CAPS:
Computational Aircraft Prototype Syntheses

Description of the Discretization Data Structure: capsDiscr

Bob Haimes
Aerospace Computational Design Laboratory
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology
haimes@mit.edu

3 June 2016

1 Introduction

The discretization data structure (capsDiscr) is composed of a number of other structures described below. capsDiscr is the basic mapping between the discretized portions of the geometry and the analytic geometry found in the Body. In general, there is a single capsDiscr for each *capsBound* Object and a particular *capsAnalysis* Object reflected in a *capsVertexSet* Object. This gets filled by the AIM via the invocation of the function *aimDiscr*. This structure is used for both data transfers (conservative & simple interpolation) and the computation of parametric sensitivities within CAPS. There are 3 vertex indexing schemes used:

- Global index. This is a tessellation related data referring to the global index in the tessellation.
- Local index. This is again a tessellation related number that is in the *space* of the tessellation returned by EG_getTessEdge or EG_getTessFace.
- Discrete Index. This is the vertex index numbering for the capsDiscr structure itself.

The following data structures are used in the CAPS AIM interface and are defined within the CAPS include file "capsTypes.h":

2 capsEleType

```
/* defines the element discretization type by the number of reference positions
 * (for geometry and optionally data) within the element. For example:
 * simple tri:  nref = 3; ndata = 0; st = {0.0,0.0, 1.0,0.0, 0.0,1.0}
 * simple quad: nref = 4; ndata = 0; st = {0.0,0.0, 1.0,0.0, 1.0,1.0, 0.0,1.0}
 * internal triangles are used for the in/out predicates and represent linear
 *   triangles in [u,v] space.
 * ndata is the number of data referece positions, which can be zero for simple
 *   nodal or isoparametric discretizations.
 * match points are used for conservative transfers. Must be set when data
 *   and geometry positions differ, specifically for discontinuous mappings.
 * For example:
 *                       neighbors                          neighbors
 *      2          tri-side   vertices         4         side    vertices
 *     / \             0        1 2           / \          0        1 2
 *    /   \            1        2 0          5   3         1        2 3
 *   /     \           2        0 1         /  6  \        2        3 4
 *  0-------1                              /       \       3        4 5
 *                                        0----1----2      4        5 0
 *                       neighbors                         5        0 1
 *  3-------2      quad-side  vertices           nref = 7
 *  |       |          0        1 2
 *  |       |          1        2 3            6             neighbors
 *  |       |          2        3 0         3---.---2    quad-side  vertices
 *  0-------1          3        0 1         |       |        0        1 2
 *                                         7.   8   .5       1        2 3
 *                       neighbors          |       |        2        3 0
 *  4-------3        side     vertices      0---.---1        3        0 1
 *  |       |          0        1 2            4
 *  |       2          1        2 3
 *  |       |          2        3 4                  nref = 9
 *  0-------1          3        4 0
 *                     4        0 1
 *      nref = 5
 *
 */
typedef struct {
  int    nref;                /* number of geometry reference points */
  int    ndata;               /* number of data ref points -- 0 data at ref */
  int    nmat;                /* number of match points (0 -- match at
                                 geometry reference points) */
  int    ntri;                /* number of triangles to represent the elem */
  double *gst;                /* [s,t] geom reference coordinates in the
                                 element -- 2*nref in length */
  double *dst;                /* [s,t] data reference coordinates in the
                                 element -- 2*ndata in length */
  double *matst;              /* [s,t] positions for match points - NULL
                                 when using reference points (2*nmat long) */
  int    *tris;               /* the triangles defined by geom reference indices
                                 (bias 1) -- 3*ntri in length */
} capsEleType;
```

This data structure defines the positions for nodes that support the spatial discretization of a particular element type. There should be one for each type of element seen in the discretization of this *VertexSet*.

The element locations are referred to as 'reference' positions and have 2 degrees of freedom (which are traditionally in the range $[0.0, 1.0]$). This should not be confused with $[u, v]$ which are the parametric coordinates for a vertex on a Face. To avoid the confusion, these reference positions are referred to as $[s, t]$.

Each element type has a unique suite of 'reference' positions (*nref*). The structure member *gst* is allocated to hold 2 times *nref* doubles, which contain the actual reference coordinates for this element type. If the data locations are the same as the geometry reference positions (for example in nodal-based discretizations), then *ndata* must be zero and *dst* should be NULL. For "cell-centered" finite-volume (or any other) discretizations where the data storage locations are not the vertices at the bounds of the element then *ndata* specifies the number of these locations in the element. *dst* must be allocated to hold 2 times *ndata* doubles, which are then filled with the data reference coordinates for this element type.

For "conservative" data transfers, an optimization scheme is used balance integrated quantities. The balancing is done by interpolating at "match points". Each element type must specify these positions within the element. If the "match" positions are the geometry reference locations, then *nmat* must be zero and *matst* should be NULL. Otherwise, *nmat* specifies the number of these "match" locations in the element and *matst* must be allocated to hold 2 times *nmat* doubles, which are then filled with the match point reference coordinates for this element type.

The number of triangles that the element is broken up into is specified by *ntri*. The member *tris* should be allocated to hold 3 times *ntri* ints. *tris* is filled with the geometry reference indices (bias 1) to represent the triangles that cover the element. Note that there should be consistency in vertex ordering so that all triangles have their normals pointing properly.

3 capsElement

```
/*
 * defines the element discretization for geometric and optionally data
 * positions.
 */
typedef struct {
  int   bIndex;                /* the Body index (bias 1) */
  int   tIndex;                /* the element type index (bias 1) */
  int   eIndex;                /* element owning index -- dim 1 Edge, 2 Face */
  int   *gIndices;             /* local indices (bias 1) geom ref positions,
                                  tess index -- 2*nref in length */
  int   *dIndices;             /* the vertex indices (bias 1) for data ref
                                  positions -- ndata in length or NULL */
  union {
    int tq[2];                 /* tri or quad (bias 1) for ntri <= 2 */
    int *poly;                 /* the multiple indices (bias 1) for ntri > 2 */
  } eTris;                     /* triangle indices  that make up the element */
} capsElement;
```

This structure defines a single element based on its type, owner and indices (from the `capsDiscr` structure and the associated *EGADS* Tessellation Object). *bIndex* is the index into the bodies returned by the AIM utility function *aim_getBodies* (bias 1). *tIndex* is the index into the `capsEleType` structure (bias 1). *eIndex* is the element owning index based on *dim* of `capsDiscr` (either an Edge or Face).

The number of "geometric" indices is defined by the *nref* member referrered to by *tIndex*, where the number of "data" indices comes from *ndata*. *gIndices* is allocated to twice *nref* ints in length and filled with index pairs. The first is the index into this discretization numbering (bias 1), where the numbers must be between 1 and *nPoints* of the `capsDiscr` structure. The tessellation vertex index is the local index into the associated EGADS Tessellation Object referred to by *eIndex*.

If *ndata* is nonzero, then *dIndices* is allocated to *ndata* ints in length and filled with indices into the data reference information (*verts*) of the `capsDiscr` structure (bias 1).

eTris must be (allocated for $ntri > 2$ and) filled with the triangle index/indices of the tessellation that make up the element. They need to be ordered as defined in *tris* of the `capsEleType` structure.

4 capsDiscr

```
/* defines a discretized collection of Elements
 *
 * specifies the connectivity based on a collection of Element Types and the
 * elements referencing the types.
 */
typedef struct {
  int         dim;              /* dimensionality [1-3] */
  int         instance;         /* analysis instance */
  void        *aInfo;           /* AIM info */

                                /* below handled by the AIMs: */
  int         nPoints;          /* number of entries in the point definition */
  int         *mapping;         /* tessellation indices to the discrete space
                                   2*nPoints in len (body, global tess index) */
  int         nVerts;           /* number of data ref positions or unconnected */
  double      *verts;           /* data ref (3*nVerts) -- NULL if same as geom */
  int         *celem;           /* element containing vert (nVerts in len) or NULL */
  int         nTypes;           /* number of Element Types */
  capsEleType *types;           /* the Element Types (nTypes in length) */
  int         nElems;           /* number of Elements */
  capsElement *elems;           /* the Elements (nElems in length) */
  int         nDtris;           /* number of triangles to plot data */
  int         *dtris;           /* NULL for NULL verts -- indices into verts */
  void        *ptrm;            /* pointer for optional AIM use */
} capsDiscr;
```

A `capsDiscr` is the fundamental data structure that defines a connected *VertexSet* in `CAPS`. It gets filled by the AIM plugin during the call to the function *aimDiscr*. The AIM utility function *aim_getBodies* should be used to get all appropriate Bodies for the AIM (based on "capsFidelity"). Each Face (if *dim* is 2) or Edge (if *dim* is 1) should be examined for the *EGADS* attribute "capsBound" and match it to the incoming transfer name. All matching Faces/Edges should be used to fill in this data structure.

All physical positions (except for those in *verts*) are found in the associated *EGADS* Tessellation Object, which should be created in the AIM and set in `CAPS` by invoking *aim_setTess*.

The first 3 members (*dim*, *instance* and *ainfo*) are filled by `CAPS` before the invocation of *aimDiscr*.

The number of geometric reference points (*nPoints*) is the total number of vertices that support this discretization. The association between these points and the EGADS tessellation Object is done by the *mapping* member.

The number of vertices used in the data positions is defined by the member *nVerts* which can be zero. If *nVerts* is nonzero then *nVerts* entries must be allocated for the member *verts* and this must be filled with the XYZ positions associated with the appropriate data reference positions defined as part of the elements. The member *celem* refers to the index of the element containing the position and must be allocated consistent with *verts*.

The number of elements types is set by the member *nTypes* and the types themselves are defined by a pointer to the allocated block of memory *types* which contains *nTypes* of `capsEleType`.

The number of elements found in this *discretization* is defined by the member *nElems*. The member *elems* will be filled with the (geometric) element definitions and optionally data representations (if *ndata* for the element is not zero).

The number of triangles associated with plotting data reference information is set by the member *nDtris*. The actual triangles are defined in *dtris*, which should be 3 times *nDtris* in length. The values stored are the indices into the *verts* member (bias 1).

The member *ptrm* is set aside for the plugin author and can be used to hold on to any data needed to communicate with and between the AIM routines.