Advanced visualization with AVS/Express

Agenda

AVS/Express application examples

Data import in AVS/Express

Application dev. in AVS/Express
Development examples

1. Ferrari Racing Team – distributed CFD data access
2. MO03D – 3D reconstruction from images
3. Elasis (Fiat Auto) – immersive CFD visualization
4. STM4 – Molecular visualization toolkit
5. Pelton Turbine project

Ferrari Racing Team

Highlights:
- Standard visualization techniques + some data cut/optimization modules added
- Fluent reader added to AVS/Express
- Big data sets
- Performance were critical
- The prototype benefited by quick prototyping capabilities offered by AVS/Express (unfortunately the project stopped at the prototype phase…)

Ferrari Racing Team
MOD3D

Highlights:
- Almost all was custom code
- GUI capabilities were critical (Data Viewer not used)
- Few visualization technique used
- Added modules for zooming, marking and navigation
- More details on: [http://www.cscs.ch/~mvalle/MOD3D](http://www.cscs.ch/~mvalle/MOD3D)

ELASIS (Fiat Auto)
**ELASIS (Fiat Auto)**

Highlights:
- Standard CFD techniques
- Input already in AVS UCD format
- Use immersive Multi Pipe Edition (with wand and head tracker)
- 3D widgets usable also from standard screen

---

**STM4**

STM4 will be the theme of this afternoon session!

Highlights:
- Library of components plus two new data types
- Benefits from module architecture
- Decoupling of AVS/Express interface from code through an interface library
**Pelton turbine project**

Highlights:
- Added CFX reader
- Added specialized visualization techniques
- Added modules to overcome AVS/Express annoyances (like extents on select cell set)
- Used a lot of validation modules (like computing integrals and surface areas)
- In depth study of colormaps that helps perception of fine details
- AVS/Express was mostly a prototyping environment

**Mesh types**

```
Scattered (no geometry)          Uniform
      /
  Connected (has geometry)      Rectilinear
          |
  Structured
```

1D, 2D, 3D, 2D in 3D
Scatter data

Unstructured grids

Uniform grids
Structured grids

Cell types for unstructured mesh

Data import in AVS/Express

- Various formats directly readable
- Rd.Txt_Columns
- Database (ODBC / Oracle)
- AVS Field file format
- File import modules
- Custom readers
**Most important user data import**

AVS Field file format

Rd_Txt_Columns

**Import choices**

<table>
<thead>
<tr>
<th>Hard</th>
<th>Custom readers</th>
<th>No constraint, but need to program and know the FLD API</th>
</tr>
</thead>
<tbody>
<tr>
<td>File import modules</td>
<td>Read whatever you want and then combine with field mappers</td>
<td></td>
</tr>
<tr>
<td>AVS Field file format</td>
<td>Convert data to FLD. Support uniform, rectilinear and structured meshes</td>
<td></td>
</tr>
<tr>
<td>Database (ODBC and Oracle)</td>
<td>Read a table from the database</td>
<td></td>
</tr>
<tr>
<td>Rd_Txt_Columns</td>
<td>Textual file. OK for scatter or Line meshes</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>Various directly readable formats</td>
<td>Simples solution</td>
</tr>
</tbody>
</table>

**Formats directly supported**

- AVS field
  - Uniform, rectilinear, structured
- AVS UCD
  - Unstructured
- HDF5, Plot3D, netCDF, CGNF
  - Standard CFD formats
- Text
  - Need a table_to_scatter/table_to_uniform module
- DXF, Polygon, Triangle, AVS Geom
  - Geometries
- Images
  - BMP, GIF, JPEG, PBM, SGI RGB, TIFF
The Read_Field Module

needs to be there
default extension is:

# AVS field file
# This is a header file for a structured field
#
# ndim = 3
dim1 = 40
dim2 = 32
dim3 = 32
nspace = 3
veclen = 5
data = float
field = irregular
label = density x-momentum y-momentum z-momentum stagnation

Various examples in
data/field

Note: FORTRAN unformatted data can be read by using, for example:

variable 1 file=./for0004.dat filetype=unformatted skip=3 stride=1

File import module
Offset can be set or computed by a File Offset Variable (see next)

Offset can be set or computed by a File Offset Variable (see next)

Field mappers
Exercise

- **Import file `conc.dat`**
  - using `Rd_Txt_Columns` (skip 1 line)
- **Transform it into a scatter field using `table_to_scatter_field`**
  - (columns: x, y, z, value)
- **Bonus:** `glyph+Diamond3D` (normalize)

---

Exercise

- **Read the AVS Field format file `splash.fld`**
  - (open it to see the format)
- **Create a surface plot using `surf_plot`**
- **Change the vertical scale**
- **Add bidirectional light**
- **Change datamap (bonus: ColormapEditor)**
- **Add Legend**

---

Exercise

- **Read the AVS UCD format file `example2.inp`**
- **Build vectors on the nodes:** `combine_vect`
- **Glyph + Arrow1** (scale 1e-5)
- **Offset** to deform the cube
- **magnitude** to color the deformed cube
Exercise

- Read the AVS Field format file hydrogen.fld
- Extract a slice using orthoslice (axis 2)
- Build a surface with surf_plot
- Read mandrill.x using Read_Image
- Spread it on the surface with texture_mesh

Why an application?

1. Because AVS/Express is a developer environment for visualization applications
2. Because the user needs an application to solve a specific problem
3. Because the user doesn’t want to think

AVS/Express application types

1. Pre-built interconnection of modules.
   Modules that could be:
   1. Modified standard modules
   2. Modules that group other modules into macros (use DV* modules)
   3. Modules that encapsulate V code directly
   4. Modules that integrate C / C++ / Fortran code
2. A toolkit. The application is done by the user composing its modules
3. Runtime application (it does not need an AVS/Express installation to work)
Two AVS/Express editions

- AVS/Express Developer Edition
  - No limitations
- AVS/Express Visualization Edition
  - Some modules not present
  - Some modules cannot be expanded
  - Cheaper

Some AVS/Express resources

- AVS homepage: http://www.avs.com/
- AVS forum: http://forum.avs.com/
- AVS/Express built-in examples
- Visualization techniques manual
- International AVS Center (IAC): http://www.iavsc.org/
- IAC training material: http://www.iavsc.org/training
- Customer projects: http://www.iavsc.org/general/links/
- Other links on: http://www.cscs.ch/~mvalle/AVS/

AVS/Express project structure

- This is the MACHINE value
- Save Project - Workspace content
- XP_PATH definition
- Sources
- savev
- expr
- lib
- include
- prj
- This is the "expr.exe" value
- user.avs
**avsenv and XP_PATH**

- In the avsenv file various environment variables could be defined, but one is critical: XP_PATH
  - `CACHE_SIZE=256`
  - `MSMSSERVER=C:\CSCSlib\msms_bin\msms.exe`
  - `XP_PATH=C:\CSCSlib $XP_ROOT`

- XP_PATH defines the search order for all the Express pieces

```
v\templ.v  C:\CSCSlib ----> C:\Express
v\templ.v  v\templ.v
```

**Express and User processes**

Express process

- Project `express.exe`
- Installed `express.exe`
- C++
- Your code

User process

- Project `user.exe`
- C

**Even remote processes**

Remote process

- Installed `express.exe`
- Machine A
- Uses `v\proc.v` to specify remote process location

Machine B

- Project `remote.exe`
- C++
Multi-developer project support

```
XP_PATH=DevDir/prj GroupDir/prj /usr/Express
```

Dataflow architecture

```
class add_num
{
    public:
    float src_1;
    float src_2;
    float res;
    update();
};
add_num::update()
{
    res = src_1 + src_2;
}
```

The module
Dataflow architecture

Enter filename

Wait

Wait

Dataflow architecture

Valid data

Valid data

Execute

Wait

Dataflow architecture

Execute
Param/method interaction

- read, write: usage direction by noted method
- notify, req: declare when noted method fires

More complex module

- More than one method
  - Each triggered by a different set of parameters
- Reference modes:
  - By value (^)
  - By reference (&)
  - By pointer (*)
- Data types:
  - Atomic: int, float, byte, enum
  - Arrays
  - Groups

on instance / on deinstance

Equivalent to C++ constructor / destructor
Steps to add a computing module

1. Create a project
2. Start the Add Module Wizard
3. Remember to do a Save Project!
4. Add your code to the skeleton
5. Compile (express.dsw, express.sln or make \-f express.mk)
6. Exec the new bin\pc\express or bin\linux\express
Add Module Wizard (3 of 3)

Interface between V and code

Beware!

Save Application ≠ Save Project

Use Workspaces
Exercise

- Create a module `MyAdd` in C (two float `p1` and `p2` and float output `out`)
- Create a module `MyAdd1` in C++ (two float `p1` and `p2` and float output `out`)
- Test it
- Find the differences between the two languages

C – C++ – Fortran differences

- Same philosophy for C and Fortran
- C++ is more similar to the Express structure: every module is a class
- In C++ interfacing to OM is hidden to the user
  - In the course we will use C++ only (also as a better C)
- When is C++ not suited? Some field creation operations, some OM operations.
  - You can always call OM or FLD routine directly (using a small trick)

Another module creation method

- Use a text editor
- Create a `YourModule_mod.v` file describing the module interface
- Add it to `templ.v` (for now using `$include`)
- The module will appears in Libraries → Templates
- Now we look at a module definition example, then we introduce the V language.
Another module creation method

```cpp
module BackboneCore<src_file="Backbone.cxx",
    out_hdr_file="Backbone_gen.h",
    out_src_file="Backbone_gen.cxx",
    cxx_hdr_files="../types/MoleculeType.h",
    build_dir="cscs_proj/stm4/tube"> {

  cxxmethod+req DrawTube{
    .molecule.xyz_lst+read+notify+req,
    .molecule.atom_name+read+notify+req,
    .tube_coordinates+write
  };
  CSCS_PROJ.STM4.TYPE.MolecularType
  molecule<NEportLevels={2,0}>;
  float tube_coordinates<NEportLevels={0,2}>[];
}
```

The V language

- It is a declarative language
  - No loops, conditionals, call, etc.
- Describes (almost) exactly all that can be done graphically
- Describes a hierarchy of objects and connections between them
- V code could be entered:
  - From the prompt (called VCP)
  - In a text file to be read inside Express
  - Saving and modifying an existing application
- V has some construct not strictly related to the network ($-command, <…> properties, built-ins)

Exercise

- Create a network with the former MyAdd1 module
- Save the application into ex2.v
- Edit the ex2.v file and look at the generated V code
### Object hierarchy

- The nesting of modules
  ```
  {
  ...
  }
  ```
- Connections
  ```
  obj1 = obj2; (it is different from obj1 = obj2)
  ```
- Moving along the hierarchy
  ```
  Obj1.subobj1.subsubobj1
  ```
- Creating / deleting objects
  ```
  Type new_obj; (eg. int myint)
  -new_obj;
  ```

### Unusual constructs

- V make simple dynamic structures:
  ```
  group A {
  int len;
  float array[len];
  }
  ```
- V build-ins work on whole arrays:
  ```
  float x[5] => init_array(5, 1, 5);
  float res[5] => log(x);
  ```
- Built-ins to interrogate the hierarchy:
  ```
  group my_group[6] {
  int idx => index_of(my_group);
  string name => name_of(<-, 1);
  }
  ```

### $-commands

- List
  List an object content
- Sprint
  Print an object value
- Sint, $float, etc.
  Convert the value
- $get_array
  Print the content of an array
- $echo
  To monitor V file loading
- $help
  -
Exercise

- From the VCP prompt create an integer variable
- Assign it a value
- Create a second one connected to the first one
- Delete the first integer

Exercise

- Create a network with the former MyAdd1 module
- Save the application into ex2.v
- Edit the ex2.v file and replace the module with a sum computed in V

V data types

- Atomic data types:
  - int, float, string, enum ())
- Arrays
  - byte pixels[768][576];
  - string names[.len];
  - Beware: access an array entry by: |pixels[12][128] = 5;
- Aggregated data types (structures):
  - group
- User defined data types
  - Whatever you want
Exercise

- Create a group, call it **MyType**, and populate it
- Pretend it is your data type
- Create a variable of type **MyType**

**Properties**

- `<...>`
- Property Editor
- NE property group (NEx, NEy, NEportLevels)
- Code Management property group (src_file)
- Specific properties for C++, C

**V code organization**

- Workspaces
  - They are simply places where I can create my modules
- Templates
  - templ.v (Remember project structure?)
- Libraries
  - Organize the access to the Template modules
**Visible access to modules**

- **lib_xp.v (lib_vxp.v)**
  - (Re)defines the Libraries content adding links (using NLink)

- **Example:**
  ```
  "$XP_PATH/0/v/lib_xp.v" Libraries {
   flibrary STM4 <NEdisplayMode="opened",NEhelpFile="STM4/index.html"> {
     NElink Readers_Writers <NEdisplayMode="open"> => Templates.CSCS_PROJ.STM4.LIB.DATAIO;
     NElink Bonds <NEdisplayMode="open"> => Templates.CSCS_PROJ.STM4.LIB.BONDS;
     NElink Modules <NEdisplayMode="open"> => Templates.CSCS_PROJ.STM4.LIB.MODULES;
     NElink Full_Apps <NEdisplayMode="open"> => Templates.CSCS_PROJ.STM4.LIB.APPS;
     NElink Experimental => Templates.CSCS_PROJ.STM4.LIB.EXPERIMENTAL; 
   };
  }
  ```

**V for gurus**

- `+nres`
  - If you have undefined errors when loading libraries
- `+notify +noreq`
  - For triggering problems
- `merge()`
  - To substitute part of a structure
- `$match`
- `#ifdef`
- `$include / $resolve / $XP_VPATH`

**merge()**

```

group original {
  int other = 99;
  float value = 7.3;
};

group substituted {
  float value = 1.2;
};

group &modified => merge(substituted, original);

$float modified.value 1.200000
$int modified.other 99
$float original.value 7.300000
$int original.other 99
```
Exercise

- Create an empty application
- Add to it the previous example
- What happens modifying the two values?

Module programming

"Put flesh around the AVS/Express generated skeleton"

1. Argument passing (C++ is simpler)
2. FLD reading/writing from a module
3. Use OM and FLD C routines from C++
4. Accessing OM flow control from C++

Argument passing

- Atomic data
  
  ```
  module_param = local_var;
  local_var = module_param;
  ```

- Array (do not forget ARRfree())
  
  ```
  float *xyz_arr =
  (float *)module_array.ret_array_ptr(OM_GET_ARRAY_WR);
  xyz_arr[123] = 45.6;
  ARRfree(xyz_arr);
  ```

- To read use OM_GET_ARRAY_RD, to modify use OM_GET_ARRAY_RW
Argument passing

- **Set array size**
  
  ```c
  module_param.set_array_size(123);
  ``

  Or directly from V

- **Set multi dimensional array sizes** (e.g. `labels_pt[1][3]`)  
  
  ```c
  int d[];
  d[1] = 1;
  d[0] = 3;
  OMset_array_dims(labels_pt.obj_id(OM_OBJ_RW), 2, d);
  ``

Argument passing

- **Get array size**

  ```c
  len = module_param.ret_array_size();
  int len;
  float xyz_arr = (float *)
  module_array.ret_array_ptr(OM_GET_ARRAY_RD, &len);
  ``

- **Get multidimensional array sizes**

  ```c
  int ndims;
  int dim_sizes[OM_ARRAY_MAXDIM];
  OMget_array_dims(my_array.obj_id(OM_OBJ_RW),
  &ndims,
  &dim_sizes);
  ``

Argument passing

- **groups**
  
  Simply use structure notation: `mygroup.param = 12;`

- **string arrays**
  
  ```c
  atom_labels.set_str_array_val(idx, "label");
  char buffer[256];
  char *s = in.atom_name.ret_str_array_val(idx, buffer);
  ```
## Field read/write

- Use the provided C++ skeleton
- Import/export arrays and use combiner modules
- Use FLD routines for specialized functions

```c
FLDset_cell_data(out_density.cell_set.obj_id(OM_OBJ_RW), 0, DTYPE_FLOAT);
FLDset_node_data(ncomp.in.obj_id(OM_OBJ_RW), &node_data);
```

- To pass the required OMobj_id use:

```c
Object_in_module.obj_id(OM_OBJ_RW)
```

## Access OM flow control

- To check who triggered the method
  ```c
  obj.changed(seq_num)
  ```

- To check if the object is valid (without using +req)
  ```c
  obj.valid_obj()
  ```

- To modify a parameter so it changes value immediately without waiting for module execution end

```c
for(i=0; i < 1000; ++i)
{
    push_ctx();  out_value = i;  pop_ctx();
}
```

## Other useful routines

- Error messages
  ```c
  ERRerror()
  ```

- Confirmation dialogs
  ```c
  ERRsync_yesno_dialog()
  ```

- Update progress bar
  ```c
  OMstatus_check()
  ```

- Execute V code
  ```c
  OMparse_buffer(pair_list.obj_id(OM_OBJ_RW), "pair_list=>;", 0);
  ```

- Make array really empty
  ```c
  obj.set_array(OM_TYPE_FLOAT, NULL, 0, OM_SET_ARRAY_FREE);
  ```
Exercise

- Read a list of 3D points from a file
- Create a module that reads them and interpolates a user given number of points between them
- What can I use to move a glyph along the interpolated path?

Organize your V code

- Module definitions
  Module_mod.v
- Macros
- Applications
- Libraries ("file" lib, .vo files)
- The visible Libraries
  lib_xp.v (lib_vxp.v)
- Correlated properties
  compile_subs

Application structure

- IAC projects structure
  (http://www.iavsc.org/repository/pfiles/)
- Divide Application / UI (UI => param, avoid the reverse)
- Low level modules / (medium) / end-user modules
- Parameter Blocks
- Static data
**Code organization**

- Build dir
- Naming conventions (*_gen, *_mod, *_struct)
- Use out_hdr_file / out_src_file
- cxx_members to add methods
- cxx_name if needed
- c_src_files / cxx_src_files etc. / link_files
- c_hdr_files / cxx_hdr_files for class definitions

**Suggested steps**

- Create the base structure (templ.v, lib_xp.v, libZZZ.v)
- Create object / test / Save Objects…
- Create modules using Wizard or cut&paste
- Use ScratchPad or directly edit files (Save Proj modifies the layout)
- Modify lib_xp.v
- Base_gen_proc express (script)
- Express.sln / make

**Interesting API**

- Asynchronous data handling (EVadd_select())
- Dynamic object creation
- GEOM API
Interesting accessories

- parse_v
  - to create connections, to instantiate objects
- instancer
- copy_on_change
  - Eg. for increment / decrement
- shell_command
- error_handler
- Macro array
  - index_of() / name_of()
Application performance

- Instancer (<instanced>=0> + instancer module)
- Use menu Configure
- Use $count_ob and reduce object count
- Create "light" objects (Cross2D, point_mesh, DataObject)
- Remove useless operations on viewers (like caching)
- Verify memory usage (set_array / $set_array_trace)
- Reduce data copy

Application performance (cont.)

- Orthoslice / TileRenderer
- Disable status / module flashing
- No autonormalize, no caching
- Link instead of &group_ref
- Define who allocate and who use the memory
- Use DV modules
- Uviewer 3D/2D
- Reduce cell sets count

Portability

- The whole world is not Windows
  - CamelCaseFileNames
  - Prefer "/" versus "\"
- Avoid absolute paths
  - Use getenv("PRJDIR") or $ICONS
  - Use FILEmap_variables("XP_PATH<1>/data/atom-data.dat", file_buf);
- Use #ifdef for V and C++
- Use relative dimensioning for GUI
**Express Runtime creation**

- Build & test (express process)
- Configure
- Save Compiled Project (selecting the application)
- Edit avsenv (XP_PATH= )
- Create go.bat (env var + bin\pc\express -novcp)
- Clean RT directory / Add application data
- License and SET XP_FEATURE=XP_RUNTIME
- Test ??? (add need_obj, lib_deps)

**ARRIVEDERCI!!**

I hope you have enjoyed this introduction

Don’t hesitate to contact me with your questions, curiosities and crazy ideas

(Remember, I’m here also this afternoon for STM4)

[Website](www.cscs.ch/~mvalle)
[Email](mvalle@cscs.ch)