matVTK - 3D Visualization for Matlab

Erich Birngruber[1,4], René Donner[1,2], and Georg Langs[1,3]

[1] Computational Image Analysis and Radiology Lab, Department of Radiology, Medical University of Vienna, Vienna, Austria, erich.birngruber@meduniwien.ac.at,
[2] Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria,
[3] Computer Science and Artificial Intelligence Laboratory, Massachussetts Institute of Technology, Cambridge, MA, USA
[4] Pattern Recognition and Image Processing Group, Vienna University of Technology, Vienna, Austria. *

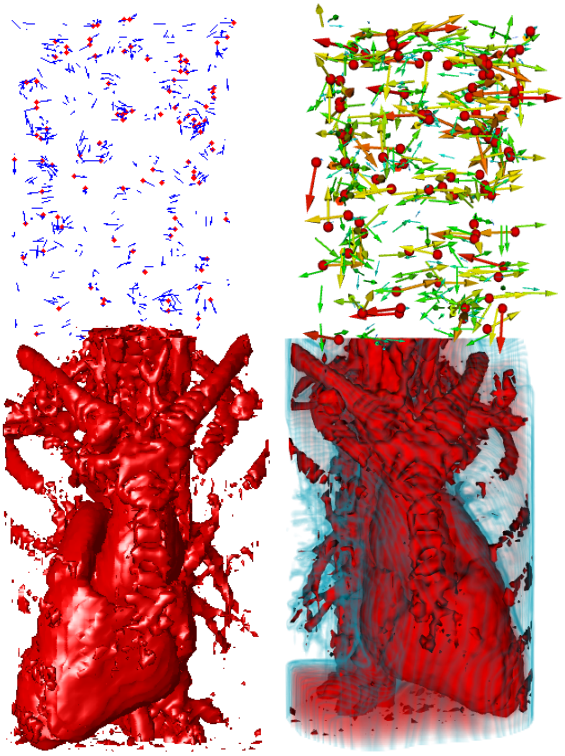
Abstract. The rapid and flexible visualization of large amounts of complex data has become a crucial part in medical image analysis. In recent years the Visualization Toolkit (VTK) has evolved as the de-facto standard for open-source medical data visualization. It features a clean design based on a data flow paradigm, which the existing wrappers for VTK (Python, Tcl/Tk, Simulink) closely follow. This allows to elegantly model many types of algorithms, but presents a steep learning curve for beginners. In contrast to existing approaches we propose a framework for accessing VTK's capabilities from within Matlab, using a syntax which closely follows Matlab's graphics primitives. While providing users with the advanced, fast 3D visualization capabilities Matlab is still lacking, it is easy to learn while being flexible enough to allow for complex plots, large amounts of data and combinations of visualizations. The proposed framework will be made available as open source with detailed documentation and example data sets.

1 Introduction

Fueled by the increased interest in medical imaging research in recent years the importance of powerful[del: high quality] frameworks covering feature computation, registration and segmentation as well as visualization have risen considerably. Researchers depend on efficient ways to formulate and evaluate their algorithms, preferably in environments facilitating rapid application development.

The Visualization Toolkit (VTK) [1] has proven to deliver high quality, efficient visualization for a variety of fields including general purpose GUIs [2] as

* This work has been supported by the Austrian National Bank Fond project Computer Based Quantification of Osteoporosis and Bone Alignment, MU Vienna, TU Graz.



(a) (b)

Fig. 1. (a) Matlab 3D graphics primitives (b) Scene produced by the proposed framework, showing additional support for volume rendering

well as medical imaging [3, 4]. Similarly, the Insight Toolkit (ITK) [5] has provided a framework for medical image registration and segmentation. It features a data-flow driven design paradigm which models algorithms as filters which are concatenated and transform source (input) data into results (output). It is implemented in C++, providing high throughput, and provides wrappers for selected scripting languages. Recently, the project VTK Edge [6] is actively investigating the use of GPU acceleration for complex visualization tasks which cannot be modeled in OpenGL.

While VTK's data-flow approach in combination with C++ is very flexible its utilization imposes a steep learning curve on the user. Furthermore, in medical imaging and computer vision Matlab is one of the major development platforms, especially in the academic field. Matlab itself provides unique capabilities for rapid application development, but severely lacks state of the art 3D visualization features [7]. There are no volume rendering methods in Matlab, and basic operations like isosurfaces are very slow. Thus, for visualizing medical data along

with meta-data like segmentation results, classification probabilities or the like external toolboxes have to be used. Recently, a Simulink based approach was proposed which automatically wraps VTK's functionality in Simulink blocks. While this allows to graphically structure a VTK plot in Matlab, it still requires the user to get familiar with the internal concepts of VTK. The user's knowledge of Matlab's built-in plot commands is not exploited. For ITK there exists a wrapper, matITK [8], which forgoes ITK's complexity by providing simple Matlab MEX commands for the most commonly used functionality. This gives Matlab users an effective tool for performing most operations directly from within Matlab without having to worry about data conversion, data-flow formulations and result structures.

Contribution Because of VTK's relevance to the computer vision community using Matlab, we propose a framework to model VTK's functionality in a way which is similar to Matlab's graphics concept, minimizing learning efforts by the user while providing VTK functionality in a flexible manner. The proposed solution is available as open source and we are working towards making it available as an additional wrapper distributed with future VTK versions. matVTK provides a rapid integration of VTK functionality in Matlab. It is a necessity for the analysis of large and complex data, and the visualization of algorithm outputs, in the medical image analysis domain.

The paper is structured as follows: In Sec. 2.1 we outline the data-flow paradigm employed by VTK along with its internal properties which are relevant for our approach. Sec. 2.2 discusses how Matlab's MEX interfacing concept can be used to interact with external libraries. In Sec. 3 we detail our approach with the presented in Sec. 4, followed by conclusion and outlook in Sec. 5.

2 Foundation & Methods

In the following the technical basis for our approach is outlined: VTK's internal design and the MEX interface provided by Matlab. Based on these components we will explain the matVTK framework in detail in Sec. 3.

2.1 VTK

VTK is a visualization library written in C++. VTK can be used to draw geometric primitives such as lines and polygon surfaces as well as render volume visualizations. Furthermore it allows fine grained control of the combination of these primitives in a scene.

The first thing noticeable to the programming end user is the data-flow based "Pipes and Filters" design pattern, used to concatenate various data processing methods. To be able to combine different and multiple filters and preprocessing steps all these algorithms share a common interface, providing their fundamental functions `SetInput(in)` and `GetOutput()`. The `SetInput` method accepts the input data, the `GetOuput` method provides the processed result. The return value of GetOutput can then again be used as input to another algorithm. At the front

of such a filter chain there is always a reader or general source that provides the initial data. The last link in the chain is a sink that either renders the output on the screen or saves it into a file. However, this concept is more sophisticated than simply handing over the whole dataset from filter to filter. On the contrary, the filter pipe helps to compute multiple steps at once and time stamps in the pipe allow to only recompute the parts of the pipeline affected by changes in source code or paramters. This allows for an economical memory footprint and fast visualization even if the underlying data changes during scene rendering.

Reference Counting In C++, to dynamically allocate an object, the `new` operator is used to allocate memory and call the contructor.while the `delete` operator calls the destructor and frees the memory. This approach should not be used directly in VTK. A base class for VTK classes provides two important methods: the static method `New` and the instance method `Delete`. The first one returns a pointer to a new instance of a class while reference counting is initialized simultaneously. If the instance is used in a setter method of another VTK object, the reference count is automatically increased, which avoids the duplication of the object in memory protects against premature deallocation and memory leaks. When calling the `Delete` method on such an object, the reference counter is decreased . When no more references to the object exist, the C++ `delete` operator is called automatically

Factory Pattern Another important concept in the VTK library is the ubiquitous use of the factory pattern, especially for the actual rendering classes. It allows to instantiate an object of an unknown specific subclass. The typical example for VTK is the rendering implementation. Assuming the task is to draw polygons, this can be done in two ways: either in software, or using graphics hardware and OpenGL. When calling `PolygonRenderer::New()`, VTK automatically checks for the underlying hardware and returns an instance of the appropriate subclass (in this case either `SoftwarePolygonRenderer` or `OpenGLPolygonRenderer`). Although VTK provides a clean and well performing code base, the first time or casual user may be overwhelmed with the software design concept and the complexity of the large API.

2.2 Matlab MEX Interface

Matlab can be extended using the Mex API, which is the name of the C/C++ and Fortran API for Matlab. The API can be used to manipulate workspace data, the workspace itself or use the Matlab engine in external applications.

The following focuses on the properties relevant to implementing a framework towhich integrates VTK into Matlab. When thinking about a Matlab framework and plotting / rendering, two important properties come to mind: First memory management should exhibit a small footprint – however the Matlab principle of non-mutable input data must not be violated. Secondly, we need to maintain an internal state over several function calls, which is accomplished using a handle based approach.

Matlab comes with its own memory management component and therefore its own implementations of `alloc()` and `free()` functions. This way the software

can ensure to release allocated memory even in the case of an error during a function call, when the program flow does not reach calls to `free()`. This means that any handle implementation must respect these cleanup calls in order to destruct/deallocate correctly in case of such an event.

As the Mex API does not provide functions to implement handles returned to the Matlab workspace [9] was used which provides a C++ template implementation for returning an object pointer back to the Matlab workspace. Furthermore the code includes checks to verify that a correct handle was passed before using the pointer.

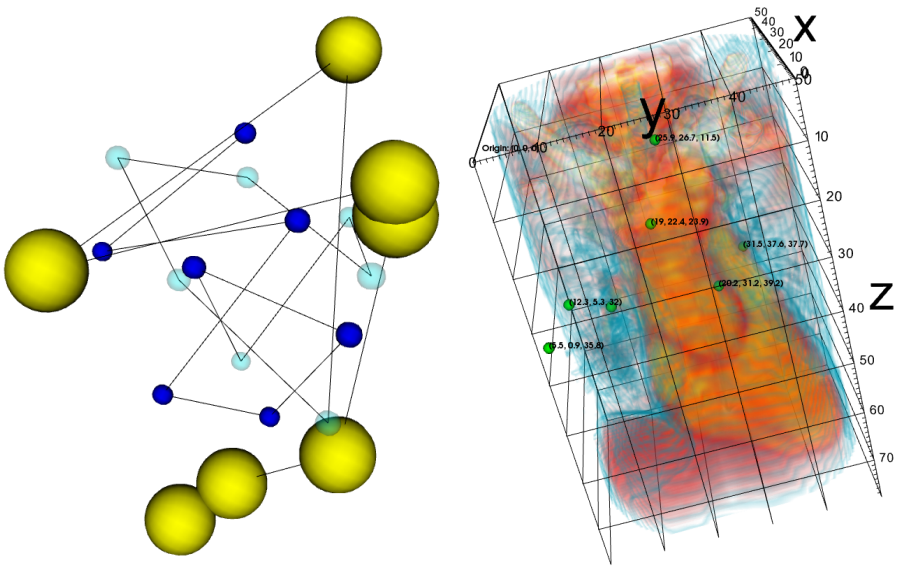
A particular problem arises on Unix platforms using the X11 window system, as Mex functions are intended to be used as single threaded and blocking calls. Because the API does not allow for interaction with the window management, this causes two problems: The Matlab GUI is blocked during the use of a user interactive window. Even more important, secondly, closing the window via the GUI shuts down the complete Matlab session. This does not occur on windows platforms. We are uncertain whether a solution for this problem exists, as in depth research showed this to be a problem deeply rooted in internal X11 design.

3 matVTK Framework

In this section, based on the above overview of the underlying environment, the proposed approach of providing a visualization API which closely follows Matlab's plotting concepts is detailed. The three main building blocks are 1. The *pipeline handle* which establishes the rendering window, 2. the *config interface* that manages the parameters of the individual plots, and 3. the *graphics primitives* that provide for the actual plotting functionality analogously to standard matlab commands.

The Pipeline Handle The VTK handle forms the core of the framework. Its main purpose is to keep track of the data sets currently used in a scene. The second most important task is the management of the render window that is used to interactively display a scene. Additionally, it controls global components that cannot be kept in a single function or that are relevant for multiple functions. The handle is either generated automatically, or can be set by `handle = vtkinit()`. Itcan automatically delete itself and all its dependencies in case Matlab is closed or all Mex memory is freed using `clear mex`. Handles can be used implicitly, or explicitly to be able to work with multiple scenes. On each of the various function calls, if no explicit handle is given, it is checked whether there exists a default handle or not. In the latter case it is automatically created and reused in subsequent calls. When using an explicit handle it must be created and destroyed by the user. Furthermore it must be handed to each plot function as the first argument.

The Config Interface VTK offers fine grained control of parameters for basically three steps: filters, scene components (actors) and the global scene (i.e. camera settings). For this reason a large amount of setters and getters exist in various places of VTK's class hierarchy. For the uninitiated VTK user this is

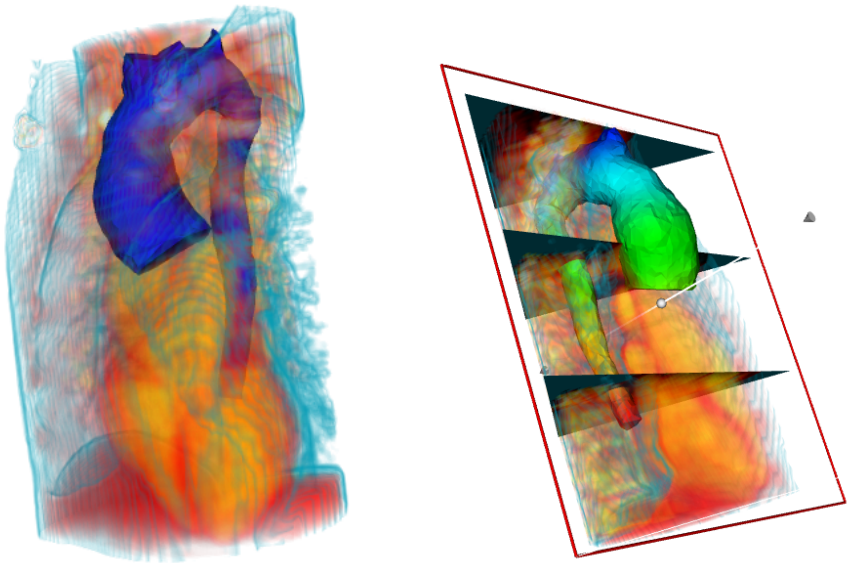


(a) (b)

Fig. 2. (a) Points showing different colors, sizes and opacities, connected with line plot (b) Labels and annotations, volume rendering

complex and opaque. This is why we decided to implement a flexible config interface that can model the complex VTK design. The first, and implementation wise simple approach is using a Matlab struct. It is a data structure that consists of named fields and values (e.g., `config.opacity = 0.3`). Constant config parameters can be easily reused by the user. *[del: However, changing parameters have to be reset before each consecutive function call.]* Alternatively the typical Matlab approach*[del: way]* for the configuration of parameters - a list of string-value pairs - is also supported, as well as a combination of the two. When using both, the Matlab style can be used to override settings in the config struct.

Graphics Primitives Matlab uses the `plot` function for different kinds of primitives - lines and points. For our framework this did not seam feasible, which is why we decided to use the prefix `vtkplot` and a specific suffix for points, lines etc. Currently the framework supports primitives for points (represented as small spheres), lines, polygon meshes, vector fields, tensors, volumes, cutplanes through volumes and isosurfaces.Also provided are functions gaining a better overview of the scene, such as functions for labels, legends and plot titles. These primitives can be easily combined to create complex scenes as we will show in the following section.



(a) (b)

Fig. 3. (a) Mesh combined with volume (b) Same data, additionally showing color mapping and planes cut through volume

Additional Functionality As the simple combination of graphical primitives does not alway meet the users needs, there are special features available including scene export. Screenshots of a scene can be created and saved as PNG images, including the support for high resolution images for printing. The `vtkshow` function, used to display the scene window, returns the camera settings at the moment of closing the window, which can then be reused as config parameters to restore or reproduce global scene settings over multiple plots. Another valuable feature is the ability to cut away certain parts of the scene. For this operation a box widget – i.e. six perpendicular planes – or a single plane widget are available. The cropping operation can be applied to scene primitives of the users choosing and therefore provide the best possible insight into the displayed data.

4 Scenarios

In the following we demonstrate*[del: show several ways of employing]* the proposed framework and the plot types available with 3 examples. All types of plots can be arbitrarily combined within one volume rendering.

– Fig. 1 shows the visualizations produced by Matlab in comparison with our approach for similar data. Matlab lacks volume rendering capabilities and

while being informative, the results lack publication quality. In contrast, using VTK provides not only state of the art volume rendering but also detailed control over the 3D rendering of the graphical primitives. Also note that rotating the Matlab figure gets slow fast as the number of actors in the scene increase, inhibiting efficient use except for simple cases.

- In Fig. 2 (a) the ability do easily render primitives with different properties is shown. The level of detail for rendering spheres / tubes can be chosen to allow for either exact representation or coarser approximation which lets the user plot up to several thousand 3D points at interactive frame rates.
- Fig. 2 (b) resents the labeling available for individual points in space as well as for the axes. Grids with arbitrary spacing as well as orientation widgets (box, arrows) are available.
- The superposition of meshes onto the volume is shown in Fig. 3 (a). Rendering this view from the Matlab command line takes below one second and the resulting view can be rotated and cropped interactively even on medium class hardware.
- Finally, Fig. 3 (b) shows the possibility to map scalar values onto the surface of a mesh and display additional views of the volume along cut planes, while cropping the whole volume (with or without the other actors). The cut planes / the crop box can be set both programmatically as well as interactively.

```
v = vtkplotvolume(volume, 'builtin1');
vtkplotmesh(vertices, faces, vertexLabel)
p = vtkplotcutplanes(volume, planePoints, planeVecs);
vtkcrop(v, 'plane')
vtkcrop(p)
vtkshow('backgroundColor', [1 1 1])
vtkdestroy
```

Fig. 4. matVTK code example for Fig. 3 (b)

Fig. 4 shows the Matlab code to create a plot using an implicit handle. First the user plots a 3D matrix as volume, using the builtin colormap "builtin1". Next a polygon surface is plotted using its vertex coordinates, triangulation. The labels in `vertexLabel` are used to colorize the surface. The function `vtkplotcutplanes` creates several planes at the given points, using the normal vectors `planeVecs`. `vtkcrop` uses the handles returned from the plotting functions, to decide whitch parts of the scene can be clipped away with the user interactive widget. After displaying the assembled scene with `vtkshow` the resources are freed with `vtkdestroy`. The call to `vtkshow` also demonstrates the config interface, setting 'backgroundColor' to the rgb value of white.

5 Conclusion and Outlook

We have proposed an approach to wrap VTK's main capabilities using an easy to use, efficient framework for the Matlab programming environment. It provides Matlab users with state of the art 3D volume rendering and visualization features while retaining Matlab's ease of use. Even complex medical visualizations can be assembled in few lines of code, without knowledge about VTK internal data-flow paradigm. The framework exposes VTK's most relevant features while being easily extendable.

Future work will focus on two areas, namely the inclusion of additional visualization and interaction features and to improve the internal structure of our framework. While user interaction in the VTK window with the output being forwarded to Matlab is definitely possible and is already being used in several cases, it is not yet fully covered by the framework. Saving widget states to return to a previous visualization will be added, as well as animations and movie export functionality. Handles, which are currently used only internally, will be exposed to the user to be able to selectively remove parts of the scene or control their visibility. Streaming visualization output via state of the art codes is another topic which is currently investigated.

References

1. (VTK), V.T.: http://www.vtk.org/
2. ParaView: http://www.paraview.org/
3. MedINRIA: http://www-sop.inria.fr/asclepios/software/medinria/
4. Osirix: http://www.osirix-viewer.com/
5. Segmentation, I., (ITK), R.T.: http://www.itk.org/
6. VTKedge: http://www.vtkedge.org/
7. Matlab 3D Visualization Capabilities: http://www.mathworks.com/access/helpdesk/help/techdoc/visualize/bqliccy.html
8. MatITK: http://www.sfu.ca/ vwchu/matitk.html
9. Bailey, T.: http://www-personal.acfr.usyd.edu.au/tbailey/software/other.htm