TABLE OF CONTENTS

Editor's Note................................................................. 1
Recent Releases ............................................................. 1
Selections in VTK .......................................................... 2
IO Streaming in ITK .......................................................... 5
Writing a ParaView Reader Plug-in ................................. 7
In Progress ............................................................................. 9
Kitware News ................................................................. 10

RECENT RELEASES

CDASH 1.4
CDash, the open-source, web-based software testing server, had its third major release in April. CDash aggregates, analyzes and displays the results of software testing processes submitted from clients around the world, conveying the state of a software system to continually improve its quality.

This new release fixes more than 60 bugs and adds more than 20 new features including:

- New memory efficient XML parsing using SAX parsing instead of DOM
- Introduction of Model View Controller (MVC) framework
- Nightly testing of CDash itself
- Support for subprojects
- Support for query filters to allow users to only view error builds, builds from multiple days and to create hyperlinks to filter results and bookmark them
- Support for labels on builds, tests, coverage files via CMake/CTest
- Improved navigation
- Improved database indexing
- Improved reporting, including email for coverage files
- Improved column sorting
- New LDAP support

More information about 1.4 is available on the CDash Wiki.

CMAKE 2.6.3
CMake 2.6.3 was released on February 23, 2009. The major changes include the following bug fixes and features:

- Add parenthetical expressions to if
- Improve scoping of policies (see cmake_policy and CMP0011)
- Teach find_package more config file locations
- Many improved Find scripts in Modules
- Added an unset command
- Qt cmake-gui is the primary GUI for windows, deprecate CMakeSetup
- Better reporting when nmake compiler environment is not correct

More information can be found on cmake.org.

IGSTK 4.0
The IGSTK development team released IGSTK 4.0 on February 7, 2009. This release includes:

- Refactored transform classes (affine transform, perspective transform)

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. Jeff Baumes explores the VTK selection architecture used by ParaView and explains how this architecture can be used to manipulate data in any VTK-based application in his article "Selections in VTK". Bradley Lowekamp's article "IO Streaming in ITK" explains how out-of-core processing can be performed on extremely large datasets through simple filter modifications which support streaming in the execution of ITK's pipeline. In "Writing a ParaView Reader Plug-in" Dr. Jeff Baumes provides a tutorial on writing reader plug-ins which allow users to open new file types from ParaView.

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.
• Support for ascension tracker (3D guidance medSafe)
• Reslicing (orthogonal and oblique modes; manual and automatic reslicing)
• Several example applications (navigator, tracker measurement, volume viewer, tracker configuration)
• Miscellaneous bug fixes

ITK 3.12
ITK 3.12 was released on March 3, 2009. The main changes in this release include the addition of classes contributed to the Insight Journal for performing:
• certain specific Mathematical Morphology operations,
• robust automatic thresholding,
• computation of shape features such as tubularness, sheetness and blobiness, and
• a more flexible flood filled iterator.

Further improvements were made for providing consistent management of image directions.

Support for image streaming on the writers was made functional thanks to the help of Brad Lowenkamp. More information on image streaming support can be found in Bradley Lowenkamp’s article on page 5.

Coding style was fixed across the toolkit by taking advantage of the KWStyle automatic style checker tool. From hereafter coding style will be verified at the moment of committing code to the CVS repository.

VTK 5.4
VTK 5.4 was released on March 26, 2009; this release is now available for download from the VTK website. A full list of changes is available through the VTK user’s mailing list archives. The major changes in this release are as follows:
• The Geovis kit is now included with VTK
• Two new views were added: TreeRingView & IcicleView
• The freetype font engine was updated to v2.3.7

More information about VTK 5.4 can be found on vtk.org.

SELECTIONS IN VTK

In the June 2008 issue of the Kitware Source, the article “Selecting Data in ParaView” discussed ParaView’s powerful new selection facility and how it can be used to explore data in new ways. In this article, we are taking things down a level to explore the VTK selection architecture that ParaView uses. We will see how to use selections to manipulate data in any VTK-based application. All code in this article is based on VTK 5.4.

WHAT IS A SELECTION?
In its most general sense, a selection is a data structure that specifies a subset of data. This subset may be highlighted to show a feature in the data, or may be extracted to analyze a portion of the data in more detail. VTK provides a framework for generating, processing, and sharing selections in applications using the vtkSelection class, along with related selection sources, filters and views. vtkSelection is a container class holding one or more vtkSelectionNode objects. Each node contains information indicating what part of the data is selected. A compound selection is interpreted as the union of the individual node selections. We allow selections to consist of multiple nodes so that in one place we can represent selections on multiple parts of the data. For example, a selection on geometry may contain a vtkSelectionNode for both points and cells. Another use case is to collect selections from multiple datasets in the same renderer into one place.

TYPES OF SELECTIONS
Each vtkSelectionNode has a selection type which indicates how the selection is to be interpreted. The selection types are constants defined in vtkSelectionNode.h. Values associated with a selection are stored in an array retrieved by GetSelectionList(). For convenience, we will use the term “element” to refer to the basic building-blocks of datasets to which attributes may be assigned. For vtkDataSet subclasses, the elements are points and cells. The elements of vtkGraph subclasses are vertices and edges. For vtkTable, the elements are the rows of the table.

INDEX SELECTIONS
This is the most basic type of selection. An index selection’s selection list is a vtkIdTypeArray containing the raw zero-based indices of the selected elements in a dataset, using the dataset’s internal ordering. Since these indices may change as a dataset is processed or filtered, an index selection is generally only applicable to a single dataset. You should not share an index selection between datasets with different topologies.

PEDIGREE ID SELECTIONS
A pedigree ID is an identifier assigned to each element in a dataset at its source, and is propagated down the pipeline. You specify pedigree IDs on a dataset in the same way that other special attributes like scalars and vectors are specified, by calling SetPedigreeIds() on a dataset’s attributes. Pedigree ID arrays may be of any type, including vtkStringArray and vtkVariantArray. A pedigree ID selection contains a list of values from a dataset’s pedigree ID array. Both pedigree ID and global ID selections refer to elements by name, instead of by a dataset-specific offset used in index selections, which makes them more robust across different datasets originating from the same source.

GLOBAL ID SELECTIONS
Global ID selections are much like pedigree ID selections, except that they refer to the global ID attribute (set with SetGlobalIds()). Global IDs are used like pedigree IDs, except that they must be numeric, and some filters re-assign global IDs to ensure that IDs are never repeated.

FRUSTUM SELECTIONS
Frustum selections simply store the geometry of a frustum in the selection. All elements within the frustum are considered selected. This is most useful in cases where the user drag-selects a region in a 3D scene. The rectangular region on the screen translates into a frustum in the 3D scene. The selection list for this type of selection must be a vtkDoubleArray with eight four-component tuples. The points, in the form \((x, y, z, 1)\), should be in the following order:
1. near lower left
2. far lower left
3. near upper left
4. far upper left
5. near lower right
6. far lower right
7. near upper right
8. far upper right

The expected order of the points in a view frustum selection.

VALUE SELECTIONS
A value selection is a selection that refers to elements in an arbitrary array. You must set the name of the selection list in a value selection to the name of the array that you want to select on. For example, suppose you have a dataset where the points contain an integer attribute “type” which varies from 0 to 10. To select only points with values 1, 3, and 7, create a vtkIntArray, add the desired values to it, then set the name of the array to “type”. Finally, with the array you’ve created call node->SetSelectionList(arr).

THRESHOLD SELECTIONS
Threshold selections work just like value selections, except you indicate value ranges with each pair of elements in the array. While value selections may be of any type, threshold selections only work on numeric arrays, and the selection list must be a vtkDoubleArray. To select points with type in the range 0-5, create a vtkDoubleArray and add the elements 0 and 5. You may add additional ranges to the threshold selection by adding more pairs of numbers.

LOCATION SELECTIONS
As the name suggests, you provide this type of selection with the 3D locations that you want selected. A location selection must be a vtkDoubleArray with 3 components per tuple. A location selection is often used to select cells that contain a point. For location selections referring to the points of a dataset, there is a way to specify the maximum distance a selected dataset point can be to a point in the selection list:

```
n->GetProperties()->Set(vtkSelectionNode::EPSILON(distance));
```

BLOCK SELECTIONS
VTK has a data object called vtkMultiBlockDataset, which can store a collection of datasets. A block selection allows you to specify which blocks to select. The selection list must be a vtkUnsignedIntArray.

CREATING A SELECTION
The following C++ code creates a simple selection which selects points where the pedigree ID attribute is “John”, “Joe”, or “Sue”.

```
vtkSelection* s = vtkSelection::New();
vtkSelectionNode* n = vtkSelectionNode::New();
n->SetFieldType(vtkSelectionNode::POINTS);
n->SetContentType(vtkSelectionNode::PEDIGREEIDS);
vtkStringArray* a = vtkStringArray::New();
a->InsertNextValue("John");
a->InsertNextValue("Joe");
a->InsertNextValue("Sue");
n->SetSelectionList(a);
s->AddNode(n);
```

There are a few more selection properties that you should be aware of. You can invert a selection by setting the INVERSE property to 1 with the code:

```
n->GetProperties()->Set(vtkSelectionNode::INVERSE(), 1);
```

Also, when performing a point selection on a vtkDataSet the CONTAINING_CELLS property indicates whether to also select cells which use at least one selected point. The test code located in VTK/Rendering/Testing/Cxx/TestExtraction.cxx creates and displays many different selection types and exercises various selection options.

USING THE HARDWARE SELECTOR
VTK also provides the class vtkHardwareSelector to assist you in generating selections from a rectangular region of the screen.

```
vtkHardwareSelector* hs = vtkHardwareSelector::New();
hs->SetRenderer(ren);
hs->SetArea(xmin, ymin, xmax, ymax);
vtkSelection* s = hs->Select();
```

The hardware selector performs special rendering passes in order to determine which datasets in the renderer are selected, and which cells within those datasets are selected. Any cell that is rendered to at least one pixel within the selection area is inserted into the output selection. The output selection contains one vtkSelectionNode for each selected actor. These nodes are cell index selections by default, although the hardware selector can also be configured to select points. You can retrieve the pointer to the associated actor in the scene by accessing the PROP property of the selection node.

EXTRACTING SELECTIONS
Now that we know how to define a selection, we must use it in some way. One of the most common tasks is to extract the selection from a dataset. To do this, use the vtkExtractSelection filter for vtkDataSets or vtkMultiBlockDataSets, or use vtkExtractSelectedGraph for vtkGraphs. Both filters work in similar ways and accept a selection of any type. To extract a selection from a vtkPolyData, we can do the following:
Note that because `vtkExtractSelection` accepts any `vtkDataSet` subclass, the output is the most general dataset, `vtkUnstructuredGrid`. The output type cannot match the input type because a selection, which denotes an arbitrary subset on a structured type like `vtkImageData`, might no longer be structured.

If you desire to simply mark selected elements of a dataset, call `ex->PreserveTopologyOn()` before updating the filter. This will pass the data structurally unchanged to the output, with the same data type. Instead of culling away rejected elements, the filter adds a boolean flag to each element indicating whether it is selected or not.

```cpp
tkPolyData* pd = vtkPolyData::New();
// Populate the poly data here
vtkSelection* s = vtkSelection::New();
// Populate the selection here
vtkExtractSelection* ex =
  vtkExtractSelection::New();
ex->SetInput(0, pd);
ex->SetInput(1, s);
ex->Update();
vtUnstructuredGrid* extracted = ex->GetOutput();
```

A portion of a `vtkImageData` extracted by a frustum selection.

**USING SELECTIONS IN VIEWS**

Selection is an important component of the new view architecture recently added to VTK in version 5.2. The views in VTK combine rendering logic, interaction, visualization parameters and selection into one place. Datasets are displayed in views by adding what are called “representations” to the view. A representation object prepares an input dataset to be displayed in a view. The view automatically generates selections of the desired type when the user performs a selection interaction (i.e. clicking or dragging a selection box). Selections may be shared across views by simply setting a common `vtkSelectionLink` object on multiple representations.

We will finish with the following example which generates a graph view and table view with linked selection. The table view displays the vertex data of the graph. When rows are selected in the table view, the graph view updates to reflect this, and vice versa.

```cpp
QApplication app(argc, argv);

// Create the graph source and table conversion
vtkRandomGraphSource* src =
  vtkRandomGraphSource::New();
vtkDataObjectToTable* o2t =
  vtkDataObjectToTable::New();
o2t->SetInputConnection(src->GetOutputPort());
o2t->SetFieldType(vtkDataObjectToTable::VERTEX_DATA);

// Create Qt table view and add a representation
vtkQtTableView* tv = vtkQtTableView::New();
vtkDataRepresentation* tr;
tr = tv->AddRepresentationFromInputConnection(o2t->GetOutputPort());

delete o2t;

delete tr;

delete src;

delete tv;

// Start application
app.exec();
```

A simple application showing linked selection between a graph layout view and a Qt table view.

**ACKNOWLEDGEMENTS**

This work was supported in part by Sandia National Labs through contract number #664044.

Dr. Jeff Baumes is an R&D Engineer in Kitware’s Clifton Park, NY office. Dr. Baumes is the main developer for Titan, whose goal is to bring information visualization and informatics capabilities to VTK.
**OVERVIEW**

Out of core image processing is necessary for dataset sizes larger than a computer's main memory. ITK's pipeline architecture typically buffers the entire image for each filter. This behavior produces memory requirements that are multiples of the dataset size, which inhibits the processing of large datasets. Fortunately ITK was designed to accommodate the sequential processing of sub-regions of the data object, a process called streaming. Previously, to easily stream data objects one had to use the StreamingImageFilter, which sequentially requests sub-regions, causing the input to stream, and then reassembles the regions into a buffered output image. However, if the dataset exceeds the size of system memory this approach will not work. For large datasets the entire image must never be in memory at once; therefore pipelines must stream from the reader to the writer.

Section 13.3 of the *ITK Software Guide* provides a detailed explanation of the internals of how streaming works in ITK's pipeline execution [1]. For streaming to work correctly, each filter in a pipeline must be able to stream. The PropagateRequestedRegion() phase of the pipeline update is crucial for streaming, as a filter negotiates what region the filter should process on its input(s) and output(s). Filters incapable of streaming will set their input requested region to the largest possible. This action will make all upstream (closer to the first) filters process the largest possible region, but they will not actually stream. That is to say that streaming must begin with the writer.

**IMPLEMENTATION**

One of the challenges for implementing streaming IO was the need for the readers and writers to negotiate the requested regions with ImageIO classes so the regions match the ImageIO object's capabilities. The readers and writers use an object factory to create an ImageIO object to read the specified file type at runtime. New methods were added to ImageIOBase to facilitate the negotiations of requested regions. The default implementation does not provide support for streaming. Currently few file types support streaming, namely Meta Image has robust support for reading and writing, while Nifti currently has support for streaming and writing. In addition to streaming an entire image piece by piece, another type of streaming has been defined for writing called pasting. Pasting is the writing of a sub-region to a file, which contains the entire image. If the file does not exist a new one will be created, otherwise the meta-data and image type must match what is in the file. If neither of these conditions is met or if the ImageIO is not capable of pasting, an exception will be thrown. The pasting operation is designed to be scalable to large files, where rewriting the entire file will be too costly.

**VISIBLE HUMAN EXAMPLES**

The Insight Toolkit was originally motivated by a need for software tools to segment and register the National Library of Medicine's Visible Human Project datasets. The data is freely available through NLM's website [3]. The original Visible Male cryosectional images are non-interlaced 24-bit RGB pixels with a resolution of 2048x1216 pixels by 1871 slices, and a physical spacing of approximately 0.33 mm in slice and 1.0 mm between slices. These dimensions results in about 13GB of data, which is an appropriate size to demonstrate streaming. The following two examples of streaming show the three IO classes capable of streaming along with the two types of streaming supported by the writer.

**STREAMING**

A coronal slice is a classic view of the Visible Male. The following is an example that reads the entire raw dataset and generates that image:

```cpp
typedef itk::NumericSeriesFileNames
    NameGeneratorType;
NameGeneratorType::Pointer nameGenerator = New();
nameGenerator->SetSeriesFormat( “a_vm$04d.raw” );
namerGenerator->SetStartIndex( 1001 );
namerGenerator->SetEndIndex( 2878 );
namerGenerator->SetIncremenIndex( 1 );

// create a ImageIO for the red channel
typedef itk::RawImageIO<PixelType, 2> ImageIOType;
ImageIOType::Pointer rimageio = New();
rimageio->SetDimensions( 0, 2048 );
rimageio->SetDimensions( 1, 1216 );
rimageio->SetSpacing( 0, 0.33 );
rimageio->SetSpacing( 1, 0.33 );
rimageio->SetHeaderSize(2048*1216*0);
```
This example creates a RawImageIO and ImageSeriesReader for each color channel in the data. Notice that there are no special methods needed to enable streaming; streaming is in response to requests from the pipeline. In the ComposeRGBImageFilter, the channels are composited into a single color image. Then the information is updated to initialize the coronal slice region to be extracted. The final filter, ImageFileWriter, writes out the file as a Meta Image type, which fully supports IO streaming.

The performance of this code is excellent, in both speed and memory usage. On a MacBook Pro laptop execution was completed within 3 minutes, while on an Intel dual quad-core Mac Pro workstation it finished in less than 30 seconds. For memory, only about 50 megabytes were used.

The most interesting aspect of this example is not the filters used, but how ITK's pipeline manages its execution. The final output image is 2048 by 1878 pixels. The ImageFileWriter breaks this 2D image into 200 separate regions, each about 2048 by 10 pixels. Each region is then processed through the pipeline. The writer makes 200 calls to its ImageIO object to write the individual regions. The extractor converts this 2D region into a 3D region of 2048 by 1 by 10 pixels, which is propagated to the ImageSeriesReader.
This pasting example only writes the small halfIO region to
the file, the remainder is not touched. The manner in which
the pipeline executed is very similar to the previous stream-
ing example. The main difference is that the writer only
breaks up the IORegion for streaming, not the entire image.
The other difference is that the reader fully supports stream-
ing and only reads the required region from the file.

The resulting image after the pasting the gradient magnitude onto
a coronal cross section of the Visible Male. This image was also run
to the ShrinksImageFilter to make the pixels square.

FUTURE WORK AND CONCLUSIONS
Currently only the Meta Image fully supports streaming,
however, with the current framework, other formats can also
be upgraded to support streaming. The ImageSeriesWriter
could also implement streaming of sequences for files when
a single output file is not desired. Many filters do not cur-
rently support streaming and as such, filter modification
may be required to support streaming.

Streaming IO was a bottleneck to support processing of very
large data. Our examples have shown out-of-core processing
can now be performed on datasets such as the Visible
Human Project and large microscopy scans.

ACKNOWLEDGEMENTS
Much of this work has been accomplished through the col-
laborative weeks provided by NAMIC. The author would also
like to thank Luis Ibáñez and Stephen Aylward for their help
with ITK and the MetaIO library along with David T. Chen for
his valuable feedback on this article.

FOOTNOTES:
[2] Thanks to Hans Johnson, Dough Alan and Luis Ibáñez
for developing streaming in the ImageFileReader during
NAMIC Project week of June 2007.
[4] The file needed the ElementType modified to MET_uchar
ARAY. The author is currently investigating this further.

ParaView is a general-purpose, open-source application
for scientific visualization. A common challenge for new
ParaView users is figuring out how to load data. The first
step is simply to go to File Open, and look any related input
types. This is a long list, and chances are you will see some-
thing that matches your data format. But what if your type
is not there? You could write custom utilities that convert
your data format into one supported by ParaView. However,
a better option in many cases is to write a reader which will
read your data into a data structure that ParaView can use.
A large part of ParaView’s immense flexibility comes from its
plug-in architecture. This article will show you how to write
a commonly used type of plug-in called a reader plug-in. A
reader plug-in allows ParaView to open a new file type from
the File Open dialog.

STEP 1: WRITE THE READER
The hardest part of this process is writing the reader itself.
We will use the simple example of reading a 2D image from
a text file with this simple delimited format. This will allow
someone to export simple matrices of data from Excel and
visualize them as heat maps or height fields in ParaView.
We assume the files will consist of strictly numeric values
separated by commas or some other delimiter.

While this is not a full tutorial on writing a VTK reader, we
will provide most of the details here for completeness (see
the VTK User’s Guide for more information on writing VTK
readers). The header file vtkCSVImageReader.h contains
the basic interface for the source. Since we are creating an
image as the output, our source will subclass vtkImageAl-
gorithm. To help with the implementation of this class, we
will use a vtkDelimitedTextReader instance which will read
delimited text into a vtkTable. Most of the methods here are
boilerplate, with the exception of the methods to set and
get the file name and field delimiter characters, which are
delegated to the vtkDelimitedTextReader instance variable.

class vtkCSVImageReader : public vtkImageAlgorithm
{
public:
  static vtkCSVImageReader* New();
  vtkTypeRevisionMacro(
    vtkCSVImageReader,vtkImageAlgorithm);
  void PrintSelf(ostream& os, vtkIndent indent);
  virtual void SetFileName(const char* fname);
  virtual const char* GetFileName();
  virtual void SetFieldDelimiterCharacters(
    const char* delim);
  virtual const char* GetFieldDelimiterCharacters();
}
```cpp
protected:
  vtkCSVImageReader();
~vtkCSVImageReader();

int RequestInformation(
  vtkInformation*,
  vtkInformationVector**,
  vtkInformationVector*);

int RequestData(
  vtkInformation*,
  vtkInformationVector**,
  vtkInformationVector*);

vtkDelimitedTextReader* Reader;

private:
  vtkCSVImageReader(const vtkCSVImageReader&);
  void operator=(const vtkCSVImageReader&);
};
```

The implementation of RequestInformation tells the VTK pipeline information about the data before creating the output. In the case of image algorithms, this means populating the extent, spacing and origin flags. In order to get this information, we need to update our internal filter to retrieve the number of rows and columns in the table; these rows and columns correspond to the x and y extent of the image. We also tell the pipeline that we want the scalar data on the image to be stored in a “float” data array.

```cpp
vtkDataArray* scalars =
  image->GetPointData() -> GetScalars();
scalars -> SetName(“Data”);
for (vtkIdType r = 0; r < rows; ++r)
{
  for (vtkIdType c = 0; c < columns; ++c)
  {
    vtkVariant val = output->GetValue(r, c);
    float f = val.ToDouble();
    scalars -> SetTuple1(c*rows + r, f);
  }
}
return 1;
```

### STEP 2: WRITE THE PLUG-IN XML

Information about plug-ins is given to ParaView through a few simple XML files. In general, plug-ins will have two related XML files: the GUI XML and the Server Manager XML. The GUI XML gives ParaView the higher-level information about where new elements will be placed in the user interface. In our example, CSVImageGUI.xml is a simple file that tells ParaView to create a new reader called “CSVImageReader” associated with the file extension “.csvimg”. This will create a new entry in the File Open dialog for CSV image files and will automatically associate files ending in “.csvimg” with our new reader.

```xml
<ParaViewReaders>
  <Reader
    name=“CSVImageReader”
    extensions=“csvimg”
    file_description=“CSV image”>
  </Reader>
</ParaViewReaders>
```

The Server Manager XML file CSVImage.xml contains information defining the client-server proxy named “CSVImageReader”, which is referenced in the GUI XML. A proxy is simply an object that represents an underlying VTK object, and has mechanisms for accessing properties and calling methods across a network (when ParaView is run in client/server mode), along with serialization, undo/redo support, and more. The “class” property of the proxy tells ParaView the name of the class to instantiate when creating this reader. ParaView receives information about how to bind property names with method calls on the class and what default values to use from the StringVectorProperty elements. Since both properties on vtkCSVImageReader are just scalar values, we set the “number_of_elements” attribute to 1. This simple piece of XML code gives ParaView enough information to interact with your reader and it will automatically create a GUI panel for you to specify parameters.

```xml
<ServerManagerConfiguration>
  <ProxyGroup name=“sources”>
    <SourceProxy
      name=“CSVImageReader”
      class=“vtkCSVImageReader”>
      <StringVectorProperty
        name=“FileName”
        command=“SetFileName”
        number_of_elements=“1”>
        <FileListDomain name=“files”/>
      </StringVectorProperty>
      <StringVectorProperty
        name=“FieldDelimiterCharacters”
        command=“SetFieldDelimiterCharacters”
        number_of_elements=“1”
        default_values=“,”/>
    </SourceProxy>
  </ProxyGroup>
</ServerManagerConfiguration>
```
STEP 3: WRITE THE CMAKELISTS FILE

The CMakeLists.txt file ties all these things together in order to build the ParaView plug-in. We must locate ParaView with FIND_PACKAGE and then import the CMake configuration parameters from ParaView by including PARAVIEW_USE_FILE.

ADD_PARAVIEW_PLUG-IN is a general-purpose macro that allows you to specify all the information needed to build the plug-in. The first two arguments are the name and version of the plug-in. Next are a series of argument lists separated by special identifiers. SERVER_MANAGER_XML and GUI_RESOURCE_FILES are the places for our client-server and GUI XML files, respectively. SERVER_MANAGER_SOURCES tells ParaView the source files to wrap into the Server Manager. All classes used in Server Manager XML must be listed here.

CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
FIND_PACKAGE(ParaView REQUIRED)
INCLUDE(${PARAVIEW_USE_FILE})
ADD_PARAVIEW_PLUG-IN(CSVImage "1.0"
SERVER_MANAGER_XML CSVImage.xml
SERVER_MANAGER_SOURCES vtkCSVImageReader.cxx
GUI_RESOURCE_FILES CSVImageGUI.xml)

STEP 4: BUILD AND TEST

When you configure your build with CMake, set ParaView_DIR to the path where ParaView was built or to the binaries of a ParaView install. When the plug-in is built, run ParaView and load the plug-in by selecting “Manage Plug-ins/Extensions...” in the Tools menu. This will bring up the Plug-in Manager.

Click the Load button and browse to your plug-in shared library (CSVImage.dll on Windows). After the plug-in is loaded, you should see the new type in the drop-down menu in the File Open dialog. Files ending with .csvimg will also be part of the default filter in the dialog. Select a .csvimg file and click OK. This will bring up the options for the reader in the Properties panel, which is automatically populated with our property for setting the field delimiter characters.

After hitting Apply, the data will be loaded into ParaView. The figure below demonstrates visualizing data exported from an Excel spreadsheet using the CSVImage plug-in. The spreadsheet was simply a block of cells containing the equation “SIN(ROW()/10)*COS(COLUMN()/10)”. In the image below, we show the result after processing the image with Extract Surface, Warp and Generate Surface Normals.

ACKNOWLEDGEMENTS

This work has been partially funded by Sandia National Laboratories through contract number DE-AC04-94AL85000.

Dr. Jeff Baumes is an R&D Engineer in Kitware’s Clifton Park, NY office. Dr. Baumes is the main developer for Titan, whose goal is to bring information visualization and informatics capabilities to VTK.

PARAVIEW 3.6

Kitware and Sandia National Laboratories are getting ready for the next major release of ParaView, version 3.6. The binaries and sources will be available for download upon release from the ParaView website. This release includes several new features along with plenty of bug fixes addressing a multitude of usability and stability issues, including those affecting parallel volume rendering.

Based on user feedback, ParaView’s Python API has undergone a major overhaul. The new simplified scripting interface makes it easier to write procedural scripts mimicking the steps users would follow when using the GUI to perform tasks such as creating sources, applying filters, etc. We are also working on saving trace as Python scripts, making it easier for beginners to get familiar with the ParaView Python API. Details on the new scripting API can be found on the ParaView Wiki.

We have been experimenting with adding support for additional file formats such as CGNS, Silo, and Tecplot using a
third party application. Since this is an experimental feature, only the Linux and Windows binaries distributed from our website support these new file formats.

ParaView now natively supports tabular data-structures thus improving support for CSV files including importing CSV files as point-sets or structured grids. This version also includes several filters for statistical analysis of the data.

We’ve completely redesigned the charting/plotting components with several performance fixes and usability improvements. Now data values can be plotted along curves obtained by slicing surfaces with planes, etc. Additionally, arrays can be plotted from arbitrary datasets directly, without having to employ the plot over line or any such filter.

In an effort to better support animations involving the camera, we have added support for specifying camera movements along splines or orbiting objects in space.

Apart from these enhancements, ParaView now ships with a brand new application developed on top of the ParaView Application Framework named OverView. OverView is a generalization of the ParaView scientific visualization application and was designed to support the ingestion, processing and display of informatics data. The ParaView client-server architecture provides a mature framework for performing scalable analysis on distributed memory platforms and OverView uses these capabilities to analyze informatics problems that are too large for individual workstations.

**OVERVIEW**

Kitware and Sandia National Labs will be releasing an alpha version of a new toolkit called OverView alongside of the ParaView 3.6 release later this month. The OverView application was designed to support informatics analysis and visualization methods which can be applied to data from sources such as relational databases, spreadsheets and XML. The application supports features such as automatic graph layout, hierarchical organization of data, graph analysis algorithms and charting.

OverView also integrates 2D and 3D geospatial views for plotting relational data on the globe. Kitware, in conjunction with Sandia National Labs, will continue to explore and develop informatics tools which combine information visualization, statistical analysis, and linked views to facilitate interactive data exploration. Please visit sandia.gov/OverView for more information on this application and kitware.com for more information on Kitware's Informatics solutions.

**COMPUTER VISION WINS 6 SBIR PROPOSALS**

In the fall, the Computer Vision Group was awarded 5 Phase I SBIRs and 1 Phase II SBIR, all from DARPA’s Information Processing Techniques Office. The awards were primarily in the area of wide area video analysis and robotic vision.

Detecting and Tracking Multiple Moving Objects from a Moving Platform involves detecting and tracking moving people, vehicles and other objects near a moving robot, from cameras mounted on the robot. The University of Maryland is a subcontractor on this award. On Activity Models for Robots we plan to develop algorithms to recognize the group activity taking place around a robot, based on the observed tracks, and to determine how the robot should participate in the activity. The University of Maryland, the University of California, Berkeley and Georgia Tech will be serving subcontractors on this award. These two efforts will proceed in parallel; however, the tracking effort will provide input to the recognition effort. Both Phase I’s will conclude in late summer, by which time we hope to have preliminary results for the Phase II proposals.

Dismount Tracking in Urban Scenes focuses on detection and tracking in low-resolution wide area video. Rensselaer Polytechnic Institute is serving as the subcontractor on this award. High Resolution 3D Reconstruction from Wide-Area Video focuses on 3D scene reconstruction at a higher spatial resolution than the original video. Brown University is serving as the subcontractor on this award. In addition, Kitware was awarded Phase II funding for Wide-Area Video Storage Techniques. During Phase II, Kitware will be developing video compression technology for an imaging sensor that produces 1.8 Gigapixels at 10 Hz. Brown University and BAE Systems are subcontractors on this Phase II effort.

Functional Interpretation of Activities and Objects is closely related to another award Kitware received last fall, Building Labels in Urban Environments. Through this new Phase I Kitware will continue to develop technology to classify objects based on observed activity. While BLUE is focused on fixed structures such as buildings, the new award has the goal of classifying moving objects based on their behavior. Our partner on this award is Georgia Tech.

Kitware’s Computer Vision group has extensive expertise developing robust solutions for real-world video and image analysis problems. If you are interested in finding out more about Kitware’s state of the art computer vision solutions please contact us at kitware@kitware.com.

**KITWARE EXCELS ON NIH PROPOSAL**

Neurosurgical navigation systems have reduced the risk of complications from surgery and have allowed surgeons to remove tumors that were once considered inoperable. However, many techniques used by neurosurgical navigation systems to align pre- and intra-operative images are inaccurate when tissue deformations occur. Deformations commonly arise from tumor resection, gravitational effects on the organ and the use of hyperosmotic drugs. Deformable intra-operative image registration remains a significant challenge for neurosurgical guidance.
In an NIH R01 grant proposal, “Image Registration for Ultrasound-Based Neurosurgical Navigation”, submitted alongside Dr. William Wells from Brigham and Women’s Hospital, Kitware proposed two new methods for registering preoperative MRI with intra-operative 3D ultrasound data during craniotomies for brain tumor resection.

The proposed methods will be delivered as part of an extensible “NeuralNav” toolkit that provides a common API for fetching tracker data and intra-operative images from commercial (VectorVision by BrainLAB) and research (IGSTK by Georgetown University) surgical guidance systems. Development and validation of methods and efforts will be conducted in collaboration with top neurosurgeons, the developers of IGSTK and the developers of BrainLAB.

The proposal received a score of 119 which places it in the top 1.4 percentile of NIH proposals. It is anticipated that it will be funded by the NCI at $2.2M for five years. The Principle Investigator position on this proposal is shared by Dr. William Wells of Harvard and Dr. Stephen Aylward of Kitware. Collaborators include the Department of Neurosurgery at Duke University and BrainLab, a leading surgical navigation systems manufacturer.

**CDASH NOW SCALABLE FOR EXTREMELY LARGE SOFTWARE EFFORTS**

Kitware has been working with the Trilinos team (http://trilinos.sandia.gov) at Sandia National Labs to help them port to CMake/CTest/CDash. Trilinos is an extremely large software effort whose goal is to develop algorithms and enabling technologies for the solution of large-scale, complex, multi-physics engineering and scientific problems.

The project is composed of more than 40 active subprojects which need to be seen as a single large entity and as individual projects by themselves. Prior to CDash 1.4, global and individual project views of this scale and size weren’t possible. With the changes made to CDash and CTest, Trilinos developers are now able to receive dashboard emails for specific subprojects on which they are working. Additionally, CDash queries can now be saved and used as bookmarks.
IGSTK USER GROUP MEETING
The first ever IGSTK User Group Meeting took place at the SPIE Medical Imaging conference in Orlando, Florida on February 7, 2009. Due to space constraints, the meeting was invitation only. The morning session was dedicated to annual review. During this session, IGSTK core developers presented their work that is incorporated in the new release: IGSTK 4.0. This was followed by an application demo by Jeff Stanley from NDI. In the afternoon session, participants presented applications developed using IGSTK. The applications included an image-guided surgical drill, intraoperative cone-Beam CT for guidance of head and neck surgery, and OpenIGTLink, which integrates Slicer and IGSTK to enable multimodal neuronavigation in neurosurgery and ultrasound-based intervention.

Participants were from:
- Kitware, Inc., Clifton Park, NY
- Imaging Science and Information Systems (ISIS) Center at Georgetown University, Washington DC.
- Harvard University Medical School, Surgical Planning Lab, Boston, MA
- French National Institute for Health and Medical Research (INSERM), University of Rennes, Rennes, France
- Northern Digital Inc (NDI), Ontario, Canada
- SINTEF Health Research Department (Trondheim, Norway)

For abstracts and presentation slides, visit the IGSTK Wiki and search for “IGSTK Meeting”. The group is planning on organizing a European User’s meeting to be held at Computer Assisted Radiology and Surgery, 23rd International Congress and Exhibition in Berlin, Germany, June 23-27, 2009.

KITWARE GROWS, EXPANDS OFFICE SPACE
Kitware’s office in North Carolina is moving! The move takes the office from its current two disjoint locations, to a single larger location with a capacity for 21 employees. The new address for the North Carolina office, starting May 1st, will be, 101 East Weaver Street, Suite G4, Carrboro, NC 27510.

Kitware’s new North Carolina office will occupy nearly 4000 sq. ft. of modern office space in downtown Carrboro. The new office is located close the current offices, is within walking distance to the University of North Carolina at Chapel Hill and is in close proximity to Duke University and Research Triangle Park. As part of the move the office will be upgrading its phone systems, acquiring an expanded conference room and adding seating for eight new employees.

The North Carolina office has grown from two employees to twelve in just over two years. The seamless integration of Kitware’s New York and North Carolina offices has made this rapid growth possible. New leaders within the group, such as Julien Jomier and Brad Davis, ensure that the southern office will continue to grow; our only concern with the new space is that it might not hold them for long.

Kitware’s New York office will also be expanding in the May time frame. The New York office is taking over approximately 7000 additional square feet of space within their present building. Currently the New York office houses 48 employees; the new space will provide room for approximately 90 employees while maintaining the Company’s existing location. This is great news as Kitware is forecasted to grow by 25 percent this year.

EMPLOYMENT OPPORTUNITIES
For the right software developers, we are always hiring. Qualified applicants will have the opportunity to work with leaders in the fields of computer vision, medical imaging, visualization, 3D data publishing and technical software development. By joining our team you will participate in a dynamic work environment and collaborate with esteemed researchers from around the world.

Kitware team members enjoy our small company environment, flexibility in work assignments and high levels of independence and responsibility. Kitware offers a comprehensive benefits package which includes flexible working hours, six weeks paid time off, a personal computer hardware budget, 401(k), health insurance, life insurance, short- and long-term disability, visa processing, a generous compensation plan, profit sharing, 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. These include:

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

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