QteVtk – a Multi-Platform, Object-Oriented Visualization Environment Extending *VTK*

Stanislav L. Stoev and Wolfgang Strasser

Computer Science Department,
University of Tübingen,
WSI/GRIS, Auf der Morgenstelle 10, C9,
72076 Tübingen, Germany
{sstoev,strasser}@gris.uni-tuebingen.de

ABSTRACT

In this study, we present a new comprehensive visualization environment, based on the *VTK*-library. We first introduce a window system independent graphical interface for the *VTK*-classes and it's object-oriented design, then we describe a set of viewer and editor classes for displaying and editing different data types available in the *VTK*-library.

The described library (*QteVtk*) provides a graphical user interface (GUI) to ease creating *VTK*-visualization pipelines with graphical appearance. Furthermore, it implements saving, loading, and adjusting of objects and object parameters, while supporting *VTK*-concepts like object oriented design and "demand driven" update of the visualization pipeline for data-flow control. *QteVtk* provides for the first time an easy to use combination of a free object oriented visualization with multi-platform GUI elements.

The presented work came into being, because in the authors' opinion the visualization still lacks a free, well designed, in terms of object oriented data structures and algorithms, but powerful and easy to use visualization library for implementing visualization applications with the latter, as well as adding new data structures and techniques to it.

Keywords: Computer graphics, data visualization, object oriented visualization, data flow, visual programming, vtk.

1 Introduction and Related Work

When visualizing data, we are mostly interested in a simple but vivid way to gain insight into the data in order to get the best possible impression of it's content. Several data exploring environments have been developed over the past years, which claim to meet the requirements of the users in this computer science area. To the most popular systems count the visualization environments *AVS* [Upson89], *IBM Data Explorer (OpenDX)* [Dat91], *Iris Explorer* [Iri91], *Khoros* [Rasur91, Konst94], and *VTK* [Schro98, Schro96]. These are very potent ones, which, however, have various limitations. *AVS* and the *Iris Explorer* are very large, multi-functional, but commercial systems, hence not allowing for code manipulation, optimization, and adaption. They do not support system independent application generation, thus the created "pseudo" applications can only be run in the *AVS*/*Iris Explorer* environment (e.g. AVS-Network Editor). This is a consequence of the utilization of a powerful, yet commercial underlying visualization library, which one can only use as a black box, without being able to optimize or adapt particular algorithms or data structures. On the other hand, one of the greatest advantages of these two systems, is their portability, allowing their application on different workstations, within different operating systems with the same interface and environment.

Conversely, *Khoros* [Rasur91], is only available for systems supporting the X11 protocol. Despite this limitation and the fact, that it only supports the generation of C-code, it provides a very useful visualization environment. Similar to these products, the *IBM Data Explorer*, supports visual programming. Another similarity with *Khoros* is that the *IBM Data Explorer* is based on X-Window and Motif, which negatively influences its portability. This holds also for the *ImageLab*, which only provides an interface to the underlying *ImageVision Library* developed, optimized, and available only for IRIX-based SGI-machines. Thus, besides its usefulness when processing and visualizing image or volume data, this environment cannot be ported and used on other platforms.

2 Motivation

The main drawback of the introduced systems is that, when a user tries to generate source code for a constructed visualization pipeline, in order to create a stand alone visualization application with a special purpose, most of them will produce insufficient results. This may be either because the code is not object oriented, hence difficult to understand, manipulate, and re-use (e.g. *Khoros*, *IBM Data Explorer*), or only part of it can be utilized freely (due to license limitations), or because the created visualization pipeline can only be evaluated in the development environment (i.e. *AVS* or *Iris Explorer*, where the underlying structures, algorithms, and in general code cannot be manipulated). Another limitation is the graphical user interface provided by these systems. Often the integrated GUI-builder has to be used, which may in turn restrict the portability of a visualization application, even if the basic algorithms and techniques are portable. Keeping in mind these disadvantages, we developed an easy to use, while powerful multi platform environment, disposing of an object oriented graphical user interface and based on the (in our opinion) most promising visualization package: *VTK*. Hereby we are addressing both visualization library users, utilizing *VTK* for creating problem specific visualization application *and* scientists, developing new data processing algorithms, new problem specific data structures, and rendering techniques.

Perhaps the most important feature of a visualization system, is its availability on a variety of common operating/windowing systems, as long as this defines its popularity and acceptance. Moreover, a visualization environment has to be well structured in terms of code design, large enough in terms of available algorithms, and simple enough (but not simpler[1]), to encourage the developers from distinct areas of the computer science to utilize it. This is in our opinion the only way to overcome the tendency during the past years, to develop promising algorithms, while condemning their reasonable utilization to sole academic existence. We see the main reason for this trend in the commercialization of the visualization libraries and toolkits, hence optimizations and accelerations in such systems is not possible. On the other hand, many users rather programme their own visualization environments, than using an expensive, commercial one.

To summarize: The success of a visualization system, excluding applications for 2D-image processing, becoming very popular in the past years, depends on its *functionality*, *availability*, *portability*, *usability*, and of course *stability*. With the term *usability*, we mean the time cost for (unexperienced) users to get acquainted with the utilization of such a library and to be able to create applications for the visualization of data (from an area) of current interest. A (subjective) comparison of the introduced packets concerning these criteria is depicted in Table 1. The column with the title *design* gives a brief idea of the underlying concept of realization, implementation language, and re-usability of the developed data structures and algorithms. This is of importance for the second addressed user group: the developer of new visualization techniques.

We believe, that the most attractive toolkit in this aspect is the *Visualization Toolkit VTK* [Schro98]. *VTK* is a comprehensive, completely object oriented library, providing an easy to use API for data access, manipulation, processing, visualization, and creation of visualization applications. One of the most valuable features of this visualization library is its free availability, which makes it widely used and supported. Furthermore, the *VTK*-software is independent of the windowing/operating system, hence *VTK*-applications can be readily ported to machines other than the one they are developed on. Moreover, *VTK* solves the performance-portability

[1]Things should be as simple as possible, but not simpler (A. Einstein).

System	*functionality*	*availability*	*portability*	*usability*	*design*
AVS	very good	commercial	any OS	average	good
IBM Explorer/OpenDX	average	free	Unix	average	average
Iris Explorer	very good	commercial	any OS	average	good
Khoros	very good	commercial	X11-based	average	average
ImageLab	average	free	IRIX	good	poor
VTK	very good	free	any OS	poor	very good

Table 1: A (subjective) comparison of visualization system's features.

tradeoff providing hardware and operating system independent classes for fast data access and implementing proper algorithms exploiting these data-structures. Another reason why we chose *VTK* is, because the use of a well designed data structures and algorithms library in the construction of visualization applications is the key to this, since it provides the user and developer with a high degree of confidence in the knowledge, that the implementation of the algorithms has been extensively tested and used in a wide range of other applications.

The main drawbacks of this toolkit are the lack of a graphical user interface and the limited number of data viewing classes. Therefore, creating a visualization pipeline, as well as changing parameters of former constructed ones ”on-the-fly” is not possible. If a user intends to use such functionality, the interpreted, Tcl/Tk-based version has to be used, which, however, is much slower, not object oriented, and does not solve the data viewing problem addressed above. In other words, despite of its great functionality, the lack of a GUI, allowing visual adjustment and programming of the existing and of user defined *VTK*-classes and the limited choice of classes for viewing data are the main disadvantages of the toolkit.

The authors' vision of the perfect visualization environment is a freely available collection of powerful visualization classes, combined with an easy to use and programme graphical interface to these classes. Such a toolkit will be used to test and embed new algorithms and data structures from distinct areas of the computer science. Moreover, if it is transparent enough, it will become a standard visualization tool even for users, who are not experienced in particular details of a visualization process and do not want to become experienced in it. This in turn will allow to concentrate on the development of new *techniques* and not of new environments, in order to solve given problems. Furthermore, such common platform will allow to readily compare existing with new algorithms utilizing common data structures, hence creating an excellent reference base for new approaches.

3 Concept and Realization

To overcome the restrictions described in the introduction and in order to make *VTK* more attractive through providing an operating systems independent GUI, without affecting the functionality and portability, we developed an extension of the *VTK*-library (*QteVtk*). Based on *Qt* [Dalhe99] – a multi-platform C++ graphical user interface toolkit – the presented work provides a flexible and portable way for graphically displaying *VTK*-object characteristics. Furthermore, the developed library is easy to use and to extend, since *VTK* and *Qt*, thus *QteVtk*, are object oriented tools (object oriented visualization concepts are discussed in [Favre94]). This applies to the additional classes we developed for viewing *VTK*-data structures (discussed in Section 3.3) and visual editing (Section 3.4) of parameters, hence allowing an easy and intuitive insight into different data types.

3.1 Why *Qt*

For our implementation we chose the *Qt*-library, because it is a multi platform, object oriented package, supporting the graphics standard *OpenGL* and being very efficient and intuitive for creating applications with a graphical user interface. The *signal/slot* mechanism, provided by this package, proved to be a vary useful feature for elegantly connecting created interface components with each other or with functions, activated by with GUI-elements (similar to “callback-functions”). Furthermore, *Qt* [Dalhe99] is freely available for non-commercial use on any *Unix*/*Linux* systems and is even commercially

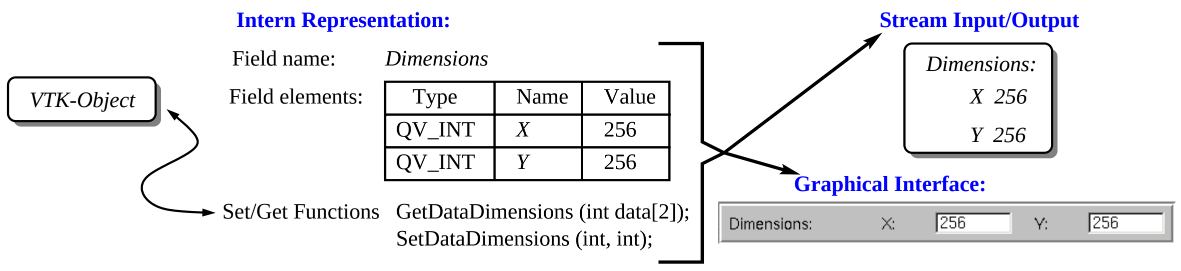


Figure 1: Intern representation, stream input/output, and graphical appearance of a simple field.

available for Windows 9x/NT.

3.2 The *QteVtk*-library

In order to simplify the access to the *VTK*-object's parameters and to provide a graphical appearance for them, we implemented a set of C++-classes, designed to contain arbitrary *VTK*-objects. Moreover, these classes allow the defined parameters to be written to and read from a data stream, to connect pairs of objects to create a visualization pipeline, and to control it's data flow and update.

3.2.1 Fields

We accomplished this concept through introducing *fields* to our library. A *field* is an object, which contains the name of an arbitrary parameter, it's type and value. In addition, a *field* implements the graphical appearance of the stored parameters and disposes of functions for setting/getting parameters' values in/from the object it is connected with. Hereby, every field is connected with exactly two functions for getting respectively setting its values from/in the connected *VTK*-object (as shown in Figure 1). Another feature of a *field* is its ability to write and read its values to and from a data stream, which makes the *field* concept very powerful and allows its utilization even in a scripting visualization language.

Next, we implemented field classes, derived from the *QVBaseField*-class for every parameter needed for the description of the associated *VTK*-object (as depicted in Figure 2). Any of these classes can use the functionality of the base class, as well as add auxiliary features (e.g. the file selection button for file browsing in Figure 3). Through this philosophy we defined a set of classes, which behave similar and can be used in a similar way.

3.2.2 Field Container

To cover all the parameters of a *VTK*-object, it's *fields* have to be pooled together in a single object. Thus, we introduced a *QVFieldContainer*-class, containing any number of fields necessary for the complete description of the *VTK*-object's parameters one is interested in. In addition, a container object includes the following items:

- definition of the data input (predecessor objects in the visualization pipeline);
- definition of the *update*-function, utilized to update the object's input(s): its predecessors in the visualization pipeline (depicted in Figures 2 and 3);
- functions for reading and parsing the values of its field elements when read from a stream;
- functions for invoking the associated fields to write their name and parameter values to a stream.

The *output* of an object is not listed here, because only the output *type* of an object $\mathcal{A}$ is important for an object $\mathcal{B}$, which receives its data from $\mathcal{A}$. Furthermore, $\mathcal{A}$ does not know the number of objects with connected input to $\mathcal{A}$'s output.

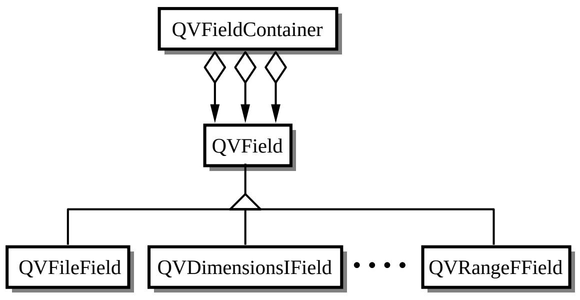


Figure 2: The class hierarchy of the *QteVtk*-library.

When added to a field container, the single fields, derived from the class *QVField*, can be displayed in a dialog allowing graphical adjustment and manipulation of the *VTK*-class' parameters (see Figure 3). Moreover, the *VTK*-object functions introduced above are displayed and activated. Such a field container is also able to visualize the input of the associated *VTK*-object, allowing to track the data flow in the visualization pipeline. In the normal case, the input of an object consists of only one predecessor, however, there are *VTK*-objects providing more than one slots for data input, which has to be declared when the object is defined. Field containers also allow for the embedded object to support input slots for different data types. For instance, an image filter can accept as input structured grids, as well as image data, which also has to be defined in the object's constructor.

The appearance of a field container can also be adapted to the user preferences. If desired, it can be declared to not appear as a dialog box (as depicted in the right of Figure 3), but to be permanently accessible as a part of a particular application's window, as shown in the example application in Section 4. Furthermore, the field container, due to its derivation from the *QWidget*-class, can have an arbitrary parent window, thus can appear in arbitrary other windows. This imparts the field container additional flexibility, when utilized in a particular application.

3.2.3 *QteVtk*

In this way we designed and developed a library on the top of *VTK*, which can be utilized to create visualization applications with a graphical user interface (an example is shown in Section 4). In order to create a visualization pipeline in *VTK*-manner, the programmer has to create for every *VTK*-object a field container, pass a pointer to this *VTK*-object, define the number of inputs and the corresponding functions, and the type of the object's output. These are the functions controlling the data flow and update in a visualization pipeline. Afterwards, an arbitrary number of fields can be added to the field container, depending on the number of parameters one intends to adjust and display (as depicted in Figure 3). When this is completed, either a *Qt*-based application can be implemented and the data processing can be adjusted through point-and-click mechanism, or the normal *VTK*-visualization can be applied. This allows even to create applications, which can be used with, as well as without a graphical user interface, without changing major parts of the application's code. Such a feature is very useful, when a particular data processing pipeline is applied to different data sets (e.g. for noise reduction after data acquisition in a medical context), which require unique interactive parameter adjustment, followed by multiple applying of the defined transformations and their ascertained parameters on many other data sets.

3.3 Data *Viewers*

One of the most important issues, defining the acceptance and attractiveness of a visualization library and, of course, of the applications utilizing it, is the way the data is visualized. Therefore, the next major task is to develop and implement a set of objects for viewing data (*viewers*), in order to discover some invisible or hidden data features. Due to the fact, that *VTK* includes only a simple image viewer and a renderer class[2], caused by the aim to maintain the toolkit's portability, the users do not have a wide choice of viewers for gaining an insight into the processed data. Through its multi platform design, *Qt* helps to overcome this obstacle. It allows to develop *viewer* objects for the different data types available in the *VTK*-library, while keeping the basic features of the toolkit like portability and object-oriented design unaffected.

The most basic viewer we implemented is an image data viewer (*QVvtkImageViewer*), allowing to interactively browse through the slices of a volume data set, zoom in/out selected image regions and view one or all channels of a multi channel data set.

The slice viewer *QVvtkSliceViewer* allows an intuitive viewing of volume data, while providing OpenGL [Woo98] based displaying of a data-cube and three cutting planes orthogonal to the x,y,z-axes respectively. Hereby, on every plane we mapped a semi transparent texture with the appropriate image extracted from the volume data (see Figure 4).

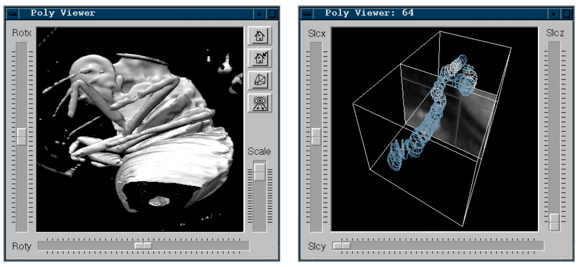
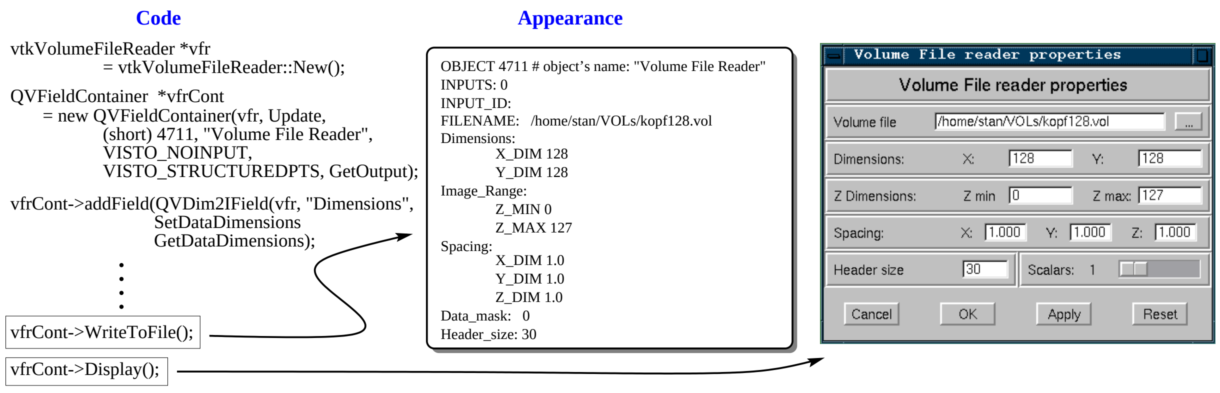


Figure 4: The *Poly* viewer (on the left) and the *Slice* viewer (on the right).

The *QVvtkPolyViewer* is an *Inventor* [Strau93]-like viewer, implementing most of the *SceneViewer*'s functionality in a *Qt/OpenGL*-based window, while significantly extending the functionality of the standard *VTK*-renderer.

[2]These classes display structured points, point sets, structured grids, and polygonal data in a very simple way.



```
vtkVolumeFileReader *vfr
            = vtkVolumeFileReader::New();

QVFieldContainer  *vfrCont
      = new QVFieldContainer(vfr, Update,
            (short) 4711, "Volume File Reader",
            VISTO_NOINPUT,
            VISTO_STRUCTUREDPTS, GetOutput);

vfrCont->addField(QVDim2IField(vfr, "Dimensions",
                  SetDataDimensions
                  GetDataDimensions);
      .
      .
      .
vfrCont->WriteToFile();
vfrCont->Display();
```

```
OBJECT 4711 # object's name: "Volume File Reader"
INPUTS: 0
INPUT_ID:
FILENAME:   /home/stan/VOLs/kopf128.vol
Dimensions:
          X_DIM 128
          Y_DIM 128
Image_Range:
          Z_MIN 0
          Z_MAX 127
Spacing:
          X_DIM 1.0
          Y_DIM 1.0
          Z_DIM 1.0
Data_mask:   0
Header_size: 30
```

Volume File reader properties

Volume file	/home/stan/VOLs/kopf128.vol				
Dimensions:	X:	128	Y:	128	
Z Dimensions:	Z min	0	Z max:	127	
Spacing:	X: 1.000	Y: 1.000	Z: 1.000		
Header size	30	Scalars:	1		

Figure 3: Appearance and functionality of the field classes.

These two viewers are conceived and realized similar to the *QVvtkImageViewer* presented next and are depicted in Figure 4.

3.3.1 The Image Viewer

The *QVvtkImageViewer* is derived from the *vtkReferenceCount*-class, since it is a special *VTK*-object and from the *QWidget*-class on the other hand to facilitate graphical appearance (see Figure 5). Furthermore, like any other *VTK*-

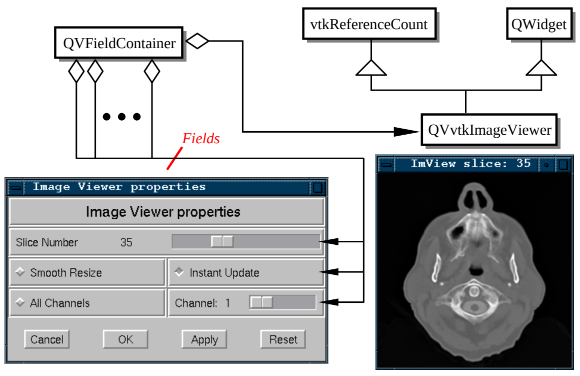


Figure 5: A viewer for displaying images and browsing through the slices of a volume data set and its parameter fields.

object, the *QVvtkImageViewer* is associated with a field container, hence it has its own parameter appearance. These parameters, however, can be immediately applied to the object, which is necessary, for instance for the real-time "browsing through the slices" feature (see Figure 5) or applying the camera transformations, when a scene is viewed. For the realization of this feature, we applied the *signal/slot*-mechanism provided by the *Qt*-library [Dalhe99], which allows to connect two arbitrary objects together in order to notify the first, when the value of the second is changed.

3.4 *Editors* for *VTK*-Data

When processing real-world data, another set of classes turned out to be of great importance. This was the motivation for us to develop a second set of auxiliary objects: the *editor*-object set. The editor class hierarchy is designed similar to this of the viewer-objects. The difference is, that the editor objects are not only able to display data, hence they are not pipeline terminating objects, but have a data output. This illustrates the two possible applications of these objects: as source objects at the beginning of a visualization pipeline, or as helper objects for particular algorithms, when additional user interaction is needful. In fact, the editor classes are derived from the viewer classes, thus providing their features and the additional ability to generate outputs like points, 2D polygons, color values picked in a color editor or displayed image, and simple 3D primitives like spheres and cubes. Moreover, there are more than one editors for a specific input type. For instance, there are two line editors, derived from the image and the slice viewer. In both cases, the output has the same type, however, different presentation and interaction with the data is applied. The line editor, for example, is utilized in the intelligent scissors algorithm [Morte95], where mouse interaction in the plane is used to define the current lines. On the other hand, when one is defining 3D polygons, the slice viewer based editor is the more appropriate and intuitive one. Both, viewer and editor classes are grouped together in the auxiliary library *QteVtkAux*.

4 An Example Application

In this section, the introduced libraries *QteVtk* and *QteVtkAux* are utilized to create a very simple sample application (depicted in Figure 6). On the top right of Figure 6, the general architecture of an application, based on *QteVtk* and *QteVtkAux* is shown, which holds for the sample application as well.

The visualization pipeline in our example consists of four objects: a data reader *VolumeFileReader* responsible for the data input, a region growth algorithm *RegionGrow*, which takes a volume data set and a set of seed-points as input, and two image viewers (see the *data flow diagram* in Figure 6). A fifth object is tentatively created for editing seed points for the region growth algorithm (Figure 6 *edit points* button). The first *ImageViewer* is utilized for viewing the data before applying the algorithm (its interface is visible in the middle left row), the second views the result of the data processing algorithm (its interface is turned off in the application).

For each of these objects the parameters we are interested in and the functions for setting their values in the associated *VTK*-object, are utilized to define the fields (as discussed in Section 3.2), which are able to display themselves in the application's interface. In the next step, the input of every object is connected with the object it gets the data from (through the field-container), like this has to be done in a regular *VTK* visualization application.

The GUI-frame, including the file-menu and its items, is also part of the library we developed. It allows to show and hide the properties of any created object. The data-flow control can be permanently turned on in this application (menubar *"Options"*) to enable automatic update of the viewed data (when necessary), since the visualization pipeline and the loaded data are rather small in the example. Another way to control the data-flow is to activate the pop-up menu, associated with each object, which appears on each object's label and activates the declared object's *update* function (as depicted in Figure 6). This function causes a backward check for modifications of the object's predecessors in the pipeline and their update if necessary. Data flow concepts in visualization applications are discussed in [Dyer90] and [Schro98].

5 Work in Progress

The libraries we introduced above are in this form ready to use, however, work on several important issues is still in progress. Currently new viewer and editor classes including a volume-previewer for real time displaying of a scaled down volume data set, a viewer, combining volume and surface rendering for more powerful data viewing, and a color/color-map editor are in development.

In order to simplify the manual binding of native *VTK*-classes with *QVFieldContainer*s we are developing a tool for the automatization of this task. This will in turn allow the simple generation of fields and field container for new *VTK* classes. Another valuable feature will be the *composed object* concept, allowing for an object to contain several other objects connected in a pipeline. This helps to hide dispensable complexity from the user, thus simplifying the interface.

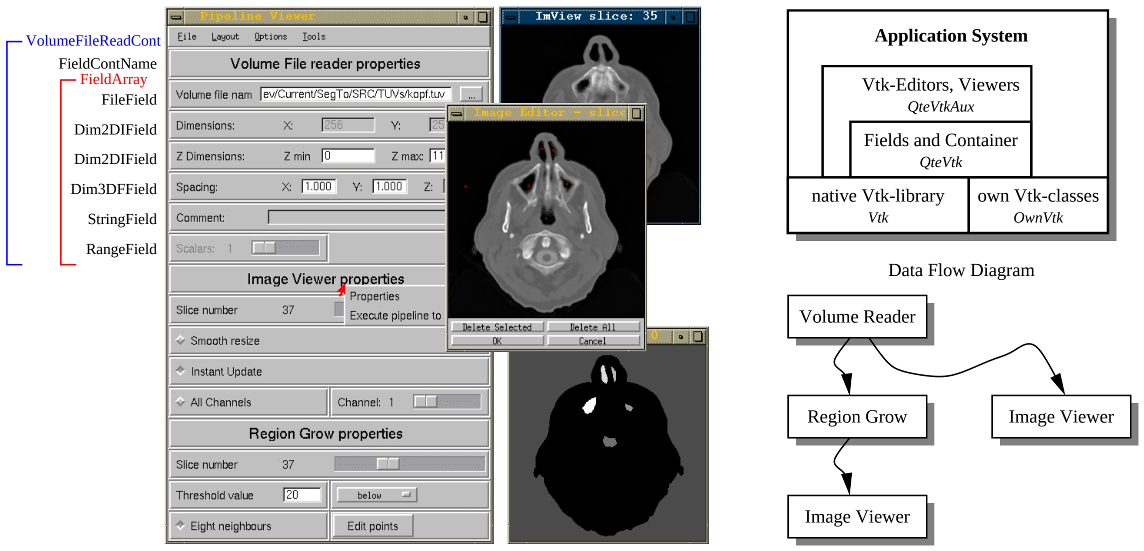
6 Summary

In this paper, we present an extension of the visualization toolkit *VTK*, which is not just "another visualization library", but provides a portable object oriented, graphical user interface to *VTK*, hence imparting reusable design to existing and newly developed classes.

Furthermore, we introduced a valuable set of visualization classes for viewing and editing different *VTK*-data types. By utilizing the GUI-toolkit *Qt* for the realization, we avoid affecting the portability, while maintaining the object oriented concept of *VTK*. This overcomes the two main drawbacks of the toolkit: the lack of a graphical user interface and the lack of classes for more vivid data viewing.

The libraries *QteVtk* and *QteVtkAux*, presented here are an excellent companion for the development of *VTK*-based applications (as shown in Section 4). They provide an attractive development environment for testing and embedding of new algorithms and data visualization techniques, and offers an excellent reference base for benchmarks with the many ones available in the *VTK*-library. Therefore, addressing both people utilising *VTK* as a visualization library, as well as developers of new techniques. Concerning the criteria introduced in Table 1, the presented libraries improve the usability of *VTK* in terms of graphical appearance and pipeline parameter adjustment, while maintaining *VTK*'s features like *functionality*, *availability*, *portability*, and *design*.

In our opinion, the visualization nowadays lacks an easy to use non-commercial system. This will, in turn, encourage not only researchers from the computer graphics area to programme their own visualization techniques with the presented environment, but also users not necessarily



Volume file nam ev/Current/SegTo/SRC/TUVs/kopf.tuv
Dimensions: X: 256 Y: 25
Z Dimensions: Z min 0 Z max: 11
Spacing: X: 1.000 Y: 1.000 Z:
Comment:
Scalars: 1
Slice number 37
Smooth resize
Instant Update
All Channels
Channel: 1
Slice number 37
Threshold value 20 below
Eight neighbours
Edit points

Figure 6: An application created with the introduced libraries *QteVtk* and *QteVtkAux*.

knowledgeable of computer graphics and visualization to use one common free environment. The presented work provides a solution to this problem. The most prominent example for the success of such systems is *Linux*, which is freely distributed, thus it has a vary large user community utilizing and further developing it.

Unfortunately, despite its great functionality, *VTK* is a complex toolkit, which is not suitable for visualizations with a portable graphical user interface, allowing on-the-fly adjustment of pipeline parameters. This, however, defines to a great degree the acceptance and popularity of a visualization environment. With *QteVtk* and the presented additional libraries, we claim to provide a valuable visualization companion with a simple (graphical) interface to the features of *VTK*, imparting additional attractiveness to the *VTK*-class hierarchy, while maintaining its great functionality and object oriented design.

REFERENCES

[Dalhe99] Matthias K. Dalheimer. *Programming with Qt*. O'REILLY, B., first edition, 1999.

[Dat91] *Data Explorer Reference Manual*. IBM Corp., Armonk, NY, USA, 1991.

[Dyer90] D. Scott Dyer. A dataflow toolkit for visualization. *IEEE Computer Graphics and Applications*, 10(4):60–69, July 1990.

[Favre94] J. M. Favre and J. Hahn. An object oriented design for the visualization of multi-variate data objects. In *Proceedings. Visualization '94. Los Alamitos, CA*, pages 319–325. IEEE Computer Society Press, 1994.

[Iri91] *Iris Explorer User's Guide*. Silicon Graphics Inc., Mountain View, CA, USA, 1991.

[Konst94] K. Konstantinides and J. R. Rasure. The Khoros software development environment for image and signal processing. *Ieee Transactions on Image Processing*, 3(3):243–52, 1994.

[Morte95] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 191–198. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[Rasur91] Rasure and Williams. An integrated visual language and software development environment. *J. Visual Languages and Computing*, 2:217–246, 1991.

[Schro96] W. J. Schroeder, K. M. Martin, and W. E. Lorensen. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. In *Proceedings. Visualization '96. San Francisco, CA, USA. 27 October–1 November 1996*, pages 93–100, New York, NY 10036, USA, 1996. ACM Press.

[Schro98] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. *The Visualization Toolkit*. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, second edition, 1998.

[Strau93] Paul S. Strauss. IRIS inventor, A 3D graphics toolkit. In Andreas Paepcke, editor, *Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 192–200, Washington, DC, USA, September 26October –1 1993. ACM Press.

[Upson89] Craig Upson, Thomas A. Faulhaber, Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The Application Visualization System: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[Woo98] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison-Wesley, Reading, Massachusetts, U.S.A., 1998.