Kitware SOURCE

SOFTWARE DEVELOPER'S QUARTERLY

Issue 15• Oct 2010

TABLE OF CONTENTS

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, Brad King, Marcus Hanwell and Bill Hoffman discuss the DCVS tools Kitware is using to support collaborative, distributed development teams. Sophie Chen provides a tutorial for creating your own VolView and V3D plugins. David Gobbi provides an overview of the recent overhaul to VTK's wrappers. David Cole introduces a new a feature in CDash which allows users to request project builds done on any client. And David DeMarle, Jonathan Woodring and James Ahrens explain the addition of adaptive streaming algorithms in ParaView and VTK.

Readers will also notice a new section in this issue called the "Community Spotlight" highlighting community members who use Kitware tools in their own development environments. Cyrus Aidun, Daniel Reasor, Jonathan Clausen and Jingshu Wu share their experiences using ParaView to to simulate thousands of deformable particles. And Luis Pérez Tato discusses how his company, Iturribizia, utilizes VTK to solve structural analysis problems. To be featured as a Community Spotlight in the future, email editor@kitware.com.

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.

RECENT RELEASES

ITKV4 DEVELOPMENT

Over the last quarter, the ITK development team released the first two Alpha versions of ITKv4. ITKv4 is a major refactoring of the ITK toolkit that introduces improved wrapping for other languages, a modular architecture and revisions to many of ITKs components. These two releases were intended to perform a general code clean up, dropping the tricks to support now-defunct compilers used in the past while paving the way for major refactoring activities to commence. The next decade of ITK has begun.

Details

One of the most significant operational changes is that the source code of ITK was moved to a Git repository and a new development workflow has been put in place in order to integrate the teams that are collaborating in this new version of the toolkit. The major changes introduced in these two releases are described below.

ITKv4-Alpha-01

The following compilers were deprecated: Borland 5.5, Visual Studio 6.0 and 7.0, SGI CC, Sun CC 5.6, Metrowerks. Source code that was intended solely to support these compilers was also removed. The original Statistics Framework was removed and replaced with the one that was refactored in 2007. Multi-threaded implementations of the image registration metrics, mean squares and Mattes mutual information, replaced those which were preexisting.

Source code marked as deprecated in previous releases was removed, classes implemented to consolidate morphology were integrated into the toolkit and the consistent computation of pixel-centered coordinates is now enforced. Additionally, several CMake options were removed; in particular, those intended to alternate between the changes made above and pre-existing code.

ITKv4-Alpha-02

For Alpha version 2, the source code was processed using Uncrustify to reformat the coding style, the openjpeg library was updated to openjpeg-v2 version from July 2010 and the jpeg library was updated from version 6b to version 8b.

Additionally, GDCM (the library that provides DICOM support in ITK) was updated to GDCM 2.0.16 and an ImageIO class specialized on managing JPEG2000 files was added from an Insight Journal paper contribution.

Details on these changes can be found in the ITK Wiki. Please check itk.org and the Wiki for continuous updates on ITKv4.

VTK 5.6.1 RELEASED

Kitware is pleased to announce the release of VTK 5.6.1, a patch release over 5.6.0 with a small set of improvements.

VTK now includes NetCDF 4.1.1 allowing users to take advantage of the latest enhancements to NetCDF. Additionally, the C++ version of the library is also included. A new file format, MPAS, has been added; it is a netcdf-based reader for ocean and atmosphere datasets.

Charts have significant improvements in axes layout and the ability to display multiple data series (up to four) in the same plot. Initial support for linked selection has been added to the charts in VTK and ParaView. Support for alpha blending in parallel coordinate plots has also been added.

We've cleaned up VTK's build and install rules so that developers don't have to worry about 'rpath' anymore. Additionally, "make install" has been vastly improved on all platforms.

As always, we value your feedback. Please report any issues on the VTK mailing list or the VTK bug tracker.

PARAVIEW 3.8.1 RELEASED

Kitware is pleased to announce the release of ParaView 3.8.1, a patch release over 3.8.0 with a small set of improvements.

ParaView now comes HDF5 1.8.5. This enables developers to easily create readers and plugins for ParaView that use newer versions of HDF5. We are now including the C++ and the high level (hl) versions of the library for those who wish to use their newer API.

Among the improvements are two new file formats. The first is a netcdf based reader for MPAS format ocean and atmosphere datasets. The second, available in source form only, is a reader plugin that allows ParaView to read multi-resolution wavelet compressed VAPOR data format files. Thanks to John Clyne and Dan Legreca for contributing the VAPOR plugin.

We have improved the support for animation scripting through Python. The new API is more coherent with the rest of the ParaView Python API, avoiding the need for script writers to know minute details regarding how to create animations using proxies. Tracing capabilies for animation were also revised.

The GPU volume ray cast mapper for voxel datasets now works in tile display mode. In keeping with our ongoing improvements of ParaView's charting capabilities, we have included a few fixes regarding labels and axes placements.

ParaView build and install rules have been cleaned up so developers no longer have to worry about 'rpath' and "make install" has been improved on all platforms.

This release includes a material interface extraction filter that takes a cell data volume fraction and generates a poly-data surface. It also performs connectivity on the particles and generates a particle index as part of the cell data of the output. The filter computes the volume of each particle from the volume fraction and is accessible from the Material Analysis category in the Filters menu.

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.

DISTRIBUTED VERSION CONTROL: THE FUTURE OF HISTORY

Collaborative software development, an approach Kitware and open source communities have used for many years, is now a mainstream development model [1] [2]. Developer teams are often distributed geographically and may even work for different organizations. Kitware frequently teams with its customers to develop software solutions. New tools and processes are becoming available to manage this collaborative development model. Distributed Version Control Systems (DVCS) are especially exciting. According to Joel Spolsky [3], "This is possibly the biggest advance in software development technology in the ten years I've been writing articles here."

Over the last nine months at Kitware we have transitioned most of our work to Git, a popular DVCS tool [10]. Although we have not yet fully realized all of the benefits, this technology will address important issues. In particular, it will help us further engage our customers and the open source community, and will improve our release process.

DVCS tools engage our customers and users as welcome participants in our open source software development model. They facilitate code reviews, allow easy incorporation of contributed modifications and shorten release cycles. Furthermore, "social coding" sites such as GitHub [7] and Gitorious [8] allow everyone to share their work outside the central repositories.

We are very excited to improve our release process using DVCS capabilities. Our new process allows all developers to take part in release preparation by deciding what changes are release-ready as they are developed. A release-ready version of the software grows in parallel with the development version. We avoid building up an inventory of partially finished work in an otherwise releasable version. This streamlines the release process and allows for stable and frequent releases.

A BRIEF HISTORY OF VERSION CONTROL

Prior to 1986, two file locking version control systems which stored the changes for a file existed: SCCS, developed at Bell Labs, and RCS. If a developer wanted to change a file, he or she had to issue the "edit" command on the file and, if no other developer was working on that file, she would get a writable version of the file to edit. After completing edits on that file, the developer would check the changes in, and this would release the lock.

Having a history for each file was helpful and locking the files made sense in order to prevent two or more developers from making the same or conflicting changes. However, in practice this did not scale very well beyond a few developers. When a developer left a file in edit mode and went on vacation or was away from the office, the edit locks were forcefully and "hackishly" taken by others on the team so that work could continue. The hand merges that would happen after the developer returned were often painful and time consuming.

In 1986, the Concurrent Versions System (CVS) was created to address many of the shortcomings of previous version control systems. CVS offered concurrent edits on files managed by a central repository using a client/server model. Changes to

files were integrated together during an update of a check-out prior to a commit to the repository by the last person to commit his/her changes on that file. This encouraged premature commits that often forced unfinished changes on the whole team. However, the model was certainly much better than the file-locking days of SCCS and RCS.

When CVS first came out it was a hard sell to the development community. Developers felt secure with systems based on file locks. The idea that two people could work on one file at the same time and that changes would be integrated automatically was "just plain crazy". However, over time CVS became the accepted norm for software development. In 2000, the Subversion project (SVN) was created to replace CVS. SVN provided atomic whole-tree commits, versioned file properties and a more solid implementation of CVS's centralized repository model.

A new model for version control, Distributed Version Control Systems (DVCS), is now unseating centralized systems as the standard for software development. These systems offer concurrent repositories, network-free access to local-disk repositories (Figure 1) and they enable new collaboration models with non-linear history and multiple shared repositories (Figures 2 and 5). In this article we describe the power of DVCS as a version control system and explain how we are using it to improve our collaborative development model.

Figure 1 – Access Information at All Times

NOTATION & TERMINOLOGY

This article uses conventions and terminology from Git, but the discussion applies to DVCS in general. Our figures denote history using a directed acyclic graph as shown in Figure 2. Nodes (circles) represent versions and directed edges point at prior versions. The subgraph reachable from any given version represents its history.

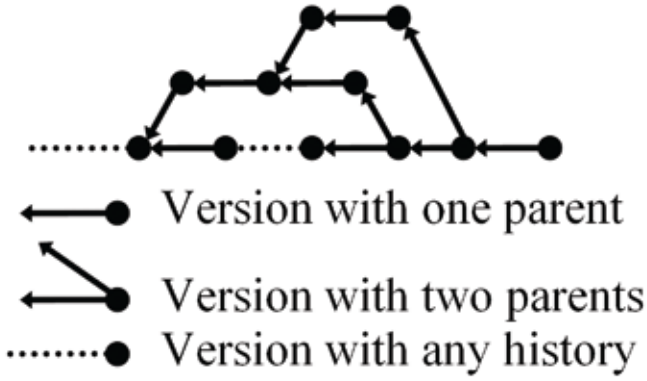


Figure 2 – Visualizing History

COLLABORATION TASKS

We divide collaborative development into three basic version control tasks:

- Create: Record a new version with meta-data describing the changes;
- Share: Publish new versions for other developers to see;
- Integrate: Combine changes from multiple versions into another new version.

The following reviews each task in more detail.

Create

Figure 3 shows the basic workflow we each follow to create new versions. First, we *checkout* a version from the repository on which to base changes. Then, we *edit* the content in our work tree to produce a new version. Finally, we *commit* the new version to the repository. All version control systems provide this basic workflow by definition.

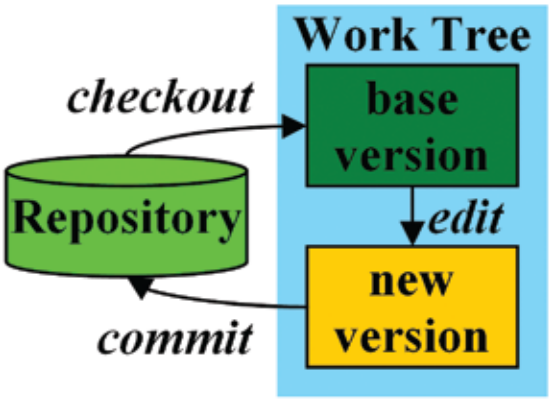


Figure 3 – Basic Workflow

Share

We share versions through repositories. Figure 4 shows the traditional model in which each developer has only a work tree. The checkout and commit operations share all versions through a single repository.

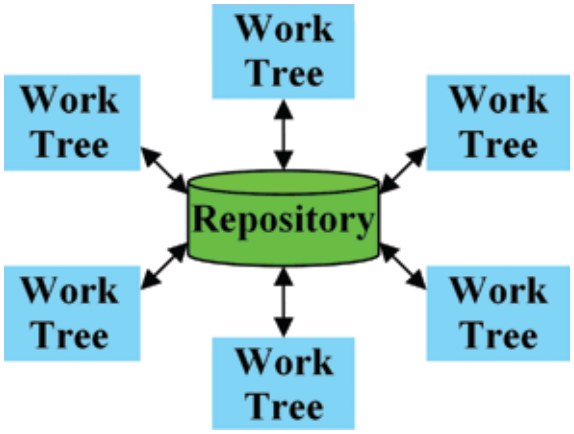


Figure 4 – Single Centralized Repository

Figure 5 shows the distributed model in which each developer has a private repository and work tree. The checkout and commit operations occur locally. Sharing occurs arbitrarily among repositories through separate operations such as *push*, *fetch*, and *pull*.

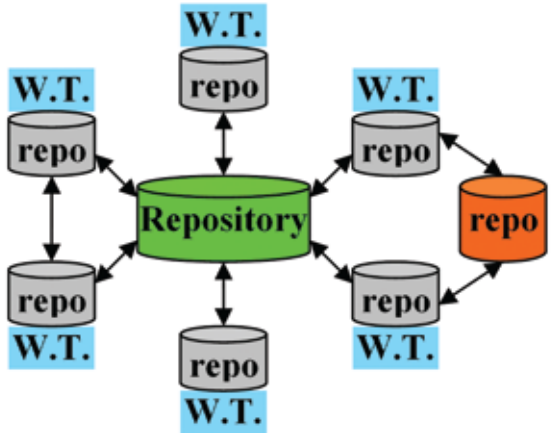


Figure 5 – Concurrent Distributed Repositories

Integrate

Figure 6 shows a case in which work has *diverged* because two developers independently created versions *C* and *D* based on version *B*. Assume version *D* has been published and we must integrate its changes with those made in version *C* to produce a new version.

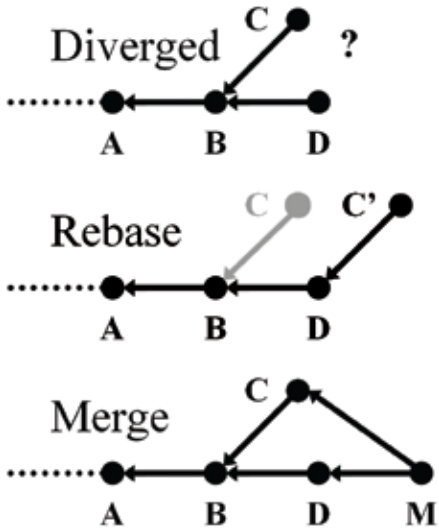


Figure 6 – Integrate by Rebase or Merge

The figure illustrates two approaches. One wherein we identify the changes made by *C*, originally based on *B*, and *rebase* them on *D* to create a new version *C'*. And another where we *merge* versions *C* and *D* together into a new version, *M*, that combines their changes.

Both approaches integrate the same changes into a single new version, either *C'* or *M*, but record different history behind said version.

In traditional version control systems the commit operation *automatically rebases* changes in new versions on the latest version in the repository. If the rebase fails it asks the user to update the work tree and integrate the changes before committing. In distributed version control systems, *rebase* and *merge* are explicit operations separate from commit.

COLLABORATION WORKFLOWS

All developer communities establish a *workflow* involving these three version control tasks in order to collaboratively develop and maintain their projects. A workflow determines when new features are introduced, how bugs are fixed and how releases are prepared and maintained. Traditional version control systems severely limit possible workflows by inseparably combining all three version control tasks under the commit operation in a single repository. Distributed version control systems provide separate operations for the three tasks and thus support many workflows [4].

Rebase v. Merge Workflows

No matter who creates changes or where they are shared, they eventually need to be integrated together. At this point workflows are distinguished by their approach to integration, either *rebase* or *merge*. Figure 7 illustrates both approaches with a series of commits belonging to three *topics*: *feature a*, *feature b*, and a *bug fix*. It also includes the creation and maintenance of a *release* branch.

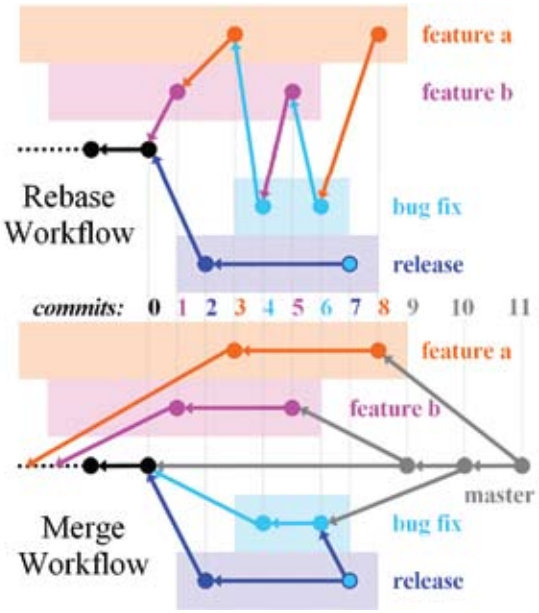


Figure 7 – Collaboration Workflow by Rebase or Merge

In the rebase workflow, each commit is rebased on top of whatever happens to be the latest version at the time it is published; there is no record of the original base on which it was developed. The commits from all topics are intermixed with one another in history. The history of the release branch has no indication that its second commit is a copy of the bug fix changes.

In the merge workflow, each commit is preserved on its original base. The commits belonging to each topic appear contiguously in history. There is an explicit *merge commit* recording the integration of each topic. The second commit on the release branch explicitly merges the bug fix topic.

Traditional version control systems automatically *rebase* every commit, and therefore support only the rebase workflow. Distributed version control systems support both rebase and merge workflows.

A "Branchy" Workflow

In the past, when using a central repository with a single development branch, the authors have seen work come to a halt when some bad code was committed. No other work could continue until the issue was fixed. With the use of DVCS developers can start new work from a stable working code base and no longer base new work on a moving target.

The use of merge commits to integrate work provides greater flexibility and motivates the use of a "branchy" workflow with DVCS tools [5]. This workflow defines two types of branches: topic and integration. *Topic* branches contain a contiguous sequence of commits developing a specific feature or bug fix. *Integration* branches contain *merge* commits to integrate topic branches together.

In the bottom of Figure 7 each of *feature a*, *feature b*, and *bug fix* is a topic branch, and *master* and *release* are integration branches. Integration branch heads are published in a designated official repository. Topic branch heads are not named explicitly in the official repository, but appear in the history of integration branches that merge them.

Each integration branch has a specific purpose such as maintenance of a current release (typically called *maint* or *release*), preparation of a future release (typically called *master*) or bleeding-edge development (typically called *next*, as in the next topics to evaluate). Each topic branch starts from the most stable integration branch to which it will be merged. Typically this is *master* for new features and *release* for bug fixes.

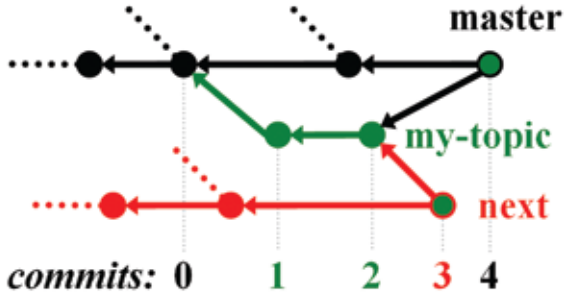


Figure 8 – Multiple Integration Branches

Figure 8 shows the use of two integration branches, *master* and *next*, while developing a topic branch, *my-topic*. The head of master is *commit* (0) when the topic starts. Commits (1) and (2) develop the topic. Merge commit (3) integrates the topic into *next* which is then published for evaluation and testing by others. Later, when the topic is deemed stable, merge commit (4) integrates the topic into *master* for publication.

Throughout this workflow the *master* branch remains stable because topics are merged into it only after they have been evaluated on *next*. Since new topics are started from *master* they have a stable base version on which to work regardless of whether unstable topics have been merged to *next*. A new stable topic may be merged back to *master* at any time independent of other (unstable) topics.

MANAGING RELEASES

Release management has two parts: preparing a new release and maintaining an existing release. In the past, Kitware placed responsibility for both parts on a release manager. We prepared new releases using a "freeze" period during which the release manager had exclusive commit access to stabilize the trunk, often by cleaning up unfinished work, before creating the release. The release manager then maintained the release branch by manually copying bug fixes from the development trunk. With this approach, development stalled during freeze periods and maintenance of releases became increasingly burdensome on the release manager as the trunk diverged over time.

Kitware's new release process is based on the DVCS branchy workflow in Figure 8. New topics are merged to *next* for evaluation and testing and only stable topics are merged into *master*, keeping it release-ready at all times. This approach amortizes the cost of release preparation over the development cycle, distributes the workload to all developers and separates release scheduling from feature development.

We now manage releases as shown in Figure 9 (for simplicity we omit *next* from the figure but we use it to test changes before merging to *master*).

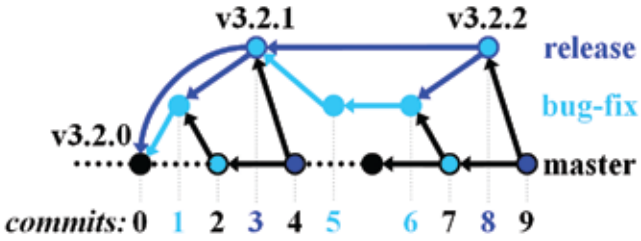


Figure 9 – Release Maintenance

The release manager tags a new release directly on *master* (0) and development proceeds normally. A developer starts a bug-fix topic (1) from the *released version* and *merges it into master* (2), making the "release+fix" version (1) available without any action by the release manager. Then the release manager merges the fix into the *release* branch (3), tags a patch release, and *merges release into master* (4), making the tag reachable. The process repeats with another *bug-fix* (5 and 6), merge to *master* (7), merge to *release* (8), tag and merge of *release* into *master* (9).

This maintenance approach relieves the release manager of all tasks except trivial merge and tag operations. Release tags are always reachable from *master* so there is no need for a separate named branch for every new release. DVCS allows developers to commit bug fixes directly on the releases that need them.

CONCLUSION

Some DVCS concepts may seem overly complex and irrelevant to many developers, but are invaluable to others. Not only are they here to stay, but they'll improve productivity and reduce waste in the software development process. As with the advent of centralized concurrent version systems, a period of education and exploration is required to fully take advantage of this new technology.

Kitware has standardized on Git, our DVCS of choice. Although powerful, some consider Git to be complicated to learn. It was even introduced at the now-famous Google Tech Talk by Linus Torvalds as "expressly designed to make you feel less intelligent than you thought you were" [9]. We encourage the reader to stay the course and hang in there as the benefits really are numerous.

Kitware is not alone in embracing DVCS. There are many other software companies exploring the features of DVCS. For example, we are investigating use of Gerrit, a code review tool developed by Google engineers to facilitate Android development [6]. It provides tight integration with Git and a web interface for performing online code reviews. The

ITK project is using it experimentally, and several developers are evaluating it internally. Gerrit can combine a human element with automated testing and multiple integration branches to provide us with a very effective workflow to collaboratively develop complex software with contributors from around the world.

Other online tools such as Gitorious and GitHub are already being used by developers outside Kitware to develop bug fixes and new features for Kitware-hosted projects like VTK, ParaView, ITK and CMake.

We hope to have armed you with some basic concepts and terminology associated with DVCS, while presenting some of the exciting new workflows that DVCS makes possible. Kitware is very excited about this technology and is looking forward to reaping the full benefits.

To our customers and co-developers: we look forward to releasing our work frequently and incorporating your contributions quickly and reliably.

REFERENCES

[1] http://en.wikipedia.org/wiki/Collaborative_software_development_model

[2] http://www-01.ibm.com/software/info/features/collaboration/main.html

[3] "Distributed Version Control is here to stay, baby", Joel on Software, March 17, 2010 http://joelonsoftware.com/items/2010/03/17.html

[4] "Distributed Workflows", Pro Git, ch. 5, http://progit.org/book/ch5-1.html

[5] "Git help workflows", http://www.kernel.org/pub/software/scm/git/docs/gitworkflows.html

[6] "Gerrit Code Review", http://code.google.com/p/gerrit/

[7] "GitHub – Social Coding", http://github.com/

[8] "About Gitorious", http://gitorious.org/about

[9] "Linus Torvalds on git", Google Tech Talk, May 3, 2007, https://git.wiki.kernel.org/index.php/LinusTalk200705Transcript

[10] "Git – the fast version control system", http://git-scm.com/

Marcus Hanwell *is an R&D engineer in the scientific visualization team at Kitware, Inc. He joined the company in October 2009, and has a background in open source, Physics and Chemistry. He spends most of his time working with Sandia on VTK, Titan and ParaView.*

Bill Hoffman *is currently Vice President and CTO for Kitware, Inc. He is a founder of Kitware, a lead architect of the CMake cross-platform build system and is involved in the development of the Kitware Quality Software Process and CDash, the software testing server.*

Brad King *is a technical developer in Kitware's Clifton Park, NY office. He led Kitware's transition to distributed version control, converted many of our project histories to Git, and conducted training sessions.*

INSIGHT TOOLKIT PLUG-INS: VOLVIEW AND V3D

VolView and V3D are applications for visualization and analysis of three-dimensional images. They both have tools which allow users to filter and analyze image data. The two applications serve two different niches: VolView was created with radiologists in mind, while V3D caters primarily to microscopists. However, a powerful part of both biomedical imaging tools is support for user defined extensions via custom plug-ins. This support allows users to extend how input data is filtered. This quick guide will help you get started with your own VolView and/or V3D plug-in.

Software applications such as Slicer, SNAP, Analyze, VolView, SCIRun, GoFigure and V3D use ITK filters as plug-ins to add desirable additional capability to their image analysis application of choice, thus removing the need to rewrite existing algorithms for each new piece of software while stamping out the hassles of requesting usage-permission.

ITK's qualifications for use in scientific research makes it important for developers to make the most of ITK's imaging tools and offer tailored combinations to those who desire them. The following table compares some of the main features of VolView [1] and V3D [2].

	VolView	V3D
Creator	Kitware, Inc.	Howard Hughes Medical Institute: Janelia Farm Research Campus, Hanchuan Peng, et al.
Platforms	Windows XP, Vista Mac Linux	Windows XP, Vista Mac Linux
Intent	3D image analysis for medical, scientific and engineering research communities. Interactively render, reformat, annotate, measure, animate and capture/print volumetric and image data.	3D image analysis such as cell segmentation, neuron tracing, brain registration, annotation, quantitative measurement, data management and more.
Download	http://www.kitware.com/products/volviewdownload.html	http://penglab.janelia.org/proj/v3d

Structure of Plug-ins

Typically, a plug-in for V3D and VolView consists of source code compiled and packaged as a shared library. The plug-in name determines the name of the shared library used for deployment as well as the name of the plug-in initialization function. The shared library is copied into a specific directory of the VolView or V3D binary installation. No libraries from VolView or V3D are required in the process. Plug-in developers only need a set of source code headers defining the plug-in API offered by the application. This is essentially the set of data structures and function calls by which the plug-in communicates with the application.

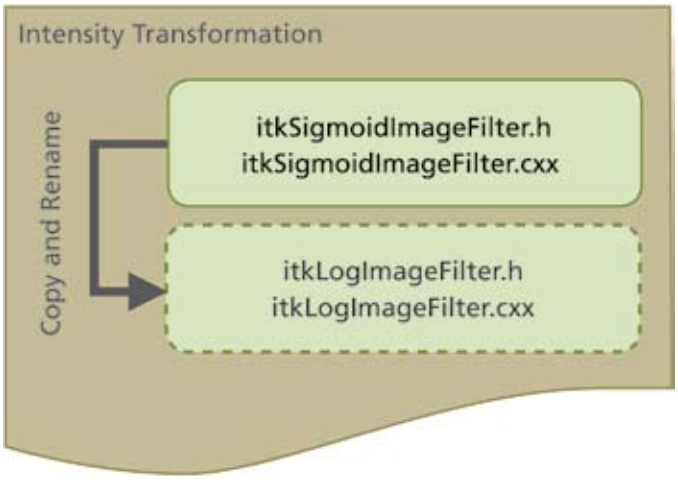
A V3D PLUG-IN

The development of ITK plug-ins for V3D serves two purposes: 1) exposing ITK functionalities to researches who analyze microscopy data and 2) uncovering the areas in which ITK requires improvements in order to better serve the microscopy community. ITK filter plug-ins were added to V3D via a collaborative effort between Kitware and Janelia Farm (HHMI).

The simplest way to implement a plug-in is to copy and paste an existing V3D plug-in and modify two methods. More advanced plug-ins, typically those requiring more than one filter, may need to be modified further. For our V3D plug-in example, we will use the existing itkSigmoidImageFilter, which can be found under the Intensity Transformation directory to create another plug-in, itkLogImageFilter. For V3D, the V3DPluginCallback is used to get data structures and callbacks.

Create a V3D Plug-in

Copy and paste existing plug-in header and source files to the binary directory where plug-ins are set up in your system. An example path is: src/ITK-V3D-Plugins/Source/IntensityTransformations. Change the file names to correspond to the goal image filter.



Then find instances in files where filter references and names ought to be replaced. In the SetupParameters section, adjust your filter's parameters, or if the section does not exist, refer to the SetUpParameters() below. If you're uncertain about a default value use your best judgment or communicate with an appropriate end-user for a general value. The value chosen should reflect some noticeable changes in an image upon testing.

```
void Execute
 (const QString &menu_name, Qwidget 50  *parent)
 {
 this->Compute();
 }

virtual void ComputeOneRegion()
  {

   this->m_Filter->SetInput
   ( this->GetInput3DImage() );

   if( !this->ShouldGenerateNewWindow() )
   {
   }

   this->m_Filter->Update();
  }

virtual void SetupParameters()
  {
   //
   // These values should actually be provided by
   // the Qt Dialog...
   // just search the respective .h file for the
   // itkSetMacro for parameters
   this->m_Filter->SetFullyConnected( true );
   this->m_Filter->SetBackgroundValue( 0 );
   this->m_Filter->SetForegroundValue( 100);
   this->m_Filter->SetNumberOfObjects( 3 );
   this->m_Filter->SetReverseOrdering( false );
   this->m_Filter->SetAttribute( 0 );
  }
```

A VOLVIEW PLUG-IN

For this example, a plug-in named vvITKGradientMagnitude will be deployed in a shared library: libvvITKGradientMagnitude.so in Unix/OsX and vITKGradientMagnitude.dll on MS Windows. The initialization function is vvITKGradientMagnitudeInit(). The result of the example will be an implementation of a simple ITK based filter with only one GUI parameter. The example may be adapted to most other toolkits or C/C++ implementation.

Given the similar structure of V3D, the directions in this VolView example should be generic enough to be applied to a V3D plug-in with the respective style/naming differences.

Communication between the plug-in and the application is facilitated by a public header file that defines the data and GUI structures. The plug-in developer simply implements the methods that are defined within the header file.

Initialization function

A plug-in's initialization function must conform to a particular API. For our particular example, this would be:

```
extern "C"
{
   void VV_PLUGIN_EXPORT vvITKGradientMagnitudeInit(
     vtkVVPluginInfo *info)
   {
   }
}
```

where the symbol VV_PLUGIN_EXPORT and the structure vtkVVPluginInfo are defined in the public header file, vtkVVPluginAPI.h. This initialization function will be invoked by VolView at start-up -- after the shared library has been dynamically loaded.

Content

Below is the typical content of the public header file, vtkVVPluginAPI.h.

Call macro vvPluginVersionCheck() to verify the plug-in/API conforms to current version of VolView's binary distribution. A plug-in cannot be executed if the versions do not match, and VolView displays an error message at run-time to indicate this when necessary.

```
vvPluginVersionCheck();
```

Information Structure is initialized. Setup Information does not change.

```
// Setup Information
```

ProcessData is set to the pointer of the function that will perform the computation on the input data. This allows for freedom in the implementation of the function. This is further covered in the next section.

```
info->ProcessData = ProcessData;
```

Similarly, UpdateGUI is also set to a function pointer.

```
info->UpdateGUI = UpdateGUI;
```

SetProperty() is used to define general properties of the plug-in – some of these properties are simply informative text that is displayed on the GUI (i.e. textual name of the plug-in, terse and extended documentation). Properties are identified by tags to further enforce the decoupling between the internal representation of information in VolView and the structure of code in the plug-in. Other non-GUI properties are also set with this method.

```
//Setup Information - SetProperty()
```

The tag VVP_NAME specifies that the string being passed as third argument of the SetProperty() method should be used for the text label of the plug-in in the GUI. VVP_GROUP specifies the grouping of the filter within the plug-in menu, and VVP_TERSE_DOCUMENTATION provides a short description of the plug-in.

```
info->SetProperty( info, VVP_NAME, "Gradient
Magnitude IIR (ITK)");

info->SetProperty( info, VVP_GROUP, "Utility");

info->SetProperty( info, VVP_TERSE_DOCUMENTATION,
  "Gradient Magnitude Gaussian
  IIR");
```

The tag VVP_FULL_DOCUMENTATION specifies the complete description string.

```
info->SetProperty( info, VVP_FULL_DOCUMENTATION,
"This filter applies IIR filters to compute the
equivalent of convolving the input image with the
erivatives of a Gaussian kernel and then computing
the magnitude of the resulting gradient.");
```

Other tags are used to specify:

Whether this filter is can perform in-place processing;

```
info->SetProperty( info,
  VVP_SUPPORTS_IN_PLACE_PROCESSING, "0");
```

Whether this filter supports data streaming (processing in chunks);

```
info->SetProperty( info,
  VVP_SUPPORTS_PROCESSING_PIECES, "0");
```

And other information about the filter implementation.

```
info->SetProperty( info, VVP_NUMBER_OF_GUI_ITEMS,
  "1");

info->SetProperty( info, VVP_REQUIRED_Z_OVERLAP,
  "0");
```

Memory consumption is an important consideration for processing. By providing an estimated number of bytes to be used per voxel of the input dataset:

```
info-> SetProperty( info,
  VVP_PER_VOXEL_MEMORY_REQUIRED, "8");
```

Memory consumption can be estimated. VolView will use this factor to ensure that the system has enough memory for completing the plug-in processing and for determining if the undo information can be kept. Note that this estimate is not based on the size of the final dataset produced as output, but on the total amount of memory required for intermediate processing. In other words, it should provide the peak of memory consumption during the plug-in execution.

The ProcessData() Function

The ProcessData() function performs the filter computation on the data. The function signature of ProcessData() is:

```
static int ProcessData( void *inf,
  vtkVVProcessDataStruct *pds)
```

where the first argument is a pointer to a vtkVVPluginInfo structure which can be downcast to a vtkVVPluginInfo pointer using:

```
vtkVVPluginInfo *info = (vtkVVPluginInfo *) inf;
```

In this assignment, the right hand side is a structure, vtkVVProcessDataStruct, that carries information on the data set to be processed. This information includes: the actual buffer of voxel data, the number of voxels along each dimension in space, the voxel spacing and the voxel type.

The vtkVVProcessDataStruct also contains the members inData and outData, which are pointers to input and output data sets, respectively. ProcessData() extracts the data from the inData pointer, processes it, and stores the final results in the outData buffer.

ProcessData() Starting Code

The typical ProcessData() starting code of this function extracts meta information about the data set from the vtkVVProcessDataStruct and vtkVVPluginInfo structures. For example, the following code shows how to extract the dimensions and spacing of the data.

First, set up a data structure.

```
SizeType size;
IndexType start;
double origin[3];
double spacing[3];

size[0] = info->InputVolumeDimensions[0];
size[1] = info->InputVolumeDimensions[1];
size[2] = pds->NumberOfSlicesToProcess;

for( unsigned int i=0; i<3; i++ )
  {
  origin[i] = info->InputVolumeOrigin[i];
  spacing[i] = info->InputVolumeSpacing[i];
  start[i] = 0;
  }
```

Image data can be imported into an ITK image using the itkImportImageFilter.

```
RegionType region;
region.SetIndex( start );
region.SetSize( size );
m_ImportFilter->SetSpacing( spacing );
m_ImportFilter->SetOrigin( origin );
m_ImportFilter->SetRegion( region );
m_ImportFilter->SetImportPointer( pds->inData,
  totalNumberOfPixels, false );
```

The output of the import filter is then connected as the input of the ITK data pipeline and the pipeline execution is triggered by calling Update() on the last filter.

```
m_FilterA->SetInput( m_ImportFilter->GetOutput() );
m_FilterB->SetInput( m_FilterA->GetOutput() );
m_FilterC->SetInput( m_FilterB->GetOutput() );
m_FilterD->SetInput( m_FilterC->GetOutput() );
m_FilterD->Update();
```

Finally the output data can be copied into the pointer provided by VolView. This is typically done using an ITK image iterator that will visit all the voxels.

```
outputImage = m_Filter->GetOutput();
typedef itk::ImageRegionConstIterator
< OutputImageType > OutputIteratorType;
OutputIteratorType ot( outputImage,
  outputImage->GetBufferedRegion() );
OutputPixelType * outData =
  static_cast< OutputPixelType * >( pds->outData );
ot.GoToBegin();
while( !ot.IsAtEnd() )
  {
  *outData = ot.Get();
  ++ot;
  ++outData;
  }
```

When memory consumption is critical, it is more convenient to actually connect the output memory buffer provided by VolView to the output image of the last filter in the ITK pipeline. This can be done by invoking the following lines of code before executing the pipeline.

```
m_FilterD->GetOutput()->SetRegions(region);
m_FilterD->GetOutput()->GetPixelContainer()
->SetImportPointer(
  static_cast< OutputPixelType * >( pds->outData ),
  totalNumberOfPixels, false);
m_Filter->GetOutput()->Allocate( );
```

The current distribution of ITK provides support for not rewriting this same code for each new plug-in. A templated class containing this code is available in the current distribution of ITK in the InsightApplications module. New plug-ins only need to define their own ITK pipelines and invoke the methods of the base class in the appropriate order.

Refreshing the GUI

After the source code has been packaged into a shared library, it should be deposited into the plug-ins directory: VolView 3.2/bin/Plugins. In order for the plug-in to load, the GUI needs to be refreshed by re-scanning all plug-ins. A circular arrow next to the filters selection menu will refresh the filters list.

Image processing algorithms can take considerable time to execute on 3D data sets. It is important to provide user feedback as to how the processing is progressing and to allow the user to cancel an operation if the total execution time is excessively long. Calling the UpdateProgress() function of the vtkVVPluginInfo structure from within the ProcessData() function accomplishes this:

```
float progress = 0.5; // 50% progress
info->UpdateProgress( info, progress,
  "half data set processed");
```

This function provides feedback to the VolView GUI allowing VolView to update the progress bar and set the status bar message. The frequency with which the UpdateFunction should be called should be well balanced. If it is invoked too often, it will negatively impact the performance of the plug-in -- a considerable amount of time will be spent in GUI refreshing. If it is not called often enough, it may produce the impression that the processing is failing and that the application is no longer responding to user commands.

A detailed skeleton plug-in and a more contextual version of this guide can be found starting on page 45 in the VolView Users Manual, available at kitware.com/volview.

REFERENCES

[1] Download VolView from: kitware.com/volview
[2] Download V3D from: http://penglab.janelia.org/proj/v3d

Sophie Chen *recently completed her second summer as a Kitware intern where she worked under Luis Ibáñez, Wes Turner and Harvey Cline on programming algorithms, ITK and VTK. Sophie is a senior at RPI where she is working toward an IT degree in Managing Information Systems.*

VTK WRAPPER OVERHAUL 2010

The VTK wrappers received their last major overhaul in 1998 and have done well to serve the VTK community. In recent years, however, the wrappers have started to show some cracks, particularly as new types such as vtkVariant and vtkUnicodeString have been introduced to VTK but have not been made available in the wrappers. One reason for the wrappers' slow development compared to the rest of VTK is undoubtedly the "intimidation factor" of the wrapper-generator code, which is very complex and lacking in documentation.

VTK wrappers were recently overhauled again. The four main goals of overhaul project were as follows:

- Cleaning up the wrapper-generator code by removing hard-coded hexadecimal constants, reducing the use of global variables, and improving code documentation;
- Proper wrapping of vtkStdString, which is a crucial interface type that is only partly wrapped;
- Wrapping vtkVariant and other new VTK types in Python;
- And eliminating the need for BTX/ETX "unwrappable section" markers in the code.

The overarching goal is to provide a new foundation for the wrapper-generators that will make it easier to move the wrappers forward.These changes have been made while maintaining backwards compatibility with existing Tcl, Python, and Java programs which use VTK. The old wrapper-generator code has not been replaced; it has only been cleaned up and enhanced.

WRAPPER PRIMER

To provide some background, the wrapper-generator code consists of a "front-end" parser that reads the VTK header files and stores the class declarations in simple data structures, and three "back-ends" that read those data structures and generate the wrapper code for each of the wrapper languages. Most of this project has focused on the front end, but enhancements have been added to the back end as well, particularly the Python back end.

The parser front-end can be further subdivided into a "lex" tokenizer (which also does rudimentary preprocessing) and

a "yacc" parser that understands the C++ grammar. These two pieces are the foundation of the wrappers, or less generously, they are the bottleneck. The wrappers are only able to wrap the class and method definitions the parser can pull from the header files. Because of the parser's importance, it has received more attention during this update than any other part of the wrappers.

An important feature of the VTK wrappers is that they wrap the VTK classes one class at a time using only the header file for that class, with a minimal amount of hinting. When combined with CMake, this approach is easily scalable to a very large number of classes in different directories or even in different packages. The new wrappers further enhance this approach by automatically generating "hierarchy" files that describe all types defined in any particular VTK source directory. These files are discussed in detail later in this article.

A SHINY NEW PARSER

The new parser is a significant improvement to the old parser code. The original code took a minimalist approach to parsing the VTK header files. It looked for the first VTK class defined in the header, and then extracted only the methods defined in that class. The rest of the file would be ignored, but since the parser lacked a full C++ grammar, it was not always successful in skipping over parts it was supposed to ignore. These troublesome patches of code had to be surrounded by BTX/ETX exclusion markers so that they could be removed at the preprocessing stage.

The new parser code reverses this minimalist approach: it reads all the declarations in the header file and stores them all for use in the wrappers. This means that typedefs, templates, constant definitions, enum types, operator overloads and namespaces are all available to the wrapper-generators. Each piece of information from the header file is stored in a C struct, with the most-used struct being the ValueInfo struct that is used for method arguments, variables, constants and typedefs. The following is from the vtkParse.h header file:

```
struct ValueInfo
{
  parse_item_t   ItemType; /* var, typedef, etc */
  parse_access_t Access; /* public, private, etc. */
  const char     *Name;
  const char     *Comment;
  const char     *Value; /* value or default val */
  unsigned int   Type; /* see vtkParseType.h   */
  const char     *Class; /* type as a string */
  int            Count;  /* for arrays */
  int            NumberOfDimensions;
  const char     **Dimensions;
  FunctionInfo   *Function; /* for function ptrs */
  int            IsStatic; /* class variables only */
  int            IsEnum; /* for constants only */
};
```

One should note that in contrast to the old parser, arrays can now be multi-dimensional. The dimensions are stored as strings, in case the dimensions are symbolic values (e.g., template parameters) that cannot be evaluated until compile time. The product of the dimensions is stored as an "int" if all the dimensions are integer literals.

Similar structs provide information for functions, classes, namespaces, templates, etcetera. The FunctionInfo and ClassInfo structs have backward compatibility sections that provide their info in the old wrapper format, so that wrappers-generators based on the old structs can easily be made to work with the new parser.

The new parser also features a preprocessor, something that was conspicuously absent before. The preprocessor stores defined macros and provides conditional parsing based on #if directives, eliminating yet another previous use of the BTX/ETX markers. Unlike a traditional preprocessor, the parser stores macros but does not expand them. This is by design, since several VTK macros have special meaning to the wrappers that would be lost if those macros were expanded. The parser can query the macros to get their value.

HIERARCHIES FOR ALL

A fundamental addition for the new wrappers is a collection of "hierarchy" files, one per source directory, that list the full genealogy of all the VTK classes in each directory. The file vtkCommonHierarchy.txt, for example, lists all classes defined in the Common directory. The file structure is simple:

```
vtkArray : vtkObject ; vtkArray.h ; ABSTRACT
vtkArraySort ; vtkArraySort.h ; WRAP_EXCLUDE
```

The name of the class comes first, followed by any superclasses, then the header file, and finally any of the CMake flags WRAP_EXCLUDE, WRAP_SPECIAL or ABSTRACT which apply to the class. Classes that have a name that is different from the name of their header file are automatically labelled with VTK_WRAP_EXCLUDE. Note that the file format might change in the future, so anyone who is interested in using this file should always use the functions defined in VTK/Wrapping/vtkParseHierarchy.h to read the file, instead of writing their own code to do so.

In addition to classes, the hierarchy files also include all typedefs and enum types encountered in the header files, in order to provide a comprehensive list of all types defined in VTK. The rationale behind the hierarchy files is as follows: previously, the wrappers would assume that any type with a "vtk" prefix was derived from vtkObjectBase, excepting types like vtkIdType that were specifically caught by the parser. This is the other reason that BTX/ETX had to be used so often (in addition to the aforementioned limitations of the parser), since methods that used new types like vtkVariant or vtkUnicodeString had to be BTX'd because these types were misidentified by the wrappers. By using the hierarchy files, the wrappers can identify any type defined in VTK.

WRAPPER COMMAND LINE ARGUMENTS

The command line for the wrapper-generators has been modified, and this will make it easier to invoke the wrappers by hand. The old calling convention still works, in order to support older CMake scripts. The new command line is as follows:

```
vtkWrapPython [options] input_file output_file

  --concrete     tell wrappers that class is concrete
  --abstract     tell wrappers that class is abstract
  --vtkobject    class is derived from vtkObjectBase
  --special      class not derived from vtkObjectBase
  --hints <file> specify a hints file
  --types <file> specify a hierarchy file
  -I <dir>       add an include directory
  -D <macro>     define a preprocessor macro
```

All of these arguments are optional. For instance, the wrapper-generators will automatically guess that any class with pure virtual methods is an abstract class. However, the –concrete option is needed in cases where an abstract class is being wrapped that will be replaced by a concrete factory-generated class at run time.

The "--hints" option provides a way of specifying the hints file that has been used by the wrappers since day one. The hints file is the only part of the wrappers that still uses hexadecimal literals to describe types. Support for it will be maintained indefinitely to ensure backwards compatibility. The new "--types" option gives the wrappers access to the new hierarchy files that were described above.

The "-I" and "-D" options are forwarded to the preprocessor, so that if the file being wrapped includes important header files like vtkConfigure.h, the preprocessor can read those files and use the macro values defined within them.

STRINGS AND UNICODE

The vtkStdString type was introduced several years ago, and when it was added to the wrappers, it was wrapped by identifying it as "const char *" in the parser, with its c_str() method used to do the conversion. This caused problems, because most VTK methods return vtkStdString by value, resulting in the creation of a temporary string object on the stack, for which the char pointer returned by c_str() is likewise only temporarily valid. Because of this, only methods that returned vtkStdString by reference could safely be wrapped, and methods that passed vtkStdString by value were blocked with BTX/ETX. One of the reasons that vtkStdString was wrapped this way is that the old parser had only 15 slots available to identify fundamental VTK types, compared to nearly 255 slots for the new parser.

The new Tcl, Python and Java wrappers have all been modified so that they correctly identify vtkStdString as a std::string subclass and wrap it appropriately, allowing the BTX/ETX markers to be removed from methods that pass vtkStdString by value. The vtkUnicodeString type has also been added to the parser as a new type, and the Python wrappers have been modified to transparently convert between vtkUnicodeString and Python's own unicode type.

WRAPPING VARIANTS IN PYTHON

There are two approaches that could have been used to wrap vtkVariant. The first approach would have been to have the Python wrappers implicitly convert Python types to and from vtkVariant, so that if a C++ VTK method returned a variant, the wrapped Python method would automatically extract and return the value stored in the variant. This would have been convenient, but also would have resulted in loss of information. For example, any integer type between "unsigned char" and "long" would automatically be converted into a Python integer, and Python users would not be able to discover the original C++ type. For this reason, an approach in which vtkVariant is wrapped as its own distinct Python type was used instead.

The technique used to wrap vtkVariant is similar to the technique used to wrap vtkObjectBase-derived objects. The main differences are the way memory management is done and the way that constructors are handled. Unlike vtkObjects, the vtkVariant is not reference-counted, so if a variant is passed as a method argument, it is copied just as if it was an int or any other basic type. Wrapping the many constructors for vtkVariant was a challenge, because the old Python wrapper code for resolving overloaded methods was inadequate for this task: it would simply try each overload in turn until one of the overloads could be called without generating an argument type error. Hence, calling vtkVariant(10) from Python would create an "unsigned char" variant since vtkVariant(unsigned char) is the first constructor in the header file. For this wrapper update project, code was added to the Python wrappers so that they compare all passed arguments against those of the various method signatures, and call the overload that provides the best match. This new code is used for all Python method calls, not just for constructors, so there might be some small backwards compatibility problems since the old code always called the first matched method, while the new code calls the best match.

Another feature that was added to the Python wrappers is automatic argument conversion via constructors. In C++, if a method requires a vtkVariant argument, but is passed as an "int", the vtkVariant(int) constructor will automatically be called to convert the "int". No such automatic conversion exists in Python; it is instead the responsibility of the method writer to have it explicitly convert the arguments. Fortunately, since all the VTK/Python methods are generated by the wrapper-generator code it only required some creative programming to have the Python wrappers automatically look through the constructors of all wrapped types and do conversions as necessary.

BEYOND VARIANTS

The new wrappers handle the Python-wrapping of vtkVariant automatically, and a few other special VTK types are similarly wrapped. These types are wrapped by marking them with VTK_WRAP_SPECIAL in the CMakeLists.txt file, and they must also have a public copy constructor and an "=" operator. Other special types that have been wrapped include the vtkArray helper types: vtkArrayCoordinates, vtkArrayExtents, vtkArrayRange, and vtkArrayWeights.

Even though the parser now recognizes all operator methods, at this point in time the Python wrappers only wrap the comparison operators "<" "<=" "==" "!=" ">=" ">", and the stream output operator "<<" via Python's "print" statement. Some types will need additional operator methods to be wrapped in order to make them truly useful from Python.

CONCLUSION

The overhauled parser and the hierarchy files provide a solid new foundation for VTK wrapper development, but work still needs to be done to update the back-end wrapper generators for Tcl, Python and Java. The Python wrappers now support vtkUnicodeString and several other special VTK types, and the code has been reorganized and documented, but entities like multi-dimensional arrays and templated classes that are parsed by the new parser are not yet wrapped in any language. All of these can hopefully be added to the wrappers in the coming years.

ACKNOWLEDGEMENTS

I would like to acknowledge the work performed by Marcus Hanwell and Keith Fieldhouse in testing the new code and merging it with VTK, and would also like to thank Bill Hoffman, Will Schroeder and Ken Martin for allowing me to be part of their open-source experiment for these past twelve years.

David Gobbi *is a software consultant in Calgary, Alberta who specializes in software for medical imaging and image-guided neurosurgery. He received his Ph.D. from the University of Western Ontario in 2003, and has been contributing to the VTK wrapper code since 1999.*

THE CDASH "@HOME" CLOUD

Imagine running a little program on your computer to volunteer your computing power to help keep your favorite open source project in tip-top shape. Extend that to a swarming cloud of volunteers running builds at your request and submitting results to your project's dashboard. Soon, with the help of CTest and CDash, you'll be able to do just that.

We've created a feature in CDash called "build management". This feature allows users to request that a project build be done on any client that has volunteered its resources. You can try it out today if you have a checkout of CDash from svn trunk. This feature is not fully implemented in the current CDash 1.6 release, so you'll have to use the bleeding edge code to try it out.

The feature works through a polling protocol. A client machine runs a CTest script that performs the following steps. First, the client machine submits an xml file describing its resources to CDash (submitinfo). The client then queries CDash to get its corresponding "site id" (getsiteid). The client then starts executing a loop, in which it queries CDash for a job to do (getjob). If the client is given a job to perform by CDash it does the job and reports the results, and then continues the loop. Otherwise, the client sleeps for a while and then continues the loop.

With several client machines connected and looping, querying CDash for jobs to do, CDash can match build requests to waiting clients.

After clients are set up and looping, a project administrator may go to his "My CDash" page and schedule a build by clicking on the "Schedule Build" icon. If no clients are currently connected, then CDash will not allow you to schedule builds. If clients are connected, you should see a list of available clients, what OS each is running, and what compilers each has available. As the build scheduler, you choose a client to run the build on and click the "Schedule" button at the bottom of the page.

After you schedule a build, the next time a client asks for a job to do, CDash assigns it a job, sending a CTest script back as the result. The client then executes that script, which yields a new submission to the dashboard from that client.

As part of scheduling a build, you may: specify an alternate repository from which to clone or checkout, specify initial CMakeCache.txt entries, add a build name suffix to identify an experiment in the dashboard results and choose which configuration to build and test. Other settings are also available in the CDash schedule build interface on the manageClient.php page.

When submitting an xml file to CDash, use the normal ctest_submit function in the CTest script. To query CDash for information, use "file(DOWNLOAD" in the CTest script.

At present, running CDash as a build management system involves setting things up on the server (server admin and project admin) and on each client machine (volunteers) that will be participating, as well as scheduling builds manually (project admin). However, the vision for this feature moving forward, is to make running CDash as a build management system as automatic as possible. Anybody should be able to request a test run of the project on any client that is presently available. Right now, only project administrators have the privilege of scheduling builds. It should eventually even be possible to have CDash observe whenever a commit is made to a project's repository and immediately start doling out build instructions to waiting clients.

See the CDash Wiki page for more (and evolving) information on build management. This feature is still in its infancy and we could use your help to shape its future. Try it out and send us feedback on the CDash mailing list with the subject "CDash Schedule Builds Feedback".

SCHEDULING BUILDS FOR YOUR CDASH PROJECT

First, you have to coordinate with your CDash server administrator to set $CDASH_MANAGE_CLIENTS to 1 in either the cdash/config.php or cdash/config.local.php file.

Next, as project administrator, go to the Edit Project page and check on the settings on the Repository tab. The "Repository" and "Repository Viewer Type" settings need to be set. If you don't really have a repository viewer, choose a type from that list that matches your repository anyway. CDash uses that setting to generate the correct checkout and update commands in the scripts that it sends to volunteer clients.

You can see a part of the script that CDash will send to the clients on the "Clients" tab of the Edit Project page. You may even customize it by modifying the script or adding to it as necessary for your project.

Next, make sure the CTestConfig file checked into your repository is correct. You may download an initial copy from CDash if you don't already have one. Click on the "Miscellaneous" tab and look for the CTestConfig download link.

Next, set up some volunteers running "looping scripts". On each volunteer client machine, you should create a "Client" directory, and then two sub-directories in Client named "base" and "tmp"

In the Client directory, we need a site description xml file and a looping script. On my client machines, I name the xml file "$SITE.cdash.xml" and I name the script "CDashClient.ctest". The xml file should contain contents like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<cdash>
 <system>
  <platform>Mac</platform>
  <version>SnowLeopard</version>
  <bits>64</bits>
  <basedirectory>/Users/davidcole/Client/base</base-
directory>
 </system>
 <compiler>
```

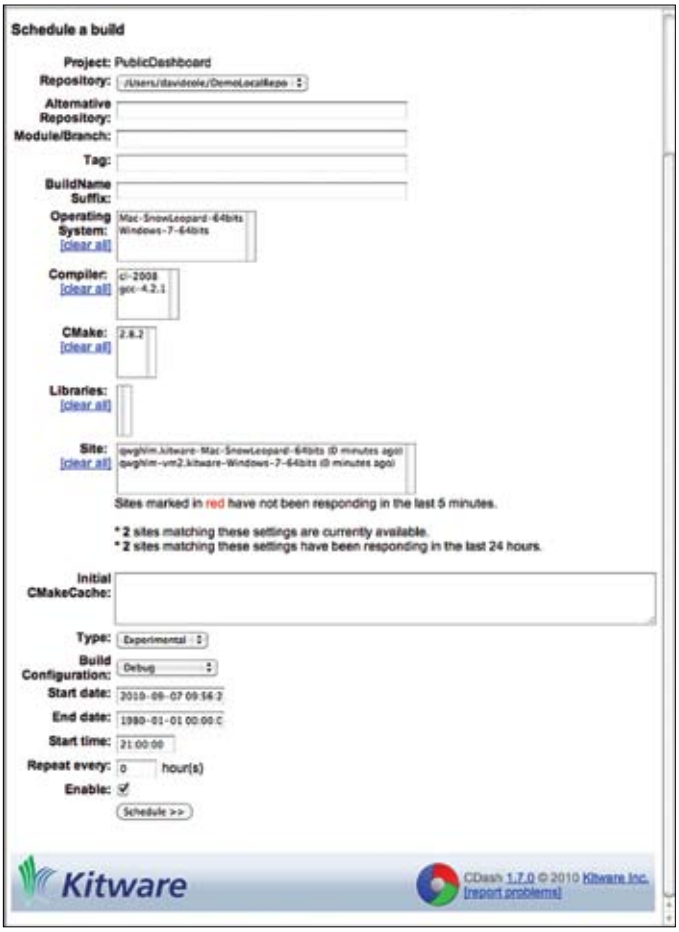
```
<name>gcc</name>
  <version>4.2.1</version>
  <generator>Unix Makefiles</generator>
 </compiler>
 <cmake>
  <version>2.8.2</version>
  <path>/Applications/CMake 2.8-2.app/Contents/bin/
cmake</path>
 </cmake>
</cdash>
```

As you can see, this xml file gives CDash sufficient information to know what OS your client is running, what compiler and CMake version it has available and what CMake generator to use for configuring the build.

The full looping script is about 65 lines of CTest scripting code, I'll post this script to the Kitware Blog so we don't have to waste space here; the only part that's different from machine to machine looks like this:

```
set(CDASH_SITENAME "qwghlm.kitware")
set(CDASH_SYSTEMNAME "Mac-SnowLeopard-64bits")
set(CDASH_SITE_CONFIG_FILE "/Users/davidcole/Client/
qwghlm.cdash.xml")
set(CDASH_TEMP_DIRECTORY "/Users/davidcole/Client/
tmp")
set(CTEST_EXECUTABLE "/Applications/CMake 2.8-2.app/
Contents/bin/ctest")
set(CTEST_DROP_SITE "www.yourcompany.com")
```

After the looping scripts are running on a client or two (or more!), you may go to the "Schedule Builds" interface of CDash and request builds from them. Go to the "My CDash" page, click on the "Schedule Builds" icon – it looks like a floppy disk, you should see a page like this:



By default, you should be able to choose a client from the "Site:" list and click on the "Schedule >>" button at the bottom. However, you may wish to enter information into any of the fields on that page to control various aspects of the build.

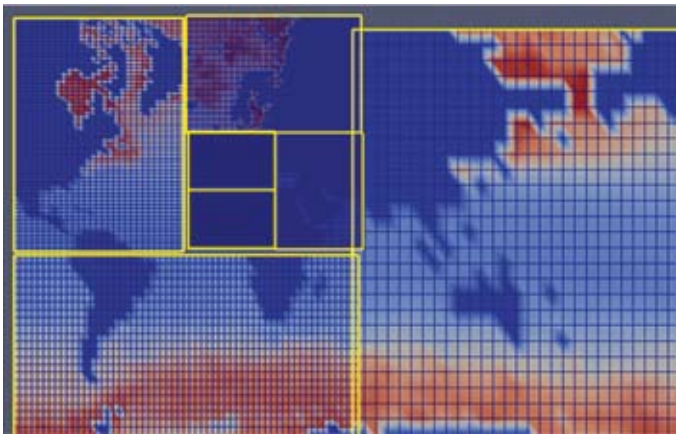
A list of projects with "Schedule Build" icons

Let us know what needs improvement. Especially let us know if you'd like to sponsor future work on this topic.

***David Cole** is an R&D Engineer in Kitware's Clifton Park office. David has contributed code to the VTK, CMake, ITK, ParaView, KWWidgets and gccxml open-source projects and to Kitware's proprietary products including ActiViz and VolView.*

MULTI-RESOLUTION STREAMING IN VTK AND PARAVIEW

ParaView's ability to process and visualize massive datasets is based on VTK's streaming functionality. In ParaView, identical copies of the visualization pipeline are run on many machines and each pipeline is asked to process a different small portion, or piece, of the input data. Together the machines process the entire dataset simultaneously, and no machine ever exceeds the capacity of its local RAM.



Refinement at work in the analysis of a 3600x2400x42 chlorofluorocarbon (CFC) concentration simulation study, being performed on a 32-bit laptop PC. Yellow outlines identify "pieces". At this point, ParaView has progressed seven levels down into a nine level deep piece refinement tree. Blue outlines show individual cells. At the lowest level cells are already at sub-pixel resolution.

VTK's ability to break up data is called streaming, because when the data can be divided it is also possible to iterate over the pieces. In data parallel processing you stretch the problem over a larger amount of RAM whereas in streaming you stretch the problem over a longer time. Iterative streamed rendering proceeds, for example, by rendering each piece in turn without clearing the color or depth buffer, using the persistent Z-buffer to resolve occlusion for individual triangles both within a piece and across pieces. In practice, streaming and data parallel processing are orthogonal and can be combined by dividing the problem into P sets of I pieces.

In many computational processes and especially rendering, it is often the case that only a small fraction of the entire data contributes to the end result. Prioritized streaming processes only those pieces that contribute to the final result (ignoring pieces that are off-screen, for example) and processes them in a most important (i.e. nearest to the camera) to least important order. This variation of streaming has great benefits including eliminating unnecessary processing and IO [1] and providing more immediate feedback to the user to speed the data discovery process [2]. Prioritized streaming is the basis for the experimental branded application called StreamingParaView. StreamingParaView was first introduced in ParaView 3.6.

VTK's streaming has an interesting feature in that it's possible to ask for data in arbitrarily small chunks. Streaming is driven by asking for different pieces at the downstream end of the pipeline (ex vtkPolyDataMapper::SetPiece()). One asks for smaller pieces by asking for pieces out of a larger set (ex v tkPolyDataMapper::SetNumberOfPieces()). What is interesting about this is that as the chunk size decreases - assuming prioritization is in effect - the work done to produce a given visualization approaches the minimal amount necessary.

We recently added the ability to ask for data at differing resolution levels to VTK's streaming support. It is now possible to not only ask for arbitrary pieces, but also to ask for them at arbitrary resolutions. The mechanics are similar to VTK's temporal support [3]. One asks the pipeline to provide data at a given requested resolution. This request is a dimensionless number that ranges from 0.0 meaning lowest resolution to 1.0 meaning full resolution. If unspecified, full resolution is assumed. The request travels up the pipeline to the reader, which decides how to interpret this number. Structured data sources use the resolution to choose how coarsely to subsample in the i, j and k dimensions and adjust their x, y and z spacing to compensate accordingly. As in the temporal pipeline, the result is free to vary from what was requested. To support this, the reader inserts a resolution answer key into the data it produces, which then travels back down the pipeline to the requester.

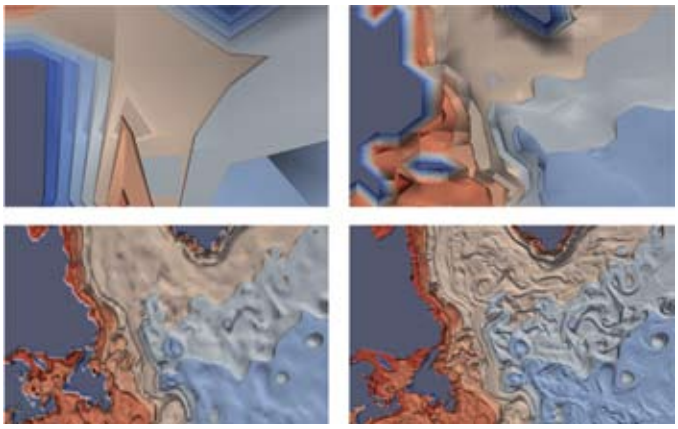
The following is a code example from VTK which asks the pipeline to: compute the priority for a particular piece at a particular resolution, conditionally update the pipeline to produce the requested piece and then examine the returned resolution result.

```
vtkStreamingDemandDrivenPipeline * filtersExec =
  vtkStreamingDemandDrivenPipeline::SafeDownCast
    (aFilter->GetExecutive());

filtersExec->SetUpdateResolution
  (port, requestedResolution);
filtersExec->SetUpdateExtent
  (port, pieceNum, numPieces, ghostLevel);
//the next call is very fast
double priority = filtersExec->ComputePriority();

if (priority > 0.0)
  {
  //the next call is potentially very slow
  aFilter->Update();
  vtkDataObject *data =
    aFilter->GetOutputDataObject(port);
  vtkInformation* dataInfo = data->GetInformation();
  double resultResolution = dataInfo->Get
    (vtkDataObject::DATA_RESOLUTION());
  }
```

ParaView 3.8 also includes AdaptiveParaView, another experimental application which exercises this new multi-resolution capability. Multi-resolution streaming begins by rendering a single piece that covers the entire domain at minimum resolution with the very first update giving the user valuable feedback about the entire domain. This is a great advantage over standard streaming in which global information sometimes becomes apparent only when the entire domain is covered at full resolution and after a much longer delay.



Refinement of isosurfaces in the CFC data off the coast of Newfoundland. A laptop computer is able to show the full resolution data since areas off-screen are ignored and only a few pieces stay resident in memory at any given instant.

The adaptive streaming algorithm then recursively splits pieces, increasing the resolution with each split, to show the data in greater detail. As it processes pieces it gathers meta-information (world extent and data ranges) and uses this information to improve its importance estimates. Throughout the algorithm, prioritization guides the choice of which pieces need to be refined immediately, which can be deferred, and which can removed from further consideration. The algorithm eventually converges to processing just the important features of the data at the fullest resolution.

THE NEED FOR CACHING

Unfortunately streaming adds overhead. Every pipeline update can take a significant amount of time. In standard VTK, the pipeline internally caches intermediate results so that data processing time can be amortized over many later updates if those do not invalidate the results. Unfortunately, changing the requested piece or resolution invalidates the internal caches. We minimize the problem by aggressively caching results at the end of the pipeline. In particular, whenever the visible geometry is small enough, our cache allows all of the data to be re-rendered in a single pass.

In this situation, camera movement proceeds as fast as it does in non-streaming ParaView. Despite caching, the convergence process itself can take a significant amount of time, therefore AdaptiveParaView has controls that allow the user to pause and restart, limit, or manually control the refinement process.

CONSERVATIVE PRIORITIZATION

A key point is that it's generally impossible to know a priori what pieces contribute the most or least without executing the pipeline. This is unfortunate because the goal of prioritization is to avoid executing the pipeline on unimportant data. Consider culling pieces that fall outside of the view frustum. Many VTK readers can determine the initial world-space bounding box of any requested piece. However filters exist to change data, and any filter along the pipeline might transform the data arbitrarily changing the bounding box before the data is rendered. In order to find which pieces are not visible, one must first know what the world space bounding box of each piece is after that piece is processed by the pipeline.

To solve this chicken and egg problem, a facility for per piece meta-information propagation was added to the VTK pipeline. Readers and sources can provide piece level meta-information, and filters can describe what types of data modifications they do. With information about what is provided and what is potentially changed, the pipeline is better able to provide a conservative estimate of the importance of any piece without doing any time consuming computations. When either type of information is missing, the pipeline falls back to the non-prioritized behavior of iteratively processing every piece. See VTK/Rendering/Testing/Cxx/TestPriorityStreaming.cxx for a demonstration.

FUTURE WORK

We continue to "refine" our implementation of adaptive streaming. Our most immediate challenge is to improve the client parallel server implementations of both streaming applications. Currently, pure streaming requires too frequent communication with the server to be efficient and adaptive streaming has only been implemented to work with serial mode ParaView runs.

Next, there are fairly major robustness and usability limitations of our preliminary experimental prototypes. Our current work is available in source format only and there are outstanding unsolved issues regarding how to deliver global up-to-date meta-information to the end user as computation progress.

Lastly, we are working to extend the work to be compatible with more data formats. Our first streaming capable reader reads simple raw structured data in either preprocessed or raw format. For this data type, changing resolution is easily achieved by sub-sampling [4]. We have since extended the framework to handle cloud data, of which the LANL cosmology format was our first target. For this we devised an importance sampling mechanism that chooses representative samples and limits the resolution so as not to overfill the displayed image resolution.

We anticipate extending the framework to work on AMR data, which has multi-resolution information written in by the simulation codes that generate it; and to wavelet compressed image data, as exemplified by NCAR's Vapor Data Format files. Extending the framework to handle non-preprocessed unstructured data types is a long-term goal.

REFERENCES

[1] Childs, H., Brugger,E., Bonnell, K., Meredith, J., Miller, M., Whitlock, B., Max, N. "A Contract Based System For Large Data Visualization." Proceedings of the IEEE Visualization Conference 2005.

[2] Ahrens, J., Desai, N., McCormick, P., Martin, K., Woodring, J. "A modular extensible visualization system architecture for culled prioritized data streaming." Proceedings of the SPIE, 2007.

[3] Biddiscombe, J., Geveci, B., Martin, K., Moreland, K., and Thompson, D. "Time Dependent Processing in a Parallel Pipeline Architecture." IEEE Transactions on Visualization and Computer Graphics, 2007.

[4] Ahrens J., Woodring J., DeMarle D., Patchett J., Maltrud M. "Interactive Remote Large-Scale Data Visualization via Prioritized Multi-resolution Streaming." Proceedings of the UltraScale Visualization Workshop, 2009

David DeMarle *is a member of the R&D team at Kitware where he contributes to both ParaView and VTK. He frequently teaches Kitware's professional development and training courses for these product applications and enjoys putting puns in Kitware Source articles. Dave's research interests are in systems level aspects of visualization, in particular memory optimizations for parallel visualization of large datasets.*

Jonathan Woodring *is a new staff scientist in CCS-7 Applied Computer Science at Los Alamos National Laboratory. He received his PhD in Computer Science from The Ohio State University in 2009. His research interests include scientific visualization and analysis, high performance supercomputing, and data intensive supercomputing.*

James Ahrens *is a team leader in the Applied Computer Science Group at Los Alamos National Laboratory. His research focuses on large-data visualization and scientific data management. Ahrens received his Ph.D. in Computer Science from the University of Washington and is member of the IEEE Computer Society.*

COMMUNITY SPOTLIGHT

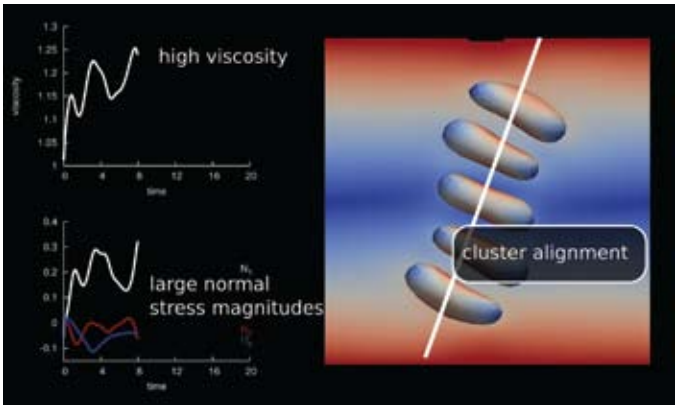
The following articles highlight how some of our community members are utilizing Kitware's open-source toolkits and products in their own development environment. If you would you like to be featured as a Community Spotlight, send your article ideas and materials to editor@kitware.com.

VISUALIZING DEFORMABLE SUSPENSION SIMULATIONS IN PARAVIEW

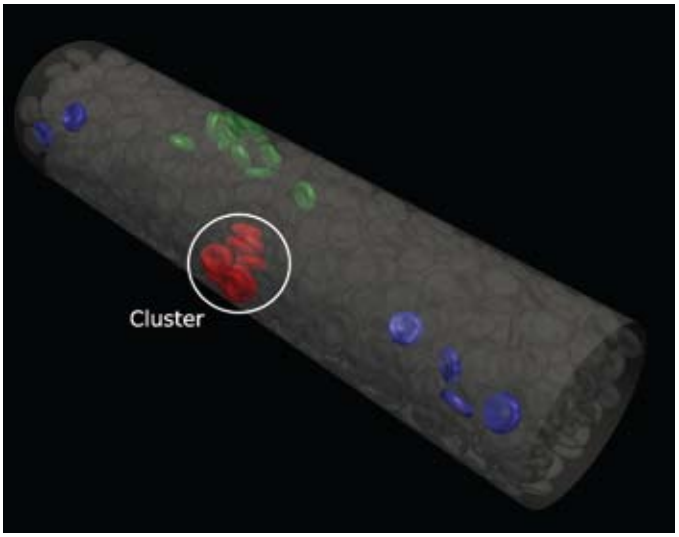
Computational simulation of blood flow and dense suspensions of deformable particles is only possible on high performance computing resources using a highly scalable computational approach such as the hybrid lattice-Boltzmann / finite element (LB-FE) method. This method, originally developed for rigid suspensions, was extended to deformable red blood cells (RBCs), particles, capsules [1] and

fibers [2, 3]. The LB-FE method resolves the fluid phase, both interior and exterior to the fluid-filled particles, with the LB method. A recent review article [4] highlights the feasibility of using the LB method for complex flows including suspension of deformable, particles, capsules and fibers.

We examine the dynamics of deformable capsules and how these dynamics affect the rheology of dense suspensions using the LB-FE method. Of particular interest are the particle pressure and a full characterization of normal stresses, which are difficult to measure experimentally. The following figure demonstrates a novel problem where the particle microstructure affects the viscosity, normal stresses, and particle pressure of a suspension. The maximum viscosity and first normal stress difference is obtained when particles are aligned in a cluster along the compressional axis.



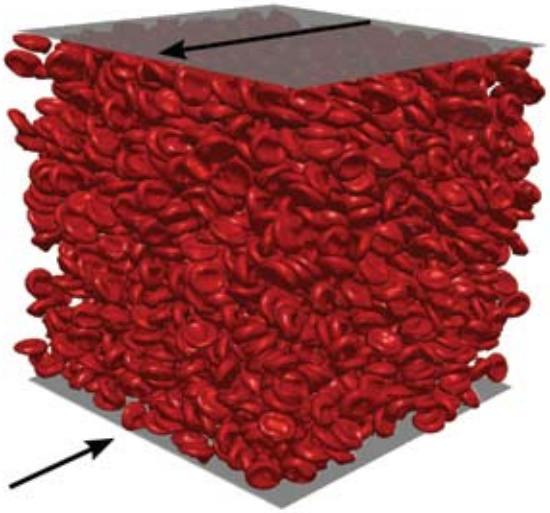
Red blood cells (RBCs), the most numerous constituent of blood, influence continuum-level measurements by altering the suspension at microscopic scales. In moderately-large-sized arteries, the nature of blood can be described as a macroscopic non-Newtonian fluid; however, flow in smaller vessels requires treating human blood as a multiphase fluid containing 40-45% RBCs by volume. Simulations of blood at a cellular level provide a tool that allows exploration of the rheology, stress, and diffusion of individual suspended cells. In our approach, the blood plasma and hemoglobin are treated as Newtonian fluids with viscosities of 1.2 and 6 cP, respectively. The multiphase nature of the blood creates non-Newtonian flow effects, such as shear thinning, commonly observed in experiments. An image of multiple clusters forming in Hagen-Poiseuille flow is shown below.



VISUALIZATION OF DATA WITH PARAVIEW

We have shown the ability to simulate thousands of deformable particles and capture non-Newtonian characteristics that agree well with experimental observations [1]. Our approach has shown the ability to scale on as many as 65,536 computational cores of Argonne National Laboratory's IBM Blue Gene/P [5]. When computing the dynamics of 1,000's to 100,000's of deformable particles suspended in fluid, datasets can easily reach 1 terabyte or greater for a single simulation. As an example, one such simulation with 100,000 deformable spheres (each with 254 triangular surfaces) requires a LB fluid domain in excess of 2 billion lattice grid points.

When fluid and solid data sets from CFD codes are O(100 GB), the use of distributed memory architectures is required for post-processing. Through the use of virtual network computing (VNC) and access to visualization clusters like Longhorn and Spur at the Texas Advanced Computing Center (TACC) via NSF's TeraGrid HPC network, visualizations not possible on shared memory resources can be performed. Visualization clusters like those at TACC offer multiple CPU, multiple GPU nodes that are created specifically for these applications. The following is an instantaneous snapshot of a dense suspension of 2472 RBCs in a wall-bounded shear.



FUTURE WORK

Recent additions [6] have shown the ability to simulate dense suspensions of RBCs in micro-vessel sized tubes at high shear rates, in which large RBC deformations extend beyond the assumptions of linear theory. This work is based on coupling the LB method with a spectrin-link model [7] for the RBC membrane.

REFERENCES

[1] R. M. MacMeccan, J. R. Clausen, G. P. Neitzel, and C. K. Aidun, "Simulating deformable particle suspensions using a coupled lattice-Boltzmann and finite-element method," J. Fluid Mech., 618:13-39, 2009.

[2] J. Wu and C. K. Aidun, "Simulating 3D deformable particle suspensions using lattice-Boltzmann method with discrete external boundary force," Inter. J. Num. Meth. Fluids, 62:765-783, 2009.

[3] J. Wu and C. K. Aidun, "A method for direct simulation of flexible fiber suspensions using lattice Boltzmann equation with external boundary force," Inter. J. Multiphase Flow, 36:202-209, 2010.

[4] C. K. Aidun and J. R. Clausen, "Lattice-Boltzmann Method for Complex Flows," Annual Rev. Fluid Mech., 42:439-472, 2010.

[5] J. R. Clausen, D. A. Reasor, and C. K. Aidun, "Parallel Performance of a Lattice-Boltzmann/Finite Element Cellular Blood Flow Solver on the IBM Blue Gene/P Architecture," Comp. Phys. Comm., 181:1013-1020, 2010.

[6] D. A. Reasor, J. R. Clausen, and C. K. Aidun, "Coupling the lattice-Boltzmann and spectrin-link methods for the direct numerical simulation of cellular blood flow," submitted to Inter.l J. Num. Meth. Fluids, June 2010.

[7] J. Li, M. Dao, C. T. Lim, and S. Suresh, "Spectrin-Level Modeling of the Cytoskeleton and Optical Tweezers Stretching of the Erythrocyte," Biophys. J., 88:3707-3719, 2005.

Dr. Cyrus Aidun *joined the Woodruff School of Mechanical Engineering at the Georgia Institute of Technology as a Professor in 2003. He began at Tech in 1988 as an Assistant Professor at the Institute of Paper Science and Technology. Prior, he was at Battelle Research Laboratories and was Senior Research Consultant at the National Science Foundation's Supercomputer Center at Cornell University.*

Daniel Reasor *received his Bachelor's and Master's degree in Mechanical Engineering from the University of Kentucky (2006, 2007). He is currently a Ph.D. student at the Georgia Institute of Technology performing research focused on the direct numerical simulation of cellular blood flow. He is funded by the U.S. Department of Defense through the ASEE SMART fellowship.*

Dr. Jonathan Clausen *received a Bachelor's degree in Mechanical Engineering from Clemson University in 2004 and a Ph.D. from Georgia Institute of Technology in 2010. His Ph.D. research focused on the rheology and microstructure of deformable capsule suspensions. Since July 2010, Jonathan Clausen has been a Senior Member of Technical Staff at Sandia National Laboratories.*

Dr. Jingshu Wu *has a Master's degree in Aerospace Engineering (2004) and a Ph.D. in Mechanical Engineering (2010) from the Georgia Institute of Technology. His Ph.D. research focused on the development of methods for multiphase flow simulation. Since June 2010, he has been an aeroacoustic engineer at Vestas Technology R&D Americas.*

ITURRIBIZIA ON HOW XC UTILIZES VTK

XC is a program designed at our company, Iturribizia [1], to solve structural analysis problems utilizing a finite element method. The program can solve various types of problems, from simple linear analysis to complex nonlinear simulations. It has a library of finite elements which can be used to modeling various geometries and multiple materials for use in various areas of structural analysis.

MOTIVATION

Someone said that, when the French climber Lionel Terray was asked about his reason to climb a mountain, he simply said "because it was there". Something similar happened with the development of this program. I began the study of the finite element method after studying the analytical solutions to elastic problems (so limited) and I became greatly interested in their use in structural problems. This, coupled with my love for computer science, made me decide to develop a finite element program that would be useful to calculate structures and could be modified and expanded in any way the user wanted.

First I wrote a Pascal version of the program which only worked with bar-type elements. Then I wrote a C++ version "from scratch" that was never able to solve any nontrivial problem. Finally, I discovered the possibilities offered by the calculation core of Opensees and decided to modify it to be suitable for an "industrial environment" (as opposed to academic use).

To achieve this objective, several significant modifications to the original code were required. Algorithms for generating finite element mesh were incorporated, allowing the modeler to create structured grids from the description of geometry by means of points, lines, surfaces and solids. Graphics were generated using the VTK library (we give more details on this later.) A new macro language was developed to make it possible to obtain the results produced by the calculation without having to extract them from predefined listings. This provides the program with the ability to interpret a sentence like "get the ratio between the vertical displacement of the node closest to the center of the beam and the total span of the beam". Utilities for the construction and calculation of design load combinations prescribed by the building codes (EHE, ACI 318, EAE, Eurocodes, etc.) were implemented to facilitate the verification of design requirements. The ability to activate and deactivate elements was introduced to enable the analysis of structures built in phases, geotechnical problems, and the strengthening of existing structures. Macros were written to verify the structure and its elements according to the criteria prescribed by building codes (e.g. axial and bending capacity, shear reinforcement). The code was changed to link with "standard" linear algebra libraries (e.g. BLAS, Arpack, LAPACK, SuperLU), eliminating the need to include in the program "ad-hoc" versions of these libraries. Finally, the material models were modified to support prescribed strains, making it possible to solve problems involving thermal and rheological actions.

DESIGN GUIDELINES

With the experience obtained from previous development works we arrived at the following conclusions: test, test carefully and test again; do not reinvent the wheel do not waste time in developing an elegant GUI; and while building codes change, physical laws do not.

Testing

For each functional element of the program a test should be written to verify the correctness of the code. Also after each code modification it must be verified that every one of the tests still execute successfully. Once an error is detected and corrected we must write a test to verify that the error will not happen again.

Reinventing the Wheel

Rather than starting from scratch every time you need to introduce something to your code, seek out open source libraries. We've learned to use open source libraries (such as VTK) as much as possible. Make sure that these libraries are easily accessible via the Internet.

GUI Development

Don't waste time developing an elegant GUI. Graphical user interfaces make the users believe that they know how to use the program. These types of interfaces make possible to take a trial and error approach which, in our view, may be appropriate to handle a word processor in which the result is visible, but not so much for a computer program whose management requires a thorough review of the input data and design assumptions. On the other hand the use of an interpreted language provides much higher flexibility. Consider a line segment definition that may involve two points, a point and a vector, a point, a length and an angle or the intersection of two planes. This flexibility is very difficult to achieve with a graphical user interface.

The Laws of Physics

While building codes are subject to change, physical laws remain constant. In structural engineering we still use Newton mechanics to solve mechanical problems. The algorithms that solve these types of equations (equilibrium, kinematic, etc.) should be written in C++ as these algorithms, by nature, are unlikely to change.

On the other hand, building codes are periodically renewed due to advances in understanding the behavior of materials and the greater demands of society regarding the level of quality required for its infrastructures. Consequently, and since there is no need to hide the code to the user, we use procedures written as scripts (that are easy to modify) to implement the verification of design requirements from building codes.

FEATURES

This section covers the program's features that make it particularly suitable for use in structural design applications.

Tools for reporting

The program must have utilities that make it easy to generate reports, both numerical and graphical, which are ready to be included in structural design reports. The basis of this report generation system is the use of the LaTeX document preparation system.

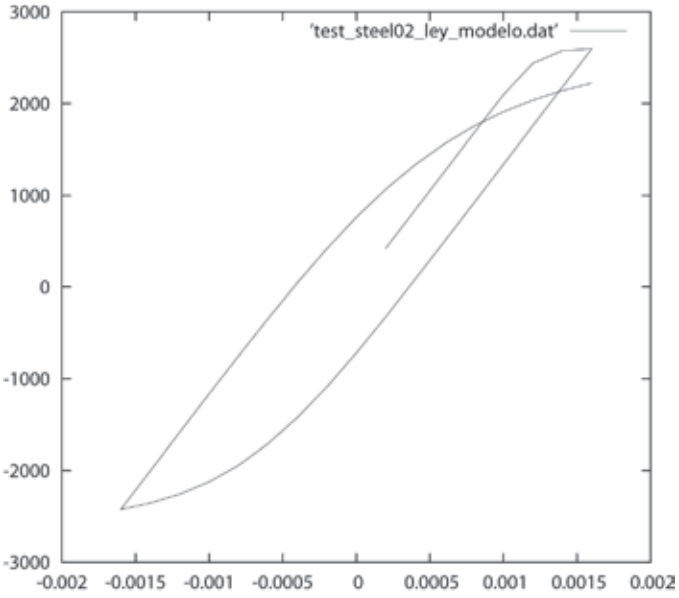


Figure 1: Steel material model

Three procedures are used for graphic generation. VTK is used to generate bi- or three-dimensional graphic information for the values presented by a scalar or vector field in the finite element mesh. Gnuplot [2] is used to generate graphs of functions or numerical data, such as that in Figure 1. The plotutils library is used to generate Postscript graphics.

Tools for calculating multiple design load combinations

The program should facilitate both the linear and non-linear calculation of multiple load combinations. Modern building codes require knowledge of a structure's response to a large number of combinations (from a few hundred in simple building structures to more than a thousand in structures with moving, thermal and seismic loads). Checks are needed for each of these combinations depending on whether the structure is made of steel, concrete, or wood and special parts (pins, anchor bolts).

When the calculation is linear, the response to the load combination may be obtained as the sum of responses to each load. On the contrary, when the analysis must be non-linear it's necessary to find a solution for each of the combinations using any intermediate results already obtained.

Tools for performing design checks

The design of a structure in accordance with the requirements of building codes is often based on the application of various criteria. In terms of stability it is necessary to check the balance of the structure and safety from buckling phenomena. A material's resistance needs to be tested to determine the maximum strains and stresses that the material can withstand. A structure's stiffness needs to be tested to determine its maximum allowable displacement (limits on beam deflection, collapse). These quantities, in general, can be approximated from the displacement of closest node in the model. A structure's dynamics is tested in terms of its conditions of comfort and to ensure that its natural frequency values vary enough from its excitation frequency. Lastly, the fatigue strength of the structure is determined based on the structure's capacity to withstand cyclic loading.

Mesh generation tools

The task of creating a finite element mesh that properly models a civil engineering structure (a building, dam, bridge) is one of the most time-consuming because the geometry of its elements rarely supports analytical representation. Moreover, the presence of holes and voids in the structures themselves and other structural reinforcements make the model even more complex. In some structures it is necessary to model pre-stressed tendons embedded in concrete or other similar items.

GRAPHICAL OUTPUT

To give the program the ability to plot the results from analysis we take into consideration the following possibilities:

- Programming the graphical output interface directly using OpenGL, present in almost any recent computer.
- Using OpenDX, an open-source library based on IBM Open Visualization Data Explorer.
- Using Kitware's VTK library.

Besides the problem of deciding which interface to use, there was the need to determine how the program would generate graphics. There were two options: develop a set of pre-defined graphics with adjustable appearance parameters that could be used to display displacements, stresses,

strains, loads, etc., or design a command language which would be a means through which users could define the graphical output.

The first approach is commonly used in some matrix structural analysis computer programs and other programs oriented to specific tasks such as slope stability analysis or solving plane elasticity problems. The main disadvantage of this solution is that it's very difficult to apply to situations that were not previously considered by the programmer.

The second solution is used in the majority of the general purpose finite element codes (ANSYS, Abacus). Its fundamental difficulty is the design of a command language that is flexible enough and easy to use. During analysis of this solution we studied other program manuals (ANSYS, Abacus, Solvia, Calculix Graphix) to determine the features needed for the command language.

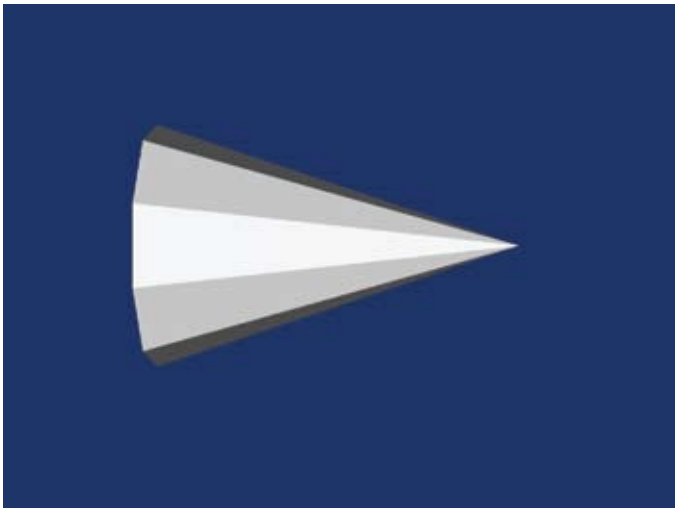


Figure 2: View of a cone generated with the script from Table 1

After studying both solutions we decided that the first was almost unworkable if we wanted the program to be able to treat multiple types of problems (e.g. analysis of buildings, bridges, reservoirs).

The second was more appropriate to deal with the problem since it provides the users with the tools they need to display results and, in some way, transfer the task to them. Once the solution was chosen, we had to design a command language powerful enough to generate graphics allowing the use of all options (transparency, lighting, textures) that modern computer graphics offer. Here the potential of VTK came into play.

The VTK library is written in C++ (the same language used for the remainder of the program), making it quite easy to call it from the rest of the code. The availability of some script languages (Tcl, Java and Python) in VTK made us realize that the command language design was already done (the VTK API itself) and all we had to do was move it to our own script language used by the rest of the program. Proceeding in this manner, we enabled the use of VTK for any purpose the user desires as opposed to just making the code available for finite element model representations. Figure 2 (which will be recognizable to all those familiarized with the examples supplied with the VTK package) shows the results of the macro shown in Table 1. A more complex example can be seen in Figure 3.

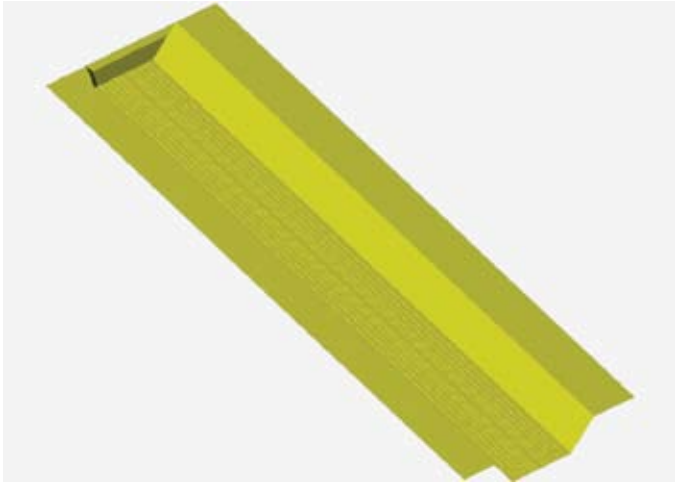


Figure 3: Shell finite element model of a bridge deck

```
\vtk
 {
   \define["vtkConeSource","cone"]
     { \altura{3.0} \radio{1.0} \resol{10} }
   \define["vtkPolyDataMapper","coneMapper"]
     { \set_input{"cone"} }
   \define["vtkActor","coneActor"]
     { \set_mapper{"coneMapper"} }
   \define["vtkRenderer","ren1"]
     {
       \add_actor{"coneActor"}
       \set_background{0.1,0.2,0.4}
     }
   \define["vtkRenderWindow","renWin"]
     { \add_renderer{"ren1"} \set_size{1024,768} }
   \define["vtkRenderWindowInteractor","iren"]
     { \set_render_window{"renWin"} }
   \define["vtkInteractorStyleTrackballCamera",
           "style"]
     {}
   \iren{\set_interactor_style{"style"}}
   \iren{\initialize{} \start{}}
}
```

Table 1: Script to display the cone on figure 2

ACKNOWLEDGEMENTS

Many thanks to all the people who contribute to open source, without their effort this work would not have been possible. Thanks to Professor Filip C. Filippou and Frank Mackenna from the Department of Civil and Environmental Engineering at the University of California, Berkeley for their email correspondence. Thank you to my family for their patience as I worked on the development of this software package. And thank you to my friends Raul Hernandez and Tomas Sanz for their encouragement.

REFERENCES

[1] Iturribizia may be downloaded here: http://www.iturribizia.com/descarga software.html.
[2] http://www.gnuplot.info

Luis C. Pérez Tato *was born in Madrid (Spain) in 1965. Studied civil engineering at the "Universidad Politécnica de Madrid". His work has been related to computer programming since 1988 and to structural analysis since 1992. Has developed programs in the fields of structural analysis, geographic information systems an dam instrumentation.*

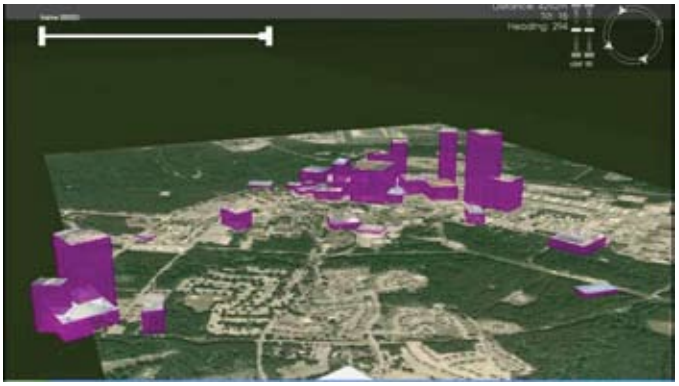
KITWARE NEWS

DARPA AWARDS KITWARE $11 MILLION FOR PHASE II OF VIRAT PROGRAM

Kitware has received an $11 million contract from the Defense Advanced Research Projects Agency to lead Phase II of its Video and Image Retrieval and Analysis Tool (VIRAT) program. Kitware was selected to lead the sole Phase II award based on the success of their Phase I effort.

In Phase II of VIRAT, Kitware will lead the development of the second-generation VIRAT system, a revolutionary video analysis capability that filters and prioritizes massive amounts of archived and streaming video based on events; presents the high-value intelligence content clearly and intuitively to video analysts; and results in substantial reductions in analyst workload while increasing the quality and accuracy of intelligence yield.

The VIRAT system is focused on aerial surveillance video taken from Predators and other drones. VIRAT automatically extracts descriptors of events and human actions in the video, emitting real-time alerts of events of interest. The system also indexes the descriptors into a database to enable subsequent search for similar and related events. Specific examples of distinguishable events range from single-person (walking, limping, gesturing) and person-to-person (meeting, following, gathering) behaviors to person-vehicle interactions, among others.



A visualization of a video query result where each purple tower is a video clip retrieved from a video archive. Height indicates the level of similarity to a query video clip supplied by a video analyst. The analyst can browse the result clips in 3D+time by watching all clips play simultaneously on the towers.

In Phase I, the Kitware team developed a number of novel algorithms and systems to address the VIRAT problem, including: algorithms capable of detecting and tracking small, slow-moving objects over time; a large bank of state-of-the-art, complementary action and object descriptors that can capture track-level and articulated events; algorithms for using analyst feedback to iteratively refine queries to quickly determine analyst intent; and innovative techniques for indexing and searching the descriptor database.

In Phase II, the Kitware team will deploy a VIRAT prototype at an end-user facility, and train analysts in its operation. Their evaluation and feedback will provide researchers with critical data to incorporate into upgrades and enhancements.

The team will also continue to develop and apply cutting-edge research into the hard problems in descriptors, indexing, iterative query refinement, and descriptor fusion. One of the many research thrusts in Phase II will be to incorporate more computed and supplied scene knowledge directly into the descriptors, and to more effectively leverage available metadata such as view point and sun angle. Augmenting system capabilities to allow for additional video sources, both from UAV's and ground cameras, will also be a top priority.

To meet the Phase II program's needs, Kitware has combined the majority of the Phase I participants into a single unified team including six leading defense technology companies: Honeywell Laboratories – ACS; Raytheon BBN Technologies; Mayachitra, Inc.; BAE Systems – Technology Solutions; General Dynamics; and Lockheed Martin Missiles and Fire Control Autonomous Systems. Lockheed Martin will serve as the system integrator for this Phase II effort. Multiple internationally-renowned research institutions round out the world-class team, including: the Computer Vision Laboratory, University of Maryland; Rensselaer Polytechnic Institute; the Computer Vision Lab at the University of Central Florida; the University of California, Riverside; the University of Southern California; Massachusetts Institute of Technology; the University of Texas at Austin; California Institute of Technology; Cornell University; Stanford University; the University of California, Berkeley; the University of California, Irvine; and Columbia University.

Phase II is expected to take 18 months to complete, and will be led by Anthony Hoogs and Amitha Perera at Kitware.

KITWARE AWARDED NIH GRANT TO IMPROVE LESION BIOPSY USING PET-CT IMAGING

Kitware received a one-year grant from the National Institutes of Health (NIH) totaling $228,458 to improve the clinical effectiveness of liver lesion biopsy using PET-CT imaging. The project will extend the open source Image Guided Surgery Toolkit (IGSTK) to enable the fusion of motion corrected PET images with CT images for liver lesion biopsy.

The main focus of the project will be the development of a robust respiratory motion correction technique aimed at producing more effective PET-CT guided biopsies. Kitware and its team of researchers will develop a respiratory motion correction algorithm to help develop a better technique, which decreases errors in imaging caused by artifacts like organ sliding which can occur just from the natural respiration process.

''While PET imaging can localize malignancies in tumors that do not have a CT correlate, the diagnostic benefit is often gravely affected by basic respiratory motion,'' said Dr. Andinet Enquobahrie, technical lead at Kitware and one of the principal investigators for the project. "The difference in acquisition times between PET and CT data often leads to discrepancy in spatial correspondence which then causes problems with tumor localization. With this grant, Kitware will develop robust respiratory motion correction technique to correct this and reduce incorrect tumor staging."

Kitware will team up with Dr. Kevin Cleary at Georgetown University for development of some of the components and clinical evaluation of the system. Under the leadership of Dr. Cleary, the Computer Aided Interventional and Medical Robotics (CAIMR) group at Georgetown has developed a number of applications for improving the accuracy of interventional procedures. Kitware and Dr. Cleary have a long and

productive relationship collaborating on several NIH-funded STTR, SBIR and RO1 projects over the past five years. In addition, the research team has contracted two consultants with extensive expertise in PET imaging and clinical use of PET-CT imaging for interventional procedures.

Upon completion, the system will display the motion-corrected, fused and side-by-side, PET and CT images for interventional radiologists to view. Electromagnetic tracking of the needle tip using the image-guided system will provide continuous monitoring of the needle relative to the PET and CT images. The radiologist would then use this virtual image display of metabolic and anatomic information to guide the needle to the lesion. This system will enable future applications beyond biopsy, such as ablations and the delivery of other minimally invasive therapies.

KITWARE RANKS #1655 ON THE 2010 INC. 5000 WITH THREE-YEAR SALES GROWTH OF 172%

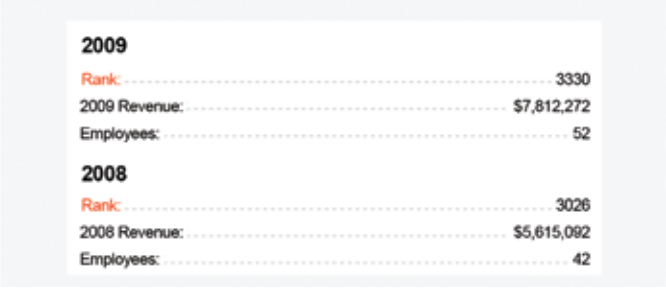
Kitware has made the Inc. 5000 list for the third year in a row, with a ranking of 1655. The Inc. 5000 is a listing of the fastest-growing, privately-held companies in America. The list represents the most comprehensive look at the most important segment of the economy—America's independent-minded entrepreneurs. Music website Pandora, convenience store chain 7-Eleven, Brooklyn Brewery, and Radio Flyer, maker of the iconic children's red wagon, are among the prominent brands featured on this year's list.

"The leaders of the companies on this year's Inc. 5000 have figured out how to grow their businesses during the longest recession since the Great Depression," said Inc. president Bob LaPointe. "The 2010 Inc. 5000 showcases a particularly hardy group of entrepreneurs."

Over the last 12 months, Kitware has grown its workforce by 28% and is continuing to hire to fill positions in computer vision, medical imaging, informatics, scientific visualization and open access publishing. For more information on Kitware's hiring needs, please visit kitware.com/jobs.

Complete results of the Inc. 5000, including company profiles and an interactive database that can be sorted by industry, region, and other criteria, can be found on www.inc.com/5000.

Previous Inc.500|5000 Rankings



Methodology

The Inc. 5000 is ranked according to percentage revenue growth from 2006 through 2009. To qualify, companies must have been founded and generating revenue by the first week of 2006, and therefore able to show four full calendar years of sales. Additionally, they have to be U.S.-based, privately held, for profit, and independent as of December 31, 2009. Revenue in 2006 must have been at least $100,000, and revenue in 2009 must have been at least $2 million.

KITWARE PARTICIPATES IN MILITARY OPEN SOURCE SOFTWARE WORKING GROUP

In August, Kitware participated in the second iteration of the Military Open Source Software Working Group (MIL-OSS). The company has had representatives at each of the iterations of this working group which aims to grow open-source adoption and contribution within the Department of Defense and its burgeoning contractor community. This year's conference featured speakers from each branch of the armed services as well as speakers from companies dedicated to open source such as Redhat and Puppet Labs.

Lockheed Martin had representatives to present their work on Eureka Streams, a new open-source project sponsored and developed within that company, which is perhaps an illustration of the greater movement toward open source software within the government contracting community. Seeing the large, traditional defense contractors embrace open-source and contribute back to the community is a testament to the growth in awareness of open technology in recent years.

The Defense Information Systems Agency (DISA) also spoke about their work with Collabnet (the custodians of Subversion) to develop Forge.mil, a Source Forge for "community source" projects within the military. This allows projects that are deemed too sensitive in nature for public consumption to be shared within the government and other trusted entities. Forge.mil and traditional open-source seem to have a bright future in the years ahead.

Kitware's Patrick Reynolds and Chuck Atkins participated in many of the lively panel discussions and after-hours philosophical arguments about how the DoD can best benefit from open source. Mr. Reynolds was given the opportunity to give an Ignite presentation on "Open-Source Continuous Integration using CMake and CDash" during one of the speaking sessions. Ignite talks are five minute talks with 20 slides that auto-advance every 15 seconds. The six Ignite speakers presented at breakneck speed over topics as diverse as large-scale system administration and combat-relevant position-location information.

As Kitware's business involves collaboration with more and more traditional defense contractors, interactions with vibrant communities like MIL-OSS will help broaden the reach of open source within government.

KITWARE AND THE NA-MIC COMMUNITY ANNOUNCE ALPHA RELEASE OF 3D SLICER 4.0

3D Slicer is an open-source (BSD-style license) platform for medical image segmentation, registration, visualization and analysis. It represents a practical integration of innovations from a multitude of fields including medical image analysis, cross-platform software development, collaborative software development and extensible platform design. Kitware has been closely involved in many of these innovations. In particular, 3D Slicer builds upon ITK, VTK, CMake, CDash, CTK and MIDAS.

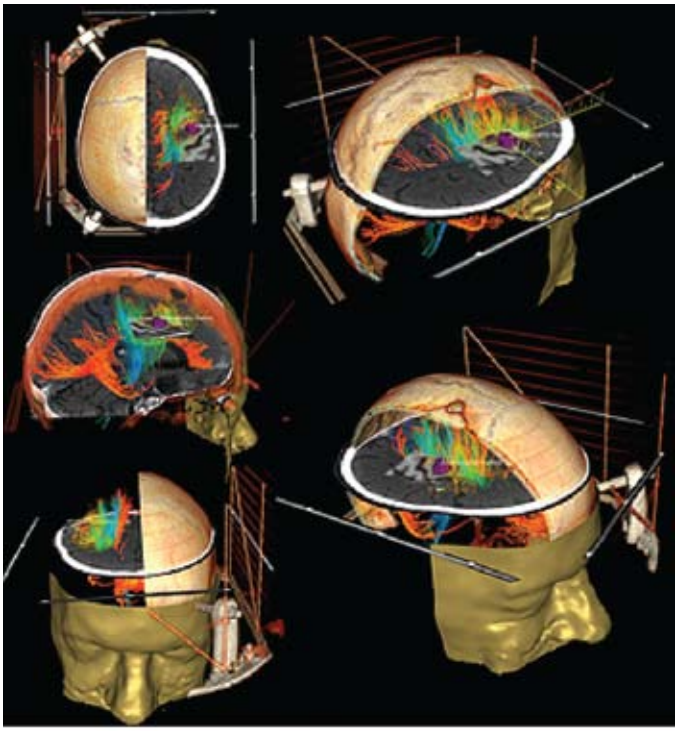
Slicer is funded by grants associated with the National Alliance for Medical Image Computing (NA-MIC) and the Neuroimaging Analysis Center (NAC) and has an extensive community behind it. The lead institute for these multi-institutional centers is Brigham and Women's Hospital (BWH). Other funded participants include the University of Utah, GE Corporate Research, the University of North Carolina at Chapel Hill, Georgia Tech, Johns Hopkins, MIND Institute, MIT, MGH, UCLA, and many others.

To date, 3D Slicer contains 95+ modules that cover a large range of applications including tractography, endoscopy, tumor volume estimation, image guided surgery and more. Special attention has been given to documenting each module and creating tutorials that cover many of the most common use-cases for Slicer. Over 400 publications acknowledge funding from one or more Slicer-related grants. Those publications span a wide range of medical, biomedical, pharmaceutical, and other basic and applied science domains.

Version 4.0 of 3D Slicer is a systematic rewrite of the Slicer platform. The main Slicer application now adheres to high programming standards (e.g., unit tests and a clean model-view-controller design) and has adopted Qt for its GUI. Compared to Slicer 3.X, Slicer 4.0 has cut the number of lines of code necessary for most features by 70 percent. Slicer 4.0 is faster, more stable, more extensible, and more developer and user friendly than Slicer 3.X. Tcl is gone and has been replaced by Python and CMake's Superbuild. The modules' user interface can be effortlessly generated using Qt Designer to allow the programmer to focus on algorithms.

Slicer 4.0 alpha (developer) was release in early September. The Beta release is scheduled for December 2010 and the full Slicer 4.0 release is scheduled for February 2011. However, much of the Slicer 4.0 platform is already complete. We are encouraging our friends to begin using Slicer 4.0 now to ensure it achieves the lofty, yet practical goals we have set.

To learn more about the NA-MIC community, visit namic.org, to download Slicer, visit slicer.org and to browse the collection of Slicer-related publications (hosted on MIDAS), visit slicer.org/publications.



Contrast enhanced CT (with the stereotactic frame of a gamma-knife device), anatomical MRI and diffusion tensor MRI have been registered and segmented to produce this image. Fiber tracking allows the clinician to examine brain connectivity. Neurosurgeons use such fused views to determine tumor margins and surgical paths. This image was generated using 3D Slicer. In particular, 3D Slicers' affine registration module, DTI fiber tracking module, and connected component segmentation module were used. All of these modules are built using VTK and ITK and are available as open-source as part of the 3D Slicer distribution. This image was generated by Andras Jakab, University of Debrecen, Medical School and Health Science Center, Hungary. This image won 2nd place in the 2008 Best Biomedical Visualization Contest hosted by Kitware (The Source volX issueY).

ITKV4

Ten years ago, the National Library of Medicine initiated a software development program to produce an open-source toolkit for the segmentation and registration of medical images, in particular the Visible Man and Visible Woman data. The outcome of that program was the Insight Toolkit (ITK). From those humble beginnings, ITK is now used in basic and applied research, commercial medical image processing and surgical guidance systems around the world.

It is estimated that ITK is contributing to projects in over 45 countries and in nearly every major academic and industry research lab involved in medical image analysis. Applications areas include radiology, neurology, pathology, oncology, neurosurgery and even satellite imagery. Data being processed by ITK includes nearly every medical imaging modality such as electron microscopy, MRI, CT, PET, ultrasound, video and OCT.

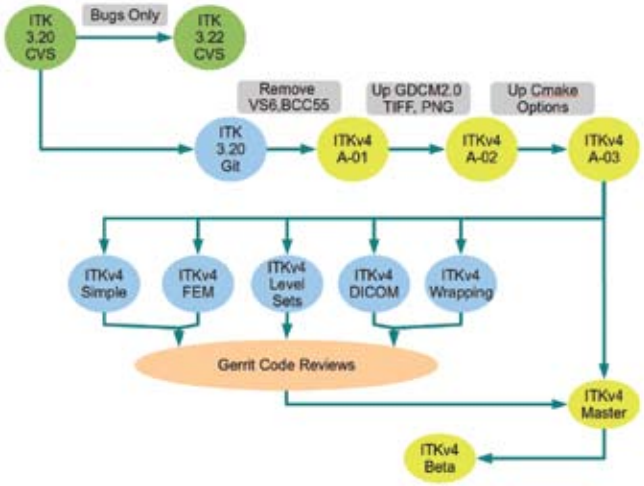
This year, ITK will be going through a major refactoring process to ensure its vitality for the next ten years. The main goals of the refactoring process are:

- Revise the ITK architecture to support modern algorithms and to adopt a module-based architecture that supports development and distribution of optional extensions and specialization of ITK for specific problem domains (e.g., a "4D confocal microscopy" module for ITK).
- Simplify the use of ITK by offering the power of ITK in intuitive packages the seamlessly integrate with Python, Java, and other programming languages.
- Accelerate the algorithms of ITK by supporting distributed and GPU-based processing.
- Improve DICOM support in ITK so that its use of clinical data and integration into clinical workflows is assured.

The new version of ITK will be designated ITKv4. ITK 3.20 will be the stable and reliable release offered to users during the time ITKv4 is being developed.

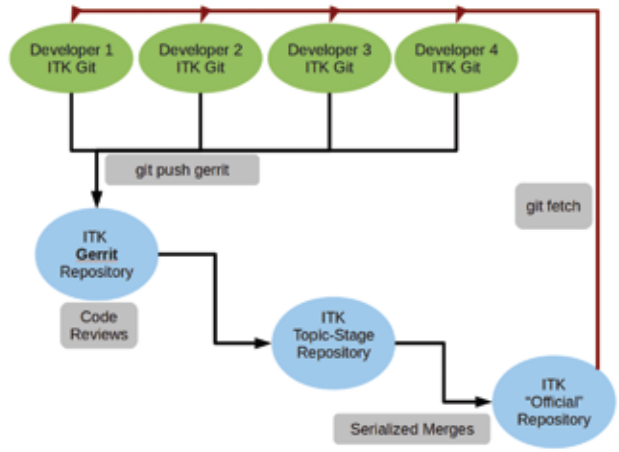
In this refactoring, ITK will be improving its support for domain fields beyond radiology. In particular, we will be working closely with application developers in the fields of microscopy, remote sensing and computer vision. Providing better support for these domains will require the need to introduce changes and improvements in ITK for fundamental features such as support for very large images (larger than 4GB), support for multi-channel images and the addition of support for new file formats (i.e., LSM, TIFF variations, JPEG2000, among others).

Information about the modifications to ITK will be disseminated following the "Release Early, Release Often" rule of Open Source software development. We've started with an initial clean up of the toolkit, following the migration plan described on the ITK Wiki and schematically described below. As you can see, one of the first steps was moving from CVS to Git. You can now clone ITK by following the instructions provided on the ITK Wiki.



The work in ITKv4 will be performed collaboratively between multiple groups including, but not limited to: GE Research / the Mayo Clinic, the University of Iowa, the University of Pennsylvania, Harvard University, Cosmo and Kitware.

The design and development activities will be coordinated on the ITK developer's mailing list and through an ITK conference call. Both of these venues are open to the public and we encourage all users to join these events and contribute their points of view to the process. In order to coordinate the work of the approximately 30 developers contributing to ITK's redesign, we have also put in place a code review process based on Gerrit, a tool that is also used by the Android community. The current workflow of software patches is summarized in the image below.



As the development process settles in, this workflow will likely be readjusted in order to better accommodate the needs of the ITK development community. We look forward to working with the larger community of ITK users to make sure that ITKv4 is a useful and powerful resource for many different applications.

KITWARE AWARDED ADMINISTRATIVE SUPPLEMENT FOR NEUROSURGERY SIMULATION

Kitware was awarded an administrative supplement by the National Institutes of Health (NIH) to fund the purchase of a multi-GPU computer produced by RenderStreams, Inc.

This supplement dovetails on a two-year grant from the NIH, totaling more than $600,000, to focus research efforts on developing approach-specific, multi-GPU, multi-tool, high-realism neurosurgery simulation The computer will enable the computational acceleration of several processes simultaneously in the context of a neurosurgery simulator. This includes an emphasis on multi-grid finite elements for tools that manipulate tissue and multi-grid meshless methods with one GPU assigned per resolution level.

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.

LATE FALL/WINTER CONFERENCES AND EVENTS

ACCV 2010

November 8 – 12, in Queenstown, New Zealand. ACCV 2010 is the tenth Asian Conference on Computer Vision. Amitha Perera will be in attendance. http://www.accv2010.org/

SC10

November 13 – 19, in New Orleans, Louisiana. SC10 is a platform for HPC researchers and developers to demonstrate, debate and discover innovative, cutting-edge advances in computation, networking, storage, and analysis. Andy Bauer, Berk Geveci, Bill Hoffman and Will Schroeder will be in attendance. http://sc10.supercomputing.org/

RSNA 2010

November 28 – December 3, in Chicago, IL. Rick Avila will be presenting at the RSNA 2010 Quantitative Imaging Reading Room during the RSNA annual meeting. Rick and Stephen Aylward will be giving a talk on open source in medical image analysis. Kitware will be exhibiting in Hall D Lakeside Center Level 3. Wes Turner and Brad Davis will also be in attendance. http://rsna2010.rsna.org/

IEEE Winter Vision Meeting 2011

January 5 – 7, in Kona, Hawaii. Anthony Hoogs will be attending the Workshop on Applications of Computer Vision which will include tracks on Biometrics and Motion and Video Computing. http://vision.cs.byu.edu/wvm2011/wvm.php

NEW EMPLOYEES

Lynn Bardsley

Lynn joined Kitware as a Program Manager for the Computer Vision Group in July. She received her M.S. in Computer Science from Union College.

Dhanannjay Deo

Dr. Deo joined Kitware as a member of the Scientific Visualization Group in August. He received his B.S. in Mechanical Engineering from the University of Pune, his M.S. from the Indian Institute of Science and his Ph.D. from Rensselaer Polytechnic Institute.

Xiaoxiao Liu

Xiaoxiao joined Kitware as a member of the Medical Imaging Group in September. She received a B.E. in Computer Science and an M.E. in Pattern Recognition and Intelligent Systems from Huazhong University of Science and Technology and her Ph.D. in Computer Science from the University of North Carolina at Chapel Hill.

Danielle Pace

Danielle joined Kitware as a member of the Medical Imaging Group in July. She received her B.CmpH in Biomedical

Computing from Queen's University and M.ESc. in Biomedical Engineering from The University of Western Ontario.

Ben Payne

Ben joined Kitware as a member of the Medical Imaging Group in September. He received his B.S. in Applied Science with a focus on Computer Engineering from the University of North Carolina at Chapel Hill and his M.S. in Computer Science from the University of Illinois at Urbana-Champaign.

Zhaohui Harry Sun

Dr. Sun joined Kitware as a member of the Computer Vision Group in June. He received his B.E. and M.E. degrees in Electrical Engineering and Information Science from the University of Science and Technology of China, and his M.S. and Ph.D. in Electrical and Computer Engineering from the University of Rochester.

Nicole Wardle

Nicole joined Kitware's as a member of the Computer Vision Group in September. She received her B.S. and M.S. in Computer Science from Rensselaer Polytechnic Institute.

Matthew Woehkle

Matthew joined Kitware as a member of the Computer Vision Group in August. He received his B.A. in Computer Science and Mathematics from Concordia University.

Ted Yapo

Ted joined Kitware as a member of the Computer Vision Group in July. Ted recieved his B.S. and M.S. in Engineering Physics from Rensselaer Polytechnic Institute.

KITWARE TO OFFER ONLINE PARAVIEW COURSE

Starting in November, Kitware will be offering instructor-led online courses covering our open-source products. This will allow our global community to obtain training without the hassle or expense of long distance travel. Each short session will be targeted at a specific topic and topics will be offered frequently, allowing our users to easily fit this productivity-boosting training into their busy schedules. For more information about the new online courses, visit our website and select Training Courses from the Product menu. For information about scheduling a customized course at your location, please contact us at courses@kitware.com.

KITWARE GOES INTERNATIONAL

To better serve our international customers and collaborators Kitware is opening two new offices. The first, in Lyon, France, will be lead by Julien Jomier. Julien has been instrumental in developing MIDAS, Kitware's Scientific Data Management system, and has extensive experience in medical imaging and agile software process. The second office, in Bangalore, India, will be led by Karthik Krishnan. Karthik is an expert developer with deep knowledge of both VTK and ITK, as well as medical application development.

These two offices reflect Kitware's growing global presence. Our software is used worldwide in commercial, research, educational and government applications. These offices will enable Kitware to respond rapidly to customer needs, provide cost-effective support and consulting services, hire outstanding talent across the globe and collaborate more closely with our international partners. Please contact Kitware if you would like further information.

EMPLOYMENT OPPORTUNITIES

Kitware is seeking candidates for the following software development positions: Informatics, Biomedical Imaging, Computer Vision, and HPC Visualization and Data Analysis.

Our comprehensive benefits package includes flex working hours, six weeks paid time off, a computer hardware budget, 401(k), medical insurance, dental insurance, vision insurance, flexible spending account (FSA), life insurance, short- and long-term disability, visa processing, a generous compensation plan, bonus, and free coffee, drinks and snacks.

Interested applicants should forward a cover letter and resume to jobs@kitware.com to ensure their immediate consideration.

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, including:

- 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)

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.

Contributors: James Ahrens, Cyrus Aidun, Lisa Avila, Michel Audette, Stephen Aylward, Sophie Chen, Jonathan Clausen, David Cole, David DeMarle, Julien Finet, Jean-Christophe Fillion-Robin, Marcus Hanwell, Bill Hoffman, Luis Ibáñez, Brad King, Nicole Messier, Dave Partyka, Daniel Reasor, Patrick Reynolds, Luis Pérez Tato, Wes Turner, Jonathan Woodring, Jingshu Wu.

Graphic Design: Steve Jordan
Editor: Niki Russell

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.