 orientation 属性。检验应用模式或专用标准是否支持在 TP_Complex 和每一个 TP_Primitives(TP_Node 和 TP_Edge)之间的 Complex 关联。检验它是否支持在 TP_Node 与 TP_DirectedNode 间以及 TP_Edge 与 TP_DirectedEdge 间的 Center 关联。检验它是否支持在 TP_DirectedNode 与 TP_Edge 之间的派生关联 Boundary 以及 TP_Node 与 TP_DirectedEdge 之间的派生关联 Coboundary。
- b) 测试方法: 检查应用模式或专用标准的文件。
- c) 引用: 7.1.7.2.1、7.3.1.7.3.2、7.3.3.7.3.8.1、7.3.8.3.7.3.9.1、7.3.9.2.7.3.9.5、7.3.9.6.7.3.10.7.3.11.7.3.12.1.7.3.12.3.7.3.12.4.7.3.13.7.4.1.7.4.2.1.7.4.2.3.7.4.2.6.7.4.2.7。
- d) 测试类型: 能力测试。

A.3.1.2 2 维拓扑复形的数据类型

- a) 测试目的: 检验应用模式或专用标准满足 A.3.1.1 的全部要求以及它实例化的 TP_Face、TP_DirectedFace。检验 TP_DirectFace 的实例是否支持 orientation 属性。检验应用模式或专用标准是否支持在不同 TP_Primitive 间的 IsolatedIn 关联, TP_Complex 和 TP_Face 间的 Complex 关联, 以及 TP_Face 与 TP_DirectedFace 之间的 Center 关联。检验它是否支持 TP_DirectedEdge 与 TP_Face 间的 Boundary 关联, 以及 TP_Edge 和 TP_DirectedFace 间的 Coboundary 关联。
- b) 测试方法: 检查应用模式或专用标准的文件。
- c) 引用: A.3.1.1、7.3.4.7.3.6.7.3.8.4.7.3.14.1.7.3.14.4.7.3.14.5。
- d) 测试类型: 能力测试。

A.3.1.3 3 维拓扑复形的数据类型

- a) 测试目的: 检验应用模式或专用标准是否满足 A.3.1.2 的全部要求以及它实例化的 TP_Solid、TP_DirectedSolid。检验 TP_DirectedSolid 的实例是否支持 orientation 属性。检验应用模式或专用标准是否支持在 TP_Complex 与 TP_Solid 之间的 Complex 关联以及 TP_Solid 与 TP_DirectedSolid 间的 Center 关联。检验它是否支持在 TP_DirectedFace 与 TP_Solid 间的 Boundary 关联, 以及 TP_Face 与 TP_DirectedSolid 间的 Coboundary 关联。
- b) 测试方法: 检查应用模式或专用标准的文件。
- c) 引用: A.3.1.2、7.3.5.7.3.7.7.3.16.1.7.3.16.4.7.3.16.5。
- d) 测试类型: 能力测试。

A.3.2 拓扑复形的简单操作

A.3.2.1 1 维拓扑复形的简单操作

- a) 测试目的: 检验应用模式或专用标准是否满足 A.3.1.1 的全部要求以及 TP_Complex、TP_Node、TP_Edge 的实例是否都支持 boundary、coboundary、maximalComplex 等操作。检验 TP_Complex 的实例是否支持 isMaximal 操作。
- b) 测试方法: 检查应用模式或专用标准的文件。
- c) 引用: A.3.1.1、7.2.2.3.7.2.2.4.7.2.2.8.7.3.12.2.7.4.2.4。
- d) 测试类型: 能力测试。

A.3.2.2 2 维拓扑复形的简单操作

- a) 测试目的:检验应用模式或专用是否标准满足 A.3.1.2 和 A.3.2.1 的全部要求以及 TP_Face 的实例是否支持 boundary、coboundary、maximal 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.3.1.2、A.3.2.1、7.3.14.2、7.4.14.3。
- d) 测试类型:能力测试。

A.3.2.3 3 维拓扑复形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.3.1.3 和 A.3.2.2 的全部要求以及 TP_Solid 的实例是否支持 boundary、coboundary、maximal 等操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.3.1.3、A.3.2.2、7.3.16.2、7.3.16.3。
- d) 测试类型:能力测试。

A.3.3 拓扑复形的完整操作**A.3.3.1 1 维拓扑复形的完整操作**

- a) 测试目的:检验应用模式或专用标准实例化的 TP_Complex、TP_Expression、TP_ExpressionTerm、TP_Node、TP_DirectedNode、TP_Edge、TP_DirectedEdge。检验这些实例是否支持除 Realization 关联外的由这些类所定义以及从 TP_Object、TP_Primitive、TP_DirectedTopo 继承而来的所有的属性、关联和操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:7.1.7.2、7.3.1、7.3.2、7.3.3、7.3.6、7.3.8.1、7.3.8.3、7.3.8.4、7.3.9~7.3.13、7.3.18、7.4.1、7.4.2.1~7.4.2.7。
- d) 测试类型:能力测试。

A.3.3.2 2 维拓扑复形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.3.3.1 的全部要求以及它实例化的 TP_Face、TP_DirectedFace。检验 TP_Face 和 TP_DirectedFace 的实例是否支持除 Realization 关联外的由这些类所定义的以及从 TP_Object、TP_Primitive、TP_DirectedTopo 继承而来的所有的属性、关联和操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.3.3.1、7.3.4、7.3.14、7.3.15。
- d) 测试类型:能力测试。

A.3.3.3 3 维拓扑复形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.3.3.2 的全部要求以及它实例化的 TP_Solid、TP_DirectedSolid。检验 TP_Solid 和 TP_DirectedSolid 的实例是否支持除 Realization 关联外的由这些类所定义以及从 TP_Object、TP_Primitive、TP_DirectedTopo 继承而来的所有的属性、关联和和操作。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.3.3.2、7.3.5、7.3.16、7.3.17。
- d) 测试类型:能力测试。

A.4 具有几何实现的拓扑复形**A.4.1 具有几何实现的拓扑复形的数据类型****A.4.1.1 具有几何实现的 1 维拓扑复形的数据类型**

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.2、A.3.1.1 的全部要求。检验它是否也

q.

支持在 TP_Primitives(TP_Node,TP_Edge) 实例与 GM_Primitives(GM_Point,GM_Curve) 实例间以及 TP_Complexes 实例与 GM_Complexes 实例间的 Realization 关联。

- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.2、A.3.1.1、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.1.2 具有几何实现的 2 维拓扑复形的数据类型

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.3、A.3.1.2 的全部要求以及它是否支持 TP_Face 的实例和 GM_Surface 的实例间的 Realization 关联以及 TP_Complexes 与 GM_Complexes 实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.3、A.3.1.2、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.1.3 具有几何实现的 3 维拓扑复形的数据类型

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.1.4、A.3.1.3 的全部要求以及它所支持的 TP_Solid 实例和 GM_Solid 实例间以及 TP_Complexes 实例与 GM_Complexes 实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.1.4、A.3.1.3、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.2 具有几何实现的拓扑复形的简单操作

A.4.2.1 具有几何实现的 1 维拓扑复形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.2.2、A.3.2.1 的全部要求,以及它是否满足支持在 TP_Primitive(TP_Node,TP_Edge)实例和 GM_Primitive(GM_Point,GM_Curve) 实例间,以及 TP_Complexes 实例和 GM_Complexes 实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.2.2、A.3.2.1、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.2.2 具有几何实现的 2 维拓扑复形的简单操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.1.2.3、A.3.2.2 的全部要求,以及它是否支持在 TP_Face 实例和 GM_Surface 实例间的 Realization 关联以及在 TP_Complexes 实例和 GM_Complexes 实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.2.3、A.3.2.2、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.2.3 具有几何实现的 3 维拓扑复形的简单操作

- a) 测试目的:检验应用模式或专用标准满足 A.1.2.4、A.3.2.3 的全部要求,以及它是否支持在 TP_Solid 实例与 GM_Solid 实例间以及 TP_Complexes 实例与 GM_Complexes 实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.1.2.4、A.3.2.3、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.3 具有几何实现的拓扑复形的完整操作**A.4.3.1 具有几何实现的1维拓扑复形的完整操作**

- a) 测试目的:检验应用模式或专用标准是否满足 A.2.3.1、A.3.3.1 的全部要求并且支持在 TP_Primitive(TP_Node, TP_Edge)实例与 GM_Primitive(GM_Point, GM_Curve)实例间以及 TP_Complexes 实例与 GM_Complexes 实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.2.3.1、A.3.3.1、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.3.2 具有几何实现的2维拓扑复形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.2.3.2、A.3.3.2 的全部要求并且支持在 TP_Face 的实例和 GM_Surface 的实例间以及 TP_Complexes 的实例和 GM_Complexes 的实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.2.3.1、A.3.3.1、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.4.3.3 具有几何实现的3维拓扑复形的完整操作

- a) 测试目的:检验应用模式或专用标准是否满足 A.2.3.3、A.3.3.3 的全部要求并且支持在 TP_Solid 的实例和 GM_Solid 的实例间以及 TP_Complexes 的实例和 GM_Complexes 的实例间的 Realization 关联。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.2.3.1、A.3.3.1、7.3.8.2、7.4.2.8。
- d) 测试类型:能力测试。

A.5 派生拓扑算子**A.5.1 布尔算子**

- a) 测试目的:检验应用模式或专用标准是否定义了与规定在 8.2 中的子项相一致的布尔算子。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:8.2。
- d) 测试类型:能力测试。

A.5.2 Egenhofer 算子

- a) 测试目的:检验应用模式或专用标准是否定义了与规定在 8.3 中的子项相一致的Egenhofer 算子。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:8.3。
- d) 测试类型:能力测试。

A.5.3 全拓扑算子

- a) 测试目的:检验应用模式或专用标准是否定义了与规定在 8.4 中的子项相一致的全拓扑算子。
- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:8.4。
- d) 测试类型:能力测试。

A.5.4 所有的拓扑算子

- a) 测试目的:检验应用模式或专用标准是否定义了与规定在第 8 章中相一致的所有的拓扑算子。

- b) 测试方法:检查应用模式或专用标准的文件。
- c) 引用:A.5.1、A.5.2、A.5.3。
- d) 测试类型:能力测试。

附录 B

(资料性附录)

按概念分类组织的术语与定义

B.1 引言

该附录按照它们的概念关系的排列陈述从第4章来的术语与定义。

B.2 通用术语

应用 application (4.1)

为支持用户的需求对数据的操作与处理。

[ISO 19101:2002]

应用模式 application schema (4.2)

对于一个或多个应用的数据需求的概念模式。

[ISO 19101:2002]

边界 boundary (4.4)

表达一个实体的界限的集合。

注：边界通常用在几何结构，这里的集合是点的集合或代表这些点的对象的集合。在别的场合，边界这个词常隐喻在实体与其他部分之间的一个跃迁。

要素 feature (4.39)

对现实世界现象的抽象。

注：要素可以类型或实例的形式出现。当只表达一种含义时，应使用要素类型或要素实例。

[ISO 19101:2002]

要素属性 feature attribute (4.40)

要素的特性。

示例1：一个名为“颜色”的要素属性有一个属性值为“绿色”，其数据类型为“文本”。

示例2：一个名为“长度”的要素属性有一个属性值为“82.4”，其数据类型为“实型”。

注1：要素属性包括名称、数据类型和与其相关的值域。要素实例的要素属性具有一个来自其值域的属性值。

注2：在要素目录中，要素属性可以包括一个值域，但并不指定要素实例的属性值。

[ISO 19101:2002]

地理信息 geographic information (4.42)

与地球上的位置直接或间接相关的现象的信息。

[ISO 19101:2002]

空间对象 spatial object (4.69)

用于表达要素空间特征的对象。

空间算子 spatial operator (4.70)

函数或程序，在它的域或范围内至少有一个空间参数。

注：UML对空间对象的任何操作都可以归为空间算子，例如本标准中第8条的查询算子。

B.3 集合与相关术语

集[合] set (4.65)

不重复的相关项(对象或值)的无序汇集。

序列 sequence (4.64)

可重复的相关项(对象或值)的有限、有序集合。

注: 逻辑上, 序列是<项, 偏移量>对的集合。它原是 LISP 语言中一种用括号括起来, 有次序限定, 元素间用逗号隔开的结构, 现用在本标准中。

包 bag (4.3)

可以重复的相关项(对象或值)的有限、无序集合。

注: 逻辑上, 包是<项, 数目>对的集合。

循环序列 circular sequence (4.6)

没有逻辑起始, 因而等同于任何自身循环移位的序列, 因此可认为序列的最后一项位于该序列的第一项之前。

记录 record (4.62)

有限的、有名称的相关项(对象或值)的集合。

注: 逻辑上, 记录是<名称, 项>对的集合。

域 domain (4.32)

定义明确的集合。

注: 域用以定义域集以及属性、算子和函数的范围。

函数 function (4.41)

从一个域(源或定义域)中的每一个元素到另一个域(目标域、从变量域、值域)中唯一元素相关联的规则。

B.4 建模术语

类 class (4.7)

具有相同的属性(域)、操作、方法、关系与语义的对象集的描述。

注: 类可以用接口的集合来定义它所提供给它的外界(环境)的操作集。该术语过去最先在面向对象的编程中使用, 后来被 UML(统一建模语言)所采纳, 并具有相同的含义。

[ISO/TS 19103:2005]

对象 object (4.59)

具有明确定义的边界与标识, 封装了状态与行为的实体。

注: 该术语最先是用在面向对象编程的一般理论中, 后来以相同的含义被 UML 所采纳。一个对象是一个类(class)的实例, 属性与关系表达状态。操作、方法和状态机制表达行为。

实例 instance (4.53)

一个类的一个实现, 即对象。

强可替换性 strong substitutability (4.73)

继承或实现另一个类、类型或接口的派生类的任何实例在任何相关环境中用于替代其祖先实例的能力。

注: 弱可替换类型对于要进行替换的环境有各种限制。

B.5 位置术语

直接位置 direct position (4.26)

用坐标参照系中的一组坐标描述的位置。

坐标 coordinate (4.19)

表示 n 维空间中点位置的 n 个序列数之一。

注: 在一个坐标参照系下, 坐标数值取决于所选单位。

[ISO 19111:2007]

坐标参照系 coordinate reference system (4.21)

通过基准与现实世界相联系的坐标系。

[ISO 19111:2007]

坐标系 coordinate system (4.22)

给点赋予坐标的数学规则集。

[ISO 19111:2007]

坐标维数 coordinate dimension (4.20)

在一个坐标系中,描述一个位置所需要的量测或坐标个数。

B.6 几何术语

B.6.1 普通几何概念

矢量几何 vector geometry (4.86)

用构造的几何单形表达的几何。

计算几何 computational geometry (4.13)

几何表达式的处理与计算,用于几何操作的实现。

示例:计算几何操作包括几何对象间的几何包含或相交测试、凸包或缓冲区计算、最短路径搜索。

几何集 geometric set (4.50)

直接位置的集合。

注:在大多数情况下,这个集合是无限的。

凸集 convex set (4.18)

一种几何集,在该几何集中连接任意两个直接位置的直线段上的任意直接位置都包含在该几何集中。

注:凸集间是“简单连接”,这意味着它们没有内部孔洞,并且通常可以认为在拓扑上是与适当维数的欧几里德球同构的。所以,球的表面可以被认为几何凸集。

凸壳 convex hull (4.17)

包含所给几何对象的最小凸集。

注:“最小”是集合理论上的最小,并不是指尺度上。这个定义也可以重写为“包含所给几何元素的所有凸集的并集”。

邻域 neighborhood (4.57)

几何集,在它的内部包含有一个指定的直接位置和距该直接位置的距离都在一个给定值之内的所有直接位置。

几何维数 geometric dimension (4.46)

在几何集内每一个直接位置都可以与其内部具有该直接位置的子集相关联,且相似(同构)于 n 维欧几里德空间 R^n 的最大数目 n 。

注:曲线的几何维数为1,因为它们实线段的连续映像。虽然曲面不能将它整体映射到2维空间 R^2 中,但围绕每一个点的位置,(在连续函数的情况下)可以找到一个类似于 R^2 单位圆的小邻域,因此曲面是2维的。在本标准中,大多数曲面片(几何曲面片 GM_SurfacePatch 的实例)通过它们所定义的插值机制映射为 R^2 中的一部分。

B.6.2 几何对象

B.6.2.1 一般概念

几何对象 geometric object (4.47)

表达一个几何集的空间对象。

0

注：几何对象由几何单形、几何单形的组合或处理为一个单独实体的几何复形构成。几何对象可以是作为要素的对象的空间表达，或要素的一个有意义的部分。

几何边界 geometric boundary (4.44)

由约束几何对象范围的几何维数更低的几何单形的集合所表达的边界。

圈 cycle (4.25)

<几何学>没有边界的空间对象。

注：圈是用来描述边界成分(见亮与环)。环没有边界，是因为它自身闭合，但它是有界的(即它不具有无限的外延)。例如圆环或球，没有边界，但却是有界的。

内部 interior (4.54)

几何对象除边界外的所有直接位置的集合。

注：拓扑对象的内部是与其相对应的几何实现的内部的同构映像。因为它遵从拓扑定理，所以不再定义。

外部 exterior (4.37)

全域与闭包之差。

注：外部这个概念适用于拓扑与几何复形。

闭包 closure (4.8)

拓扑对象或几何对象的内部与边界的并集。

简单 simple (4.67)

几何对象的一种特性，其内部是等向的(所有点都具有同构的邻域)，因而它处处都局部同构于适当维数欧几里德坐标空间的开子集。

注：这暗示几何对象内部的直接位置不会出现任何形式的自相交。

连通的 connected (4.15)

几何对象的一种特性，指该对象内的任何两个直接位置能置于完全在该对象内的一条曲线上。

注：当且仅当一个拓扑对象的所有几何实现是连通的，该拓扑对象才是连通的。由于它遵从拓扑定理，所以未写在定义内。

缓冲区 buffer (4.5)

包含距一个指定几何对象的距离都小于或等于一个给定值的所有直接位置的几何对象。

几何聚集形 geometric aggregate (4.43)

没有内部结构的几何对象的集合。

注：不对这些元素间的空间关系作任何假设。

几何边界 geometric boundary (4.44)

由约束几何对象范围的几何维数更低的几何单形的集合所表达的边界。

B.6.2.2 几何单形与相关术语

几何单形 geometric primitive (4.48)

表达空间内单一、连通和同质元素的几何对象。

注：几何单形是在能表达几何结构信息的层次上不可再分的对象。几何单形包括点、曲线、曲面和体。

点 point (4.61)

0 维几何单形，表示一个位置。

注：点的边界是空集。

曲线 curve (4.23)

1 维几何单形，表达一条线的连续映像。

注：一条曲线的边界是曲线两个端点的集合。如果曲线是一个圈(cycle)，则这两个端点相同，并且该曲线(如果拓扑上是闭合的)被认为没有边界。这第一个点被称为始点，最后一个点被称为终点。曲线的连通性被“一条线的连续映像”子句所保证，拓扑定理认为一个连通集合的连续映像仍然是连通的。

始点 start point (4.72)

起点

曲线的第一个点。

终点 end point (4.36)

曲线的最后一个点。

曲线段 curve segment (4.24)

具有相似的插值与定义方法,用来表达一条曲线的连续组分的1维几何对象。

注:用一条单独的曲线段表达的几何集与一条曲线等价。

环 ring (4.63)

是圈的简单曲线。

注:环用来描述在2维和3维坐标系下的曲面的边界成分。

曲面 surface (4.75)

2维几何单形,局部表达某平面的一个区域的连续映像。

注:曲面的边界是定义曲面界限的有向、闭合曲线的集合。与一个球面或 n 圆环面(具有 n 个把柄的拓扑球面)同构的曲面没有边界,这样的曲面被称之为圈。

曲面片 surface patch (4.76)

2维且黑体的几何对象,用于表达使用相同插值与定义方法的一张曲面的连续部分。

壳 shell (4.66)

为圈的简单曲面。

注:壳用来描述3维坐标系中体的边界成分。

体 solid (4.68)

3维几何单形,表达欧几里德3维空间中一个区域的连续映像。

注:体可局部地实现为直接位置的三参数集合。体的边界是构成体的界限的有向闭合曲面的集合。

B.6.2.3 几何复形

几何复形 geometric complex (4.45)

分离的几何单形的集合,其每一个几何单形的边界都可以表达为该集合内维数更低的几何单形的并集。

注:集合中的几何单形是分离的,意味着没有一个直接位置能同时在两个或两个以上的几何单形的内部。集合在边界操作中闭合,意味着几何复形中的每一个元素,存在一个代表该元素边界的几何单形的组合(即一个几何复形)。点(几何中唯一的0维单形对象)的边界是空。这样,如果最高维的几何单形是体(3维),按照该定义,边界算子的运算最多三步结束。这就是任何对象的边界是一个圈(cycle)的情形。

子复形 subcomplex (4.74)

所有元素都在一个更大复形里的复形。

注:因为几何复形与拓扑复形的定义只要求它们在边界操作中闭合,所以一个特定维数以及更低维数的任何单形的集合总是原来更大复形的子复形。这样,任何完全的平面拓扑复形包含一个作为子复形的边-结点图。

组合曲线 composite curve (4.10)

曲线的序列,其各条曲线(第一条除外)均从该序列的上一条曲线的终点开始。

注:组合曲线作为直接位置的集合,具有曲线所具有的所有特征。

组合体 composite solid (4.11)

沿着共有边界曲面相邻且相连的体的集合。

注:组合体作为直接位置的集合,具有体所具有的所有特征。

组合曲面 composite surface (4.12)

沿着共有边界曲线相邻且相连的曲面片的集合。

注:组合曲面作为直接位置的集合,具有曲面所具有的所有特征。

B.7 拓扑术语

B.7.1 拓扑概念

计算拓扑 computational topology (4.14)

在计算几何中通常用来辅助、增强或定义对拓扑对象进行操作的拓扑概念、拓扑结构和拓扑代数。

B.7.2 拓扑对象

B.7.2.1 一般概念

拓扑边界 topological boundary (4.77)

由约束拓扑对象范围的更低拓扑维的有向拓扑单形的集合所表达的边界。

注：拓扑复形的边界对应于该拓扑复形的几何实现的边界。

拓扑复形 topological complex (4.78)

在边界操作中闭合的拓扑单形的集合。

注：在边界操作中闭合，意味着如果一个拓扑单形在该拓扑复形中，则它的边界对象也在该拓扑复形中。

拓扑对象 topological object (4.81)

一种空间对象，用来表达在连续变换下不变的空间特征。

注：拓扑对象可以是拓扑单形、拓扑单形的一个集合或拓扑复形。

余边界 coboundary (4.9)

与该拓扑对象相关联(该拓扑对象在其边界上)，具有更高维数的拓扑单形的集合。

注：如果一个结点在边的边界上，这个边就在这个结点的余边界上。任何定向参数，只要与这些关系中的一个相关联也就与其他的关系相关联。所以，如果一个结点是一条边的终结点(定义为正向有向边的终点)，那么这个结点的正定向(定义为正向有向结点)将有边在它的余边界上，见图 35。

拓扑维 topological dimension (4.79)

在几何对象内区分邻近但不同的直接位置所需自由变量的最小个数。

注：上面提到的自由变量通常可以认为是一个局部坐标系。在 3 维坐标空间中，平面可以写为 $P(u, v) = A + uX + vY$ ，这里 u 和 v 是实数， A 是平面上的任意点， X 和 Y 是该平面的两个切向量。因为平面上的地点可以用 u, v 这两个变量来区分(没有例外)，所以平面是 2 维的，并且 (u, v) 是该平面上对这些点的一个坐标系。在一般曲面上，并不总能这样。但是，如果我们取该曲面的切平面，并且将曲线上的点投影到该切平面上，我们通常可以在该切平面上得到曲面在该切点的小邻域的局部同构。对于这个潜在曲面，该“局部坐标系”足以建立该曲面为一个 2 维拓扑对象。

由于本标准仅涉及空间坐标，任何 3 维对象可以依靠坐标建立它的拓扑维。在一个四维模型(空间-时间)中，切空间(tangent spaces)在将对象在 3 维上建立拓扑维的过程中也起着重要的作用。

B.7.2.2 拓扑单形及相关术语

拓扑单形 topological primitive (4.82)

单一的、不可再分的拓扑对象。

注：拓扑单形对应于几何实现中同维几何单形的内部。

结点 node (4.58)

0 维拓扑单形。

注：结点的边界是空集。

相连结点 connected node (4.16)

起始于或终止于一条或多条边的结点。

孤立结点 isolated node (4.55)

没有与任何边相连的结点。

始结点 start node (4.71)

在边的边界中对应于边的始点的结点。在使用该边的拓扑复形的任何有效几何实现中，边为曲线。

终结点 end node (4.35)

在边的边界中对应于边的终点的结点。在使用该边的拓扑复形的任何有效几何实现中,边为曲线。

边 edge (4.33)

1 维拓扑单形。

注:边的几何实现是曲线。边的边界是拓扑复形中关联到该边的一个或两个结点的集合。

[拓扑]面 face (4.38)

2 维拓扑单形。

注:拓补面的几何实现是曲面。拓补面的边界是在该拓补复形内通过边界关系关联到该面的有向边的集合。这些有向边的集合可以组织成若干环。

拓补体 topological solid (4.83)

3 维拓补单形。

注:拓补体的边界由有向拓补面的集合组成。

B.7.2.3 拓补复形及相关术语

拓补复形 topological complex (4.78)

在边界操作下闭合的拓补单形的集合。

注:在边界操作下闭合,意味着如果一个拓补单形在拓补复形中,则它的边界对象也是在该拓补复形中。

全域面 universal face (4.84)

在 2 维复形中无界的拓补面。

注:全域面通常不是任何要素的组成部分,它用来表达数据集的无界部分,通常认为它的内边界(它没有外边界)是由数据集所表达的地图的外边界。本标准并不将全域面当特殊情况对待,但应用模式可发现将其特殊处理也很方便。

全域体 universal solid (4.85)

在 3 维复形中无界的拓补体。

注:3 维中全域体与 2 维中的全域面相似,通常也不是任何要素的组成部分。

拓补表达式 topological expression (4.80)

有向拓补单形的集合,这些有向拓补单形可以像多变量多项式一样进行操作。

注:拓补表达式用在很多计算拓补的计算中。

有向拓补对象 directed topological object (4.31)

表达拓补单形和它的某一定向逻辑关联的拓补对象。

有向结点 directed node (4.29)

表达结点和它的某一定向相关联的有向拓补对象。

注:有向结点用在余边界中以保持边与结点间的空间关联,结点的定向是关于某条边的,终结点为“+”,始结点为“-”,这与“结果=结束一起始”的矢量概念相一致。

有向边 directed edge (4.27)

表达边和它的某一定向相关联的有向拓补对象。

注:与边的定向一致的有向边为正定向(+),反之,它为反定向(-)。有向边用在拓补中以区分同一条边的右侧(-)与左侧(+),同一条边的始结点(-)与终结点(+)。在计算拓补中使用这些概念。

有向面 directed face (4.28)

表达面和它的定向之一相关联的有向拓补对象。

注:从有向拓补面定向的矢量方向来看,组成该面的外边界的有向边的定向为正;约束一个拓补体的有向拓补面的定向指向离开该拓补体。与两个相邻拓补面与其共享边之间的关联相一致,共享一个边界拓补面的两个相邻拓补体将具有不同的定向。有向拓补面用在余边界关系中,以维持拓补面与边之间的空间关联。

0

有向拓扑体 directed solid (4.30)

表达拓扑体和它的定向之一相关联的有向拓扑对象。

注：有向拓扑体用在余边界关系中以维持拓扑面与拓扑体之间的空间关联。一个拓扑体的定向是关于某一拓扑面的，如果拓扑面的外法向指向外，则拓扑体的定向为“+”；如果拓扑面的外法向指向内，则拓扑体定向为“-”。这与约束一个体的曲面“向上=向外”的概念相一致。

B.7.2.4 拓扑复形的类型**图 graph (4.51)**

结点集,有些结点与边相连。

注：在地理信息系统中,一个图可以有多于一条边连在两个结点上,也可以一条边的首尾结点相同。

边-结点图 edge-node graph (4.34)

镶嵌在拓扑复形中的图,该拓扑复形由其内的所有边和相连结点构成。

注：边-结点图是镶嵌在复形中的子复形。

平面拓扑复形 planar topological complex (4.60)

具有一个几何实现并且可以嵌入欧几里德 2 维空间中的拓扑复形。

B.8 几何与拓扑复形的关系**同态 homomorphism (4.52)**

两个域(例如两个复形)之间存在一种函数关系,能从一个域保持结构地映射到另一个域。

注：同态区别于同构在于它没有反函数的要求。在同构中,两个同态互为反函数。连续函数是拓扑同态的,因为它们保留了“拓扑特征”。拓扑复形到其几何实现的映射保持了边界的概念,因此是同态。

同构 isomorphism (4.56)

在两个域(例如两个复形)之间的一种关系,它们具有从一个域到另一个域——对应地保持结构的函数和相对应的反函数,并且按两种顺序组合这两个函数得到相应的恒等函数。

注：如果几何复形与拓扑复形的元素个数、维数、边界都——对应,则该几何复形与该拓扑复形同构。

几何实现 geometric realization (4.49)

几何复形,其几何单形与一个拓扑复形中的拓扑单形——对应,且这两个复形的边界关系一致。

注：在该实现中,拓扑单形表达相应的几何单形的内部。组合几何对象是闭合的。

附录 C

(资料性附录)

空间概念模式示例

C.1 几何

C.1.1 语义

对于可实例化的类,本示例使用本文档中的正文部分的类型的名称。该处没有使用通常的 UML 语义作为类型,主要有下面几点考虑。首先,一致性条款并不要求在应用模式中包含本标准的类型(type),而是要求应用模式中的类实现这些类型。在逻辑上并不要求实例化类的名称与本标准的类型或接口名称不同。第二,假设一个设计系统使用了强名称空间规定,不同名称空间的项可以具有相同的局部名称。换句话说,局部名称不必全局统一。第三,这些例子对于任何实现相同类型的可执行类都是有效的。任何实现(应用模式)必须有一个将这些类型与实现它们的执行类关联起来的模式映射。正确地使用这个映射就可以导出有效的句法。

通常,可以将普通名称用于“隐喻性相同”而技术上不同的实体。本标准中的 UML 模型定义抽象类型,应用模式定义概念化的类,各种软件系统定义执行的类或数据结构,来自于编码标准的 XML 定义实体标记(tag)。所有这些定义引用相同的信息内容。即便是在更深的层次上,在数字化实体的具体实施上具有重大的技术差别,但在允许使用相同的名称来表达相同的信息内容这方面并没有什么困难。这就“允许”定义在 UML 模型中的类型直接使用在应用模式中。

C.1.2 2 维坐标参照系下的几何对象

这个例子基于一个简单的解码过程。由于隐去了创建一个可视的拓扑编辑器的细节讨论,因而它被用作编码的逆操作。对于(遵从应用模式规则定义的)应用模式作如下的假定:

- a) 本示例的几何与拓扑模式都遵从定义在本标准中的空间模式,因而本示例中包含了定义在本标准正文部分的主要的几何与拓扑类型的可实例化的子类。为了可读性,在使用这些可实例化子类的场合,这些类型的名称使用本标准正文部分的名称。
- b) 本示例包括使用完整的平面拓扑的需求。
- c) 本示例包括 2 维坐标参照系。
- d) 要素模式包括专题、要素和在附录中最小拓扑里描述与讨论的要素成分。
- e) 在创建后插入到被称为“数据仓库”的数据存储里保存起来作为持久对象。

图 C.1 表达了一个基于平面流形(manifold)的 GM_Complex 的几何。为了构造该复形,下面的例子使用了一个用基于由图表所给定的坐标系统的构造函数来创建对象的功能串联步骤。一旦该对象被创建,它就可以在任何后来的公式中被使用。由于在标准正文部分没有给出正式的构造函数,假定一个缺省的构造函数采用一个代表对象状态的记录作为参数。这是与将要用到的 SQL99 标准相一致的。SQL 基于插入语义自动为任何 UDT(用户定义类型)创建缺省的构造函数。调用“<”表示一个记录或一个作过排序的集合(列表),“{ }”表示一个未排序的集合或包。

构造可以从创建点开始。由于 GM_Point 是一个类型不能被实例化的类型,因而这里有一小问题。为了与应用模式相一致,必须包含一个是 GM_Point 的子类型的可实例化的类,并且在创建的串联步骤中每一个用到 GM_Point 的地方,用这个类去代替。首先,创建 7 个 GM_Point,这 7 个 GM_Point 代表 7 个点,并记为 {P1, ..., P7}。

P1=GM_Point < position=< 1.00,5.00 > >

P2=GM_Point < position=< 3.00,5.00 > >

P3=GM_Point < position=< 3.00,2.00 > >

ϕ
 P4=GM_Point < position= $\langle 1.75, 2.75 \rangle$ >
 P5=GM_Point < position= $\langle 1.50, 4.50 \rangle$ >
 P6=GM_Point < position= $\langle 2.00, 3.25 \rangle$ >
 P7=GM_Point < position= $\langle 5.00, 4.00 \rangle$ >
 插入 P1,P2,P3,P4,P5,P6,P7 到数据库中。

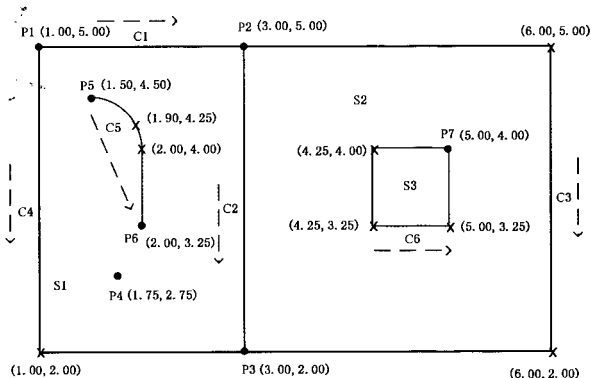


图 C.1 由 GM_Primitive 组成的数据集

用这些点,该串联步骤可以继续创建 7 个 GM_CurveSegment,记为{CS1,CS2,CS3,CS4,CS5,CS6,CS7},它们将接着用来创建曲线。GM_CurveSegment 的子类型是不能持久保持标识的数据类型。这样,下面用来定义曲线段的变量是“堆”或在构造函数范围内的局部变量,它们不是可持久保存的,除非它们作为一个对象类型的成员被包含在一个对象里(在这种情况下,接下来定义曲线)。所有的曲线段这里都定义为线串或弧。

```

CS1=GM_CurveSegment <controlPoint= $\langle P1,P2 \rangle$ ,interpolation="linear" >
CS2=GM_CurveSegment <controlPoint= $\langle P2,P3 \rangle$ ,interpolation="linear" >
CS3=GM_CurveSegment <controlPoint= $\langle P2,(6,5),(6,2),P3 \rangle$ ,interpolation="linear" >
CS4=GM_CurveSegment <controlPoint= $\langle P1,(1,2),P3 \rangle$ ,interpolation="linear" >
CS5=GM_CurveSegment <controlPoint= $\langle P5,(1.9,4.25),(2,4) \rangle$ ,interpolation="arc" >
CS6=GM_CurveSegment <controlPoint= $\langle (2,4),P6 \rangle$ ,interpolation="linear" >
CS7=GM_CurveSegment <controlPoint= $\langle P7,(4.25,4),(4.25,3.25),(5,3.25),P7 \rangle$ ,
interpolation="linear" >
  
```

这里隐藏了一个假设条件,就是变量的持久化,例如 P1,它已经事先先进数据库里,并且可以存取,因而可以支持同步地进行局部拷贝和持久拷贝。允许在使用 GM_Position 数据类型的数据成员时,进行曲线段(作为曲线下面的数据成员)的插入。在一个只使用 SQL 语言的应用程序接口(API)的对象关系数据库里,该应用程序将跟踪变量的引用,在其后的插入语句中使用它们。以这类的方式,在相同的数据池里使用一个对象的接口,数据库 API 将这个跟踪问题转交给程序员解决。

曲线段现在可以被用来构造持久对象:6 个 GM_Curve,记为{C1,...,C6}。对于可实例化类型应用的同样问题,在局部应用模式里要求用 GM_Curve 的子类型来代替 GM_Curve。

```

C1=GM_Curve segments={CS1}
C2=GM_Curve segments={CS2}
  
```

C3=GM_Curve segments=<CS3>

C4=GM_Curve segments=<CS4>

C5=GM_Curve segments=<CS5,CS6>

C6=GM_Curve segments=<CS7>

插入 C1,C2,C3,C4,C5,C6 到数据库中。

这些曲线可以被用来构造曲面。对于本例,可以使用平面多边形构造函数,因为我们的坐标空间是 2 维的。曲面的外法向是该曲面的标准外法向(通常记为 k),并且不需要专门指定。因为我们的目的是定义全拓扑复形,我们需要用这些坐标面上的区域表面表达一个完整的覆盖(coverage)。因为全域面经常被用作“Face 0”,在这里我们定义 S_0 为该面的几何实现。这样,这 4 个 GM_Surface 被记为 $\{S_0, S_1, S_2, S_3\}$ 。

S_0 =GM_Surface patch=<GM_Polygon interior=<< C1,C3,-C4 >>>

// 该全域面仅用于构造具有完整的平面图(full planar graph)的拓扑复形

S_1 =GM_Surface patch=<GM_Polygon exterior=< C4,-C2,-C1 >,
interior=<< C5,-C5 >>>

S_2 =GM_Surface patch=<GM_Polygon exterior=<-C3,C2 >,
interior=<<-C6 >>>

S_3 =GM_Surface patch=<GM_Polygon exterior=< C6 >>

插入 S_0, S_1, S_2, S_3 到数据库中

构造一个 GM_Complex 所必须的几何片段都具备了,该 GM_Complex 是 GM_Object 的组合类型,这只需要给出一个所需对象的完整详尽的列表。在接下来的步骤里就直接进入创建一个 TP_Complex。

GComplex=GM_Complex < surfaces={ S_0, S_1, S_2, S_3 },
curves={C1,C2,C3,C4,C5,C6}
points={P1,P2,P3,P4,P5,P6,P7}>

TComplex=TP_Complex < realization=GComplex >

插入 GComplex,TComplex 到数据库中

这就完成了在本段开始的示意图(见图 C.1)中所描述的几何与拓扑的几何创建。虽然超出本文档的范围,要素的构造(见图 C.2)可以演义如下:

Lake=AreaFeature featureType="Hydrography::WaterBody",extent= S_3

RoadCenterline=LineFeature featureType="Transportation::Road",
centerline=C2

RoadArea=RoadCenterLine.centerline.buffer < distance=10m >

RoadExtent=AreaFeature featureType="LandCover::Road"
extent=RoadArea

RoadInstance=ComplexFeature featureType="LandUse::Road",
featureComponents={RoadCenterline,RoadArea}

Trail=LineFeature featureType="CulturalFacilities::HikingTrail",
centerline=C5

School=PointFeature featureType="CulturalFacilities::School",
Location=P4

插入 Lake,RoadCenterline,RoadExtent,RoadInstance,Trail,School 到数据库中

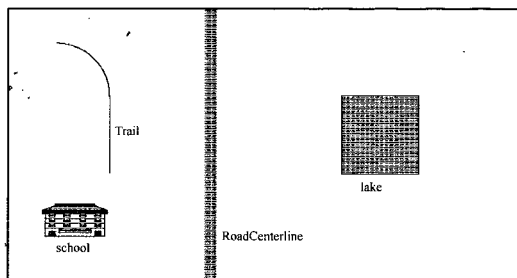


图 C.2 对示例数据的简单图形化表达

C.1.3 3 维坐标参照系下的几何对象

在图 C.3 有一个具有平面小面的 3 维体。这是一个长方块，在里面切掉了一个长方槽，形成一个反相凹陷。

$P1=GM_Point\ position=(2.00,5.00,4.00)$
 $P2=GM_Point\ position=(5.00,5.00,4.00)$
 $P3=GM_Point\ position=(5.00,3.00,4.00)$
 $P4=GM_Point\ position=(2.00,3.00,4.00)$
 $P5=GM_Point\ position=(2.00,5.00,2.00)$
 $P6=GM_Point\ position=(5.00,5.00,2.00)$
 $P7=GM_Point\ position=(5.00,3.00,2.00)$
 $P8=GM_Point\ position=(2.00,3.00,2.00)$
 $P9=GM_Point\ position=(1.00,5.00,1.00)$
 $P10=GM_Point\ position=(9.00,5.00,1.00)$
 $P11=GM_Point\ position=(9.00,1.00,1.00)$
 $P12=GM_Point\ position=(1.00,1.00,1.00)$
 $P13=GM_Point\ position=(1.00,5.00,7.00)$
 $P14=GM_Point\ position=(9.00,5.00,7.00)$
 $P15=GM_Point\ position=(9.00,1.00,7.00)$
 $P16=GM_Point\ position=(1.00,1.00,7.00)$

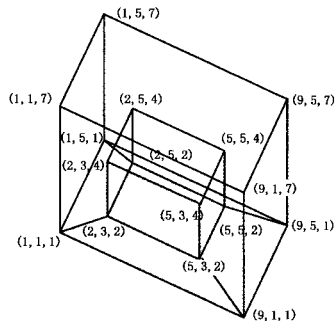


图 C.3 具有坐标系统的 3 维几何体

该曲面可以表达为一个 GM_GriddedSurface(绕着它自己包裹构成一个拓扑柱面),两个 GM_Polygon(作为拓扑柱面两端的盖子),它们都由平面插值生成。

```

S1 = GM_Surface patch =
  ( < GM_BilinearGrid rows=4, columns=5,
    controlPoint = ( < P1, P2, P3, P4, P1,
                    < P5, P6, P7, P8, P5)
                    < P9, P10, P11, P12, P9,
                    < P13, P14, P15, P16, P13 > ),
    GM_Polygon exteriorVertices = < P1, P2, P3, P4, P1 >,
    GM_Polygon exteriorVertices = < P16, P15, P14, P13, P16 > )

```

示例中图 C.4 由一个 GM_Point [P1]、一个 GM_Curve [C1] 和一个 GM_Surface [S1] 构成。GM_Surface 关联的段对应于 9 个 GM_SurfacePatch。第一个 GM_SurfacePatch 代表虚线左边的区域,另外 8 个 GM_SurfacePatch,都是 GM_Triangle 代表虚线右边的区域。

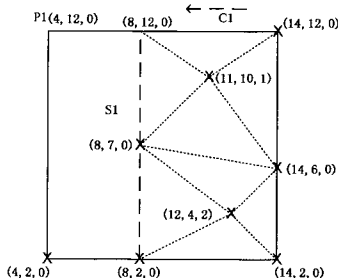


图 C.4 曲面示例

```

P1 = GM_Point(4, 12, 0)
C1 = GM_Curve segment = (Segment 1)
Segment1 = GM_CurveSegment controlPoint = ((4, 12, 0), (4, 2, 0), (14, 2, 0), (14, 12, 0),
(4, 12, 0))
Patch1 = GM_Polygon exterior = (P1, (4, 2, 0), (8, 2, 0), (8, 12, 0), P1)
Post1 = GM_Position(8, 12, 0)
Post2 = GM_Position(14, 12, 0)
Post3 = GM_Position(11, 10, 1)
Post4 = GM_Position(8, 7, 0)
Post5 = GM_Position(14, 6, 0)
Post6 = GM_Position(12, 4, 2)
Post7 = GM_Position(8, 2, 0)
Post8 = GM_Position(14, 2, 0)
T1 = GM_Triangle exterior = (Post1, Post2, Post3, Post1)
T2 = GM_Triangle exterior = (Post1, Post3, Post4, Post1)
T3 = GM_Triangle exterior = (Post3, Post5, Post4, Post3)
T4 = GM_Triangle exterior = (Post2, Post5, Post3, Post2)
T5 = GM_Triangle exterior = (Post4, Post5, Post6, Post4)

```

0.

T6=GM_Triangle exterior=(Post4,Post6,Post7,Post4)

T7=GM_Triangle exterior=(Post5,Post8,Post6,Post5)

T8=GM_Triangle exterior=(Post7,Post6,Post8,Post7)

S1=GM_Surface patch=(Patch1,T1,T2,T3,T4,T5,T6,T7,T8)

注意,同样的示例可以被描述为两个 GM_Surface 的集合,其一为一个单独的 GM_Surface-Patch——Patch1,其二是由 8 个 GM_Triangle 组成的一个 GM_TriangulatedSurface。这两个 GM_Surface 然后又可以组合为一个等价于一个单独的 GM_Surface 的 GM_CompositeSurface。

附录 D

(资料性附录)

应用模式示例

D.1 概述

根据 ISO 19109 建立的应用模式,可以使用定义在本标准中的包,使用方式为用这个包的类或接口来定义子类,并扩展其定义在这里的成员协议(属性、操作或者两者)。

该机制定义可实例化的类,这些类通过构造性多态支持从本标准的包中来的所需接口。

D.2 简单拓扑

D.2.1 简单拓扑包

除了使用对于双重的拓扑与几何对象都使用的多重继承的可选项之外,具体的拓扑类的创建类似于几何类的创建。这并不产生与多重继承模式相关联的问题,这种多重继承模型与实现的多重继承相关联,但是必须考虑到当一个对象在几何边界(GM_Object::boundary)和拓扑边界(TP_Object::boundary)有区别时的情况。

D.2.2 简单拓扑类

D.2.2.1 语义

定义在这个包(图 D.1、图 D.2)的类型都双重继承了边界操作,一方面来自各种类型的 TP_Primitive,另一方面来自各种类型的 GM_Primitive。虽然我们使用允许为两个不同的遗传路线定义各自语义的多重继承,但在这里却不是这样。即使两个边界操作开始于不同的继承树,但在简单拓扑(TS)中却是相同的。

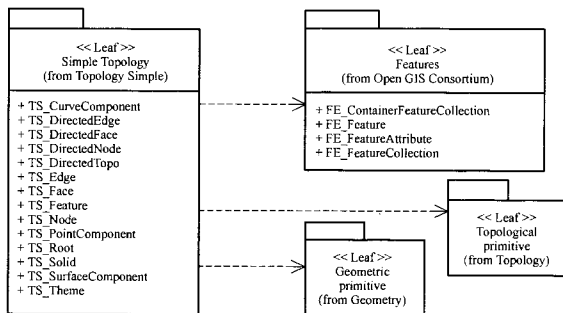


图 D.1 用于简单拓扑的包与类

D.2.2.2 TS_根 (TS_Root)

TS_Root 作为根类,允许该模式对用在该包中的所有的几何与拓扑类作出限制。TS_Root 对象的边界仅包含其他的 TS_Root 对象。

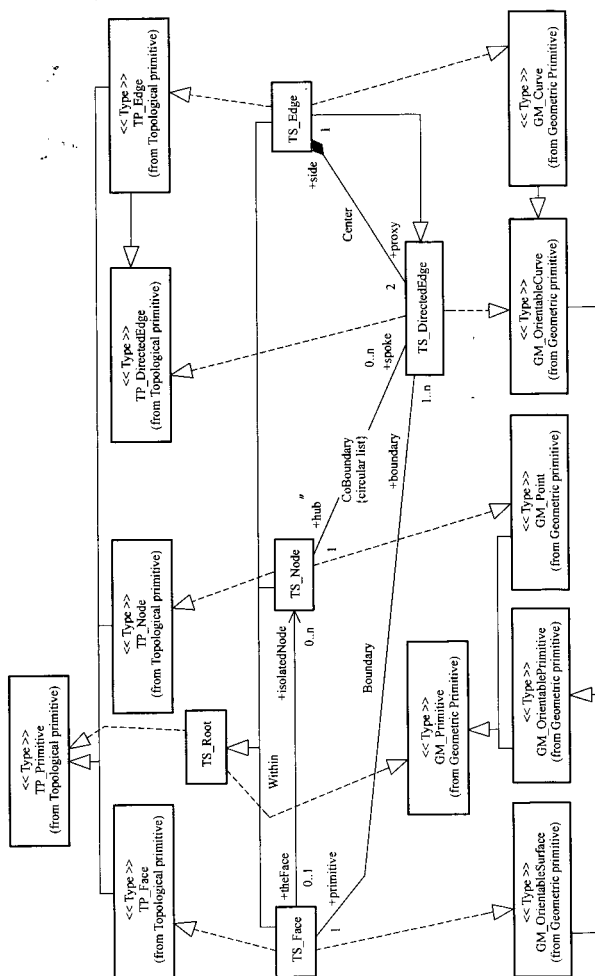


图 D.2 简单拓扑中的拓扑与几何类

TS_Root;

TP_Primitive::boundary→isTypeOf(TS_Root);

TP_Primitive::boundary=GM_Primitive::boundary;

注：TS_Root 是从 TP_Primitive 和 GM_Primitive 派生的子类，这意味着边界操作是双重定义的。这第二个限制说即使它们是相同的，也允许已经形成的约束基于边界操作而不加区分。由于对于 GM_Primitive 和 TP_Primitive 的边界操作是同构的，因而它们在这种情况下是相同的，该约束在从其他单形继承来的边界操作上也很容易实现。

D.2.2.3 TS_结点 (TS_Node)

TS_Node(TS_结点)从 TP_Node 和 GM_Point 多重继承而来。允许支持拓扑和几何两者的数据与操作。

D.2.2.4 TS_边 (TS_Edge)

TS_Edge 从 TP_Edge 和 GM_Curve 多重继承而来。允许支持拓扑和几何两者的数据与操作。

D.2.2.5 TS_有向边 (TS_DirectedEdge)

TS_DirectedEdge 从 TP_DirectedEdge 和 GM_OrientableCurve 多重继承而来。允许支持拓扑和几何两者的数据与操作。

D.2.2.6 TS_拓扑面 (TS_Face)

TS_Face 从 TP_Face 和 GM_OrientableSurface 多重继承而来。允许支持拓扑和几何两者的数据与操作。

D.2.2.7 TS_专题 (TS_Theme)

TS_Theme(见图 D.3)的作用类似于 GM_Complex,用以将相似的几何对象(在该示例中为各种要素和相关类型的要素组分,例如交通线或行政边界)聚集在一起。TS_Theme 从 GM_Complex 继承而来。它也可以是从 Feature 类派生的子类,以允许它持有 Feature 的属性。

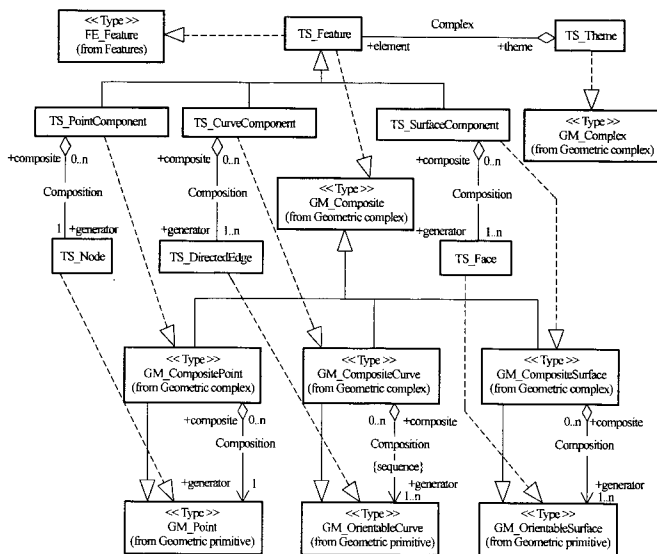


图 D.3 简单拓扑中的要素成分

D.2.2.8 TS_要素 (TS_Feature)

TS_Feature 作为该包中所有要素类的根类。它由描述在 ISO 19109 中的通用要素模型中的 Feature 类派生而来,允许该包中所有的要素对象拥有任何适当类型的属性。

D.2.2.9 TS_点组分 (TS_PointComponent)

TS_PointComponent 从 TS_Feature 和 GM_CompositePoint 多重继承而来,允许它的行为像一个独立的几何对象和要素一样。

D.2.2.10 TS_曲线组分 (TS_CurveComponent)

TS_CurveComponent 从 TS_Feature 和 GM_CompositeCurve 多重继承而来,允许它的行为像一个独立的几何对象和要素一样。

D.2.2.11 TS_曲面组分 (TS_SurfaceComponent)

TS_SurfaceComponent 从 TS_Feature 和 GM_CompositeSurface 多重继承而来,允许它的行为像一个独立的几何对象和要素一样。

D.3 要素拓扑 (FT)

D.3.1 语义

在这个包后面的基本概念是允许组合形几何对象,在这里定义为要素组分,被组织成一个拓扑结构独立于(但仍然由它们组成)它们的空间属性的拓扑结构。这样,在一个专题内(专题是 TP_Complex 和 GM_Complex 下的子类),要素组分可以与另一个要素组分相关联,而这另一个要素组分基于拓扑结构上相同于那些被用来作为基本的泛专题(所有专题)的几何对象。这里做了一个假定,要素组分在与它们的专题内的其他要素对象的结合点处分开。

D.3.2 专题级要素拓扑类

D.3.2.1 FT_复形 (FT_Complex)

FT_Complex(见图 D.4) 从 TP_Complex 和 GM_Complex (通过 TS_Theme)多重继承而来,允许它聚合拓扑与几何两者的信息。这种方法允许在数据集内的每一个专题携带专题特有的拓扑信息。这简单地假定每一个要素组分是在一个并且只在一个专题内,可以随着一个维持拓扑和几何对象二分的略为更复杂的结构而提升。

D.3.2.2 FT_单形 (FT_Primitive)

FT_Primitive 支持作为 TP_Primitive 相同的功能,并作为基本构件来构建 FT_Complex 内实例化的 TP_Complex。

D.3.2.3 FT_结点 (FT_Node)

一个 FT_Node 既是一个 TP_Node 又是一个 TS_PointComponent。这样,如果必要,在一个专题内的这些点要素组分在一个要素拓扑复形内始结点的作用。

D.3.2.4 FT_边 (FT_Edge)

一个 FT_Edge 既是一个 TP_Edge 又是一个 TS_CurveComponent。这样,如果必要,在一个专题内的这些曲线要素组分在一个要素拓扑复形内起边的作用。

D.3.2.5 FT_拓扑面 (FT_Face)

一个 FT_Face 既是一个 TP_Face 又是一个 TS_AreaComponent。这样,如果必要,在一个专题内的这些区域要素组分在一个要素拓扑复形内起拓扑面的作用。

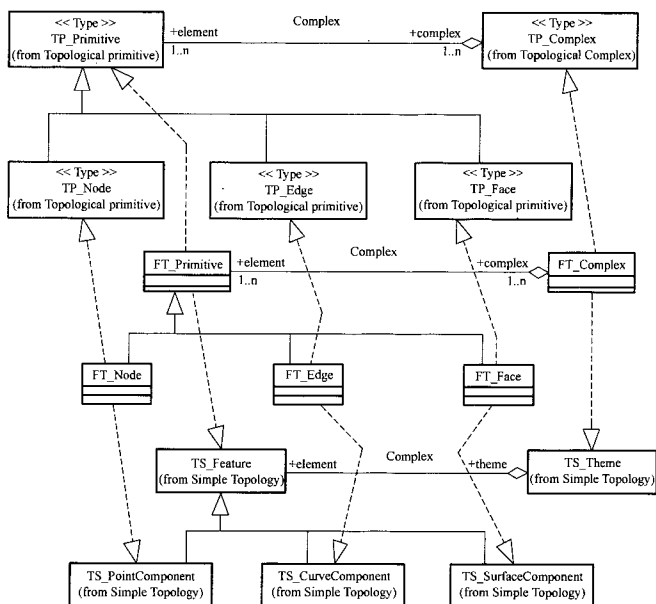


图 D.4 基于要素拓扑的专题

D.4 最小拓朴

这个最小拓朴(MiniTopo)专用标准并不定义任何新的类,而只是设置已有类的使用约束。它把与“Within”相关联的“IsolatedIn”特化,仅仅用来表达孤立于拓朴面的结点。

DIGEST 所隐含的拓朴模型,从一个更早的称为 MC&G 或最小拓朴的模型中派生出来,关于拓朴邻接的大多数信息是由一些“对(pair)”所携带的,每个“对”由若干有向边(DE)关联到一个拓朴面上,见图 D.5。在该模式(见图 D.6)的对应信息是由边界和余边界算子/关系携带的。表 D.1 将最小拓朴与在现在的模型的适当信息对应起来。

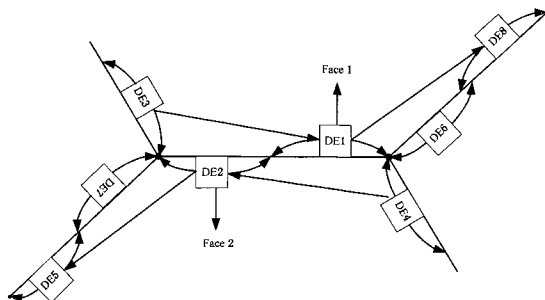


图 D.5 “最小拓朴”拓朴结构的集合示例

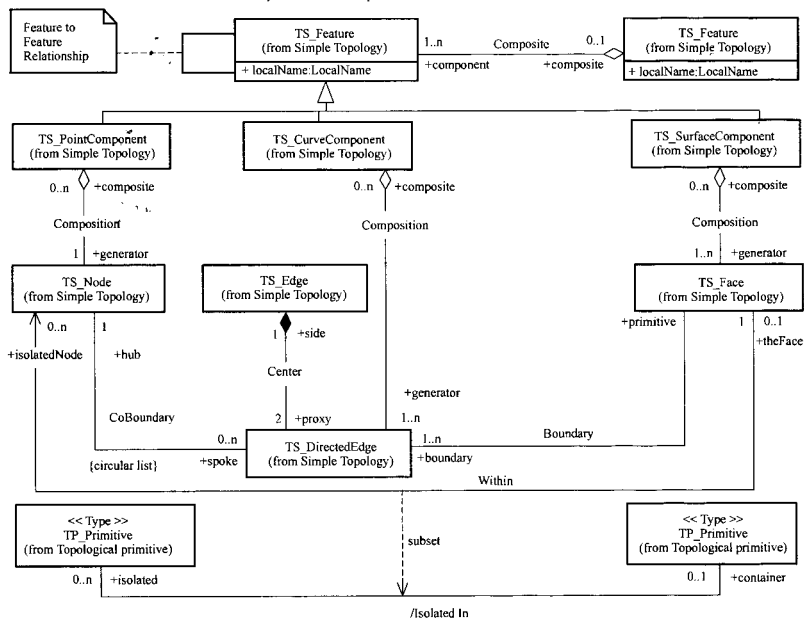


图 D.6 最小拓扑

这个最小拓扑记录结构有 9 个基本类型的记录, 4 个针对要素, 4 个针对几何与拓扑, 还有一个补充的概念针对这些通常通过记录数目实现的类型。这个最小拓扑“拓扑—几何”记录类型是结点、边、有向边和拓扑面。

对于有它们自己的结构定义的几何与拓扑的记录类型由下面给出(在句法上像 SQL99)。

Create Node record as {

```
nodeID; RecordIdentifier NOT NULL PrimaryKey,
containingFace; RecordIdentifier ForeignKey to Face,
--NULL for node connected to edges,
position; CoordinatePoint NOT NULL }
```

Create Edge record as {

```
edgeID; RecordIdentifier NOT NULL Primary Key,
positiveDE; RecordIdentifier NOT NULL Foreign Key to DirectedEdge,
negativeDE; RecordIdentifier NOT NULL Foreign Key to DirectedEdge,
coordinateList; Variable Array Of CoordinatePoint NOT NULL }
```

Create DirectedEdge record as {

```
directedEdgeID; RecordIdentifier NOT NULL Primary Key,
nodeID; RecordIdentifier NOT NULL Foreign Key to Node,
```

```

nextDE:RecordIdentifier NOT NULL Foreign Key to DirectedEdge,
face:RecordIdentifier NOT NULL Foreign Key to Face }

Create Face record as {
    faceID:RecordIdentifier NOT NULL Primary Key }

```

该结构的主要优点之一就是每一个记录大小是固定的,并且具有很高的标准化水平。该结构被认为包含最小数目的冗余,因而得名最小冗余拓扑,简称最小拓扑。

假定边的坐标是作为对图形记录的引用而保持,则每一个最小拓扑具有固定的大小。这种结构的有些变量基于是否为反向键而赋予各种关系。原始的简单交换文件结构不携带这种反向键,因为它们必须携带各种长度信息。还有一个变量也与边和有向边的记录有关,这就是给一个记录以三个语义上的主键,通常写为 edgeID、+edgeID 和 -edgeID。

除了几何/拓扑记录之外,有 4 种要素记录。

```

Create PointFeature as {           // 本质上是一个多点
    featureID : RecordIdentifier NOT NULL Primary Key,
    nodeID : Variable Array Of RecordIdentifier NOT NULL Foreign Key to Node,
    attribute : Variable Array of
        {Name :CharacterString, Value :CharacterString} }

Create LineFeature as {           // 本质上是一个组合曲线
    featureID:RecordIdentifier NOT NULL Primary Key,
    directedEdgeID : Variable Array Of RecordIdentifier NOT NULL
        Foreign Key to DirectedEdge,
    attribute : Variable Array of
        {Name :CharacterString, Value :CharacterString} }

Create AreaFeature as {           // 本质上是一个组合曲面
    featureID :RecordIdentifier NOT NULL Primary Key,
    faceID : Variable Array Of RecordIdentifier NOT NULL Foreign Key to Face,
    attribute : Variable Array of
        {Name :CharacterString, Value :CharacterString} }

Create ComplexFeature as {        // 本质上是一个聚集要素
    featureID :RecordIdentifier NOT NULL Primary Key,
    pointComponentID : Variable Array Of RecordIdentifier
        Foreign Key to PointFeature,
    lineComponentID : Variable Array Of RecordIdentifier
        Foreign Key to LineFeature,
    areaComponentID : Variable Array Of RecordIdentifier
        Foreign Key to AreaFeature,
    subfeatureID : Variable Array Of RecordIdentifier
        Foreign Key to ComplexFeature,
    attribute : Variable Array of
        {Name :CharacterString, Value :CharacterString} }

```

这些记录的变量包括(1)不再包括其他任何要素的复形要素类型的专题记录概念,(2)携带每一个要素与拓扑记录信息的专题掩码属性,即专题所涉及的数据成员的属性信息。

```

Create Theme as { // 本质上是一个复形要素变量
    featureID :RecordIdentifier NOT NULL Primary Key,
    pointComponentID :Variable Array Of RecordIdentifier
        Foreign Key to PointFeature,
    lineComponentID :Variable Array Of RecordIdentifier
        Foreign Key to LineFeature,
    areaComponentID :Variable Array Of RecordIdentifier
        Foreign Key to AreaFeature,
    subfeatureID :Variable Array Of RecordIdentifier
        Foreign Key to ComplexFeature,
    // 因为 theme 不是一个复形变量,它不能被任何其他复形要
    // 素或 theme 所拥有(所有的指针都是强类型)

Attribute: Variable Array of
    <Name :CharacterString, Value :CharacterString>
ThemeMask:Integer //作为一个字节掩码加到每一个复形要素和要素组分上,
    //专题掩码的大小通常限制专题的数目最多到 32 个
}

```

图 D.7 所示的是平面布置图的标准记录。

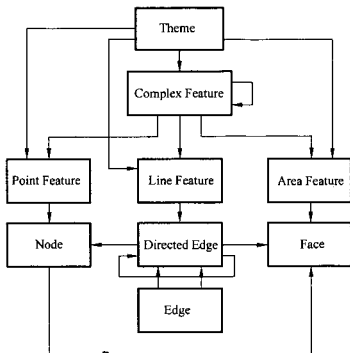


图 D.7 典型的最小拓扑记录示意图

现行的模式与最小拓扑模式间的主要差别来自它们的起源。最小拓扑最先是作为交换结构设计的，因为连续平坦的文件结构的限制，使其不具有丰富的对象模型。在最小拓扑结构后面的概念级的模型不能与现行模型在边与面之间以及边与结点之间的边界结构上精确吻合。为了节省空间和加速从交换结构到计算结构间的转换，引入了最小拓扑。最小拓扑使用一个分散连接的列表结构（如图 D.5 所示），使用有向边来代表在现行模型中表达清晰的边界和余边界的内部结构。随着从由现行模型的UML中继承而来的越来越丰富的对象建模能力的推广，应用将由原始概念上的最小拓扑模型越来越多地向现行的清晰对象结构模型靠近。

富对象(rich object)建模环境的一个最大优点就是缩小“语义差别”。“语义差别”是用来描述概念模型与实现模型之间的差别的一个不太正规的名词。大多数的差别是由于需要将概念创建重新映射到程序语言创建而引起的。由于富对象模型给出了十分丰富的语言建造词汇，可以缩小这种“差距”。最小拓扑模型来源于在算法复杂性和数据结构复杂性以及数据量之间的一个折中。1984年设计最小拓

扑模型时,经成本折中,倾向于更小、更紧凑的数据结构,并以此为基础构建更鲁棒的概念模型,而其成本计算主要基于内存的对象结构。由于围绕最小拓扑结构的发展的制约,其原作者从来没有对其进行完整论述。只有做了一定折中处理后的交换结构,见诸于公开文献,并且也没有对折中处理进行解释。现在,这种折中已经得到明显改善,并且在计算和持久保存模型两者的概念结构的保存受到了重视。付出的代价包括更详尽、更广泛的持久存储模型以及计算模型的复杂度的增加。带来的好处是模型间的一致性更好,“语义差别”更小,计算环境与逻辑更一致、语义更丰富的概念模型更贴近。事实上,该模型与原来的最小拓扑概念模型并不抵触,而是将其文档化了,并且通过更现代的对象环境予以实现。

表 D.1 原始最小拓扑指针与现行模型之间的对应关系

最小拓扑指针	现行模型	描 述
边对应正有向边	TS_Edge 关联到它自己,该关联来自 TP_Edge 的 Center::side; TP_DirectedEdge	每一条边是它自己的正有向边
边对应负有向边	TS_Edge 以负定向关联到 TS_DirectedEdge, 该关联来自 TP_Edge, Center::side; TP_DirectedEdge	每一条边关联到一条负有向边
正有向边对应终结点	TS_Edge 以正定向关联到 TS_DirectedNode, 该关联来自 TP_Edge, 从边界算子/boundary::boundary; TP_DirectedNode 衍生而来	边的边界为两个有向结点,其定向一个是正,一个是负。正定向的有向结点对应于终结点,负定向的有向结点对应于始结点
负有向边对应始结点	TS_Edge 以负定向关联到 TS_DirectedNode, 该关联来自 TP_Edge, 从边界算子/boundary::boundary; TP_DirectedNode 衍生而来	
正有向边对应左拓补面	TS_Edge 以正定向关联到 TS_DirectedFace, 该关联来自 TP_Edge, 从余边界算子/co-Boundary::spoke; TP_DirectedFace 衍生而来	边的余边界为两个有向拓补面,其定向一个是正,一个是负。正定向的有向面对应于左边的拓补面,负定向的有向面对应于右边的拓补面
负有向边对应右拓补面	TS_Edge 以负定向关联到 TS_DirectedFace, 该关联来自 TP_Edge, 从余边界算子/co-Boundary::spoke; TP_DirectedFace 衍生而来	
有向边对应下一个有向边(绕着拓补面)	从 TS_Face 到 TS_DirectedEdge 的关联角色“boundary”的结构继承自 TP_Face 从/boundary::boundary; Set(TP_Ring)派生来的 boundary 算子	拓补面的边界是环的一个集合。每一个环是有向边的一个循环序列。在该循环中相邻的边是从原来的最小拓扑模型来的有向边→(下一条有向边)对

附 录 NA

(资料性附录)

不同维数的几何对象与拓扑对象构成对照

表 NA.1 不同维数的几何对象与拓扑对象构成对照表

对象维数	几何对象		拓扑对象	
	中文	英文	中文	英文
0	点	Point	结点	Node
1	曲线	Curve	边	Edge
2	曲面	Surface	[拓扑]面	Face
3	体	Solid	[拓扑]体	[Topological] Solid

参 考 文 献

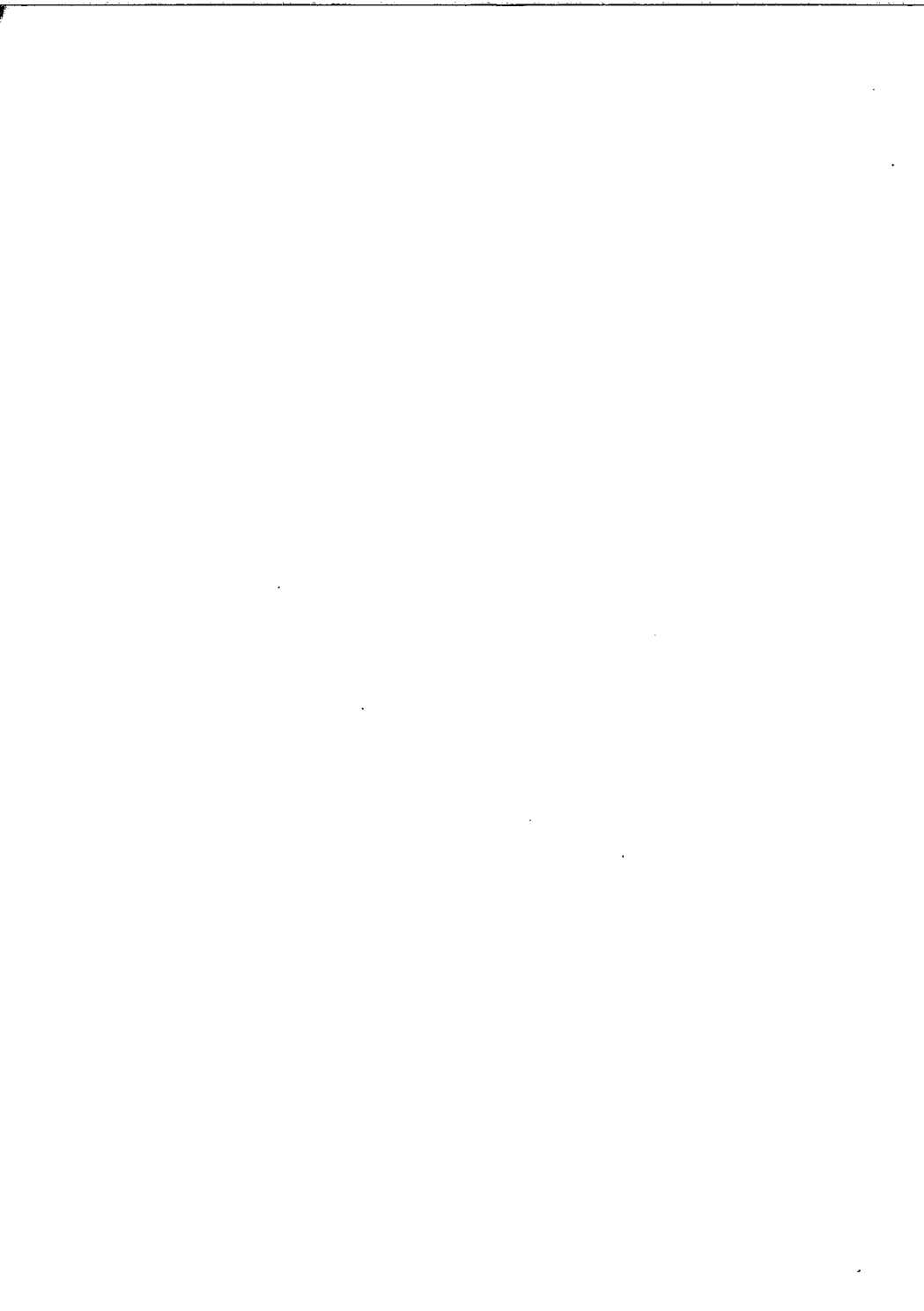
- [1] ISO 19101:2002, Geographic information—Reference model
- [2] ISO 19111:2007, Geographic information—Spatial Referencing by Coordinates
- [3] ISO 19118:2005, Geographic information—Encoding
- [4] ISO/TS 19103:2005, Geographic information—Conceptual schema language
- [5] ABADI, M. and CARDELLI, L. A Theory of Objects, Springer-Verlag, New York, 1996
- [6] BARTELS, R. H., BEATTY J. C. and BARSKY, B. A. An Introduction to Splines for Use in Computer Graphics & Geometric Modeling, Morgan Kaufmann Publishers, Inc., 1987
- [7] CLAPHAM, C., Concise Dictionary of Mathematics, Second Edition, Oxford University Press, 1996
- [8] CLEMENTINI, E. and DI FELICE, P. A Comparison of Methods for Representing Topological Relationships. Information Sciences 80, 1-34, 1994
- [9] CLEMENTINI, E. and DI FELICE, P. A Model for Representing Topological Relationships Between Complex Geometric Features in Spatial Databases. Information Sciences 90 (1-4): 121-136, 1996
- [10] DAINTITH, J. and NELSON, R. D. Dictionary of Mathematics, Penguin Books, London, 1989
- [11] Dictionary of Computing, Fourth Edition, Oxford University Press, 1996
- [12] EGENHOFER, M. F. and FRANZOSA, Point Set Topological Spatial Relations. International Journal of Geographical Information Systems, vol 5, no 2, 161-174, 1991
- [13] EGENHOFER, M. J., CLEMENTINI, E. and DI FELICE, P. Topological relations between regions with holes. International Journal of Geographical Information Systems, vol 8, no 2, pp 129-142, 1994
- [14] FARIN, G. Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide, Second Edition, Academic Press, Inc., Boston, 1990
- [15] FARIN, G. NURB Curves and Surfaces, From Projective Geometry to Practical Use, A. K. Peters, Wellesley, Massachusetts, 1995
- [16] FAUX, I. D. and PRATT, M. J. Computational Geometry for Design and Manufacture, Ellis Norwood Publishers, reprinted with corrections, 1981
- [17] ISO/IEC 13249-3: 1999, Information technology—Database languages—SQL Multimedia and Application Packages—Part 3: Spatial, ISO/IEC JTC1 SC 32, 1999
- [18] KOSTOV, V. and DEGTIARIOVA-KOSTOVA, E. Some properties of clothoids. INRIA, Nice—Sophia Antipolis, Research Report NO 2752, December 1995
- [19] OMG/UML, Object Constraint Language Specification, version 1. 3, 1997. Available at <<http://www.rational.com/uml/resources/documentation/formats.jsp>>
- [20] OMG/UML, UML Notation Guide, version 1. 3, 1997. Available at <<http://www.rational.com/uml/resources/documentation/formats.jsp>>
- [21] OMG/UML, UML Semantics, version 1. 3, 1997. Available at <<http://www.rational.com/uml/resources/documentation/formats.jsp>>
- [22] OpenGIS Consortium, The OpenGIS Abstract Specification, Volumes 1-14, 1997
- [23] PRENTER, P. M. Splines and Variational Methods, John Wiley & Sons, New York, 1975,

GB/T 23707—2009/ISO 19107:2003

republished in Wiley Classics Edition, 1989

[24] ROGERS, D. F. and ADAMS, J. A. Mathematical Elements for Computer Graphics, Second Edition, McGraw Hill Publishing Company, 1990

注：本标准的一些定义和附加信息来自以上文献，为使本标准表述得清晰与简洁，对其中一些内容进行了编辑。



中 华 人 民 共 和 国
国 家 标 准

地理信息 空间模式

GB/T 23707—2009/ISO 19107:2003

*

中国标准出版社出版发行
北京复兴门外三里河北街16号
邮政编码:100045

网址 www.spc.net.cn

电话:68523946 68517548

中国标准出版社秦皇岛印刷厂印刷
各地新华书店经销

*

开本 880×1230 1/16 印张 8.75 字数 257 千字
2009年8月第一版 2009年8月第一次印刷

*

书号: 155066·1-38345 定价 102.00 元

如有印装差错 由本社发行中心调换

版权专有 侵权必究

举报电话:(010)68533533



GB/T 23707-2009