McMaster University • Department of Computing and Software • CAS 741-
Development of Scientific Computing Software • Winter 2009

Module Interface Specifications
for
MAG: A Delaunay Mesh Generator

By: Mustafa Elsheikh
Submitted to: Dr. Spencer Smith
As part of course project

Version: MAG.MIS.1.0 • Date: March 27, 2009

Contents

1 Introduction **4**

1.1 Overview 4

1.2 The template 4

1.3 The notation 4

2 Module Specifications **6**

2.1 Point 6

2.2 Vertex 7

2.3 Triangle 8

2.4 Geometry 9

2.5 BBox 11

2.6 Mesh 12

2.7 Topology 13

2.8 Format 13

2.9 Domain 15

2.10 PointLocator 16

2.11 Cavity 16

2.12 Delaunay 17

2.13 File 18

2.14 Container 19

2.15 Main 19

2.16 CommandLine 19

2.17 Statistics 19

2.18 Validation 19

1 Introduction

This section gives an overview of the document, and the notation used.

1.1 Overview

This document provides the module interface specifications.

1.2 The template

The template for the specification follows [HS99] with minor modifications.

1.3 The notation

The notation used is the predicate logic with types, and the language of Zermelo-Frankel set theory.

A typed variable declaration takes the following form

$$x_1 : T_1, x_2 : T_2, ...$$

or the shorthand

$$x_1, x_2, x_3, ... : T$$

for the same types.

Quantification takes the form

$$Q(\text{typed variable declarations} \mid \text{boolean range} \bullet \mathit{expr})$$

where Q is either $\forall$, $\exists$, or ε. The later denotes the Hilbert choice operator. The type of the quantified expression is the same type of the *expr*. The meaning of $\bullet$ follow the common denotation in first order logic, i.e., $\Rightarrow$ for universal quantification and $\wedge$ for existential quantification, choice operator, and the set membership $\in$ (described below). An omitted range means *true*.

Set comprehension has the following form

$$\{\text{typed variable declarations} \mid \text{boolean range} \bullet \mathit{expr}\}$$

to denote the set of values resulting from *expr*. As in quantification, an omitted range means *true*.

Individual types are denoted by adding a prefix T to the end of the type name. General types (e.g., natural number, real numbers) are denoted with their conventional symbols. Subset notation is supported for these types. For example

$$\mathbb{N}[0..2]$$

would mean the *inclusive* subset of natural numbers between 0 and 2. Type constructors for set and tuple types are denoted by $set[\cdot]$, and $tuple[\cdot]$, respectively. However, the expression constructors are $\{\cdot\}$, and $\langle\cdot\rangle$, respectively.

For state transitions, a primed variable denotes a next state value.

The symbol $\triangleq$ means a definition. While $\equiv$ is the equivalence of the boolean formulas that allow substitution. It is restricted to the definition of the exception aliases. The symbol $=$ denotes equality. No assignment operator is used. Instead the primed notation is used.

The symbol $\circ$ is the sequential composition of two access program calls. It is used in the state transition specification to denote a specific order of calls. For example

$$A() \circ B()$$

means call $A()$ first then $B()$.

2 Module Specifications

2.1 Point

1. Module Name: Point
   2. Uses:
      2.1. Imported Modules:
         None
   3. Interface Syntax:
      3.1. Exported Constants:
         NoPoint$\triangleq \varepsilon(\{p : PointT \mid p \notin PointT \bullet p\})$
      3.2. Exported Data Types:
         PointT $\triangleq$ tuple$[x : \mathbb{R}, y : \mathbb{R}]$
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
init	$\mathbb{R}, \mathbb{R}$		
getX		$\mathbb{R}$	
getY		$\mathbb{R}$	
setX	$\mathbb{R}$		
setY	$\mathbb{R}$		

   4. Interface Semantics:
      4.1. State Variables:
         $x : \mathbb{R}, y : \mathbb{R}$
      4.2. Invariant:
         None
      4.3. Assumptions:
         None
      4.4. Access Program Semantics:
         init$(ix : \mathbb{R}, iy : \mathbb{R}) \triangleq$
            Transition: $(x' = ix \wedge y' = iy)$
         getX() $\triangleq$
            Output: x
         getY() $\triangleq$
            Output: y
         setX(ix: $\mathbb{R}$) $\triangleq$
            Transition: $x' = ix$
         setY(iy: $\mathbb{R}$) $\triangleq$
            Transition: $y' = iy$

2.2 Vertex

1. Module Name: Vertex
   2. Uses:
      2.1. Imported Modules:
      Point, Triangle
   3. Interface Syntax:
      3.1. Exported Constants:
      NoVertex $\triangleq \varepsilon(\{v : VertexT \mid v \notin VertexT \bullet v\})$
      3.2. Exported Data Types:
      VertexT $\triangleq$ tuple[p : PointT, t: TriangleT]
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
create	PointT, TriangleT	VertexT	InvalidPoint
setPoint	PointT		InvalidPoint
getPoint		PointT	
setTriangle	TriangleT		InvalidTriangle
getTriangle		TriangleT	

   4. Interface Semantics:
      4.1. State Variables:
      p : PointT, t :: TriangleT
      4.2. Invariant:
      ($p \neq$ NoPoint $\wedge\, t \neq$ NoTriangle)
      4.3. Assumptions:
      None
      4.4. Access Program Semantics:
      create(ip : PointT, it : TriangleT)$\triangleq$
         Exception: InvalidPoint $\equiv$ (ip = NoPoint)
         Transition: ($p' = ip \wedge t' = it$)
         Output: $\langle p, t\rangle$
      setPoint(ip : PointT)$\triangleq$
         Exception: (InvalidPoint $\equiv$ (ip = NoPoint))
         Transition: ($p' = ip$)
      getPoint()$\triangleq$
         Output: p
      setTriangle(it : TriangleT)$\triangleq$
         Exception: (InvalidTriangle $\equiv$ (it = NoTriangle))
         Transition: ($t' = it$)
      getTriangle()$\triangleq$
         Output: t

2.3 Triangle

1. Module Name: Triangle
   2. Uses:
      2.1. Imported Modules:
         Vertex
   3. Interface Syntax:
      3.1. Exported Constants:
         NoTriangle $\triangleq \varepsilon(\{t : TriangleT \mid t \notin TriangleT \bullet t\})$
      3.2. Exported Data Types:
         TriangleT $\triangleq$ tuple[v_0 : VertexT, v_1 : VertexT, v_2 : VertexT, n_0 : TriangleT, n_1 : TriangleT, n_2 : TriangleT]
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
create	VertexT, VertexT, VertexT	TriangleT	InvalidVertex
getVertex	$\mathbb{N}[0..2]$	VertexT	
getVertices		set[VertexT]	
setNeighbor	$\mathbb{N}[0..2]$, TriangleT		InvalidTriangle
getNeighbor	$\mathbb{N}[0..2]$	TriangleT	
isNeighbor	TriangleT	$\mathbb{B}$	InvalidTriangle
whichNeighbor	TriangleT	$\mathbb{N}[0..2]$	InvalidTriangle, NotNeighbor
isIntersecting	TriangleT	$\mathbb{B}$	InvalidTriangle
whichSide	TriangleT	$\mathbb{N}[0..2]$	InvalidTriangle, NotIntersectin
isEqual	VertexT, VertexT, VertexT	$\mathbb{B}$	InvalidVertex
hasVertex	VertexT	$\mathbb{B}$	InvalidVertex

   4. Interface Semantics:
      4.1. State Variables:
         v_0 : VertexT, v_1 : VertexT, v_2 : VertexT, n_0 : TriangleT, n_1 : TriangleT, n_2 : TriangleT
      4.2. Invariant:
         $(v_0 \neq v_1 \neq v_2 \neq$NoVertex$)$
      4.3. Assumptions:
         None
      4.4. Access Program Semantics:
         create(iv_0 : VertexT, iv_1 : VertexT, iv_2 : VertexT)$\triangleq$
            Exception: (InvalidVertex $\equiv \exists(i : \mathbb{N}[0..2] \mid \bullet iv_i =$ NoVertex$)$)
            Transition: $\forall(i : \mathbb{N}[0..2] \mid \bullet v_i' = iv_i \wedge n_i' =$ NoTriangle$)$
         getVertex($i : \mathbb{N}[0..2]$)$\triangleq$
            Output: v_i
         getVertices()$\triangleq$

Output: $\{i : \mathbb{N}[0..2] \mid \bullet v_i\}$
setNeighbor($num : \mathbb{N}[0..2], ntri :$ TriangleT) $\triangleq$
Exception: (InvalidTriangle $\equiv$ ($ntri$ = NoTriangle))
Transition: $(n'_{num} = ntri)$
getNeighbor($i : \mathbb{N}[0..2]$)$\triangleq$
Output: n_i
isNeighbor($it : TriangleT$)$\triangleq$
Exception: (InvalidTriangle $\equiv$ (it = NoTriangle))
Output: $\exists(i : \mathbb{N}[0..2] \mid \bullet n_i = it)$
whichNeighbor($it : TriangleT$)$\triangleq$
Exception: (InvalidTriangle $\equiv$ (it = NoTriangle) $\wedge$ NotNeighbor $\equiv$ $\neg\exists(i : \mathbb{N}[0..2] \mid \bullet n_i = it)$)
Output: $\exists(i : \mathbb{N}[0..2] \mid n_i = it \bullet i)$
isIntersecting($it : TriangleT$)$\triangleq$
Exception: (InvalidTriangle $\equiv$ (it = NoTriangle))
Output: $\mid getVertices() \cap it.getVertices() \mid = 2$
whichSide($it : TriangleT$)$\triangleq$
Exception: InvalidTriangle $\equiv$ (it = NoTriangle)$\wedge$NotIntersecting $\equiv\mid getVertices() \cap it.getVertices() \mid \neq 2$ —
Output: $\exists(i, j : \mathbb{N}[0..2] \mid v_i, v_j \in getVertices() \cap it.getVertices() \bullet if(i = 0 \wedge j = 1, 0, if(i = 1 \wedge j = 2, 1, 2)))$
isEqual($iv_0, iv_1, iv_2 : VertexT$)$\triangleq$
Comment: Shallow equality regardless of the order of the vertices.
Exception: InvalidVertex $\equiv \exists(i : \mathbb{N}[0..2] \mid \bullet iv_i =$ NoVertex)
Output: $\{i : \mathbb{N}[0..2] \mid \bullet v_i\} = \{i : \mathbb{N}[0..2] \mid \bullet iv_i\}$
hasVertex($iv : VertexT$)$\triangleq$
Exception: InvalidVertex $\equiv iv =$ NoVertex
Output: $iv \in \{i : \mathbb{N}[0..2] \mid \bullet v_i\}$

2.4 Geometry

1. Module Name: Geometry
  2. Uses:
    2.1. Imported Modules:
    Point
  3. Interface Syntax:
    3.1. Exported Constants:
    EPSILON : $\mathbb{R}$
    3.2. Exported Data Types:
    SignT $\triangleq$ {POS, NEG, ZERO}
    3.3. Exported Access Programs:

Name	Input	Output	Exceptions
sign	$\mathbb{R}$	SignT	
positive	$\mathbb{R}$	$\mathbb{B}$	
zero	$\mathbb{R}$	$\mathbb{B}$	
negative	$\mathbb{R}$	$\mathbb{B}$	
distance	PointT, PointT	$\mathbb{R}$	
orient	PointT, PointT, PointT	SignT	
inside	PointT, PointT, PointT, PointT	SignT	
inCircle	PointT, PointT, PointT, PointT	SignT	

4. Interface Semantics:

4.1. State Variables:

None

4.2. Invariant:

None

4.3. Assumptions:

None

4.4. Access Program Semantics:

positive(r : $\mathbb{R}$) $\triangleq$

Output: $(r > \text{EPS})$

negative(r : $\mathbb{R}$) $\triangleq$

Output: $(r < \text{EPS})$

zero(r : $\mathbb{R}$) $\triangleq$

Output: $(-\text{EPS} \leq r \leq \text{EPS})$

sign(r : $\mathbb{R}$) $\triangleq$

Output: (positive(r) $\Rightarrow$ POS) $\wedge$ (zero(r)$\Rightarrow$ ZERO) $\wedge$(negative(r)$\Rightarrow$NEG)

distance(p_0 : PointT, p_1 : PointT) $\triangleq$

Output: $(\sqrt{((p_0.x - p_1.x)^2 + (p_0.y - p_1.y)^2)})$

inCircle(p_0: PointT, p_1: PointT, p_2: PointT, p: PointT) $\triangleq$

$$\text{Output: positive(det} \begin{bmatrix} p_0.x & p_0.y & p_0.x^2 + p_0.y^2 & 1 \\ p_1.x & p_1.y & p_1.x^2 + p_1.y^2 & 1 \\ p_2.x & p_2.y & p_2.x^2 + p_2.y^2 & 1 \\ p.x & p.y & p.x^2 + p.y^2 & 1 \end{bmatrix})$$

orient(p_0: PointT, p_1: PointT, p: PointT) $\triangleq$

$$\text{Output: positive(det} \begin{bmatrix} p_0.x & p_0.y & 1 \\ p_1.x & p_1.y & 1 \\ p.x & p.y & 1 \end{bmatrix})$$

inside(p_0: PointT, p_1: PointT, p_2 : PointT, p : PointT) $\triangleq$

Output: $\forall(i : \mathbb{N}[0..2] \mid \bullet(orient(p_i, p_{(i+1) mod\, 3}, p) = orient(p_i, p_{(i+1) mod\, 3}, p_{(i+2) mod\, 3})))$

2.5 BBox

1. Module Name: BBox
   2. Uses:
      2.1. Imported Modules:
      Point, Domain, Vertex
   3. Interface Syntax:
      3.1. Exported Constants:
      None
      3.2. Exported Data Types:
      BBoxT $\triangleq$tuple$[v_0, v_1, v_2, v_3 : VertexT]$
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
create	MeshT, DomainT, $\mathbb{R}$	BBoxT	

   4. Interface Semantics:
      4.1. State Variables:
      None
      4.2. Invariant:
      None
      4.3. Assumptions:
      None
      4.4. Access Program Semantics:
      create(m: MeshT, d: DomainT, r: $\mathbb{R}$)$\triangleq$
      Exception: BadRatio $\equiv r \leq 1$
      Transition:

let

$x_{max} = max(\{p : PointT \mid p \in d \bullet p.x\}), x_{min} = min(\{p : PointT \mid p \in d \bullet p.x\}), y_{max} = max(\{p : PointT \mid p \in d \bullet p.y\}), y_{min} = min(\{p : PointT \mid p \in d \bullet p.y\})$

in let

$x_{mid} = (x_{min}+x_{max})/2, y_{mid} = (y_{min}+y_{max})/2, x_{diff} = x_{max}-x_{min}, y_{diff} = y_{max} - y_{min}$

in

$v'_0 = m.createVertex(d.addPoint(x_{mid} - r * x_{diff}, y_{mid} - r * y_{diff}))$,$v'_1 = m.createVertex(d.addPoint((x_{mid}+r*x_{diff}, y_{mid}-r*y_{diff}), v'_2 = m.createVertex(d.addPoint((x_m$ $r * x_{diff}, y_{mid} + r * y_{diff}),\ v'_3 = m.createVertex(d.addPoint((x_{mid} - r * x_{diff}, y_{mid} + r * y_{diff})$

Output: $\langle v_0, v_1, v_2, v_3\rangle$

2.6 Mesh

1. Module Name: Mesh
   2. Uses:
      2.1. Imported Modules:
      Point, Vertex, Traingle
   3. Interface Syntax:
      3.1. Exported Constants:
      None
      3.2. Exported Data Types:
      MeshT $\triangleq$ tuple[T : set[TriangleT], V : set[VertexT]]
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
create		MeshT	
createVertex	PointT	VertexT	
createTriangle	VertexT, VertexT, vertexT	TriangleT	
deleteVertex	VertexT		InvalidVertex
deleteTriangle	TriangleT		InvalidTriangle

   4. Interface Semantics:
      4.1. State Variables:
      None
      4.2. Invariant:
      None
      4.3. Assumptions:
      None
      4.4. Access Program Semantics:
      create()$\triangleq$
         Transition: $V' = \phi \wedge T' = \phi$
         Output: $\langle V, T\rangle$
      createVertex(ip: PointT)$\triangleq$
         Transition: $V' = V \cup \{Vertex.create(ip, NoTriangle)\}$
         Output: $\exists(v : VertexT \mid v = Vertex.create(ip, NoTriangle) \bullet v)$
      createTriangle(iv_0 : VertexT, iv_1 : VertexT, iv_2 : VertexT)$\triangleq$
         Transition: $T' = T \cup \{Triangle.create(iv_0, iv_1, iv_2)\}$
         Output: $\exists(t : TriangleT \mid t = Triangle.create(iv_0, iv_1, iv_2) \bullet t)$
      deleteVertex($v : VertexT$)$\triangleq$
         Exception: InvalidVertex $\equiv v =$NoVertex
         Transition: $V' = V\backslash\{v\}$
      deleteVertex($t : TriangleT$)$\triangleq$
         Exception: InvalidTriangle $\equiv t =$ NoTriangle
         Transition: $T' = T\backslash\{t\}$

2.7 Topology

1. Module Name: Topology
   2. Uses:
      2.1. Imported Modules:
         Vertex, Triangle, Mesh
   3. Interface Syntax:
      3.1. Exported Constants:
         None
      3.2. Exported Data Types:
         None
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
validNeighbors	MeshT		
validIncidence	MeshT		

   4. Interface Semantics:
      4.1. State Variables:
         None
      4.2. Invariant:
         None
      4.3. Assumptions:
         None
      4.4. Access Program Semantics:
         validNeighbors(m : MeshT) $\triangleq$

Comment: For all triangles t_1, t_2 in the mesh, if they intersect in two vertices, then the neighbor of t_1 on the side of the intersection is t_2, same for t_2. Also, there is no other triangle t_3 that intersect with either t_1 or t_2, or is a neighbor to either.

Output:

$\forall(t_1, t_2 : TriangleT \mid t_1, t_2 \in m.T \bullet t_1.isIntersecting(t_2) \Rightarrow (t_1.getNeighbor(t_1.getSide(t_2)) = t_2 \wedge t_2.getNeighbor(t_2.getSide(t_1)) = t_1 \wedge \neg\exists(t_3 : TriangleT \mid t_3 \in m.T - \{t_1, t_2\} \bullet t_3.isIntersecting(t_1) \vee t_3.isNeighbor(t_1) \vee t_3.isIntersecting(t_2) \vee t_3.isNeighbor(t_2)))$

validIncidence(m : MeshT) $\triangleq$

Comment: For every vertex v in mesh, ensure that the supporting triangle $v.t$ has v as one of its vertices.

Output: $\forall(v : Vertex \mid v \in m.V \bullet v.t.hasVertex(v))$

2.8 Format

1. Module Name: Format

1.1. Comment: This module has to refined into two modules, one for input and another for output.

2. Uses:

2.1. Imported Modules:

Point, File

3. Interface Syntax:

3.1. Exported Constants:

None

3.2. Exported Data Types:

FormatT $\triangleq \{\text{POLY}, \text{PLY}\}$

3.3. Exported Access Programs:

Name	Input	Output	Exceptions
create	FileT	FormatT	InvalidFile, InvalidForma
setFile	FileT		InvalidFile
getFile		FileT	
setFormat	FormatT		InvalidFormat
getFormat		FormatT	
readHeader (*)			ReadingError, MalformedHe
getTotalPointCount (*)		$\mathbb{N}$	ReadingError, MalformedD
getNextPoint (*)		PointT	ReadingError, MalformedD
setTotalPointCount (*)	$\mathbb{N}$		EmptyPoints
setTotalTriCount (*)	$\mathbb{N}$		EmptyTriangles
writeHeader (*)			WritingError
writePoint (*)	PointT		WritingError, InvalidPoir
writeTriangle (*)	PointT, PointT, PointT		WritingError, InvalidTrian

(*) The semantics of these access programs are omitted.

4. Interface Semantics:

4.1. State Variables:

file: FileT

4.2. Invariant:

None

4.3. Assumptions:

None

4.4. Access Program Semantics:

create(ifile: FileT)$\triangleq$

Exception : InvalidFile $\equiv$ (ifile = NoFile)

Transition : (file' = ifile)

setFile(ifile: FileT)$\triangleq$

Exception: InvalidFile $\equiv$ (ifile = NoFile)

Transition: file' = ifile

getFile()$\triangleq$
Output: file
setFormat(iformat: FormatT)$\triangleq$
Exception: InvalidFormat $\equiv$ (iformat = NoFormat)
Transition: format' = iformat
getFormat()$\triangleq$
Output: format

2.9 Domain

1. Module Name: Domain
   2. Uses:
      2.1. Imported Modules:
      Point, Format
   3. Interface Syntax:
      3.1. Exported Constants:
      None
      3.2. Exported Data Types:
      PointListT $\triangleq$sequence[PointT]
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
create		DomainT	
loadPoints (*)	FormatT		
savePoints (*)	FormatT		
getTotalPointCount (*)		$\mathbb{N}$	
getNextPoint (*)		PointT	
addPoint (*)	PointT		
lastPoint (*)			

(*) The semantics of these access programs are omitted.
   4. Interface Semantics:
      4.1. State Variables:
      P: PointListT
      4.2. Invariant:
      None
      4.3. Assumptions:
      None
      4.4. Access Program Semantics:

2.10 PointLocator

1. Module Name: PointLocator
   2. Uses:
      2.1. Imported Modules:
      Mesh, Point, Triangle, Geometry, Topology
   3. Interface Syntax:
      3.1. Exported Constants:
      None
      3.2. Exported Data Types:
      None
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
locate	MeshT, PointT	TriangleT	NoBase

   4. Interface Semantics:
      4.1. State Variables:
      None
      4.2. Invariant:
      None
      4.3. Assumptions:
      None
      4.4. Access Program Semantics:
      $\text{locate}(m : \text{MeshT}, p : \text{PointT}) \triangleq$
      Exception: $\text{NoBase} \equiv \forall(t : TriangleT \mid t \in m.T \bullet \neg inside(t, p))$
      Output: $\varepsilon(\{t : TriangleT \mid t \in m.T \wedge inside(t, p) \bullet t\})$

2.11 Cavity

1. Module Name: Cavity
   2. Uses:
      2.1. Imported Modules:
      Mesh, Triangle, Point, Geometry
   3. Interface Syntax:
      3.1. Exported Constants:
      None
      3.2. Exported Data Types:
      None
      3.3. Exported Access Programs:

Name	Input	Output	Exceptions
build	MeshT, TriangleT, PointT	CavityT	InvalidBase, Invalide InvalidPoint
getBoundary		set[tuple[tri: TriangleT, side: $\mathbb{N}$[0..2]]]	

4. Interface Semantics:
   4.1. State Variables:
   m : MeshT, base : TriangleT, p : PointT, C : set[TriangleT]
   4.2. Invariant:
   None
   4.3. Assumptions:
   None.
   4.4. Access Program Semantics:
   build(im : MeshT, it : TriangleT, ip : PointT)$\triangleq$
   Exception: (InvalidPoint $\equiv$ (ip = NoPoint))$\wedge$(InvalidBase $\equiv$ (it = NoTriangle)) $\wedge$ (NoCavity $\equiv$ ($C' = \phi$))
   Transition: $m' = im \wedge \text{base}' = it \wedge p' = ip \wedge C' = \{t : TriangleT \mid t \in m.T \wedge t \in inCircle(t.v_0.p, t.v_1.p, t.v_2.p, p) \bullet t\}$
   getBoundary()$\triangleq$
   Output: $\{e : tuple[tri : Triangle, side : \mathbb{N}[0..2]] \mid e.tri \in C \wedge e.tri.getNeighbor(e.side) \notin C \bullet e\}$

2.12 Delaunay

1. Module Name: Delaunay
   2. Uses:
   2.1. Imported Modules:
   Point, Vertex, PointLocator, Cavity, Mesh
   3. Interface Syntax:
   3.1. Exported Constants:
   None
   3.2. Exported Data Types:
   None
   3.3. Exported Access Programs:

Name	Input	Output	Exceptions
init	MeshT, DomainT		
triangulateBBox	MeshT, BBox		
triangulate	MeshT, DomainT		
insert	MeshT, PointT		
retile	MeshT, CavityT, VertexT		

4. Interface Semantics:
   4.1. State Variables:
      None
      4.2. Invariant:
      None
   4.3. Assumptions:
      None
   4.4. Access Program Semantics:
      triangulateBBox(m: MeshT, b: BBoxT)$\triangleq$
         Exception: None
         Transition: $\exists(t_1, t_2 : TriangleT \mid t_1 = m.createTriangle(b.v_0, b.v_1, b.v_2) \wedge t_2 = m.createTriangle(b.v_3, b.v_1, b.v_2) \bullet \text{validIncidence}(m') \wedge \text{validNeighbors}(m'))$
      init(m: MeshT, d: DomainT)$\triangleq$
         Exception: None
         Transition: $\exists(B : BBoxT \mid B = BBox.create(m, d) \bullet triangulateBBox(m, B))$
      triangulate(m: MeshT, d: DomainT)$\triangleq$
         Exception: None
         Transition: $init(m, d) \circ \forall(p : PointT \mid p \in d \bullet insert(m, p))$
      insert(m: MeshT, p: PointT)$\triangleq$
         Exception: None
         Comment: A vertex v is created by wrapping the point p, and the access program retile() is called with a cavity C built from the base triangle t.
         Transition: $\exists(v : VertexT, t : TriangleT, C : CavityT \mid v \in m'.V \wedge v.p = p \wedge t = locate(m, p) \wedge C = build(m, t, p) \bullet retile(m, C, v))$
      retile(m: MeshT, C: CavityT, v: VertexT)$\triangleq$
         Exception: None
         Comment: For every edge e in the boundary of the cavity, there exists a triangle t in the next state of the mesh m', such that t is formed by v and the two vertices of e. And the triangle of that edge e is removed from m'. And the both the neighborhood and the incidence information of m' are updated to be valid.
         Transition: $\forall(e \mid e \in C.\text{getBoundary}() \bullet \exists(t \mid t \in m'.T \bullet t.isEqual(v, e.tri.getVertex(e.si$
$\forall(t : TriangleT \mid t \in C \bullet t \notin m'.T) \wedge \text{validIncidence}(m') \wedge \text{validNeighbors}(m')$

2.13 File

1. Module Name: File
   This module is bound to the underlying OCaml implementation.

2.14 Container

1. Module Name: Container
   This module is bound to the underlying OCaml implementation.

2.15 Main

1. Module Name: Main
   This module is not specified.

2.16 CommandLine

1. Module Name: CommandLine
   This module is not implemented.

2.17 Statistics

1. Module Name: Statistics
   This module is not implemented.

2.18 Validation

1. Module Name: Validation
   This module is not implemented.

References

[HS99] Daniel Hoffman and Paul Strooper. Software design, automated testing, and maintenance A practical approach, July 24 1999.