**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor’s Note</td>
<td>1</td>
</tr>
<tr>
<td>Recent Releases</td>
<td>1</td>
</tr>
<tr>
<td>Visual Debugging of ITK</td>
<td>3</td>
</tr>
<tr>
<td>The CMaking of a Humanoid</td>
<td>7</td>
</tr>
<tr>
<td>Paraview in Aerodynamics</td>
<td>9</td>
</tr>
<tr>
<td>Kitware on Apple OS X: It Just Works, in MacPorts</td>
<td>11</td>
</tr>
<tr>
<td>MRCAD for Daily Clinical Analysis of Prostate MR</td>
<td>14</td>
</tr>
<tr>
<td>MATLAB® and GNU R Integration with VTK</td>
<td>16</td>
</tr>
<tr>
<td>In Progress</td>
<td>17</td>
</tr>
<tr>
<td>Kitware News</td>
<td>18</td>
</tr>
</tbody>
</table>

**EDITOR’S NOTE**

The Kitware Source contains articles related to the development of Kitware projects in addition to a myriad of software updates, news and other content relevant to the open source community. In this issue, Matt McCormick provides a tutorial for using Python pretty-printers to debug ITK. Giorgio Metta, Paul Fitzpatrick and Lorenzo Natale explain how the use of CMake improved their development of a humanoid robot. Jean Favre and Jan Vos show how the Navier Stokes Multi Block solver plugin for ParaView supports a wide range of data analysis tasks. Darren Webber provides a tutorial for VTK and ITK MacPorts to get Kitware tools up and running on Darwin-OSX systems. Henjan Huisman and Pieter Vos share their experience using VTK, ITK and IncrTCL to achieve a robust, versatile, clinical research platform for prostate MR. And Tom Otahal provides a follow-up article along with the locations of integration classes for his October 2009 article on the integration of MATLAB and GNU R with VTK.

The Source is part of a suite of products and services offered to assist developers in getting the most out of our open source tools. Project websites include links to free resources such as: mailing lists, documentation, FAQs and Wikis. Kitware supports its open source projects with textbooks, consulting services, support contracts and training courses. For more information, please visit our website at www.kitware.com.

**PARAVIEW 3.8**

Kitware, Sandia National Laboratories and Los Alamos National Laboratory are proud to announce the release of ParaView 3.8. The binaries and sources are available for download from the ParaView website. This release includes several performance improvements, bug fixes for users, and plenty of new features for plugin and application developers. We have made it easier to locate cells/points in your dataset using queries. Search the ParaView Wiki for “Find Data using Queries” for more information.

The plugin loading and management dialog was redesigned to make it easier to load plugins. It’s now possible to configure plugins to be auto-loaded every time ParaView starts. We’ve added support for plotting over curves and intersection lines using the filters “Plot On Sorted Lines” and “Plot On Intersection Curves”.

A couple of GPU-based rendering/visualization techniques have been incorporated along with GPU-based volume rendering support for 3D image volumes, which is accessible through the "Volume Mapper" option on the Display tab. Support for Line Integral Convolution (LIC) is available as a plugin; this support can be used for visualizing vector fields over arbitrary surfaces.

ParaView now includes (in source form only) an interface to the University of Utah’s Manta interactive software ray tracing engine. The Manta plugin provides a new 3D View type which uses Manta instead of OpenGL for rendering. The plugin is primarily being developed for visualization of large...
datasets on parallel machines. In single processor configuration it has the benefit of allowing realistic rendering effects such as shadows, translucency and reflection.

In terms of performance improvements, we’ve greatly improved the first render time for datasets with large numbers of blocks. Raw image reading for parallel file systems underwent a major overhaul making it fast and efficient. Options were added to the Settings dialog to fine tune image compression, improving interactivity when remote rendering over connections with varying bandwidths.

After the introducing Python tracing in the previous release, we have expanded the purview of tracing to include selections, lookup tables, and implicit functions.

For climate simulation folks, this release includes support for NetCDF with CF (Climate and Forecast) conventions. For cosmology researchers, the Cosmo plugin has been substantially revised. The major improvement is that the plugin now works in a data parallel fashion, so that it can be used with higher resolution simulation results. We have reintroduced basic support for CAVE rendering which was lost since the major overhaul for ParaView 3.0.

AdaptiveParaView, a new experimental application developed using the ParaView application framework is also now available in source format. Like StreamingParaView, AdaptiveParaView processes structured datasets in a piecewise fashion, omitting pieces which are unimportant, in order to make it possible to visualize datasets which do not otherwise fit in RAM. AdaptiveParaView differs from prior work in that it renders pieces in a multi-resolution manner, initially producing low-resolution images and then progressively filling in greater detail within the viewing frustum. This application still contains many experimental features and is not yet documented, but we encourage users to try it out and report bugs and feature requests.

PVBlot is a command tool for batch or interactive processing of Exodus data files. It is provided as a plugin. The commands create various mesh visualizations and XY plots of variable versus time, or variable versus variable. The plugin adds an interactive pvblot console to the ParaView Tools menu. Documentation for PVBlot is built into the tool, just type ‘help’ or ‘help <command>’. The SierraTools plugin provides pvblot-like features but exposes the functionality through toolbar buttons and dialogs in place of text commands.

There are several other fixes including those for charting and plotting, wireframes for quadratic surfaces, and for dealing with temporal ranges.

For developers, this release includes major changes to the core ParaView libraries making it easier to create and deploy custom applications based on the ParaView framework. This enables developers to create applications with fundamentally different workflows than that of ParaView while still leveraging ParaView’s parallel processing and large data visualization capabilities. Search for “Writing Custom Applications” on the ParaView Wiki for details.

The plugin framework has undergone an overhaul as well, making it easier to debug issues with loading of plugins as well as support for importing plugins in static applications.

Starting with ParaView 3.8, we will be releasing development binaries for ParaView, which will make it easier for developers to build and distribute plugins that are compatible with the binaries downloaded from our website.

As always, we rely on your feedback to make ParaView better. Please use http://paraview.uservoice.com/ or click on the “Tell us what you think” link on paraview.org to leave your feedback and vote for new features.

**CMAKE 2.8.1**

We are happy to announce the release of CMake 2.8.1. This is a minor release from 2.8.0. In addition to addressing many issues, this version of CMake adds some new features as well. The parallel nmake replacement from Nokia jom, now has its own tested generator in CMake.

CTest can correctly kill process trees when a timeout occurs. The ctest_* commands have been enhanced to take more options. When using the Intel compilers CMake can correctly determine 32 bit vs. 64 bit builds. When running in parallel mode (-j N), CTest can now compute and use a cost value for each test to ensure the tests are run in an optimal order.

**VTK 5.6**

Kitware is planning to release VTK 5.6 this spring. Since the previous release, 270 new C++ classes have been added. The new features in VTK include improved support for line charts and bar charts that don’t depend on Qt. With this release VTK provides a new text analysis module. Another major improvement is support for parallel rendering.

This release also includes changes that improve performance. VTK can now locate and insert non-uniformly distributed points three times faster than the previous version.

We’ve improved GPU support in VTK. The vtkGPUVolumeRayCastMapper has been migrated from VTKEdge to VTK. VTK 5.6 contains a GPU-based real time Line Integral Convolution (LIC) with enhanced image contrast and LIC + geometry compositing for dense vector field / flow visualization. These are just the highlights. For a complete list of changes, visit the VTK Wiki and search “VTK 5.6 Release Planning.”

**ITK 3.18**

ITK 3.18 will be released in the early spring. The main changes in this release include:

- Improvements in implementation of mathematical traits, particularly for fast rounding, contributed by Brad Lowenkamp and Tom Vercauteren.
- Addition of classes for managing internationalization of strings, contributed by Tom Vercauteren.
- Updated the version of NrrdIO library in Utilities, thanks to Gordon Kindlmann and Michel Audette.
- Fixed shadowing warnings reported by gcc.
Recently added classes were included in the Wrapping, thanks to Gaetan Lehmann.

Improved support for Symmetric Tensor operations, thanks to Luke Bloy.

Improved behavior of the Object Factories, thanks to Brad Lowenkamp.

Improved testing for Discrete Gaussian derivative filter, thanks to Ivan Macia.

Thanks to Michel Audette, BSplineDeformableTransform now clones its bulk transform in the multi-threaded image registration metrics. Bill Lorensen's work on GDCMImageIO computation of proper orientation and origin has enhanced MR Image Storage for DICOM images. Mattes Mutual Information metric now works on binary images. Thanks to Serena Fabri and Hans Johnson for reporting this bug. Brad Lowenkamp and Kent Williams fixed support for analyzing images of large size. The behavior of the VTKPolyDataReader was fixed by Bill Lorensen.

Affine registration of 3D images with Mean Squares metrics, affine registration of 2D binary images using MatchCardinalityMetric and AmoebaOptimizer, and computation of bounding boxes for segmented images were added in Insight. Extraction of iso-surfaces from segmented images and generation of screenshots from iso-surfaces were added in Insight Applications.

The following papers from the Insight Journal were moved into the Code/Review directory:

- “Go-Go Gabor Gadgetry” by Nick Tustison and James Gee, Penn Image Computing and Science Laboratory University of Pennsylvania. (http://hdl.handle.net/1926/500)

- “A Label Geometry Image Filter for Multiple Object Measurement” by Dirk Padfield and James Miller, GE Global Research. (http://hdl.handle.net/1926/1493)

Several third party files were replaced after it was brought to our attention that they were copyrighted by ACM and that they were subject to a license incompatible with the BSD license used by ITK. Many thanks to Professor Michael Saunders (Stanford University) for his patient assistance.

It is well known that a good debugger is an essential programming tool. Debuggers serve not only to effectively resolve defects, but they provide a means to examine the mechanics of an unknown code base and enable rapid development of new code.

Until recently, debuggers had limited utility for complex, high-level libraries like ITK. The built-in print capability of debuggers was limited to simple, low-level data types such as float, char* or C arrays. Examining complex C++ classes, such as an itk::Image, in an informative manner was difficult.

However, the PythonGDB project [1] has solved this problem by allowing the creation of custom pretty-printers for arbitrary data types. GDB [2] now has support for an embedded Python interpreter that hooks into much of the debugger’s functionality. In this article we describe how to use custom Python pretty-printers to improve development of ITK code.

AN EXAMPLE

Note: The code for this example can be found in the examples subdirectory of the author’s ITK pretty-printer repository, gdb-pretty-itk [3].

CUSTOM TEXT PRINTER

Run GDB on the executable that has been built with debugging symbols:

gdb ./itk-example

Set a break point after we have called Update() on an itk::ImageFileReader object:

(gdb) break 33
Breakpoint 1 at 0x41df69: file /tmp/gdb-pretty/ ...
(gdb) run lena.bmp heart.mhd
[Thread debugging using libthread_db enabled]
Breakpoint 1, main (argc=3, argv=0x7fffffffd518 ...
 33      reader2D->Print( std::cout );

ITK has written into the code Print() methods for almost all objects. This is intended to provide debugging information about the object’s state. We evaluate the next line of code to see the output of reader2D’s Print() method:

(gdb) next
ImageFileReader (0x7257f0)
RTTI typeinfo: itk::ImageFileReader<itk::Image<unsigned char, 2u>, itk::DefaultConvertPixelTraits<unsigned char>>
  Reference Count: 1
  Modified Time: 10
  Debug: Off
  Observers: none
  Number Of Required Inputs: 0
  Number Of Required Outputs: 1
  Number Of Threads: 4
  [...] m_Filename: lena.bmp
  m_UseStreaming: 1
  35 - Image2DType::Pointer itkImage2D = reader2D->GetOutput();

The same output can be produced with our custom ITK pretty-printer.

(a) 2-D Gabor image and (b) a 2-D slice from the 3-D Gabor image.
The next part of the code talks about doing some kind of beautification. What does it really do?

```cpp
33 reader2D->Print(std::cout);
34
35 Image2DType::Pointer itkImage2D = reader2D->GetOutput();
36
37 // Beautification...
38 Image2DType::IndexType index;
39 index[0] = 285;
40 index[1] = 354;
41 itkImage2D->SetPixel(index, 5);
```

The GDB Python interface has other simple, effective, and powerful capabilities in addition to creating custom pretty-printers. For example, custom commands can be created. These commands can be loaded using 'require'. Let us load a command that allows selection of a specific pretty-printer (assuming many are available for printing the same data type):

```cpp
(gdb) require command view
```

The 'view with' command takes two arguments: a regular expression that specifies the module that the desired pretty-printer lookup_function is defined in (explained later), and an expression to be passed to the print command. Available modules are shown in itk-example-gdb.py. We issue:

```cpp
(gdb) view with matplotlib.imshow itkImage2D
```

And we immediately see the contents of the image's BufferedRegion.

We see it added a beauty mark above her lip.
3D IMAGE PRINTER

Custom, visual pretty-printers can be generated for 3D images too. For example, iso-contours can be viewed using Mayavi [4]:

```
(gdb) b 63
Breakpoint 3 at 0x41e473: file /tmp/gdb-pretty/ ... 
(gdb) c
Breakpoint 3, main (argc=3, argv=0x7fffffffd518 ... 
63 return EXIT_SUCCESS;
(gdb) view with mayavi.contour itkImage3D
```

Or we can interrogate the data with an application such as ParaView or VV [5]:

```
(gdb) view with vv itkImage3D
```

This loads an IPython shell with variables in the local namespace corresponding to the printed value.

```
In [1]: val7
Out[1]: <enthought.mayavi.sources.vtk_file_reader.VTKFileReader object at 0x6397fb0>
In [2]: mlab.pipeline.volume(val7)
```

Subregions of the image can be easily extracted and examined.

```
In [3]: arz4
Out[3]: <numpy.lib.io.NpzFile object at 0x1c27390>
In [4]: arz4.items() 
Out[4]: [('scalars', array([[16, 15, 16, ..., 24, 26, 30],
[18, 13, 15, ..., 24, 27, 31],
[15, 10, 11, ..., 20, 26, 31],
[17, 12, 14, ..., 22, 28, 32],
[44, 45, 39, ..., 39, 36, 28],
[ 0, 0, 0, ..., 0, 0, 0],
[ 0, 0, 0, ..., 0, 0, 0]], dtype=uint8)),
('origin', array([ 0., 0., 0.]),
('spacing', array([ 1., 1., 4.]))),
[...]
[58, 59, 47, ..., 43, 41, 35],
[62, 57, 43, ..., 38, 33, 26],
[55, 54, 49, ..., 34, 26, 18]],
dtype=uint8)),
('origin', array([ 0., 0., 0.]),
('spacing', array([ 1., 1., 4.])))
```

We can also save images as we progress through our program for comparison by subtraction or other exploratory methods in our IPython session.

```
In [5]: plt.plot( arz4['scalars'][:,10,10] )
```

ExAmInIng IN AN InTERRACive INPyThOn SHEll

We can dynamically explore the data in an IPython [6] shell using Matplotlib’s 2D pyplot facilities [7] or Mayavi’s 3D Mlab facilities:

```
(gdb) view with mayavi.ipython itkImage3D
$7 = /tmp/tmpKd8u/7.vtk /tmp/tmpKd8u/7.npz mlab
pid = 14823
```

We can also save images as we progress through our program for comparison by subtraction or other exploratory methods in our IPython session.
A lookup function can be any Python function that performs filtering on the value. In practice, the lookup function usually consists of a few common steps focusing on the type attribute associated with the gdb.Value. If the type is a reference, it is converted to the type that it references. If the type is a typedef, it is converted to the original type. Filtering can examine template arguments. If the type is an itk::SmartPointer<> , the template argument is extracted. Finally, a pretty-printer Python dictionary associates a regular expression matching the typename to the corresponding pretty-printer class.

Step 3: Associate the lookup function with an object file
Lookup functions can be associated with GDB objfiles. Here objfiles refer to executables or libraries. A file in the same directory as the executable or library with the same name except for an appended -gdb.py gets executed when GDB loads the objfile. This script is used to register the lookup functions associated with the program or library. This allows only the pretty-printers needed by the program to be loaded, which reduces startup time. The order of the lookup functions also determines the default pretty-printer.

FURTHER READING
The author’s GDB branch used for this article can be found on gitorious [8]. ITK pretty-printers can be found at the same location, gdb-pretty-itk. Documentation for GDB and the Python API can be generated by calling ‘make html’ in the gdb/doc directory of GDB’s source. A screencast demonstrating custom pretty printers can be found at archive.org [9].

REFERENCES
[5] VV is an open-source and cross-platform image viewer, designed for fast and simple spatio-temporal images visualization: 2D, 2D+t, 3D and 3D+t (or 4D) images http://www. creatis.insa-lyon.fr/rio/vv/

ACKNOWLEDGEMENTS:
Thanks to Tom Tromey, Jan Kratochvil, Joel Brobecker, Phil Muldoon, et al. for their work on the GDB project and helpfulness on the mailing lists and IRC.

Matt McCormick is a PhD candidate at the University of Wisconsin-Madison in the Department of Biomedical Engineering. His research interests include diagnostic ultrasound strain imaging and parametric tissue characterization. He enjoys scientific computing and is an active member of UW-Madison Hacker Within (hackerwithin.org).
In the RobotCub project [1], we’ve been building a humanoid robot called the iCub. There are about a dozen iCubs in existence today (see Figure 1). The iCub design is free and open source. The designs for the mechanical and electrical components of the robot, its bill of materials, and all of its software are released publicly under free and open source licenses [2,3]. The iCub was created as part of a European project which began in September 2004 and ended in January 2010. There are several follow-on projects using and extending the robot in many ways, and we hope it will be a catalyst for research in robotics, embodied cognition, experimental psychology, and perhaps fields we’ve never even heard of. There were a lot of challenges in building the robot and making it usable. On the software side, CMake has been very important to us. We’d like to say a big “thank-you!” to the CMake developers, and explain a little about how CMake helped us along the way.

Figure 1: The iCub is a full humanoid with legs, dexterous hands, and a simple expressive face. The iCub is the size of a young child.

Making a humanoid robot is a huge task, requiring both novel research and patient integration work. It is practically impossible for any one person to be an expert in everything involved. Typical members of our development team are graduate students and postdocs who are experts in their fields, but may not necessarily be accustomed to working with others on software development. CMake turned out to be a wonderful tool for us, for the following reasons.

CMAKE HELPS DEVELOPERS WORK TOGETHER
People with all sorts of backgrounds end up working in humanoid robotics, so we see a lot of diversity in the development environments which people are comfortable. The biggest divide, when we began, was between users of Microsoft Visual Studio on Windows, and users of gcc+make+emacs/vi/… on Linux. From the start, RobotCub committed to formally supporting both these groups. But it was a lot of tedious work to keep all of the build files synchronized, even with some custom scripting to help. And it was difficult to force individual developers to spend time dealing with tedious details of environments they didn’t use or care about. This was a recipe for long unproductive discussions about why everyone should be using the one true development environment (whichever that was).

The first CMakeLists.txt file entered one of our repositories in April 2006 with the commit message “CMake, for easier Windows distribution” (this was clearly the perspective of a Linux developer tired of dealing with Windows). After some experimentation, doubts, and dabbling with alternatives, it became clear that CMake was a huge win. It gave a first-class experience to MSVC users when compiling projects developed by Linux users, and vice versa. Most of the time, developers could now forget about any development environment other than their own, and our community became a lot less “balkanized”.

CMAKE HELPS INDIVIDUAL DEVELOPERS
Although we did not anticipate this, we discovered that CMake saved individual developers a huge amount of time, even before their work was used by others. One of our developers said “I can’t imagine how much time I spent in my life cloning Visual Studio project files”. CMake proved to be a better way of managing projects than was possible within MSVC itself. Similarly, beginner Linux users were very happy to stop looking at Makefiles that they didn’t really understand and had a hard time modifying. CMake democratized the build process through simplification.

Another unanticipated benefit was noticed over time. For MSVC users, as new compiler versions were released (VC8, VC9, VC10), they were saved from dealing with a lot of problems because their projects were already described in a compiler-neutral way, and CMake was prompt in its support for those compilers. Linux users found they could easily experiment with alternative development environments such as KDevelop or Eclipse. It was great for the RobotCub project not to have to dictate or even care about the development environment our developers worked with.

Since 2007, there’s been a big movement from Windows toward Linux amongst students, principally driven by the phenomenal growth of Ubuntu. At our 2006 summer school, Windows was dominant; by 2009, the numbers were completely reversed. However, people whose background lies outside of computer science seem more likely to be still using Windows. Today, some robotic software projects neglect support for Windows (typical quote: “let’s be serious: it’s a robotic tool!”). Since humanoid robotics is a very interdisciplinary field, we believe it would be a mistake for us to adopt that attitude. In any case, good Windows support, via CMake, has actually helped individuals working on RobotCub make the leap from Windows to Linux, since they know they’ll be able to compile their code from day one. In general, CMake reduces lock-in to any particular development environment, which feels very liberating.

CMAKE HELPS SUMMER SCHOOLS SUCCEED
Since 2006, the RobotCub project has had an annual summer school (“VVV 06-09”, and the school will continue in 2010 under the auspices of the iTALK project). The schools are essentially hackathons where we take whatever hardware we have to a nice locale in Italy and invite anybody with the necessary skills to come play. The schools are BYOL (Bring Your Own Laptop), and have proven to be an excellent “forcing function” for making the iCub usable and integrated. From the start, CMake has been a big help in getting such a mob of programmers up and running and collaborating. Here’s how we introduced CMake at the first summer school, VVV06 (see Figure 2):

“We’d like you all to use the development environment you are used to, and not force you to switch to something else -- no Linux guys+emacs vs Windows/DevStudio vs Mac… fights please! To achieve this without complete chaos, we ask you to install "CMake". CMake lets us describe our programs and libraries in a cross-platform way. CMake takes care of creating the makefiles or workspaces needed by whatever development environment you like to work in.”
We suspect that the open attitude that CMake facilitates has played no small role in the success of the school.

**CMAKE HELPED US SCALE UP OUR REPOSITORY**

RobotCub’s community of developers is open and dispersed. The member institutes of the formal RobotCub consortium are scattered throughout Europe and we have collaborators worldwide. Software development mostly happens within an academic environment, where project and thesis schedules may conflict and the backgrounds or skill-level of developers vary widely. The most interesting software is often novel, experimental, created by researchers whose main skills (naturally and rightfully) lie in their problem domain rather than in software engineering. Due to paper submission deadlines, this software is often written in a hurry with little thought to reuse by others. For all these reasons, building up a common software repository for the iCub robot has been an interesting experience.

With a large community of loosely-coordinated developers it was difficult to reduce the list of dependencies required to compile the whole repository. In addition, some of the low-level code was not fully redistributable and thus, could not be included in the repository (this was the case for some device drivers required to interface with the hardware). Quite soon, being able to compile every single module in the repository required a large list of dependencies. This is something that often scares users away, especially in Windows, where package management is not as evolved and helpful as in other operating systems. For understandable reasons, new users are not very eager to spend a long time setting up software. CMake turned out to be very helpful in this. Our repository has a modular structure, where a group of flags govern the activation of groups of modules (e.g., graphical user interfaces, low-level modules providing hardware support, the simulator and so forth). Users that do not have all the required libraries can still compile a subset of the code, and start playing with it. Our policy is that the build process should not raise fatal errors when dependencies are not met, but rather should make available only the group of modules that can be compiled with the available dependencies. These flags controlling the build are made clear in the CMake GUI so that the user is aware that some parts of the build are not available. Getting and building the repository is an incremental and friendly process that avoids scaring away (new) users with incomprehensible error messages about unmet (and sometimes unwanted) dependencies.

For contributing to the repository, we have kept a low barrier of entry. RobotCub folks are told, in the top-level README, to "feel free to place new modules in [the repository] that aren't yet well tested, are currently broken, came to you in a deranged dream, etc." We found that it was very important to keep the initial barrier to entry very low, or code tended to stay hidden within institutions until it was fully done and perfect (i.e. never). With a low barrier to entry, there was mutual visibility between developers in the different institutions, and collaboration was easier to get going at the programmer level. Once developers are sure their module is no longer a "deranged dream" and, more specifically, are confident that it compiles on machines other than their own, they are encouraged to add it to the global build for the repository. This is quite a simple process even for inexperienced users: in most cases it is enough to customize a simple template CMakeLists.txt file and include it in the main build. Once in the global build, the module is tested automatically on our "officially supported" Linux/gcc and Windows/MSVC version combinations and we do some quality control at that point.

The plot below illustrates how the RobotCub software repository has grown during the project from its establishment in 2006, to the end of the project. At the end of the project our community consists of 32 developers (all contributing code in the past year), with almost 1 million lines of code and about 120 modules (programs and libraries) compiled in a single project (see Figure 3):

**CONCLUSIONS**

CMake helped keep our developers from being divided by operating system, and no one was given the impression that they were a second-class citizen. Everyone spent more time working and less time fiddling around. Our repository grew to the scale we needed to operate a multi-functioned humanoid robot, without trouble. New users were able to build the parts of the project they were curious about first, and work their way up to everything else as they needed to. Being relatively early adopters of CMake, we made a bit of a muddle of our build structure (we had never used CMake before RobotCub), but things worked out remarkably well.
in spite of that. More recently, we’ve noticed many peer projects starting to use CMake, for example Player/Stage and ROS. It is good to see this “bottom-up” convergence, since it makes collaboration easier.

It is striking how many free and open source projects played a vital role in developing the iCub: CMake, Doxygen (which allowed us to easily pool and browse documentation, and provided “soft” pressure on developers to actually write that documentation by making its absence very obvious), SWIG (was very useful in expanding our pool of users), the OpenCV library (which served as a de facto standard for basic computer vision algorithms, rather than everyone writing their own). The list goes on and on. We hope that the iCub robot or a descendent will find a similarly useful role in robotics, and are grateful to our funder, the European Commission, for allowing us to give back to the free software community that has served us so well.

REFERENCES

Paul Fitzpatrick received his MEng in computer engineering from the University of Limerick, Ireland, and his PhD in computer science from MIT for work addressing developmental approaches to machine perception, implemented on the humanoid robots Cog and Kismet.

Giorgio Metta is senior scientist at the IIT and assistant professor at the University of Genoa where he teaches courses on anthropomorphic robotics and intelligent systems for the bioengineering curricula. He holds a MS with honors and PhD in electronic engineering from the University of Genoa. His research interests include biologically motivated and humanoid robotics, in particular lifelong artificial systems which demonstrate the abilities of natural systems.

Lorenzo Natale is Team Leader in the Robotics Brain and Cognitive Sciences Department at the Italian Institute of Technology. Since 2000 he has been involved in various humanoid robotic projects, working on aspects related to perception, motor control and learning. He is also interested in software development and is one of the main developers of YARP, an open-source middleware for robotics.

PARAVIEW IN AERODYNAMICS

CFS Engineering is a consultancy company founded in August 1999, based in the business park at the Swiss Federal Institute of Technology in Lausanne, Switzerland. CFS Engineering offers services in the Numerical Simulation of Fluid Mechanics and Structural Mechanics Engineering Problems, and in particular, is active in the development of the Navier Stokes Multi Block (NSMB) solver used in a variety of aerospace problems ranging from flows over civil and military aircraft to hypersonic flows including air chemistry over future re-entry vehicles.

The NSMB solver solves the Navier Stokes equations using the finite volume approach. Space discretization schemes include central and upwind methods. The equations are integrated in time using the LU-SGS scheme.

NSMB offers a variety of modern turbulence models. Different levels of chemistry modeling are available for hypersonic flows, ranging from chemical equilibrium to thermal chemical non-equilibrium. NSMB has been parallelized using the SPMD paradigm using MPI as message passing.

ParaView has been an important asset to this small company's portfolio of tools, and is used in mesh and fluid flow visualization tasks.

The NSMB solver archives its data in a proprietary file format. Large and complex flight geometries can produce gigabytes of data for steady and transient simulations.

Our interface has grown alongside the release cycles of ParaView, and we are today, extremely grateful to Kitware and the open source community at large for all the features available. Our ParaView plugin for the NSMB solver supports a wide range of data analysis tasks, which we succinctly present here.

HIERARCHICAL MESHES, QUANTITATIVE AND QUALITATIVE DISPLAYS

Our reader plugin relies on a low-level access API for the coordinates and data arrays, and lets us concentrate on building the VTK objects used by a hierarchy of multiple containers of 3D and 2D grids. Enabling parallel reading was particularly important, given today's multi-core servers and desktops. In fact, our rendering needs are often easily handled by a single hardware renderer (the ParaView client), whereas a pvserver can share the work load amongst several machines.

A typical flight configuration may include several thousands of body-fitting 3D curvilinear grids, along with surface patches for the solid geometry, and the different boundary condition's walls. The data will include standard flow variables (density, pressure, velocity) and wall shear stress vectors. ParaView enables the graceful presentation and sub-setting of such hierarchies of data blocks along with data fields and multiple time-cycles.

Multi-parameter studies, where flow conditions may change and the simulation is re-executed can take advantage of the side-by-side viewing available in Comparative View mode, as shown below for a hypersonic flow simulation.
The integrated view possibilities enabling 3D geometry viewing, spreadsheet displays of tabular data, XY plots, and synchronized selection (highlighted in purple below), between different viewports are also particularly handy.

**PARALLEL EXECUTION AND PYTHON SCRIPTING**

Scripting is the key to a great visualization tool, and is vastly improved if done with Python. ParaView 3.6.2 now integrates a trace tool, assisting the user in generating human-readable Python scripts. We foresee that the use of state files will eventually disappear thanks to this new feature.

We rely on Python scripting to automate the more mundane tasks, or to add functionality not available in the GUI. The following example illustrates how we can operate on FieldData (there are currently no filters enabling access to the FieldData). Our multi-patch surface geometries are encoded with a single integer, telling us what the boundary condition type is (e.g. solid, symmetry, inflow/outflow walls) or what the solid object is (wheel, wing, nose, body, nacelle). Part selection requires access to this 1-tuple array called “BCcode” and stored in FieldData. A Python Programmable Filter is the perfect tool to traverse the hierarchy of structured grids, shallow-copying only those matching a user-selectable set of codes, stored in a Python list:

```python
input = self.GetInputDataObject(0, 0) output = self.GetOutputDataObject(0) if input.IsA("vtkMultiBlockDataSet"): output.CopyStructure(input) iter = input.NewIterator() iter.UnRegister(None) while not iter.IsDoneWithTraversal(): in = iter.GetCurrentDataObject() out = in.NewInstance()
```

ParaView’s trace tool enables rapid-prototyping, fine tuning, and execution with pvbatch, the parallel server of ParaView. Visualization campaigns that would last several days, and were driven from state files after some cumbersome and error-prone hand-editing, can now be replaced by Python scripts. The scripts are also our preferred method for regression testing.

**EXAMPLES OF A FEW CFD-SPECIFIC FILTERS**

The following Python script drives our drawing of wall-bound shear stress lines of Figure 4.

```python
# defines a list of patch indices for the tailwing tailwing = ExtractBlock(reader) tailwing.BlockIndices = list1 rep2 = Show(tailwing) merge1 = MergeBlocks(Input = tailwing) clean1 = CleantoGrid(Input=merge1) mask1 = MaskPoints(Input=clean1) mask1.OnRatio = 7 mask1.Random = 1 stream1 = StreamTracerWithCustomSource(Input=clean1, Source=mask1) stream1.Vectors = ['POINTS', 'TauW'] stream1.IntegrationDirection = 'BOTH' rep3 = Show(stream1)
```

Plot-over-sorted-lines, a recent addition to ParaView, allows us to draw an XY plot of pressure contours across wing side-cuts (left Figure 3). This is a must-have for CFD studies.

**DATA QUERIES**

A new addition to ParaView is a query engine, enabling selection of nodes or cells by field values. Finding, for example, the low-pressure points downstream of a flying aircraft to study wing-tip vortices is now a trivial task with a simple query: *Select Points where Pressure is less than 121000.*

The points are then used as seeds for streamlines and they highlight quite well the downstream high vorticity region, as shown below:

![Figure 4: Wing-tip vortices ending in a low-pressure zone.](image)

New representation plugins, are of outstanding value, to enhance our visual experience. Take the new Line Integral Convolution plugin, integrated into ParaView 3.8, and compare the two pictures below which depicts the wall shear...
Getting Kitware tools up and running on a Darwin-OSX system is not as easy as loading binaries from the Kitware servers. This is because there are binaries for CMake on OSX, but none for VTK and ITK. If you’ve rolled your own Kitware toolkits on Linux, you would think building from the source is the next step. At least this is what I did.

After downloading the release bundles for CMake, VTK and ITK, configuring, compiling and installing CMake and then configuring, compiling and installing VTK and ITK, I found myself wondering….. Did I get shared libs? Where are the .so files? I went back to cmake to check the configuration and build again. The configuration specified shared libs, but I got a bunch of .dylib files. What are they and how do I look into them? (So Google and ohhhh! We use otool, not ldd, on OSX.)

I also wanted a lot of other options, including wrapping for Python, connectivity to databases, and threading. So, I started to collect several more downloads, including binaries for Python, MySQL and PostgreSQL. I had no idea where these binaries were installed. I was wondering - what are frameworks? Did I really ask for that? Do I really want to link against a framework? What were those config options again?

After hopping on the email list to ask for help from the gurus I ended up, several days later, with several documents on how to configure a dozen packages, lots of mysterious build and run-time dependencies, tons of email threads on several lists, and I lost a couple of handfuls of hair!

Building these tools on Darwin-OSX is not so easy. I thought to myself, there has to be a better way! I soon realized that ports exist to provide access to some of the Kitware tools. However, I also realized that they do not provide all the options that I want. And then I found MacPorts. After learning a few things about how to use and develop ports, it was clear that MacPorts provides (a) a way to effectively document the build and install process, (b) automatic resolution and installation of build and library dependencies, (c) an easy path to replicate installations across systems (including binary packages), and (d) easy maintenance of the installation and upgrade process.

MacPorts

I have provided several references at the end of this article to help other users begin working with MacPorts [1], [2], [3]. MacPorts have a few easy build and installation options for Kitware tools on OSX. What could be easier than these installation commands?

```bash
$ sudo port install cmake
$ sudo port install InsightToolkit
$ sudo port install vtk-devel
```

The port system has automatic detection of CMake as a build dependency, so this works too:

```bash
$ sudo port install InsightToolkit vtk-devel
```

If CMake is not already installed, this command will run the installation. (The build for ITK includes a build for CableSwig,

**REFERENCES**


Jean M. Favre is a research engineer at the Swiss National Supercomputer Center (CSCS) in Manno, Switzerland, where he supports multiple visualization projects with ParaView, from astrophysics, to biomechanics, fluid dynamics, geophysics, and molecular dynamics.

Jan Vos, leads CFS Engineering with his experience in fluid mechanics simulations, including Magnetohydrodynamic Flows, combustion, hypersonic flows and flows for aeronautical applications. Jan is also lecturer at the Laboratory of Computational Engineering at the EPFL.
hacking the port to provide the right variant may be the solution for you and thousands of others too.

Bugs! The best way to contact a port maintainer about any installation problem is to create a ticket at http://trac.macports.org/. Also, you might get some help from macports-users@lists.macosforge.org. If you discover functional bugs after the installation, that's likely an upstream issue in the source of the toolkit.

MACPORTS FOR VTK
At the time this article was written there were several ports available for VTK (5.4.2 is in ‘vtk-devel’). To find all the VTK ports, try a search like this:

```
$ port search vtk
```

Then, check out the description and the variants with

```
$ port info vtk-devel
$ port variants vtk-devel
```

There are variants available for documentation, wrapping, database connectivity, MPI, and so on. Almost all of the configuration options available in CMake have a variant in the port.

MACPORTS FOR ITK
The MacPorts for ITK are available at several recent release versions. To find all the current versions available, use

```
$ port search InsightToolkit
```

To get the port description and variants, try

```
$ port info InsightToolkit
$ port variants InsightToolkit
```

The default variants are indicated with a [+] and variant contingencies may be indicated with ‘*requires’ and ‘*conflicts’. At the time this article was written, the defaults included documentation, shared libs, and wrapping for Java, Python, and Tcl.

For a new installation of ITK, at the latest release available, try

```
$ sudo port selfupdate
$ sudo port install InsightToolkit
```

It's recommended that you run the installation overnight. For detailed progress on the build and install, use

```
$ sudo port -d install InsightToolkit
```

The -d option is used for 'debug' information, but also serves as a useful progress meter. When it's complete, you can see everything installed with

```
$ port contents InsightToolkit
```

ITK UPGRADES
To upgrade to a new release of ITK use the following command

```
$ sudo port selfupdate
$ sudo port upgrade InsightToolkit
```

Issues with MacPorts
It may take hours to build and install Kitware libraries. Nevertheless, it saves all the headache of fetching release bundles, manual configuration, and fixing tricky link dependencies and bugs. By default, MacPorts builds from source, so it is not as efficient as some binary systems, like Yum/rpm and Debian. (If you prefer a Debian system on OSX, check out “fink”.) There are options to enable creation of binary archives from your builds, which can be shipped to other systems in your domain. A significant advantage of MacPorts is the dependency resolution architecture that facilitates careful library linking to other open-source ports that can coexist with the Darwin-OSX system. The dependency resolution architecture provides library links that can be isolated from Apple system upgrades and it can facilitate integrated updates of all packages within MacPorts.

As with almost any package management system, there are some issues with complex software integration. For instance, consider the problem of integrating VTK with Qt. Just run a search for this online and you will find numerous discussions of many troubles trying to explore all the version compatibility and build contingencies to get these and other libraries to cooperate. Sure, you may choose to go it alone and get something working for your system (probably under /usr/local), but you might first consider learning MacPorts and becoming yet another contributor to solving the same problem for thousands of developers.

Some of the greatest trouble in building for OSX lies within choices for the display engine (cocoa, carbon, x11). Some of the libraries use one or another by default for various reasons. If you need to change a default display, first look at the variants for the port and, if that's not your solution,
The following command can be used to remove a previous installation when performing the upgrade

$ sudo port upgrade -u InsightToolkit

The latest version can coexist with prior versions, such as ITK 3.14. The ports are designed to install into version-specific paths and to rename binaries with version-specific names. The latest release of the port should have symlinks from generic paths and binaries to the latest version specific installation.

To keep an older version after an upgrade, it may be necessary to install a prior version-specific port (e.g., "InsightToolkit314"). That process could look like this:

$ sudo port selfupdate
$ sudo port upgrade -u InsightToolkit
$ sudo port install InsightToolkit314

WRAPPING FOR ITK
Some port hacks were required to get all the library link settings right for the Tcl and Python libraries in WrapITK/lib (by changing the library paths using install_name_tool). Further hacks were required for the itkwish shell script in $(prefix)/bin/itkwish and the binary in WrapITK/bin/itkwish. At the time of testing, itkwish worked and 'import itk' worked for Python 2.5. Java wrapping is also available, but it was not tested for the purposes of this article.

CMAKE ISSUES
To build against a VTK or ITK library, your CMakeLists.txt may have to set the VTK_DIR or the ITK_DIR to be certain about linking

SET (ITK_DIR /opt/local/lib/InsightToolkit-3.16)
SET (VTK_DIR /opt/local/lib/vtk-5.4)

The ITK port currently adds a version-specific CMake module, based on modifications to FindITK.cmake (found in $(prefix)/share/cmake-*/Modules/). So, it may be possible to use the following for ITK:

FIND_PACKAGE(ITK-3.16 REQUIRED)

Most, if not all, of the default CMake modules will not look for libraries in the MacPorts $(prefix), which is /opt/local by default [4].

PORTFILES
For the curious, the details of the port are all in the ‘Portfile’ that contains pseudo-Tcl code for the build and install process. If you want to edit and install your own custom Portfile, follow the instructions on “local repositories” in the http://guide.macports.org.

The most current Portfile is in the svn server, which can be browsed at http://svn.macosforge.org - navigate to the trunk/dports/ area of interest. For the ports currently in sync on your system, view their properties and content using

$ port file InsightToolkit
$ port cat InsightToolkit

DEVELOPMENT WITH MACPORTS: INSIGHTTOOLKIT
For a MacPorts package, most of the library dependencies are configured for other open-source packages provided by MacPorts (some utilities or libraries may be provided by Apple). To see dependencies of a port, try

$ port deps InsightToolkit

To take advantage of this when developing ITK, first learn how to create a “local repository” [5]. Once you have a local repository, copy the current Portfile for ITK so that you can modify it (find it using ‘port file InsightToolkit’).

The port installation process actually comprises several phases [5]. A developer can stop the installation after the build phase (see ‘man port’ for all the developer targets). At that point, there is a complete source tree and a separate build tree (with no tweaks to the Portfile, you get a release version of ITK, see ‘port info InsightToolkit’). To try this

$ sudo port build InsightToolkit
$ macportPath="~/opt/local/var/macports"
$ buildPath=$(macportPath)/build/*InsightToolkit/
$ ls $buildPath

Then you might try to develop directly in the MacPorts build tree. If you’re tempted to copy the InsightToolkit tree somewhere else to develop with it, there are some rpath issues to contend with. One problem with moving the build tree somewhere else is that the OSX convention is to use rpath, so the build tree has a lot of specific links within the build tree (many of these are reset during the "destroor" phase and there are a ton of hacks in the InsightToolkit port to get the rpath right after the "destroor" phase).

Use the following commands if you need to check or correct any CMake configuration settings, or if you need to read and edit your Portfile

$ sudo port configure InsightToolkit <+VARIANTS>
$ cd $buildPath # see above
$ sudo cmake -LAH ../InsightToolkit-<version>

If you develop directly in the MacPorts build tree, use gmake directly in the build path or rebuild with 'port build InsightToolkit' (this should be an incremental build, unless the Portfile is changed). If you use the port build command, it is likely to say the build is complete; to override that you could use the -f flag or you can edit the 'state' file (work/macports.InsightToolkit.state in the build path) to remove the line that indicates the build is done. The port command can sometimes clean out a build tree, but that can be disabled with a -k option to keep the working build tree. For instance, if you run all the phases of a full installation, without a -k option, the MacPorts install process will ‘clean’ away the entire source and build trees (that’s the default set in $(prefix)/etc/macports/macports.config).

If you need a current CVS source tree, tweak the Portfile in your local repository. Just modify the version definition and the fetch phase, which can be configured to use CVS retrieval [5]. The Portfile assumes there is a release version and the version string is used throughout the Portfile to provide version specific installations (you may need to set a pseudo-version string). Note that version, epoch, and revision strings are used by port to identify the most current release for installation.

ACKNOWLEDGEMENTS
This work involved help from generous developers in the open source community, including folks dwelling @cmake.
The above figure shows an MR detection study of a patient with elevated PSA, where prior ultrasound biopsy could not confirm the presence of cancer. The detection protocol includes: T2-w, ADC, and DCE-MRI sequences. The radiologist marks a finding, which is suspicious for cancer, with a transparent red sphere (vtkUnstructuredGrid) and is presented with CADx output to assist in differentiating between benign and malignant. A malignancy score of .33 (green and red are scores of benign and malignant training findings) indicates that the region is unlikely to contain cancer. This patient is currently in an active surveillance program.

A 59-year-old patient with a PSA of 12 ng/ml and a confirmed biopsy Gleason 8 tumor in the right peripheral zone. The MRCD hanging protocol shows T2-w transversal (right view) and in color overlays: Pharmacokinetic DCE-MRI; and ADC-map (left upper row); and choline metabolite concentration; coronal T2-w image (left bottom row); The separate window shows a time-concentration curve and an MR Spectroscopy spectrum at the cursor location. The reporting radiologist rated all 4 modalities 5 on a 5 point scale: where 1 is no tumor, and 5 is definitely tumor. These MR findings were confirmed retrospectively with a stage T3a determined after prostatectomy.

OVERVIEW OF THE SOFTWARE
The workstation runs an OpenSuse Linux OS. A DCMTK server daemon receives DICOM images automatically forwarded by the PACS system. A polling mechanism triggers automatic
pre-processing of MR studies (e.g., fitting DCE-MRI curves). The studies are then ready for viewing and analysis. MRCAD features are: compressed, multiframe DICOM Reading/ Writing; multi-modal color overlay; DCEMR pharmacokinetic modeling and registration; manual and semi-automatic segmentation; region of interest statistics; CADx, CADE; MRS (using LCModel), and Structured Reporting.

The MRCAD software has a layered structure as shown below. The main programming language is Tcl/Tk with the object oriented extension IncrTcl. Processing and display are performed in VTK pipelines that are setup using the Tcl wrapped interface. Local VTK classes provide additional high computational functionality at the C++ level or as means to include wrappable ITK functionality.

**ANNOTATION AND STRUCTURED REPORTING**

Clinical findings and ground truth for CAD are annotated and stored in XML. Example tags are: tumor location, tumor grade, scorings, and access time. For structured reporting, findings can be collected and added to the XML database. A PDF file can be generated from the XML file (using LaTeX) and, after inspection, sent to an Eletronic Patient Database using SOAP requests. The XML data is additionally used for training CAD systems, and performing clinical observer studies. An XML example of an observer scoring is:

```xml
<markdatasetlists PatientsName="Anonymous"
PatientID="12345" StudyDate="20100217"
InstitutionName="UMC_St_Radboud"
StudyDescription="ABDOMEN-ONDERBUIK">
  <marklist username="pieter" mode="detect" repeat="1" studysetname="Default">
    <mark id="mark1"
      <StudyDate>20091220</StudyDate>
      <T2>5</T2><DCE>4</DCE><DIFF>4</DIFF
      <Type>Tumor</Type>
      <AlignedModalityBG>T2Tra1</AlignedModalityBG>
      <Stage>T2c</Stage>
      <AP>46</AP>/<CR>57</CR><CC>46</CC>
      </mark>
    </marklist>
  </markdatasetlists/>
```

The node markdatasetlists is the root node for all annotations of the MR study. In this example (user ‘pieter’ in the marklist node) a tumor is detected and is rated 5-4-4 on 3 MR modalities. Furthermore, the finding is linked to another finding in a prior MR study (Link1). The patient diagnosis is in the markpatient node: a T2c tumor. After reporting the patient case, MRCAD can use the findings stored in the XML database to collect screenshots and format the information into a PDF file using dedicated stylesheets. This PDF report is then sent to the Electronic Patient Database using SOAP.

```
set token {[::http::geturl $posturl -type "text/xml; boundary="" -headers [list Cookie $sessionid] -query [$soaprequest asXML]]
  set body [http::data $token]
  http::cleanup $token
```

**COORDINATE SYSTEM**

MRCAD uses three coordinate systems to handle the variety of image volume and voxel dimensions and orientations in an MR scan: ijk voxel location in an MR series, xyz patient position, uvw viewport coordinate. A vtkDCMTKImageReader directly reads DICOM images in single slice, multiframe, and/or compressed format and produces a vtkImageData and a vtkTransform (ijk2xyz). The latter transform maps voxel location ijk to an xyz patient position. The viewer uses an additional uww2xyz transform to handle: zooming, panning and slice selection. This allows for accurate (sub-voxel) overlay of any image (e.g transversal 0.5x0.5x3mm) onto any other orientated image (oblique (15deg) sagital (1x1x4mm)). Arbitrarily shaped 3D volume annotations are created and stored in xzy coordinates using the vtkUnstructuredGrid format. These annotations can be added and overlayed to any view and are also used to compute region of interest statistics in the various multi-modal images.

**DATA AND WIDGET FACTORY**

The core of the MRCAD application is designed as a factory method pattern. The approach ensures fast initializing of the application, by constructing data objects on demand. As a result, VTK pipelines are only created when needed. An IncrTcl coded part of the factory:

```tcl
class MRCADDataFactory {
  ...
  public {
    # static method that returns the singleton
    proc Instance {} {
      # check whether the data object ''mapname''
      if { ![isAvailable $mapname] } {
        # Register the data object named ''mapname''
        set _rgContainer($mapname) $object
      }
    }
  }
  # Implementation
  body MRCADDataFactory:GetData {mapname} {
    # Returns the data object from the factory
    set data _rgContainer($mapname)
    # Register the data object named ''mapname''
    method AddData {mapname data} {
      # Returns the data object from the factory
      return [namespace which $object]
    }
  }
}
```

The factory is implemented as a singleton and can be queried using the static member function Instance. The factory is filled with possible data during MRCAD initialization, for example a diffusion-w series might be present, then:
adds an imagedata object with mapname "Diff1", which is constructed by a specialized class DiskImageData (an IncrTcl wrapper of the vtkDCMTKImageReader).

To view the diffusion-w image in the application:

```csharp
if {{[MRCADDataFactory::Instance] IsAvailable "Diff1") { # Get the data object and create the VTK pipeline necessary to produce the data set obj {{[MRCADDataFactory::Instance] GetData "Diff1"}} # Display the image in the viewer viSetImage BG [$obj GetVolume] viSetTransform BG [$obj GetTransform]
```

Data objects can be accompanied by data widgets to allow user interaction with the data object. In the image below, the spectroscopic metabolite image data object output is displayed as a transparent, color-coded overlay on top of a T2-w image. Multiple metabolite maps are available and the accompanying data widget allows selection of the metabolite and provides other interactions with the data object. Here, the 3th available metabolite, choline (Cho), is selected:

**RESEARCH OVERVIEW**

MRCAD is used in a number of research projects. MRCAD was used to detect and annotate the Dominant Intraprostatic Lesion for radiotherapy IMRT planning [1]. A successful Computer Aided Diagnosis system has been researched and implemented to discriminate benign from malignant suspicious regions using region of interest statistical features derived from DICOM images directly or processed parametric maps [2]. Pattern recognition as well as ROC analysis was performed using the statistical package R. Pharmacokinetic features computed from Dynamic Contrast Enhanced MRI provided the highest diagnostic accuracy. Subsequently in [3], more modalities/features were added to improve the discriminating performance. An ITK-based registration method was extended with a local incompressibility term to add MR series features that were misaligned by possible patient movement during acquisition. Currently, [4] and [5] an initial system for fully automatic detection of prostate cancer is being developed using a locally developed vtkClass based on the itkHessianRecursiveGaussianImageFilter to detect lesions. MRCAD is also used in several observer studies that are performed by radiologists to determine the diagnostic value of a certain MR modality or protocol. In [6] MRCAD was used for scoring DCE-MRI derived parameters and establish the value for localizing prostate cancer. Recent use includes detecting MR suspicious lesions for MR or ultrasound guided biopsy and detecting recurrence after radiotherapy [7,8].

**REFERENCES**


**Henkjan Huisman** is an Assistant Professor in the Diagnostic Image Analysis Group in the Department of Radiology at Radboud University Nijmegen Medical Centre. His research interests include computer aided diagnosis, image segmentation, prior knowledge integration in image analysis, and pharmacokinetic MRI.

**Pieter Vos** is a PhD student at the Diagnostic Image Analysis Group in the Department of Radiology at Radboud University Nijmegen Medical Centre where he’s working on Computer Aided Diagnosis of prostate cancer using multimodal MR.

**MATLAB® and GNU R Integration with VTK Now Available**

The software integration work discussed in the October 2009 issue of Kitware Source, **MATLAB® and GNU R Integration**, is now available on the main development branch of VTK. The integration classes are located in the VTK/Graphics directory. The MATLAB integration classes are: vtkMatlabMexAdapter, vtkMatlabEngineInterface, and vtkMatlabEngineFilter. The GNU R integration classes are: vtkRAdapter, vtkRInterface, vtkRCalculatorFilter, and vtkRRandomTableSource.

The VTK/Graphics/Testing/Cxx directory contains unit tests for these integration classes which illustrate the basic usage
of their interfaces. There are also Python examples located in the VTK/Examples/Infovis/Python directory.

Applications of the GNU R to VTK interface for statistical hypothesis testing are already available. More specifically, an option to perform statistical tests has recently been added to the vtkDescriptiveStatistics and vtkContingencyStatistics classes: respectively, the Jarque-Bera normality and $X^2$ independence tests. Although the test statistics themselves are directly computed in VTK, the calculation of the corresponding p-values requires that one-tailed probability values of the $X^2$ distribution be available. This functionality is readily available with GNU R, by the means of the pchisq(x,df) function which computes the CDF of the $X^2$ distribution, where each $x_i$ in the vector $x$ has $df$ degrees of freedom (plus an optional non-centrality parameter which we are not using). We have integrated this feature with the EvalScript() method defined in the VTK to GNU R interface. A single GNU R script specifying all the p-value calculations for each set of interest in the VTK filter is generated and executed. This calculation is done once for all p-values sets of interest, in order to reduce the compute time overhead of calling GNU R from VTK.

Examples that exercise and verify this functionality are available in VTK/Infovis/Testing/Cxx/TestDescriptiveStatistics.cxx and VTK/Infovis/Testing/Cxx/TestContingencyStatistics.cxx, both of which can be executed from InfovisCxxTests program in the VTK build directory.

**BUILDING VTK WITH MATLAB® INTEGRATION**

A licensed installation of the base MATLAB software package must be present on the build machine. The integration work has been tested with MATLAB 2008b and newer. On Windows machines, ensure that the full paths to the MATLAB installation's bin and bin/win32 (or bin/win64) directories are present in the system PATH environment variable.

This is normally done automatically by the MATLAB installer program on Windows. For Linux machines, ensure that the PATH environment variable contains the full path to the MATLAB installation's bin directory, and that the LD_LIBRARY_PATH environment variable contains the full path to the MATLAB installation's bin directory. For Macs, ensure that the PATH environment variable contains the full path to the MATLAB installation's bin directory, and that the DYLD_LIBRARY_PATH environment variable contains the full path to the MATLAB installation's bin/macos directory.

The CMake option to turn on the MATLAB integration code during the build is under advanced options as VTK_USE_MATLAB_MEX. If the system environment variables are set as described above, the MATLAB integration CMake macros should find everything needed to start the VTK build.

**BUILDING VTK WITH GNU R INTEGRATION**

A build from source of GNU R 2.9.0 or newer is required. Please consult http://www.r-project.org/ for downloads and build instructions. A word of caution, the GNU R build process is considerably easier on Mac and Linux than on Windows.

The CMake option to turn on the GNU R integration code during the build is under advanced options as VTK_USE_GNU_R. The CMake system will attempt to locate the R installation, but most likely some manual specification of library and include paths will be necessary. The CMAKE variable R_INCLUDE_DIR is the full path to the R installation include directory, normally under <R_ROOT_DIR>\include.

The R libraries are normally located in <R_ROOT_DIR>\bin. The CMake variable R_LIBRARY_BASE is the full path to the main R library, libR.lib (Windows), libR.dylib (Mac), or libR.so (Linux). The CMake variables R_LIBRARY_BLAS and R_LIBRARY_BLAS specify the full paths to the R Lapack and R Blas, if built separately. If these libraries do not exist on your system, just specify the same path used for R_LIBRARY_BASE in these CMake variables. The CMake variable R_LIBRARY_READLINE is necessary only if R was built with GNU READLINE support enabled. Once the above CMake variables are set, the VTK build can then be started.

**RAY CASTING ON THE GPU IN VTK**

In the July 2008 issue of the Source, Francois Bertel described the GPU ray casting volume mapper in VTKEdge that was the result of our NSF-funded project aimed at AMR visualization. We are happy to announce that due to recent funding from Ron Kinkinis at Brigham & Women’s Hospital, that volume mapper has now migrated to VTK. It is available in the VolumeRendering kit and is called vtkGPURayCastMapper. This migration included a name change (to remove the "KWE" from the class name) as well as several bug fixes. Currently, this mapper only works with NVidia cards that support the required extensions (most recent cards do) and unfortunately there is still an issue on Mac OSX.

This mapper has been added to the suite of volume mappers available in Slicer. As part of a Sandia-funded effort, this mapper is also being added to ParaView’s available volume mappers.

**IN PROGRESS**

**Source**

François Bertel

**Description**

A micro-CT scan of a trilobite fossil rendered with the GPU ray caster. Dataset courtesy of Thomas Kircher. Samples were from the University of Alaska Fairbanks Department of Geology and Geophysics collection courtesy of Dr. Sarah Fowell.
KITWARE AWARDED $600K NIH GRANT TO ADVANCE NEUROSURGICAL SIMULATION

Kitware has received a two-year grant from the National Institutes of Health (NIH) totaling more than $600,000 to focus research efforts on developing approach-specific, multi-GPU, multi-tool, high-realism neurosurgery simulation. The goal of the research is to work toward an interactive simulator that replicates future neurosurgery cases of young surgeons, enabling hospitals faced with compressed intern schedules to accelerate training and improve skills.

“Simulation has made limited inroads in neurosurgery despite its proven advantages over traditional surgical training,” said Dr. Michel Audette, R&D Engineer at Kitware. “Our objective is to empower the neurosurgical field with simulation training methods that directly correlate to better, more accurate patient care.”

Historically, simulation has over-emphasized needle insertion and failed to address requirements of surgeons-in-training on potential caseloads. Existing simulators do not ably reconcile the conflicting clinical requirements of neurosurgery; their underlying models are insufficiently sparse to resolve large tissue displacements and they are insufficiently descriptive of critical tissues at risk in each intervention.

As always, Mastering CMake includes:
- Instructions for downloading, installing, and running CMake on many platforms including Linux (UNIX), Microsoft Windows, and Mac OSX.
- Details on how CMake works and how it impacts your build scripts.
- Instructions for running the different GUIs, including command line options.
- Information on how to use CTest to perform software testing.
- A full reference of CMake’s commands and variables with descriptions.

The Fifth Edition of Mastering CMake may be purchased through Kitware's secure online store or through Amazon (ISBN-13: 978-1-930934-22-1). Please contact Kitware directly at sales@kitware.com for discounted rates on purchases of 10 or more books.
VTK USER’S GUIDE NOW AVAILABLE
Kitware is pleased to announce the release of the Eleventh Edition of the VTK User’s Guide. The VTK User’s Guide is a companion text to The VTK Textbook; while The VTK Textbook stresses algorithmic and data structure details, the VTK User’s Guide stresses how to use the software.

The Eleventh Edition features a new chapter detailing VTK’s information visualization functionality, including the table, graph, and tree data structures required to represent non-spatial data; enhanced details on 2D and 3D widgets and other forms of user interaction required to create an exploratory visualization application; a new chapter on geospatial visualization for geographically organized data; additional details on time dependent and composite data; and a discussion of advanced rendering techniques, including depth peeling for transparency.

As always, the VTK User’s Guide includes:
- A description of the structured and unstructured data types supported by VTK.
- Details on the pipeline architecture model.
- An overview of the rendering engine used to display data.
- Many examples showing the use of various scalar and vector visualization and data processing algorithms.
- Chapters dedicated to image processing and volume rendering.
- Information on how to write your own algorithms in VTK, including details on the VTK coding style.

The Eleventh Edition of the VTK User’s Guide may be purchased through Kitware’s secure online store or through Amazon (ISBN-13: 978-1-930934-23-8). Please contact Kitware directly at sales@kitware.com for discounted rates on purchases of 10 or more books.

VTK USER’S GUIDE NOW AVAILABLE
Kitware is pleased to announce the release of the Eleventh Edition of the VTK User’s Guide. The VTK User’s Guide is a companion text to The VTK Textbook; while The VTK Textbook stresses algorithmic and data structure details, the VTK User’s Guide stresses how to use the software.

The Eleventh Edition features a new chapter detailing VTK’s information visualization functionality, including the table, graph, and tree data structures required to represent non-spatial data; enhanced details on 2D and 3D widgets and other forms of user interaction required to create an exploratory visualization application; a new chapter on geospatial visualization for geographically organized data; additional details on time dependent and composite data; and a discussion of advanced rendering techniques, including depth peeling for transparency.

As always, the VTK User’s Guide includes:
- A description of the structured and unstructured data types supported by VTK.
- Details on the pipeline architecture model.
- An overview of the rendering engine used to display data.
- Many examples showing the use of various scalar and vector visualization and data processing algorithms.
- Chapters dedicated to image processing and volume rendering.
- Information on how to write your own algorithms in VTK, including details on the VTK coding style.

The Eleventh Edition of the VTK User’s Guide may be purchased through Kitware’s secure online store or through Amazon (ISBN-13: 978-1-930934-23-8). Please contact Kitware directly at sales@kitware.com for discounted rates on purchases of 10 or more books.

THE END OF THE EXPERIMENT
Back when we first created VTKEdge several years ago, we decided to experiment with a dual licensing model in an attempt to satisfy both our open source community and the reviewers of our SBIR / STTR proposals. Before that, we would either release the code into our open source projects (keeping the community happy, but making the reviewers concerned about our sanity when our revenue model started with giving everything away for free) or we would keep the code proprietary and attempt to license it (alienating our open source community, but satisfying the commercialization requirements for our proposals). After a few years we can now safely say that this experiment was a failure. Not only did we annoy our open source community by adopting a reciprocal license, we did not actually satisfy the reviewers who were now even more perplexed by a plan that included both giving away and selling the same code.

We are pleased to announce the end of this experiment. The VTKEdge toolkit has migrated from using the GPL to a BSD license compatible with VTK and ITK. Some of the VTKEdge functionality has actually been ported into VTK and ParaView (the LIC painters and the GPU ray cast mapper). As funding permits, more functionality will migrate and, eventually, VTKEdge will cease to exist. We are also happy to say that reviewers are no longer quite as bewildered by an open source business model, although it is still challenging to quantify an agency’s return on investment when the results are all distributed open source. So, if you are using one of our open source tools in your work, please let us know. Your success stories and your wish lists for new functionality can help us justify future research by showing that the impact of the effort goes far beyond Kitware.

DEVELOPER’S TRAINING WEEK
Kitware’s Developer’s Training Week will be held May 3 - 6, in Albany, NY. The course is a hands on experience covering VTK, ParaView, ITK, and CMake and is suitable for both new users of these open source projects as well as more advanced developers. Basic knowledge of C++ is recommended.

Additional course information is available on our website or by emailing courses@kitware.com. Kitware also offer customized courses at both on- and off-site locations. If you have training needs which aren't met by our standard course or if our current training dates don’t work for your organization, a customized course may be right for you. Please contact a Kitware representative to discuss customized options.

SUMMER CONFERENCES AND EVENTS
If you’re interested in meeting with a Kitware representative at one of these events, email us at kitware@kitware.com.

• Turning Images to Knowledge: Large-Scale 3D Image Annotation, Management, and Visualization
  May 9 - 12, in Ashburn, VA. This conference will stress the importance of extracting useful and biologically relevant knowledge from image contents in a systematic, comprehensive and high-throughput way. Dr. Luis Ibáñez will be presenting a paper about collaboration platforms at this year’s event. http://www.hhmi.org/janelia/conf-045.html

• CVPR 2010

• NA-MIC Project Week
  June 21 - 25, in Boston, MA. Will Schroeder, President and CEO, and Stephen Aylward, Director of Medical Imaging, will be in attendance. http://www.na-mic.org/Wiki/index.php/2010_Summer_Project_Week

• CARS 2010
OSCON 2010
July 19 - 23, in Portland, OR. Dr. Luis Ibáñez will be presenting a talk on "Educating the Next Generation of OSS Developers" on Thursday July 22 at 1:40 p.m. in D133 as part of OSCON’s first designated education track. http://en.oreilly.com/oscon2010

SIGGRAPH 2010

ICPR 2010

IEEE Cluster 2010

MICCAI 2010

NEW EMPLOYEES
Robert Maynard
Robert Maynard joined the Scientific Computing Group at Kitware in March 2010. Robert received his B.S. in Computer Science from Laurentian University in Sudbury, Ontario in May 2007. After graduating Robert spent 3 years at MIRARCO where he was the primary programmer on ParaViewGeo, a fork of ParaView designed for the mining industry. While at MIRARCO, he worked on several internal projects focusing on visualization for the mining industry.

Sébastien Jourdain
Sébastien Jourdain joined Kitware in February 2010 where he is currently working on the collaborative aspects of ParaView. Prior to joining Kitware, Sébastien developed open source software projects at Artenum for research laboratories and various French companies. Sébastien was also previously a member of the research team at INRIA-Lorraine where he worked on the LibreSource project. He received a Master's Degree in Computer Science from the ESSTIN Engineering School in France.

INTERNSHIP OPPORTUNITIES
Kitware Internships provide current college students with the opportunity to gain hands-on experience working with leaders in their fields on cutting edge problems. Our business model is based on open source software—an exciting, rewarding work environment.

At Kitware, you will assist in the development of foundational research and leading-edge technology across our five business areas: supercomputing visualization, computer vision, medical imaging, data publishing and quality software process. We offer our interns a challenging work environment and the opportunity to attend advanced software training. Apply by sending your resume to internships@kitware.com.

EMPLOYMENT OPPORTUNITIES
Kitware has an immediate need for talented Software Developers, especially those with experience in Computer Vision, Scientific Computing and Biomedical Imaging. Qualified applicants will have the opportunity to work with leaders in computer vision, medical imaging, visualization, 3D data publishing and technical software development.

Kitware team members enjoy our small company environment, flexibility in work assignments and high levels of independence and responsibility. We offer comprehensive benefits including: flexible work hours; six weeks paid time off; a computer hardware budget; 401(k); health insurance; life insurance; short- and long-term disability; visa processing; and free drinks and snacks.

Qualified candidates, especially those with experience in computer vision, scientific computing, and biomedical imaging, are encouraged to send their cover letter and resume to jobs@kitware.com.

In addition to providing readers with updates on Kitware product development and news pertinent to the open source community, the Kitware Source delivers basic information on recent releases, upcoming changes and detailed technical articles related to Kitware’s open-source projects. These include:

- The Visualization Toolkit (www.vtk.org)
- The Insight Segmentation and Registration Toolkit (www.itk.org)
- ParaView (www.paraview.org)
- The Image Guided Surgery Toolkit (www.igstk.org)
- CMake (www.cmake.org)
- CDash (www.cdash.org)
- MIDAS (www.kitware.com/midas)
- BatchMake (www.batchmake.org)
- VTKEdge (www.vtkedge.org)

Kitware would like to encourage our active developer community to contribute to the Source. Contributions may include a technical article describing an enhancement you’ve made to a Kitware open-source project or successes/lessons learned via developing a product built upon one or more of Kitware’s open-source projects. Authors of any accepted article will receive a free, five volume set of Kitware books.

Kitware’s Software Developer’s Quarterly is published by Kitware, Inc., Clifton Park, New York.


Creative Director: Melissa Kingman, www.elevationda.com
Graphic Design: Steve Jordan
Editor: Niki Russell

Copyright 2010 by Kitware, Inc. or original authors.

The material in this newsletter may be reproduced and distributed in whole, without permission, provided the above copyright is kept intact. All other use requires the express permission of Kitware, Inc. Kitware, ParaView, and VolView are registered trademarks of Kitware, Inc. All other trademarks are property of their respective owners.

To contribute to Kitware’s open source dialogue in future editions, or for more information on contributing to specific projects, please contact the editor at editor@kitware.com.