Animation of Scientific Data Using VTK Designer

G.M.Lingaraju & Dr. C S Bagewadi.
Head R&D , Reva ITM, Kattegenahalli , Yelahanka
Bangalore, Karnataka, India 560064

Abstact

Visualization is the science of giving visual appearance to raw data for humans to understand the content of the raw data. This broad definition of visualization encompasses Information visualization, Data Visualization (Databases) and scientific 3D Visualization (VTK, OpenDX, AVS etc). VTK is an open-source IDE for creating and editing VTK visualization networks for Rapid Application Development. In this paper we introduce the design concepts behind the Visualization Toolkit Designer (VTK Designer), the objective of which was to write a reliable and comprehensive application that will make the task of designing any type of VTK based visualization pipeline easy by making use of graphical means for constructing and configuring the pipeline, also generate the source code for the constructed pipeline in any of the languages supported by VTK. This paper targets researchers of any discipline who have 2D or 3D data and want more control over the visualization process than a twist-input system can provide. It also assists developers who would like to incorporate VTK into an application as a visualization or data processing engine. Even though this paper can only provide an introduction to this feature-rich visualization pipeline designer, we've also provided references to additional material. This paper also shows the demonstration of the fluid animation as an application.

1. Introduction

Major apprehension in Computer Graphics is with the realistic rendering of scenes, improving the efficiency of algorithms and advanced modeling techniques. Visualization is related to but diverse from the subject of computer graphics. Visualization uses computer graphics to make pictures that give us insight into abstract data and signs. The related techniques of Graphical Simulation use graphics to simulate and animate the behaviors or phenomena, which we encounter in real existence. In distinction, visualization is frequently concerned with visualizing things we could not otherwise see, such as scalar fields, vector fields and tensor fields or the relationships among pieces of data in a random folder (1). Volumetric data Visualization extracts meaningful information in 3D Visualization. For instance, a succession of 2D- segment images obtained from MFU or a CT scan of a human body can be reconstructed into 3D Human model and visualized for medical diagnosis or the planning of surgery. Volume rendering attempts to represent full 3D data sets as 2D images, passing on more information than surface rendering.

Visualization has become a key enabler for pattern recognition also. It allows the large quantities of data generated either by direct observation or by simulation to be analyzed visually, providing insight and understanding the complex data, which would be difficult in the conventional recognition methods. Thus we can recognize some of the complex pattern, where as in case of Computational Steering, Simulation and Visualization are strongly tied (2).

In computer graphics, efforts to develop a concept of rendering have proceeded on two fronts. One concentrates on the physics of light transport, leading to equations that explain how light passes through a medium and reflects from an object. The other concentrates on the human visual system and the brain's interpretation of an image that is received. The first approach lends itself to well known and to novel algorithms for solving complex equations. Its basic primitives are light sources, geometry, and reflectance functions. The second effort requires anticipating how a human will respond to a rendered image, which is a much more insubstantial venture (4). In the same way, a theory of visualization could be comprised of two different aspects: one depends only on the underlying data, while the other concentrates on the human response to imagery. A data-driven theory of visualization serves as a preparatory step for rendering. An abstract data set is someway changed into lights, geometry, and reflectance. After that point, rendering is performed as a post process. Another theoretical item that deserves attention is the need for techniques that handle complex and dynamic geometry. While there are many techniques for performing operations on regular volumetric grids, the development of techniques for unstructured data has lagged behind. Visualization techniques for handling large, dynamic, unstructured grids are essentially missing (3). Here is an attempt to practice the theory which we have learnt, and document the innovation we made in the process of research.

There are numerous simulation tools designed for explicit

Manuscript received March 5, 2008
Manuscript revised March 20, 2008

applications. Users need to plug in application specific parameters or models, after which the simulation tool will carry out simulation, graphics rendering, and animation(4). Various valued visualization products and packages exist, including visualization toolkits such as VTK, AVS, IRIS-Explorer, and Khoros, and user interface development toolkits such as Tcl/Tk and Motif. These simplify the implementation of a visualization system, like developers can assemble components instead of constructing every thing from scratch . Some of these packages are also relatively expensive. To some extent for these reasons, many interactive Visualization and Computational Steering systems are implemented for specific applications like in-house, drawing on the skills of specialists from engineering, computer graphics, and other areas. Recent software systems are on a tendency towards integrating Scientific Computation, Interactive Visualization, Computational Steering, and multimedia. Hardware systems appear more and more to be integrations of distributed supercomputers, visualization workstations, virtual reality tools, and other hardware devices, connected by high-speed networks (5).

2. Related Work

The landmark report (2) on Visualization in Scientific Computing by McCormick et al (1987) sparked the development of a number of interactive visualization systems, based on the dataflow paradigm. Harber and McNabb in 1990a elegantly described the reference model for this type of system. Raw visualization data is fed into a pipeline of processes, progressively filtering the data, converting it to an abstract geometric representation and finally rendering the geometry. This model is the basis for a number of visualization systems, including the system IRIS Explorer from NAG (IRIS Explorer, 2003). Here modules from a library are loaded into a visual workspace and connected into a network reflecting the Haber-McNabb model. An important aspect of Simple visualization pipeline in IRIS Explorer is their openness and extensibility. While many standard modules are delivered with this system, a user can encapsulate their own code as a module, add this to the library, and load into the workspace like any standard module. This extensibility has allowed these systems to evolve with advances in computing technology, and IRIS Explorer. Example: It remains widely used even today, though its more than a decade old. One particular extension is important in the context of e-science: we are able to create special collaborative modules that link separate pipelines being executed by users at different geographical locations. This was first achieved in the EPSRC COVISA project for IRIS Explorer (Wood et al, 1997) and is now an integral part of the system.

Accurate visualizations and user interfaces are essential in computational steering applications. By allowing the researcher to construct his own visualization and interface with the simulation, they will be according to the researcher's wishes and demands, and can easily be adapted if the researcher's focus of interest changes. In addition, the visualization and user interface satellite can be kept general applicable and therefore be used for different kinds of steering applications. Building one's own custom 3D visualization and/or interface from geometric primitives is a topic of interest in both the visualization and user interface community. In (6) a tool called Glyph maker is described which is developed for data visualization and analysis. It allows users to build customized representations of multivariate data and provides interactive tools to explore the patterns and relations between the data. With the provided primitives points, lines, spheres, cuboids, cylinders, cones, and arrows, the user can draw glyphs using the 3D Glyph Editor. Properties of the glyphs can be bound to data using the Glyph Binder. Raw (simulation) data is transcribed by the Read Module into Explorer data structures that are used by the Glyph Binder. These bindings however, are only unidirectional from the data to the glyphs. Therefore, Glyph maker does not allow the user to steer the simulation by manipulating the geometric objects. In (7) architecture for an extensible 3D interface toolkit is presented. The toolkit can be used for construction and rapid prototyping of 3D interfaces, interactive illustrations, and 3D widgets. By direct manipulation of 3D primitives through a visual language we can construct widgets, interface objects, and application objects whose geometry is affine constrained. The four basic primitives of the toolkit are: the point primitive, the vector primitive, the plane primitive, and the graphical object primitive. Although the system does allow a high degree of direct manipulation to construct an interface, there remains several operations that have to be performed in an indirect manner, such as the definition of constraints between objects in a separate window with no visual feedback from the objects themselves.

3. Preamble about VTK

VTK(8) is an open-source , portable (WinTel/Unix), object-oriented software system for 3D computer graphics, visualization, and image processing. Implemented in Ç, VTK also supports Tcl, Python, and Java language bindings, permitting difficult applications, quick application prototyping, and straightforward scripts. Even though VTK doesn't provide any user interface mechanism, it can be included with obtainable widget sets such as Tk or X/Motif. VTK provides a diversity of data representations together with unorganized point sets, polygonal data, images, volumes, and structured, rectilinear, and unstructured grids. VTK comes with readers/importers and writers/ exporters to swap data with other applications. Hundreds of data processing filters are available to work on these data, ranging from image convolution to Delaunay triangulation. VTK's rendering model supports 2D, polygonal, volumetric, and texture-based approaches that can be used in any combination. VTK is one of several visualization systems available today. AVS (9) was one of the first commercial systems available. IBM's Data Explorer (10) originally a commercial product, is now an open source and also known as OpenDX. NAG Explorer(11) and Template Graphics Amira (see http://www.tgs.com/Amira/index.html) are other well-known commercial systems. VTK is a general-purpose system used in a variety of applications. Because VTK is open source, faculty at many universities like, Rensselaer Polytechnic Institute, State University of New York at Stony Brook, the Ohio State University, Stanford, and Brigham and Women's Hospital use VTK to teach courses and also use it as a research tool. National labs such as Los Alamos are adapting VTK to large-scale parallel processing. Commercial firms are building proprietary applications on top of the open-source foundation, including medical visualization, volume visualization, oil exploration, acoustics, fluid mechanics, finite element analysis, and surface reconstruction from laser-digitized, unorganized point-clouds. VTK began in December 1993 as companion software to the text. VTK: An Object-Oriented Approach to 3D Graphics by Will Schroeder, Ken Martin, and Bill Lorensen (Prentice Hall). In 1998 the second edition of the text appeared with additional authors Lisa Avila, Rick Avila, and Charles Law. Since that time a sizable community has grown up around the software including dozens of others as developers, often submitting bug fixes or full-blown class implementations. These community efforts have helped the software evolve. For example, David Gobbi in the Imaging Research Laboratories at the John P. Robarts Research Institute, University of Western Ontario, has reworked VTK's transformation classes and is now an active developer.

Architecture of VTK consists of two major sections: a compiled core (implemented in C++) and an automatically generated and interpreted layer. The interpreted layer currently supports Tcl, Java, and Python. VTK currently supports three types of volume rendering ray tracing, 2D texture mapping, and a method that uses the VolumePro graphics board (12).

4. VTK: Visualization Toolkit

VTK, as the name suggests, is a toolkit for performing data visualization. VTK is implemented in C++, and is hence an object-oriented system. It makes use of MesaGL or OpenGL to render graphic primitives. More information on VTK can be obtained from (8).

In VTK, data visualization is done with the help of something called pipeline. Simply put a pipeline is a collection of VTK objects connected in a specific way to render data. A pipeline consists of the following elements

1. **Data Source**: The source forms the starting point of visualization. As the name suggests the source provides data that needs to be visualized. The data source can provide this data by reading a file, socket or any input device or by mathematically evaluating an equation. The function of the source it to simply provide raw data.
2. **Data Filter**: Data sources may provide unwanted data at times. A data filter takes the data provided by the source, and filters out the unwanted data in it. The filtered data can then be passed on to the next stages in the pipeline.
3. **Mapper**: A mapper is the geometric representation of an actor.
4. **Actor**: An actor represents an object rendered in the scene, along with its properties and position. It can be treated as logical entity in the scene.
5. **Renderer**: A renderer coordinates the rendering process involving lights, cameras and actors.
6. **Render Window**: Manages a window on the display device, where the rendered graphics will be drawn. The above elements when connected in sequence form a pipeline.

The class hierarchy of VTK classes is organized into class trees. Here each tree is heading a host of classes, which fall into one of the above categories.

1. **vtkSource** heads the family of classes that produce data for visualization
2. Filters take some input from sources and produce a filtered output to its consumers. Hence filters can be thought of as sources, ones that produce filtered data. That is why we do not have a vtkFilter in VTK. Different kinds of filters are headed by classes like **vtkDataSetToPolyDataFilter**, **vtkDataSetToDataSetFilter** etc.
3. **vtkMapper** heads the family of mappers
4. **vtkProp** heads the family of actors. (Though we will be working with vtkActor and vtkAssembly most of the time)
5. **vtkRenderer** heads the family of renderers
6. **vtkRenderWindow** heads the family of render windows

5. VTK Designer

VTK Designer is an object oriented, plug-in based software written using VTK and Qt to make the task of designing VTK visualization pipelines easy. VTK Designer does make the life of VTK programmers and VTK users very easy.

VTK Designer is made of the following components.

1. Wrapper Class Library
2. Designer front-end
3. Code Generator
4. Plug-ins
5. XML IO Module

This paper briefly describes the function of each of the above components and also explains the relation between them. The paper also explains why and how Qt is made use of in VTK Designer.

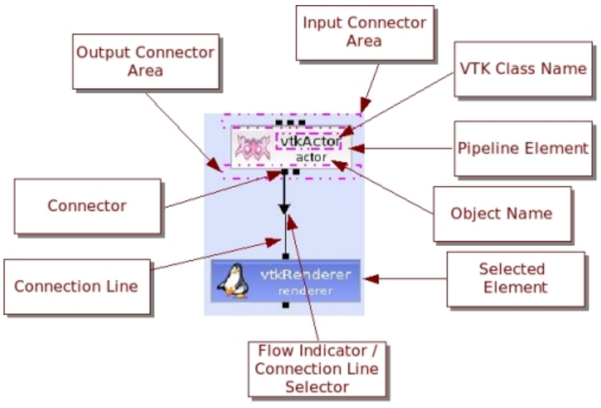


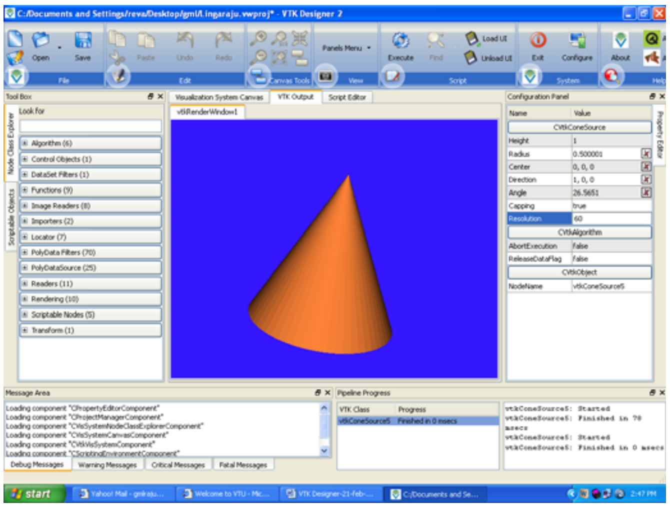
Illustration 1: Pipeline Element and Connectors

In VTK Designer each connector has a data type associated with it. The data type of an input and output connector should match for a connection to be successful.

Output connectors have relation type in addition to data type associated with them. Most connections are input/output and hard connector types, but some connections communicate association rather than input/output. Such connection lines are drawn using as dotted lines. Selected connections lines are drawn in red color. Only one connection can be selected in a canvas at any given point of time.

Some input connectors allow multiple links while most don't allow multiple links. If a connection already exists on an input connector of a pipeline element, another connection to the same connector will be discarded unless multiple links are allowed.

Given below is a simple pipeline and its corresponding output.



Name	Value
CVtkConeSource	
Height	1
Radius	0.500001
Center	0, 0, 0
Direction	1, 0, 0
Angle	26.5651
Capping	true
Resolution	60
CVtkAlgorithm	
AbortExecution	false
ReleaseDataFlag	false
CVtkObject	
NodeName	vtkConeSource5

```
Loading component "CPropertyEditorComponent"
Loading component "CProjectManagerComponent"
Loading component "CVisSystemNodeClassExplorerComponent"
Loading component "CVisSystemCanvasComponent"
Loading component "CVtkVisSystemComponent"
Loading component "CScriptingEnvironmentComponent"
```

VTK Class	Progress
vtkConeSource5	Finished in 0 msecs

```
vtkConeSource5: Started
vtkConeSource5: Finished in 78
msecs
vtkConeSource5: Started
vtkConeSource5: Finished in 0 msecs
```

Illustration 2: VTK Designer FrontEnd (cone example)

Qt: C++ GUI Toolkit

Qt is a cross platform C++ GUI toolkit developed and maintained by Trolltech A. S. Qt is used in VTK Designer for the following reasons:

1. **GUI** components for use across different windowing platforms.
2. **Robust Meta Object System**: Qt has a well-designed Meta object system for QObject subclasses. The Meta object system provides a transparent way for accessing Meta information of QObject subclasses. VTK Designer uses Qt's Meta Object system mainly for providing transparent access to configurable properties in the wrappers.
3. **Signal Slot Mechanism**: Qt has the best inter-object communication mechanism. It is very easy to use and extremely robust.
4. **XML**: Qt's XML module provides a fast and easy way for managing DOM trees. Qt's DOM classes are used by VTK Designer to store and retrieve VTK Designer Pipeline Files.

Wrapper Class Library

The key motivation behind writing VTK Designer was to have a system that makes pipeline construction easy. One of the primary goals of VTK Designer was to create a framework via which properties of VTK classes could be queried and modified transparently and connections between different elements in the pipeline could be made visibly.

The Wrapper Class Library (referred to as WCL here afterwards) does just that. The WCL provides a framework for writing wrappers that provide transparent access to properties and connections of a VTK class.

The WCL is written in Qt/C++. This is because Qt provides a robust meta-object system using which properties and features can be accessed transparently.

Designer Front-end

The designer front-end provides a workspace for a pipeline designer to pull Wrappers off the WCL and arrange them on a canvas and connect elements to form a pipeline.

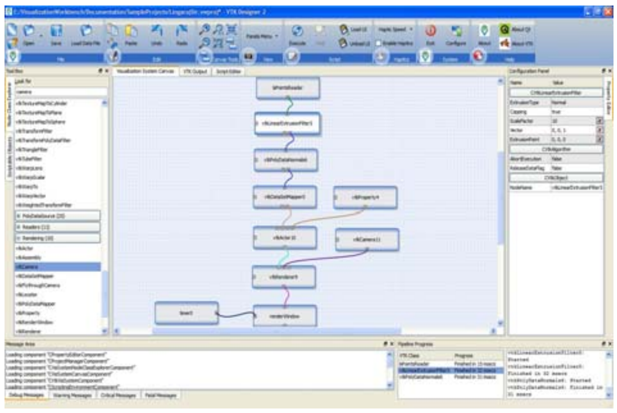


Illustration 3: Pipeline Execution for fluid animation.

The designer front end consists of the following key components

1. **Wrapper List**: This is a list of wrappers presented as a tree list on the left hand side of the main window. Querying the plugin importer module populates the contents of this window. It provides a means for displaying wrappers in an organized way and creating wrappers on demand.
2. **Pipeline Editor**: This is a canvas like widget. Pipeline elements are placed on the wrapper and connections are made. The pipeline editor provides support for pipeline parts. A group of elements within a pipeline can be put into a pipeline part. This part can be used as a single component in another pipeline.
3. **Pipeline Executor**: This module checks whether the constructed pipeline is correctly made. If so, it will execute the pipeline and display the result.

The interface between the WCL and the Front-end comprises of a limited set of classes in the WCL. This interface is described in detail in the document on WCL

Code Generator

VTK has been systematically designed and implemented. The function names for property methods and pipeline connections are predicable and form a pattern. The WCL captures this pattern and stores them as Meta data. The Code Generator module to generate the code for the designed pipeline queries this Meta data.

The WCL internally stores the pipeline structure. It stores information about the elements used to make the pipeline, their properties and also how they are connected. If elements are organized into parts/modules information about that is also stored. The code generator interface provides means to access this data in a programmer friendly way for concrete implementations to generate the code.

The Code Generator interface is sufficiently extensible to fit code generators for any language as long as VTK provides wrappers for that language. Currently the C++ code generator has been implemented.

Plug-ins

The WCL can be extended via plug-ins. One of the components in VTK Designer is a plug-in importer. The plug-in importer loads installed VTK Designer wrapper plug-ins at load time.

The plug-in interface provides a means for VTK Designer to import wrappers.

XML IO Module

The XML IO Module does the job of saving and retrieving VTK Designer Files. Pipelines can be described in terms of the elements that make it, their properties and connection paths. XML provides a very good way for representing such data. The XML IO Module makes use of Qt's XML module to save and retrieve VTK Designer pipeline information.

Given below is a block diagram of VTK Designer. It shows the key components and the relations between them.

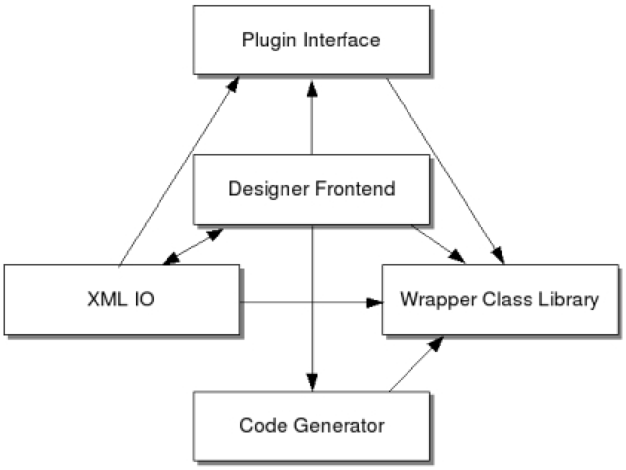


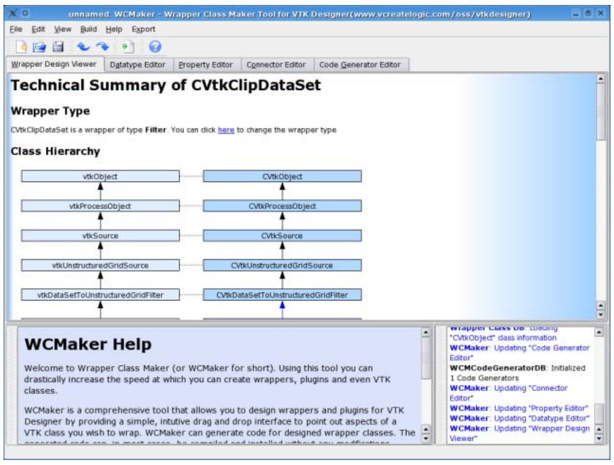
Illustration 4: VTK Designer Modules

The arrows are meant to indicate whether a component knows the existence of the other or not. For example, the Designer Front-end knows the existence of the Code Generator, but the reverse is not true and so on.

Other Modules

WCMaker

WCMaker is a comprehensive tool that allows you to design wrappers and plug-ins for VTK Designer by providing a simple, intuitive drag and drop interface to point out aspects of a VTK class you wish to wrap. WCMaker can generate code for designed wrapper classes. The generated code in most cases can be compiled and installed without any modifications.


unnamed: WCMaker - Wrapper Class Maker Tool for VTK Designer(www.vcreatelogic.com/oss/vtkdesigner)
File Edit View Build Help Export
Wrapper Design Viewer
Datatype Editor
Property Editor
Connector Editor
Code Generator Editor
vtkObject
CVtkObject
vtkProcessObject
CVtkProcessObject
vtkSource
CVtkSource
vtkUnstructuredGridSource
CVtkUnstructuredGridSource
vtkDataSetToUnstructuredGridFilter
CVtkDataSetToUnstructuredGridFilter

Technical Summary of CVtkClipDataSet

Wrapper Type

CVtkClipDataSet is a wrapper of type **Filter**. You can click here to change the wrapper type

Class Hierarchy

WCMaker Help

Welcome to Wrapper Class Maker (or WCMaker for short). Using this tool you can drastically increase the speed at which you can create wrappers, plugins and even VTK classes.

WCMaker is a comprehensive tool that allows you to design wrappers and plugins for VTK Designer by providing a simple, intutive drag and drop interface to point out aspects of a VTK class you wish to wrap. WCMaker can generate code for designed wrapper classes. The

```
"CVtkObject" class information
WCMaker: Updating "Code Generator Editor"
WCMCodeGeneratorDB: Initialized 1 Code Generators
WCMaker: Updating "Connector Editor"
WCMaker: Updating "Property Editor"
WCMaker: Updating "Datatype Editor"
WCMaker: Updating "Wrapper Design Viewer"
```

Illustration 5: WCMaker Tool

vdf2cpp

vdf2cpp is a **command line utility** for converting vdf (VTK Designer XML files) to C++ code.

Wrapper Class Library

This Section defines wrapper classes and describes how the wrapper class library provides transparent control of the VTK classes to the VTK Designer system

CVtkPlugin, CVtkPluginImporter : Plug-ins provide a way by which one can extend the basic library; dynamically without having to recompile the whole library.

Plug-ins are accessed via the plug-in importer; which is an object of the class CVtkPluginImporter. CVtkPluginImporter looks for plug-ins in a directory whose path relative to the path stored in the environment variable VTKD_HOME. It initializes each plug-in in that directory.

In WCL a single plug-in can provide many wrappers. Wrappers are created from plug-ins via the create() method. The create() method accepts the class name of the wrapper as a string and based on that it will create the wrapper.

Each plug-in maintains a plug-in table which it will refer to when a new plug-in is to be created. So writing a plug-in will now become as simple as declaring the plug-in table. The language used here is a little confusing. The plug-in table in the plug-in actually holds information about wrappers, whereas the plug-in table in CVtkPluginImporter holds information about the plug-ins. The following diagram should help make things clear.

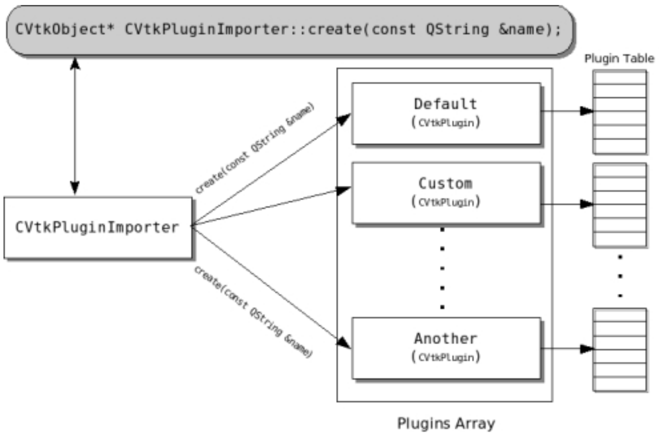

CVtkObject* CVtkPluginImporter::create(const QString &name);
Plugin Table
Default
(CVtkPlugin)
Custom
(CVtkPlugin)
Another
(CVtkPlugin)
CVtkPluginImporter
create(const QString &name)
create(const QString &name)
Plugins Array

Illustration 6: VTK Designer Plugins

VTK Designer Front-end

This Section briefly describes the different components in the VTK Designer front-end. The implementation of these components is fairly straight forward.

The main window consists of the following key components

1. **Pipeline Objects**: This component provides a visual representation of the wrappers provided by different plug-ins. It presents these wrappers grouped by type name and also as a tree list under each type. It also interfaces with CVtkPluginImporter to create wrappers on demand.
2. **Pipeline Editor**: This component stacks pipeline views one behind the other on which the pipeline can be created and edited.
3. **Property Editor**: This component displays properties or the pipeline element currently selected in a list view.
4. **Pipeline Thumbnail View**: This component provides a snapshot of the pipeline and also a window control to navigate within the canvas.

5. **Pipeline Display**: This component shows the output of a pipeline in a rectangular window.

Application

Considering the scientific data which has been generated in our simulation of fluid flow, we have animated this flow pattern using the VTK Designer as a part of our research work. The snap shots of the Fluid Flow Animation have been shown in illustration 7 & 8.

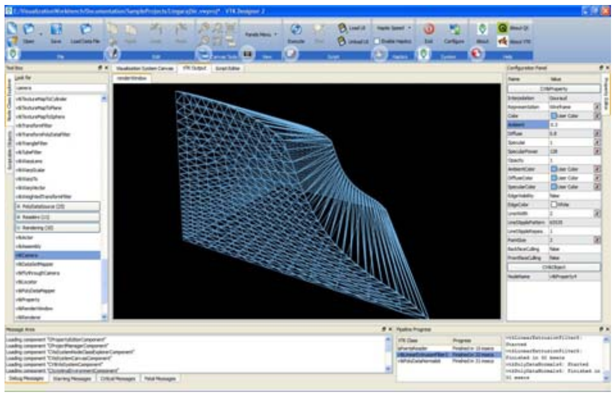


Illustration 7:Animation of Fluid Flow (Wireframe model)

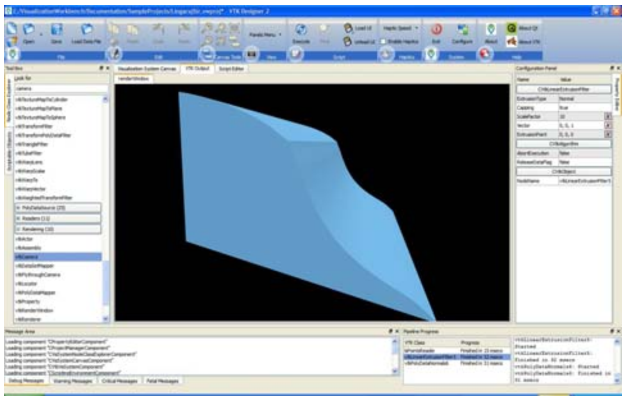


Illustration 8: Animation of Fluid Flow (after rendering).

6. Conclusion

Visualization is a complex science that involves a lot of computational details. VTK is a well tested framework of C++ classes that helps in abstracting a large part of these details to perform accurate visualization of scientific data. While VTK has a lot to offer in the visualization space, it lacks a good GUI that can help users to assemble VTK objects into a pipeline to visualize a specific kind of data set. VTK Designer fills in this gap. VTK Designer helps increase the speed at which VTK visualization networks can be prototyped, tested and tweaked.

VTK Designer was created to help rapidly prototype VTK based pipelines by using visual means and have the code for the pipeline generated. While the project set out to achieve a lot of goals like design of visualization pipeline modules, complete pipelines, parallelization of pipelines and generation of code in several languages, it has achieved a majority of the goals if not all. Parallelization, Computational Steering and Haptic Display are some of the areas that still needs to be worked on in VTK Designer and they are the future goals for the software.

Acknowledgment

Special thanks to Vcreatelogic, Bangalore for the contribution in creation of visualization pipeline .This is of a great support for the scientist community in their research work. [14].

References

[1] Jim X. Chen and David Rine Horst D. Simon "Theme Editors' Introduction: Advancing interactive Visualization and Computational Steering " IEEE computational science & engineering winter **1996**

[2] Jason Wood, Ken Brodlie "gViz – Visualization and Steering for the Grid "School of Computing, University of Leeds Jeremy Walton, NAG Ltd

[3] J.X. Chen and O. Frieder, "The Applications of Computer Graphics and Software Tools," *Computing in Science & Engineering*, vol. 1, no. 6, Nov./Dec. 1999, pp. 83–87

[4] *Jim X. Chen, Yonggao Yang, and Xusheng Wang,"* physics-basedmodeling and real-time simulation" computing in science & engineering may/june 2001

[5] V; Anupam et **al.,** "Distributed and Collaborative Vidzation," *Cmpltw*, Vol. 27, No. 7, July 1994, pp. 37-43.

[6] W. ltibarsky, E. Ayers, J. Eble, and ***S .*** Mukherjea. Glyph maker: "Creating customized visualizations of complex data" *IEEE Computer***,** 27(4):57-64, July 1994.

[7] M.P. Stevens, R.C. Zeleznik, and J.F. Hughes." An architecture for an extensible 3D interface toolkit." In *Proceedings of the UIST '94 Conference***,** pages 59-67, November 1994.

[8] W. Schroeder, K. Martin, and W. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 2nd ed., Prentice-Hall, Old Tappan, N.J., 1998.

[9] C. Upson et al., "The Application Visualization System: A Computational Environment for

Scientific Visualization," *IEEE Computer Graphics and Applications*, Vol. 9, No. 40, July 1989, pp. 30-42.

[10] *Data Explorer Reference Manual*, IBM, Armonk, New York, 1991.

[11] *IRIS Explorer User's Guide*, Numerical Algorithms Group, Oxford, UK, 2000.

[12] H. P.ster et al., "The VolumePro Real-Time Ray-Casting System," *Proc. Siggraph 99*, ACM Press, New York, Aug. 1999, pp. 251-260.

[13] William J. Schroeder, Lisa S. Avila, and William Hoffman *Kitware* " Visualizing with VTK: A Tutorial "September/October 2000 IEEE Computer Graphics and Applications

[14] http://www.vcreatelogic.com/

Dr.C.S.Bagewadi, received his Master's degree in Mathematics with Gold Medal and Ph.D. in Differential Geometry from Karnatak University, Dharwad, India, In 1982 he joined as a Reader in the department of Mathematics and became a professor in 1993. He served as an Acting Vice Chancellor Kuvempu University, Dean, and faculty of Science & Technology, Chairman Department of Mathematics & Computer Science, Kuvempu University. His research interests include the areas of Fluid Mechanics, Differential Geometry and Computer simulation and Graphics. Presently he is working as a Chairman, Department of Mathematics & Computer Science Kuvempu University, Shimoga, India

Mr .G M Lingaraju received his Bachelor of Engineering and Master of Technology (CADS) from University of Mysore, Karnataka. He has worked in the department of Computer Science & Engineering as Lecturer/assistant Professor, from 1995, at present He is a Research Scholar at Kuvempu University in the Department of Computer Science, Karnataka. His areas of interest include Computer Graphics, Real time Scientific Visualization, Haptics and Computational Steering.