

# Proposal For Integrating the DRI and VSIPL Specifications

Jamie Kenny, Ken Cain

## I. Overview/Purpose

This document is a preliminary proposal for integrating the VSIPL (Vector Signal and Image Processing Library) with the DRI (Data-Reorg Interface) specifications. While it is clear that the two interfaces can be used together in the same application, a more integrated solution is desired that will eliminate most, if not all, of the VSIPL related setup and initialization.

## II. Requirements

The main requirement is to allow the library vendor to reduce application source lines of code by having the DRI library perform most of the VSIPL related setup. Providing canned example code is not the idea, since that can be done now, without any consideration on the part of the DRI and/or VSIPL Forums.

The implementation must not force the application to separately reference any VSIPL header files or libraries if the use of VSIPL is not desired in the application. In other words, applications that don't use VSIPL shouldn't need to include vsip.h or link in a VSIPL library.

## III. Functional Definitions

Function `DRI_Reorg_setup_vsip()` creates the requested VSIPL views for the specified reorg.

```
DRI_Reorg_setup_vsip( DRI_reorg reorg,  
                     long nviews,  
                     long *ndims_list )
```

```
reorg      - previously created reorg object.  
nviews     - number of views to be created. There will  
            be nviews VSIPL view object created for  
            each buffer (in the bufferset).  
ndims_list - [nviews] is a list that contains the  
            number of dimensions (ndims) for each view  
            created.
```

The dimensionality of the VSIPL view can not be larger than the dimensionality of the GDO, but it can be smaller. Each view will be applied to those global data dimensions that are most contiguously arranged in local memory. If the requested view calls for fewer data dimensions than there are global data dimensions, then the dimensions with the most contiguous layout specifications will be affiliated with the view.

Also, all views will map to the first data in the local buffer and could potentially overlap each other. If a view is to be applied to all of the data dimensions, then the view will be able to operate over every local data element. If the view is to be applied to fewer dimensions than there are global data dimensions, the view will operate over the first instance of a sub-dimensional structure. For a 2-D VSIPL view of a 3-dimensional dataset, then the view will be initialized to operate over the first matrix stored in the local buffer.

Example: Make a vector view and matrix view for a DRI 3-dimensional data set.

```
long ndims_list[3];
ndims_list[0] = 1;
ndims_list[1] = 2;
ndims_list[2] = 1;

DRI_Reorg_setup_vsip1( reorg, 3, ndims_list );
```

Assume the data is distributed on only 1 processor, packed, and the layout order is defined by array long layout[3].

The first view will be a vector view with

```
stride = 1
length = global_dims[layout[0]].
/* global_dims passed to DRI_Global_Data_create */
```

The second view will be a matrix view with

```
row_stride = 1
row_length = global_dims[layout[0]]
col_stride = row_stride * row_length
col_length = global_dims[layout[1]]
```

The third view will be a vector view with

```
stride = 1
length = global_dims[layout[0]].
```

All the views will start out "pointing" to the same address. The purpose of allowing multiple views is to allow the DRI library to create them during setup. The user can then efficiently change the view attributes to move the view around the GDO.

The reorg object knows the data type, so it can pick the correct VSIPL view data type.

-----

Function DRI\_Buffer\_get\_vsip1\_views( ) returns the VSIPL views for that buffer.

```
DRI_Buffer_get_vsip1_views( DRI_Buffer buf,
                           void **views,
                           long **ndims_list,
                           long *nviews );
```

```
buf          - a buffer object returned from DRI_Reorg_get_buffer()
vviews       - array of VSIPL views.
ndims_list   - [nviews] is a list that contains the
               number of dimensions(ndims) for each view
               created.
nviews       - number of views affiliated with buf. There will
               be nviews VSIPL view objects created for each buffer (in the
bufferaset).
```

ndims\_list and nviews are the same as what was passed to DRI\_Reorg\_setup\_vsip1().

Given the previous example for DRI\_Reorg\_setup\_vsip1(), you may have code like:

```
vsip_vview_f *view1;
vsip_mview_f *view2;
vsip_vview_f *view3;
```

```

void *views;
long *ndims_list;
long nviews;

/* assume a call to DRI_Reorg_get_buffer() here */

DRI_Buffer_get_vsip_views( recv_buffer, &views,
                           &ndims_list, &nviews );

view1 = (vsip_vview_f *)views;
view2 = (vsip_mview_f *) (view1+1);
view3 = (vsip_vview_f *) (view2+1);

```

Code to check the dimensionality could be added, but the order is always the same as specified in `DRI_Reorg_setup_vsip()`.

-----

During runtime, the following VSIPL operations are implied by the corresponding DRI operations.

DRI operation/function	Implied VSIPL operation/function
DRI_Reorg_setup_vsip()	vsip_init() vsip_[c]block_bind/create_*( vsip_[c][m/v]view_bind_*(
DRI_Reorg_get_buffer()	vsip_[c]block_release_*(
DRI_Reorg_put_buffer()	vsip_[c]block_admit_*(

All VSIPL objects and resources will be destroyed or released when the reorg object is destroyed.

## IV. Design Considerations

One possible implementation is to have a table of VSIPL function pointers in the library. The table would initially be filled with NULL pointers and populated with legal function pointers when `DRI_Reorg_setup_vsip()` is invoked by the application.

In the case where the application doesn't call `DRI_Reorg_setup_vsip()`, there will be no VSIPL references and the application source code doesn't have to include `vsip.h`.

When the application does call `DRI_Reorg_setup_vsip()` and in the inner loop `DRI_Buffer_get_vsip_views()`, a VSIPL library will be required to resolve references and the application source code will need to include `vsip.h`.