VSIPL, from API to Product

Sharon M. Sacco
SKY Computers

The API for VSIPL is complete, and a number of implementations are available. The number of users is growing. To make VSIPL truly successful implementers need to do more than write a library.

User acceptance of VSIPL requires education and support. VSIPL's object based approach requires more effort than the older FPS based libraries. While the VSIPL API document is rich with examples, new users will benefit from additional documentation that focuses on their needs. Illustrative tutorial examples can quickly communicate the concepts of VSIPL programming, accelerating the user's productivity. Implementers that communicate with their users can also document the most common barriers to good VSIPL code. Good documentation reduces the user's code development frustrations in this new environment which should reduce repetitive user inquiries.

The original idea to help users debug their applications was to provide an option known as development mode. The purpose of development mode is to allow the user to detect errors such as memory collisions, mismatched vector sizes, unintialized objects, and some memory leaks during application development. While this has not been widely adopted, it is described by the API for future inclusion. The details of development mode, how it might be implemented is an open issue.

When possible, VSIPL implementations will benefit from integration with standard tools in a programming environment. Where compiler technology allows optimization based on custom libraries, this ability can be expanded to encompass VSIPL. Sensitizing a compiler to VSIPL is also the first step to allowing debugging of VSIPL code within the normal debugger environment. Debuggers can be adapted to allow users to examine data contained in blocks and views as easily as it is to examine arrays. This level of effort does come at a cost to implementers, either through work or contracting of the work through tools vendors.

When budgets are tight or there is no control over the tools, another solution to allowing good debugging practices is to provide the user with a data extraction library. Since VSIPL uses opaque data types, users cannot use "printf" with any confidence. A data extraction library allows users to examine intermediate results during the course of a VSIPL program for error detection. While this is not an elegant solution, it is effective and within the abilities of any VSIPL implementer.

Finally, one of the most important aspects of a quality VSIPL implementation is performance. Users of the FPS based libraries require performance, and VSIPL must

deliver similar performance if it is to survive. Typically, an implementer is asked what percentage of performance is lost with VSIPL. This is the wrong way to approach VSIPL performance since typically it is an overhead cost over an FPS based library.

The purpose of this talk will be to examine these issues, particularly from handling new VSIPL users, as well as presenting performance comparisons between VSIPL and an FPS based library.