Concepts and Vishions

Werner Benger
Scientific Visualization group at CCT
FAQ

- Can VISH visualize my data and make pretty images?
  - Simple answer: **NOT YET 😞**

- Then, what is it good for?
  - It will soon be able to do so, and do it better than anything else 😊

- Next slides: how?
VISH Status

- VISH is currently under development
- Close to reach end-user productivity level, but not yet fully at this stage
- Development is demand-driven
- Screenshots!
Example: Particle System
Example: Trajectories
Example: Fragmented Orthoslice
GUI: Iconized Network
User Interface Representation

Is just a plugin
A view to the underlying kernel

A student can write an alternative (much prettier) representation without messing with the entire system.
Mesh Refinement

Currently supports orthoslices of uniformly spaced mesh refinement grids
Time Slider

Design Goal:
Navigation should be equally easy and fast in space and time
Example: Specular Streamlines
Example: Vector Arrows
SPH Particle Data set
(2.6 mio particles, interactive)
VISH Components

Application
Interface Implementation

User Interface
Event Handling, User Interaction

Data Storage
Memory Layout

VISH Kernel
Abstract Interfaces and Plugin Mechanisms

FiberLib2
Data Model for Scientific Data

Visualization Algorithm
VISH Components

- Kernel with object management and (runtime) plugin mechanisms
- User Interface plugins (e.g., QT frontend)
- Data model (systematic treatment of scientific data via a common approach)
  - I/O layer as runtime plugins (file formats, streaming)
- Visualization infrastructure
  - OpenGL caching mechanisms
What stands “VISH” for?

- A **Visualization Shell**
- A framework for realizing **Visualization Wishes**
- Something else... (free to imagination)
- Pronunciation (proposal):
  - Even times: “fish”
  - Odd times: “wish”
  - Maritime naming convention
VISH Kernel: ocean

- **Database-like kernel**
  - Abstract objects with inputs and outputs
  - Plugin mechanisms
  - Data and control flow management
- **OpenGL support library**
  - Layered rendering
  - Multidimensional Cache management
VISH GUI: qVISH

- Reference prototype implementation
  - based on QT (both qt3 and qt4 possible)
  - Plugin to the kernel
  - Consists of several plugins itself, such as
    - Network, scripting, tree, story representation
- Other GUI’s possible: GTK, wxWindows, FLTK, none (batch mode)
- Interfacing existing applications possible:
  - E.g. run VISH within Amira, or Scirun, or AVS or ...??
VISH data model: FiberLib2

- Systematic treatment of a wide category of scientific data, based on the mathematics of fiber bundles
- Common denominator for otherwise diverse grid types
- Plugin to VISH ("fiber-VISH" or "FISH")
- No need to use it, customized data types also possible in VISH (but bypassing the FISH infrastructure then)
• **Systematic approach for scientific data:**
  - Particle systems $\rightarrow$ unstructured grid $\rightarrow$ regular grids $\rightarrow$ uniform grids $\rightarrow$ block-structured uniform grids $\rightarrow$ curvilinear multi-block grids ... 
  - Incremental transition from one such category to next one 
  - Can cover multiple timesteps, grids, fields...
FiberLib2 I/O features

- I/O layers are plugins (shared libraries) independent of core implementation
  - Distinction among data and metadata
  - On-demand loading and creation of data
  - Cache-management
- Most powerful I/O layer is "F5"
  - 1:1 representation of the FiberLib2 into HDF5
Fiber Bundle HDF5 ("F5")

Applications
- HDF5
- Fiber Bundle HDF5
- CGNS F5 I/O

Libraries
- E.g. CGNS HDF5
- CGNS I/O
- Amira
- qVISH
- Simple Postprocessing Tools
- FiberLib2 plugin

Other File formats
- Other File formats

Data Model
- Fiber Bundle Data Model
  - C++ Kernel

Operators
- Grid and Field Operators
  - Render Operators
  - Interactive Viewer
  - Analysis Tools
C++ Coding
#include <ocean/plankton/VCreator.hpp>
#include <ocean/0Lvis0h/VERIFYObject.hpp>

using namespace Vist;

class DefaultBackground : public VERIFYObject {
    typedef Slot<int> Red, Green, Blue;

    override void render(VERIFYContext&Context) const {
        GLintnpt red = 1, green = 1, blue = 1, alpha = 1;
        int r = 100, g = 100, b = 100;

        Red << Context >> r;
        Green << Context >> g;
        Blue << Context >> b;

        red = r / 100.;
        green = g / 100.;
        blue = b / 100.;

        glClearColor(red, green, blue, alpha);

        glClear(GL_COLOR_BUFFER_BIT);
    }

public:
    const std::type_info&getType() const {
        return typeid(DefaultBackground);
    }

    DefaultBackground(const std::string&name, int p, const RefPtr<VCreationPreferences>&VP) :
        VERIFYObject(name, BACKGROUND_OBJECT+p, VP),
        Red (this, "red", 65, new VCreationPreferences("local")),
        Green(this, "green", 65, new VCreationPreferences("local")),
        Blue(this, "blue", 53, new VCreationPreferences("local"))
    {};

    static VCreator<DefaultBackground> myBackground("Background");
Hierarchical tree of substructures, five levels:
1. Time dependency (parameter space)
2. Grid object (computational domain/mesh)
3. Topological information (vertices, cells, ...)
4. Coordinate representations & relationships
5. Fields (scalar, vector, tensor)
6. (field fragments)
Simplest case: Equidistant static scalar field (float data[X][Y][Z])

- Hierarchical tree of substructures, five levels:
  - Time dependency (parameter space)
  - Grid object (computational domain/mesh)
  - Topological information (vertices, cells, ...)
  - Coordinate representations & relationships
  - Fields (scalar, vector, tensor)
  - (field fragments)
Multiple fields on same domain

- Hierarchical tree of substructures, five levels:
  - Time dependency (parameter space)
  - Grid object (computational domain/mesh)
  - Topological information (vertices, cells, ...)
  - Coordinate representations & relationships
  - Fields (scalar, vector, tensor)
  - (field fragments)
Time-dependent fields on same domain

- Hierarchical tree of substructures, five levels:
  - Time dependency (parameter space)
  - Grid object (computational domain/mesh)
  - Topological information (vertices, cells, ...)
  - Coordinate representations & relationships
  - Fields (scalar, vector, tensor)
  - (field fragments)
Multiple blocks (multiprozessor output)

- Hierarchical tree of substructures, five levels:
  - Time dependency (parameter space)
  - Grid object (computational domain/mesh)
  - Topological information (vertices, cells, ...)
  - Coordinate representations & relationships
  - Fields (scalar, vector, tensor)
  - (field fragments)
Mesh refinement or unstructured grids

- Hierarchical tree of substructures, five levels:
  - Time dependency (parameter space)
  - Grid object (computational domain/mesh)
  - Topological information (vertices, cells, ...)
  - Coordinate representations & relationships
  - Fields (scalar, vector, tensor)
  - (field fragments)
Multiblock Curvilinear grids

• Hierarchical tree of substructures, five levels:
  – Time dependency (parameter space)
  – Grid object (computational domain/mesh)
  – Topological information (vertices, cells, ...)
  – Coordinate representations & relationships
  – Fields (scalar, vector, tensor)
  – (field fragments)
Multiblock grids with refinement

- Hierarchical tree of substructures, five levels:
  - Time dependency (parameter space)
  - Grid object (computational domain/mesh)
  - Topological information (vertices, cells, ...)
  - Coordinate representations & relationships
  - Fields (scalar, vector, tensor)
  - (field fragments)
Availability

• **Code development management:**
  
  
  – Available via SVN in source code for registered users
  
  – No binary release yet
Develop Plans and Priorities

- **Visualization Algorithms**
  - Tensorfield visualization
  - *This* is research area

- **VISH and FiberLib2:**
  - Basic requirements, on-demand implementation and support (e.g. other file formats than F5)

- **User interface:**
  - Reduction to practice (not research)
  - Need manpower here
Isosurface Implementation

1. Do explicit Coding in FISH
2. Call external library such as VTK
3. Supercede isosurface computation by modern graphics hardware algorithms
   • Volume rendering with isolevels
   • Gpu-assisted histogram computation