"A comparative study on Modular Visualization Environment"
CRS4 Technical Report

E. Gobbetti, A. O. Leone

March 1997

1 Introduction to MVE

Modular Visualization Environment (MVE) systems constitute a particular class of Visualization Packages. They are not visualization programs, but rather environments to build visualization applications, accordingly to specific needs of data representation. It is now clear that MVE are forced to belong to the class of "general purpose" Visualization Packages, because this is the main property that characterize them.

From the end-user point of view, MVE are extemely versatile and flexible. In many cases, setting-up a personalized representation of data, even complex, is so easy as to interconnect in a network pre-existing "modules" with atomic functionality, in order to create the specific visualization pipeline that ends with the rendering of the data. The longer is the list of available "modules", the bigger is the number of different "visualization applications" the user can build in the MVE.

2 Packages under evaluation

The following MVEs will be reviewed in this document:

Application Visualization System (AVS)

By Advanced Visual Systems Inc., supported in Italy by AVS/UNIRAS Srl. (http://www.zeropiu.it/avs/), currently available version 5.3

Data Explorer

By IBM Inc., currently available version: 3.1

IRIS Explorer

By NAG Ltd., supported in Italy by Lasertec Srl.(http://www.itsyn.it/lasertec), La Spezia, currently available version: 3.0 (3.5 just released in Italy)

They will be examined by functionality to facilitate the comparison between them.

3 Data types

Here a comparison in data types supported by the examined MVE will be given.

When data are read into the MVE, they are stored in structures to allow data management, manipulation and transmission between MVE modules.

What we call "field" in this section is, from a computational point of view, an N dimensional array of M dimensional vector data, along with associate coordinate data, from a physical point of view, a scalar/vector/tensorial field sampled over a discrete structured grid.

What we call "geometric" data type is a type that allows users to describe 3D objects and scenes by shapes and properties (color, material, lights, ...), typically used to define physical structures. In the visualization pipeline, this is generally the kind of data that feeds only the MVE render module for the final display of the input data.

What we call "unstructured" data type is essentially a geometric type with values defined on it. This kind of data type is typically used for Finite Element analysis and Computational Fluid Dynamics applications.

3.1 AVS

AVS has two main types of data: *primitive data* and *aggregate data*. Primitive data includes simple scalar types. They are the basic building blocks of aggregate data and are also used to represent parameters. Aggregate data are used to represent the major data types for scientific visualization.

Scalar types Defined in AVS as *primitive data*, they includes: byte (an 8-bit value), integer (32 or 64 bits, depending on machine architecture), single and double-precision floating point (one and two machine words long) and text strings.

Field types Field data types are fully implemented in AVS. The physical location of each data element of the array (if it exist) is either implied or specified explicitly with 3D coordinates. Accordingly to the specification of computational-to-physical space mapping, AVS field data types are classified in three basic categories: *uniform*, *rectilinear* and *irregular*.

Geometric types They are implemented via the *AVS Geometric data type*. It consists of three-dimensional objects constructed out of one or more of the available primitives: polyhedra, polygons, meshes, spheres and polytriangles. There is also support, by the *AVS edit list*, for the definition of light sources, texture mapping and control over the camera parameters to perform clipping, depth cueing and perspective viewing of scenes.

Unstructured Types This data type is implemented by the *AVS unstructured cell data (UCD)*. It consists of a geometric model built of individual cells. The cells are defined by nodes or vertices. Data (scalar or vector) can be assigned to the entire model, each individual cell and each vertex of each cell. The UCD data type provides a way to aggregate 3D primitive objects and associate data into a single data structure. UCDs are useful to represent volume information that is not structured enough to be represented as a field data type.

User defined types AVS provides limited capabilities for user to implement their own data types.

Other types Colormaps are defined as instances of the *AVS colormap data type*, a specific type to manage this kind of data structures. Futhermore, an *AVS Molecule Data Type (MDT)* has been added to AVS data types to address the general needs of the chemistry field.

3.2 Data Explorer

In Data Explorer, all data are stored in the form of *objects* for use by modules. An object is a data structure that contains an indication of the object's type, along with additional type-dependent information. The bulk of the data is encapsulated in *array* objects.

Scalar types Primitive data types in Data Explorer are: signed and unsigned byte, signed and unsigned short, signed and unsigned integer and single and double precision floating-point.

Field types Field types are implemented using *DX field and array objects.*

Field objects are the fundamendal objects in the Data Explorer. They are represented by some number of named *components* with a *value* (that is an object) associated to each of them. For example, the "position" component is an array object specifying the set of sampled points; the "connection" component is an array object specifying a means to interpolate between positions; and the "data" component is an array object specifying the data values.

Different fields can share components. This allows, for example, several fields to share the same positions and connections while having different data.

The grid of a field is defined by the "position" component. The array object that hold this information can be *compact* (that is regular), *irregular* or a combination of them (via the *product array*), specifying all kind of structured grids.

Geometric types Geometric types are implemented using *DX group objects.*

A group is a collection of *members* that themselves may be field objects or other group objects. Geometry shapes are defined in field members by the "position" and "connection" components. There are also other Data Explorer data types to construct geometric model for rendering, such as: *transform, clipped, light* and *camera objects.*

Unstructured Types Unstructured types are implemented via the *field objects* by using *irregular* or *mesh array* for the "position" component and specifying the connection between position with the "connection" component to build *n-dimensional simplexes* (triangles, tethraedra, tethraedra 4D, ...) or *n-dimensional cuboids* (lines, quads, cubes, cubes 4D, ...).

Data can be defined on the sampled grid points or on the connections between the sample points (cell-centered data).

User defined types No support is given in Data Explorer for user defined data types.

Other types Is it possible to represent a field as a collection of separate fields, each with its own grid, using the *DX multigrid group.* This is the case in some kind of multigrid or multidomain simulations.

It is also possible to define time series data using the *DX series group* to collect field objects each with a time stamp associated.

3.3 IRIS Explorer

Scalar types They can be byte, short, long integer and single or double precision floating point number.

Field types IRIS Explorer implements field data type with the *IE lattice data type.*

The IRIS Explorer lattice data structure has two parts. One holds data values and the other holds node coordinates. The data and coordinate arrays are optional, giving the ability to create a lattice with an empty data structure, and node coordinates only, or one with data values and no coordinate values.

Data can be stored in one of the IRIS Explorer primitive types.

Coordinates are always stored in single-precision floating point (float) format. The lattice data type allows for three types of coordinate mapping to physical space: uniform, perimeter, and curvilinear. The interleaving of the coordinate storage varies from type to type.

Geometric types The *IE geometry data type* enables handling of geometry objects by providing a container for Open Inventor scene graph specifications. Open Inventor is an object-oriented graphics library of objects and methods used to create interactive 3-D graphics. The IRIS Explorer Render module and other geometry modules draw on Open Inventor technology. The geometry objects that IRIS Explorer uses to visualize numerical data are created by connecting nodes to form an Open Inventor scene graph.

Unstructured Types The *IE pyramid data type* holds two kinds of data: irregular or unstructured grid data, such as finite element data, and molecular modeling data. In IRIS Explorer, the pyramid data type is used primarily for finite element modelling and creating irregular grids.

The pyramid data structure defines the relationship between the different layers of data required to build a pyramidal structure. In finite element data, you can consider the element grid as a collection of vertices (points), edges (lines), faces (polygons), elements (three-dimensional cells), objects (collections of 3-D elements), assemblies (collections of objects), and so on. You can fit these together to make an object from faces, bounded by edges, which are in turn delimited by vertices.

A pyramid consists of three main parts:

- the several layers of pyramidal data; for example, points, lines, and faces. These values are collected in IE Lattices.
- the relationships between these layers.
- optional references to predefined pyramid elements, which are stored in a dictionary.

User defined types The *IE user-defined data type (UDT)*, is designed specifically to define new data types in case the existing IE data types do not adequately describe the data the user wants to handle and visualize.

IRIS Explorer provides a data typing language (ETL) for defining data types that can be passed among modules. Using ETL, it is possible to create new data types for use in custom-built IRIS Explorer modules. The built-in IE data types (Lattice, Parameter, Pyramid, Geometry and Pick) were created using ETL.

Other types The *IE Parameter data type* holds a single scalar value. Its purpose is to pass scalar values between modules in one of three forms: long integer, double precision floating point or character string.

3.4 Summary

See table 1 for the summary of MVE data types.

4 Data import/export

Each of the three MVEs is provided with a set of modules for data import and export in a variety of data format. Often other reader and writer modules are provided from MVE users as public domain software and collected on world wide ftp sites. Sometimes such modules can be obtained from a commercial supplier. Anyway, when the data format specifications are well known, is it always possible to write a module reader on your own.

There are also tools provided by all MVEs that allow end-user to import and export arrays of data using a graphical interface. They are: ADIA for AVS, Data Prompter for DX and DataScribe for IE.

type	AVS	Data Explorer	IRIS Explorer
scalar	primitive type	primitive type	primitive type
field	uniform, rectilinear, irregular fields	field objects	uniform, rectangular and curvilinear lattices
geometric	Geometry	group objects	Geometry (by Open Inventor)
unstructured	Untructured Cell Data (UCD)	irregular fields	Pyramid
user defined	yes	no	yes
other	colormap, Molecule Data Type (MDT)	multi domain, time series	Parameter, Pick Data Type

Table 1: MVE data types

5 Supported visualization techniques

All three MVEs support main visualization techniques both for 2D and 3D data. Here is a non comprehensive list of such available techniques:

- Image Processing
- Isosurfaces and Slice Planes
- Streamlines and Particle Advection
- Volume Rendering
- Finite Element Data Visualization

6 Supported platforms

6.1 AVS

AVS is currently supported on DEC, HP, IBM, SGI, and SUN UNIX platform.

6.2 IE

IRIS Explorer is currently supported on Digital, HP, IBM, SGI, SUN workstations, Windows NT PC and Cray Y-MP supercomputers.

6.3 DX

Data Explorer is currently supported on IBM, SUN, HP, SGI, DG and DEC workstations and on IBM SP super-computer machines.

7 Module writing

Each MVE provide tools for constructing and installing your own custom-built modules.

7.1 AVS

There are two kinds of AVS modules: *subroutine* modules and *coroutine* modules. A subroutine module's computation function is invoked by AVS whenever its input or parameter change. A coroutine moudle executes indipendently, obtaining inputs from AVS and sending output to AVS whenever it wants. An example of a coroutine module is given by the computing of particle trajectories through a vector field during the integration of AVS with a simulation.

AVS provide a facility for debugging a module during the execution of an AVS network.

7.2 DX

The *DX Module Builder* is a point-and-click interface that facilitates the work of creating all necessary files for a user written module: a module description file, a C-code framework (or template) file and a makefile.

A module can be added to Data Explorer in one of three forms: *inboard* (linked to DX executive), *outboard* (controlled by DX executive) or *runtime-loadable* (started from inside DX).

7.3 IE

IE module builder The *IE Module Builder* is a tool the lets the users build their own modules. The great virtue of the module builder is its graphical user interface, which lets users build a basic module with no programming beyond that needed to write the computational function.

The module-building process has three main stages:

- Defining the internal structure, or "engine", by: creating a user function or subroutine and defining the input and output ports, the function arguments and their calling sequence, the relationships between the inputs and outputs and the function arguments;
- Defining the external structure, or user interface by: laying out a module control panel, associating input parameters with widgets, or control mechanisms and creating menu bar items;
- Building and installing the module in IRIS Explorer.

Module Prototyping with Shape It is possible to simplify the module building process in case of lattice-based operations. A *LatFunction module* is provided with Iris Explorer to execute programs written in the *Shape* language.

The LatFunction module is an interpreter for lattice manipulation that makes it easy to operate on lattices. It lets the user interactively change the way LatFunction acts on the data it receives, which provides a quick way to prototype a new module without going through the Module Builder or writing C or Fortran code. It also gives the user programmatic control beyond the range he would have with dials and sliders, and it eliminates the need to compile, link, and install a module each time he modifies it.

8 Memory managment

This is the most crucial topic in MVEs. Memory is a limited resource of the systems and data flow models, to which MVEs belongs, can be very consuming. This is intrinsic to this computational architecture because of the fact that any time a module receive a data as input, it allocates other memory where to store the new computed data to be sent to the next module in the network. Thus, during the visualization pipeline, some memory is consumed in order to

compute intermediate data. The release of the allocated memory is not a straightforward problem and can easily conducts to memory leaks. Furthermore, for most problems in visualization, the data sets are quite large, and it's very easy to saturate machine memory.

A set of strategies can be applied to reach the minumum redundancy of data, to avoid memory leaks, to optimize interprocess communications and to improve system overall performance.

8.1 AVS

Multiple modules in a single process Many implementations of data flow networks require a single process for each module in the network. AVS supports more than one module in a single process. This has an advantage for data transfer between modules: when two modules share the same process, local process memory can be used, eliminating the need for interprocess communications and having a significant improve in the overall system.

Shared memory data passing Shared memory are memory buffer regions setup by the operating system to allow access to the same physical memory by different processes. Multiple modules in different processes can then access the same data with no duplications.

Direct module communications In order to facilitate the flow of data between modules, the supervisor of the data flow, the *AVD Flow Executive*, allows modules to set up a direct data communication link. By this way, data copying between modules is eliminated.

8.2 DX

The *executive part* of Data Explorer is responsible for managing DX objects succesfully returned as output by modules and for the memory allocated to those objects. To deallocate memory no more used by any modules, DX uses the *data referece count* method.

Data reference counting All objects are created with a reference count of zero. If an attemp of deleting is performed when the object has a reference count of 0 or 1, then the object is invalidated and the space is freed. If the reference count is greater than 1, then the reference count is simply decremented by 1. This applies also for all the objects incorporated in that object.

8.3 IE

Shared memory arena IRIS Explorer transfers data between modules using shared memory if the modules are on the same machine and the machine supports shared memory. The advantage of using shared memory is that large quantities of data can be transferred from one module to another with very little communication overhead. All modules access the shared memory arena simultaneously, so only the address of the data has to be transferred, not the data itself. However, the data must be managed somehow, and this is done by means of reference counting.

Data reference counting All modules are responsible for collectively managing the data in shared memory. One module creates data and sends it to other modules. When all the modules are finished with that data, the space it occupies must be reclaimed. IRIS Explorer uses reference counting to manage this process. With reference counting, a module does not need to know anything about which other modules use the data. It must simply perform certain bookkeeping actions on the data it references, so that the system knows when the data should be reclaimed.

Reference counting is essential when using shared memory, but the mechanism is also general enough to be used by a single process on private data. Thus, modules executing on a machine that does not support shared memory can still use the same code for managing data memory.

As long as every module does its bookkeeping correctly and consistently, there will be no memory leaks, and data will not be reclaimed while a module still refers to it.

9 Modules distribution

9.1 AVS

AVS supports the synchronous execution of AVS modules on remote AVS hosts. The network communication and data transfert mechanism are based on Unix TCP/IP protocols while data representation is based on "External Data Representation" (XDR) allowing remote execution on heterogeneous environment.

Most frequent situations where module distribution can be successfully applied are, for example: a compute-intensive module that run best on a particular platform and input data residing on a remote host.

Furthermore, AVS has the capability to run modules in parallel when the opportunity to do so is encountered in a network.

9.2 DX

Data Explorer provides the capability of distributing the execution over multiple workstation on a network. Distrubuting the execution provides parallelism, by the simultaneous execution of different portions of the visualization on each of the workstation, and enhanced resource utilization, for example by assigning computational intensive portion of the visualization to a more powerful worstations.

Finally, Data Explorer provide some help in managing parallel processes and data partitioning on machines able to perform parallel computation.

9.3 IE

Iris Explorer allows users to run modules on remote machines. This is useful when machine resources are limited with respect to available memory or to computation capabilities when a particular data representation is required.

The local communication servers on remote machines are implemented by the *rsh* command. Data passing between modules (both local and remote) is performed across socket links.

10 Languages

All three MVEs support C, C++ and FORTRAN languages to write new modules and/or incorporate existing computation routines in MVE modules.

11 Additional information

11.1 AVS

There is another component of AVS called "AVS/Express" (also known as AVS6), bundled on request free of charge with the AVS package. Here will be summarized the main features and improvements of this package:

- unified data type which can be used to represent AVS classes of data within the same data structure. This removes the need for different visualization modules to perform the same function on different types of data; e.g., there is only one isosurface module for both field and unstructured cell data.
- a facility to extend this base data structure with user-defined fields. For example one could add patient attribute information to image scans and the modules within AVS would still recognize and process this extended data type as an image.
- support for cylindrical, polar and spherical coordinate systems and the provision for users to specify NULL (undefined) data values. These underlying data types will also be supported by the visualization modules in AVS6.
- improving in handling of large data sets. These features include:
  - data reference instead of data flow;
  - direct rendering of large data sets;
  - data chunking for processing large data sets a section at a time.
- supports on UNIX/Motif and Windows '95/NT for a distributed, multi-platform environment. Communication is supported through the TCP/IP protocol and the *External Data Representation (XDR)* format is used to provide a machine-independent representation of data transmitted between platforms.

AVS supports, by a separate product option called AVS Animator, the generation of animations based on keyframe technology to produce

11.2 DX

If a workstation has hardware support for three-dimensional graphics, then DX can utilize it. Where available, the OpenGL standard or the GL extension is used. The user has the ability to move back and forth between software and hardware rendering and progressive approximations in the same window(s) to tradeoff interactivity vs. image quality, depending on the data and the workstation configuration.

11.3 IE

IRIS Explorer has a scripting language, Skm, a command interface that allows user to run IRIS Explorer with a script.

IE's render module provides direct manipulators to view the scene.

IRIS Explorer is an Open Inventor based product. Open Inventor is emerging as a de facto standard for 3-D scene definition and is the basis of the Virtual Reality Modeling Language definition.

IRIS Explorer renders visualisations using OpenGL.

12 Conclusion and summary

MVEs are so flexible and extendible to be successfully used as post-processor modules in CFD simulations.

MVEs can also be used to help scientists optimizing their simulation experiments. For example, they can be customized to run concurrently to simulation programs. In this way, it is possible to keep track of a simulation program by visualizing its data while it is still running, without waiting for the program to end. This can be very useful during the debugging phase

of a simulation code, saving development time and computational resources (cpu time and disk space). MVEs can also be configured to change simulation parameters while the simulation program is running, giving the user the ability to steer the simulation and to see results due to the parameter changes, in order to better understand the physical process being simulated.

All MVEs examined in this document are almost quite similar, at least in theory. In practice:

- IRIS Explorer has some serious bugs in the render module, due to the fact that the current release has been completely rewritten using OpenGL and Open Inventor standard.
- Data Explorer has a good general data model for field and unstructured data but maybe too complex to be managed by a user written modules. It surely lacks a more friendly user interface.
- AVS has a longest experience that start from 1989. It is definitely the most diffuse MVE in the world. It has a good memory managment for distributed environment.

13 Reference

There are official web pages for each of the MVE examined in this document. Their URLs are:

- `http://www.avs.com/`
- `http://www.nag.co.uk/1h/Welcome_IEC.html`
- `http://eagle.almaden.ibm.com/dx/DXHome.html`