VSIPL, from API to Product

Sharon M. Sacco
VSIPL Forum designed a library
- The API is well defined
- The mathematics are understood

VSIPL Forum tried to minimize advice to implementers
- No standards for performance are included in the API
- There are no requirements for accuracy in the API
  - Test Suite Lite does have accuracy parameters, but they can be adjusted
- Advice to implementers is limited to explaining incomplete type definitions and checking for object validity in development mode
Defining VSIPL Success

- **Purpose of VSIPL is reusability**
  - Applications need to be written with VSIPL
  - VSIPL needs to be available for the next platform those applications use

- **Reusability requires both users and products**
  - Users avoid products that are hard to use
  - Vendors avoid development on products that users do not use

- **Success is having both VSIPL users and products**
A Successful VSIPL Product

- **VSIPL products can have varying levels of implementation**
  - Minimal requirement is a Core Lite implementation with three data types
  - A more functional implementation is a Core or Core Plus implementation

- **A successful VSIPL product should be more than just a library**
  - Most steps that contribute to a successful product are not difficult to add to a basic library
The VSIPL API document is designed to define the API

- Although the examples are numerous, it was not designed to be a learning tool
- For a product, implementers need to provide documentation beyond the API
- Users would benefit from tutorial documentation to lead them through the program building process
  - Tutorials should help organize the ideas of VSIPL for the users
  - Tutorial examples running from simple examples to complex provide programming templates for users
Users need to be trained to use VSIPL

- VSIPL’s object based design requires a different programming pattern than traditional FPS based libraries
- Classes allow users to get answers as the material is learned
- Users become proficient VSIPL programmers faster with training, requiring fewer customer support calls
Implementers need to provide tools to improve the user’s development process. Debugging tools are required.

- Development mode:
  - This is the original idea from the VSIPL Forum
  - It checks for errors described in the API document such as functions running out of array bounds
  - Development mode is not intended to replace a source level debugging tool
Debugging Environment (cont.)

- Customized debugging tools:
  - Debuggers can be customized to be sensitive to VSIPL so that the data can be as easily accessed as native C structures
  - Advantage to customization is that the user works in a natural debugging environment
  - Disadvantage is that this is difficult to implement
Debugging Environment (cont.)

− Data mining library
  − Data mining libraries allow users to examine VSIPL data during the debug process
  − Data mining libraries have two styles:
    − TASP VSIPL library contains functions to write data to files. These require modification of the user code.
    − Libraries can be designed to be callable from the command line of a debugger. This style requires no modification of the user’s code. The debugging style is interactive.
  − Data mining libraries are easy to implement
Performance

- Performance difference between FPS based libraries and VSIPL is fixed overhead, not a percentage
  - Libraries need to minimize overhead for success
- TASP VSIPL is a good start, but not the final product
  - It is designed to give answers, not great performance
  - A lot of the overhead is in the calculational functions
    - Moving address calculations and other set up code to the support functions will reduce overhead
Library portability does not require inter-library compatibility

- Users should not assume that VSIPL functionality can be added by mixing libraries
- Library designers should tune performance for the hardware and software environment of the platform