Volume II

Khoros Programmer's Manual

Chapter 10

VIFF

Khoros Visualization and Image File Format

Primary Author(s):

John Rasure

Copyright © 1992, 1993 University of New Mexico. All rights reserved.

Printed: September 16, 1992

VIFF

A. VISUALIZATION and IMAGE FILE FORMAT (VIFF)

The Khoros Visualization and Image File Format, VIFF, is both an internal data structure and a file storage format (see `$KHOROS_HOME/include/kformats/viff.h`). VIFF was designed to facilitate the interchange of data, not only between diverse data processing functions within Khoros, but also between researchers who want to exchange data and information. VIFF was designed to specifically support the extremely diverse fields of image processing, computer vision, and data visualization. But in general, VIFF can be used to store complicated data structures located in four dimensional space and time.

The VIFF data structure, *typedef viff*, is an internal format that programmers should **never** directly access. The data services library provides all of the data access functions and attribute setting functions that the programmer needs to get and set VIFF data or data structure members. The abstract data type defined by the data services is called the *kobject*. This abstract applications programmer's interface, sometimes referred to as an API, allows Khoros to utilize other internal data formats and supports the evolution of VIFF.

The Khoros **data model** encompasses more than just VIFF, but it is important to first understand the overall design of the VIFF. This knowledge will lead to a better understanding of the implementation of the Khoros data model. The other components of the Khoros that implement the data model are the data services and data transport libraries of the **phantom** distributed computing toolkit. (These libraries are concerned with other issues as well as the data model).

The VIFF design came from hundreds of hours of discussions. I would like to thank Peter Averkamp, Wes Bethel, Chuck Hansen, Donna Koechner, Jeremy Worley, and Mark Young for their contributions.

A.1 VIFF ORGANIZATION

The Khoros Visualization and Image File Format is organized as data attributes followed by up to seven related data segments; tag data, map data, explicit location data, explicit time data, connectivity data, mask data, and finally value data. The VIFF data structure contains eleven categories of attributes in order to represent time dependent image and visualization data located in N-dimensional space. The data attributes, sometimes referred to as the file header, provide all of the information necessary to determine the size of the data segments and to convert the data to conform to the the local architecture. The term *data* is used to generically mean all data from all seven segments. The term *value data* is used to mean the data that represents actual values, not the data's location, connectivity, color interpretation, or mapped interpretation. The term *value vector* is a structure of value data (it could even be another VIFF); a specific example of a value vector is the set of three data values composing a color vector. In the following discussion, one must always be careful to realize that a value vector is a set of value data.

The VIFF structure is designed to accommodate value vectors at both implied locations (typically a uniform grid) and at explicit locations. The VIFF also accommodates value vectors at both implied time (typically uniform sampling) and explicit time. If one is only concerned about value vectors that have spatial and time locations **implied** by an array index, then the VIFF

is referred to as a five-dimensional structure. Three spatial dimensions and one time dimension provide the location of a value vector. A value vector is considered, for simplicity, to have one dimension; thus the VIFF is said to be a five-dimensional structure. If one is concerned about value vectors that have spatial locations **explicitly** defined, the VIFF is referred to as a structure of value vectors in a N-dimensional location space and time. Location data in the VIFF does not change as a function of time, only value data can change with time and space.

Information on an existing file can be obtained by using **--info--**. The header field categories are:

Administrative Information

The administrative information is contained in the first few bytes of the VIFF header. A VIFF file is identified by the **identifier, file type, release major**, and **release minor** fields.

Data Storage

The data storage fields describe how the data is stored but not how it is to be interpreted. The fields in this group are **machine identification, storage orientation** (vector or plane), and **data encode scheme**. The data storage orientation can greatly effect the efficiency of the processing of the data. The encoding scheme is used to indicate how the data is compressed or represented. The fields in the header are valid after the data is decoded.

Value Data

The amount of the value data is determined from the **width, height, depth, time**, number of value vector **elements**, and **value storage type**. In the case of implicit locations, the VIFF can be thought of as a `time` series of volumes that are of size `width*height*depth`. At any location within the series of volumes, there can be a value vector of size `elements`. Currently, only value vectors that are simple one-dimensional arrays of fixed length are supported. Using a true color image as an example, a value vector of `elements=3` specifies how many data values make up a special kind of value vector called a **color vector**. In other words, a color vector is made up of several value data.

The physical size or the sampling interval of the value data is specified by **delta width, delta height, delta depth, delta time** and **delta elements**. The units might be seconds or meters, but are application dependent. (Unit names can be put in the tag field.)

As explained below, the **width, height, depth,** values are used to index other data segments besides the value data segment. Since it is possible to have a VIFF with no value data segment, the header field **value type** is used to indicate the presence of value data.

Mask Data

A mask data segment may optionally be present. The mask size and storage orientation is exactly the same as that for the value data (i.e. five dimensional), but the storage type of the mask data is always bit. The interpretation of the mask is up to the application programmer, but typical uses are to gate operations on the data values or to indicate the validity of data. The header field **mask type** is used to indicate the presence of mask data.

Map Data

The value data can be used as an index into the optional map data segment to retrieve attributes about that value data. The map size is determined from the fields **map storage type**, number of map vector **elements**, number of entries in the map **range**, and number of map

planes. The **map enable** and **mapping scheme** fields determine how the map is to be used during value data processing. It is interesting to note, that a value datum results in a vector of data when mapped through a map vector.

Color Space

The color space coordinate system and model of the value data is determined by the **color space model** field. Color space models can be generic, or meet a specific standard. For example;

- the elements of the color vector can be represented as a value datum acting as a index into a color gamut stored in the map data,
- or the elements of the color vector can be a set of three data values stored as RGB planes (red plane, green plane and then blue plane),
- or the elements of the color vector can be a set of three data values stored as an RGB vector.

The key concept is that one or more data values is used to make a color vector and that a color space model may optionally use the map data.

Location Data

The **location type** field indicates whether the value data has implicit or explicit location data. Explicit location data requires that a location data segment be present. The **location dimension** field indicates the number of spatial coordinates or the size of the location vector. The number of spatial location vectors is equal to the number of value vectors at one instant of time. Therefore, the total amount of location data is a function of **width, height, depth, location dimension** and the **location storage type**. The location data of a value vector is found by using the same index ([w][h][d]) that points to the value vector. Note from the description above that all data values in a value vector have the same location in N-space and that the location of a value vector can not change with time.

Time Data

The **time type** field indicates whether the value data has implicit or explicit time data. Explicit time data requires that a time data segment be present. For each volume of value vectors, there can be a single scalar time value. The total amount of time data is a product of **time** and **time storage type**.

Connectivity Data

VIFF can be used to store data **primitives**. In most cases, the data in the VIFF is not given any specific primitive name, but when the correct interpretation of the VIFF depends on knowing what kind of primitive the VIFF contains, the name of the primitive is stored in the **connect type** field. The connect type field also describes the kind of value vector connectivity that the primitive has. The **connect type** of a primitive's value vectors can be none, implied by value data storage order, or explicitly listed as indices.

As an example, explicit connectivity data might be a list of triangle vertices or a list of polygon vertices. Where a vertex number (a scalar) is a function of **width, height, depth**. The calculated index is used to find the value vector and locaton vector that corresponds to each vertex. As another example, value vectors might be used to store attributes of spheres or directed points that do not have any connectivity and the **connect type** field just indicates that the VIFF contains that kind of primitive.

The size of the connectivity data is determined by the **number of primitives** (one when connectivity of the primitive is none), and the first entry in the list of the connectivity data for each primitive, which specifies how many indices there are for that primitive. This calculated total size is stored in **connect size**.

Tag Data Tag data can be used to add custom fields and comments to the VIFF. The number of tags in the tag data is specified by the **tag number** field and the total size of all of the tag data in bytes is given by **tag size**.

Name Data The VIFF structure includes pointers to the data, but when one or more of the data segments is in some sort of data transport mechanism supported by the data services library (disk file, shared memory or socket, etc), a reference to the data, stored as a character string is stored instead of the actual data. This allows for the VIFF header to be separate from the data and allows multiple VIFFs to share data segments.

It is possible for the size of any of what the data segment pointers point to to be zero, but not all can be zero. In other words, a VIFF must contain at least one segment. The seven data segment pointers are: tag data, map data, location data, time data, connectivity data, mask data, and finally, value data.

The above discussion uses several terms that have not been clearly defined. These terms are defined in the following sections by describing the concepts that are essential to understanding VIFF and the data services library. Examples of storing specific kinds of data are given to reinforce the terminology and concepts. Each data segment that is stored in the VIFF is covered, starting with the value data segment and progressing to the more complex connectivity data segment. Hopefully, this document will give the reader an intuitive understanding of the VIFF and how it might apply in his/her specific application.

A.2 VALUE DATA TERMINOLOGY and DESCRIPTION

A value data segment is used to store **points, lines, planes, volumes, sequences** and **data structures**. All of these terms have an associated number of dimensions and represent a set of value data. The reader needs to be careful not to expect the definitions given for these and other terms to perfectly fit terminology they might already have in mind or be familiar with. For example, a VIFF plane can be defined using any two of the five dimensions of VIFF, but depending on the two dimensions used, the term plane takes on very different meanings and has different synonyms (i.e. band, image, raster). This introductory discussion only considers value data with implicit location and connectivity; explicit location data, explicit time data, and explicit connectivity data are discussed later.

A.2.1 Points, Lines, Planes, Volumes and Sequences

Implicit value data is considered five dimensional. The terms point, line, plane, volume, and sequence are zero-dimensional (0D), one-dimensional (1D), 2D, 3D and 4D, respectively. Three of the dimensions are spatial (width, height, depth) and the other one is time. At any 4D position there can be a data structure or more typically a line of value data referred to as a **value vector**. The 1D value vector is indexed using elements. Thus, any point of data or single value datum is indexed using `width` (w), `height` (h), `depth` (d), `time` (t), and `elements` (e).

A value vector is found using the w, h, d, and t indices and indexed using the element number e. Even though a value vector is a *line* of data, a line of value data can be taken from any orientation of an implicit VIFF structure. An orientation is set by tying one of the w, h, d, t, or e, indices to be the line index. A row of data from a plane, or a column of data from a plane, or a value vector, are all lines of data values.

Later, an implicit image is described as a plane of value vectors perpendicular to depth. But in general, a plane can be taken from anywhere in a VIFF using any two indices out of five. As an example of this, it is conceivable to process planes of data that are indexed using e and t as the plane dimensions and w, h, and d as indices to the series of planes.

The terms point, line, plane, volume, and sequence, only indicate dimensionality and are not tied to specific indices. However, the indices of width, height, and depth do correspond specifically to space, the index of time corresponds to time, and the index of elements corresponds to a value datum of a value vector.

The next two sections define the terms signal and image. An image is defined to be a plane tied to specific indices, or in other words an image is a plane with a required orientation. While a signal is defined to be a line with a default orientation because there is not a single best definition for a line. To restate, the terms signal and image, specify the relationship between the indices w, h, d, t, and e, and the terms line and plane.

A.2.2 Storage of Signals (1D)

There are at least four useful interpretations of VIFF value data storage that make sense in the context of a signal processing.

1. A time series of value data could be considered to be a signal. Every w, h, d, e index combination has a signal of length t. If w, h, d, and e, are all 1, then there is just one signal.

This kind of signal is a function of time. (It is useful to note that the time index may not represent time in the conventional sense, it could be one over time if the signal is in the Forier domain).

2. A value vector, as defined earlier, could be treated as a signal. Every w, h, d, t index combination has a signal of length e. This kind of signal is not a function of time or space, but just a collection of data at one time and at one location in space.
3. A row or column of values within a plane could be a signal.
4. Any plane of the value data within a VIFF could be treated as a signal. Even though a plane is defined to be 2D, there will be applications that wish to treat it as 1D.

For each interpretation, the number of elements in the signal and the number of signals is determined from the independent variables width, height, depth, time and elements.

There is not a single method, or even a common few methods, of storing signals that are sufficient for all applications. Since the term signal is no more specific than the term line, a **signal** is defined as a line tied to the time index. A signal is therefore equivalent to a line with the default orientation. The data services library functions and the programmer have the freedom to choose a line orientation that best fits what the programmer is calling a signal.

A.2.3 Storage of Images (2D)

An *implicit image* is a two-dimensional spatial array of value vectors. A single value datum at a specific `width` (w) and `height` (h) is defined to be a *pixel*. When an image has more than a single value at one two-dimensional implied location, then the data at that point is defined to be a *pixel vector*.

The VIFF default implicit image origin (0,0) is at the upper, left-hand corner of the plane perpendicular to the depth axis. The column or width index, w, proceeds across the image to the right (horizontal); the row index or height, h, proceeds from the top of the image to the bottom (vertical); and the depth index, d, proceeds from front to back. The value data in a pixel vector is indexed using e. (The default origin comes from how raster image data is acquired and can be overridden using the tag fields).

Multi-band images, multi-spectral images, and color images can be stored in VIFF. By the above definition, the `depth` (d) of all of these multi-element images is one (1), and the number of bands or spectra is set using (e), the number of `elements` per pixel vector. In general, these images are two-dimensional spatial arrays of value vectors. For example, a seven band, multi-spectral, remotely sensed, image has `elements=7`. A true RGB image has `elements=3`, and the color space model field set to an appropriate model so that the multi-band image is treated as a color image. It is left up to the programmer to decide whether to access the image as `width*height` pixel vectors or `element` bands. For example, the RGB image can be accessed as RGB triplets for each pixel or as image planes of Red, then Green, and then Blue.

A pixel vector is defined to be a **color vector** if the **color space model** header field is set. A color vector is stored as a combination of several data values or, as described later, by mapping a value datum through map data.

An image is characterized by having two spatial dimensions and one or more value data at each point in the image. This so called pixel vector adds a third dimension to the data set, but does not make it a volume of data. A volume of data is characterized by having three spatial dimensions and is stored in VIFF as described later.

A.2.4 Storage of Volumes (3D)

An *implicit volume* is a three-dimensional spatial array of value vectors. A single value datum at a specific `width` (w), `height` (h), and `depth` (d), is defined to be a *voxel*. When a volume has more than a single value at one three-dimensional implied location, then the data at that point is defined to be a *voxel vector*.

Remember how the previous discussions on signals and images defined the terms **signal** and **image** tied to specific indices? An image is tied to w and h, while a signal is tied to the index t. The term volume was introduced at the beginning of the discussion on value data as a general 3D set of data, but now it is being defined to be tied to specific indices (w, h, and d provide the index to a voxel vector, e is the vector index, and t is the index of a series of volumes). Although thought was given to inventing a new term for the more specific definition, no term seemed adequate. So, the reader must infer whether a **default orientation volume** or a general volume definition is appropriate based on the context of the discussion.

The VIFF default volume origin (0,0,0) is at the upper left-hand, front corner. The column or width index, w, proceeds across the front plane to the right (horizontal); the row index or height, h, proceeds from the top of the volume to the bottom (vertical); and the depth index, d, proceeds from front to back. The value data in a voxel vector is indexed using e.

A data volume is sometimes referred to as a multi-band image. By the definitions above, this is only true if each image band, slice, or plane has a different spatial location or depth value. So, the term multi-band image is confusing because the band number can come from e or d depending on whether or not the band data corresponds to attributes of a pixel or describes another voxel. That is why this document avoids the use of the ill defined term **multi-band**.

A volume or 3D set of true color RGB images is stored as `width * height * depth` value vectors of `elements=3`. As another example, consider a magnetic resonance spectroscopy experiment that produces a volume of 512-point spectra. If the data was gathered at 10 evenly spaced times, the data would be stored as `width * height * depth` voxel vectors of `elements=512` and `time=10`. This time series of volumes fully utilizes the five dimensions of the VIFF.

A.2.5 Storage of Sequences (4D)

The general 4D definition for the term **sequence** given earlier is the same definition that is used here except that it is given a default orientation. Remember, a **sequence** is defined to be a 4D set of lines with any posssible orientation. A signal is a specific kind of line, and an image is a specific kind of plane, and the term volume takes on its definition (general or default) given the context of the discussion. A **default sequence** is tied to w, h, d, and t; thus, a default sequence is a set of value vectos in 3D space and time.

A set of sequences is stored in a VIFF using w, h, d and t as the index to a specific value datum in the sequence, and using e to index a specific sequence. Just as there are pixel vectors for images and voxel vectors for volumes, there are sequence vectors for sequences. To repeat using alternate words, the five-dimensional, implicit location VIFF, can contain a time series of volumes, i.e. a sequence, of value vectors.

To summarize the discussion of points, lines, signals, planes, images, volumes, and sequences, a few examples are given.

Example 1.

VIFF can store a time series of RGB volumes using the three variables width, height and

depth, to specify the volume, the elements variable to specify the number of data values in a value vector or color vector, and the time variable to specify the number of volumes in the series.

Example 2.

VIFF can store a time series of multispectral images using the two variables width and height to specify the size of the image, the elements variable to specify the number of spectral components, and the time variable to specify the number of multispectral images. The depth variable is equal to 1.

Note that the terms sequence and series SHOULD NOT BE USED as synonyms.

A.2.6 The Value Vector as a General Data Structure

The value vector is intended to eventually be treated as a general data structure located in space and time. But this can not fully be realized as long as the elements variable and the value data storage format is constant for the entire VIFF. The application programmer is still free to access a value vector as a line of data and then assign the value vector to any kind of data structure they want. The data services library may automate this in the future.

A.2.7 Storage Orientation

The storage orientation of the 5D value data can greatly affect the performance of data processing due to the operation of the computer's virtual memory system. (If the value data indices cover a large space, striding the indices can cause a great deal of swapping.)

The discussion above describes a series of volumes of value vectors as `time` three-dimensional arrays (`volume = width * height * depth`), of vector size `elements`. This five dimensional data set can be stored and indexed in many possible ways:

- a set of consecutive two-dimensional arrays (`number of planes = depth * elements * time, plane size = width * height`),
- a set of consecutive three-dimensional arrays (`number of volumes=elements * time, volume size=width*height*depth`),
- or a set of consecutive one-dimensional arrays (`line length=elements` and `number of lines = width * height * depth * time`),
- ..... etc.

The point is that the implementation of the VIFF reading and writing utilities become too complex if all possible storage orientations are allowed. At this time, the **storage orientation** field can indicate one of two possible storage formats. The two formats where chosen based on the data organization needs of the more common algorithms. Remember that the storage orientation is transparent to the programmer because of the data service functions. A time series volume of VIFF value vectors is stored either as:

- A five-dimensional array with index order of `width` by `height` by `depth` by `time` by `elements`. This can be thought of as a consecutive series of planes or planer storage. Many image processing programs prefer data stored in this orientation.
- A five-dimensional array with index order of `elements` by `width` by `height` by `depth` by `time`. This can be thought of as vector storage. This storage orientation is often used for color image processing, clustering, and classification.

NOTE The terms interleaved and non-interleaved are often used when discussing image storage formats, but these terms are not used in this discussion because VIFF contains too many dimensions in which to interleave or not to interleave. It is just too confusing to try to define interleaving in the context of VIFF.

A.2.8 Mask Data Segment

A VIFF can optionally store mask data. The 5D mask data has a one-to-one correspondence with the value data. Mask data can only have the values of "1" or "0", i.e. it is bit data. The mask data can be used to indicate validity of value data or operations on value data.

A.2.9 Map Data Segment, Map Types, Map Enable and Map Schemes

The map data segment allows a single value datum to be expanded into a vector of values. A VIFF file may contain contain *map data* without value data. Most often, the map data segment is used to provide color information, but is not limited to the storage of color vectors.

The value data serves as an index to a map vector. In the case that the color storage type is set, the result is a color vector. The range of the value data specifies the number of vectors in a map plane. The number of elements in a map vector determines the number of attributes that are provided by the map to describe the value data that accessed it. For example, when map data is used to store the RGB colors of a byte image, a pixel value of 5 will index vector (or row) 5 of the map. The map contains 256 vectors to cover the range of byte value data. At row 5, three values will be found: one each for the red, green, and blue color components of value datum 5. The resulting mapped data is then referred to as a color vector.

Associated with maps are two fields that describe dependencies of value data on the map data; these are **map enable** and **map scheme**. Two different map types are set with the **map_enable** header field; `optional` and `forced`. If the map enable field is set to optional, this indicates that the value data is valid both by itself or after indexing into the map.

If the map enable field is set to forced, the value data is NOT valid until it is used as an index into the map. When the map enable field is set to forced, the value data by itself is not meaningful; it serves only as an index into the map, where the actual information is stored.

The **map scheme** field defines how the maps are to be used with respect to the five dimensional implicit value data or the N dimensional explicit value vectors. A map is a plane of data with its width specified by the number of elements in a map vector (me) and its height specified by the range of data values (mr) that it can map. It is often desirable to have multiple map planes (mp), where the map plane chosen is a function of one of the data dimensions. If the number of map planes (mp) is greater than one (1), then the *map scheme* field is used to determine which value data dimension is used to index the plane.

A map scheme of `depth` means that the map plane index is tied to the value data volume dimension depth. A map scheme of `elements` means that the map plane index is tied to the value vector index. A time series of maps is used if the map scheme is set to `time`. A map scheme of `cycled` means that a single data value has multiple map planes. This can be used to create an animation sequence of a single data value by just changing its mapped color. The total size of the maps is give by `map vector size * map range size * map planes size` or `me*mr*mp`. If `map planes = 1`, then the map data is shared across all of the data values in the VIFF.

A.2.10 Implied Location Data

The term *implied location* of a value vector comes from the concept of using the width, height, and depth indices as the three-dimensional location of the vectors in space. For example, `data[h*width+w] = value vector`, is used to find value vectors in an image. Where w is the horizontal (row) index of the data values, and h is the vertical (column) index of the data values. The implied location of the value vector is (w,h). For clarity, depth (d), time (t), and elements (e) are ignored in this discussion.

Data that is obtained from an evenly sampled grid usually relies on the index to the data to define the location of the data, thus implicit data locations. When this method of determining a location is insufficient, explicit location data is required.

A.3 EXPLICIT LOCATION AND TIME DATA

Implicit location data is limited to three dimensions (width, height, depth), but *explicit* location data is N-dimensional. The explicit spatial location of value data is constant with time and the same for all elements of a value vector. The amount of explicit location data is given by `width * height * depth * location dimension`.

Time data is limited to one dimensions (time) for both implicit and *explicit* time data. Implicitly located data requires less space to store since the indices w, h, d, t and e along with the "delta's" can be used to compute a location, but when the data is not obtained using uniform sampling or is sparse, the explicit storage of time and location becomes more efficient.

There are two types of explicit location and time data, *forced* and *optional*. Data values that have forced explicit location data can NEVER have implied locations given via the width, height and depth; while data values that have optional explicit location data can be processed with respect to either implicit or explicit locations. For example, a sparse data set with 15 values in 3-D space are stored as `width=15, height=1, depth=1`, explicit location dimension is, `n=3`, and the location data is *forced*. As another example, an irregularly sampled data set of 15 consecutive rainfall values from a single location are stored with `time=15, width=1, height=1, depth=1`, explicit location dimension is 3, the location data is *optional*, and the time data is *forced*. The values in the time data in this case are the dates that the rainfall was measured.

Explicit location data is *shared* across a value vector and time. In other words, a value vector is a set of attributes at some location in N-dimensional space and time. Both the value vector and the location data are found using an index calculated from w, h and d. For example, consider a value vector that has a 2D spatial location. The x location value is found using the index `location[0][h*width+w];` and the y location value is found using the index `location[1][h*width+w]`. W is the width index of both the value data and location data, and h is the height index of both the value data and location data. And `[0]` and `[1]` index the location plane, where `location dimension = 2`. As in the *implicit* image, the value vector is located at `data[h*width+w]`.

A.3.1 Storage of Explicit Surfaces

It is common to drape an image over a three-dimensional surface. The discussion above on implicit data storage might lead one to think that a volume of data will need to be stored, where most of the points are empty. Instead of storing a surface as a sparse volume of data, it can be stored as an image with explicit location data. The number of locations is equal to `width *`

`height` and the location dimension (n) is three. There can be a value vector at each location for such things as color, surface normals, texture, etc.

A.3.2 Storage of Explicit Points

The explicit surface described above utilizes an implied 2D connectivity in order to calculate a perspective view of the image draped on the explicit surface. If this connectivity was not used or known, then the value vectors describe attributes of data points.

The issue of connectivity introduces the next section. So far, the explanation of the VIFF has covered both implicit and explicit location and time data, but only implicit connectivity.

A.4 CONNECTIVITY DATA AND GEOMETRY PRIMITIVES

Value vectors with either implied or explicit locations can have optional connectivity information. Connectivity is used to specify how sparse value vectors are related in space to form a geometry primitive. A VIFF can only store a single type of connectivity or a single kind of primitive, but can store multiple unrelated primitives. The type and number of primitives is stored in the header fields **connect type** and **primitive number**, respectively.

For each primitive, there is a list of integers, with the first integer typically being the number of indices to follow for that primitive. An index is a single number that points to either or both a value data vector and/or a location data vector. The index is calculated from w, h, and d, and a list of indices describes a connected primitive. The primary goal of connectivity data is to provide for efficient storage of **geometry primitives** such as polygons, contours, polytriangles, polylines, meshes, polyhedra, directed points, etc. At this time only a few fundamental primitives have defined storage methods. Readers that are familiar with geometry visualization will realize that not all of the information necessary to render or display a VIFF stored primitive is in the VIFF. The VIFF depends on "scene", "renderer", or "group" files for the rest of the geometry information. VIFF is intended for storage of "raw" geometry data, nothing more.

A.4.1 Polytriangle Primitive Storage

The polytriangle is one of the most important geometry surface primitives and it is also one of the easier primitives to explain. The term vertex is used to mean the intersection of two or more lines.

There are two types of polytriangles, connected polytriangle strip and disjoint polytriangle strip. A **connect type** of `polytriangle disjoint` refers to a set or strip of disjoint triangles. A single primitive, in this case, is a strip of disjoint triangles, the number of strips is stored in the header field **primitive number**. The first integer in the connectivity list for each primitive is the number of vertices that follow, the integer should be a factor of three.

Each vertex has an explicit location and a value vector that is found using the vertex as an index. The attributes or features or characteristics (color, texture, normals, etc.) of each vertex are stored either as a value vector, or as a value datum that points to a row of map data. A set of twenty disjoint polytriangle strips in 3D cartesian space are stored using `width=number of vertices=20*3, height=1, depth=1, location dimension=3`, and `number of primitives = 1`. If each vertex has a 3D normal and an RGB color, then `elements=6`. The value data storage type and the location data storage will usually be float.

A **connect type** of `polytriangle connected` refers to a set or strip of triangles with implied connectivity. A single primitive is a strip of connected triangles, the number of strips is stored in the header field **primitive number**. The first integer in the connectivity list for each primitive is the number of vertices that the strip has, the integer should be a factor of three minus two (number of triangles=number of vertices-2). For this primitive, there is NO list of vertices since the connectivity is implied by the order in which the value vectors and location data are stored. The **edges** of the triangles are implied by the order of the vertices.

A.4.2 Polyline Primitive Storage

A polyline is similar to a polytriangle except for the fact that it really can't describe a surface since it has no normal or thickness. For example, polylines are used to store 2D and 3D contours. There are two types of polylines, connected polyline strip and disjoint polyline strip.

A **connect type** of `polyline disjoint` refers to a set or strip of lines. A single primitive, in this case, is a strip of disjoint lines, the number of strips is stored in the header field **primitive number**. The first integer in the connectivity list for each primitive is the number of vertices that follow, the integer should be a factor of two.

Each vertex has an explicit location and an optional value vector that is found using the vertex as an index. A set of twenty disjoint polyline strips in 3D cartesian space are stored using `width=number of vertices=20*2, height=1, depth=1, location dimension=3`, and `number of primitives = 1`. If each vertex has an RGB color, then `elements=3`.

A **connect type** of `polyline connected` refers to a set or strip of lines implied connectivity. A single primitive is a strip of connected lines, the number of strips is stored in the header field **primitive number**. The first integer in the connectivity list for each primitive is the number of vertices that the strip has, the integer should be a factor of two minus one (number of lines=number of vertices-1). For this primitive, there is NO list of vertices since the connectivity is implied by the order in which the value vectors and location data are stored.

A.4.3 Mesh Primitive Storage

A implicitly connected 2D Mesh is stored in the same way that the explicit location data surface storage was described earlier. A **connect type** of `mesh` refers to a 2D plane that defines a surface. Only a single mesh primitive can be stored in a VIFF; the header field **primitive number** must be 1, but the **connect size** is zero. The size of the mesh is determined by width and height, while depth must be 1.

If each implicitly connected vertex has a 3D normal and an RGB color, then `elements=6`. The value data storage type and the location data storage will usually be float.

A.4.4 General Comments on Geometry Primitive Storage

There are several other geometry primitives that are desirable to store; such as polyhedra, spheres, and directed points. Polyhedra storage is similar to disjoint polytriangle storage, spheres and directed points are like meshes in that they do not require a connectivity list.

It is useful to think of volume primitives as well as the surface primitives discussed above. A uniformly sampled volume primitive with implicit location data was discussed in the section on volume value data. Rectilinear and curvilinear grid volumes can also be stored. The location data can be implicit if it is calculated from a function stored in the tag data or it may be explicit.

If the location data is explicit, the connectivity can be implied by w, h, and d, or it can be explicitly stored in the connectivity list. In this later case, each point is a primitive and has a connectivity list.

A.4.5 More on Time Series of Value Data

The concept that VIFF contains a series of volumes of value vectors, needs to be revisited in the context of explicit connectivity data. Every value vector and corresponding location vector is found by using a function of width, height, and depth to calculate the index. A list of these indices is stored in the connectivity data. Therefore, a primitive stored in VIFF can have changing attributes at a fixed place in space and time.

A.5 KHOROS DATA PROCESSING CONVENTIONS

The flexibility of the data representations provided by VIFF can cause programming in Khoros to be incredibly complex unless assumptions are made with respect to processing of images, signals, and volumes. To reiterate, this is where the data service functions of Khoros come in. The data service functions provide a limited, simple, and abstract interface to VIFF.

Consult the programming manual for details on how signals and images are handled using the data service functions.

A.6 VIFF POINTERS

When the VIFF data is in memory, pointers that are part of the VIFF structure are used to address the data. The pointers below are declared as char pointers of various dimensionalities, the functions that access the data segments cast the pointer to the desired type, for example `double *`. A brief description of each of the pointers in the VIFF structure is provided below:

char *tags — On disk (or data transport), the tag data is stored as a sequence of identifiers and corresponding values and strings. The format for a single tag is
`number of bytes: identifier: parsing function: parsing information: value(s)`
The number of bytes per tag is specified as an unsigned long integer, and the identifier is an ASCII string that "names" the data that follows, the name must be in a #define that is included with the application code being written. Khoros supports a few tag names that can be found in `viff_tags.h`. The parsing function string is used to determine the which supported parsing function to call (i.e. kscanf()). The parsing information specifies how the value(s) are to be read (i.e. "%g %g"). The values can be ASCII or raw data. (The data services provide a machine independent method of reading and writing tag data).

char *maps — Points to a series of two-dimensional maps; a map is organized as an array of rows.

char *location — The location pointer uses the w, h, and d indices of the value data to point to a single location vector. In other words, the location pointer points to a 3D array coordinate vectors. If there are two

	dimensional explicit locations (location dimension=2), then for each w, h, and d combination, there is a plane of x's followed by a plane of y's.
char *time	Points to a 1D array of time values.
unsigned long **connect	Points to an array of lists, where each list corresponds to a single primitive.
char *mask	Points to bit data that is organized exactly like the value data.
char *values	Points to an array of planes or an array of lines depending on the data storage orientation. The VIFF header also supports detached data and data shared across multiple VIFF files. This is done by providing an ASCII name of where the data actually resides. The following pointers are in the VIFF structure.
char *tags_name	
char *maps_name	
char *location_name	
char *time_name	
char *connect_name	
char *mask_name	
char *values_name	

If the name pointer is empty, a single \n (new line), then the data is in the current VIFF. If the name pointer contains a valid string terminated with a \n, then that string is used to find the actual storage location of the data.

B. VIFF HEADER FIELDS

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
char identifier;	A one byte magic number that identifies a VIFF file.	VFF_MAGIC
char file_type;	A one byte code indicating the VIFF file type (currently only one type).	VFF_TYPE
char release_major;	A one byte code indicating the specific major release of the viff.h file (currently 4); this does not have to agree with the Khoros system release number.	VFF_RELEASE_MAJOR
char release_minor;	A one byte code indicating the specific minor release of the viff.h file (currently 0); this does not have to agree with the Khoros system version number. For example, major release 4, minor release 0, is referred to as VIFF 4.0.	VFF_RELEASE_MINOR
char machine_id;	A one byte code indicating how the particular architecture that the data was last processed on treats data. The identifier indicates both byte ordering and floating point format. Most architectures are supported via these defines.	VFF_DEP_APOLLO VFF_DEP_DECSTATION VFF_DEP_NS32000 VFF_DEP_IBMRT VFF_DEP_IBM032 VFF_DEP_MC680x0 VFF_DEP_MC68k32 VFF_DEP_SONY VFF_DEP_SUN VFF_DEP_TITAN VFF_DEP_VAX VFF_DEP_MIPSEB VFF_DEP_MIPS VFF_DEP_SGI VFF_DEP_NEXT VFF_DEP_HP9000 VFF_DEP_RISC6000 VFF_DEP_CRAY VFF_DEP_CONVEX VFF_DEP_MACII VFF_DEP_DGUX VFF_DEP_I386 VFF_DEP_M88K VFF_DEP_LUNA88K VFF_DEP_UNKNOWN

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
char orientation;	A one byte code indicating how the 5D value data is stored. Even though the data is described as 5D data, it is actually stored as 3D data. The functions that read the VIFF value data segment return a 5D pointer, but read either consecutive planes. The PLANE orientation is indexed using [w][h][d][t][e] after it is read, but stored using [w][h][e * time * depth + t * depth + d]. A simple image has elements=1 (index starting at zero), depth=1, and time=1, thus only a single plane would be stored using [w][h][1]. A sequence of sequence vectors (5D) has d*t*e planes of size w*h; first all d planes are stored, followed by t planes, followed by e planes. The VECTOR orientation is indexed using [e][w][h][d][t] after it is read, but stored using [w * elements + e, h, t * depth + d]. A single true RGB image would have elements=3 and depth=1; giving a VECTOR index of [w*3+e,h,1] or a PLANE index of [w,h,e].	VFF_ORI_VECTOR VFF_ORI_PLANE
unsigned long data_encode_scheme;	This unsigned long field contains the information that specifies the encoding or compression method used for storage of all of the data segments.	VFF_DES_RAW VFF_DES_COMPRESS VFF_DES_RLE VFF_DES_TRANSFORM VFF_DES_CCITT VFF_DES_JPEG VFF_DES_MPEG VFF_DES_ADPCM VFF_DES_GENERIC

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
unsigned long width, height, depth, elements, time;	These five unsigned long fields specify the size of value data, mask data, and location data. Any of these three data segments can be empty. Width specifies the size of a row or the number of columns; w is the horizontal index. Height specifies the size of a column or the number of rows; h is the vertical index. Depth specifies the number of height by width planes; d is the index from front to back. Elements specifies the number of data values at a single implicit or explicit location; e is the index of a value vector. Time specifies the number of height by width by depth implicit or explicit data sets; t is the index. The minimum valid value for a size is 1, if the any dimensions is equal to 0, then there is NO value, mask or location data. A VIFF with no data value is valid. The product of all of the dimensions is the total amount of value data. The product of width, height, and depth, equal the number of location vectors.	none

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
unsigned long delta_width, delta_height, delta_depth, delta_elements, delta_time;	This unsigned long field specifies the data value width, height, depth, time and elements, physical sizes or sampling interval. This information is needed to do true measurements and calculate true frequencies. The ratio of width and height deltas will give you the aspect ratio of a digitized pixel. Most CCD camera's are not one-to-one. These values may not have a meaning if the data is VFF_LOC_EXPLICIT_*.	none
char value_storage_type, map_storage_type, time_storage_type, location_storage_type;	This byte field indicates the data storage type of value, location, or map data. Currently, bit, byte, short, integer, long, float, float complex, double, and double complex data types are supported. The BIT storage type stores the bits in packed unsigned chars and pads to a byte; the order is Least Significant Bit (LSB) first.	VFF_TYP_BIT VFF_TYP_UCHAR VFF_TYP_CHAR VFF_TYP_USHORT VFF_TYP_SHORT VFF_TYP_UINT VFF_TYP_INT VFF_TYP_ULONG VFF_TYP_LONG VFF_TYP_FLOAT VFF_TYP_COMPLEX VFF_TYP_DOUBLE VFF_TYP_DCOMPLEX
char location_type;	This byte field indicates whether the value data has implicit or explicit locations. If the locations are implicit or NONE, the field location_dim must be set to zero (0), and the location data will be empty. If the locations are explicit, the field location_dim indicates the dimensionality of the space or the size of the location vector. The explicit location vector is pointed to by location, and is stored as planes of coordinates. For example, if (location_dim = 2), implying 2D locations, the location data would be stored; x1, x2, . . . , xn; y1, y2, . . . , yn. If the explicit locations are FORCED, then the value data index (w, h, d) can not be used to find determine an implied location. If the explicit locations are OPTIONAL. The location of a data value can come from either the implied location given by width, height, depth, or the explicit location found by using width, height, depth, to derive the index to a location vector. The explicit locations are shared through the elements of the value vectors and time.	VFF_LOC_NONE VFF_LOC_IMPLICIT VFF_LOC_EXPLICIT_FORCED VFF_LOC_EXPLICIT_OPTIONAL VFF_LOC_IMPLICIT_SPHERICAL VFF_LOC_IMPLICIT_CYLINDRICAL VFF_LOC_IMPLICIT_HEX_GRID VFF_LOC_IMPLICIT_RECT_GRID VFF_LOC_IMPLICIT_CURV_GRID

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
char value_type, mask_type, time_type;	These byte fields indicate whether the value, mask and time data is present. For consistency with other type fields, the terms FORCED, OPTIONAL and NONE are used.	VFF_VALUE_NONE VFF_VALUE_OPTIONAL VFF_VALUE_FORCED VFF_MASK_NONE VFF_MASK_OPTIONAL VFF_MASK_FORCED VFF_TIME_NONE VFF_TIME_IMPLICIT VFF_TIME_EXPLICIT_FORCED VFF_TIME_EXPLICIT_OPTIONAL VFF_TIME_IMPLICIT_RECT_GRID VFF_TIME_IMPLICIT_CURV_GRID
unsigned long location_dim;	This unsigned long indicates the dimensionality of the location space of data. One-, two-, and three-dimensionalities will be most common. If the location_type is implicit, this field must be set to zero (0). If the location_type is explicit, this field is set to the the size of the location vector. An explicit 2D image is represented using location data with location_dim = 2 and the image intensity values stored in the value data.	none
char connect_type;	This byte field specifies the kind of connectivity being used or the kind of primitive that is stored. In some cases, the connect type requires that connect_size be zero or that the primitive number be one.	VFF_CON_NONE VFF_CON_IMPLICIT VFF_CON_GENERIC VFF_CON_POLYTRIANGLE_CON VFF_CON_POLYTRIANGLE_DIS VFF_CON_POLYLINE_CON VFF_CON_POLYLINE_DIS VFF_CON_MESH VFF_CON_POLYHEDRA VFF_CON_SPHERE VFF_CON_POINT
unsigned long primitive_num;	This unsigned long field specifies the number of primitives in the connectivity array. The length of each primitive index list is stored as the first number in the list.	none
unsigned long connect_size;	This unsigned long field specifies the total number of bytes stored in the connectivity array.	none

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
char map_scheme;	This byte field specifies the type of mapping that should occur with respect to the width, height, depth, time and elements of the value data. The map is a sequence of planes, where the map scheme indicates which data value dimension is used as the map plane index. A volume of width by height planes, where each plane has a different map would use a map_scheme of DEPTH. A map_scheme of SHARED requires that there only be one map plane and that the map be shared across all value data. A map_scheme of CYCLE means that a single data value has more than one map and that all value data share the multiple map planes. The map scheme is only valid if map_enable is NOT VFF_MAP_NONE. The default map_scheme is VFF_MS_SHARED.	VFF_MS_SHARED VFF_MS_DEPTH VFF_MS_TIME VFF_MS_ELEMENTS VFF_MS_CYCLE
char map_enable;	This byte field indicates whether or not there is a map and if the map must be used. A map enable of NONE specifies that there will be NO map data and that the map_scheme and map size fields can be ignored. The color_space_model may still be valid. A map enable of OPTIONAL means that the application makes the choice on whether or not to use the map. A map enable of FORCE means that the value data must be mapped in order for it to have a valid meaning.	VFF_MAP_NONE VFF_MAP_OPTIONAL VFF_MAP_FORCE

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
unsigned long map_vector, map_range, map_planes;	These three unsigned long fields specify the size of the map data. A map vector is a row of a map plane with the index `me` indicating the number of elements in the vector. The map range is a column of a map plane with the index `mr` indicating the number of vectors in a map plane. The term range is used because it is the range of the value data that determines the size of the map columns. The number of maps is given by map_planes and uses `mp` as the index. `me` is analogous to width, `mr` is analogous to height, and mp is analogous to depth. The maps are stored as a sequence of rows from the front plane to the back plane. The minimum valid value for a size is 1, if any one of the dimensions is equal to 0, then there is NO map data and the map_enable field should be set to NONE. The **map_scheme** determines how the `mp` should be used. The map_planes index can be associated with the data depth, time or elements index. The `me` index allows a single data value to have a vector associated with it and the `mr` index covers the range of the data values.	none

Khoros Visualization and Image File Format Header Fields		
Field	**Description**	**Defines**
unsigned long color_space_model;	This unsigned long field indicates the color space or the coordinate system being used to interpret a value vector. In the case that the value vector is derived by mapping a data value through a map (the map_enable is NOT VFF_MAP_NONE), the map_scheme field needs to be checked to see how to apply the color space model. The color space model defines use the following convention: NTSC: National Television Systems Committee CIE: Commission Internationale de L'Eclairage UCS: Universal Chromaticity Scale RGB: Red Component, Green Component, Blue Component CMY: Cyan Component, Magenta Component, Yellow Component YIQ: Luminance, I and Q represent chrominance HSV: Hue, Saturation & Value IHS: Intensity, Hue & Saturation For example, when the elements=1, depth=1, the map_enable is set to VFF_MAP_NONE, the color_space_model is set to VFF_CM_genericRGB, the map_vector = 3, and the map_planes = 1, you are specifying a pseudo colored image. VFF_CM_GENERIC indicates that a color space is valid, but is being defined by the user. A color space of NONE means that the value vector does not fit a color space model.	VFF_CM_NONE VFF_CM_ntscRGB VFF_CM_ntscCMY VFF_CM_ntscYIQ VFF_CM_HSV VFF_CM_HLS VFF_CM_IHS VFF_CM_cieRGB VFF_CM_cieXYZ VFF_CM_cieUVW VFF_CM_cieucsUVW VFF_CM_cieucsSOW VFF_CM_cieucsLab VFF_CM_cieucsLuv VFF_CM_GENERIC VFF_CM_genericRGB VFF_CM_grey
unsigned long tag_number;	This unsigned long field specifies the number of tags. The size of each tag in bytes is the first unsigned long integer.	none
unsigned long tag_size;	This unsigned long field specifies the total number of bytes used by the tags.	none

Khoros Visualization and Image File Format Header Fields

Field	Description	Defines
char **tags;	The tag defines are used as identifiers in the tag data. The `comment` tag is used in Khoros as a comment field to document the VIFF data file. The `map_subrow_size` tag specifies the number of subrows in a map. This is useful when using the output vector from the map as a 2D image, rather than just a vector. The size of the 2D image would be: map_subrow_size (columns) by map_vector/map_subrow_size (rows). The `subrow_size` tag specifies the length of any subrows in the data. This is useful when one wants pixel vectors to represent 2D objects (images). The size of each pixel "image" would be subrow_size (columns) by elements/subrow_size (rows). The `start_w, start_h, start_d, start_e, start_s` tags indicate the location of a subimage or subvolume in a parent image. The image is a sub image if start_w and start_h have values greater than zero. start_w and start_h locate the upper left hand corner of where the image was extracted. This applies to 5D data that has implicit locations. The `grid_w, grid_h, grid_d, grid_e, grid_s` tags specify irregular grid intervals. This is an alternative to explicit location data. The `kernel_w, kernel_h` tags specify the origin of a morphology or convolution kernel.	VFF_TAG_COMMENT VFF_TAG_CREATION_DATE VFF_TAG_MAP_SUBROW_SIZE VFF_TAG_SUBROW_SIZE VFF_TAG_START_W VFF_TAG_START_H VFF_TAG_START_D VFF_TAG_START_E VFF_TAG_START_S VFF_TAG_GRID_W VFF_TAG_GRID_H VFF_TAG_GRID_D VFF_TAG_GRID_E VFF_TAG_GRID_S VFF_TAG_KERNEL_W VFF_TAG_KERNEL_H VFF_TAG_SPECTRUM_MASK VFF_TAG_SPECTRUM_NUM

Khoros Visualization and Image File Format Header Fields

Field	Description	Defines
char **tags;	Continued. The `null_value` tag specifies what value should be treated as missing data. The `000_location` tag specifies specifies the origin of implicit data and what coordinate system should be used.	VFF_TAG_NULL_VALUE VFF_TAG_000_LOCATION

C. EXAMPLES USING THE VIFF FORMAT

The table above succinctly describes each field and its meaning, but it is helpful to have an example to follow to reinforce the ideas. The following narrative describes the use of the most difficult fields (location_type, location_dim. num_data_bands, map_scheme, map_vector, map_range, map_enable, and color_space_model) by tracking an image through a typical Khoros image processing application.

You will probably begin your image processing task by acquiring an image using an eight bit digital to analog converter. When this image is first brought into the Khoros, it would usually be declared as:

Khoros Image File Format Header Fields

Field	**Defines**	**Description**
machine_id	VFF_DEP_?	assigned by **raw2viff**
data_encode_scheme	VFF_DES_RAW	no compressing
data_storage_type	VFF_TYP_UCHAR	8 bit data
width	w	number of columns
height	h	number of rows
depth	1	a one band image or a single value pixel or a 2D data set
location_type	VFF_LOC_IMPLICIT	a normal digitized image
map_scheme	VFF_MS_SHARED	default
map_enable	VFF_MAP_NONE	don't have a map
color_space_model	not applicable	map_enable is NONE

Next, you probably will process this image with several of the Khoros programs. As you do this, you are not likely to change the above fields but may change the data_storage_type field from VFF_TYP_UCHAR to some other type.

As you view the resulting image with editimage, you may want to *enhance* the image. This will change the fields so that the data is interpreted correctly. For instance, if you add "color" to your grey image and save the result, the header fields should be:

Khoros Image File Format Header Fields		
Field	**Defines**	**Description**
data_storage_type	VFF_TYP_UCHAR	8 bit data
depth	1	a 2D data set
map_scheme	VFF_MS_SHARED	the value data shares the single plane (3x256) map
map_vector	3	map each data value through three columns
map_range	256	256 rows in the map
map_storage_type	VFF_TYP_UCHAR	8 bit map result
map_enable	VFF_MAP_OPTIONAL	optionally may use map
color_space_model	VFF_CM_genericRGB	pseudo colored

The displayed image may be enhanced even further by adding overlays to the image. The displayed overlay and image are saved by executing **getimage** (kman getimage). getimage actually does a screen dump into a VIFF file. If the screen is an eight plane display, then the resulting fields are:

Khoros Image File Format Header Fields		
Field	**Defines**	**Description**
data_storage_type	VFF_TYP_1_BYTE	8 bit data
depth	1	a 2D data set
map_scheme	VFF_MS_SHARED	the value data shares the single plane (3x256) map
map_vector	3	map data values through three columns
map_range	256	256 rows in the map
map_storage_type	VFF_TYP_UCHAR	8 bit map result
map_enable	VFF_MAP_FORCE	must be mapped!
color_space_model	VFF_CM_genericRGB	RGB interpretation

This means that the data value only has meaning if it is mapped through the three color map columns.

Let's return to the image as it was when it was first brought into the Khoros and look at vector data representations. You may use a Khoros routine that extracts features of an image at sparse, explicit locations. This is best stored as vector data.

Khoros Image File Format Header Fields		
Field	**Defines**	**Description**
location_type	VFF_LOC_EXPLICIT_FORCED	explicit locations
location_dim	2	x and y locations
data_storage_type	VFF_TYP_FLOAT	real numbers
depth	1	a 2D data set
elements	7	each value vector has 7 data values
map_enable	VFF_MAP_NONE	no map

In the case that you want to have an RGB scanner and have digitized 8 bits each for red, green and blue, then the fields should be specified as follows:

Khoros Image File Format Header Fields		
Field	**Defines**	**Description**
machine_id	VFF_DEP_?	assigned by **raw2viff**
data_encode_scheme	VFF_DES_RAW	no compressing
data_storage_type	VFF_TYP_UCHAR	8 bit data
depth	1	a 2D data set
elements	3	three data values for each color vector
location_type	VFF_LOC_IMPLICIT	a normal digitized image
map_vector	1	one data value maps to one value
map_range	255	8 bit map
map_planes	3	one plane per color band
map_scheme	VFF_MS_ELEMENTS	elements indexes map planes
map_enable	VFF_MAP_OPTIONAL	don't have to map
color_space_model	VFF_CM_ntscRGB	red, green, blue

CONTENTS

Chapter 10 - VISUALIZATION and IMAGE FILE FORMAT (VIFF)

A. VISUALIZATION and IMAGE FILE FORMAT (VIFF) 1
A.1 VIFF ORGANIZATION 1
A.2 VALUE DATA TERMINOLOGY and DESCRIPTION 5
A.2.1 Points, Lines, Planes, Volumes and Sequences 5
A.2.2 Storage of Signals (1D) 5
A.2.3 Storage of Images (2D) 6
A.2.4 Storage of Volumes (3D) 7
A.2.5 Storage of Sequences (4D) 7
A.2.6 The Value Vector as a General Data Structure 8
A.2.7 Storage Orientation 8
A.2.8 Mask Data Segment 9
A.2.9 Map Data Segment, Map Types, Map Enable and Map Schemes 9
A.2.10 Implied Location Data 10
A.3 EXPLICIT LOCATION AND TIME DATA 10
A.3.1 Storage of Explicit Surfaces 10
A.3.2 Storage of Explicit Points 11
A.4 CONNECTIVITY DATA AND GEOMETRY PRIMITIVES 11
A.4.1 Polytriangle Primitive Storage 11
A.4.2 Polyline Primitive Storage 12
A.4.3 Mesh Primitive Storage 12
A.4.4 General Comments on Geometry Primitive Storage 12
A.4.5 More on Time Series of Value Data 13
A.5 KHOROS DATA PROCESSING CONVENTIONS 13
A.6 VIFF POINTERS 13
B. VIFF HEADER FIELDS 15
C. EXAMPLES USING THE VIFF FORMAT 25