

File Format Conventions for the SimBio Project

Frithjof Kruggel & Markus Svensén
Max-Planck-Institute of Cognitive Neuroscience
Stephanstraße 1, 04103 Leipzig, Germany
e-mail: {kruggel,svensen}@cns.mpg.de

Version 1.3 (February 21, 2002):

Added convention on implicit connectivity in graphs, altered storage convention for graph attribute images, added ncomponents as compulsory image attribute, included a definition for sparse attribute images for graphs.

Version 1.2 (July 13, 2000):

XML references removed, Knee coordinate system added, signal data sets defined, graph attribute image definition included.

The SimBio environment provides tools for bio-numerical simulations using finite- element modelling techniques. To ensure an efficient data flow across SimBio tools, a versatile file format is required which allows a mixed storage of volumetric images and meshes in a single file. This document provides a description of a suitable format and suggests conventions required for a successful application in the target domain.

General Terms: SimBio environment, file formats, coordinate conventions, Vista toolkit

1. INTRODUCTION

This document describes file format conventions for SimBio tools [5]. It is assumed to be relevant for SimBio software developers involved in work packages 1 and 3-6. Because SimBio tools will operate on medical data, some additional conventions are required to ensure a reliable use of the tools and to ease communication with users in the target domain.

It is agreed that most SimBio tools use the Vista file format for data storage to provide a maximum compatibility between program modules. Providers of other SimBio tools may convert from their data format to Vista (or vice versa), based on the description in this document. Example Vista data sets conforming to these conventions will be provided on the SimBio web-site.

Vista was developed as a toolkit for computer vision research by the University of British Columbia in 1994 and released to the public domain [1]. The Vista data format is suggested for the following reasons: The C source code is available [6], well documented, easily portable, tested and stable. The file format is machine-independent, very efficient, versatile and easily extendible. Three SimBio partners have significant amount of experience with using these tools.

A Vista data file can contain a variety of different sorts of objects, images and graphs being the most interesting for SimBio. A file's contents are organised as a list of attributes, each comprising both an attribute name and an attribute value.

The attribute name is an alphanumeric string. The attribute value may have one of several forms depending on whether the value is a number, a string, a keyword, a nested list of attributes, an image, or some other object.

2. COORDINATE SYSTEMS

Although the definition of coordinate systems does not strictly belong into a document on file formats, the medical application domain requires to introduce such definitions, which are implicitly used in the format. To achieve clarity with respect to this point, we include coordinate system definitions for the SimBio bodyparts-of-interest in this document.

To ease orientation when visualising data sets of the human head, we suggest to align the head with the stereotactical coordinate system [4], e.g. by a procedure outlined in [2]. This convention will map an axial slice of the body (see definition below) to the x-y plane, with the x axis running from the left to the right body side, the y axis from front to back, the z axis from head to feet.

For the knee, we suggest to follow the conventions introduced by MacWilliams *et al.* [3]: the x axis is parallel to the long axis of the femur, the y axis is parallel to the anterior-posterior direction, and the z axis parallel to the left-right direction. The origin of the coordinate system is in the center of the joint.

3. IMAGE FORMAT

The Vista image format is most efficient for storing data defined on regular grids. An entry on this grid is called a voxel and may contain a scalar-, a vector- or tensor-valued quantity in different pre-defined number representations. Image attributes which provide a minimal description of the grid and the data representation are generated automatically by the Vista library routines (and thus called "automatic attributes").

Medical imaging data must be further qualified by additional attributes, which are listed in the section "Compulsory Attributes". Although none of the SimBio tools should rely on the presence of these attributes (i.e., provide reasonable defaults), their inclusion is necessary to allow a reliable use of medical imaging data.

Any number of optional attributes may be added. Note that this mechanism may be used to communicate information from one routine to another (such as a transformation matrix, material constants etc.).

3.1 Automatic Attributes

The three voxel dimensions of an image are called *band*, *row*, and *column*. The band dimension can serve a variety of purposes, including representing such things as the color channels of an RGB image, vector- or tensor-valued data. The remaining dimensions, row and column, index pixels by their vertical and horizontal coordinates. Vista's convention is to number rows and columns from the upper left pixel, which is at row 0, column 0.

The following Vista attributes are automatically generated when allocating a Vista image structure:

- data**: an unsigned integer, points to the first byte of the data of this image, as counted from the end of the header.

- length**: an unsigned integer, denotes the number of image data bytes.
- nbands**: a non-zero unsigned integer, denotes the total number of 2D slices (the z coordinate). If `nbands == 1`, this attribute may be left out.
- nframes**: a non-zero unsigned integer, denotes the number of 2D slice packages. If voxels represent scalar quantities, the number of frames equals the number of bands, in which case this attribute may be omitted. For vector- or tensor-valued quantities, a set of consecutive bands are collected as a single frame.
Example: a 3D RGB image is stored as a set of 2D frames, where each frame consists of three slices, representing the red, green and blue components. Thus, `nbands = nframes * 3`.
- nrows**: a non-zero unsigned integer, denotes image dimension in the y direction.
- ncolumns**: a non-zero unsigned integer, denotes image dimension in the x direction
- repn**: a string containing the voxel representation of the image:
 - bit** represents an unsigned integer in the range [0,1]
 - ubyte** represents an unsigned integer in the range [0,255]
 - sbyte** represents a signed integer in the range [-128,127]
 - short** represents a signed integer in 16 bits
 - long** represents a signed integer in 32 bits
 - float** represents a floating point number in 32 bits
 - double** represents a floating point number in 64 bits

Example: the following header describes an image of dimensions $256 * 256 * 128$ voxels in unsigned byte format:

```

image: image {
  data: 0
  length: 8388608
  nbands: 256
  nframes: 256
  nrows: 256
  ncolumns: 128
  repn: ubyte
}

```

3.2 Compulsory Attributes

The following Vista attributes are compulsory for image data in the SimBio environment. These attributes are required to describe medical image data:

- voxel**: a string containing three floating point numbers denoting the real world dimensions of a voxel in *mm* along the x-y-z axes.
- orientation**: a string describing the relation of an image slice (the x-y plane) to the body plane:
 - axial** a slice is perpendicular to the body axis
 - coronal** a slice is perpendicular to the fronto-occipital (i.e., nose-to-back) axis
 - sagittal** a slice is perpendicular to the left-right (i.e., ear-to-ear) axis

component_repn	#	Representation	component_interp (possible values)
scalar	1	a single component representing scalar information	intensity, potential uV, label, npartn, srcnode, elabel, epartn, bccode
vector3	3	any 3D vector valued component in the sequence x-y-z	force N, displacement mm, dipole, curvature, gradient
tensor6	6	any rank 2 symmetric tensor valued component in the sequence xx-xy-xz-yy-yz-zz	conductivity, elasticity, stress
rgb	3	three color channels in the sequence red-green-blue	
complex	2	two components (real and imaginary)	

Table 1. Possible image component representations and corresponding interpretations, with units where applicable.

For head data sets aligned with the stereotactical coordinate system, the orientation is axial by definition.

- convention**: a string describing the relation of image and body w.r.t. the body symmetry axis:
 - natural** the left image side corresponds to the left body side ¹
 - radiologic** the left image side corresponds to the right body side
- patient**: a string referencing a patient code. Note that the use of patient names is deprecated here. This information is only used to relate image data at a certain processing level to a specific subject.
- date**: a string containing the date and time of the examination. This information is only used to identify time-series examinations of subjects.
- component_repn**: a string defining the component representation of a voxel quantity, as shown in table 1.
- component_interp**: a string providing a meaningful interpretation of a voxel quantity and an optional measure unit, as shown in table 1. (This attribute is compulsory only where applicable.)
- ncomponents**: the number of elements in a voxel quantity, given in the column headed '#' in table 1; may be replaced by **ncolors** in RGB images and can be left out for images containing a single frame.

Example: the following header contains attributes describing voxel size, image orientation and convention, patient code, examination date and component representation and interpretation.

```
image: image {
  ...
  voxel: "1.5 0.976562 0.976562"
  orientation: axial
  convention: natural
```

¹Note that no default for the convention is provided. Thus, the coordinate system is either right-handed (natural) or left-handed (radiologic).

```

patient: PS1T000410
date: "11:56:34 10 Apr 2000"
component_repn: scalar
component_interp: potential uV
}

```

3.3 Additional Optional Attributes

Any number of optional Vista attributes may be added to further qualify an image. SimBio tools may simply ignore attributes which they do not recognize.

New attributes intended to communicate information between different SimBio tools should be communicated to the authors, providing name, format and intended use, for inclusion in a revision of this document.

3.4 Signal Images

Time-dependent information (such as EEG or MEG data) may also be stored as a Vista image. Most typically, this will be a 2D image, where each row represents a single channel, i.e., the columns represent the time points.

The following optional attributes characterize and identifies a Vista signal data set:

- nChannels**: the number of measurement channels (i.e., rows of the image). The presence of this attribute discriminates images from signal data sets. For signal data sets, the presence of this attribute is considered to be compulsory.
- sampleInterval**: a number specifying the sampling interval in ms.
- origin**: a number specifying the column corresponding to the trigger point.
- xAxisLabel**: a string specifying the x coordinate label (e.g., ms).
- yAxisLabel**: a string specifying the y coordinate label (e.g., μV).
- chanDDD**: a string containing information regarding channel DDD: electrode label, biosignal type, AD converter range, lower and upper limit of the frequency band.

Example: the following header contains the attribute of a 120-channels signal image:

```

image: image {
  ...
  nChannels: 20
  sampleInterval: 2
  origin: 0
  xAxisLabel: ms
  yAxisLabel: uV
  chan00: " Fp1/G19   EEG  300  0.530 70.000  0.000  0.000"
  chan01: " Fp2/G19   EEG  300  0.530 70.000  0.000  0.000"
  ...
}

```

The **sampleInterval** attribute may be also be present in non-signal 3D images (not containing any other signal attributes), in which case it indicates that the third (i.e. frame) dimension should be assumed to be located in the temporal rather than the spatial domain (i.e. the 3D image represent a sequence of 2D images).

3.5 Naming of Image Objects

All the images appearing in the preceding examples have appeared under the name ‘image’ (i.e. “image: image{...}”); here, the name is underlined). However, using more informative names can make the human readable header of SimBio files easier to read and also makes ‘filtering’ out images of a specific kind easier. Assigning a specific name to an image means that this image must possess certain properties in terms of its attribute values:

- IntensityImage**: used to denote a scalar image where the element values should be interpreted as intensities; `component_repn = scalar` and `component_interp = intensity`.
- SegmentedImage**: used to denote a segmented image where the elements have integer values that should be interpreted as class labels; `component_repn = scalar`, `component_interp = label` and `repn = {bit, ubyte, sbyte, short, long}`.
- ComplexImage**: used to denote an image where the elements have complex values; `component_repn = complex` and `repn = {float, double}`.
- RGBImage**: used to denote an image where the elements have RGB values; `component_repn = rgb`.
- 3DVectorField**: used to denote a field of 3D vector values; `component_repn = vector3`.
- DisplacementField**: used to denote a displacement field; attributes as `3DVectorField` and `component_interp = displacement`.
- ForceField**: used to denote a force field; attributes as `3DVectorField` and `component_interp = force`.
- SymmR2TensorField**: used to denote a field of symmetric rank2 tensor values; `component_repn = tensor6`.
- DipoleField**: used to denote dipole moments in the brain; `component_repn = vector3` and `component_interp = dipole nA`.

Note that the naming of images is optional and that applications using the SimBio file format are not required to recognize or be able to interpret the above names. For that reason, compulsory attributes can not be omitted from an image, even if they would be uniquely determined by name of the image. There is no requirement for applications using the SimBio file format to make sure that the name of an image agrees with its attribute values, but doing so would be considered as good software engineering practice.

Additional names for images with specific properties can be added if needed.

3.6 Working with Vista Images

This excursion is intended to give a brief overview of some Vista routines to work with the image format. Additional information about Vista images is compiled in the manpage `VImage`.

Images are created with the command `VCreateImage`:

```
VImage im = VCreateImage(nz, ny, nx, VUByteRepn);
```

where `nx`, `ny` and `nz` give the image dimensions (note the ordering!) and `VUByteRepn` corresponds to the `ubyte` voxel representation. In analogy, the line:

```
VDestroyImage(im);
```

releases storage allocated by the previous call.

An attribute is added to an existing image using:

```
char pat[MAX_NAME_LENGTH];
```

```
VSetAttr(VImageAttrList(im), "patient", NULL, VStringRepn, pat);
```

and, likewise, queried by:

```
VGetAttr(VImageAttrList(im), "patient", NULL, VStringRepn, pat);
```

An attribute is destroyed by the following operations:

```
VAttrListPosn posn;
```

```
if (VLookupAttr(list, "condition", &posn) == True)
    VDeleteAttr(&posn);
```

Note that a newly created image may inherit a complete set of attributes by the call:

```
VCopyImageAttrs(src, dst);
```

The `Vattribute` manpage provides more information about Vista attributes and their usage.

A specific voxel may be accessed by one of the following mechanisms. A generic function retrieves a value at a given position:

```
double v = VGetPixel(im, z, y, x);
```

The symmetric call `VSetPixel(im, z, y, x, v)` stores `v` at position `(x, y, z)`. Note that these functions are independent of the image representation, however, some overhead due to the function call must be expected.

A more efficient method is to use a macro:

```
double v = (double)VPixel(im, z, y, x, VUByteRepn);
```

or a pointer:

```
VUByte ***p = VPixelArray(im, VUByteRepn);
double v = (double)p[z][y][x];
```

Images are being read from and written to files using the commands `VReadFile` or `VReadImages` and `VWriteFile` or `VWriteImages`, respectively. These commands operates on Vista attribute lists containing the images stored under their respective names.

```
VAttrList outlist = VCreateAttrList();
VAppendAttr ( outlist, "image", 0, VImageRepn, im );
FILE outfile = fopen( "im.v", "w" );
VWriteFile( outfile, outlist );
fclose( outfile );
```

4. GRAPH FORMAT

The Vista graph format may be used to store a general graph structure in a file. It is the preferred format for storing data defined on irregular grids (such as surfaces in 3D or finite element nets). A Vista graph is comprised of a list of nodes and connections between nodes. Nodes and connections may have weights, connections may be uni- or bidirectional in a graph. As for images, a graph may have an arbitrary list of associated attributes. Attributes which provide a minimal description of the graph layout are generated automatically by the Vista library routines (and thus called "automatic attributes").

A node consists of some book-keeping fields and some user-defined fields. Customized node representations are instantiated by subclassing from the structure `VNodebaseStruct`. Two restrictions are enforced for Vista graphs: all nodes in a graph instance must have the same class (i.e., occupy the same amount of storage); all user fields in a node must have the same representation.

A node in a graph may be referenced either by sequencing operations (i.e., iterators), by walking along links, or by directly referencing entries in the node table. Entries in the node table may be empty (e.g., as a result of a node deletion). In order to distinguish between valid and invalid node references, the first entry (at table position 0) is empty by definition, i.e., 0 denotes an invalid or empty node reference.

In a sample application within SimBio, a single graph contains a set of vertices in 3D; links between nodes correspond to edges. Let us call such a structure a *vertex graph*. While this graph is sufficient to represent a surface or volumetric mesh, it is useful to add a second graph containing geometrical primitives. For surface meshes, this *primitive graph* contains polygons as nodes, for volumetric meshes, nodes represent cells (i.e., tetrahedra or hexahedra) of the finite element mesh. Links between nodes denote neighborhood relationships between primitives: in the case of surfaces, neighbors share edges, in the case of volumetric meshes, neighbors share faces.

In the presence of a primitive graph only containing triangular, quadratic, tetrahedral or hexahedral primitive elements, it will be possible to allow these primitive elements to also implicitly define the connectivity in the associated vertex graph, where the explicit links in this case may be omitted, leading to savings in terms of storage.

The implicit links for the different primitive elements are shown in figure 1. Note that, this also restricts the orientation of the primitive element in terms of

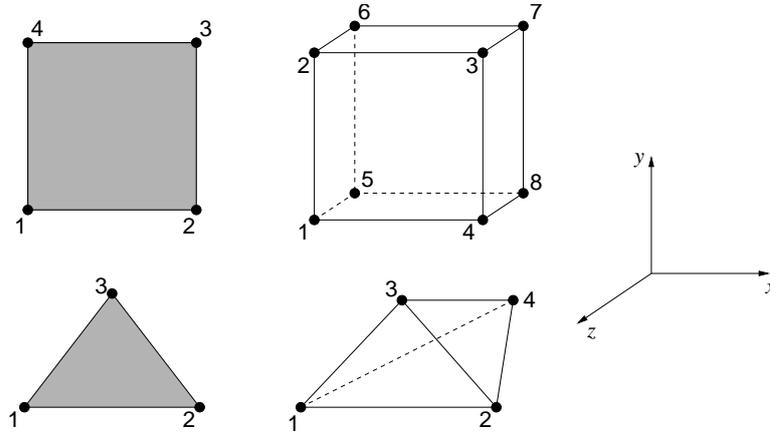


Fig. 1. The implicit connectivity and ordering of vertices in primitive elements. The vertex numbers corresponds to the user defined fields of the primitive nodes (see sec. 4.6). For the triangular and quadratic surface elements (left), the numbers also orders the vertices, which follow a counter-clockwise ordering when the element is aligned with the x - y -plane and the frontside facing in the direction of the positive z -axis of a right hand oriented coordinate system (the positive z -axis pointing towards the reader). For the tetrahedral element (bottom right), the orderings of the vertices along the sides of the element, when aligned with the x - y -plane and the outside facing in the direction of the positive z -axis, are: (1, 2, 3), (2, 4, 3), (4, 1, 3), (1, 4, 2). For the hexahedral element (top right), the orderings of the vertices along the sides of the element, when aligned with the x - y -plane and the outside facing in the direction of the positive z -axis, are: (1, 2, 3, 4), (8, 7, 6, 5), (4, 3, 7, 8), (5, 6, 2, 1), (2, 6, 7, 3), (5, 1, 4, 8). Note that, this ordering is clockwise, in contrast to the other elements, where the ordering is counter-clockwise.

the spatial ordering of its vertices with respect to the coordinate system, as defined in the caption of figure 1. Intuitively, this spatial ordering defines the front- and backsides for surface elements and in- and outsides for volume elements. A general, mathematically strict definition of these restrictions for hexahedral elements can be found in [7, sec. 3.5]. To use this implicit connectivity, the attribute `implicit_links` must be defined in the attribute list of the primitive graph with the string value `true`. Implicitly defined links are, in contrast to explicitly defined links, always bi-directional.

The correspondence between images and graphs is given by the following convention: Vertices represent points in the domain, primitives represent some subvolume of the domain. Thus, an image may be considered as a mesh containing hexahedral primitives with vertices on the corners of the hexahedra.

4.1 Automatic Attributes of General Graphs

Vista graphs have automatic attributes with keywords `data`, `length` and `repr` which have the same meaning as for Vista images. In addition, the following attributes are predefined for any graph:

- `useWeights`: a boolean variable denoting whether the nodes and links in the graph contain additional weight fields. For SimBio application, most likely, weights are not used.

- `nnodes`: an unsigned integer, denotes the number of nodes in a graph.
- `nfields`: a non-zero unsigned integer, denotes the number of fields of representation `repn` in the user portion of the node structure.

4.2 Compulsory Attributes of General Graphs

For SimBio applications, it is suggested to include the attributes `patient`, and `date`, as defined for the image format, in all graph structures.

The attribute `component_interp` contains a string defining the interpretation of a node. For SimBio, the following values are of interest:

```
vertex      this graph contains vertices
primitive   this graph contains primitives
```

If there is only a single graph in a file, it is assumed to contain vertices.

4.3 Node Attributes of General Graphs

Vertex and primitive graphs together define a geometrical model of an object, in case of SimBio, (a part of) a head or a knee. Typically, we are interested in describing not only the geometry of such objects, but also other properties (e.g. material properties), which vary across the geometrical model, and maybe associated with either vertices or primitive elements.

We do this by storing the properties of interest as an attribute in the form of a table, represented by a Vista image, where each column correspond to a unique node in the graph (a vertex or a primitive node).

Example: part of a header for a primitive graph representing a volumetric mesh, where the elasticity tensor associated with each volume element is stored in the correspondingly labelled attribute image.

```
graph: graph {
...
        nnodes: 10
        nfields: 4
        repn: long
        component_interp: primitive
        primitive_interp: volume
        implicit_links: true
        image: image {
            data: 0
            length: 240
            nbands: 6
            nframes: 1
            nrows: 1
            ncolumns: 10
            ncomponents: 6
            repn: float
            component_repn: tensor6
            component_interp: elasticity
        }
}
```

Note that several node properties may be represented this way, by separate attribute images. Moreover, using attribute images with multiple frames, it will be possible to represent temporally changing characteristics of a temporally constant geometrical model.

Just like ordinary images, the attribute images need certain attributes of their own. In addition to the automatic image attributes (sec. 3.1), `component_repn` and `component_interp` attributes are required to qualify the image content. Other attributes may be deduced from the containing graph (e.g. `patient` and `date`) or may not be applicable (e.g. `voxel` and `orientation`).

If the `sampleInterval` attribute is present and `nframes` is greater than one, the frames are understood to contain a sequence of values. A special case of this occurs when the attribute image is a signal image (sec. 3.4).

Note that attribute images may also be used to store other attribute information, which may be node specific or not, e.g. partitioning information for FE applications and striping information for visualisation. These images usually contain information which is specific to a single or a few closely connected applications and will have unique attribute names. All applications supporting the SimBio file format must be able to deal with the presence of such images, in the simplest case by just ignoring them.

4.3.1 Sparse attribute images. Certain attribute values may be associated with only a subset of the nodes in the graph. When the number of nodes is large and the subset of nodes that have attribute values associated with them is small, using an attribute image with a column for each node can waste a lot of memory, especially when the attribute values themselves are storage demanding, like vector or tensor values.

For these cases, a dedicated sparse attribute image definition is available. This consists of a Vista attribute list [1] containing two Vista images. The first image is the *reversed index image* (ri-image) and the second the *value image* (v-image). Both images have the same number of columns. The ri-image has an integer representation (`ubyte`, `sbyte`, `short`, `long`) and all ‘voxels’ have strictly positive values. The v-image has the same characteristics as ordinary attribute image, except that it may have fewer columns than there are nodes in the associated graph. The value stored in column i of the ri-image gives the index of the node in the graph (starting from 1) to which the value stored in column i of the v-image belongs. A sparse attribute image (i.e. the corresponding attribute list) should always be stored under the name `sparse_image` in the attribute list of the graph. In the sparse attribute image, the ri-image should always be stored under the name `revidx_image`, whereas the v-image could have any name, but typically would have one of the descriptive names from section 3.5.

The v-image may have the additional attribute `empty_value`, which contains the value, represented as a string with space as a separator, associated with all nodes for which the sparse attribute image does not specify a separate value.

Example: header for a vertex graph with a sparse attribute image.

```

graph: graph {
  data: 0
  length: 240
  useWeights: 0
  nnodes: 10
  nfields: 4
  repn: float
  patient: "K. Rank"
  date: "12:23:16 8 Jan 2002"
  component_interp: vertex
  sparse_image: {
    revidx_image: image {
      data: 240
      length: 12
      nrows: 1
      ncolumns: 3
      repn: long
    }
    ForceField: image {
      data: 252
      length: 36
      nbands: 3
      nframes: 3
      nrows: 1
      ncolumns: 3
      repn: float
      component_repn: vector3
      component_interp: "force N"
      ncomponents: 3
      empty_value: "0.0 0.0 0.0"
    }
  }
}

```

4.4 Nodes in a Vertex Graph

In order to agree on how user-defined fields in a node are used in SimBio applications, the first field contains a type code. Currently, the following type codes have been defined:

Type	Usage	Field #	Fields
1	simple vertex	4	x y z
2	vertex + normal	7	x y z nx ny nz
3	vertex + normal + curvature	10	x y z nx ny nz cn cg cm
4	vertex + scalar	5	x y z s
5	vertex + normal + scalar	8	x y z nx ny nz s
6	vertex + normal + curvature + scalar	11	x y z nx ny nz cn cg cm s

Note that this table is easily extendible by definition of additional type code and agreement on their usage.

Because all fields must have the same representation, the preferred data representation for vertex graphs in SimBio applications is `VFloat`.

It is convenient to have vertex coordinates in real world dimensions (i.e., a voxel attribute is not necessary). In addition, if images are to be used together with vertex graphs, it is assumed that they live in the same coordinate frame.

4.5 Optional Attributes of Vertex Graphs

For SimBio applications, it is necessary to further qualify the information contained in a graph.

- `partition(s)`: the number of processors the application that is based on this mesh will run on.
- `vmlinestrip`: attribute image containing containing pre-computed information for visualisation.
- `vertex_interp`: a string qualifying the vertex information. This attribute may indicate that vertices are to be interpreted as electrode or SQUID positions, current or force sources.
Example: vertex_interp: electrode
- `scalar_interp`: a string qualifying the scalar information. This attribute should indicate the meaning and unit of the scalar.
Example: scalar_interp: "voltage uV"

In order to interpret the information correctly (e.g., display electrodes by predefined glyphs), these attributes should be considered as compulsive.

4.6 Nodes in a Primitive Graph

A node in a primitive graph represents a geometrical object in a mesh. The first field (field 0) contains the number of vertices belonging to this primitive, subsequent fields contain the references to the vertices in the table of the vertex graph.

Example: The primitive node containing the fields `3 57 1 14` denotes a triangle referenced by nodes 57, 1, and 14 of the vertex graph.

It is possible to mix primitives (e.g., triangles and quadrilaterals) in a graph. However, since only a single node type is allowed in a graph, all nodes must provide

the same number of fields. Because fields contain references to the vertex graph, the preferred data representation for primitive graphs is `VLong`.

For SimBio applications, surface meshes may contain triangles and quadrilaterals, volumetric meshes may consist of tetrahedra and hexahedra.

4.7 Compulsory Attributes of Primitive Graphs

In order to distinguish between surface and volumetric meshes, the presence of following attribute is required:

—`primitive_interp`: a string indicating the mesh type:
 `surface` a surface mesh
 `volume` a volumetric mesh

4.8 Optional Attributes of Primitive Graphs

—`implicit_links`: if and only if this attribute present with the string value `true`, the primitive graph defines implicit connections between the vertices in its corresponding vertex graph.
 —`vmlinestrip`, `vmpolystrip`: attribute images containing containing pre-computed information for visualisation.

4.9 Working with Vista Graphs

As for images, we now give a brief overview of some Vista routines to work with the graph format. Complete information about Vista graphs is compiled in the manpage `VGraph`.

Graphs are created with the command `VCreateGraph`:

```
VGraph vtx = VCreateGraph(nnodes, nfields, VFloatRepn, False);
```

where `nnodes` corresponds to the initial table length, `nfields` to the number of fields of representation `VFloat`. The last argument indicates that weights are not used in this graph. Note that the table may grow automatically when nodes are added to the graph. The call `VDestroyGraph(vtx)` releases all storage allocated for this graph. Attributes are added in a similar fashion as described for images:

```
VSetAttr(VGraphAttrList(vtx), "scalar_interp", NULL,  
         VStringRepn, "voltage uV");
```

Nodes are added to the graph by the call:

```
VNode node;
```

```
unsigned int ref = VGraphAddNode(vtx, node);
```

Note that the information in `node` is copied in the graph, so re-using (or deallocating) `node` is safe. The routine `VGraphAddNode` checks if a node containing the same information was added before using a linear search through the table. For certain situations this performance penalty may be avoided by using the call `VGraphAddNodeAt(vtx, node, ref)`, which places a node at the specified position. Here, the specified position must be within the table.

A link is added between two nodes by the command:

```
VGraphAddLink(vtx, ref1, ref2);
```

Note that this link is uni-directional (i.e., from node `ref1` to node `ref2`). A bi-directional connection between the two nodes can be established by simply also adding a link from node `ref2` to node `ref1`:

```
VGraphAddLink(vtx, ref2, ref1);
```

The simplest way of traversing a graph is to use an iterator:

```
for (VNode node = VGraphFirstNode(vtx); node;
     node = VGraphNextNode(vtx)) { ....
```

which is equivalent to:

```
for (unsigned int ref = 1; ref <= VGraphNNodes(vtx); ref++) {
    VNode node = VGraphGetNode(vtx, ref);
    if (node == 0) continue;
    ...
}
```

An alternative is to walk along the links of a node:

```
VNode node;

for (VAdjacency adj = node->adj; adj; adj = adj->next) {
    unsigned int ref = adj->id;
    VNode neighbor = VGraphGetNode(vtx, ref);
    if (neighbor == 0) continue;
    ...
}
```

5. SUMMARY

The Vista toolkit defines a compact, efficient and machine-independent file format, which is suitable to map data structures within the SimBio environment. Additional conventions are described in this document which are necessary for the target application in the medical field.

REFERENCES

- [1] Pope A.R., Lowe D.G. (1994) Vista: A software environment for computer vision research. In *Computer Vision and Pattern Recognition (CVPR'94)*, pp. 768-772. IEEE Press, Los Alamitos.
- [2] Kruggel F., von Cramon D.Y. (1999) Alignment of magnetic-resonance brain datasets with the stereotactical coordinate system. *Medical Image Analysis* 3, 1-11.
- [3] MacWilliams B.A., DesJardins J.D., Wilson D.R., Romero J., Chao E.Y.S. (1998) A repeatable alignment method and local coordinate description for knee joint testing and kinematic measurement. *J. Biomechanics* 31, 947-950.
- [4] Talairach J., Tournoux P. (1988) *Co-Planar Stereotactic Atlas of the Human Brain*. Thieme, Stuttgart.
- [5] The SimBio Consortium (2000) SimBio: A generic environment for bio-numerical simulation. <http://www.simbio.de>.
- [6] Pope A.R., Lowe D.G. (1998) The Vista toolkit: <http://www.cs.ubc.ca/nest/lci/vista/vista.html>.
- [7] Hughes T.J.R. (1987) *The Finite Element Method*, Prentice-Hall.