Kitware is pleased to present this year's Insight Journal special edition of the Source which features several of the strongest Insight Journal submissions from 2011. The Insight Journal was designed to provide a realistic support system for disseminating scientific research in the medical image processing domain. Recognizing the need for a mechanism whereby the medical image analysis community can collectively share their ideas, code, data and results, Dr. Luis Ibáñez has worked with the ISC and championed the open-science cause to make the Insight Journal a reality.

By providing a platform for open-access scientific publication, Kitware continues to foster its commitment to the open-science community. To continue celebrating this cause we annually publish this special Insight Journal Edition of the Source. Anyone may submit their work to the Insight Journal by registering, for free, at insight-journal.org; perhaps your work will be featured here next year!

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.

INTRODUCING VISOMICS
Kitware is pleased to announce Visomics, an exciting new open-source tool for the exploration of biological "omics" data with a focus on genomics, proteomics, transcriptomics, and metabolomics. Visomics is an integrated, end-to-end bioinformatics tool that incorporates data ingestion, analysis and visualization tasks for a variety of data types and sizes. As a result, Visomics provides researchers with powerful statistical methods, intuitive visualizations, and pre-existing chemical pathway diagrams necessary for the analysis and evaluation of relationships and behaviors among large numbers of analytes and biomarkers.

A collaborative effort between Kitware, Inc., the Hamner Institute for Health Sciences, the University of Washington, and the University of North Carolina at Chapel Hill, Visomics is a flexible, user-friendly tool that can be used by developers and researchers without requiring expert knowledge.

Clinical researchers are already using Visomics to perform deep analysis in research areas including reproductive and perinatal health, cardiovascular disorders, and breast cancer with the goal of gaining new insight into disease pathogenesis and developing preventative, diagnostic, and therapeutic treatment strategies.

Through facilitation of data analysis and mining activities for hypothesis generation and testing, Visomics aims to enable more widespread use of omics data and ultimately lead to improved, personalized medicine. To learn more about Visomics, visit www.visomics.org.
ITK 4.0

ITK 4.0 was released on December 20th, 2011 after 18 months of refactoring efforts, generously sponsored by the National Library of Medicine (NLM) with funds provided by the American Recovery and Reinvestment Act (ARRA).

The main features of this new release include Apache 2.0 licensing, removal of the patented directory, code modularization, a new simplified layer “SimpleITK”, new frameworks for registration and level sets, and a newly-refactored FEM framework. There is also improved support for video processing including bridges to OpenCV and VXL; and updated support for DICOM-based on GDCM 2.0, and large microscopy images.

A general code cleanup of the toolkit took place with a focus on supporting modern C++ compilers and removing code intended for obsolete ones.

**Modularization** - ITK’s source tree was restructured into modules with limited dependencies among them. The main goals of the modularization are to facilitate the management of further growth of the toolkit; improve the software quality by facilitating local computation of quantitative measures of quality, such as code coverage; facilitate the addition of modules from community contributions; better insulation from third party libraries; improved management of optional components, packaging, and redistributions of subsections of ITK.

**SimpleITK** - A simplified layer, SimpleITK, was built on top of ITK to ease the use of ITK in rapid prototyping, education, and interpreted languages. SimpleITK is an easy-to-use, object-oriented tool that features a C++ library and binary distributions for Python and Java. Its procedural interface hides the ITK C++ templates from its API. SimpleITK is open-source software distributed under the Apache 2.0 license.

Additionally, due to the absence of C++ templates in its API, it is now possible to use SWIG to wrap SimpleITK for many different languages. Current support is available for Python, Tcl, Java, R, and Lua.

**Registration Framework** - ITK’s new registration framework was contributed by the PICSL Laboratory of the University of Pennsylvania. The new image framework, largely based on the features of the ANTS tool developed at PICSL, provides better support for dense deformation fields including a new transform that encapsulates a dense deformation field; composite transforms (grouping multiple transforms into a single one); symmetric registration (where the fixed and moving images make unbiased contributions to the registration); multi-threaded metrics, a new Demons metric, point set registration methods, a new mutual information metric, diffeomorphic deformable registration, and parameter estimation for transform parameter scaling.

**Level Sets Framework** - A new level sets framework was contributed by the Megason Lab at Harvard Medical School. The main features of the new framework are implicit level sets, the flexible addition of level set terms via a new term container, support for large numbers of level sets running simultaneously, the introduction of geometrical constraints, region-based level sets, new support for dense and sparse level set representations, and representations for Whitaker, Shi, Malcom, as well as, Chan and Vese.

**FEM Framework** - The Finite Elements Methods (FEM) framework was refactored by a group at the University of Iowa. This refactoring made the framework more consistent with the rest of the toolkit. It also made it possible to deploy a fully-functional FEM infrastructure that can be used to solve actual mechanical and modeling problems. The code changes included the removal of incompatible smart pointers, improved encapsulation, removal of IO functionalities from filters, the addition of SpatialObject and IO support for FEM and related elements. These changes were accompanied by corresponding modifications in the FEM deformable registration framework.

**DICOM** - Support for DICOM files was improved in the toolkit. In particular, GDCM was upgraded to version 2.0 and support for RT-Structs was added.

**Statistics Framework** - The statistics framework that was refactored around 2007 has now been made the official framework. The main changes include conversion of calculators into filters, making it so samples are not DataObjects, and the addition of better traits and types, especially for 64bit machines and large samples.

**C++ Modernization** - Many changes were made across the toolkit to take advantage of modern C++ features. This involved the elimination of further support for several old compilers, including Borland 5.5, Visual Studio 6.0 and 7.0, Sun CC before version 5.9, SGI compilers, MetroWerks, GCC previous to 3.4, and the entire Cygwin platform. Support for 64 bits was improved across the toolkit, particularly for the Windows platform.

Filename extensions of templated implementation files (.txx) were replaced with (.hxx) in order to be more consistent with standards of IDE tools.

**Software Process** - The Software Development process for ITK was modernized through migration from CVS to Git, adoption of the code review tool Gerrit, development of cdash@home for distributed testing of patches, and adoption of a new process for dealing with data for testing.

**Insight Journal** - The Insight Journal was also adapted to take advantage of these new software tools. As a result, it is now possible to submit contributions to the Insight Journal by pointing to existing Git repositories in which the code contribution is continuously being developed. Several features were added for encouraging participation including support for quick comments and rating.

**Doxygen Documentation** - Doxygen documentation generation has been improved to include groups and modules, wiki example links, and diagrams. An important addition is the support for crowdsourcing fixes from the community by enabling them to edit the doxygen web pages directly; such edits create Gerrit patches that are directly submitted to developers for review and subsequent merge into the codebase.

**Wrapping** - ITK wrapping was improved at two different but related levels. On one hand, WrapiTK was embraced as the standard method for wrapping the toolkit to be used from languages such as Python and Java. On the other hand, the SimpleITK layer, wrapped via SWIG, provides an alternative option for using ITK from many languages.

**WrapITK** - The infrastructure for wrapping ITK was updated to use WrapITK 3.0. Wrapping for Tcl was deprecated, and an effort was made to improve support for Python and Java.
MIDAS PLATFORM 3.2
Midas 3.2 is an extensive redesign of the platform that provides greater flexibility, robustness, and speed. This new version of Midas is built on a new PHP framework, Zend, and the core system has been completely refactored to provide better stability and greater modularity. The web interface has also been completely redesigned to provide quicker user interaction and overall a better user experience.

Some of the main features of the 3.2 release are easier installation (click and configure); support for PgSQL, MySQL and other common databases; a redesigned user interface; an improved search system; a new plugin mechanism for modularization; and an improved software process with GIT integration, coding style, and code coverage.

Some of the main features of the 3.2 release are easier installation (click and configure); support for PgSQL, MySQL and other common databases; a redesigned user interface; an improved search system; a new plugin mechanism for modularization; and an improved software process with GIT integration, coding style, and code coverage.

Midas 3.2 allows for easy upload, access, and viewing of imported data, such as videos. The sidebar shows user-added metadata for quick search and retrieval.

We encourage users to download and try the latest release and join the Midas community.


CMAKE 2.8.7 RELEASED
CMake 2.8.7 was released on December 30, 2011 and is now available for download on www.cmake.org. This release includes several notable improvements and bug fixes.

Support has been improved for Eclipse and incremental updates were made to the AUTOMOC feature, which was added in the previous release. CMake 2.8.7 also contains additional cryptographic hash implementations for file(...) and string(...) commands, including SHA1, SHA224, SHA256, SHA384, SHA512. There is also a new module, DeployQt4, and additional options, LINK_PUBLIC and LINK_PRIVATE, for the target_link_libraries command.

Like the previous release, there are not pre-built binaries for SunOS. However, there are now two sets of installers for the Mac; the "Darwin" versions are "ppc;i386" universal binaries for Mac OSX 10.4 and later. The "Darwin64" versions are "x86_64;i386" universal binaries for 10.6 and later.

A complete list of changes and bug fixes is available in “CMake 2.8.7 Available for Download” by David Cole on the Kitware blog, or can be derived from the CMake git repository. Please try this version of CMake on your projects and report any issues to the list or the bug tracker.

We present a system for filling holes in an image by copying patches from elsewhere in the image. These patches should be a good continuation of the hole boundary into the hole. There is typically no “right answer” as we are extrapolating image data into an unknown region. For this reason, human observers giving varying levels of “it looks pretty good/realistic” responses to the output image is usually the only metric used to judge the quality of such algorithms. This implementation is intended to serve as a reference implementation of a frequently cited algorithm for patch-based inpainting, as well as to provide a framework for users to easily experiment with modifications to such methods.

Several ideas implemented in this work were taken from or inspired by the algorithm described in “Object Removal by Exemplar-Based Inpainting” by Criminisi et. al.

The latest code is available here: https://github.com/daviddoria/PatchBasedInpainting

TERMINOLOGY
Concepts are too often obfuscated by complex notation, so we will refrain from using such notation as much as possible. Before proceeding, we must define some terms:

• The source region is the portion of the image that is known (is not part of the hole) at the beginning, or has been already filled.
• The target region is the hole to be filled.
• An isophote is simply an image gradient vector rotated by 90 degrees. It indicates the direction of “same-ness” rather than the direction of maximum difference, as the gradient indicates.

DEMONSTRATION
Below we show a synthetic demonstration of the algorithm. The inputs to the algorithm consist of an image and a binary mask that is the same size as the image. Non-zero pixels in the mask indicate the region of the image that is considered the hole to inpaint/complete/fill. In this example, the image consists of a black region (top) and a gray region (bottom). This simple example is used for testing because we know the result to expect - the dividing line between the black region and gray region should continue smoothly.

Figure 1 (a) Image to be filled. The region to be filled is shown in bright green. (b) The mask of the region to inpaint. (c) The result of the inpainting.

In real images, the result will never be good. The next image we show is an example of completing a real image.
This result shows the typical quality of inpainting that the algorithm produces.

Figure 2 (a) Image to be filled. The region to be filled is shown in bright green. (b) The mask of the region to inpaint. (c) The result of the inpainting. This took about 30 seconds on a P4 3GHz processor with a 206x308 image and a patch radius = 5.

ALGORITHM OVERVIEW

The algorithm reads an image and a binary mask. Non-zero pixels in the mask indicate there is a hole to fill. Set the size of the patches that will be copied. Determining a good patch size is a very experimental process. Once a patch size is decided, locate all the patches of the image that are completely inside the image and entirely in the source region.

Take note that the main loop does the following:

- Computes the priority of every pixel on the hole boundary.
- Determines the boundary pixel with the highest priority. We will call this the target pixel. The region centered at the target pixel and the size of the patches is called the target patch.
- Determines which source patch to copy into the target patch.
- Copies the corresponding portion of the source patch into the target region of the target patch.
- Updates the mask/hole to reflect the copied patch.
- Determines which image patches are newly fully valid and adds them to the list of source patches.
- Repeats until the target region consists of zero pixels.

ALGORITHM DETAILS

Two parts of the algorithm deserve much discussion:

1. How do we choose which boundary pixel has the highest priority?
2. How do we decide which source patch to copy into a specified target patch?

PRIORITY COMPUTATION

Since these patch-based methods are greedy (there are a few attempts at globally optimal, patch-based solutions including “Image Completion Using Efficient Belief Propagation via Priority Scheduling and Dynamic Pruning” by Komodakis), selecting a good order in which to fill the hole is very important. We provide several priority computation classes that work up Criminisi’s method. The base class, Priority, has a simple interface so users can implement their own priority functions for experimentation.

Random Priority

This class simply returns a random pixel on the hole boundary. This is fast, but likely not a very good choice.

Onion-Peel Priority

Filling in pieces near the edge of the hole should intuitively be easier than filling in pieces deep within the hole. This class encapsulates the idea that the outside of the hole should be preferred over boundary pixels that are now deep inside of the original hole. This technique gets its name because with a regularly shaped hole, the algorithm will chew away at the entire outside of the hole before moving further inside, an order that resembles peeling an onion. To enforce this behavior, a confidence image is maintained. Initially, the confidence outside of the whole is 1 (very sure) and the confidence inside of the hole is 0 (totally unsure). You can think of confidence as a measure of the amount of reliable information surrounding the pixel. In Criminisi’s method, the confidence of a pixel is defined as:

\[ C(p) = \sum \frac{\text{Confidences of the patch pixels in source region}}{\text{Area of the patch}} \]

When a patch is filled, Criminisi updates all pixels in the hole region of the target patch in the confidence image with the confidence value of the target pixel.

Criminisi Priority

Criminisi noted that continuing/filling linear structures first is very important in making the result look believable. Therefore, a data term is computed as:

\[ D(p) = \frac{\text{dot(isophote, boundary normal)}}{\alpha} \]

This function encourages first filling target pixels that have strong isophotes in a similar direction to the hole boundary normal.

The priority \( P(p) \) of a pixel \( p \) is then given by the product:

\[ P(p) = C(p)D(p) \]

Alpha is a normalization factor that should be set to 255 for grayscale images, but that value also seems to work well for RGB images. In fact, in Criminisi’s priority term, alpha is a scalar multiple of a term that is only used once minimized; the value of alpha is actually irrelevant. No initialization is necessary to compute this priority term because it is not recursive. That is, it can be computed from the image+hole information directly at each iteration.

CHOOSING THE BEST SOURCE PATCH

Once a target pixel is selected, we must find the “best” source patch to copy to its location. Criminisi proposed comparison of a source and target patch by computing the normalized sum of squared differences between every pixel that is in the source region of the target patch and the corresponding pixels in each source patch.

IMPLEMENTATION DETAILS

There is a caveat that should be noted in the computation of the isophotes. We originally tried to compute the isophotes using the following procedure:

- Convert the RGB image to a grayscale image.
- Blur the grayscale image.
- Compute the gradient using itkGradientImageFilter.
• Rotate the resulting vectors by 90 degrees.
• Keep only the values in the source region.

The figure below shows the gradient magnitude with and without the modification, which we will explain.

![Gradient Magnitude Examples](image)

**Figure 3 (a) The image to be filled. The target region is shown in green. (b) The naively computed gradient magnitude. (c) The gradient magnitude of the image with the slightly expanded hole.**

The high values of the gradient magnitude surrounding the target region are very troublesome. The resulting gradient magnitude image using this technique is sensitive to the choice of the pixel values in the target region, which we actually want to be a completely arbitrary choice (as it is unknown, it should not affect anything). More importantly, the gradient plays a large part in the computation of the pixel priorities, and this computation is greatly disturbed by these erroneous values. Simply ignoring these boundary isophotes is not an option because the isophotes on the boundary are exactly the ones that are used in the computation of the Data term. To fix this problem, we have discovered two solutions.

Immediately dilate the mask specified by the user. This allows us to compute the isophotes as described above (naively), but now we have image information on both sides of the hole boundary, leading to a valid gradient everywhere we need it to be. The only drawback to this approach is that we have made the problem a bit harder by making the hole slightly bigger.

As an alternative solution, use a non-patch based technique to coarsely fill the hole (e.g. Poisson editing/hole filling - see the following Source article). This solution is very blurry deep inside of a large hole, but is reasonable at the hole boundary. This procedure produces values inside the boundary of the hole that when used in a gradient computation yield a gradient image that is very close to what we would expect.

**CODE SNIPPET**

The PatchBasedInpainting class must be instantiated using the type of image to be inpainted. Then the patch radius must be set, the image and mask provided, and the Inpaint() function called.

```cpp
PatchBasedInpainting<ImageType> inpainting;
inpainting.SetPatchRadius(5);
inpainting.SetImage(image);
inpainting.SetMask(mask);
inpainting.Initialize();
inpainting.Inpaint();

ImageType::Pointer result = inpainting.GetCurrentOutputImage();
```

A full demonstration is provided in the file: `PatchBasedInpaintingNonInteractive.cpp`.

**GUI**

Debugging an algorithm like this can be very hard. Even when things are working correctly, inspecting each step can lead to valuable insights. We have created a Qt GUI to enable this type of inspection. Its features include:

• All of the important images can be toggled on/off including the image, the mask, the boundary, and the priority.
• The inpainting can be advanced automatically or one step at a time.
• The top N matching source patches at each iteration are displayed in a table along with the scores.
• The locations of the target patch and top N source patches can be overlayed on the image.

A screenshot of the GUI is shown below:

**CONCLUSION**

In this article we described a common algorithm for filling image holes in a patch-based fashion. We also provided and detailed an implementation of such an algorithm that is written in such a way to hopefully promote future experimentation and research in this area.

**David Doria** is a Ph.D. student in Electrical Engineering at RPI. He received his B.S. in EE in 2007 and his MS in EE in 2008, both from RPI. David is currently working on hole filling in LiDAR data. He is passionate about reducing the barrier of entry into image and 3D data processing. Visit [http://daviddoria.com](http://daviddoria.com) to find out more or email him at daviddoria@gmail.com.

**POISSON EDITING IN ITK**

Two very related problems in image processing are hole filling and compositing. This article explains an implementation of similar algorithms to solve both of these problems - poisson editing and poisson cloning; both algorithms were presented by Perez in “Poisson Image Editing”.

**POISSON EDITING/HOLE FILLING**

First, we address the problem of image hole filling. The goal is to attempt to fill a missing region in an image so that the result looks plausible. The region can either actually be missing (in the case of a damaged photograph, or missing data in a digital file) or synthetically removed. In this example, our goal is to remove the jet from the following image, $I$:
To do this, we must manually specify the region to be removed with a hole mask, \( H \):

\[
\nabla^2 H = 0
\]

while ensuring \( H = I \) on the boundary of the hole. This setup is a Laplace equation with Dirichlet (also known as first order) boundary conditions.

**THEORETICAL SOLUTION**

It has been shown, using calculus of variations, that the best solution for the region inside the hole, \( H \), is given by the solution to:

\[
\text{while ensuring } H = I \text{ on the boundary of the hole. This setup is a Laplace equation with Dirichlet (also known as first order) boundary conditions.}
\]

**DISCRETE SOLUTION**

The continuous Laplace operator is given by:

\[
\nabla^2 f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}
\]

A common discrete approximation to this function at a pixel \((x,y)\) is given by:

\[
\nabla^2 f(x, y) \approx f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4f(x, y)
\]

Another way to write this is by multiplying the pixel and its neighbors by a kernel \( K \):

\[
K = \begin{pmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{pmatrix}
\]

From there, for each unknown pixel \( u_{ij} \), the equation is:

\[
-4u_{ij} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} = 0
\]

To solve the Laplace equation over the entire image, we can write a linear system of equations. Create a variable for every pixel to be filled; if the pixels are vector-valued (e.g. in an RGB image, the pixels are 3-vectors), \( N \) of these systems must be solved independently (where \( N \) is the dimensionality of the pixels) and stacked to produce the final result.

To solve the system of equations, a matrix \( U \) is constructed row-by-row, one row per variable. \( U \) is incredibly sparse, so a sparse solver should definitely be used. In each row, the value in the Laplacian kernel corresponding to the pixel's location, relative to the current variable pixel, is placed in the column corresponding to the variable ID. When one of the pixels is on the border of the hole (and is therefore known), \( u(\cdot, \cdot) \) is replaced with \( p(\cdot, \cdot) \), the value of the pixel from the original image. In this case, rather than place the value of the Laplacian kernel in the \( U \) matrix, we instead multiply it by the image value and subtract the result from the corresponding entry of the \( b \) vector. That is, we move the known value to the right side of the equation.

A vector, \( H_v \), the vectorized version of the solution to the set of hole pixels, is created as the unknown vector to be solved in a system of equations.

The linear system is then:

\[
U^T H_v = b
\]

After solving for \( H_v \), the resulting \( H_v \) is then remapped back to the pixels corresponding to each variable ID to construct \( H \).
In this implementation, the UMFPACK interface provided by Eigen3 is used to perform the computations. On a single channel image, approximately 500x500 in size, the hole filling procedure takes only a few seconds.

**CODE SNIPPET**

We have packaged this implementation in a class called `PoissonEditing`. Using this class is very straightforward. The image and mask variables must be set and the `FillMaskedRegion()` function does the work. If the image consists of vector-valued pixels, it is internally decomposed, filled, recombined, and returned by the `GetOutput()` function.

```cpp
PoissonEditing<ImageType> editing;
editing.SetImage(image);
editing.SetMask(mask);
editing.SetGuidanceFieldToZero();
editing.FillMaskedRegion();
ImageType* outputImage = editing.GetOutput();
```

**GUI**

A Qt GUI is provided to allow users to easily load an image/mask pair, fill the hole, and inspect the results.

**REGION COPYING**

In the problem of region copying (a.k.a. seamless cloning or compositing), we are interested in copying a region from one image into another image in a visually pleasing way. Again, this is best motivated with an example. In our example, the goal is to copy the jet from the Poisson Editing example into the image of the canyon shown below:

The final result is shown below:

```cpp
PoissonCloning<ImageType> cloning;
cloning.SetImage(sourceImage);
cloning.SetTargetImage(targetImage);
cloning.SetMask(mask);
cloning.PasteMaskedRegionIntoTargetImage();
ImageType::Pointer outputImage = cloning.GetOutput();
```

**GUI**

A Qt GUI is in the works to allow users to easily load an image/mask pair, fill the hole, and inspect the results.

**IMAGE MANIPULATION IN GRADIENT DOMAIN**

There are many image processing techniques that involve manipulating the gradient or Laplacian of an image. After these manipulations, it is usually necessary to return the result to the original image domain. While these operations are not necessarily related to image hole filling or cloning, we have already presented the framework needed to solve this problem, so we will explain how to use it.

**CONCEPT**

In Perez’s original paper, it was argued that a good way of copying a region from one image into another image is to do something very similar to hole filling, but with the additional introduction of a “guidance field,” $G$. That is, the way to copy a region from one image into another is by solving the equation:

$$
\nabla^2 H = \text{div}(G)
$$

The boundary condition is again first order and specifies that the resulting $H$ be equal to the target image, $T$, at the hole boundary.

The suggested guidance field $G$ is the gradient of the source image. That is:

$$
G = \nabla S
$$

In this case, the right-hand-side of the equation has become exactly the Laplacian of $S$.

**DISCRETE SOLUTION**

Just as before, the discrete Laplacian equation is written for each pixel, but this time the right-hand-side is set to the Laplacian of $S$ at $(i,j)$, rather than 0. When the right side of this equation is non-zero, it is referred to as a Poisson equation rather than a Laplace equation. The equation for each pixel is now:

$$
-4u_{i,j} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} \approx \nabla^2 S(i,j)
$$

The linear system:

$$
U^T H = b
$$

is created and solved identically as before.

**CODE SNIPPET**

We have implemented this functionality in a class called `PoissonCloning`. Using this class is very straightforward. In fact, `PoissonCloning` derives from `PoissonEditing` and the only addition is that `PoissonCloning` provides a `SetTargetImage` function. The rest of the functionality is identical.

```cpp
PoissonCloning<ImageType> cloning;
cloning.SetImage(sourceImage);
cloning.SetTargetImage(targetImage);
cloning.SetMask(mask);
cloning.PasteMaskedRegionIntoTargetImage();
ImageType::Pointer outputImage = cloning.GetOutput();
```
To return to the image domain from the gradient domain, one must find the least squares solution to a system of equations involving the derivatives of the image; however, this technique is not often used. More commonly, the derivatives are first combined into the Laplacian. We will describe the procedure for both methods below.

**RECONSTRUCTING IMAGES FROM DERIVATIVES**

For the task of reconstructing an image directly from its derivatives, we can use a method that is very similar to solving the Laplace equation. This time, however, there are two equations to be solved simultaneously:

\[
\begin{align*}
\frac{d}{dx}(U) &= D_x \\
\frac{d}{dy}(U) &= D_y
\end{align*}
\]

where \(D_x\) and \(D_y\) are the known derivative images. Of course, the first order boundary conditions must be known. This time the boundary is the actual border of the image. To relate this back to the Poisson concepts, what we are doing is using the entire inside of the image as the “hole,” the image gradients as the guidance field, and the image boundary as the hole boundary.

We can construct the same type of linear system to solve for the components of \(U\) that best satisfy both equations. Any discrete derivative operator can be used. We have chosen to use the Sobel operators:

\[
S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}
\]

\[
S_y = \begin{pmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}
\]

By applying these operators to every pixel in the unknown image \(U\), we can write a system of equations, two for each pixel:

\[
\begin{align*}
-u_{i-1,j-1} - 2u_{i-1,j} - u_{i-1,j+1} + u_{i+1,j-1} + 2u_{i+1,j} + u_{i+1,j+1} &= D_x(i,j) \\
u_{i-1,j-1} - 2u_{i-1,j} - u_{i-1,j+1} + u_{i+1,j-1} + 2u_{i+1,j} + u_{i+1,j+1} &= D_y(i,j)
\end{align*}
\]

Again, we simply place the coefficient of each term in the column of the matrix that corresponds to the variable ID of the pixel. As usual, we move the term to the right side of the equation (to contribute to the \(b\) vector) if the pixel is known. Below we show an image, its derivatives, and the image reconstructed using only the border of the image in and the derivatives.

![Original Image](a) ![Derivative Image 
\(D_x\)](b) ![Derivative Image 
\(D_y\)](c) ![Resulting Image](d)

We have provided this implementation in a file called DerivativesToImage.cxx.

**RECONSTRUCTING IMAGES FROM LAPLACIANS**

The technique for reconstructing an image from its Laplacian is even more similar to Poisson Cloning than the reconstruction from derivatives. We specify the Laplacian in the non-boundary region as the guidance field. We’ve provided this implementation in a file called LaplacianToImage.cxx.

**CONCLUSION**

We have presented the concept of Poisson image editing as well as described an implementation using ITK and Eigen3. The intention is to serve as a reference implementation, as well as a springboard for future research in these areas.

David Doria is a Ph.D. student in Electrical Engineering at RPI. He received his B.S. in EE in 2007 and his MS in EE in 2008, both from RPI. David is currently working on hole filling in LiDAR data. He is passionate about reducing the barrier of entry into image and 3D data processing. Visit [http://daviddoria.com](http://daviddoria.com) to find out more or email him at daviddoria@gmail.com.

---

**SIMULATED DIFFUSION-WEIGHTED IMAGING**

**ABSTRACT**

A recent article by Van Hecke et al. [3] describes a framework for creating simulated diffusion-weighted images (DWI) for algorithmic evaluation purposes. This framework permits the modeling of intersubject variability, regional pathology, and noise. This article describes our ITK implementation of this work, which we provide to the community as open source.

**INTRODUCTION**

The popularity of diffusion tensor imaging (DTI) has resulted in a proliferation of corresponding analysis algorithms. The need for an evaluative framework for such algorithms motivated the developments of Van Hecke and colleagues in a recent paper [3] that encompasses modeling intersubject variability, regional pathology, and noise in creating tailored DWI from an existing DTI cohort. It should be noted that where Van Hecke’s paper also describes the registration processes to both align the DTI cohort to the common reference space and back to the individual subject space, we omit that part of the framework in this submission and allow the user to select their preferred method of image normalization. In addition, the user will need to reconstruct the resulting DTI from the simulated DWI, which can easily be performed with several publicly-available packages such as Camino [2].

**OVERVIEW OF SIMULATED DWI CREATION**

In addition to the modeling described above, the DWI simulation code also allows for the creation of a control group and an experimental group consisting of an arbitrary number of directional image sets, which can then be fitted to create individual DTI.

Intersubject variability is modeled using an eigendecomposition of the aligned sample DTI cohort. Specifically, for each aligned DTI, the first (i.e. longitudinal) eigenvalue and the average of the second and third (i.e. transverse) eigenvalues of each voxel within a specified brain mask are concatenated.
within a single vector. The matrix constituting the grouping of image vectors of eigenvalues is then decomposed to create the projections describing the variation found within the aligned cohort. These projections are used to add random intersubject variations to each of the control and experimental subject images.

Based on recent literature, pathology is introduced into the cohort using a simultaneous decrease in the longitudinal eigenvalues and an increase in the transverse eigenvalues. In propagating the change in transverse diffusion, we enforce that the ratio of the second to the third eigenvalues be the same before and after a transverse diffusion decrease.

Finally, the application of standard Rician noise is included as a possibility for enhancing the realism of the simulated DWI data. For further details, we encourage the reader to consult Van Hecke's paper on which this work is based.

**IMPLEMENTATION**

This work was originally developed within our Advanced Normalization Tools software package and uses our ANTs command line classes. After compilation, one can invoke the help menu (by typing “CreateDTICohort --help” at the command line), which will list the following set of options and corresponding descriptions:

- **-a, --dti-atlas inputDTIAtlasFileName**

  A diffusion tensor atlas image is necessary input for creating the cohort.

- **-x, --label-mask-image maskImageFileName lowerThresholdValue**

  A mask image can be specified to determine the region(s) to which the simulated pathology operations are applied. See also the option ‘--pathology’. If no mask is specified, one is created by thresholding the atlas FA map at 0.2 unless a lower threshold is specified.

- **-n, --noise-sigma <noiseSigma=18>**


- **-p, --pathology label{<percentageChangeEig1>0.05>, <percentageChangeAvgEig2&3>0.05>, <numberOfVoxels=all or percentageOfVoxels >}**

  Users can specify the simulated pathology in a given area using a label mask. If no label is prepended to parameters, the specified parameters are applied to all labels. Pathology is simulated by changing the eigenvalues. This typically involves a decrease in the largest eigenvalue and an increase in the average of the remaining eigenvalues. Change is specified as a percentage of the current eigenvalues. However, care is taken to ensure that diffusion direction does not change. Additionally, one can specify the number of voxels affected in each region, or specify the percentage of voxels affected. The default is to change all voxels. Note that the percentages must be specified in the range [0,1]. For dimension=3, where the average transverse diffusion eigenvalues are altered, this change is propagated to the distinct eigenvalues by forcing the ratio to be the same before the change.

- **-w, --dwi-parameters [B0Image,directionFile,bvalue] [B0Image,schemeFile]**

  This option specifies the parameters of the output diffusion-weighted images including the directions and b-values. The directions are specified using a direction file which has as its first line the number of directions. Each successive three lines contains the x, y, and z directions, respectively, and a single b-value. Note that several direction files of this format are distributed with the Camino DTI toolkit. Alternatively, one can specify a scheme file where each direction is specified, followed by a b-value for that direction, i.e. <x1> <y1> <z1> <bvalue1> ... <xN><yN><zN><bvalueN>.

- **-r, --registered-population textFileWithFileNames.txt**

  If one wants to introduce inter-subject variability, a registered DTI population to the DTI atlas is required. This variability is modeled by a PCA decomposition on a combination of the first eigenvalue image and the average of the second and third eigenvalues. The registered image file names are specified using a text file where each line is the name of an individual DTI.

- **-o, --output [outputDirectory , fileNameSeriesRootName , <numberOfControls=10>, <numberOfExperiments=10>]**

  The output consists of a set of diffusion-weighted images for each subject. Each file name is prepended with the word ‘Control’ or ‘Experimental’. The number of control and experimental subjects can be also be specified on the command line. The default is 10 for each group.

- **-h, --help**

  Print short/long help menus.

---

**SIMULATED DWI EXAMPLE**

We have included a subsampled and aligned dataset derived from the NKI/Rockland INDI sample, (b) the FA image created from the DTI average, and (c) the label image that can be used to apply anatomically specific pathology.

---

Figure 1 Axial slices of the included images: (a) the B0 image created by averaging the individual B0 images of the NKI/Rockland INDI sample, (b) the FA image created from the DTI average, and (c) the label image that can be used to apply anatomically specific pathology.
Please see the full paper for a full example of a terminal output from a sample call to the routine CreateDTICohort contained in the script createDWI.sh, which is also included.

REFERENCES


ACKNOWLEDGMENTS:

Additional authors include Philip A. Cook, Brian B. Avants, and James R. Stone.

Nicholas J. Tustison recently transferred to the University of Virginia, where he continues to extol the virtues of ITK and open source to the heathen hoi polloi. Currently, he’s navigating the pernicious waters of methodological circularity in neuroscience in the hopes that any breakthroughs land him a prime spread in Oprah’s magazine (or at least a free burrito from Chipotle).

CREATING A 2D ACTIVE SHAPE MODEL IN ITK

ITK is an excellent example of an open-source framework in extreme programming. Strengths of the toolkit include superb documentation and list serve support, and a supportive and vibrant programming community. One missing area of documentation in the ITK software guide includes the implementation of itk::GeodesicActiveContourShapePriorLevelSetImageFilter. The generation of the active shape model (ASM) in this example is provided with no example code or images. Here, we provide sample images and commented code to generate an ASM using itk::ImagePCAShapeModelEstimator. We use the challenging problem of segmenting the femoral condyle cartilage of the knee in our example.

Many classic described segmentation algorithms such as active contours (snakes), level set, and watershed rely on edge-based criteria [1-4]. For segmentation of articular femoral knee cartilage, a solely edge-based algorithm is inadequate because of the poor contrast at the cartilage and soft tissue interfaces. One approach is to use a priori information to create an active shape model to help guide segmentation [5].

The goal of an active shape model is to appropriately describe all of the allowed statistical variation of some generic shape. The model is formed by using a training set of objects that are already segmented either manually or by some other automated or semi-automated method. The variation among the set of these previously segmented shapes is used to describe the variation of the shape model and therefore the training set needs to be a good representation of the overall distribution of allowable shapes. This can be accomplished with a large sample size or good variation among the training images.

Principal component analysis (PCA) is used to decompose the large variation in the natural shapes of a given object into a set of variables that describe the majority of the variation. The basic steps of PCA include aligning the training images, finding the corresponding landmarks, computing the covariance matrix of these landmarks, and finally finding the determinate of the covariance matrix.

In the first step of PCA, the training images should be transformed iteratively to maximize their overlapping area. In the classical implementation, the N_i iteration consisted of the training images being registered with the first training image using a similarity transform that can rotate, scale, and translate the image. After this first iteration, the mean image is computed. For the following iterations, the training images are registered to the mean image, which is then recomputed. This process eventually converges and results in a set of training images aligned with maximal overlap. A subset of aligned training images used to create the femoral cartilage shape model are shown in Figure1.

From this updated set of transformed, registered training images, a mean model is made by taking the mean of each pixel in the training set.

$$\overline{I} = \frac{\sum_{i=1}^{N} I_i}{N}$$

In classical shape model construction, the first step of the creation of an active shape model is the identification of a set of landmarks along the outline of the shape, or in this case, femoral cartilage. Landmarks are a set of points (x,y) in the training images that describe the outline of the shapes. Each image should have the same number of landmarks and they should correspond to the same location on the general shape. The itk::ImagePCAShapeModelEstimator, however, implements automatic landmark detection by sending the images through a distance filter that changes pixel values to their distance from the border of the shape. Pixels inside the shape are given a negative distance and pixels outside the shape are given a positive distance. Each pixel location (i,j) in the set of training images corresponds to a landmark.

Each of these landmarks is then tracked on each training image. The N landmarks are represented by variables X_0, X_1, ..., X_N. Each training image is subtracted from the mean image so that the data is normalized for principal component analysis.

$$I'_i = I_i - \overline{I}$$

The nxn covariance matrix is then computed. The size of this matrix is equal to N^2 where N is the number of landmarks.

$$Cov = \begin{bmatrix} Cov_{x0x0} & \cdots & Cov_{x0x(n-1)} \\ \vdots & \ddots & \vdots \\ Cov_{x(n-1)x0} & \cdots & Cov_{x(n-1)x(n-1)} \end{bmatrix}$$
After principal component analysis is performed by finding the determinate of the covariance matrix, we have a set of eigenvalues and corresponding eigenvectors [6,7]. All of the training images can be reconstructed by combining a linear combination of the eigenvectors and the mean image. An ample number of eigenvectors are chosen such that the sum of their eigenvalues yield a large portion of variation.

This model is used with a normal active contour algorithm to help constrain the region, growing to only statistically likely representations of the shape [8]. For an implementation of this segmentation, see the ITK Software Guide’s description of GeodesicActiveContoursWithShapeGuidance [9].

**Figure 1** Subsample of the training images used

### IMPLEMENTATION

Before a set of training images can be used, they must be registered to maximize overlap, which can be done manually by centering each image over a predefined landmark or other methods. For large datasets, an automated registration method is preferable. In this example, itk::TranslationTransform based on ImageRegistration1.cxx in the ITK example library is used. Each training image is registered through multiple iterations.

For the first round, the fixed image was chosen to be the first image in the training set, and the remaining training images were defined as the moving images. For the nth iteration, the mean image of the registered n-1 iteration was defined as the fixed image and the registered n-1 iteration images were defined as moving images. This process will eventually result in a set of training images that are registered to each other.

After the training images have been aligned through registration, an active shape model can be built.

First, the number of training images being used and the prefix of training images’ file names are set. In this example, there are six training images names fixed1.jpg, fixed2.jpg,... fixed6.jpg. The number of principal components is also set.

```cpp
int NUMTRAININGIMAGES = 6;
int NUMPC = 3;
std::string IMAGENAME = "fixed";
```

The input to the PCAShapeModelEstimator is expected to be signed distance images. The signed distance image is the distance to the edge of the shape. A negative distance is given for pixels inside the shape and a positive distance is given for pixels outside the shape.

```cpp
typedef itk::SignedDanielssonDistanceMapImageFilter < OutputImageType,MapOutputImageType> FilterType;
```

Finally, for the ImagePCAShapeModelEstimator a typedef is created and a smart pointer instance of this filter is generated. The number of training images and principal components are set.

```cpp
typedef itk::ImagePCAShapeModelEstimator <MapOutputImageType, InternalImageType> PCAEstimatorType;
PCAEstimatorType::Pointer model = PCAEstimatorType::New();
model->SetNumberOfTrainingImages(NUMTRAININGIMAGES);
model->SetNumberOfPrincipalComponentsRequired(NUMPC);
```

Next, each training image is first read and thresholded to ensure it is a binary image, sent through the signed danielsson distance filter, and then set as the kth input to the PCAShapeModelEstimator.

```cpp
for ( unsigned int k=0; k < NUMTRAININGIMAGES; k++ )
{
    ThresholdingFilterType::Pointer thresholder = ThresholdingFilterType::New();
    thresholder->SetLowerThreshold( 255 );
    thresholder->SetUpperThreshold ( 255 );
    thresholder->SetOutsideValue(  0  );
    thresholder->SetInsideValue(  255 );
    ReaderType::Pointer reader = ReaderType::New();
    FilterType::Pointer filter = FilterType::New();
    reader->SetFileName(trainingImageNames[k].c_str());
    thresholder->SetInput(reader->GetOutput());
    filter->SetInput(thresholder->GetOutput());
    filter->Update();
    model->SetInput(k, filter->GetOutput());
}
```

Next, update is called to invoke the estimator. When it is finished, its print method is called to output information about the filter.

```cpp
model->Update();
model->Print(std::cout);
```

The principal component and mean image are saved. The 0th output of the ImagePCAShapeModelEstimator is the mean image, and the nth principal component image is the n+1th output of the filter. The principal component images’ names are created using itkNumericSeriesFileNames.

```cpp
WriterType::Pointer writer = WriterType::New();
writer->SetFileName("meanImage.mha");
writer->SetInput(model->GetOutput(0));
writer->Update();
```

...
The eigenvalues are printed to the console.

vnl_vector<double> eigenValues(NUMPC);
eigenValues = model->GetEigenValues();
std::cout << eigenValues << std::endl;

Finally, each of the principal components are written.

for(unsigned int i = 0; i < NUMPC; i++)
{
    //normalizing the images
    DivideFilterType::Pointer divider = DivideFilterType::New();
    divider->SetInput(model->GetOutput(i+1));
    divider->SetScale(1.0/sqrt(eigenValues(i)));

    WriterType::Pointer myWriter = WriterType::New();
    myWriter->SetFileName(outputImageNames[i].c_str());
    myWriter->SetInput(model->GetOutput(i+1));
    myWriter->Update();
}

VIEWING VARIATIONS OF THE SHAPE MODEL
The variations of the shape model can be viewed to verify accuracy and to visually appreciate the source of deviation in the model. In theory, we can view any linear combination of the principal components, but in practice it is easier to view each principal component individually. The following code can be found in ViewShapeModel.cxx and is used to view any real number multiples of standard deviations of a principal component.

The mean image and principal component image are read. The MultiplyByConstantImageFilter is used to multiply the scale constant to the principal component image.

typedef itk::MultiplyByConstantImageFilter<
    InternalImageType, float, InternalImageType>
    MultType;
MultType::Pointer multFilter = MultType::New();
multFilter->SetInput(reader2->GetOutput());
multFilter->SetConstant((float)SCALE_CONSTANT);
multFilter2->SetInput(reader3->GetOutput());
multFilter2->SetConstant((float)SCALE_CONSTANT);
try
{
    multFilter->Update();
    multFilter2->Update();
} catch( itk::ExceptionObject & err )
{
    std::cout
        << "multFilter exception caught !" << std::endl;
    std::cout << err << std::endl;
}

The AddImageFilter is used to add the offset to the mean image to get the positive deviation from the mean shape.

typedef itk::AddImageFilter<
    InternalImageType, InternalImageType>
    AddFilterType;
AddFilterType::Pointer addFilter = AddFilterType::New();
addFilter->SetInput(reader1->GetOutput());
addFilter->SetInput2(multFilter->GetOutput());

The SubtractImageFilter is used to subtract the offset from the mean image to get the negative deviation from the mean shape.

typedef itk::SubtractImageFilter<
    InternalImageType, InternalImageType>
    SubFilterType;
SubFilterType::Pointer subFilter = SubFilterType::New();
subFilter->SetInput1(reader1->GetOutput());
subFilter->SetInput2(multFilter2->GetOutput());

The outputs of the filters are thresholded. A value of zero should correspond with the boundary of the shape.

typedef itk::BinaryThresholdImageFilter<
    InternalImageType, OutputImageType>
    ThresholdingFilterType;
ThresholdingFilterType::Pointer thresholder = ThresholdingFilterType::New();
thresholder->SetInput(subFilter->GetOutput());
thresholder->SetInsideValue(  255 );
thresholder->SetOutsideValue(  0  );
thresholder->SetUpperThreshold( high );
thresholder->SetLowerThreshold( low );
thresholder2->SetInput(subFilter->GetOutput());
thresholder2->SetInsideValue(  255 );
thresholder2->SetOutsideValue(  0  );
thresholder2->SetUpperThreshold( high );
thresholder2->SetLowerThreshold( low );

The output images are then written

writer2->SetInput( thresholder2->GetOutput() );
writer->SetFileName(outputfile1);
writer2->SetInput( thresholder->GetOutput() );
writer2->SetFileName(outputfile2);

RESULTS
The mean image for the femoral cartilage is shown in Figure 2. It clearly resembles the average of all the training images. The first principle component contains the majority of the variation. Using the first four principle components, about 98% of the variation of the knee cartilage can be represented [8-9].

Figure 3 shows the first four principle components influence on the mean shape and the percent of the variation they describe. The first principle component controls the shift in the mass of the cartilage from the left to the right side. The influence of the other components is less easily describable, but can be easily observed.

There are some common pitfalls in using active shape models for segmentation. One is not having an adequately sized training set. Here, our training image set contains only 6 images to demonstrate implementation of the code and not the method's accuracy. As our training image set is limited in size, this is a primitive example with decreased accuracy. In practice, an image set should contain an order of magnitude of 100 images to capture the necessary statistical variance in the shape to have acceptable segmentation accuracy.

Another pitfall is lacking significant overlap in the training images. This will manifest itself as a completely black mean image or a mean image with a smaller than expected mass. Finally, attempting to segment an object that is outside the deformability of the shape model can result in poor results; for example, attempting to segment severely degenerative cartilage with an active shape model created entirely from individuals with normal cartilage.
REFERENCES


Image-guided interventions, such as radiotherapy and minimally invasive surgery, normally require the registration of pre-operative and intra-operative images. It is common to find situations in which the pre-operative data comes in the form of a 3D dataset, such as a Computerised Axial Tomography (CAT) scan, while the intra-operative data consists of sets of multiple 2D images, such as radiographies, acquired at known locations.

The alignment process of the 3D images to the 2D ones is known as 2D-3D registration, a problem that has produced a large research effort in recent years [1]. The Insight Toolkit (ITK) is able to handle 2D-3D registrations, but with serious limitations: only one 2D image can be used per registration, multi-threading is not supported, and only one suitable interpolator class is available, which only accepts rigid transforms and completely ignores the orientation axes of the 3D images. To overcome some of these problems, we developed a new registration framework for 2D-3D registrations that accepts inputs with any number of 2D images and takes into account the orientation axes of the 3D images. The source code is free and was made available to the open source community on the Insight Journal (http://insight-journal.org/browse/publication/800). The proposed framework does not solve all of ITK’s problems, but interested readers could also consult the work of Steiniger et al, who proposed a new set of ITK classes that accept non-rigid transforms and offers support for multi-threading [2].

The proposed framework was designed to be as similar as possible to ITK, to ease its adoption by developers. The heart of any registration is an instance of the MultiImageToImageRegistrationMethod class, which like ITK’s ImageRegistrationMethod class, calculates the optimal alignment after a call to its update method. It also expects the rest of the registration components to be plugged into it in a very similar way to the one used in ITK. In fact, the moving (3D) image, transform and optimizer are connected.
using the standard SetMovingImage, SetTransform, and SetOptimizer methods respectively. As mentioned, only rigid transforms are supported, although the proposed framework supports virtually all of ITK’s single value optimizers.

The MultiImageToImageRegistrationMethod differs when setting the fixed (2D) images, regions and interpolators, as multiple instances are required instead of the single ones in the standard framework. To implement this, the MultiImageToImageRegistrationMethod offers a convenient set of methods to add the needed objects one by one. This can be made more clear using an example.

```cpp
typedef itk::Image<short,3> FixedImageType;
typedef itk::Image<short,3> MovingImageType;
typedef itk::PatchedImageCastInterpolateImageFunction<MovingImageType,double> InterpolatorType;
typedef itk::MultiImageToImageRegistrationMethod<FixedImageType, MovingImageType> RegistrationType;
const unsigned int FImgTotal = 2;
RegistrationType::Pointer registration = RegistrationType::New();
for(unsigned int f=0; f<FImgTotal; f++)
{
    FixedImageType::Pointer fixedImage;
    //do something here to read or create a fixed image
    registration->AddFixedImage(fixedImage);
    registration->AddRegion(fixedImage->GetBufferedRegion());
    InterpolatorType::Pointer interpolator;
    interpolator = InterpolatorType::New();
    //do something to set the interpolator's parameters
    registration->AddInterpolator(interpolator);
}
```

As shown in the code snippet above, the fixed images are read (or created) individually and plugged into the registration object by means of the AddFixedImage method. Each image’s region is set using the AddRegion method with each of the images’ buffered region as argument. Also, as each fixed image requires an interpolator, the latter are created after each image is plugged and are included in the registration by the AddInterpolator method.

Metric objects are connected in a different way than fixed images or interpolators. In fact, only one object is needed, despite that similarity metrics in 2D-3D registration are calculated as the sum of individual metrics between the moving image and each of the fixed ones [1]. This is made using a new class named MultiImageMetric, which internally contains all the individual metrics. This approach was taken to keep the compatibility with ITK’s optimizers, which expect a single function object, not a collection of them. The MultiImageMetric class is purely virtual, but the proposed framework includes four subclasses that implement metrics commonly found on 2D-3D registration applications: Sum of Squared Differences, Gradient Difference, Normalized Gradient Correlation, and Pattern Intensity. As an example, the code snippet below creates an instance of the Gradient Difference metric and plugs it into the registration object.

```cpp
typedef itk::Image<short,3> FixedImageType;
typedef itk::Image<short,3> MovingImageType;
typedef itk::MultiImageToImageToImageMetric<FixedImageType, MovingImageType> MultiMetricType;
MultiMetricType::Pointer multiMetric = MultiMetricType::New();
registration->SetMultiMetric(multiMetric);
```

As 2D-3D registration is an error-prone problem, likely to produce incorrect solutions if the initial misalignment between images is large, multi-resolution strategies are commonly used and these were made available by means of the MultiResolutionMultiImageToImageRegistrationMethod class. This was designed to be as similar as possible to ITK’s existing multi-resolution classes, keeping the SetNumberOfLevels and SetSchedules methods, which give control over the amount of downsampling per resolution level. Finer adjustment of the downsampling is possible using SetSchedules, which is recommended in 2D-3D registration problems. As 2D images are in fact 3D images with a single slice, setting the downsampling level on the slice direction to 1 is required on all levels. This is possible with SetSchedules, but not with SetNumberOfLevels.

The MultiResolutionMultiImageToImageRegistrationMethod also expects the user to plug downsampling filters, one per fixed image, thus the corresponding AddFixedImagePyramid method was also implemented. It’s use is quite similar to the other ‘Add’ methods, as shown in the code snippet below.

```cpp
typedef itk::MultiResolutionPyramidImageFilter<FixedImageType, FixedImagePyramidType> FixedImagePyramidType;
for(unsigned int f=0; f<FImgTotal; f++)
{
    FixedImagePyramidType::Pointer fixedPyramidFilter = FixedImagePyramidType::New();
    fixedPyramidFilter->SetNumberOfLevels(2);
    fixedPyramidFilter->SetScales(f);
    registration->AddFixedImagePyramid(fixedPyramidFilter);
}
```

The proposed framework includes one interpolator class, PatchedImageCastInterpolateImageFunction, which projects a 3D image onto a plane using the ray casting algorithm, integrating intensity values along rays that go from all voxels on the imaging plane to a defined focal point. Ray casting offers a simplified model of the X-ray formation process, ignoring effects such as scattering, and has become a widely used algorithm for generation of digitally reconstructed radiographs. The proposed class is based on ITK’s RayCastInterpolateImageFunction, which has considerable shortcomings that have prevented its widespread use. Most importantly, the orientation axes of the moving image are completely ignored, which greatly complicated the problem definition. However, not all of its problems could be addressed. Currently, the PatchedImageCastInterpolateImageFunction class only supports rigid transforms and a bilinear scheme for interpolation, which could be extended to trilinear.

Included in the framework are two sample applications, MultiImageSearch and MultiImageRegistration. MultiImageSearch samples the similarity metric in a given grid, which is useful to characterise it and design a suitable optimization strategy. The MultiImageRegistration application implements a registration algorithm with three resolution levels using the Normalized Gradient Correlation metric and the Fletcher-Reeves Polak-Ribiere optimizer. This application was tested with a realistic input, using the public dataset provided by the Imaging Sciences Institute of the University of Utrecht (http://www.isi.uu.nl/Research/Databases/GS/) comprised of CAT, Magnetic Resonance (MR), 3D radiography (3DRX), and fluoroscopy images of eight different vertebrae, complemented with a standard evaluation protocol for registration methods [3]. The aforementioned protocol defines the ground-truth transform for each image and 200 initial transforms with mean Target Registration
Error (mTRE) uniformly distributed between 0 and 20 mm. Combination of the vertebrae and transforms gives a total of 1600 registrations for evaluation of each algorithm, which are considered successful if their final mTRE is below 2 mm. The protocol also establishes calculation of the methods' capture range as the value of the initial mTRE with a percentage of successful registrations of 95%.

The MultilimageRegistration application was evaluated with the CAT and 3DRX images, as MR ones are not compatible with the existing ray-casting algorithm. For the 3DRX datasets, the mean mTRE of successful registrations was 0.5827 mm, success rate was 62.06%, and capture range was 6 mm. For the CAT data, mean mTRE was 0.3511, success rate 61.31%, and capture range was 7 mm. Comparison of these results with other algorithms evaluated with the same protocol show that MultilimageRegistration fares well [4]. Its accuracy level is good, but is robustness needs to be increased to improve its success rate and capture range.

In summary, we have proposed a new framework for 2D-3D registration, which solves many of the shortcomings of ITK: multiple fixed images can now be included in the registration and the orientation of the moving image is taken into account. The framework’s interface was kept as similar as possible to ITK, so any developer familiar with the toolkit should become accustomed to the proposed framework with little effort.

Future work should address limitations such as support for additional transforms beside rigid ones, and extra interpolation algorithms. Currently, only intensity-based ray-casting is implemented, which is only useful when a direct relation in intensity exists between the moving and fixed X-ray images, as is the case for 3DRX and CAT data but not with MR images. Finally, we would like to say that we are happy to have released our work to the open-source community and hope that many other developers find it useful for the development of new medical imaging applications.

REFERENCES


Álvaro Bertelsen is a bioengineering PhD student at the University of Navarra and researcher at the Centro de Estudios e Investigaciones Técnicas de Guipúzcoa (CEIT), Spain. He obtained his Master in Engineering Sciences degree from the Pontifical Catholic University of Chile and has worked as a Research Assistant at the Imaging Sciences Department of Imperial College in London.

This contribution includes classes for threshold estimation using the following methods: Huang[1], Intermodes and Minimum[8], IsoData[9], Li[5, 6], MaxEntropy[2], KittlerIllingworth[3], Moments[11], Yen[12], RenyiEntropy[2], Shanbhag[10], and Triangle[13].

All classes are largely derived from the AutoThresh [4] package for ImageJ. Parts of the brief outline below are taken from the presentation associated with the HistThresh[7] Matlab toolbox, which was also a source of information for the AutoThresh package. The exception is the triangle method, which was written before discovery of the AutoThresh package.

These classes have been included in ITK 4.0 and are implemented using the histogram framework.

THRESHOLDING ALGORITHMS

2.1 Huang

*itkHuangThresholdImageFilter* implements Huang’s fuzzy thresholding using Shannon’s entropy function[1]. The measure of fuzziness represents the difference between the original image and its binary version.

For a given threshold level, the fuzzy membership function for a pixel is defined by the absolute difference between the pixel gray level and the average gray level of the region to which it belongs, with a larger difference leading to a smaller membership value. The optimal threshold is the value that minimizes the fuzziness, as defined by Shannon’s entropy function, applied to the fuzzy membership functions.

2.2 Intermodes

*itkIntermodesThresholdImageFilter* implements the methods described in [8]. The histogram is iteratively smoothed until only two peaks remain. In one variant, the threshold is the midpoint of the two peaks, while in the other it is the minimum point between the peaks. The two variants are selected using the UseIntermodeOff method. This is not good for histograms with very unequal peaks.

2.3 IsoData

*itkIsoDataThresholdImageFilter* implements Ridler and Calvard’s [9] isodata method. It computes the average of voxels below and above the initial threshold. Threshold is set to the average of the two. Repeat until the threshold is larger than the average of the brightness of the two regions.

2.4 Li

*itkLiThresholdImageFilter* implements Li’s minimum cross entropy method[5, 6], which selects a threshold that minimizes the cross entropy between original and thresholded images.

2.5 MaxEntropy

*itkMaxEntropyThresholdImageFilter* implements the method described in [2], which chooses a threshold such that the entropies of distributions above and below threshold are maximized. This is one of several entropy-based approaches.

HISTOGRAM-BASED THRESHOLDING
2.6 RenyiEntropy

*itkRenyiEntropyThresholdImageFilter* is similar to MaxEntropy, but uses a different entropy measure [2].

2.7 KittlerIllingworth

*itkKittlerIllingworthThresholdImageFilter* implements the minimum error thresholding method [3]. A threshold is selected that minimizes the number of misclassifications between the two normal distributions with the given means, variances, and proportions. This method assumes a Gaussian mixture model, similar to the Otsu method.

2.8 Moments

*itkMomentsThresholdImageFilter* implements Tsai’s moment preserving approach [11], which chooses a threshold such that the binary image has the same first three moments as the grey level image.

2.9 Yen

*itkYenThresholdImageFilter* implements thresholding based on a maximum correlation criterion [12] as a more computationally efficient alternative to entropy measures.

2.10 Shanbhag

*itkShanbhagThresholdImageFilter* implements Shanbhag’s extension of the Kapur method [10], which includes a distance from the threshold in the entropy measure.

2.11 Triangle

*itkTriangleThresholdImageFilter* implements a variant of [13]. The triangle method constructs a line between the histogram peak and the farthest end of the histogram. The threshold is the point of maximum distance between the line and the histogram. This implementation uses robust (default is 1% and 99%) estimation of histogram ends.

RESULTS

The BrainProtonDensitySlice image, cropped and smoothed, is used to demonstrate the results. Smoothing has been applied to avoid gaps in the histogram and cropping has been applied to remove a peak at 0 that was caused by padding in the original. This is a slightly challenging image to threshold as there are three classes, but the two brighter ones overlap. The mode-based approaches result in Figure 2; select thresholds between the darkest (background) class and the foreground, thus merging the two foreground classes. Some of the entropy measures, shown in Figure 3, attempt to split the two foreground classes with varying degrees of success. The thresholds are shown on the histogram in Figure 4.

![Figure 1: The input image.](image1)

![Figure 2: Thresholded images.](image2)

COMMENTS, CONCLUSIONS & FURTHER WORK

Histogram-based approaches to estimating thresholds are very useful, but also can be surprisingly sensitive to changes in image characteristics. Padding images, for example, can easily add a large spike to a histogram that can cause unexpected outputs from many methods. In the example illustrated here, changing the framing of the scene could easily change the significance of the background class, leading to very different behavior. Underlying assumptions built into some methods, such as Gaussian mixture models, can become unrealistic if image acquisition conditions (e.g. lighting) change, and such changes are not necessarily obvious to the naked eye. Always use them with care and always attempt to constrain the areas to which such estimation is applied in order to minimize the chances of changes in image characteristics affecting application performance.

This contribution is a set of standard thresholding methods available in other packages. The implementation is derived from the ImageJ Java plugin. These classes have been included recently in the ITK 4 distribution via Gerrit and include a variety of tests.

Several of the articles cited describe extensions to the basic methods to support multi-level thresholding. Additional control of the histogramming parameters can also be included, as only the most basic are in the current version. Finally, masked versions of these classes are also likely to be useful. This should be relatively simple to implement using the masked histogram infrastructure. Hopefully these extensions can be implemented in the future.

USAGE

These classes can be used in a similar way to the well-established *OtsuThresholdImageFilter*. Classes are templated over input and output image types and have methods to set the output inside and outside pixel values, the number of histogram bins, *SetNumberOfHistogramBins*. Intermodes have a method to select the intermodes or minimum selection option, *SetUseIntermodes*, and the maximum number of smoothing iterations - *SetMaximumSmoothingIterations*. All thresholding classes have an associated calculator class that operates on the histogram to estimate the threshold.
**REFERENCES**


**Richard Beare** Richard works in the Stroke and Ageing Research Group at Monash University and the Developmental Imaging Group at the Murdoch Childrens Research Institute, where he is responsible for the development of automated image analysis tools for brain, cardiac and other imaging studies. He has extensive experience in theoretical and applied aspects of image analysis.

**Gaëtan Lehmann** is a biologist/software engineer at the French National Institute for Agronomical Research (INRA) where he studies the dynamic organization of the genome in early mammalian embryos. His strong open source background has led him to contribute most of his developments to ITK, including several packages in the mathematical morphology field and WrapITK.

**INSIGHT JOURNAL REACHES 500 ARTICLES**

On November 30th, the Insight Journal received its 500th paper submission, marking a major milestone for open science. Congratulations to B. Moreau and A. Gouaillard on their paper "Exact Geometrical Predicate: Point in Circle," the 500th published paper.

The Insight Journal was launched with support from the National Library of Medicine in 2005 as a fully open-access database for researchers who want to share their work with the larger medical imaging community for peer review.

Six years later, the Insight Journal now contains 500 articles representing 80 submissions per year, 819 peer reviews, and more than 2,480 registered users. In this time, the Journal has been established as a central place for open-access research and a model for supporting reproducible science. It has become an important part of events in the medical imaging community, and has been hosting open-access articles for major conferences such as MICCAI.

We are very excited and honored to see such widespread adoption of the Insight Journal and to be able to include some of those articles in this edition of the Source. We are looking forward to the next 500 articles submitted to the Insight Journal, and encourage your contributions.

**ROBOT ASSISTED PROSTATE SURGERY**

Kitware has been awarded Phase I SBIR funding from the National Institute of Biomedical Imaging and Bioengineering, a unique joint agency initiative of the National Institutes of Health (NIH), Defense Advanced Research Projects Agency
(DARPA), and the National Science Foundation (NSF). The funding will be used to develop an augmented reality visualization prototype for the improving the efficacy and safety of robot-assisted prostate surgery.

Andinet Enquobahrie is the Principal Investigator for this project and will oversee the development of a human/machine collaborative system that creates an augmented reality (AR) view for surgeons. This system will fuse pre-operative MRI models of the prostate, tumor, and surrounding tissues with the da Vinci system laparoscopic video, while compensating for non-rigid prostate deformation with intra-operative transrectal ultrasound imaging.

The project is a collaboration between Kitware, the Center for Computer Integrated Surgical Systems and Technology (CISST ERC) at Johns Hopkins University, and Blue Torch Medical Technologies. The development of this innovative navigation system will address some of the most pressing challenges surgeons face, resulting in safer, more effective surgery and improved prognoses for the quarter-of-a-million men in the United States who will be diagnosed with prostate cancer in 2011.

**VTK WINS HPCWIRE EDITORS’ CHOICE AWARD**

For the second year in a row, Kitware was recognized in the annual HPCwire Readers’ and Editor’s Choice Awards. This year VTK won the award for best HPC visualization product or technology in the editors’ choice category. Winners in the editors’ choice category are determined by a panel of HPCwire editors and industry leaders.

Berk Geveci accepted the award at the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’11) on behalf of Kitware, the VTK community, and sponsors including Los Alamos National Laboratory, the National Library of Medicine, the National Alliance for Medical Image Computing (NA-MIC), the U.S. Department of Energy, Sandia National Laboratories, and the Army Research Laboratory.

**INTRODUCING SOFTWARE RELEASE WEBINARS**

To better support the open-source toolkits and communities, Kitware has started a new software release webinar series. With each major release of any of the open-source toolkits or applications, one of the Kitware developers working on the project will present a brief webinar. These free webinars will highlight new developments and give a preview of what is in the pipeline for the next release. These will be hosted live and we will be answering questions during the webinar. If you cannot attend, you can view the recording a few days after from the toolkits’ website (e.g. paraview.org/webinars). Registration information can be found on www.kitware.com or the various toolkit websites.

**VISWEEK 2011**

For the tenth straight year, Kitware sponsored VisWeek, the premier forum for advances in scientific and information visualization. The team was very active at this conference, presenting papers, participating in panel discussions, and meeting with collaborators. Jeff Baumes presented the paper “Geospatial and Entity Driven Document Visualization for Non-Proliferation Analysis” as part of the Workshop on Interactive Visual Text Analytics for Decision Making. Will Schroeder spoke as part of the “Verification in Visualization: Building a Common Culture” panel with Robert Kirby, Claudio Silva, and Robert Laramee.

The team was also busy giving demonstrations of Kitware’s new VTK OpenGL ES Rendering Toolkit (VES) and KiwiViewer, the mobile visualization app for geometric data.

**PARAVIEW AT 2011 LDAV SYMPOSIUM**

The 2011 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV), held in conjunction with VisWeek, included a number of presentations highlighting customizations and extensions of ParaView.

Joe Insley of Argonne National Laboratory discussed how he and his team used ParaView to develop visualization tools for data coupled from continuum-atomistic simulations in the paper “Visualizing Multiscale, Multiphysics Simulation: Brain Blood Flow”.

Venkat Vishwanath, also from Argonne, presented “Toward Simulation-Time Data Analysis and I/O Acceleration on Leadership-class Systems,” highlighting work on GLEAN, a flexible framework for data-analysis and I/O acceleration at extreme scales, and ParaView’s coprocessing capabilities. Also speaking to coprocessing was Nathan Fabian of Sandia National Laboratory who presented “The ParaView Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library”. Fabian presented results from integrating the library into various simulations codes, such as PHASTA, CTH, and S3D.

John Patchett of Los Alamos National Laboratory presented the paper “Revisiting Wavelet Compression for Large-Scale Climate Data using JPEG 300 and Ensuring Data Precision,” by Jon Woodring, et al. The paper argued that the majority of data bottlenecks are due to data movement associated with bandwidth limitations. Using ParaView, the team developed reader plugins, new streaming views, and enhancements for the VTK pipeline.

**HPC CONSORTIUM DISCUSSION OF OPEN SOURCE**


ParaView was highlighted in a few different presentations, hitting on topics such as CFD simulation by Cameron Smith of SCOREC and RPI, and massively parallel CFD using anisotropic grids by Ken Jansen and Onkar Sahni. Kirk Jordan, Program Director of the Computational Science center IBM Waston, discussed the use of ParaView Web as a good example of making HPC functionalities more accessible.
During a panel that focused on HPC software, audience members chimed in to comment on merits of open source tools in comparison to current prohibitive pricing models for commercial products. Based on some of the responses made by vendors and academics, there seems to be an incorrect perceived notion that open source tools are fragile and hard to use. Despite great advances in recent years, the open source community still has obstacles to overcome when it comes to winning over commercial HPC users and advocates.

**MICCAI 2011**

Stephen Aylward, Brad Davis, Andinet Enquobahrie, Luis Ibáñez, Danielle Pace, and Matt McCormick were actively involved in the 14th annual International Conference on Medical Image Computing and Computer Assisted Interventions (MICCAI) in Toronto this October.

Andinet led a systems and architectures workshop on computer-assisted interventions. Danielle Pace presented “Sliding Geometries in Deformable Image Registration” as part of the Abdominal Imaging workshop. Luis led tutorials on ITKv4, and SimpleITK along with Daniel Blezek of the Mayo Clinic.

Stephen Aylward acted as co-chair for oral presentations was a co-author on the paper “Geometric Metamorphosis,” along with Marc Niethammer, Gabriel L. Hart, Danielle F. Pace, Paul M. Vespa, Andrei Irimia, and John D. Van Horn. Marc Niethammer presented the paper.

The team also participated in the student event “Your Career in Industry and Academia,” an informal interactive session for students to discuss career opportunities in each arena.

Kitware also sponsored the Young Scientist Publication Impact Award, which recognizes scientists who are early in their careers and are shaping the medical image analysis field. We commend Dr. Boon Thye Yeo who received the inaugural award for his paper “Effects of Registration Regularization and Atlas Sharpness on Segmentation Accuracy.” Honorable mentions went to Dr. Kang Li for his paper, “Cell Population Tracking and Lineage Construction with Spatiotemporal Context” and Dr. Paul M. Novotny for his paper “GPU Based Real-time Instrument Tracking with Three Dimensional Ultrasound.”

**KITWARE PARTICIPATES IN RSNA 2011**

Kitware attended RSNA in Chicago, where we met with collaborators, demonstrated our newest work, taught workshops and participated in exhibits.

We taught two courses, “Open Source Applications for Medical Imaging Research” and “Introduction to Open-Source Software Libraries for Medical Imaging”. These courses had standing room only with more than 100 attendees and received high acclaim.

The open-source libraries course spoke to several open-source software libraries used in medical imaging, including VTK, ITK, Midas, and VES. We discussed the rigorous software processes used to ensure stability in the toolkits and specific use case applications for medical imaging. Participants were engaged with live demonstrations of coding of SimpleITK and new features of VES.

We also had two exhibits in the Quantitative Imaging Reading Room (QIRR): “3D Slicer: An open-source software platform for segmentation, registration, quantitative analysis and 3D visualization of biomedical image data,” a collaborative exhibit with NA-MIC, and “Understanding and Improving CT Image Quality With Automated Pocket Phantom Technology.”

**KITWARE SAS CELEBRATES FIRST ANNIVERSARY**

In October, Kitware's subsidiary office in Lyon, France celebrated their first anniversary. In their first year, Kitware SAS built a strong foundation for European operations and began collaborations on several fronts on projects covering medical imaging, high-performance computing, visualization, and data management. They were also busy giving multiple-day classes on Kitware projects including VTK, ITK, ParaView, and CMake, to major organizations across the EU.

Kitware SAS also doubled their team this year with the addition of two new members, Dr. Philippe Pêbay and Dr. Jérôme Velut. To accommodate their growing team, they recently relocated to 26 Rue Louis Guerin, 69100 Villeurbanne.

**KITWARE TALKS WITH POPULAR SCIENCE**

This November, Anthony Hoogs was featured as an industry thought leader in the article “Can Technology Save the Military From a Data Deluge?” from Popular Science. The piece investigates the U.S. military’s challenges with handling massive amounts of data and the current pace of technology acceleration.

Anthony spoke to the computer vision work being done at Kitware, which includes major projects such as DARPA’s Video Image Retrieval and Analysis Tool (VIRAT) program. “In video we have an important cue, which is motion. It turns out that motion is relatively easy to detect, and you don’t have to know what the object is. If something is moving, the intensity values at that location where the object is or where it was will change, and that’s relatively easy to pick up,” said Anthony.

**DR. TURNER BECOMES SPIE SENIOR MEMBER**

Dr. Wesley Turner, a technical leader on the medical imaging team, has been elected as a Senior Member of SPIE, an international society for optics and photonics.

The designation of senior membership is given to members who are heavily involved with the optics community, have at least five years with SPIE, and minimum of ten years professional experience.

Wes has been with the Society since May 2006 and has contributed his technical expertise and management skills to a range of projects including the exploitation of CT and MRI patient scans, microscopy, and bioinformatics.
KITWARE COURSES TAKE ON WEBINAR FORMAT
In September, Kitware announced free, monthly introductory courses for VTK, CMake, ParaView, and ITK. Within days, all scheduled classes were full. To accommodate community demand, we’ve moved to a webinar format that allows for up to 100 attendees per session.

We have scheduled the next round of online webinars. Please visit our website for the official enrollment links.

**Introduction to VTK**
Date: February 15, 2012, 10:30am - 12:00pm EST

**Introduction to CMake**
Date: March 14, 2012, TIME EST

**Introduction to ParaView**
Date: April 18, 2012, 2:00pm - 3:30pm EST

**Introduction to ITK**
Date: May 16, 2012, 11:00am - 12:30 pm

NEW EMPLOYEES

**Jason Li**
Jason Li joined the medical team at Kitware’s Clifton Park office in October. He holds a master’s degree in computer science from the University of Kentucky at Lexington and a bachelor’s degree in mechanical engineering from Beijing Jiao Tong University in China.

**Kyle Lutz**
Kyle Lutz joined the scientific visualization team at the Clifton Park office in June 2011. He holds a bachelor’s of science in biology from the University of California, Santa Barbara.

**Roland Kwitt**
Dr. Roland Kwitt joined the Kitware team at the Carrboro office in December. He holds a Ph.D. in computer science from the University of Salzburg in Austria, where he also received his masters of science and taught classes on network management and object oriented programming.

**Philippe Pébay**
Dr. Philippe Pébay joined Kitware SAS in Lyon, France in October as Director of Visualization and High Performance Computing. He holds a Ph.D. in applied mathematics from the Universite Pierre et Marie Curie in Paris.

**Jérôme Velut**
Dr. Jérôme Velut joined Kitware SAS in Lyon, France in November. He recently completed a post-doctoral fellowship in medical imaging from the University of Rennes after receiving his Ph.D. in information processing from the National Institute for Applied Science (INSA) in Lyon.

INTERNSHIP OPPORTUNITIES
Kitware Internships provide current college students with the opportunity to gain hands-on experience working with leaders in their fields on cutting edge problems in a unique open source environment.

Our interns assist in developing foundational research and leading-edge technology across six business areas: scientific computing, computer vision, medical imaging, data management, informatics and quality software process. To apply send, your resume to internships@kitware.com.

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

We offer comprehensive benefits including: flex hours; six weeks paid time off; a computer hardware budget; 401(k); health and life insurance; short- and long-term disability, visa processing; a generous compensation plan; yearly bonus; and free drinks and snacks. To apply, please forward a cover letter and resume to jobs@kitware.com.

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

- The Visualization Toolkit (www.vtk.org)
- The Insight Segmentation and Registration Toolkit (www.itk.org)
- ParaView (www.paraview.org)
- The Image Guided Surgery Toolkit (www.igstk.org)
- CMake (www.cmake.org)
- CDash (www.cdash.org)
- Midas (www.midasplatform.org)
- BatchMake (www.batchmake.org)
- KiwiViewer (www.kiwiviewer.org)
- Visomics (www.visomics.org)
- OpenChemistry (www.openchemistry.org)
- Slicer (www.slicer.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. Kitware’s Software Developer’s Quarterly is published by Kitware, Inc., Clifton Park, New York.


Graphic Design: Steve Jordan
Editors: Katie Osterdahl, Katie Sharkey

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

This work is licensed under a Creative Commons Attribution 3.0 Unported License. Kitware, KiwiViewer, ParaView, CMake and VolView are registered trademarks of Kitware, Inc. All other trademarks are property of their respective owners.