SOFTWARE DEVELOPER’S QUARTERLY

TABLE OF CONTENTS

Editor’s Note ........................................................................... 1
Recent Releases ..................................................................... 1
Composite Datasets in VTK .................................................. 4
Information Visualization in VTK with Titan ...................... 5
itkQuadEdgeMesh Introduction ........................................... 7
Depth Peeling In VTK ........................................................... 11
In Progress .......................................................................... 12
Community News ................................................................. 13
Kitware News ...................................................................... 13

EDITOR’S NOTE

The Kitware Software Developer’s Quarterly Newsletter 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, Dr. Alex Gouaillard and Dr. Eric Boix provide an overview and brief tutorial on the itkQuadEdgeMesh algorithms they’ve developed to support two-manifold mesh processing in ITK. Dr. François Bertel reveals how VTK’s new depth peeling filter supports visualization by allowing users to render translucent polygonal geometry without sorting polygons. Utkarsh Ayachit provides brief tutorial on the implementation of composite datasets in VTK, including the new functionalities for multipiece datasets and the addition of new extraction filters. And Jeff Baumes’ discussion of the new Kitware project, Titan, exposes VTK as a very powerful information visualization resource.

The Kitware Source is just one of a suite of products and services that Kitware offers to assist developers in getting the most out of its open-source products. Each project’s website contains links to free resources including mailing lists, documentation, FAQs and Wikis. In addition, Kitware supports its open-source projects with technical books, user’s guides, consulting services, support contracts and training courses. For more information on Kitware’s suite of products and services, please visit our website at www.kitware.com.

RECENT RELEASES

CMAKE 2.4.8
In January 2008, Kitware released CMake 2.4.8. This is expected to be the last CMake release in the 2.4.X branch. 2.4.8 has many bug fixes over 2.4.7. Some highlights are:

• Visual Studio 9 support
• CMakeSetup.exe works with Windows Vista with no elevation of permissions.
• Improved FindQt4
• Fixes to FindPkgConfig
• Allow NODEFAULTLIBS to have more than one value
• Suppress regeneration of makefiles during try compile avoid infinite. Loop in trycompile with some VS 2005 builds.
• Allow for debug libraries in InstallRequiredSystemLibraries.
  cmake for vs8
• Fix for cpack and messing up PATH with NSIS
• Fix for exception handling flags in VS 2003 and up
• Fix for kde4-config location
• Fix for self extracting .sh files on solaris
• Fix for version numbers on NetBSD
• Add more search directories (install prefix and cmake location)
• Include WindowsPaths in Windows.cmake not just Windows-cl.cmake
• Fix incorrect file names in the vtk*Kit.cmake files
• Make #cmakedefine output match autoconf when undefined
• Add support for DragonFly and GNU hurd

For a full list of features and bug fixes, see the announcement in the CMake mailing list archives.

VTK PATCH RELEASE

The VTK 5.0.4 patch release was announced on January 22, 2008. Changes from 5.0.3 to 5.0.4 are listed below. We are planning to create the VTK-5-2 branch in CVS soon in preparation for the (upcoming) release of VTK 5.2.

• XML precision fix
• Suppress deprecation warnings when using cl with Unix Makefiles generator
• Java weak reference issue
• Use vtkIdType instead of int for correct 64-bit id builds
• Fixed and simplified vtkOpenGLExtensionManager and updated OpenGL header files
• Remove legacy include of GL/glaux.h
• Add static method VTKTypeID and merge long long bug fix in vtkTypeTraits.h
• Fix incorrect file names in the vtk*Kit.cmake files
Replace MPProcessors() for better Mac OS X 64-bit compatibility

Selected Utilities/kwsys/SystemTools.cxx fixes for KWWidgets file browser dialog

Fix problems when multiple observers invoke further events recursively on the same object

Fixed bug in vtkTransform::GetOrientation when matrix is not positive semi-definite

Set the upper limit of vtkMergeCells::PointMergeTolerance to VTK_LARGE_FLOAT

Eliminate memory leak in vtkUnstructuredGrid::BuildLinks

CDASH BETA RELEASED

Kitware is happy to announce the release of CDash Beta, a new tool in our Quality Software Process chain. CDash is a follow up effort to Dart1 and Dart2. It uses many of the concepts that have been developed over the past decade for testing Dashboards. For a full history see http://cdash.org/about.html.

CDash is implemented as a web application using standard web tools like php and MySQL.

CDash Features

- Client/Server model for testing and reporting
- Separation of data from presentation using XML and XSLT
- Interactive dashboards with content updated in real-time
- Summarization of build logs to highlight any errors and warnings
- Comparison of testing results across configurations
- Multiple sites can contribute testing results
- Empowers developers to test software in an extreme programming environment
- Email notifications can be sent to developers when errors occur
- Compatible with existing versions of CMake 2.4 and greater
- Easy conversion from Dart1 dashboards
- Formatted for the iPhone (demo)
- Graphing of test times and failures over time for individual tests

System requirements

- MySQL database (5.x and higher)
- PHP (5.0 recommended)
- XSL module for PHP
- cURL module for PHP (for site statistics)
- GD module for PHP (for regression tests)

Kitware is now using CDash to test our major open-source projects. Those dashboards can be found here: http://public.kitware.com/CDash

CDash is being released under the BSD license. Kitware offers courses and hosting options for CDash and the Kitware Quality Software Process. For more information about CDash see www.cdash.org.
PARAVIEW 3.2 RELEASED (NOVEMBER 2007)

In November 2007, Kitware, Sandia National Laboratories, CSimSoft and Los Alamos Laboratory announced the release of ParaView 3.2.1 (stable). The ParaView 3.2 release is available for download from the ParaView website: http://www.paraview.org/HTML/Download.html. It is also available through CVS; the tag is ParaView-3-2. This release includes the following enhancements/fixes:

- New animation view for viewing, creating and modifying animations, making ParaView more intuitive and easier to use.
- The “Element Inspector” has been replaced by a brand new “Spreadsheet View” which makes it possible to view raw data produced by any source/filter.
- Filters with multiple outputs and inputs are now supported; this has opened doors for several new filters such as glyph with custom source, stream tracer with custom point source, etc.
- There are several improvements to charts including support for vector data in line charts (magnitude or component); support for labels; titles and legends; and custom chart axes end points.
- Selection support has undergone a major overhaul. It is now possible to select cells (or points) on geometry surfaces or within a view frustum by simply dragging on the 3D view. The properties of the selection can be inspected/changed using the brand new “Selection Inspector” accessible through the “View” menu. The “Extract Selection” filter can be used to extract the selected cells or points. Labels can be used to label the selected cells or points.
- Support for eXtensible Data Model and Format (XDMF) files is back and improved to support composite datasets, quadratic elements and mixed topologies. It is now possible to change domain and grid selections as well.
- We are working on making ParaView function with ease when working with composite datasets. As a part of the process histogram (and soon probe) filters now work with composite datasets. The python client can fetch composite datasets as well as perform operations such as min/max/sum on these dataset.
- A new python module is introduced which makes dealing with proxies convenient. A separate document is available describing these python enhancements.
- This version has many GUI usability improvements including, but definitely not limited to:
  - View based and application settings
  - Using mouse clicks for picking the center of rotation and point widget location
  - We are constantly trying to make the documentation better. The online help has been extended to include the new functionalities added since 3.0.

KWWIDGETS

Besides bug fixes and minor tweaks to widgets, the major change was to upgrade both VTK and KWWidgets to support the new Tcl/Tk major 8.5 release. Both toolkits are compatible with 8.4 and 8.5 and while KWWidgets will behave the same way with either version, it will benefit from a few improvements when built against 8.5 (i.e. a better look and feel, antialiased fonts, slightly better performance, etc.)

IGSTK

IGSTK (The Image Guided Surgery Toolkit) developed in collaboration between the ISIS Center at Georgetown University and Kitware released its version 3.0 on February 15. The new release was demonstrated at the SPIE Medical Imaging conference in San Diego, CA.

Both books may be ordered through Kitware's online store (www.kitware.com) or through Amazon by searching their ISBN.

TECHNICAL BOOKS UPDATE

Kitware is pleased to announce availability of the fourth edition of the Mastering CMake book and the updated, for Version 3, ParaView Guide.

Both books may be ordered through Kitware's online store (www.kitware.com) or through Amazon by searching their ISBN.

CMake Book

CMake, a key part of the Kitware Quality Software Process, is the tool used to drive the building, testing and packaging of our software projects. By generating native makefiles and workspaces that can be used in the compiler environment of your choice, CMake allows you to keep your current development environment! CMake is both sophisticated and easy to use: it is possible to support complex environments requiring system configuration, compiler feature detection, pre-processor generation, code generation, and template instantiation using simple platform and compiler independent configuration files. CMake supports testing and is integrated with the new CDash regression testing system and is used by many large and complex systems including VTK, The Visualization Toolkit; ITK, The Insight Segmentation and Registration Toolkit; and the popular Linux desktop software KDE.

The 385-page Mastering CMake book (ISBN 1-930934-20-3) contains everything you need to know about building, testing and packaging projects with CMake.

ParaView Guide

The ParaView Guide has been updated for ParaView 3.2; this is the first edition of the book to cover the latest major revision of ParaView. The book, presented in full color, describes how to use ParaView to display and analyze data and also describes how the program is constructed so that software developers can make use of and extend its functionality.

The 366-page ParaView Guide (ISBN 1-930934-21-1) contains everything you need to know about ParaView's new Qt-based client interface, multiple view visualization; new chart, plot and spreadsheet view types; visualization of time varying data; interactive selection and data labeling; python scripting and C++ plug-in development.
VTK 5.0 introduced composite datasets. Composite datasets are simply datasets comprised of other datasets. This notion is useful in defining complex structures comprising other smaller components (e.g., an unstructured grid for a car made of grids for the tires, chassis, seats, etc.). It is also used for representing datasets with adaptive mesh refinement (AMR). AMR refers to the technique of automatically refining certain regions of the physical domain during a numerical simulation.

The October 2006 Kitware Source included an article describing the composite datasets in VTK and how to use them. Since then, the implementation of composite datasets in VTK has undergone some major reworking. The main goal was to make the use of these datasets simple and intuitive, this article describes those changes. For a better understanding of composite datasets and their usage please refer to the original October 2006 article. These changes should make it into VTK 5.2.

COMPOSITE DATASETS

The new class hierarchy for composite datasets is as follows:

As is obvious from the above diagram, we have 3 concrete subclasses of vtkCompositeDataSet. vtkMultiBlockDataSet is a dataset comprising of blocks. Each block can be a non-composite vtkDataObject subclass (or a leaf) or an instance of vtkMultiBlockDataSet itself. This makes it possible to build full trees. vtkHierarchicalBoxDataSet is used for AMR datasets which comprise refinement levels and uniform grid datasets at each refinement level. vtkMultiPieceDataSet can be thought of as a specialization of vtkMultiBlockDataSet where none of the blocks can be a composite dataset. vtkMultiPieceDataSet is used to group multiple pieces of a dataset together.

vtkCompositeDataSet is the abstract base class for all composite datasets. It provides an implementation for a tree data structure. All subclasses of composite datasets are basically trees of vtkDataSet instances with certain restrictions. Hence, vtkCompositeDataSet provides the internal tree implementation with protected API for the subclasses to access this internal tree, while leaving it to the subclasses to provide public API to populate the dataset. The only public API that this class provides relates to iterators.

Iterators are used to access nodes within a composite dataset. Here’s an example showing the use of an iterator to iterate over non-empty, non-composite dataset nodes.

As we can see, accessing nodes within a composite dataset hasn’t really changed. However, the generic API provided by vtkCompositeDataSet for setting datasets using an iterator makes it possible to create composite dataset trees with identical structures without having to downcast to a concrete type. The following figure is an example of an outline filter that applies the standard outline filter to each leaf dataset with the composite tree; the output is also a composite tree with each node replaced by the output of the outline filter.

By default, the iterator only visits leaf nodes (i.e. non-composite datasets within the composite tree). This can be changed by toggling the VisitOnlyLeaves flag. Default behavior to skip empty nodes can be avoided by setting the SkipEmptyNodes flag to false. Similarly, to avoid traversing the entire sub-tree, instead of just visiting the first level children, set the TraverseSubTree flag to false.

To make it possible to address a particular node within a composite tree, the iterator also provides a flat index for each node. Flat index for a node is the index of the node in a preorder traversal of the tree. In the following diagram the preorder traversal of the tree yields: A, B, D, E, C. Hence the flat index for A is 0, while the flat index for C is 4. Filters such as vtkExtractBlockFilter use the flat index to identify nodes.
MULTIPiece DATASET
vtkMultiPieceDataSet is the simplest of all composite datasets. It is used to combine a bunch of non-composite datasets together. It is useful to hold pieces of a dataset partitioned among processes, hence the name. To reiterate, a piece in a multi-piece dataset cannot be a composite dataset.

MULTIBLOCKDATASET
A vtkMultiBlockDataSet is a composite dataset comprised of blocks. It provides API to set/access blocks such as SetBlock, GetBlock, GetNumberOfBlocks, etc. A block can be an instance of vtkMultiBlockDataSet or any other subclass of vtkDataSet object which is not a vtkCompositeDataSet. MultiBlock datasets no longer support the notion of subdatasets with a block. To achieve the same effect, one can add a vtkMultiPieceDataSet as the block and then put the subdatasets as pieces in the vtkMultiPieceDataSet.

HIERARCHICALBOX DATASET
vtkHierarchicalBoxDataSet is used to represent AMR datasets. It comprises levels of refinements and datasets associated with each level. The datasets at each level are restricted to vtkUniformGrid. vtkUniformGrid is vtkImageData with blanking support for cells and points. Internally, vtkHierarchicalBoxDataSet creates a vtkMultiPieceDataSet instance for each level. All datasets at a level are added as pieces to the multipiece dataset.

vtkHierarchicalBoxDataSet has been deprecated and is no longer supported because it is not much different from a vtkMultiBlockDataSet.

PIPELINE EXECUTION
It is possible to create mixed pipelines of filters which can or cannot handle composite datasets. For filters that are not composite data aware, vtkCompositeDataPipeline executes the filter for each leaf node in the composite dataset to produce an output similar in structure to the input composite dataset. In the previous implementation of this executable, the output would always be a generic superclass of the concrete composite datasets. In other words, if the vtkCellDataToPointData filter was inserted into a composite data pipeline and the input was a vtkHierarchicalBoxDataSet, the output would still be vtkMultiGroupDataSet. This has been changed to try to preserve the input data type. Since vtkCellDataToPointData does not change the data type of the input datasets, if the input is vtkHierarchicalBoxDataSet now, the output will be vtkHierarchicalBoxDataSet. However, for filters such as vtkContourFilter where the output type is not a vtkUniformGrid, the output will be vtkMultiBlockDataSet with structure similar to the input vtkHierarchicalBoxDataSet.

EXTRACTION FILTERS
A few new extraction filters have been added which enable extracting component dataset from a composite dataset.

EXTRACT BLOCK
vtkExtractBlock filter is used to extract a set of blocks from a vtkMultiBlockDataSet. The blocks to extract are identified by their flat indices. If PruneOutput is true, then the output will be pruned to remove empty branches and redundant vtkMultiBlockDataSet nodes (i.e. vtkMultiBlockDataSet node with a single child which is also a vtkMultiBlockDataSet). The output of this filter is always a vtkMultiBlockDataSet, even if a single leaf node is selected to be extracted.

EXTRACT LEVEL
vtkExtractLevel is used to extract a set of levels from a vtkHierarchicalBoxDataSet. It simply removes the datasets from all levels except the ones chosen to be extracted. It always produces a vtkHierarchicalBoxDataSet as the output.

EXTRACT DATASETS
vtkExtractDataSets is used to extract datasets from a vtkHierarchicalBoxDataSet. The user identifies the datasets to extract using their level number and the dataset index within that level. Output is a vtkHierarchicalBoxDataSet with same structure as the input, but with only the selected datasets.

CONCLUSION
With the redesign, composite datasets now use a full tree data structure to store the datasets rather than the table-of-tables approach used earlier. This makes it easier to build/pause the structure. Iterators have been empowered and can now be used to get, as well as set, datasets in the composite tree, thus minimizing the need to downcast to concrete subclasses for simple filters.

Utkarsh Ayachit is an R&D Engineer in Kitware’s Clifton Park, NY office. Mr. Ayachit is the project lead for GPU-based extensions to VTK (www.vtk.org), and is also a developer on the ParaView project (www.paraview.org).

Although VTK has traditionally been used in scientific visualization fields, there are a number of new capabilities aimed at making this toolkit a very powerful information visualization resource. These new features are collectively called the Titan Informatics Toolkit, a joint effort between Sandia National Laboratories and Kitware. As opposed to scientific visualization, where the data has inherent spatial or temporal attributes, general information visualization requires the extra step of embedding the data in two or three dimensions before being visualized. One of the most widely used forms of information visualization is visualizing networks of information. In this introduction we will call these networks “graphs”, which is a set of objects (“vertices”) with connections between these objects (“edges”). This article focuses on creating and visualizing graphs with Titan.

GETTING STARTED
The following is a “Hello, World” program that generates a small random graph and displays it in a view. All code is available in the Examples/Infovis/Cxx directory of VTK.

```
#include "vtkRenderWindowInteractor.h"
#include "vtkRandomGraphSource.h"
#include "vtkRenderWindow.h"
#include "vtkGraphLayoutView.h"

int main(int argc, char* argv[]) {
  vtkRandomGraphSource* source = vtkRandomGraphSource::New();
  vtkGraphLayoutView* view = vtkGraphLayoutView::New();
```
We can also show labels on the vertices using an array to label the vertices by vertex degree, add:

```cpp
view->SetVertexLabelArrayName("VertexDegree");
view->SetVertexLabelVisibility(true);
```

This results in the following when we run our application (Theme.cxx):

![Graph Visualization]

**CREATING GRAPHS AND TREES**

Now suppose we have a hierarchical structure (e.g. a corporate structure) which we want to encode into a graph programmatically instead of randomly, we could define these relationships in a vtkGraph, but in this case it makes sense to use the more specialized structure, vtkTree. Before creating a tree, we will need to understand the class hierarchy of Titan graph data structures.

The blue classes are the main data structure classes that flow through a VTK pipeline. At the top level, we distinguish graphs whose edges have inherent order from source to target (directed graphs) from graphs whose edges do not indicate direction (undirected graphs). The directed graph subclasses are naturally structured by specialization. A directed acyclic graph (i.e. a graph with no paths that lead back to the same place) is a subset of the class of all directed graphs. A vtkTree further restricts this by enforcing a hierarchy: every vertex but the root must have exactly one parent (incoming edge). The green classes are the mutable classes used to create or modify graphs. The structure of a graph may be copied into any other graph instance, given that it first passes a compatibility test.

Coming back to our example, the hierarchical graph is first created using an instance of vtkMutableDirectedGraph. As long as we created the tree properly, we can pass this into a vtkTree instance using CheckedShallowCopy. We also create an array of strings used to label the vertices in the graph.

```cpp
vtkMutableDirectedGraph* graph = vtkMutableDirectedGraph::New();
vtkIdType a = graph->AddVertex();
vtkIdType b = graph->AddChild(a);
vtkIdType c = graph->AddChild(a);
vtkIdType d = graph->AddChild(c);
vtkIdType e = graph->AddChild(c);
vtkIdType f = graph->AddChild(c);
```
vtkStringArray* labels = vtkStringArray::New();
labels->SetName("Label");
labels->InsertValue(a, "a");
labels->InsertValue(b, "b");
labels->InsertValue(c, "c");
labels->InsertValue(d, "d");
labels->InsertValue(e, "e");
labels->InsertValue(f, "f");
graph->GetVertexData()->AddArray(labels);

vtkTree* tree = vtkTree::New();
tree->CheckedShallowCopy(graph);

This tree may then be placed in our view to get the following result (CreateTree.cxx):

**WORKING WITH MULTIPLE VIEWS**
The real power of information visualization comes from the ability to look at information in many different ways. Each method of visualization provides a unique look into the data and provides its own insights. Titan supports multiple views with linked selection. Now let’s add a tree layout view, which is more informative than the general graph layout. We do this with the following:

vtkTreeLayoutView* view2 = vtkTreeLayoutView::New();
vtkDataRepresentation* rep2 = view2->AddRepresentationFromInput(tree);
view2->SetLabelArrayName("Label");
view2->SetLabelVisibility(true);
vtkRenderWindow* window2 = vtkRenderWindow::New();
view2->SetupRenderWindow(window2);

In order to share the same selection between the two views, we use an object called vtkSelectionLink this object links selections between multiple representations. To link the views, simply add the following lines:

vtkSelectionLink* link = vtkSelectionLink::New();
rep->SetSelectionLink(link);
rep2->SetSelectionLink(link);

Now, both views refer to the same selection. When the user starts the application, two windows appear. The only issue is that when a selection is made in one view, the other view does not know to update. To resolve this, we make a listener that updates the other view when one view receives a SelectionChangedEvent. For the details of how to do this, we direct the reader to MultiView.cxx.

**ACKNOWLEDGMENTS**
Thanks to an adapter to the Boost Graph Library, we are able to use the rich set of algorithms in that library to implement several VTK filters. The upcoming VTK 5.2 release will be the first to contain these new information visualization capabilities. We are currently working on distributed graph data structures and algorithms with Sandia and Indiana University. We are also creating general-purpose interactive Qt-based charting capabilities.

We have touched the surface of what types of information visualization can be done in VTK. Please investigate the VTK/Examples/Infovis directory and the nightly VTK class documentation at vtk.org for additional capabilities.

Jeff Baumes is an R&D Engineer in Kitware’s Clifton Park, NY office. Mr. Baumes is a developer for the Titan Informatics Toolkit.
Unfortunately, the existing openly-available implementation of QuadEdge data structures are in libraries that only deal with mesh processing.

itkQuadEdgeMesh is an implementation of the [1] data structure in ITK. It respects most of the existing itk::Mesh API as far as 2-manifolds are concerned. itk::Mesh is still the only choice for representing n-manifolds.

Euler Operators
One advantage of the itkQuadEdgeMesh data structure is having only two operators to modify its connectivity. It is thus very robust and easy to maintain. Unfortunately, the splice operator is not very intuitive, to say the least. There is a need for user-friendly higher order operators. Many discrete surface processing filters in the publications are based on Euler operators (operators that do not modify the Euler Characteristic of the discrete surface). Subdivision, remeshing, decimation, and many other filters can be very easily written using Euler Operators. Naturally, we also implemented Euler Operators for itkQuadEdgeMesh.

A mesh data structure can quickly become too big to be maintainable. Each new feature, if added as a new method of the class, makes the class more difficult to maintain and more complicated to use. The alternative is to provide the functionality as a filter at the cost of duplicating the data from the input to the output. When you want to implement a very local operation, like flipping an edge, this is overkill.

We decided to implement the Euler Operators as functors.

Adding Functionality: Functions versus Filters

Flexible and Maintainable Solution: Functions
We decided to implement in the base class (itkQuadEdgeMesh) the minimum set of features to be as backward-compatible as possible with itkMesh. Additional features, as much as possible, have been implemented as Functors.

QuadEdgeMeshFunctionBase offers a lightweight alternative to an ITK Filter. Subclasses of QuadEdgeMeshFunctionBase, which should override Evaluate(), are function objects that apply reduced and localized modifications (geometry or connectivity) to the InputType mesh. It is up to filter developers to use those functions inside the filter and enforce const-correctness. The user can take a look at all the Euler Operators for examples.

Using Functions in a Filter

itkQuadEdgeMeshToQuadEdgeMeshFilter provides an abstract class for those who want to develop a filter using the above mentioned MeshFunctions. It basically copies the input mesh to the output, with possibly different types, enforcing the const correctness of the input. It is specialized for itkQuadEdgeMesh.

PROS AND CONS OF USING itkQuadEdgeMesh

Existing Code

itkQuadEdgeMesh respects, as much as possible, the API of itkMesh. ITK code being templated, it is enough in most cases to redefine the MeshType to be an itkQuadEdgeMsh instead of an itkMesh to have the existing code work. Testing of the existing code base is in progress, and the most recent results are displayed on the wiki page here [3].

The filters using the cell links will not be able to use itkQuadEdgeMesh. Neither will filters that use Boundary features.

The reason is that itkQuadEdgeMesh handles those features internally and we did not want to replicate external structures and arrays just to give API compatibility. Fortunately, it does not seem to be a problem for most of the code in ITK today. Users whose code does not work straightforwardly are welcome to contact us on the mailing list. We also provide some migration details on the above-mentioned wiki page.

New Code

The new code should be smaller! Lot of cases that needed to be dealt with are now taken care of in the structure. New operators, iterators, and circulators are defined to simplify the coding. In brief, focus on the algorithm, not on the structure. For an example of how you can use these new operators to shorten your code see the tutorial later in this article.

Preliminary Results

The readers (like itkVTKPolyDataReader) will work more slowly. QuadEdgeMesh has one more layer than itkMesh, and thus, it takes more time. The readers will be three times slower on average. The good news is that the gain in speed in the pipeline will be much bigger, and overall the user should experience a gain in speed. One example of this is the itkRegularMeshSphereSourceTest. This test creates a spherical mesh of a few points, then subdivides it until it is dense enough. It mixes mesh creation (cell insertion/deletion) with processing, and the same source code is faster using itkQuadEdgeMesh than using itkMesh.

FUTURE WORK

Remodel the Existing Filters and Framework

Now that the data structure and Euler Operators are implemented, we would like to redesign the simplex mesh active surface framework and the actual mesh processing filters. The mesh processing filters are not that difficult, and the remodeling effort has already started (see [3]). Remodeling the simplex mesh active surface framework will impact quite a few filters and should be more challenging as the backward compatibility must be maintained. We hope to involve the community in this effort.

Transferring Mesh-Processing Filters from VTK to ITK

We want to recode the usual mesh processing filters (again, see [2]) in ITK. We would like to stay very close to the VTK implementation API when such implementations exist. We already have working implementation of vtkCleanPolyData, vtkConnectivityFilter, vtkDecimate, vtkSmoothPolyData, etc., on top of itkQuadEdgeMesh. They need a little bit of polishing before we can transfer them to ITK. That would help in avoiding time/memory-consuming and pipeline-breaking transitions between ITK and VTK. We would still need to switch to VTK for visualization but not for processing.

New Filters

We would also like to add more geometry and topology filters.

It would be nice to have a subdivision framework that deals with subdivisions like people do in publications: separating connectivity updates from geometry updates with different well-defined schemes.

It would then be nice to have multithreading, clustering, and/or streaming support. (Everyone needs to dream.)
WALKING ON THE SURFACE: A TUTORIAL

Tutorial (1): Global Mesh Traversal with Containers
Traversing a Mesh can be achieved through the “classic” usage of the inherited itk::Mesh containers and the associated iterators.

Traversal of Points Container
The following code snippet illustrates a traversal of the points container. (See itk::Mesh::GetPoints().)

typedef MeshType::PointsContainer PointsContainer;
typedef PointsContainer::Iterator PointsContainerIterator;
MeshType::Pointer MeshInstance;

PointsContainer* points = MeshInstance->GetPoints();
for (PointsContainerIterator pointIterator = points->Begin();
    pointIterator != points->End();
    pointIterator++)
{
    MeshType::PointType PointVariable =
        pointIterator.Value();
    MeshType::PointIdentifier PointIndex =
        pointIterator.Index();
    ... }

Traversal of Cells Container
The following code snippet illustrates a traversal of the cells container. (See itk::Mesh::GetCells().)

typedef MeshType::CellType CellType;
typedef MeshType::CellType::CellsContainer CellsContainer;
typedef CellsContainer::Iterator CellsContainerIterator;
MeshType::Pointer MeshInstance;

unsigned int numberOfEdges = 0;
CellsContainer cells = MeshInstance->GetCells();
for (CellsContainerIterator cellIterator =
    cells()->Begin();
    cellIterator != cells()->End();
    cellIterator++)
{
    CellType* cell = cellIterator.Value();
    ... }

Tutorial 2: Walking Locally on an itk::QuadEdgeMesh
Accessing Adjacent Edges
Based on itk::QuadEdge::GetOnext() and itk::QuadEdge::GetRot(), Guibas and Stolfi define derived operators (refer to [1], or take a look at the insight journal paper [4]) which act as convenient shortcuts for accessing the edges adjacent to a given edge.

The basic (first order) operators follow.
• itk::QuadEdge::GetOnext()
• itk::QuadEdge::GetRot()
• itk::QuadEdge::GetSym()
• itk::QuadEdge::GetLnext()
• itk::QuadEdge::GetRnext()
• itk::QuadEdge::GetDnext()
• itk::QuadEdge::GetOprev()
• itk::QuadEdge::GetLprev()
• itk::QuadEdge::GetRprev()
• itk::QuadEdge::GetDprev()

The so called “inverse operators” follow.
• itk::QuadEdge::GetInvRot()
• itk::QuadEdge::GetInvOnext()
• itk::QuadEdge::GetInvLnext()
• itk::QuadEdge::GetInvRnext()
• itk::QuadEdge::GetInvDnext()

Technically, the above accessors are defined by the itk::QuadEdge layer and inherited by itk::GeometricalQuadEdge.

Accessing Adjacent Edges
Based on itk::QuadEdge::GetOnext() and itk::QuadEdge::GetRot(), Guibas and Stolfi define derived operators (refer to [1], or take a look at the insight journal paper [4]) which act as convenient shortcuts for accessing the edges adjacent to a given edge.

The basic (first order) operators follow.
• itk::QuadEdge::GetOnext()
• itk::QuadEdge::GetRot()
• itk::QuadEdge::GetSym()
• itk::QuadEdge::GetLnext()
• itk::QuadEdge::GetRnext()
• itk::QuadEdge::GetDnext()
• itk::QuadEdge::GetOprev()
• itk::QuadEdge::GetLprev()
• itk::QuadEdge::GetRprev()
• itk::QuadEdge::GetDprev()

The so called “inverse operators” follow.
• itk::QuadEdge::GetInvRot()
• itk::QuadEdge::GetInvOnext()
• itk::QuadEdge::GetInvLnext()
• itk::QuadEdge::GetInvRnext()
• itk::QuadEdge::GetInvDnext()

Technically, the above accessors are defined by the itk::QuadEdge layer and inherited by itk::GeometricalQuadEdge.

One Local Step Beyond
Of course, those accessors can be concatenated to walk a little further on the edges. When the geometry is associated with the topology, the GetDestination() and GetOrigin() methods provide access to the two points to which the QuadEdge links.

In the following code snippet, when given an edge adjacent to a triangle on its Left() side (which is asserted), we retrieve the itk::PointIdentifier of the point opposite the given edge of this triangle.

```cpp
if (edge->GetLeft()->IsTriangle())
{
    // Grab the point that is opposite to “edge” and
    // “do something”
    PointIdentifier oppositePoint =
        edge->GetLnext()->GetDestination();
    ... }
```

Iterating Even Further
By definition, Onext() and Rot() operators yield cyclic groups. More precisely, the successive images of any given edge (either with Onext() or Rot()) describe a closed path that eventually yields the initial edge. Other derived operators also possess this property (e.g., Lnext() and Lprev()). For any such operator, it is therefore natural to define an associated iterator that, seeded with any given edge, will follow the orbit of the given operator until it reaches the initial given edge.

For a given edge, the following code snippet builds the list (order matters) of the point identifiers of all the points sharing an edge with the Org() of the given edge.
// Assuming MeshType of type itkQuadEdgeMesh
// is defined above:
typedef MeshType::QEType QEType;
MeshType::PointIdList pointSharingEdgeList;
QEType* givenEdge = ...;
for (QEType::IteratorGeom it =
givenEdge->BeginGeomOnext();
it != otherBorderEdge->EndGeomOnext();
it++)
{
    pointSharingEdgeList.push_back(
        it.Value()->GetDest());
}

The following example illustrates a possible usage of the
itk::QuadEdgeMesh::BeginGeomLnext() iterator. Given a
boundary on a surface, the following snippet “removes” that
boundary by building a polygonal face (which has as many
vertices as there are points on the designated boundary) and
 gluing it to that boundary. The code follows.

// We assume bdryEdge is an edge on the boundary
// of the surface AND the surface is on the
// Right() side of bdryEdge:
QEType* bdryEdge = ...;
// Follow the boundary, with Lnext()
typename OutputMeshType::PointIdList pList;
typename QEType::IteratorGeom it =
bdryEdge->BeginGeomLnext();
while (it != bdryEdge->EndGeomLnext())
{
    pList.push_back(it.Value()->GetOrg());
    it++;
}
mesh->AddFace( pList );

Tutorial 3: Specialized Walks
Front-Based Traversal: FrontIterator
An itk::QuadEdgeMesh can be visited with a front method
which takes the form of an iterator. When running on
the primal edges (i.e., itk::QuadEdgeMesh::QEType), one
should use FrontIterator, but when running on the dual
edges (i.e., itk::QuadEdgeMesh::QEDual) one should use the
FrontDualIterator. Both fronts are initialized with an arbi-
trary seed by default. The following code snippet illustrates
the usage of these.

// The following counts the number of points in the
// first encountered component of a given Mesh. For
// this we use a FrontIterator (i.e., on Primal
// edges) to visit all the points of the component.
// We assume MeshType was defined as an
// instantiation of itk::QuadEdgeMesh.
typedef MeshType::PointIdentifier PointIdentifier;
typedef MeshType::FrontIterator FrontIterator;
typedef FrontIterator::QEType QEType;
// Get our hands back on the instance of this
// MeshType.
MeshType::MeshPointer mesh = this->GetInput();
typedef std::set< PointIdentifier > visitedSet;
FrontIterator it = mesh->BeginFront( )
while (it != mesh->EndFront())
{
    QEType* edge = it.Value();
    PointIdentifier org = edge->GetOrigin();
    if (!visitedSet.count( org ))
        visitedSet.insert( org );
    PointIdentifier dest = edge->GetDestination();
    if (!visitedSet.count( dest ))
        visitedSet.insert( dest );
    it++;
}
    std::cout << "Number of points of component: " << visitedSet.size();

The dual version usage is completely similar (which comes as
no surprise since itk::QuadEdgeMesh::FrontIterator and itk::
QuadEdgeMesh::FrontDualIterator are two instantiations of
the same template). The following code snippet illustrates
the syntax.

typedef MeshType::FrontDualIterator
FrontDualIterator;
typedef FrontDualIterator::QEType QEDual;
for (FrontDualIterator it =
m_Mesh->BeginDualFront();
it != m_Mesh->EndDualFront();
it++)
{
    QEDual* edge = it.Value();
    ...

Traversal of Mesh Boundaries
Consider a mesh with boundaries (e.g., a cylinder) that
one wants to display. Assume the boundaries convey some
meaningful information that one wants to visually enhance
when displaying the mesh (e.g., the borders carry some
bright color). One would then need to walk on the mesh
boundaries. The following code snippet achieves this goal
by combining itk::QuadEdgeMeshBoundaryEdgesMeshFunct
ion and an iteration of QEType::IteratorGeom::BeginGeomL
ext(). (See “Iterating Even Further”.)

• itk::QuadEdgeMeshBoundaryEdgesMeshFunction builds a
list of references to edges (as QuadEdgeGeom::RawPointer)
each of which represent a different boundary compo-
nent.

• Consider each boundary representative edge, and walk
the boundary by iteratively using BeginGeomLnext().

With our representative boundaryEdge in hand, we can now
walk (or follow) the boundary that boundaryEdge repre-
sents. For this, it suffices to use the Lnext() operator, and
within this loop we can do something smart with the current
edge (i.e., it.Value()).

typename TMesh::PointType p;
p = mesh->GetPoint( it.Value()->GetOrigin() );
...

REFERENCES
the manipulation of general subdivisions and the computa-
tion of Voronoi diagrams”. ACM Transactions on Graphics,
ar?q=guibas+stolfi+primitives+voronoi
Levy, Stephan Bischo, Christian Rssl "Geometric Modeling
Based on Polygonal Meshes", SIGGRAPH 2007 Course
sg07-course.pdf
Transparency is an effective visualization tool for viewing inner details while maintaining global structural information. However, since the blending operation for transparent rendering requires all primitives contributing to a pixel to be considered in depth-sorted order, interactive transparency is notoriously difficult to implement in an object-oriented visualization toolkit, such as VTK.

This article covers new functionalities recently added to VTK to make use of OpenGL extensions that can be utilized to essentially perform the depth sorting as a multi-pass rendering technique. For simple scenes, this can provide interactive rendering with correct transparency effects.

In VTK, opacity can be found in many places including the Opacity ivar in the vtkProperty, the alpha channel in a vtkTexture of a vtkActor, or the alpha channel of a vtkLookupTable. Note that opacity and alpha are equivalent terms and mean the opposite of transparency, since an object that is 100% opaque is 0% transparent. The following code snippets illustrate some of the ways transparency can be introduced into a scene in VTK.

C++ code:

```cpp
vtkActor *actor=vtkActor::New();

// Opacity on the property.
vtkProperty *property=vtkProperty::New();
property->SetOpacity(0.3);
actor->SetProperty(property);

// Opacity on the texture
// Assuming the file as an alpha channel.
vtkPNGReader *reader=vtkPNGReader::New();
reader->SetFileName("texture_with_alpha.png");
reader->Update();
vtkTexture *texture=vtkTexture::New();
texture->SetInput(reader->GetOutput());
actor->SetTexture(texture);
```

In order to render a full scene mixing some opaque polygonal geometry and some translucent polygonal geometry, the opaque polygonal geometry is rendered first, using and updating the depth buffer. In a second pass, the translucent geometry is rendered using an alpha blending operation. As this operation is not commutative, the translucent polygons of the whole scene have to be rendered back-to-front. Therefore, the translucent polygons must be depth-sorted according to the viewpoint. Sorting is an expensive operation, but if sorting is not performed, visual artifacts will occur.

So far in VTK, the rendering of translucent polygonal geometry is performed through alpha blending but it is up to the user to perform the sorting operation with a vtkDepthSortPolyData filter. However this filter can only sort polygons per actor not for the whole scene.

**Figure 1:** alpha blending without depth sorting

**Figure 2:** alpha blending with depth sorting (depthSort.png)

**Figure 3:** depth peeling without depth sorting

In order to render a full scene mixing some opaque polygonal geometry and some translucent polygonal geometry, the opaque polygonal geometry is rendered first, using and updating the depth buffer. In a second pass, the translucent geometry is rendered using an alpha blending operation. As this operation is not commutative, the translucent polygons of the whole scene have to be rendered back-to-front. Therefore, the translucent polygons must be depth-sorted according to the viewpoint. Sorting is an expensive operation, but if sorting is not performed, visual artifacts will occur.

So far in VTK, the rendering of translucent polygonal geometry is performed through alpha blending but it is up to the user to perform the sorting operation with a vtkDepthSortPolyData filter. However this filter can only sort polygons per actor not for the whole scene.

In order to render a full scene mixing some opaque polygonal geometry and some translucent polygonal geometry, the opaque polygonal geometry is rendered first, using and updating the depth buffer. In a second pass, the translucent geometry is rendered using an alpha blending operation. As this operation is not commutative, the translucent polygons of the whole scene have to be rendered back-to-front. Therefore, the translucent polygons must be depth-sorted according to the viewpoint. Sorting is an expensive operation, but if sorting is not performed, visual artifacts will occur.

So far in VTK, the rendering of translucent polygonal geometry is performed through alpha blending but it is up to the user to perform the sorting operation with a vtkDepthSortPolyData filter. However this filter can only sort polygons per actor not for the whole scene.
Recently a new technique was implemented in VTK that allows users to render the translucent polygonal geometry of the whole scene without sorting the polygons. This order independent transparency (OIT) method is called depth peeling [1].

Using depth-peeling in VTK requires following a few simple steps:

1. Make sure the render window has alpha bits (as the initial flag value is 0 (false)):

   ```
   vtkRenderWindow *renderWindow=vtkRenderWindow::New();
   renderWindow->SetAlphaBitPlanes(1);
   ```

2. Tell the renderer to use depth peeling (the initial flag value is 0):

   ```
   vtkRenderer *renderer=vtkRenderer::New();
   renderer->SetUseDepthPeeling(1);
   ```

3. Set the parameters that control the ending of the depth peeling loop (more on this later).

   3.a Set the maximum number of rendering passes (initial value is 4):

   ```
   renderer->SetMaximumNumberOfPeels(100);
   ```

   3.b Set the occlusion ratio (initial value is 0.0):

   ```
   renderer->SetOcclusionRatio(0.1);
   ```

4. Render the scene:

   ```
   renderWindow->Render();
   ```

5. Check if depth peeling was actually used:

   ```
   int used=renderer->GetLastRenderingUsedDepthPeeling();
   ```

Even if the UseDepthPeeling flag is set, depth peeling is only performed if the following features are available with the OpenGL driver in use, otherwise regular alpha blending method is used:

- GL_ARB_depth_texture or OpenGL>=1.4
- GL_ARB_shadow or OpenGL>=1.4
- GL_EXT_shadow_funcs or OpenGL>=1.5
- GL_ARB_vertex_shader or OpenGL>=2.0
- GL_ARB_fragment_shader or OpenGL>=2.0
- GL_ARB_shader_objects or OpenGL>=2.0
- GL_ARB_occlusion_query or OpenGL>=1.5
- GL_ARB_multitexture or OpenGL>=1.3
- GL_ARB_texture_rectangle
- GL_SGIS_texture_edge_clamp or GL_EXT_texture_edge_clamp or OpenGL>=1.2

On Unix, glxinfo lists the version and available extensions of the OpenGL driver. On Windows, glview does a similar job (http://www.realtech-vr.com/glview)

Playing with the depth peeling parameters requires some understanding of the algorithm itself. The algorithm has two stages. The first one peels the translucent polygonal geometry in an iterative process from the nearest geometry to the farthest geometry at a pixel level, using an extra depth buffer and saving each peel in a color buffer. The second stage just performs alpha blending of the intermediate color buffers in back to front order.

Each peeling iteration renders all the translucent geometry: just having 4 peels slows down the rendering of the scene by a factor of 4. The iteration of the first stage stops if either the maximum number of peels set by the user is reached (set through SetMaximumNumberOfPeels()), or if the number of pixels modified in the current peel is under a threshold set by the user (set through SetOcclusionRatio() as a ratio of the number of modified pixels over the total number of pixels in the color buffer).

Setting a low value for the MaximumNumberOfPeels prevents the depth peeling process from being too slow but the result is not guaranteed to be exact. Using the occlusion ratio to limit the number of depth peeling passes allows the termination criteria to be controlled by visual impact, but may result in prohibitively slow rendering for complex scenes.

Similar information and more implementation details are available on the VTK Wiki located at VTK.org. Use the keywords “Depth Peeling” to find the article.

REFERENCES


Dr. François Bertel is a technical developer at Kitware, Inc. His current interests include GPU programming applied to visualization.

IN PROGRESS

**PARAVIEW 3.4**

Composite DataSet Redesign

One of the major targets for 3.4 is to support composite datasets natively. As a starting point, we restructured the composite dataset hierarchy to make it more use-case driven and easy to understand than before. Composite datasets now support a full tree data structure, instead of the table-of-tables approach used earlier. Composite Data iterators are now more powerful in that they can be used to not only access datasets in a composite dataset, but also initialize/set them. This makes it possible for filters to work with iterators directly without having to downcast to a concrete subclass, as was the case earlier.

ParaView GUI is also undergoing major overhaul to support the user’s need to explore the composite data tree. Filters dealing with composite datasets such as ExtractBlock, ExtractLevel, ExtractDataSet now show the entire composite data set tree on the client making it possible for the user to choose the nodes that he is interested in directly using
the GUI. Similarly, the information tab also shows the entire composite tree allowing the user to see the data information about individual nodes as well.

Currently, we are working on other ParaView components such as selection, spreadsheet view so that they work seamlessly with composite dataset as well.

**Image View**

ParaView finally supports a mode in which 2D image data is no longer rendered as points, instead we upload the image as a texture and apply it to a quad. This has been extended to 3D images as well, where the user can choose a slice of the 3D image data to view.

Currently, we are sorting out issues with parallel rendering.

**Custom Filters**

Since the beginning, ParaView 3 has toyed with the idea of user-definable filter packages that could be built on the fly, include pipelines to other filters and be used simply, just like ordinary filters. This is what we call Custom Filters. Custom filters were redesigned ground up so now they work seamlessly with regular filters. It's now possible to have multiple pipelines within these filters and expose the ends of multiple pipelines as the output.

**VTK 5.2**

VTK 5.2 is scheduled for an early-April 2008 release. The VTK team is currently working on stabilizing the dashboards and cleaning up the bugs. This release is set to include Infovis and Views kits, new Widgets architecture, new and updated utilities, more than 300 C++ classes and 100 C++ tests, improvements to PLY file support, and bug fixes including improved JAVA wrapping and those specific to Mac OS X.

**SCIENTIFIC ORIGINOLOgy**

On November 2, 2007, an Invited Talk was presented at the MICCAI Open Source Workshop in Brisbane, Australia. The talk, titled “Principles and Practices of Scientific Originology”, is a satire of the current obsession with intellectual property, innovation and originality that plagues the field of medical image analysis.

The presentation makes the point that most journals and conferences focus on originality, but despise reproducibility and verification, demonstrating great disrespect for the essential elements of the scientific method. The practice of “peer-review” is offered, in most cases, as a poor substitute for the actual verification of reproducibility. In a nutshell, this presentation intends to bring awareness to the fact that authors of papers are treated, by publishers and conference organizers, as if they were writers of novels or poems instead as reporters of scientific findings based on facts. In that confusion, authors are required to be “original” instead of being required to report “reproducible” findings. Such practice obliterates critical thinking and erodes the basic principles of the scientific community. The slides of the talk (with and without voice recording) are available under a Creative Commons Attribution License 3.0 at the Insight Journal website. Go to www.insight-journal.org and search for Scientific Originology.

**HARVARD FACULTY GOES OPEN ACCESS**

On February 12, 2008, Arts and Sciences faculty at Harvard University embraced the concept of Open Access as the logical venue for the dissemination of scholarly works. This is the first open access initiative that emanates from a faculty organization (as opposed to a research funding agency). The mechanism chosen by Harvard FAS was to assign a worldwide license to all of their copyrighted articles, provided that they are not sold for a profit. For more information visit: http://www.fas.harvard.edu, go to news and events, then press releases and select the release for February 12, 2008.

**NEW NIH PUBLIC ACCESS POLICY**

As of April 7, 2008, all publications resulting from NIH funded research must be made publicly available at the PubMed Central database. As opposed to the previous NIH policy that was only an encouragement for researchers to send their papers to PubMed Central, this new directive is a mandate that requires them to do so. The mandate is an implementation of a new law passed by the US Congress on December 2007. For more information, visit: http://publicaccess.nih.gov/

**KITWARE NEWS**

**DEVELOPER’S TRAINING WEEK**

Kitware’s Developer’s Training Week will be held June 9 – 13, 2008, in Albany, NY. These hands-on courses will cover our open-source projects including VTK, ITK, ParaView and CMake, and are appropriate for both new users wishing to quickly gain proficiency and experienced developers requiring advanced customization skills.

As an added bonus, this year the VTK course will be split into two tracks: Beginner and Advanced. Users who have taken the VTK course in the past are encouraged to attend the Advanced VTK track to learn new skills and functionalities that haven’t been covered in previous VTK trainings.

The cost for registration, per half-day session, is $350 on or before May 15th and $400 from May 16th on. We are also offering a discounted full week registration rate of $2600 on or before May 15th and $2950 from May 16th on. Additional student or group discounts are available upon request. (Note that attendees must register for a minimum of 3 half-day sessions.)

No prior experience is necessary; however, attendees are expected to have basic knowledge of C or C++ in order to fully participate in the interactive exercises. Additional course information is available on our website or by emailing courses@kitware.com.

**NEW HIRES**

Dr. Andy Bauer joined Kitware in January as an R&D Engineer. Andy’s current research is in developing visualization techniques for large-scale, PDE-based numerical simulations. Prior to joining Kitware his work was primarily focused on mesh based numerical simulations of partial differential equations. Andy obtained his BS in Mechanical Engineering from Binghamton University and his PhD from the University at Buffalo, with his dissertation on Efficient Solution Procedures for Adaptive Finite Element Methods – Applications to Elliptic Problems.
Dr. Harvey Cline joined Kitware, part-time, in January. His research at Kitware involves improving both CT coronary artery segmentation and visualization. Harvey received his PhD in Material Science and BS in Physics from the Massachusetts Institute of Technology in Cambridge Massachusetts for research in superconductivity, 1965 and 1962. He was awarded a Coolidge Fellowship in 1979 and studied Physics at Stanford University. He has published more than 100 papers, holds more than 175 patents, was awarded inventor of the year for magnetic resonance angiography, and is a member of the National Academy of Engineering.

Burlen Loring joined Kitware in April as an R&D Engineer. Burlen joins us from the University of New Hampshire’s Science Center (EOS-SSC) where he developed Python-Tk and Qt-C++ visualization applications using VTK for the OpenGGCM global magnetospheric CFD code and an application for converting CHOMBO HDF5 datasets into VTK format. Burlen received his BS in Interdisciplinary Mathematics-Physics and his MS in Applied Mathematics from UNH. His current interests are high performance computing, parallelization and numerical methods.

Julien Finet joined Kitware on March 17. He arrived from France where he worked as a consultant to GE’s Advantage Workstation Group, specifically on the cardiac analysis project. He is an outstanding C++ programmer and has extensive image analysis experience. Julien has an MS in computer science and engineering from the University of Technology of Compiègne and will be working in Kitware’s office in North Carolina.

Niki Russell joined Kitware’s Clifton Park office in February as technical writer. As Kitware’s technical writer, Niki will aid in the development of product documentation, proposals, Kitware’s website and its company publications. She will also begin to develop Kitware’s marketing efforts in the public sector. She has five years’ experience in corporate communications developing presentations, foundation grants and added-value programming. Niki graduated from Syracuse University with a BS in Magazine Journalism and Marketing in 2001, and her MS in Education in 2005.

MICCAI 2007

Kitware and open source efforts made a strong showing at the 2007 Medical Image Computing and Computer-Assisted Intervention (MICCAI) conference that was held in Brisbane, Australia. Indications of the impact of ITK and VTK were evident throughout many of the talks and posters at the conference.

From Kitware, Julien Jomier, Brad Davis, and Stephen Aylward attended; Luis Ibanez gave an invited talk via video conferencing; Brad Davis made a presentation on atlas formation to the Medical Image Registration Workshop, Andinet Enquobahrie and Stephen Aylward were lead authors on two papers at the Open-Source and Open-Data Workshop; and Stephen Aylward and Luis Ibanez were two of the co-organizers of the 3rd Open Source and Open-Data for MICCAI workshop.

Approximately 25 people attended the Open Source and Open Data Workshop. The papers at the workshop spanned the topics of robotics, cardiac motion analysis, image guided surgery, and vascular image analysis. The papers and presentations from the workshop are available on the Insight Journal by going to: http://insight-journal.org, and searching for Open Source and Open Data Workshop.

BRIGHAM AND WOMEN’S HOSPITAL NEUROIMAGE ANALYSIS CENTER: OPTIMIZING ITK’S REGISTRATION FRAMEWORK

ITK’s registration framework has been updated to take advantage of multi-core processors -- providing faster performance and the ability to handle larger problem sizes. Luis Ibanez, Matt Turek, and Stephen Aylward have delivered an initial implementation of the improved methods in ITK’s Review Directory. The improved methods offer 3 to 100x speed increase for select combinations of ITK’s metrics, optimizers and transforms on quad-core machines. Special emphasis has been placed on improving the speed and memory requirements when the Mattes mutual information metric is combined with b-spline transforms. Included with the improved registration methods are new cross-platform methods for tracking the speed and memory usage of any ITK application.

Ongoing work will produce a compact open-source application for running a suite of registration speed and memory tests on nearly any machine and automatically generating a PDF report of the test results. These reports can be uploaded to the Insight Journal to help ITK users choose appropriate ITK algorithms and hardware for their registration needs.

UNIVERSITY OF NORTH CAROLINA: WEB-BASED MURINE IMAGE ANALYSIS

As part of an NIH STTR with Dr. Martin Styner, Kitware has installed MIDAS in the Department of Psychiatry at the University of North Carolina at Chapel Hill. MIDAS is Kitware’s suite of web-based tools for publication and data management. UNC’s MIDAS installation is the first outside of Kitware that also includes the ability to initiate and monitor complex image analysis pipelines over the web. Those pipelines use UNC’s GRID resources to register and segment the medical images stored on the MIDAS server. Results are uploaded back to the server for online review and distribution to collaborators.

The initial application is the study of the brain morphology of mice populations that have been genetically engineered to exhibit behaviors associated with various human mental disorders. Kitware’s project lead is Julien Jomier. He is being assisted by Patrick Reynolds, Matthieu Philippe, and Stephen Aylward.

METAIO: SHARING CODE BETWEEN ITK AND VTK

One challenge associated with managing multiple, large, open-source efforts is synchronizing and updating the common components in those libraries (e.g. XML, TIFF, and DICOM). Kitware has developed a solution. As a proof of concept, Brad King, Stephen Aylward, Sebastian Barre, and Julien Jomier have created a shared repository for MetaIO. MetaIO is a library for reading and writing images and objects in a user-friendly, tag-based format.

Cron jobs as well as svn and cvs locks replicate the repository to the appropriate subdirectories in ITK and VTK. Changes made to the shared repository are automatically distributed to those toolkits. Work will soon begin to replicate this shared repository technique to other libraries distributed with ITK, VTK, ParaView and other Kitware projects.
The MIDAS Publication Database is a repository for scientific papers, reports and associated multimedia. The Publication Database is a computational infrastructure intended to facilitate the outreach activities of scientists. Researchers often need more flexibility than PubMed currently offers, most importantly, PubMed does not cover all journals used to report on NIH, and other agencies’ funded research. The Publication Database enables companies and institutions to comply with recent law which includes a provision directing the National Institutes of Health (NIH) to provide the public with open, online access to findings from its funded research.

The main features of the Publication Database are:

- Easy installation on popular web servers
- Automatic image gallery
- Easy import from PubMed
- Submission workflow
- World map of downloads
- Download statistics
- Export as Bibtext and PubMed XML
- Advanced Search
- Compliant with the Open Archives Initiative
- Harvested by Google and other major search engines

Using the Publication Database, researchers can easily store and keep track of their bibliographies. Papers can also be uploaded and made available to the community for download. By exporting a catalog of publications as a Bibtext or XML, principal investigators can quickly add a bibliography to a grant report and scientists can quickly integrate related work in their references when writing a paper. After an easy setup, the Publication Database is ready to receive publications submitted by restricted users. User management is provided through a secure login. Submitters can optionally upload associated multimedia such as movies, spreadsheets, screenshots and images. Moreover, the Publication Database provides an easy import from PubMed allowing managers to fill in their database with just a few mouse clicks.

After a complete review during the approval stage, the publication and its media are made publicly available; at the same time, images and screenshots associated with the publication are automatically organized and displayed in the Image Gallery. The ability to measure the impact of publications on the scientific community is important, especially when writing reports. The powerful statistics functionality of the Publication Database provides a simple view of the most downloaded papers, as well as a world map of the download locations.

The Publication Database provides simple and quick searches through the entire collection of publications as well as more advanced search functionalities; users can independently search by title, keywords, authors, abstract, tags and sponsors.

Additionally, the Publication Database is fully compliant with the Open Archives Initiative protocol, which develops and promotes interoperability standards that aim to facilitate the efficient dissemination of content. Harvested by major search engines, such as Google and Yahoo!, the Publication Database is significantly increasing the presence and visibility of institutions on the World Wide Web.
CONNECTOME
Kitware continues to push the edges of visualization science by collaborating with researchers at Harvard’s Institute in Innovative Computing. The Connectome project (http://iic.harvard.edu/projects/connectome.html) led by Dr. Hanspeter Pfister, along with biologists Jeff Lichtman and Clay Reid at Harvard, aims to map, store, analyze, and visualize the actual neural circuitry of the peripheral and central nervous systems in experimental organisms (e.g., mice), based on a very large number of images from high-resolution microscopy. The images are expected to grow to sizes of over a terabyte.

Interacting with data of this size is challenging. Because the team wishes to fly through the data at interactive frame rates in order to segment the neural circuitry, advanced computational techniques from supercomputing are required. Using some of the underlying VTK and ParaView parallel communications infrastructure, Charles Law and Rusty Blue of Kitware have built a client-server application which pre-processes data, as well as generates imagery in parallel on the IIC cluster. The resulting image is then sent to a client running on a desktop computer. Aside from handling incoming data, the client is also responsible for managing user interaction and sending data requests back to the server. Typical performance for remote viewing of the data at Kitware in Clifton Park, NY, with the server located at Harvard, is several images per second.

Kitware and the IIC hope to release the source code for the Connectome project in the near future; at that point we will be seeking partners who wish to use or help develop the system.

EMPLOYMENT OPPORTUNITIES
Kitware is seeking talented software professionals with experience in medical image analysis, image processing, 3D graphics, graphical user interface, visualization, computer vision, and/or software engineering. Candidates must show initiative, flexibility, and the focus necessary to deliver quality software (both open-source and proprietary code). Applicants must demonstrate software development skills and have experience in C++.

Qualified candidates will work with leaders in the field on cutting-edge problems. Kitware offers significant growth opportunities; an annual bonus; six weeks total paid time off; health, dental, long-term disability, and life insurance benefits; and a 401(k) plan with company contributions.

Kitware is an equal opportunity employer.

Send your resume to jobs@kitware.com with “Kitware Employment” as the subject line. Please include a plain text cover letter in the body of the email.