Kitware SOURCE

SOFTWARE DEVELOPER'S QUARTERLY

Issue 6 • July 2008

TABLE OF CONTENTS

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. François Bertel discusses a new volume rendering technique for regular, rectilinear grids which utilizes the power of GPU in order to accelerate ray casting. Bill Hoffman details the new features of CMake 2.6, the first major release of CMake since May 2006. Some of our most influential Insight Toolkit (ITK) users and developers discuss the issue of backward compatibility to express the complexity of the issue and the variety of diverse opinions about this topic in the community. And Utkarsh Ayachit provides a use-case driven approach to demonstrate how selection can be described and used to support quantitative analysis in ParaView 3.

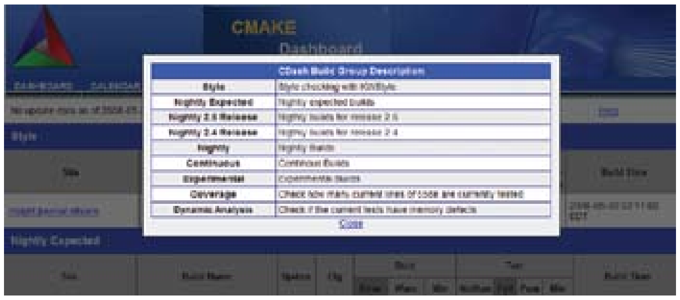
The Kitware Source is just one of a suite of products and services that Kitware offers to assist developers in getting the most out of its open-source products. Each project's website contains links to free resources including mailing lists, documentation, FAQs and Wikis. In addition, Kitware supports 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

CDASH 1.0

CDash the open-source, web-based software testing server has had its first official release. CDash aggregates, analyzes and displays the results of software testing processes submitted from clients located around the world, conveying the state of a software system to continually improve its quality. We have transitioned almost all Kitware software projects to CDash at this point; the database for the public Kitware Open Source projects is about 20 gigabytes and at this point is very reliable and stable. The 1.0 release adds the following features:

- CDash automatically detects when the timing for a test has slowed down significantly
- The number of nightly changes is now displayed on the main project page
- Ability to mark compile warnings, compile errors and test failures with "fix in progress" or "fixed"
- CTest 2.6 timestamp information is used when available
- Support was added for different cvs/svn web viewers including Fisheye
- Ability to remove builds that are not valid
- Upload a cvs/svn user name and email address list to CDash allowing projects to quickly setup all developers
- Output from tests and coverage tests are compressed in the database by only storing unique information
- Sort by any column on the viewTest and testSummary pages
- Bbuild group links only appear when the mouse is on a build group
- Clicking on a header for a build group displays a help dialog



CDash Build Group Description	
Style	Style checking with KWStyle
Nightly Expected	Nightly expected builds
Nightly 2.6 Release	Nightly builds for release 2.6
Nightly 2.4 Release	Nightly builds for release 2.4
Nightly	Nightly Builds
Continuous	Continuous Builds
Experimental	Experimental Builds
Coverage	Check how many current lines of code are currently tested
Dynamic Analysis	Check if the current tests have memory defects

- The project logo is now a hyperlink to the project home page
- Separate lists for failing tests and passing tests.
- Support for external authentication
- Slow queries have been optimized with better response times
- Many other small bugs have been fixed

CMAKE 2.6

CMake 2.6 is the first major release of CMake since 2.4 was released in May 2006. A brief list of the significant new features and bug fixes is provided below. In addition, an article on the CMake 2.6 release is included in this edition of the *Source*.

- Introduction of the cmake_policy command
- New cross platform Qt-based GUI
- Improved Fortran Support
- Mac OSX Framework creation support
- Project generators for Eclipse and CodeBlocks
- Bullseye coverage support
- Uses full paths for linking, and no longer separates into –L and –l
- CPack supports .deb and rpm creation
- Cross compilation support
- Enhanced find_package that can find project installed FooConfig.cmake files
- New CMAKE_PREFIX_PATH environment variable to specify user search directories for find_xxx
- Automatic project reload of project in Visual Studio 7 and greater if cmake is re-run because a CMakeLists.txt file is out of date.
- Improved online and built in documentation including the documentation of CMake variables.
- Ability to change rpath during install instead of relinking
- Ability to import/export targets from a project
- Added functions with local scope variables
- Added return, break, and PARENT_SCOPE to set

ITK 3.6

ITK 3.6 was released on April 15, 2008. The main changes in this release relate to improvements of the image registration framework. In particular, a set of multi-threaded image metric implementations have been added to the Code/Review directory. Stephen Aylward led the team that developed these new classes. Team members included Brad Davis, Sebastién Barré and Matt Turek. Very valuable feedback was provided by Jim Miller (GE), Serdar Balci (MIT) and Golina Polland (MIT).

The registration framework was also revised in order to:

- Provide correct management of Oriented Images
- Reduce memory requirements of BSpline deformable registration
- Reduce memory requirements of Mattes Mutual Information image metric
- Add nightly testing for 3D image registration cases

There were a limited number of classes added to the Code/Review directory.

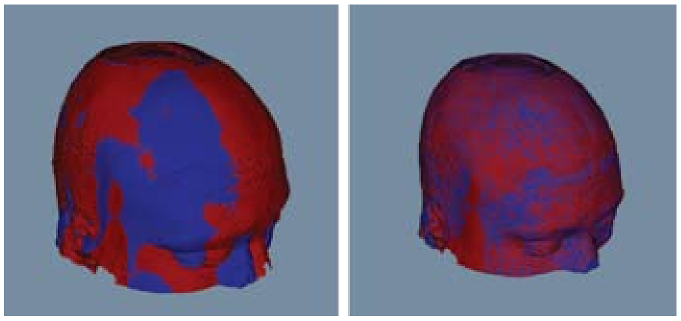
In particular:

- Image Projection Filters
- Helper classes for profiling memory consumption
- Multi-threaded image registration metrics (described above)

The image projection filters were kindly contributed by Gaetan Lehmann in a reproducible paper submitted to the Insight Journal: http://insight.journal.org/midas and search for "image projections along an axis."

For a full list of classes added and changes made in this release, visit itk.org/Wiki and search for "ITK Release 3.6".

The release of ITK 3.8 is currently scheduled for the end of July 2008: http://www.itk.org/Wiki/ITK_Release_Schedule#Release_3.8_Schedule.



Results of an image registration performed with ITK on MRI images, and visualized with VTK surface rendering.

VTK 5.2

VTK 5.2 is the first major release of VTK since 5.0 was released in January 2006. This release includes:

- The new Infovis kit for processing and visualizing "information"(non-geometric) data
- The new Views kit to combine groups of filters, visualization techniques, interaction, and selection mechanisms into a render window to view data
- A new Widgets architecture and more than a dozen new 3D widgets
- Improved time support
- Improved multi-block / composite data support
- Improved Java wrapping
- Improved Mac OS X support
- New Utilities: freerange, verdict, libxml2, metaio, sqlite
- Updated Utilities: freetype, zlib
- More than 300 new C++ classes since VTK 5.0
- More than 100 new C++ tests since VTK 5.0

*A 128*128*128 dataset of a mummy head visualized using the GPU Ray Cast Mapper.*

GPU RAY CASTING IN VTKEDGE

Volume rendering is a visualization technique for representing volumetric data in a 2D image. It is a powerful tool for visualizing biomedical data or the results of volumetric simulations (e.g., weather analysis, astrophysical simulation). This article covers a new volume rendering technique, for regular, rectilinear grids (vtkImageData), that utilizes the power of GPU in order to accelerate ray casting. This new volume mapper is called vtkKWEGPURayCastMapper and is available as part of VTKEdge. Please read the Kitware news article on VTKEdge in this edition of the *Source* for more information on this new open-source toolkit.

OVERVIEW

VTK supports several volume rendering techniques for both regular, rectilinear grids (vtkImageData), and tetrahedral meshes (represented by vtkUnstructuredGrid). Some of these volume mappers primarily utilize the CPU (relying on the GPU only for the final display of the resulting image), while other mappers make use of the resources available on the GPU such as 2D and 3D texture memory and texture mapping functionality. The new vtkKWEGPURayCastMapper in VTKEdge uses the latest advancements available on recent GPUs including fragment programs with conditional and loop operations, multi-texturing and frame buffer objects in order to deliver significantly improved performance over the CPU-based ray casting, while still maintaining high rendering quality.

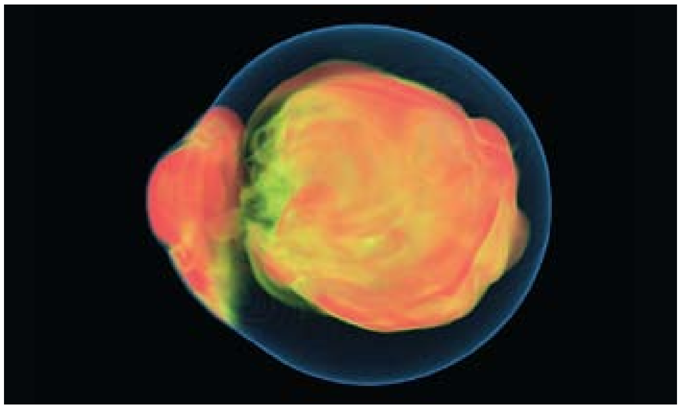
The basic ray casting concept is quite simple. For each pixel in the final image a ray is traced from the camera through the pixel and into the scene. As the ray passes through a volume in the scene the scalar data is sampled along the ray and those samples are processed and combined to form a final RGBA result for the pixel. The implementation of the ray casting algorithm within an object-oriented visualization system, such as VTK, that utilizes object-order rendering (polygonal projection) for other data objects in the scene is a bit more complex, requiring the ray casting process to consider the current state of the frame buffer in order to intermix the volume data with other geometric data in the scene.

In our GPU implementation of ray casting, the volume data is stored on the GPU in 3D texture memory. Ray casting is initiated by rendering a polygonal representation of the outer surface of the volume. A fragment program is executed at each pixel to traverse the data and determine a resulting value that is combined with the existing pixel value in the frame buffer computed during the opaque geometry phase of rendering.

BLENDING OPERATIONS

The vtkKWEGPUVolumeRayCastMapper supports three basic blending operations for the values encountered along the ray. These are:

1. Maximum Intensity Projection (MIP): With a MIP blending operation, the ray caster determines the maximum scalar value encountered along the ray, and then passes this value through the user-defined transfer functions to produce a color and opacity. This technique is often used to view data where the highest scalar values represent the most important features. Typically the color transfer function is set to constant white, while the opacity transfer function is allowed to linearly vary from 0.0 at the minimum scalar value to 1.0 at the maximum scalar value.
2. Minimum Intensity Projection (MinIP): The MinIP projection is similar to the MIP projection, but instead determines the minimum value along the ray. This blending operation is useful when the minimum data values represent the most important features. A reverse opacity ramp (going from 1.0 at the minimum scalar value to 0.0 at the maximum scalar value) would be used for MinIP visualizations.
3. Composite Method: For a composite projection, the ray caster will consider each sample encountered along the ray and blend these values using a standard "over" operation. If the alpha component approaches 1.0 (and therefore further data samples would not contribute to the image) the computation along the ray can be terminated. If shading is enabled in the volume property, then a gradient value will be computed using a central differences method on the GPU at each sample and used as a normal vector for lighting calculations.



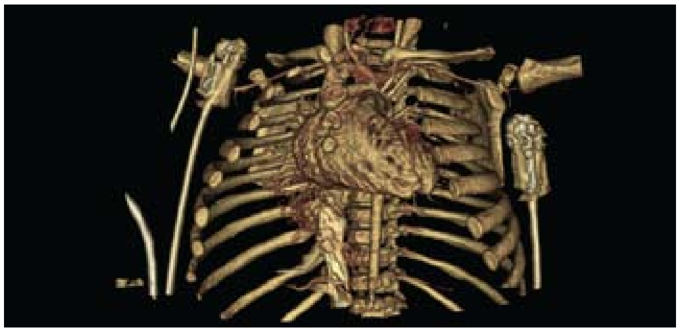
Composite rendering without shading.
Data courtesy of the Terascale Supernova Initiative.

CURRENT FEATURES

The vtkKWEGPUVolumeRayCastMapper currently supports a diverse set of standard volume visualization features. These include:

- Input Scalar Type: The mapper can accept input data of any scalar type from unsigned char through double. Note that internal calculations are performed at single precision floating point accuracy.
- Speed / Accuracy Trade-Off: The computed image resolution is independent of the screen resolution and will be automatically adjusted to achieve the allocated rendering time for the volume. The computed image is rendered to the frame buffer using bilinear interpolation. Typically, the number of rays cast can be reduced by a factor of 4 to 9 during interaction (resulting in frame rates of nearly 4 to 9 times faster) with only a small loss in image quality.
- Camera Projection: The mapper works for both parallel and perspective camera projections.
- Intermixing: Opaque geometry will be correctly intermixed with the volume rendering produced by this mapper. Multiple volumes that do not overlap can be correctly rendered together provided that the vtkFrustumCoverageCuller (or another vtkCuller subclass) is used to order the volume in a back to front manner.
- Point Data / Cell Data: This mapper supports rendering of both point data and cell data.

- Contrast and Brightness: A window / level operation can be applied to the final image before display in order to adjust the contrast and brightness.
- Clipping / Cropping: The mapper supports data axis-aligned cropping as well as arbitrary clipping planes.
- Large Data: If the volume data is too large to fit in texture memory, it will be streamed from the CPU to the GPU in slabs to perform rendering.


Composite rendering with shading of a DICOM medical image data set.

REQUIREMENTS

The vtkKWEGPUVolumeRayCastMapper uses the OpenGL shading Language (GLSL) as well as several advanced features of OpenGL that are only available on recent mid-to-high end graphics cards. The specific features required to use this mapper are:

- GL_ARB_shading_language_100 or OpenGL>=2.0
- GL_ARB_shader_objects or OpenGL>=2.0
- GL_ARB_fragment_shader or OpenGL>=2.0
- GL_ARB_GL_ARB_texture_non_power_of_two or OpenGL>=2.0
- GL_ARB_draw_buffers or OpenGL>=2.0
- GL_EXT_framebuffer_object
- GL_ARB_depth_texture or OpenGL>=1.4

The optional (used if present) OpenGL features are:

- GL_ARB_texture_float
- GL_ARB_pixel_buffer_object or OpenGL>=2.1
- GL_ARB_vertex_buffer_object or OpenGL>=1.5

The card is required to support "while" statements in the fragment shader. NVIDIA GeForce 5 series does not support the fragment shader; it should work with an NVIDIA GeForce 6 or higher series.

EXAMPLE

The vtkKWEGPUVolumeRayCastMapper can be utilized in the same manner as any other volume mapper, and it should be a simple process to change existing code to use this mapper. However, since this mapper uses some advanced OpenGL features, you must check at run-time to see if the mapper is supported by calling IsRenderSupported(). If the mapper is not supported due to hardware limitations on the platform, then an alternate mapper should be created. The following example illustrates this process.

```
// Create and initialize the mapper
vtkKWEGPUVolumeRayCastMapper *mapper =
  vtkKWEGPUVolumeRayCastMapper::New();
mapper->
  SetInputConnection(reader->GetOutputPort());
mapper->SetBlendModeToComposite();

// Create and initialize the volume
vtkVolume *volume = vtkVolume::New();
volume->SetMapper(mapper);
volume->SetProperty(property);

// Create the renderer and render window
vtkRenderWindow *renwin = vtkRenderWindow::New();
vtkRenderer *ren = vtkRenderer::New();
renwin->AddRenderer(ren);

// Make sure we have a valid OpenGL context
renwin->Render();

// Check for support
if(mapper->IsRenderSupported(renwin, property))
 {
  ren->AddViewProp(volume);
  renwin->Render();
 }
```

In the preceding example, if the support check fails, a different volume mapper that is supported on the current OpenGL context given the properties of the volume would need to be created. Note that several other VTK volume mappers (e.g. vtkVolumeTextureMapper3D, vtkVolumeProVP1000Mapper) have this same requirement of checking for support. VTK Edge provides a convenience volume mapper called vtkKWEVolumeMapper that will check the hardware capabilities at run-time and will create and delegate to the best supported volume mapper given current OpenGL context, properties of the volume, and the allocated render time of the volume.

ALGORITHM DETAILS

The first step of the rendering process is to ensure that all the necessary data is located in texture memory. This includes the 3D texture representing the volumetric data, as well as some 1D textures representing the transfer functions mapping scalar value to color and opacity. Care is taken to copy these textures from main memory to the GPU texture memory only when they have changed.

The next step is to create a bounding polyhedron that represents the bounding box of the volume clipped with the near, far, and arbitrary clipping planes that may be applied to the volume. The back-facing polygons of the bounding polyhedron are drawn into the current Zbuffer in order to facilitate ray termination during the ray casting process. This ZBuffer also contains the depth values for the opaque geometry in the scene which was rendered prior to the start of the volume rendering phase.

In the next step, the front-facing polygons of the bounding polyhedron are rendered, and the rasterization of each face automatically triggers on each pixel a stream processor to execute the fragment shader code. The fragment shader uses the pixel position as the starting point to trace a ray through the 3D texture representation of the volume. The scalar values sampled from the 3D texture are used as texture coordinates into the 1D textures defining the color and opacity mappings. If shading is enabled, additional 3D texture samples are taken to perform a central difference operation to estimate the gradient. The properties of the samples are processed and combined according to the selected blending mode. Ray traversal is terminated when either the ray passes the depth value stored in the ZBuffer, or full opacity is reached.

In order to support the ability to decrease rendering time (at the expense of image quality) the frame buffer object (FBO) into which the image is rendered is independent in size from the pixel dimensions of the renderer. This size can be adjusted to control rendering speed. The last stage is to render a quad the size of the window on which the color buffer of the FBO is mapped. During the rasterization of the quad, another fragment shader is triggered to manage final brightness and contrast of the rendered image.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under award number OII-0548729.

__Dr. François Bertel__ is a technical developer in Kitware's Clifton Park, NY office. His current interests include GPU programming applied to visualization.

BACKWARD COMPATIBILITY IN ITK

The topic of backward compatibility has generated animated discussions among ITK users and developers for several years. We have gathered multiple points of view on this topic from some of our most influential users and developers. Condensed versions of these position statements are published in this article; the full text can be found on the ITK wiki at http://www.itk.org/Wiki (search with the keyword "backward" to find it.) The purpose of this article is not to declare a conclusive solution, but rather to show the complexity of the issue and the variety of diverse opinions in the community. We welcome your feedback on this controversial topic.

POSITION STATEMENT 1 (BILL LORENSEN)

One of the major criticisms of open-source software is that new revisions are not compatible with old revisions. Breaking compatibility impedes the acceptance and utility of open-source software. On the other hand, strict backward compatibility polices can impede innovation in software. The tension between these two viewpoints is not easily resolved.

As projects mature and the customer base grows, backward compatibility becomes more important. Commercial hardware and software products call this customer base, the installed base. Commercial products usually have a known customer base consisting of those who have purchased or licensed the software. Also, commercial systems seldom expose internal API's. Open source projects rarely know the identities of their customers. And, since the source is open, customers have access to all public and protected classes, methods and data in the code. For open source software, it is almost impossible to determine how the customer base is using the software.

During the early years of ITK development, API changes were required as we refined the system architecture and refactored code contributed by the distributed ITK development team. Now, ITK is over eight years old. The ITK developers have the responsibility to ITK's customers to ensure that released software conforms to software guidelines, respects intellectual property, and compiles and runs on supported platforms.

It must always be difficult to change an existing API. Every change, no matter how small, should be questioned. The burden for change is on the ITK developers. The primary goal is to minimize API changes, but when necessary, those changes should never cause user code to fail to compile. Compilation errors cannot report to a user how to correct the code in error. Documentation in user mailing lists or online forums like wiki's is not acceptable as the only venues for reporting how to achieve backward compatibility.

__Bill Lorensen__ is a software engineer residing in Ballston Lake, NY. Bill is one of the original authors of the Visualization Toolkit software/text and a main developer of the Insight Toolkit. He is a co-author of the textbook Object-Oriented Modeling and Design. His current interests include open source software quality and golf.

POSITION STATEMENT 2 (STEVE PIEPER)

This is primarily a policy discussion, and so the central issue is how to effectively communicate with users of the toolkit about what they get when they use a particular piece of code. The general policies could be summarized as:

- Different things should have different names.
- Similar things should have similar names.
- If two things have the same name, you can assume they will behave the same.

A way to interpret this is that if you come up with non-backwards compatible version of an algorithm, you should give it a new class name, like MyFilter2, rather than relying on the toolkit version number to indicate that it is different. Deprecation warnings at compile time can inform the developer that MyFilter is out of date. Dropping support for deprecated classes should happen when the toolkit itself gets a new name (like ITK4 instead of ITK3). Developers can choose when to migrate to a new class and/or a new version of the toolkit.

We should keep in mind that it is basically impossible to be absolutely backwards compatible. Even adding a new class or method could lead to a compile error if it conflicts with a name the user selected, so we are always talking about degrees of backwards compatibility which again emphasizes the importance of setting a policy that allows change and effectively communicates when these changes take place.

In addition to formulating naming conventions that explicitly indicate changed behavior, I would also propose that the community develop something that might be called 'testing contracts' between toolkits and their users. That is, toolkit users should be able to submit code that makes use of the toolkit in explicitly the way they depend on it behaving. These tests would be independent of the user's own evolving code so that anytime the tests fail it would be known to come from changes in the toolkit. A cross-platform build farm that continually rebuilt these 'testing contracts' would be a great service to the community.

__Steve Pieper__ is CEO of Isomics, Inc. in Cambridge, MA. Dr. Pieper works on several NIH sponsored research projects involving application of medical image computing technologies to research and therapeutic applications. Dr. Pieper is active in both academic and commercial uses of advanced technologies.

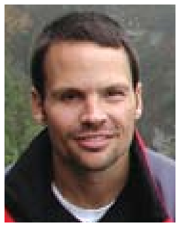
POSITION STATEMENT 3 (STEPHEN AYLWARD)

Backward compatibility is paramount in a toolkit, particularly one that is used by researchers. If a toolkit continually requires researchers to re-develop/re-test code that had previously worked, the toolkit will become viewed as an impediment to their work and eventually the user-pool will dwindle.

Backward compatibility applies to the API and the operation of a toolkit. One person's bug is another person's feature. An incorrectly spelled function name is not a bug once that function has been called by a user. Subsequently, changing the function's name to the correct spelling does create a bug in the user's code. The same bug/feature dichotomy exists when the API of a set of functions is perceived to be inconsistent. The same bug/feature dichotomy may even exist when a function has side-effects that are perceived as unwanted or even when a function has outputs that are perceived as incorrect. The guiding philosophy should be: *Once a function is released and it performs a particular operation, even if the operation it performs is not what was originally intended, its operation cannot be considered a bug.*

Admittedly, radical changes are occasionally needed in a toolkit to keep it current. When making those changes, it is important to apply an otherwise contrary adage: "if you are going to break a standard, then you should REALLY break the standard." That is, the changes to the toolkit should be planned and drastic. Planned means that the changes (1) should be announced and discussed well in advance of their release; (2) should be well supported, with clear transition paths and documentation; and (3) should be driven by the needs of the community. Drastic means that the changes should be extensive. If the changes being introduced are subtle then they could instead be done in a backward compatible way or as an extension of the existing framework. Breaking backward compatibility should only be allowed if the collective voice of the user community calls for a change that necessitates the complete overhaul of a framework or function to support the trends in research, hardware or compilers.

Balancing the above issues is best handled by a systematic process for adopting new features, testing backward compatibility and implementing alternative frameworks. The Insight Toolkit has a well established method for adopting new features: the Insight Journal. Testing backward compatibility is enabled by CTest and coverage counts, but disciplined application of those technologies are and will probably always be a challenge. An established mechanism for implementing, reviewing and releasing alternative frameworks does not currently exist. Establishing such a mechanism (policy and software) is critical to the continued success of the Insight Toolkit; if such a mechanism could be provided, then daily backward compatibility challenges would have a controlled outlet.



***Stephen R. Aylward, Ph.D.** is Chief Medical Scientist at Kitware, Inc. and manager of their North Carolina office. Dr. Aylward's research has recently focused on developing model-to-image registration strategies for intra-operative images and statistically characterizing vascular network variations for disease detection. He is working at Kitware to integrate those technologies into commercial products and to collaborate with other companies and universities on funded research.*

POSITION STATEMENT 4 (SIMON WARFIELD)

Backward compatibility in the Insight Toolkit is an important issue that must balance the needs of the Insight community for stability, innovation and clarity.

One of the key reasons for the success of open source software has been the credo of 'release early, release often'. With this approach, the early deployment of software before it is fully tested and validated, has been found to enable the rapid development of useful and important software. A consequence of this approach is that software is deployed to the community as it is developed rather than after it is developed.

In a young code base undergoing rapid expansion code can be implemented and regression tested faster than the community can fully appreciate. When this occurs, the interface to functionality erected by a particular implementation needs to be considered on its merits and not regarded as a sacrosanct interface. Being first doesn't mean being best, and shouldn't mean immortality. A regression test will ensure that the implementation achieves what the developer wanted, but doesn't ensure that what the developer wanted was what the community comes to understand is the best strategy to preserve under the obligation of backward compatibility. Indeed, it may be valuable to trial several different interfaces to particular functionality before it becomes clear what will be easiest to use and what will lend the greatest clarity to the largest number of developers.

In general, maintaining compatibility with previously released versions is desirable, because it allows code that utilizes prior releases to adopt new releases easily, with no burden on the developer community, while providing the benefits of new or improved functionality included in the new release. However, an excessive insistence on backward compatibility can hamper innovations, prevent bugs from being fixed, and destroy the aesthetic pleasure of a well-designed application programmer interface.

In particular, when a bug is discovered, a decision to maintain and preserve code that functions incorrectly for the sake of backward compatibility is wrong. Bugs should be fixed, and in cases where external code depends on wrong results from a function to operate correctly, those in the community who chose to adopt new versions or new releases of the code base, will need to be updating their code when new code to establish correct operation is implemented.

Similarly, code that implements a poor API or which has made wrong design choices, needs to be carefully and thoughtfully eliminated as the toolkit matures, rather than having that code add to the cost and investment that the user community makes in maintaining the toolkit.

***Dr. Simon Warfield** is the founder and Director of the Computational Radiology Laboratory (CRL) in the Department of Radiology at Children's Hospital Boston, a Research Affiliate of CSAIL at Massachusetts Institute of Technology and an Associate Professor of Radiology at Harvard Medical School. His research in the field of medical image analysis has focused on methods for quantitative image analysis through novel segmentation and registration approaches, and in real-time image analysis, enabled by high performance computing technology, in support of surgery.*

POSITION STATEMENT 5 (ROSS WHITAKER)

In this case of backward compatibility we should exercise moderation, and avoid the extremes. The extremes are both a) we change what we need to, when we need to and the users beware and b) that we must support, indefinitely, every feature that has ever been introduced into the API. The right solution is somewhere in between, and the proper choice depends on the use of the toolkit, the number of users, the types of users, they way they use the product, and the mechanisms by which the product is supported and maintained.

The argument to support legacy interfaces and functionality is clear. If we want serious users, who build big systems or products, we need to offer a degree of stability. Failing to be backward compatible is a serious concern, and once people have invested time and money and have been burned by a changing toolkit that fails to support their legacy code, we lose an important base of customers, supporters, and developers.

So why not have a policy of full backward compatibility that continues indefinitely? Supporting every feature indefinitely in every API is neither practical nor desirable. A community supported, cutting edge toolkit such as ITK must evolve. Furthermore, size matters. If a toolkit is too big or too confusing to understand it is not useful. A toolkit such as ITK must be organized in such a way that it is comprehensible to people in a reasonable amount of time. All of this is undermined by a huge set of redundant interfaces or functionalities that are to support the legacy of code. Furthermore, legacy code must be maintained, and we have limited resources.

If we decide on full backward compatibility, it seems to me that we decide that the interface is either stagnant or constantly growing. In either case we reduce the lifetime of the active code (the active lifetime, the legacy uses could continue indefinitely). Developers and users who want real change will have to start with a clean slate (this is currently under discussion among some developers in the ITK community). Furthermore, full backward compatibility should not be expected of users. It is not such a worthwhile goal. Users of the API will change their code to account for new hardware, operating systems, drivers, and compilers. There is no reason to expect to compile the same unmodified, applications against ITK over a space of more than a couple of years.

The middle ground is achieved by careful, thoughtful, well-implemented changes to the API combined with tools and procedures for helping users with legacy code manage this process. How can we be careful? Changes to the API that are not backward compatible must be proposed and reviewed by diverse groups who have a vested interest, not by single individuals who may not understand all the issues. Second, we need to allow these changes to proceed slowly with proper warnings to users during compile time. Thirdly, we need to provide users who don't want to modify their legacy code a way out, for instance, building against old versions. They might not have access to bug fixes and new functionality, but expecting full compatibility and progress is not realistic. Finally, we need to inform users of our policy and what they can expect in terms of compatibility if they decide to use our tools.

***Dr. Ross Whitaker** is an Associate Professor in the School of Computing and a faculty member of the Scientific Computing and Imaging Institute at the University of Utah. He teaches image processing, computer vision, and scientific visualization. His research interests include computer vision, image processing, medical image analysis, surface modeling and visualization.*

SELECTING DATA IN PARAVIEW

INTRODUCTION

One of the major design goals of ParaView 3 is to add support for quantitative analysis. In addition to better charting, Python-based filtering and statistical analysis tools, we have been working on the capability of focusing the analysis on a specific subset of a dataset. This can be achieved using the selection mechanism described in this article.

Selection is the mechanism for identifying a subset of a dataset by using user specified criteria. This subset can be a set of points or cells or blocks in a composite dataset. This functionality allows users to focus on a smaller subset that is important. For example, the elements of a finite-element mesh that have pressure above a certain threshold can be identified very easily using the threshold selection. Furthermore, this selection can be converted to a set of global element IDs in order to plot the attribute values of those elements over time.

ParaView supports a single active selection. This selection is associated with a data source (here data source refers to any reader, source or filter) and is shown in every view that displays the data source's output. This article uses a use-case driven approach to demonstrate how this selection can be described and used. In the next section, we introduce the main GUI components that are used in the article. The subsequent sections address different use cases.

Please note that many features discussed in this article are recent additions and are not available in 3.2. You may want to download a development snapshot or build your own binary from the CVS source base. Otherwise, you will have to wait until 3.4 is out.

SELECTION INSPECTOR

ParaView provides a *selection inspector* (referred to simply as the inspector in this article) to inspect and edit the details about the active selection. One can toggle the inspector visibility from the View menu. The inspector can be used to create a new active selection, view/edit the properties of the active selection as well as change the way the selection is displayed in the 3D window (e.g., change the color, show labels, etc.).

We will look at each of these options as we try to explore the different selection types in ParaView.

SPREADSHEET VIEW

Spreadsheet View provides data exploration capabilities. One of the common complaints many users have is not being able to look at the raw data directly; spreadsheet view allows the user to look at the raw cell data, point data or field data associated with a dataset.

ParaView treats spreadsheet view exactly like the other views such as the 3D view and Bar Chart view. To create the spreadsheet view, first split the workspace and then choose *Spreadsheet View* from the options listed. At this time if any source is currently selected in the *Pipeline Browser*, the spreadsheet view will automatically shows its point data. We can choose the source whose output we want to view using the *eyeball* in the *Pipeline Browser*, which is equivalent to turning on the visibility of the source in this view. Spreadsheet view can only show one dataset at a time.

When the spreadsheet view is active (selected by clicking on its toolbar) and the visible source is selected in the pipeline browser the *Display* tab in the *Object Inspector* panel can be used to control what is shown. This panel contains a selector that can be used to toggle between point and cell data. For composite datasets (multi-block and AMR datasets) it shows a *Composite Data Structure* tree that can be used to select the block that is shown in the view. Another widget named *Show only selected elements* can be selected to restrict what is shown in the spreadsheet to only selected cells or points. We will revisit this check box later.

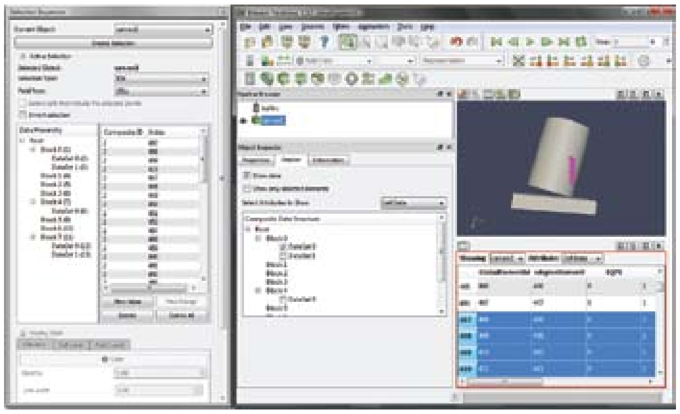


Figure 1: Spreadsheet View in ParaView showing the display tab for a source producing a multi-block dataset. The selected cells are highlighted in the 3D view as well as the spreadsheet view. The active selection can be inspected using the selection inspector.

CREATE A SELECTION

In this section we will discuss different ways of creating a selection.

SELECT CELLS/POINTS ON THE SURFACE

One of the simplest use-cases is to select cells or points on the surface of the dataset. It is possible to select surface cells by drawing a rubber-band on the 3D view. With 3D view showing that the dataset is active click on *Select Cells* (or *Points*) *On* in the *Selection Controls* toolbar or under the *Edit* menu (you can also use the '*S*' key as a shortcut for '*Select Cells On*'). This will put ParaView into a selection mode. In this mode, click and drag over the surface of the dataset in the active view to select the cells (or points) on the surface. If anything was selected it will be highlighted in all the views showing the data and the source producing the selected dataset will become active in the *Pipeline Browser*. ParaView supports selecting only one source at a time. Hence even if you draw the rubber band such that it covers data from multiple sources, only one of them will be selected (the one that has the largest number of selected cells or points).

As mentioned earlier, when data from a source is selected all the views displaying the data show the selection, this includes spreadsheet view as well. If the spreadsheet view will show cell or point attributes of the selected data, then it will highlight the corresponding rows. When selecting points, the spreadsheet view will show the selection only if point attributes are being displayed. When selecting cells, it will highlight the cells in the cell attribute mode and highlight the points forming the cells in the point attribute mode. For any decent sized dataset it can be a bit tricky to locate the selected rows. In that case, the *Show only selected elements* on the display tab can be used to hide all the rows that were not selected.

When selecting cells (or points) on the surface, ParaView determines the cell (or point) IDs for each of the cells (or points) rendered within the selection box. The selection is simply the IDs for cells (or points) thus determined.

SELECT CELLS/POINTS USING A FRUSTUM

This is similar to selecting on the surface except that instead of selecting the cells (or points) on the surface of the dataset, it selects all cells (or points) that lie within the frustum formed by extruding the rectangular rubber band drawn on the view into 3D space. To perform a frustum selection, we use *Select Cells (or Points) Through in the Selection Controls* toolbar or under the *Edit* menu. As with surface selection, the selected cells/points are shown in all the views in which the data is shown including the spreadsheet view. Unlike surface selection, the indices of the cells or points are not computed after a frustum selection. Instead, ParaView performs intersections to identify the cells (or points) that lie within the frustum. Note that this selection can produce a very large selection. This may be time consuming and can increase the memory usage significantly.

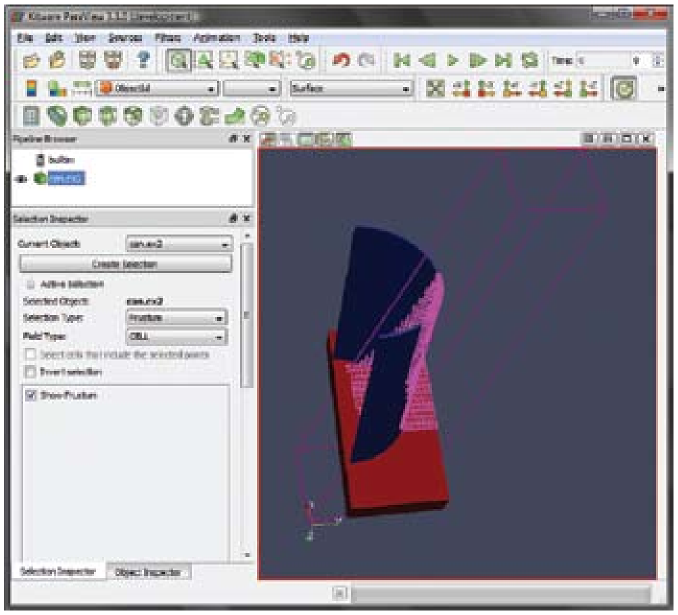
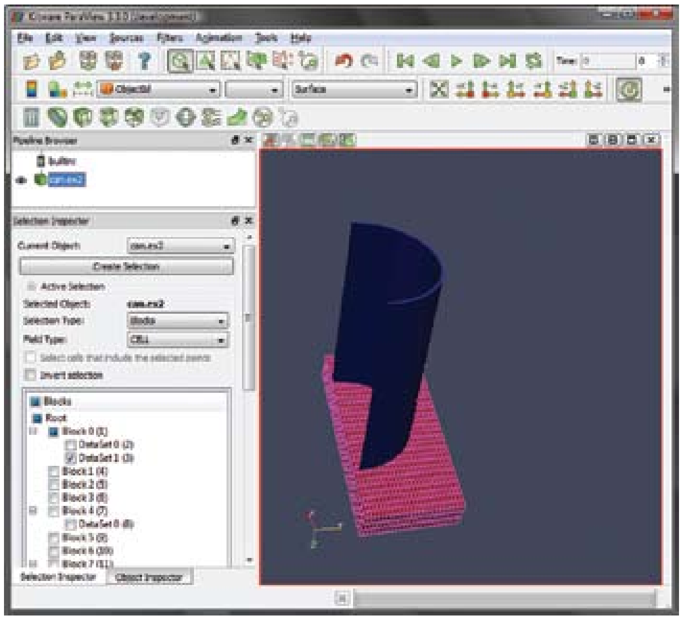


Figure 2: Selection using a Frustum. Note that all cells that lie within the specified frustum are selected. The selection inspector shows the details of the selection.

SELECT BLOCKS IN A COMPOSITE DATASET

Composite datasets are multi-block or AMR (adaptive mesh refinement) datasets. In the case of multi-block datasets, each block may represent different components of a large assembly (e.g., tires, chassis, etc.) for a car dataset. Just like selecting cells or points, it is possible to select entire blocks. To enter the block selection mode use *Select Block* in the *Selection Controls* toolbar or under the *Edit* menu (you can also use the '*B*' key as a shortcut for *Select Block*). Once in block selection mode, you can simply click on the block in the 3D view to select a single block or click and drag to select multiple blocks. When a block is selected its surface cells will be highlighted.



Current Object: can.ex2
Selected Object: can.ex2
Selection Type: Blocks
Field Type: CELL

Figure 3: Selecting a block in multi-block dataset. All the cells in the selected block are highlighted. The selection inspector shows the selected block.

SELECT USING THE SPREADSHEET VIEW

Until now we have been looking at defining the selection on the 3D view. Now we will see how to create selections using the spreadsheet view. As we discussed earlier, the spreadsheet view simply shows the raw cell (point or field) data in a table. Each row represents a unique cell (or point) from the dataset. Like with any other spreadsheet application, one can select a cell (or a point) by simply clicking on the row to select. One can expand the selection using *Ctrl, Shift* keys while clicking. If the spreadsheet view is currently showing point attributes, then selecting on it will create a point based selection. Similarly, if it's showing cell attributes then it will create a cell-based selection. Selection cannot be created when showing field attributes which are not associated with any cell or point.

We know that all views showing a selected dataset show the selection. The spreadsheet view showing the data from a source selected in the 3D view highlights the corresponding cells/points. Conversely, when we create a selection in the spreadsheet view, the corresponding cell (or point) gets highlighted in all of the 3D views showing the source.

SELECT USING THE SELECTION INSPECTOR

We have seen how to use the different views for creating different types of selections however sometimes you may want to tweak the selection or create a new selection with a known set of cell (or point) IDs or create selections based on values of any array, location, etc. This is possible using the selection inspector.

Whenever a selection is created in any of the views it becomes the active selection. The active selection is always shown in the selection inspector. For example, if you select cells on the surface of a dataset, as shown in Figure [1], the selection inspector will show indices for the cells selected.

The selection inspector has three sections: the topmost *Current Object* and *Create Selection* are used to choose the source whose output you want to create a selection on. The *Active Selection* group shows the details of the active selection, if any. The *Display Style* group makes it possible to change the way the selection is shown in the active 3D view.

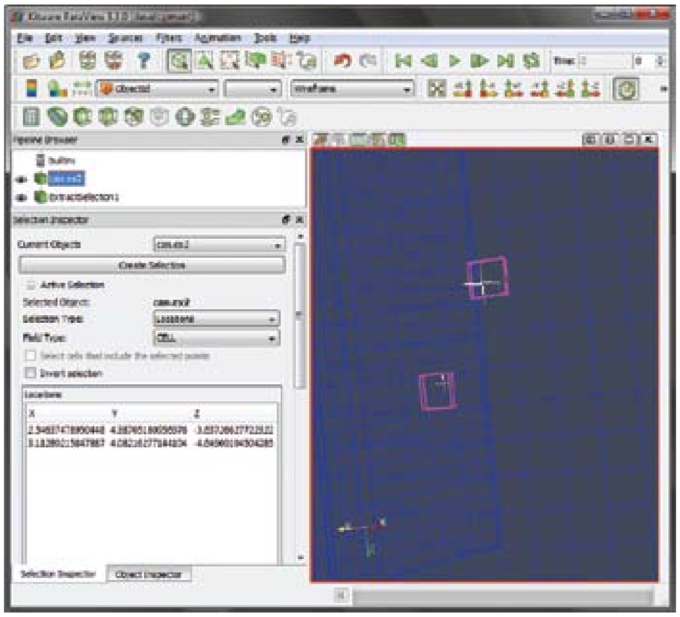
To create a new selection, choose the source whose output needs to be selected in the *Current Object* combo-box and then hit *Create Selection*. An empty selection will be created and its properties will be shown in the active selection group. Alternatively, you can use any of the methods described earlier to create a selection. It will still be shown in the selection inspector.

When you select cells (or points) on the surface or using the spreadsheet view, the *selection type* is set to *IDs*. Creating a frustum selection results in a selection with the *selection type* set to *Frustum*, while selecting a block in a composite dataset creates a *Block* selection. *Field Type* indicates whether cells or points are to be selected. In the active selection group, *Selection Type* indicates the type of the active selection. One can change the type by choosing one of the available options.

As shown in Figure [1], for IDs selection, the inspector lists all the selected cell or point indices. You can edit the list of IDs to add or remove values. When connected to a parallel server, cell or point IDs are not unique. Therefore, one has to additionally specify the process number for each cell or point ID. Process number -1 implies that the cell (or point) at the given index is selected on all processes. For multi-block datasets, we also need to indicate the block to which the cell or point belongs; for AMR datasets, we need to specify the (AMR level, index) pair.

As shown in Figure [2], for *Frustum* selection, currently only the *Show Frustum* option is available. When this option is turned on, ParaView shows the selection frustum in the 3D view. In the future, we will implement a 3D widget to modify the frustum. As shown in Figure [3], for Block selection, the full composite tree is shown in the inspector with the selected blocks checked. Using the selection inspector, one can create a selection based on thresholds for scalars in the dataset. Choose the scalar array and then add value ranges for the selected cells (or points).

Selection inspector can be used to create location based selections. When field type is CELL, cells at the indicated 3D locations will be selected. When field type is POINT, the point closest to the location within a certain threshold is selected. If *Select cells that include the selected points* is checked, then all the cells containing the selected point are selected. It is possible to specify more than one location. To aid in choosing positions, one can turn the *Show location widgets* option on and ParaView will show crosshairs in the active 3D view which can be moved interactively (as shown in Figure [4]).


Pipeline Browser

builtin:

can.ex2

ExtractSelection1

Selection Inspector

Current Object: can.ex2

Create Selection

Active Selection

Selected Object: can.ex2

Selection Type: Locations

Field Type: CELL

Select cells that include the selected points

Invert selection

Locations

X	Y	Z
2.94837478950448	4.88785186058078	-3.83728617722932
3.18286015847887	4.08216277184834	-4.84960194504285

Selection Inspector | Object Inspector

Figure 4: Location based selection showing the cross hairs used to specify the locations.

Selection inspector also provides a means to create global ID-based selection. This is similar to index-based selection however since global IDs are unique across all processes and blocks, one does not need to specify any additional IDs as needed by the ID-based selection.

CONVERT SELECTIONS

Selection inspector can also be used to convert a selection of one type to another. With some valid active selection present, if one changes the selection type then ParaView will try to convert the current selection to the new type, still preserving the cells (or points) that were selected, if possible. For example, if one creates a frustum-based selection and then changes the selection type to IDs, ParaView will determine the indices for all the cells (or points) that lie within the frustum and initialize the new index-based selection with those indices. Note that the number of cells or points that get selected in frustum selection mode can potentially be very large; hence this conversion can be slow and memory expensive. Similarly, if the dataset provides global IDs, then it is possible to convert between ID selection and global ID-based selection.

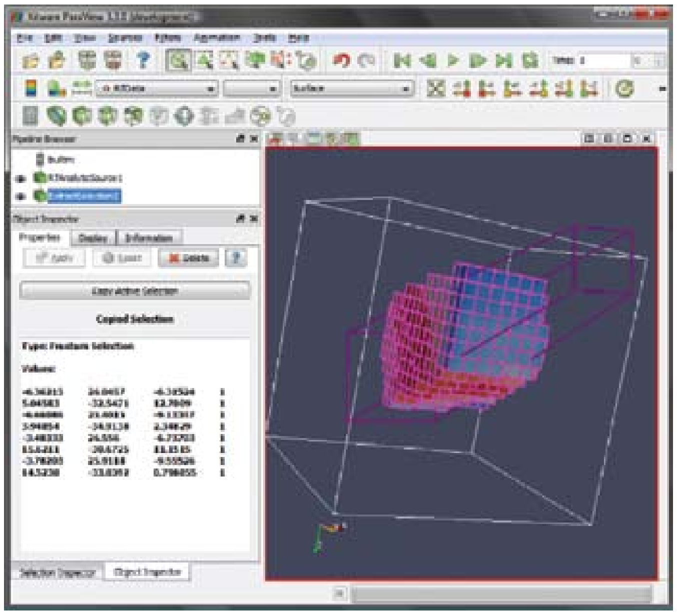
It is not possible to convert between all types of selections due to obvious reasons. Conversions between ID-based and global ID-based selections, conversions from frustum to ID-based selections, and conversions from frustum to global ID-based selections are supported by the selection inspector.

LABEL SELECTED CELL/POINTS

Once an active selection is created, we can label the selected cells or points in a 3D view. This can be done using the selection inspector. At the bottom of the selection inspector panel there are two tabs, *Cell Label* and *Point Label*, which can be used to change the cell or point label visibility and other label attributes such as color and font. These tabs are enabled only if the active view is a 3D view. Any changes done in the *Display Style* group (including the labels) only affect the active 3D view.

EXTRACT SELECTION

Selection makes it possible to highlight and observe regions of interest. Oftentimes once a region of interest has been identified, one would like to apply additional operations on it, such as filters to the selected section of the data. This can be achieved using the *Extract Selection* filter. To set the selection to extract, create a selection using any of the methods already described. Then apply the extract selection filter to the source producing the selected data. To copy the active selection to the filter, use the *Copy Active Selection* button. One can change the active selection at any time and update the filter to use it by using this button. Figure [5] shows the extract selection filter applied after a frustum selection operation. Now, one can treat this as any other data source and apply filters to it, save state, save data, etc.


Pipeline Browser

builtin:

RTAnalyticSource1

ExtractSelection1

Object Inspector

Properties | Display | Information

Apply | Reset | Delete | ?

Copy Active Selection

Copied Selection

Type: Frustum Selection

Values:

```
-6.36815    24.0457    -6.34534    1
5.04583     -32.5471   12.7609     1
-6.66086    21.4015    -9.13317    1
3.59854     -34.9138   2.34829     1
-3.48333    26.556     -6.73703    1
15.6211     -30.6725   11.1515     1
-3.78205    25.9188    -9.55526    1
14.5238     -33.8392   0.798855    1
```

Selection Inspector | Object Inspector

Figure 5: Extract selection using a frustum selection

PLOT SELECTION OVER TIME

For time varying datasets, you may want to analyze how the data variables change over time for a particular cell or a point. This can be done using the *Plot Selection Over Time* filter. This filter is similar to the *Extract Selection* filter, except that it extracts the selection for all time-steps provided by the data source (typically a reader) and accumulates the values for all the cell (or point) attributes over time. Since the selection can be comprised of multiple cells or points, the display tab provides the *Select Block* widget which can be used to select the cell or point to plot, as shown in Figure [6]. Currently only one cell (or point) can be plotted at once in the same xy-plot view. One can create multiple plot views to show multiple plots simultaneously.

CONCLUSION

Although we touched on every aspect of the selection functionality in ParaView, we only scratched the surface of what can be achieved with this extremely powerful feature. The selection mechanism provides many ways of focusing on the important parts of your dataset and, with the help of the spreadsheet view, a direct way to view the raw data. With some experimentation, you will probably discover new ways of using it that we have not anticipated. That is the way of ParaView.

ACKNOWLEDGEMENTS

This work has been partially funded by Sandia National Laboratories through contract number DE-AC04-94AL85000 and the Los Alamos National Laboratory through contract number DE-AC52-06NA25396.

***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, and is also a developer on the ParaView project.*

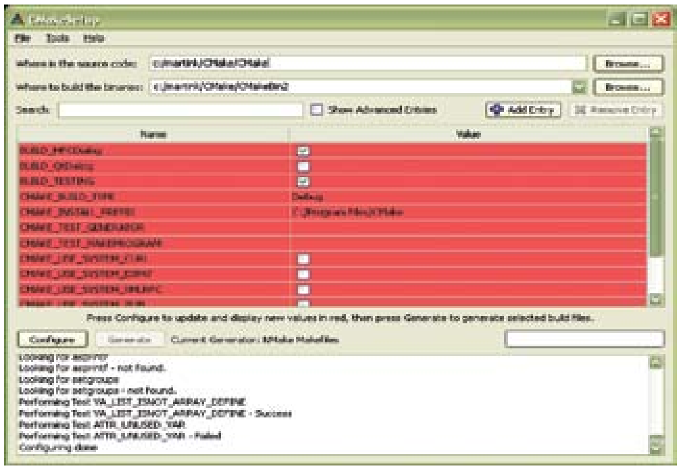
***Berk Geveci** is a project lead in Kitware's Clifton Park, NY office. Dr. Geveci is the project manager for the open source ParaView application. He contributes to the technical development of VTK and ParaView.*

CMAKE 2.6

CMake 2.6 is the first major release of CMake since the 2.4 release in May of 2006. This release incorporates many new features including a Qt-based interface, a CMake Policy mechanism, functions, Fortran support, OSX framework creation, automatic project reloading in Visual Studio, and support for Bullseye coverage. Details on these new features are provided below.

USER INTERFACE

The new Qt-based CMake user interface is shown below.



Where is the source code: c:/martink/CMake/CMake
Where to build the binaries: c:/martink/CMake/CMakeBin2

Name	Value
BUILD_MFCDialog	✓
BUILD_QtDialog	☐
BUILD_TESTING	✓
CMAKE_BUILD_TYPE	Debug
CMAKE_INSTALL_PREFIX	C:/Program Files/CMake
CMAKE_TEST_GENERATOR	
CMAKE_TEST_MAKEPROGRAM	
CMAKE_USE_SYSTEM_CURL	☐
CMAKE_USE_SYSTEM_EXPAT	☐
CMAKE_USE_SYSTEM_XMLRPC	☐

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: NMake Makefiles

```
Looking for asprintf
Looking for asprintf - not found.
Looking for setgroups
Looking for setgroups - not found.
Performing Test VA_LIST_ISNOT_ARRAY_DEFINE
Performing Test VA_LIST_ISNOT_ARRAY_DEFINE - Success
Performing Test ATTR_UNUSED_VAR
Performing Test ATTR_UNUSED_VAR - Failed
Configuring done
```

Figure 1: New Qt-based CMake GUI

Since Qt is a cross-platform GUI toolkit, the new CMake interface can run on Linux, Windows, and the Mac. The CMake 2.6 GUI has some very powerful features such as incremental search ability. It also has a scrolling text window that stores output from the CMake run.

POLICY MECHANISM

The first new feature that users will most likely run into is the *cmake_policy* command. This was added to provide a way to make changes to CMake in a backwards compatible way. As CMake evolves it is sometimes necessary to change existing behavior in order to fix bugs or improve implementations of existing features. The CMake Policy mechanism is designed to help keep existing projects building as new versions of CMake introduce changes in behavior. Each new policy (behavioral change) is given an identifier of the form "CMP<NNNN>" where "<NNNN>" is an integer index. Documentation associated with each policy describes the OLD and NEW behavior and the reason the policy was introduced. Projects may set each policy to select the desired behavior. When CMake needs to know which behavior to use it checks for a setting specified by the project. If no setting is available the OLD behavior is assumed and a warning is produced requesting that the policy be set. The CMake policy level is tied to the *cmake_minimum_required* command, this command sets the policy to NEW for each policy that was created in this version of CMake. So, for CMake 2.6.0, if *cmake_minimum_required* is set to 2.6.0, then all policies will be set to NEW behavior.

The first policy that users are likely to see is CMP0000. CMake now requires that projects specify the version of CMake to which they have been written. This policy has been put in place so users trying to build the project may be told when they need to update their CMake. Specifying a version also helps the project build with CMake versions newer than that specified. Use the *cmake_minimum_required* command at the top of your main CMakeLists.txt file:

```
cmake_minimum_required(VERSION <major>.<minor>)
```

The "<major>.<minor>" is the version of CMake you want to support (such as "2.6"). The command will ensure that at least the given version of CMake is running and help newer versions be compatible with the project. See documentation of *cmake_minimum_required* for details.

Note that the command invocation must appear in the CMakeLists.txt file itself; a call in an included file is not sufficient. However, the *cmake_policy* command may be called to set policy CMP0000 to OLD or NEW behavior explicitly. The OLD behavior is to silently ignore the missing invocation. The NEW behavior is to issue an error instead of a warning. An included file may set CMP0000 explicitly to affect how this policy is enforced for the main CMakeLists.txt file.

The next policy that users are likely to see is CMP0003. This policy is directly related to the new feature of CMake using full paths to link libraries and no longer separating the link into a path *(-L/some/path)*, and a link library *(-lsomelib)*. This change allows users to get the exact library that they asked CMake to link which is a good thing. However, the change has a side effect that may break linking for some projects. It turns out that some projects (including VTK) were depending on the linker being told the paths for all the libraries. Here is a short example of the problem:

```
# Link library A and B to target foo
target_link_libraries(foo /path/to/libA.a B)
# library B is in the same directory as A
```

So, in CMake versions prior to 2.6, the link line would be *–L/path/to –lA –lB*. However, In CMake 2.6.0, the link line would be */path/to/libA.a –lB*, and the linker would fail to find B, because of the missing *–L/path/to*. CMake has easy way to detect when this error is going to occur at CMake time. So, to avoid giving users unexpected errors at build time, if CMake finds libraries that are linked without a full path at the same time as other libraries with full paths, it issues a CMP0003 warning. Along with the warning CMake will add the –L paths for all full path libraries being linked into the target.

The correct fix if you find this warning in your project is to set the policy to NEW. This will change the behavior of CMake and the extra –L paths will not be added. If the project has link trouble at build time, then the extra link paths should be added or full paths should be used for the non full path link items. A common misconception is that CMake now requires full paths to all libraries. In cases such as system libraries like libm or pthreads, it is often better to let the compiler find the library, and in fact adding the full path may cause new errors.

If you still want your project to work with prior versions of CMake, a good way to set the policy level is to first check to make sure that the cmake_policy command exists. For example,

```
if(COMMAND cmake_policy)
  cmake_policy(SET CMP0003 NEW)
endif(COMMAND cmake_policy)
```

For more information on policies see http://www.cmake.org/Wiki/CMake_Policies and http://www.cmake.org/HTML/cmake-2.6.html#section_Policies.

FUNCTIONS

In addition to macros, CMake now supports the creation of functions in CMake code. The functions differ from the macros in that they are more like a function in C. A simple function would look like this:

```
function(myfunction argument)
  message("argument = ${argument}")
endfunction(myfunction)
```

You can pass arguments into it. The arguments passed in become variables within the function. Likewise some standard variables such as ARGC, ARGV, ARGN, and ARGV0, ARGV1, etc are defined. Within a function you are in a new variable scope, much like how when you drop into a subdirectory you are in a new variable scope. All the variables that were defined when the function was called are still defined, but any changes to variables or new variables only exist within the function. When the function returns those variables will go away. Put more simply when you invoke a function a new variable scope is pushed and when it returns that variable scope is popped.

SCOPE, RETURN, AND BREAK

The set command has a new argument PARENT_SCOPE, which allows you to set a variable in the parent's scope as opposed to the current scope. This can be used to pass variable values from a function to the calling function or CMakeLists.txt file.

The return and break commands have been added. They behave the same as a C return or break command would. A break within a foreach or while loop will break out of the loop. A return from within a function or list file will return from that function or CMakeLists.txt file.

FORTRAN SUPPORT

The Fortran support that has been added into CMake handles complex Fortran dependency generation. The CMake dependency scanner will determine the correct build order and automatically keep the dependencies up to date. In addition to makefile support for Fortran. CMake can now generate Intel Fortran Visual Studio projects. To enable Fortran in a project, either add Fortran to the language list in the *project* command, or use the *enable_language* command. One thing to note is that with Visual Studio project, you cannot have mixed C or C++ targets with Fortran. The library or executable must be all Fortran, this is a restriction of the IDE, and not CMake.

OSX FRAMEWORK CREATION

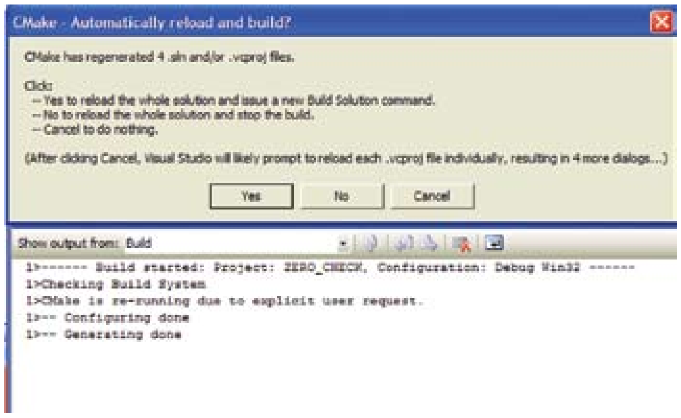
To create an OSX framework with CMake, you create a SHARED type library target. Then set the target property FRAMEWORK to true. To create a simple framework, the following commands would be used:

```
add_library(Foo  SHARED a.c b.c)
set_target_properties(foo PROPERTIES FRAMEWORK TRUE)
```

The library must be SHARED for a framework to be created, and it must have the FRAMEWORK property set to TRUE. The frameworks can have other target properties set as well. PUBLIC_HEADER, RESOURCES PRIVATE_HEADER, FRAMEWORK_VERSION, INSTALL_NAME_DIR. These properties are all documented in the CMake documentation section "Properties on Targets".

AUTOMATIC PROJECT RELOAD IN VISUAL STUDIO

CMake now installs some macros into the Visual Studio environment for Visual Studio 7 or newer. These macros allow CMake to be correctly re-run if an input to CMake has changed when a build is run. CMake has always re-run, but Visual Studio would prompt the user to reload each .vcproj file that was changed by CMake, and for large projects that could be a very slow process. Now CMake will prompt the user once, and Visual Studio will prompt once, and the project will correctly reload. If you run a build on a CMake project and CMake is re-run, you will see the following dialogs:



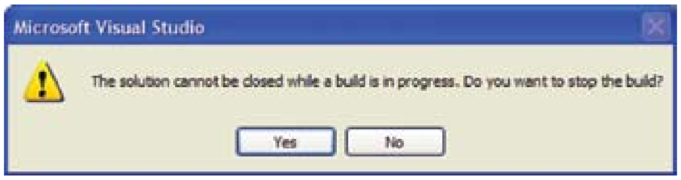
CMake has regenerated 4 .sln and/or .vcproj files.

Click:
-- Yes to reload the whole solution and issue a new Build Solution command.
-- No to reload the whole solution and stop the build.
-- Cancel to do nothing.

(After clicking Cancel, Visual Studio will likely prompt to reload each .vcproj file individually, resulting in 4 more dialogs...)

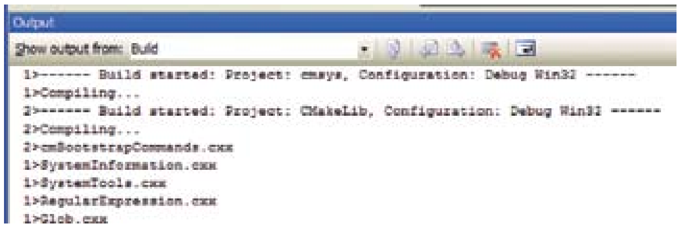
```
1>------ Build started: Project: ZERO_CHECK, Configuration: Debug Win32 ------
1>Checking Build System
1>CMake is re-running due to explicit user request.
1>-- Configuring done
1>-- Generating done
```

Figure 2: Visual Studio Reload Dialog One



The solution cannot be closed while a build is in progress. Do you want to stop the build?

Figure 3: Visual Studio Reload Dialog Two



```
1>------ Build started: Project: cmsys, Configuration: Debug Win32 ------
1>Compiling...
2>------ Build started: Project: CMakeLib, Configuration: Debug Win32 ------
2>Compiling...
2>cmBootstrapCommands.cxx
1>SystemInformation.cxx
1>SystemTools.cxx
1>RegularExpression.cxx
1>Glob.cxx
```

Figure 4: Visual Studio Continuous the Build After Reload

BULLSEYE COVERAGE

CTest includes support for a commercial tool called BullseyeCoverage (www.bullseye.com). This is a cross-platform tool that is easy to use, and integrates well with the new CDash. The following script illustrates how Bullseye can be used to supply coverage information for a project dashboard. In the first part of the script, we include the BullseyeCoverage binary directory in the path, and make sure that we can find the cov01 tool.

```
set(ENV{PATH}
  “C:\\BullseyeCoverage\\bin;$ENV{PATH}”)

find_program(COV01 cov01)
if(NOT COV01)
  message(FATAL_ERROR “Could not find cov01”)
endif(NOT COV01)
```

Now we define a function that will use the cov01 program to set the coverage mode on or off. In addition, we define two convenience functions for adjusting the coverage mode. We’ll use these convenience functions later in this script.

```
function(set_coverage_mode mode)
  set(RES 1)
  execute_process(COMMAND ${COV01} ${mode}
                  RESULT_VARIABLE RES)
  if(RES)
    message(FATAL_ERROR “could not run cov01 -0”)
  endif(RES)
endfunction(set_coverage_mode)

function(turn_coverage_off)
  set_coverage_mode(“-0”)
endfunction(turn_coverage_off)

function(turn_coverage_on)
  set_coverage_mode(“-1”)
endfunction(turn_coverage_on)
```

This section of the script defines various parameters including the site, build, and project names, the location of the source, and the command used to update the repository. This is a standard portion of the script that is not specific to the Bullseye coverage.

```
set(CTEST_SITE “dash1vista32.kitware”)
set(CTEST_BUILD_NAME “Win32Vista-vs80-cov”)
set(CTEST_NOTES_FILES
  “${CTEST_SCRIPT_DIRECTORY}/${CTEST_SCRIPT_NAME}”)
set(CTEST_DASHBOARD_ROOT “C:/Dashboards/My Tests”)
set(CTEST_SOURCE_DIRECTORY
  “${CTEST_DASHBOARD_ROOT}/MyProject”)
set(CTEST_BINARY_DIRECTORY “
  ${CTEST_DASHBOARD_ROOT}/MyProjectVS8Nightly”)

set(CTEST_UPDATE_COMMAND
  “C:/Program Files/Subversion/bin/svn”)

set(CTEST_CMAKE_GENERATOR “Visual Studio 8 2005”)
set(CTEST_PROJECT_NAME “MyProject”)
```

We need to set an environment variable to indicate the location for the generated coverage file. We’ll also start by removing our previous build tree to ensure a clean build process.

```
set (ENV{COVFILE} “${CTEST_BINARY_DIRECTORY}/CMB.
cov”)
ctest_empty_binary_directory(
  “${CTEST_BINARY_DIRECTORY}”)

# Create a cache file and fill with any useful
# variables
file(WRITE
  “${CTEST_BINARY_DIRECTORY}/CMakeCache.txt”
  “BUILD_SHARED_LIBS:BOOL=ON”)
```

Now the dashboard process begins. We’ll turn coverage off for the beginning update and configure steps. Then we turn coverage back on for the build and test steps. Finally, we turn coverage back off, collect the coverage information, and submit the dashboard.

```
turn_coverage_off()
ctest_start(Nightly)
ctest_update(SOURCE “${CTEST_SOURCE_DIRECTORY}”)
ctest_configure(BUILD “${CTEST_BINARY_DIRECTORY}”)

# re-read ctest custom files after configure step
ctest_read_custom_files(“${CTEST_BINARY_DIRECTORY}”)

turn_coverage_on()
ctest_build(BUILD “${CTEST_BINARY_DIRECTORY}”)
ctest_test(BUILD “${CTEST_BINARY_DIRECTORY}”)

turn_coverage_off()
ctest_coverage(BUILD “${CTEST_BINARY_DIRECTORY}”)
ctest_submit()
```

ACKNOWLEDGEMENTS

This work has been partially funded by the National Library of Medicine ITK project, and by the NA-MIC National Center of Biomedical Computing, NIH Roadmap for Medical Research, Grant# U54 EB005149.

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

IN PROGRESS

PARAVIEW 3.4

We are working hard on the release of ParaView 3.4. There is still a large list of issues to address before we can branch. Our goal is to address enough of these issues by July so that we can start the release process. We will continue to release development snapshots, visit paraview.org/New/download.html for the latest one.

ITK 3.8

The release of ITK 3.8 is scheduled for the end of July. This release benefits from an unprecedented participation of the world-wide ITK community. About twelve users have enthusiastically joined the ITK development team through the "Adopt a Bug" program. This influx of bright new brains have already brought fruition in the form of a wide spectrum of bug entries being addressed.

Some of the notable bugs include:

- Fixing crashes in particular conditions of the ExecptionObjects contributed by Niels Dekker
- Performance improvements in the SmartPointer transactions contributed by Tom Vercauteren

Among the highlights of this release we can mention:

- Support for the new gcc 4.3 and Visual Studio 9 2008 compilers, thanks to Steve Robbins and Casey Goodlett for contributing Nightly dashboard submission on gcc 4.3.
- Support for reading Philips PAR/REC and Bruker image file formats kindly contributed to the Insight Journal by Don Bigler
- Support for CUB image file format, contributed to the Insight Journal by Paul Yuskevich and collaborators from the University of Pennsylvania.
- New mesh processing filters for the QuadEdgeMesh, contributed by Alex Gouaillard and Arnaud Gelas, from Harvard University.

KITWARE NEWS

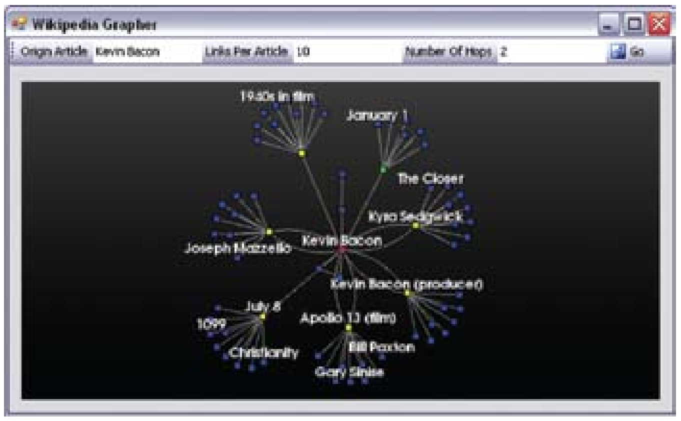
NAMIC YEAR 5

Kitware is a member of the National Alliance of Medical Image Computing (NA-MIC) and NIH National Center of Biomedical Computing (NCBC). The fifth year of this $20 million project begins August 2008. The focus of the coming year's work will be expanding the Slicer3 application, including additional support for image segmentation and registration, and the addition of 3D VTK widgets.

ACTIVIZ.NET

Kitware is pleased to announce the release of ActiViz .NET, Personal and Commercial Editions. The ActiViz .NET product offers a pre-built .NET integration for the Visualization Toolkit (VTK) system (vtk.org). ActiViz .NET enables C# and VB .NET developers to access the power of VTK from their familiar .NET environment. Supported functionality includes 3D graphics, volume rendering, information visualization, 3D interaction widgets, application callbacks, and geometric visualization and processing. ActiViz .NET is simple to use--after installation simply reference the Kitware.VTK.dll assembly from a Visual Studio project. Then drag and drop a VTK RenderWindowControl directly onto a Windows Forms application while editing the GUI in the Visual Studio designer.

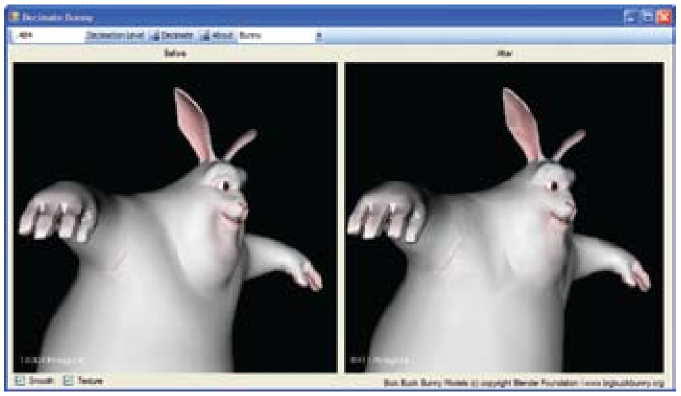
ActiViz .NET is based on the latest release of VTK version 5.2. The Personal Edition is available free of charge with a watermarked display. The Commercial Edition is distributed without the watermark and is licensed on an annual basis for $2,900/year per developer. With a paid commercial license, a developer may download any updates that are released during that year. See http://www.kitware.com/products/activiz.html for more information on both ActiViz/COM and ActiViz .NET.



Wikipedia Grapher

Origin Article Kevin Bacon   Links Per Article 10   Number Of Hops 2   Go

The 6 degrees of Kevin Bacon is illustrated as an example of the ActiViz RenderWindowControl in action inside Kitware's Wikipedia Grapher C# application.



This rendering example shows decimation of the character from a large number of polygons to a small number of polygons while still maintaining the character's basic shape. The character image is courtesy of the open source movie Big Buck Bunny (www.bigbuckbunny.org), ©copyright Blender Foundation.

ITK ADOPT-A-BUG PROGRAM

A new program has been created for improving the quality of the Insight Segmentation and Registration Toolkit. This program aims to directly engage users in the overseeing and resolution of reported bugs in the Toolkit. The program invites ITK users to "adopt a bug" thereby becoming responsible for making sure that the bug receives the treatment it deserves. Bug adopters are immediately promoted to developers and are entrusted with write access to the ITK CVS repository and associated resources. Thanks to all of you who have already signed up for this program! You are living proof of the power of open source.

To learn more about the Adopt-a-Bug Program and adopt your bug today head to itk.org/Wiki and search for "ITK Adopt a Bug Program."

NIH R01 FARSIGHT

Kitware has teamed with Dr. Badri Roysam of RPI on the Farsight Project-- a next generation image analysis toolkit to enable quantitative studies of complex multi-cellular structures such as stem-cell niches, brain tissue, and tumors, from multichannel three-dimensional fluorescence microscopy images. The team has been awarded a multi-year R01 grant from the National Institutes of Health.

MAVERICK BETA RELEASE TO AF

As part of an Air Force SBIR, Kitware has delivered a beta version of a new library and companion applications for 3D medical image segmentation and label map editing. A common, key feature of the applications is their workflow-based interfaces that allow users to intuitively apply complex methods for creating, editing, and analyzing objects in a scene. The intended audience for the toolkit is biologists, anatomists, and research physicians. This work is leading to a new product line at Kitware. Look forward to this project being featured in future issues of the *Source*.

KITWARE GUIDES STUDENT PROJECT AT UNC

CS 523 is a course on software engineering offered by The University of North Carolina. The course is designed to teach undergraduate engineering students the skills necessary for building a software product as a team. Projects are assigned real clients who also double as mentors for student development teams. Kitware's Patrick Reynolds was selected as a client for a development team comprised of UNC's Meg Sorber, Zach Mullen and Sam Brice after delivering a 5 minute sales presentation to their class.

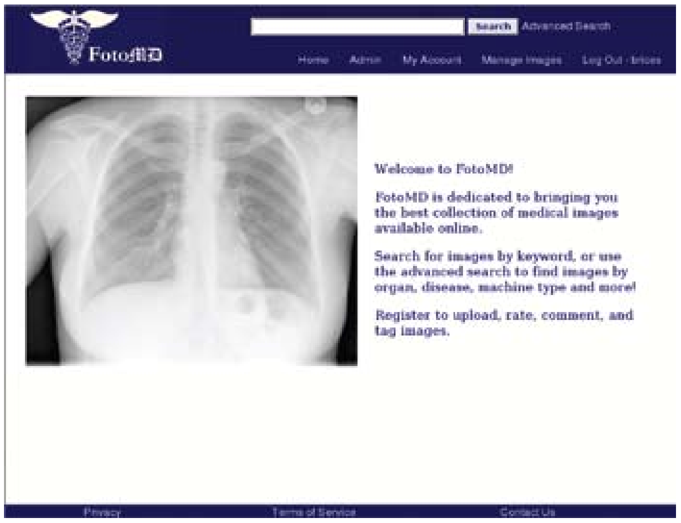
Reynolds met weekly with the team of student programmers as they experienced the challenges and rewards associated with moving a software project from early concepts to specification, implementation, testing and delivery. Through his involvement with the project Reynolds acted as an educator, consultant and customer guiding the student team in their selection of software and meeting their development objectives. Using popular photo sharing, blog, and consumer review websites for inspiration the team chose to create an innovative web-based tool for sharing and annotating medical images called FotoMD.

Developed in PHP and perfectly suited for a LAMP setup, FotoMD enables users to archive, browse, view, comment on and rate 2D and 3D medical images. It also brings many of the capabilities of desktop 3D image viewing to the web so users can view 3D images without having to pay for expensive software or download multi-gigabyte files. Practically, the FotoMD application could help researchers who can post images of their results, linking those results to published articles so others can look at them. Doctors could then look at relevant images and, perhaps, help formulate a diagnosis.

"This project has definitely provided us with skills that we can use in the real world. I personally learned HTML, CSS, some flash, PHP/web browser communication, JAVA applets, and applet/server communication. Sam learned C++, VTK and ITK. I think Zach already knew most the PHP backend and database part, but practice never hurts," said team member Meg Sorber.

"The experience as a whole was great," said Reynolds "They brought some fresh ideas that I've used in my work at Kitware. I would definitely do this again and plan to." Reynolds himself took CS 523 last year directly before his own graduation wherein he developed a project Sami Says that allows blind children to compose stories by recording their own voice, insert sound effects.

To view the FotoMD project page head to: http://www.assembla.com/wiki/show/FotoMD. For more information on Reynolds' own Sami Says project visit: http://code.google.com/p/samisays.

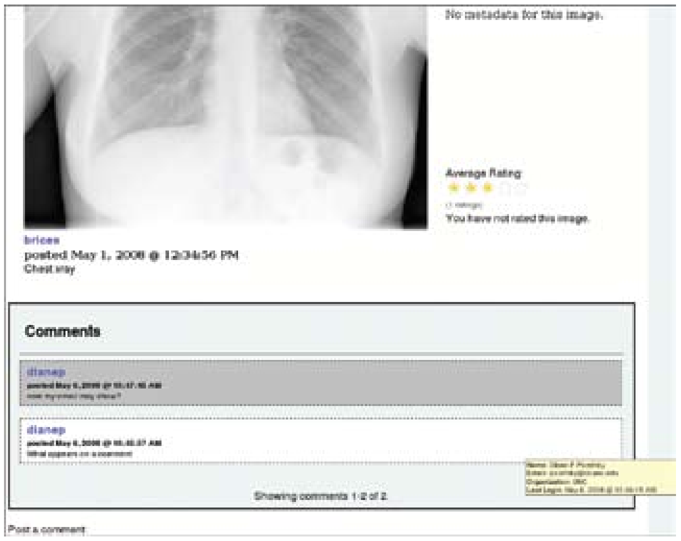


Welcome to FotoMD!

FotoMD is dedicated to bringing you the best collection of medical images available online.

Search for images by keyword, or use the advanced search to find images by organ, disease, machine type and more!

Register to upload, rate, comment, and tag images.



No metadata for this image.

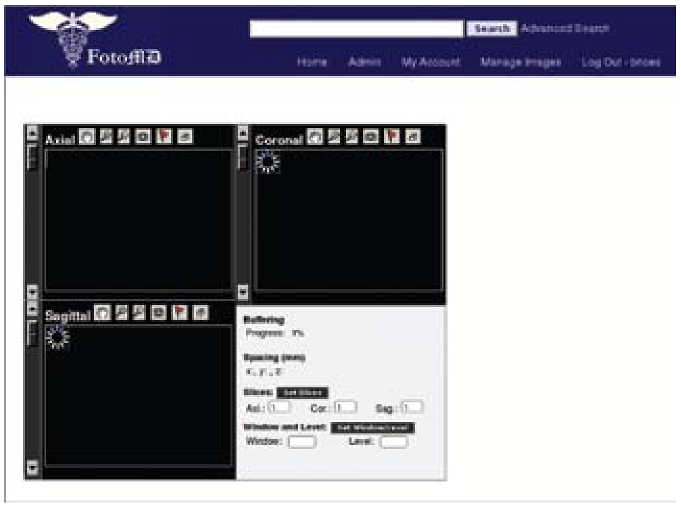
You have not rated this image.

brices
posted May 1, 2008 @ 12:34:56 PM
Chest xray

dianep

dianep

Showing comments 1-2 of 2

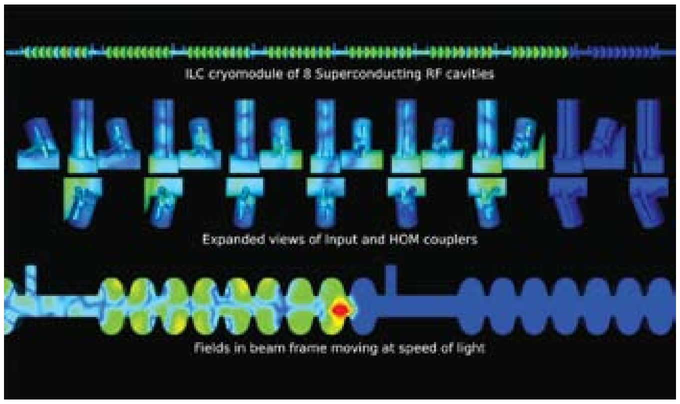


AIR FORCE PHASE I STTR

Kitware has been awarded a Phase I STTR from the Air Force. Teaming with Mississippi State University (David Thompson) and Ohio State University (Raghu Machiraju), Kitware will deploy leading-edge feature detection algorithms based on spectral analysis, wavelets, vortex identification, and topological theory in order to extract important flow features from large, computation fluid dynamics simulations. This technology will be deployed to the ParaView application.

DOE PHASE I SBIR

Kitware has been awarded a Phase I SBIR from the Department of Energy. Teaming with the Stanford Linear Accelerator Center (a DOE Office of Science National Lab), the proposed work aims to develop large-data, collaborative visualization tools. The near-term goal is to deploy this technology in ParaView, with a longer term commercialization strategy to offer a ParaView Professional application, as well as extending the ParaView Enterprise Edition web-based visualization system.


ILC cryomodule of 8 Superconducting RF cavities
Expanded views of input and HOM couplers
Fields in beam frame moving at speed of light

A snapshot of wakefield generated by an electron bunch traversing the ILC cryomodule which consists of 8 superconducting rf cavities.

VTKEDGE

Kitware is pleased to announce the creation of a new open source toolkit: VTKEdge. This new toolkit will allow Kitware to make the results of some of our Small Business Innovation Research (SBIR) and Small Businesses Technology Transfer (STTR) endeavors available to the open source community.

SBIR / STTR programs require that the small business commercialize the results of the project and the commercialization success of past projects is used to determine eligibility for future awards. In the past, Kitware has had to choose between contributing our results to our existing open source projects which negatively impacts our commercialization history or keeping the results proprietary and available only through a commercial license. Creating VTKEdge using GPLv3 terms for licensing allows us to make the technology available to the open source community while still retaining the ability to license the technology to customers who wish to avoid the reciprocal terms of the GPL.

VTKEdge is a C++ class library that can be built alongside VTK to provide additional functionality. The first version of VTKEdge will be released in July 2008 and will include GPU-based volumetric ray casting, specialized painters for illuminated lines and surface LIC rendering, GPU-accelerated filters for computing 2D LIC on vtkImageData or vtkStructuredGrid, CPU and GPU versions of Feldkamp back projection, several GPU-accelerated image processing algorithms, and various helper classes, readers and writers. VTKEdge will become a regular feature of the *Source*, with details on the GPU-based ray caster provided in this edition. Please contact us at kitware@kitware.com for more details on this new toolkit.

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.

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)

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

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

Contributors: Lisa Avila, Stephen Aylward, Utkarsh Ayachit, François Bertel, David Cole, Berk Geveci, Bill Hoffman, Luis Ibàñez, Bill Lorensen, Steve Pieper, Patrick Reynolds, Will Schroeder, Simon Warfield, and Ross Whitaker.

Design: Melissa Kingman, www.elevationda.com

Editor: Niki Russell

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