Editor's Note ........................................................................... 1  
Recent Releases ..................................................................... 1  
A Label Geometry Image Filter For Multiple Object Measurement ............................................................ 2  
RGB Colormapping And ITK ................................................. 5  
Morphology With Parabolic Structuring Elements .............. 8  
Label Object Representation And Manipulation With ITK .......................................................... 9  
In Progress ........................................................................... 13  
Kitware News ...................................................................... 14

Kitware is pleased to present a special edition of the Source which features several of the strongest Insight Journal submissions to date. 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 will 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.

VTK 5.2.1
The VTK 5.2.1 patch release is now available for download from the VTK web site (vtk.org). A full list of changes from 5.2.0 to 5.2.1 is available through the VTK user’s mailing list archives.

The VTK 5.4 release is slated for Summer 2009. To find out about the new features being implemented in VTK 5.4, visit the VTK Wiki and search for “VTK 5.4 Release Planning”.

ITK 3.10 RELEASE
ITK 3.10 was released on November 4, 2008. The major changes in this release include:

- Implementation of kernel-base filters contributed by Richard Beare and Gaëtan Lehmann. (http://hdl.handle.net/1926/555). In particular, kernel-base filters applied to Mathematical Morphology and filters that can be computed using moving histograms.
- Gaussian derivative operators contributed by Ivan Macia (http://hdl.handle.net/1926/1290). This contribution includes methods for computing Hessians and their eigenvalues at given points in an image.
- Mesh processing filters for surfaces using the QuadEdge mesh representation, contributed by Arnaud Gelas, Alexander Gouaillard, and Sean Megason (http://hdl.handle.net/1926/1495).
- Parameterization of discrete surfaces contributed by Arnaud Gelas and Alexander Gouaillard (http://hdl.handle.net/1926/1315).
- Introduction of classes for profiling Memory Usage contributed by Matt Turek and Julien Jomier.

Changes to the itk::Image class allow it to behave like itk::OrientedImage. This modification has profound implications for anyone working on image registration and image resampling. We strongly recommend users carefully review the results of their image registration programs once this release is adopted. More release details can be found on the ITK Wiki by searching for “ITK Release 3.10”.

ITK 3.10.1 AND 3.10.2 PATCH RELEASES
The major changes in these patch were related to fixing filters so that they take image orientation into account and fixes for the itkMultiThreader Class related to the global settings of maximum and default numbers of threads. Many thanks to Hans Johnson (University of Iowa) and Tom Vercauteren (Mauna Kea Technologies) for their hard work on identifying and fixing these problems.

The full lists of modifications can be found on the ITK Wiki by searching for “ITK Release 3.10.1” or “ITK Release 3.10.2”.

Kitware is pleased to present a special edition of the Source which features several of the strongest Insight Journal submissions to date. 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 will 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.
A LABEL GEOMETRY IMAGE FILTER FOR MULTIPLE OBJECT MEASUREMENT

The itkLabelGeometryImageFilter enables the measurement of geometric features of all objects in a labeled ND volume. This labeled volume can represent, for instance, a medical image segmented into different anatomical structures or a microscope image segmented into individual cells. The measurement of the various geometric features of these objects can provide additional insight into the image.

This filter is closely related to the itkLabelStatisticsImageFilter which measures statistics of image regions defined by a labeled mask (i.e., min, max, mean intensity, intensity standard deviation, bounding boxes). However, the itkLabelGeometryImageFilter measures the geometry of the labeled regions themselves. It measures features similar to the “regionprops” command of Matlab. The set of measurements that it currently enables are:

Shape Features
- Volume
- Centroid
- Eigenvalues
- Eigenvectors
- Eccentricity (2D)
- Elongation (2D)
- Orientation (2D)
- Bounding box
- Bounding box volume
- Bounding box size
- Oriented bounding box vertices
- Oriented bounding box volume
- Oriented bounding box size
- Rotation matrix

Shape & Intensity Features
- Integrated intensity
- Weighted centroid

While the majority of features are measured in ND, some are currently restricted to 2D (these are explicitly marked as 2D). The features listed here represent the set of currently calculated features, but the framework of the filter is designed so that it can be easily expanded to measure a wide variety of other features.

This article provides an overview of the itkLabelGeometryImageFilter. More details and supplemental material can be found on the Insight Journal at http://hdl.handle.net/1926/1493.

IMAGE MOMENTS

Image moments are particular averages of either binary objects (unweighted) or their pixel intensities (weighted). They are useful for describing objects and form the building blocks of many of the objects’ useful features. The definitions below are mostly given for 2D objects but can be directly extended to ND. For a 2D continuous function \( f(x,y) \), the raw moment of order \((p+q)\) is defined as:

\[
M_{p,q} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y) dx dy
\]

where \( x \) and \( y \) are indices of the first and second dimensions of the function. The discrete counterpart of this function is:

\[
M_{p,q} = \sum_{y=0}^{Y-1} \sum_{x=0}^{X-1} x^p y^q I(x,y)
\]

where \( I \) is the discrete image (weighted or unweighted).

Central moments are translationally invariant versions of the raw moments. This is achieved by subtracting the centroid \( \bar{x} = \frac{\mu_0}{\mu_0} \) and \( \bar{y} = \frac{\mu_0}{\mu_0} \) of the function from the indices. For a 2D continuous function, the central moments are defined as:

\[
\mu_{p,q} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q f(x,y) dx dy
\]

and the discrete version is:

\[
\mu_{p,q} = \sum_{y=0}^{Y-1} \sum_{x=0}^{X-1} (x - \bar{x})^p (y - \bar{y})^q I(x,y)
\]

Rather than calculating the raw and central moments separately for an image, the central moments can be directly derived in terms of the raw moments. For example, in 2D:

\[
\mu_{00} = M_{00}
\]

\[
\mu_{01} = 0
\]

\[
\mu_{10} = 0
\]

\[
\mu_{11} = M_{11} - \bar{x} M_{01} = M_{11} - \bar{x} M_{10}
\]

\[
\mu_{20} = M_{20} - \bar{x} M_{10}
\]

\[
\mu_{02} = M_{02} - \bar{y} M_{01}
\]

Note that the commas separating the \( p \) and \( q \) have been dropped for notational simplicity. The proof of these identities for one first order moment \((p = 0, q = 1)\), one second order cross moment \((p = 1, q = 1)\), and one second order moment \((p = 2, q = 0)\) for the continuous case are given in the online material.

A particularly useful form of moments is the normalized second order central moments. In 2D these are given by:

\[
\mu'_{20} = \frac{\mu_{20}}{\mu_{00}} = \frac{M_{20} - \bar{x}^2}{\mu_{00}}
\]

\[
\mu'_{02} = \frac{\mu_{02}}{\mu_{00}} = \frac{M_{02} - \bar{y}^2}{\mu_{00}}
\]

\[
\mu'_{11} = \frac{\mu_{11}}{\mu_{00}} = \frac{M_{11} - \bar{x} \cdot \bar{y}}{\mu_{00}}
\]

The covariance matrix consists of the normalized second order central moments organized as entries in a matrix. In 2D this becomes:

\[
\begin{bmatrix}
\mu'_{20} & \mu'_{11} \\
\mu'_{11} & \mu'_{02}
\end{bmatrix}
\]

This can be generalized to ND. Regardless of the dimension, each entry in this matrix represents the product of two elements (hence, it is second order). The dimension \( D \) of the moment determines the matrix size as \( D \times D \). To generalize
to ND, we introduce the notation $\mathcal{M}_D^p(v_1, v_2, \ldots, v_n)$ for raw moments and $C_N^D(v_1, v_2, \ldots, v_n)$ for normalized central moments, where $D$ is the dimension, $N$ is the order, and the values in the parentheses separated by commas represent the indices of the dimensions that are turned on (thus the number of values must be equal to $N$). For example, $\mu_{00}$ in the previous notation is $C_0^2(1,1)$, and $\mu_{020}$ is $C_2^3(1,1,1)$. This notation is more convenient for ND because it doesn’t require the explicit listing of a subscript value for each dimension as in $\mu_{020}$ in 3D. This notation is used in the following algorithm to calculate the normalized second order central moments, where $D$ is the image dimension and $N$ is the moments order:

```plaintext
for i = 0:D-1 do
    for j = 0:D-1 do
        if i == j then
            $C_2^D(i,j) = \frac{1}{12}$
        end if
    end for
end for
```

Since the resulting matrix is symmetric, in practice only half of the values need to be calculated. This algorithm simplifies the task of calculating second order moments in ND.

Notice in line 5 of the algorithm above that a constant is added when both elements of the second order moment are the same. This constant represents the normalized second order central moment of a pixel. This is required because the measurements are based on discrete pixels rather than continuous values.

**HYPERELLIPSOID FITTING**

Several useful features are calculated based on the features of a hyperellipsoid fit to each object. A hyperellipsoid can be fit using the eigenvalues/eigenvectors, which in turn depend upon the measurement of the covariance matrix. A 2D ellipse is shown in Figure 1. An ellipse is defined as a set of points, the sum of whose distances from the foci is a constant, $2a$, where $2a$ is the full length of the ellipse on the x-axis. With an ellipse centered on the origin and aligned with the x/y-axes, the foci of the ellipse are located at $(f,0)$ and $(-f,0)$. Then, the locations where the ellipse intersects the y-axis are at $(0,b)$ and $(0,-b)$, and the lines connecting the foci to these locations have length $a$. From the triangle thus formed, it is clear that $f^2 = a^2 - b^2$.

![Figure 1: Ellipse notation illustration](image)

These definitions can be expanded to ND and will be used in the following feature definitions. First, however, the covariance measurement will be defined in terms of the normalized second order central moments, and these will, in turn, be used to define the eigenvalues / eigenvectors.

**Covariance Matrix (ND)**

The covariance matrix is constructed directly from the normalized second order central moments in ND using the loop defined in the previously defined algorithm. For example, in 2D, this becomes:

$$
cov(I(x,y)) = \begin{bmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{bmatrix}
$$

and in 3D it becomes:

$$
cov(I(x,y)) = \begin{bmatrix} \mu_{200} & \mu_{110} & \mu_{011} \\ \mu_{110} & \mu_{020} & \mu_{001} \\ \mu_{011} & \mu_{001} & \mu_{002} \end{bmatrix}
$$

**Eigenvalues and Eigenvectors (ND)**

The eigenvectors and eigenvalues of the objects are needed for calculating many features. These are calculated directly by eigen-decomposition of the ND covariance matrix. The notation we use here is:

- $\lambda_i$ = $i^{th}$ eigenvalue of covariance matrix
- $\vec{v}_i$ = eigenvector corresponding to $\lambda_i$

For example, in 2D, $\lambda_1$ is the eigenvalue along the main axis, and $\lambda_2$ is the eigenvalue along the axis perpendicular to the main axis. Note that, in 2D, the calculation of the eigenvalues can be simplified as:

$$
\lambda_i = \frac{\mu_{20} + \mu_{02}}{2} + \frac{\sqrt{4\mu_{11}^2 + (\mu_{20} + \mu_{02})^2}}{2
$$

**CALCULATED OBJECT FEATURES**

**Volume and Centroid (ND)**

The volume and centroid of the objects can be directly calculated from the image moments. For example, in 2D: Volume = $M_{00}$, and Centroid = $[\frac{M_{00}}{M_{00}}, \frac{M_{00}}{M_{00}}]$. When calculating the unweighted centroid, $I$ is a binary object, and when calculating the weighted centroid, $I$ is the intensity image.

**Axes Lengths (ND)**

The length of the axes of the ND hyperellipsoid can be found directly from the eigenvalues as $4\sqrt{\lambda_i}$ where $i$ is the index of the eigenvalue from $i = 0, ..., D-1$. If the eigenvalues are in increasing order, the axes lengths will correspond to the increasing lengths of the hyperellipsoid axes. Thus, for a 2D object, the major axis length is $4\sqrt{\lambda_1}$, and the minor axis length is $4\sqrt{\lambda_0}$. Referring to Figure 1, these definitions mean that $a = 2\sqrt{\lambda_1}$ and $b = 2\sqrt{\lambda_0}$.

**Eccentricity (2D)**

Eccentricity is defined in 2D as the ratio of the distance between the foci to the length of the major axis. Using Figure 1, this is equal to $\frac{2f}{2a} = \frac{f}{a}$. Since $a = 2\sqrt{\lambda_1}$, $b = 2\sqrt{\lambda_0}$, and $f^2 = a^2 - b^2$, then

$$
f = \frac{\sqrt{(2\lambda_1)^2 - (2\lambda_0)^2}}{2\sqrt{\lambda_1}} = \frac{\lambda_1 - \lambda_0}{\lambda_1}
$$

Note that as the ellipse approaches a circle, $\lambda_0$ → $\lambda_1$, and the foci become $f^2 \approx \lambda_1^2 - \lambda_0^2 = 0$ leading to an eccentricity of 0. As the ellipse approaches a line, $\lambda_0$ → 0, and $f^2 \approx \lambda_1^2$, so the eccentricity becomes 1.
Eigenvector and the origin.

Orientation can be extracted from the angle between this eigenvector and the origin.

\[
\theta = \tan^{-1}\left(\frac{\mathbf{v}_1(1)}{\mathbf{v}_1(0)}\right)
\]

Note that in 2D this \(\theta\) can also be simply calculated without calculating the eigenvalues/eigenvectors as:

\[
\theta = \frac{1}{2} \tan^{-1}\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right)
\]

Bounding Box (ND)

The bounding box is calculated as the minimum and maximum indices in each dimension of the object. It is represented as a set of min/max pairs for each dimension. In 2D it is \([\text{min}(X), \text{max}(X), \text{min}(Y), \text{max}(Y)],\) and in 3D it is \([\text{min}(X), \text{max}(X), \text{min}(Y), \text{max}(Y), \text{min}(Z), \text{max}(Z)].\)

Oriented Bounding Box Vertices (ND)

The oriented bounding box is defined as the bounding box aligned along the axes of the object. It is more complex to compute than the standard axes-aligned bounding boxes and cannot be defined by simply using min/max pairs since the axes are no longer aligned with the image axes. The oriented bounding box is calculated using the eigenvectors to define the rotation of the object. First, the centroid of the region is subtracted so that the rotation will be about the center of the region. Then, the object is rotated to the new coordinate system defined by the eigenvectors. The bounding boxes are calculated in the rotated space and cannot be transformed directly back to the original space. This is because an oriented bounding box cannot be specified simply by the min and max in each dimension. Instead, the oriented bounding box is defined by its vertices, and these are transformed back to the original coordinate frame. Finally, the centroid is added back to yield the correct rotated bounding box vertices.

In our implementation, the order of the ND vertices corresponds with binary counting, where min is zero and max is one. For example, in 2D, binary counting will give \([0,0], [0,1], [1,0], [1,1],\) which corresponds to \([\text{min}(X),\text{min}(Y)], [\text{min}(X),\text{max}(Y)], [\text{max}(X),\text{min}(Y)], [\text{max}(X),\text{max}(Y)].\) In ND there will be 2N vertices.

Three additional features that are measured in the process are:
- the rotation matrix
- the oriented bounding box volume
- the oriented bounding box size, which is an ND vector describing the length of the bounding box in each direction

Oriented Image Region (ND)

The rotation matrix calculated in the oriented bounding box calculation can be used to rotate the image region of each object. When all objects are rotated around the centroid to align with the coordinate system defined by the eigenvectors, it has the effect of aligning all of the objects along common axes. This operation can be applied to either the original image or the intensity image and results in cropped images.

**IMPLEMENTATION**

This filter is implemented using a LabelGeometry class which contains all of the information for a particular label value. A mapper from the label value to the LabelGeometry class is populated for each label found in the label image.

To calculate the features, the code first loops through all of the pixels of the image to populate parts of the LabelGeometry structure, and then the code loops through each of the labels to calculate the remaining features. In the loop through all pixels in the label image, the filter calculates the following ND values for each label:
- Label value
- Raw zero order moment (volume)
- Raw first order moments
- Raw second order moments
- Bounding boxes

The output of the first loop is a mapping from all label values in the input label image to a LabelGeometry structure with the above listed features calculated. The mapper is used because, depending on the labels in the input label image, it may be that not all labels from 1 to max (labelValue) will be present (the label 0 is assumed to be background). The code then loops through all of the labels to calculate the rest of the features, which can be derived from the ones above as described in the section Calculated Object Features. In this iteration, the following values are calculated:
- Centroids
- Second order moment axes
- Normalized second order moment axes
- Covariance matrices
- Eigenvalues & Eigenvectors
- Axes lengths
- Eccentricity
- Elongation
- Orientation
- Bounding box volume
- Bounding box size
- Image regions defined by the bounding boxes

All of these features are calculated by default. If an intensity image is also defined, the code will also loop through the intensity image and calculate the integrated intensity and weighted centroid by default. In addition, if the corresponding methods are called, the following features are also calculated. These features require more computation and/or memory than the others:
- Pixel indices
- Oriented bounding box vertices
- Oriented bounding box volume
- Oriented bounding box size
- Rotation matrix
- Oriented label image
- Oriented intensity image (if intensity image is defined)

**INPUTS AND FEATURE ACCESSOR METHODS**

The only required input is a labeled image. This should be an image with unique label values for each individual object and
the value 0 for the background. An optional intensity image can also be supplied, in which case the features based on both intensity and shape will be calculated. To calculate only the default values, the following code can be used where relabeler is an instantiation of the itkRelabelComponentImageFilter that has been applied to the connected components of a binary image:

```cpp
typedef itk::LabelGeometryImageFilter<LabelImageType> LabelGeometryType;
LabelGeometryType::Pointer lgFilter = LabelGeometryType::New();
lgFilter->SetInput( relabeler->GetOutput() );
lgFilter->Update();
```

If it is also desired to calculate the features based on intensity and/or to calculate the features that take more time and memory, a desired selection of the following lines can be placed before the Update() command on the filter.

```cpp
tlgFilter->SetIntensityInput( reader->GetOutput() );
lgFilter->CalculatePixelIndicesOn();
lgFilter->CalculateOrientedBoundingBoxOn();
lgFilter->CalculateOrientedLabelRegionsOn();
lgFilter->CalculateOrientedIntensityRegionsOn();
```

The object features are accessed using the label of the object. In this case, the labels are assigned by the relabeler. In the following code, a label value is specified, and several features are queried for this label:

```cpp
LabelGeometryType::LabelPixelType labelValue = 9;
std::cout << "Volume: " << lgFilter->GetVolume(labelValue) << "\n";
std::cout << "Centroid: " << lgFilter->GetCentroid(labelValue) << "\n";
std::cout << "Axes Length: " << lgFilter->GetAxesLength(labelValue) << "\n";
std::cout << "Eccentricity: " << lgFilter->GetEccentricity(labelValue) << "\n";
std::cout << "Bounding box: " << lgFilter->GetBoundingBox(labelValue) << "\n";
```

**CONCLUSION**

The itkLabelGeometryImageFilter is a filter for measuring features of objects in a labeled image. Several core feature calculations are implemented and the code is designed so that measurements of other features can be easily added.

**Dr. Dirk Padfield** is a computer scientist at the GE Global Research Center in Niskayuna, NY. He also recently defended his PhD thesis in Computer Science at RPI. His research interests include image segmentation, tracking, registration and cellular image analysis.

**Dr. Jim Miller** is a Senior Scientist at GE Research where he has developed techniques for segmentation, registration, detection and quantification for oncology, pulmonology, cardiology, neurology and surgery applications in CT, PET, MR, and X-Ray. Jim was also the architect for the image processing pipeline for ITK and for the Slicer3 generic execution model.

Although grayscale intensity values are primarily used in image data visualization, there are times when mapping the grayscale image to a user-defined colormap is desired for aesthetic reasons (whether they be self-imposed or collaborator-suggested). In this article, we describe a framework in the ITK library for converting intensity-valued images to user-defined RGB colormap images. We also include several colormaps (shown in Figure 1) that can be readily applied for visualization of images in such programs as ITK-SNAP or for use as examples when creating new colormaps. More details and supplemental material can be found on the Insight Journal at http://hdl.handle.net/1926/1370.

**COLORMAP FUNCTOR BASE CLASS**

We introduce a new abstract base functor class called ScalarToRGBColormapFunctor. In mapping the set of input grey pixel intensity values to RGB values, it is often helpful to rescale the input values to the range [0;1] before performing the mapping to normalized RGB values also in the range [0;1]. One can then rescale the normalized RGB values to the desired range. Therefore, the abstract base class defines two helper rescaling functions: RescaleInputValue and RescaleRGBComponentValue. The former function then takes the normalized RGB values assumed to be in the range [0;1] and linearly rescales them to the range [m_MAXIMUMRGBCOMPONENTVALUE, m_MAXIMUMRGBCOMPONENTVALUE]. We demonstrate the

**Figure 1: Implemented colormaps**

To utilize the colormapping framework, the user can define colormap functor classes or use those which are already defined. Users are strongly encouraged to submit their user-defined colormap functor classes to the Insight Journal. The user could then "plug in" any one of these colormaps to the new image-to-image filter we created modeled after the UnaryFunctorImageFilter class to map a scalar image to an RGB image.
use of these functions when we discuss one of the derived RGB colormap functor classes.

Also, to be consistent with the various other functors that have been defined in ITK, we define the following operator functions:

```cpp
template <class TScalar, class TRGBComponent >
typename ScalarToRGBCopperColormapFunctor <TScalar, TRGBComponent >::RGBPixelType operator() ( const TScalar & v ) const 
{
    // Map the input scalar between [0, 1].
    RealType value = this->RescaleInputValue ( v );

    // Apply the color map.
    RealType red = 1.2 * value;
    red = vnl_math_min( 1.0, red );
    RealType green = 0.8 * value;
    RealType blue = 0.5 * value;

    // Set the components after rescaling the values.
    RGBPixelType pixel;
    pixel[0] = this->RescaleRGBComponentValue(red);
    pixel[1] = this->RescaleRGBComponentValue(green);
    pixel[2] = this->RescaleRGBComponentValue(blue);
    return pixel;
}
```

The reader will note the pure operator() function which is the only function we redefined in each of our derived colormap functor classes.

**DERIVED COLORMAP FUNCTOR CLASSES**

Each derived class defines a unique colormap functor class. For example, for each of the different colormaps in Figure 1, we have a .h and a .txx file which define that class. The filenames are labeled <name>ColormapFunctor followed by the respective file extensions (e.g. CopperColormapFunctor, HotColormapFunctor, AutumnColormapFunctor, etc.).

To demonstrate the facility of creating a new colormap functor class, we invite the reader to inspect the ScalarToRGBCopperColormapFunctor class, specifically the code defining the operator() function which describes how to perform the mapping between scalar intensity values and the copper colormap.

```cpp
template <class TScalar , class TRGBComponent >
typename ScalarToRGBCopperColormapFunctor <TScalar, TRGBComponent >:: RGBPixelType operator( const TScalar & other ) const 
{
    return this->RescaleRGBComponentValue(red);
}
```

The reader will note the pure operator() function which is the only function we redefined in each of our derived colormap functor classes.

**CUSTOM COLORMAP IMAGE FUNCTOR CLASS**

In addition to being able to derive other colormap classes, we provide a custom colormap functor class which allows one to specify a piecewise uniform sampling of the different RGB profile channels. Each channel (red, green, or blue) is specified by a vector of points in the range [0;1]. The mapping is performed by linearly interpolating between sample profile points. For example, the red, green and blue channels of the cool colormap class can be described, respectively, as:

- {0,1}
- {1,0}
- {1}

where the red channel varies linearly from 0 to 1 over the entire domain of the input scalar image, the green channel varies linearly from 1 to 0 over the domain of the input scalar image, and the blue channel has a constant value of 1 over the entire domain. Other colormaps require more samples for adequate description. We provide several of these colormaps described by piecewise samples (and shown in Figure 2) in the accompanying Source/CustomColormaps/ directory in the Insight Journal article.

**THE COORDINATING SCALAR IMAGE TO RGB IMAGE FILTER**

In previous sections we described the various colormap functor classes. In this section we describe the coordinating filter, ScalarToRGBColormapImageFilter, which accepts one of these colormap functor classes and an input scalar image and produces, as output, an RGB image where the pixel values have been mapped according to the functor class. We derive this class from the ImageToImageFilter class with a large portion of the code taken and adapted from the UnaryFunctorImageFilter class. In this way, the class has multi-thread capabilities. Also, if no colormap is specified, the output behavior is defaulted to a grayscale mapping.

Usage is best illustrated by examining the test code which produced the images contained in the next section. This test code is included in the full version of this article on the Insight Journal site, with snippets provided here.
After specifying the input scalar and output RGB image types, we calculate the minimum and maximum input pixel values to supply to the specified colormap. The selected colormap functor is instantiated and supplied to the image filter. We also specify the minimum and maximum RGB component values. We then write the output to an image and can visualize the result in ITK-SNAP or other software capable of viewing RGB images.

For a custom colormap defined by piecewise samples, the following code which uses the text files in the Source/CustomColormaps/ directory, illustrates its use. The reader will note that the channels do not necessarily need to be of the same length.

```cpp
else if ( colormapString == "custom" )
{
    typedef itk::Functor::
        ScalarToRGBCustomColormapFunctor
    <ImageType::PixelType, RGBImageType::PixelType::
        ValueType> ColormapType;
    ColormapType::Pointer colormap =
        ColormapType::New();
    ifstream str( argv[3] );
    std::string line;
    // Get red values
    std::getline( str , line );
    std::istringstream iss( line );
    float value;
    ColormapType::ChannelType channel;
    while ( iss >> value )
    {
        channel.push_back( value );
    }
    colormap->SetRedChannel( channel );
    // Do the same for the green and blue channels
    // (code not shown here for brevity)
    rgbfilter->SetColormap( colormap );
}
```

Finally, the min and max input values and RGB component values are set.

```cpp
rgbfilter->GetColormap()->
    SetMinimumRGBComponentValue( 0 );
rgbfilter->GetColormap()->
    SetMaximumRGBComponentValue( 255 );
rgbfilter->GetColormap()->
    SetMinimumInputValue( stats->GetMinimum() );
rgbfilter->GetColormap()->
    SetMaximumInputValue( stats->GetMaximum() );
```

**VISUALIZING RGB COLOR IMAGES WITH ITK SNAP**

In Figure 3 we showcase recently added functionality to ITK-SNAP which allows the user to visualize RGB image layers. After converting the RGB image shown in the top left of Figure 3 to an RGB image using the spring colormap, we can visualize the two images with varying levels of opacity. (Note that only RGB images of the metaoio format are currently readable in ITK-SNAP.)

![Figure 3](image.png)

**AN ITK HOMAGE TO ANDY WARHOL**

As a pioneer in the Pop Art movement Andy Warhol is famous for depicting various iconic figures using different coloring schemes for artistic purposes. We use our new framework to produce our own creation very much in the spirit (although perhaps lacking Warhol's artistic sensitivities) of the Pop Art movement in Figure 4 (wikipedia.org/wiki/Lenna).

![Figure 4](image.png)

**ACKNOWLEDGEMENTS**

Thanks to Hui Zhang, Gaëtan Lehmann, Paul Yushkevich, James Gee for their invaluable contributions to this article.

**Nick Tustison** consciously engages primarily at the Penn Image Computing and Science Laboratory (PICSL) at UPenn where he orchestrates the composition of medical image analysis software while secretly sonneting about the more poetic of the ITK filters.
MORPHOLOGY WITH PARABOLIC STRUCTURING ELEMENTS

Morphological erosion and dilation filters employ a structuring function, with flat structuring functions being the most common example. Parabolic structuring functions are less well-known but theoretically very important and practically very useful. This article briefly introduces morphology using parabolic structuring functions, describes the ITK classes used to implement them and includes a number of sample applications. More details and supplemental material can be found on the Insight Journal at http://hdl.handle.net/1926/1370.

INTRODUCTION
Parabolic structuring functions (PSF) have the following important properties:

- They are closed under dilation—i.e., dilating two PSFs results in a third PSF.
- An n-dimensional PSF can be obtained by combining n one-dimensional PSFs in independent directions.
- PSFs are rotationally symmetric allowing the dimensional decomposition by dilation.

These properties make PSFs the morphological counterpart of the Gaussian in linear image processing [1]. The dimensional decomposition properties lead to efficient algorithms for implementing parabolic morphological operations. Parabolic morphology operations are useful in image sharpening and distance transforms. They are also a less vigorous alternative to conventional morphological operations based on flat structuring elements and a potentially useful, faster alternative to shaped structuring elements such as the "rolling ball" (often used in background estimation in tools such as ImageJ).

BRIEF BACKGROUND THEORY
Dilation by a one dimensional parabolic structuring element is illustrated in Figure 1. The dilation of the signal at point A is found by lowering the structuring function centered at A until the structuring function comes into contact with the signal, in this case at point B. The dilation at this point is given by point C, the height of the parabola turning point. The equivalent erosion is calculated by raising an inverted parabola into the signal from below.

ITK CLASSES
The filters discussed in this section implement the “point of contact” algorithm [1]. Efficient Algorithms for larger kernel sizes are available and are discussed by van den Boomgaard et al., but haven’t been implemented in this package. The classes itkParabolicErodeImageFilter, itkParabolicDilateImageFilter, itkParabolicCloseImageFilter and itkParabolicOpenImageFilter provide the range of parabolic morphology operations. These classes derive from itkParabolicErodeDilateImageFilter and itkParabolicOpenCloseImageFilter, which implement the core functionality. Open and close filters offer an optional border padding facility. Controlling methods available are:

- Set/GetScale: a PSF is usually defined as \( f(x) = \frac{1}{2\rho} x^2 \). This method sets \( \rho \).
- Set/GetUseImageSpacing: defines whether the units for \( \rho \) are voxels or image spacing.
- SetSafeBorder: controls whether border padding is used in opening or closing operations.

APPLICATIONS
Sharpening
Image sharpening using parabolic operations was analyzed in [3]. This form of sharpening is based on the following formula:

\[
\epsilon[f](x, \rho) = \begin{cases} 
F^\oplus(x, \rho), & F^\oplus(x, \rho) - F(x, 0) < F(x, 0) - F^\ominus(x, \rho) \\
F^\ominus(x, \rho), & F^\oplus(x, \rho) - F(x, 0) > F(x, 0) - F^\ominus(x, \rho) \\
F(x, 0), & \text{otherwise}
\end{cases}
\]

where \( F^\oplus(x, \rho) \) indicates dilation by a parabolic structuring element with scale \( \rho \) and \( F^\ominus(x, \rho) \) indicates erosion. In summary, the output of the sharpening process at a pixel is the dilation, if the difference between the dilation and the input is less than the difference between the erosion and the input. This formula describes one step in an iterative process. The parameters controlling the process are scale and number of iterations. Scale selection will require experimentation, but will tend to be related to the size of blurring.

This sharpening algorithm has been implemented in the itkMorphologicalSharpeningImageFilter, and testSharpening illustrates its use. Figure 2 illustrates the results of sharpening using profiles through an artificial image.
Distance Transforms

Parabolic erosions and dilations can be used to construct distance transforms. Two classes have been provided in this package:

- itkMorphologicalDistanceTransformImageFilter for one-sided distance transforms.
- itkMorphologicalSignedDistanceTransformImageFilter for signed distance transforms.

The concept is fairly simple. Consider applying a parabolic erosion, with scale 1, to a mask image with values of 0 and inf. The value of the erosion at mask locations with value 0 will remain 0. The value at mask locations of inf are equal to the square of the distance to the nearest 0 value. The idea can be easily extended to allow computation of the inside and outside distances without rethreesholding. This method should be more accurate than basic chamfer approaches and appears to be slightly faster than the SignedDanielssonDistanceTransform currently provided in ITK, as illustrated in Table 1.

The parabolic version does not return voronoi tessellations or vectors pointing to the nearest inverse pixel because such things aren’t required in construction of the distance transform. The Danielsson approach produces an approximate distance transform (albeit a good one for most purposes), while the approach using parabolic morphology is exact. The parabolic morphology approach does not perform as well as the Maurer approach, but may be able to do better in the future since it is easily threadable (see Section on Multi-Threaded Implementations) and other optimizations that haven’t yet been implemented are available. In addition, no attempt has been made to use integer arithmetic.

<table>
<thead>
<tr>
<th>Image</th>
<th>Parabolic</th>
<th>Maurer</th>
<th>Danielsson</th>
</tr>
</thead>
<tbody>
<tr>
<td>256x256</td>
<td>0.0457</td>
<td>0.0152</td>
<td>0.0568</td>
</tr>
<tr>
<td>258x258x182</td>
<td>10.4</td>
<td>7.28</td>
<td>16.1</td>
</tr>
</tbody>
</table>

Table 1: Execution times, in seconds, for computing signed distance transforms using parabolic morphology, Maurer and Danielsson methods in ITK. The standard, 256x256 ctbrain image was used for 2D, the bunnyPadded image for 3D, and the test was repeated 10 times. These figures were obtained using perfDT and perfDT3D, in this package. Tests were single threaded and run on an Intel Core2 Duo CPU, E4600, 2.40GHz, 2M cache running Linux.

CONCLUSIONS AND FURTHER WORK

This article introduces implementations of an important class of morphological operations - erosions and dilations as parabolic structuring elements. Applications of these operations to sharpening and distance transforms are also presented. The latter appears to outperform some existing implementations and may do better as improved algorithms are implemented.

ACKNOWLEDGEMENTS

Thanks to Paul Jackway for valuable discussions.

REFERENCES


Richard Beare

Richard Beare works in the Neuroscience Research Group at Monash University where he is responsible for the development of automated image analysis tools for use with medical imaging studies. His interests include automated image segmentation, image motion and tracking, and techniques for exploratory analysis of fMRI at DTI neuroimaging data. His previous position gave him extensive experience in theoretical and applied aspects of image analysis, working on projects as diverse as real-time control of robotics using ultrasound and high-content screening for drug discovery.
belong to the same connected component in the data. A label can be represented by any data type and only needs to be unique within the image. Labels do not need to be ordered; however, in practice we often use integral number types as labels for several reasons: they are commonly used in image analysis, they efficiently represent the label in memory and its easy to find the next label by adding 1.

Label image
A label image is an image which contains labeled pixels. The labels are often used to represent objects placed on a background, so the label image may use a particular label to identify the background.

Binary image
A binary image is an image with two labels: a foreground and a background. In practice, binary images utilize a pixel type that is capable of storing more than those two values. The foreground is thus defined with a particular label, and the other labels in the image are considered part of the background. This allows us to treat a label image as a binary image and by treating that label as the foreground label we can manipulate a single object in the label image.

Attribute
An attribute is a value of any type associated with a label. For example, it can be the size of an object, the mean of its pixels intensities, etc.

DATA REPRESENTATION
Label images are often used to represent the connected components of an image. In this contribution, another representation has been chosen. The objects contained in the image as connected components can be efficiently stored in memory as a set of lines using the run-length encoding: a starting point for each line, and the length of the line on a given dimension (by convention, the dimension 0).

The image is a collection of those objects and it also stores some values of the image like its size, its spacing, etc.

ITK::LABELMAP
The itk::LabelMap class is in charge of managing the image's collection of label objects and storing the metadata associated with the image like the spacing, the physical position—all the metadata found in itk::Image. It has been chosen to simplify the implementation and the tests, and because the feature is rarely useful in practice, to not implement the conditional background in the itk::LabelMap class. All the images represented by a itk::LabelMap object have a background. If the user wants to manipulate such an image with no background, s/he has to avoid the background label, for example, by using a larger label type.

The itk::LabelMap provides a part of the API of the itk::Image class and can be manipulated as an image in many cases. Based on the different data structures used, the performance can vary.

The itk::LabelMap is a templated class which takes a single parameter—the type of label object stored by that class. The dimension of the image is taken from the label object class thus, it doesn’t need to be defined as a class template parameter. The pixel type of the image also comes from the label object class.

ITK::LABELOBJECT
The itk::LabelObject class represents the label objects. It has two main features: it manages the set pixels which compose the object and it has a label.

No attribute, except the label, is stored in this class which can thus be seen as the base class for the objects with attributes or used when no attributes are required. The itk::LabelObject class is templated and takes two required template parameters, the type of the label and the dimensions of the image.

Several subclasses are provided to cover the most common usage of the label object's manipulation:

- itk::AttributeLabelObject is able to store a generic attribute. It is generic in the sense that its type is given in template parameter.
- itk::ShapeLabelObject contains numerous attributes related to label object shape. Computing the values of those attributes does not require a feature image.
- itk::StatisticsLabelObject contains numerous statistics about the grey values of a feature image in the same place as the label object. Computing the values of those attributes does require a feature image.

To reduce the number of filters made to manipulate the attributes and to make the computation of all attribute sets more efficient the itk::ShapeLabelObject and itk::StatisticsLabelObject classes were created.

The scalar values of the attributes of the itk::ShapeLabelObject and the itk::StatisticsLabelObject classes are often given both in pixel and in physical units, in order to give some parameters independent of image spacing.

The position in the images are given as an index position when the position is the exact position of a pixel (for example, the position of the maximum value in the feature image) or as a physical position when the position is given at subpixel resolution (for example, the centroid). In both cases the position can easily be converted to the other representation with the itk::LabelMap::TransformPhysicalPointToIndex() and itk::LabelMap::TransformIndexToPhysicalPoint() methods.

Both itk::ShapeLabelObject and itk::StatisticsLabelObject are templated classes. They take the same template parameters as itk::LabelObject. The two first template parameters of the itk::AttributeLabelObject class are the same as the ones for itk::LabelObject class. The third one is the attribute type.

ITK::SHAPELABELOBJECT
The itk::ShapeLabelObject has the following properties:

- Size is the size of the object in number of pixels.
- PhysicalSize is the size of the object in physical units. It is equal to the Size multiplied by the physical pixel size.
- Centroid is the position of the center of the shape in physical coordinates. It is not constrained in the object, thus it can be outside of the object if the object is not convex.
- Region is the bounding box of the object given in the pixel coordinates. The physical coordinate can easily be computed from it.
- RegionElongation is the ratio of the longest physical size of the region on one dimension and its smallest physical size. This descriptor is not robust and, in particular, is sensitive to rotation.
• SizeRegionRatio is the ratio of the size of the object region (the bounding box) and the real size of the object.

• SizeOnBorder is the number of pixels in the objects which are on the border of the image. A pixel on several borders (a pixel in a corner) is counted only one time, so the size on border can’t be greater than the size of the object. This is useful for removing objects too close to the image border.

• PhysicalSizeOnBorder is the physical size of the objects which are on the border of the image. In 2D it is a distance, in 3D a surface, etc. Unlike the PhysicalSize attribute, which is directly linked to Size, this attribute is not directly linked to SizeOnBorder. PhysicalSizeOnBorder is particularly useful for removing objects too close to the image border.

• FeretDiameter is the diameter of the sphere in physical units including all of the objects. The feret diameter is not computed by default because of its high computation.

• BinaryPrincipalMoments contain the principal moments.

• BinaryPrincipalAxes contain the principal axes of the object.

• BinaryElongation is the elongation of the shape, computed as the ratio of the largest principal moment by the smallest principal moment. Its value is greater than or equal to 1.

• EquivalentRadius is the equivalent radius of the hypersphere which is the same size as the label object.

• EquivalentPerimeter is the equivalent perimeter of the hypersphere which is the same size as the label object.

• EquivalentEllipsoidPerimeter is the size of the ellipsoid which is the same size and the same ratio on all the axes as the label object.

ITK::STATISTICSLABELOBJECT

The itk::StatisticsLabelObject class has the following properties: Minimum, MinimumIndex, Maximum, MaximumIndex, Mean, Sum, Sigma, Variance, Median, CenterOfGravity, Kurtosis, Skewness, PrincipalMoments, PrincipalAxes, Elongation, Histogram.

ITK::LABELMAP USAGE

The itk::LabelMap class provides some methods to manually fill in the image (e.g., the usual SetPixel() method.) However, the most efficient way to fill in the image is to convert a label image or a binary image stored in an itk::Image to an itk::LabelMap by using itk::BinaryImageToLabelMapFilter or itk::LabelImageToLabelMapFilter.

The label objects produced by those filters have no attribute value set thus, the attributes must be evaluated. The following are some of the most commonly used filters:

• itk::ShapeLabelMapFilter generates the attributes of the itk::ShapeLabelObjects

• itk::StatisticsLabelMapFilter generates the attributes of the itk::StatisticsLabelObjects.

For the itk::AttributeLabelObject class or other classes the user must set the values manually. For example, implementing a subclass of itk::InPlaceLabelMapFilter.

MANIPULATING THE ITK::LABELMAP

Once created and optionally evaluated, several filters are provided to manipulate the itk::LabelMap.

Openings can be performed with the OpeningLabelMapFilter classes. Those classes will remove all of the objects with an attribute value lower or higher than a given value.

Because we can often use criteria which haven’t been used during the segmentation procedure (i.e., size of the object, mean value of its pixels) the attribute opening is a very efficient way to enhance segmentation. For example, after a thresholding of a grayscale image, the objects too small or too big to be of interest can be removed. The class AttributeSelectionLabelMapFilter and its subclass LabelSelectionLabelMapFilter can be used to remove some objects based on their attribute values, even if the attribute type has no ordering property.

A fixed number of objects can be kept with the KeepNOBJECTsLabelMapFilter classes. They are chosen according to the values of their attribute. The user can choose to keep those with the highest or the lowest attribute values.

Objects can be relabeled with the RelabelLabelMapFilter classes. The order of the label is dependant on the value of the attribute. The user can choose to have the objects with the highest attribute value in the first labels or to have the objects with the lowest attribute values in the first labels.

The region covered by the itk::LabelMap can be changed with itk::ChangeRegionLabelMapFilter and its subclasses (itk::CropLabelMapFilter, itk::PadLabelMapFilter, itk::RegionFromReferenceLabelMapFilter, itk::AutoCropLabelMapFilter).

The itk::LabelMap is useful for getting the attribute values associated with the objects. The classes provided can be used in place of itk::LabelStatisticsImageFilter, or for obtaining data regarding the shape or the position of the object.

Figure 1: Some examples of opening with different attributes and parameters. Note that labels are kept unchanged in the output image.
GENERATING AN ITK::IMAGE FROM THE ITK::LABELMAP

Once manipulation of the objects is done, it is useful to go back to a more classic itk::Image. Several classes to do that are provided below:

- The itk::LabelMapToLabelImageFilter class simply converts an itk::LabelMap to a label image stored in an itk::Image.
- The itk::LabelMapToBinaryImageFilter puts all the objects in the foreground of a binary image stored in an itk::Image. It is intended to be used with an image produced by the itk::BinaryImageToLabelMapFilter. The background values of the original image can also be restored by this filter.
- The itk::LabelMapMaskImageFilter class can be used to mask an image with the objects of the itk::LabelMap. With that filter, the image can be cropped to contain only the non-masked zone or the non-masked zone padded by a user-defined number of pixels.
- The itk::LabelMapToAttributeImageFilter produces an itk::Image with the value of the objects attribute from the itk::LabelMap. This filter is useful for providing a global view of the attribute values in the image.
- The LabelMapOverlayImageFilter produces a color itk::Image with itk::RGBPixel as its pixel type. The label objects are in color, as in Figure 2. This class is useful for quick visual validation without going outside of ITK.

![Figure 2: (a) the label image of connected components in (b) is the same image with labels colored with itk::LabelToRGBImageFilter.](image)

- The LabelMapToRGBImageFilter produces a color itk::Image with itk::RGBPixel as its pixel type. The label objects are in color on top of a grayscale image, as in Figure 3. This is useful for quick visual validation without going outside of ITK.

![Figure 3: The segmented nuclei. The too small objects and those on the border have been excluded.](image)

PREBUILT MINI-PIPELINE FILTERS

The general view of the previous sections show very common ways to use those classes. To make them easier to use, some prebuilt classes have been made to perform a mini-pipeline:

- creation of the itk::LabelMap from an itk::Image
- evaluation of the attribute(s) of the objects
- filtering of the itk::LabelMap
- creation of an itk::Image from the filtered itk::LabelMap, with a specific attribute.

When using the itk::ShapeLabelObject or the itk::StatisticsLabelObject class, we usually want them to be evaluated. Some classes are provided to perform the mini-pipeline:

- creation of the itk::LabelMap from an itk::Image
- evaluation of the object’s attributes with itk::ShapeLabelObject or the itk::StatisticsLabelObject

Because the objects come from label images or binary images, those filters have been made for both binary and label images.

- The following binary filters produce evaluated attributes:
  - itk::BinaryImageToShapeLabelMapFilter
  - itk::BinaryImageToStatisticsLabelMapFilter
- The following binary filters fully hide the usage of label objects:
  - itk::BinaryAttributeKeepNObjectsImageFilter
  - itk::BinaryAttributeOpeningImageFilter
  - itk::BinaryShapeKeepNObjectsImageFilter
  - itk::BinaryShapeOpeningImageFilter
  - itk::BinaryStatisticsKeepNObjectsImageFilter
  - itk::BinaryStatisticsOpeningImageFilter
- The following label filters produce evaluated attributes:
  - itk::LabelImageToShapeLabelMapFilter
  - itk::LabelImageToStatisticsLabelMapFilter
- The following label filters fully hide the usage of label objects:
  - itk::LabelAttributeKeepNObjectsImageFilter
  - itk::LabelAttributeOpeningImageFilter
  - itk::LabelShapeKeepNObjectsImageFilter
  - itk::LabelShapeOpeningImageFilter
  - itk::LabelStatisticsKeepNObjectsImageFilter
  - itk::LabelStatisticsOpeningImageFilter
  - itk::ShapeRelabelImageFilter
  - itk::StatisticsRelabelImageFilter

All of the following filters are using a morphological reconstruction, implemented internally as an attribute opening with the itk::AttributeLabelObject class. Some binary filters are implemented:

- itk::BinaryClosingByReconstructionImageFilter
- itk::BinaryFillholeImageFilter
- itk::BinaryGrindPeakImageFilter
- itk::BinaryOpeningByReconstructionImageFilter
- itk::BinaryReconstructionByDilationImageFilter
- itk::BinaryReconstructionByErosionImageFilter

and some label filters which are useful when the label objects are connected or when you want to avoid losing the label by using a binary filter. Because the notion of reconstruction by erosion is difficult with labels, only a few filters are implemented:

- itk::LabelReconstructionByDilationImageFilter
THREADING SUPPORT
When possible, the filters have been multithreaded. Some of them, however, are not (easily) threadable (the KeepNOObjects and Relabel filters), or would not see performance improvement in a threaded version (the Opening filters). The itk::BinaryImageToLabelMapFilter class is a slight modification of the itk::ConnectedComponentImageFilter which has been threadable to enhance performance on a multi-core system.

The classical thread architecture is used when the input image is an itk::Image. The image is split into several regions (one per thread), and each thread works on its own region.

Because the itk::LabelMap image is not an array of pixels, it can’t be split that way. Instead, several threads are created; these threads try to take an object in the collection. If they get one, they process that object individually and try to get another one when the object is processed. If no object can be obtained the thread ends. An itk::FastMutexLock is used to ensure that only one thread takes an object at a time.

For the developer, the use of threading support is simplified by subclassing itk::LabelMapFilter or itk::InPlaceLabelMapFilter, and implementing the method void ThreadedGenerateData(LabelObjectType * labelObject) in the new class. This method only has to process the labelObject parameter; all of the threading code and mutex lock management is already implemented.

IN PLACE FILTERING
All of the filters which take itk::LabelMap as input and produce itk::LabelMap as output are implemented as a subclass of InPlaceLabelMapFilter and run in place by default.

The user can modify this behavior with the SetInPlace(bool) method, along with the usual InPlaceOn() and InPlaceOff() methods, along with the usual InPlaceImageFilter. To use this feature, a developer would subclass InPlaceLabelMapFilter and implement the virtual void ThreadedGenerateData(LabelObjectType * labelObject) method, which is the input image if the filter runs in place, or a copy of the input image if the filter is not running in place.

ACKNOWLEDGMENTS
The author would like to thank Richard Beare for his suggestion to use the run length encoding to represent the binary objects, and Julien Jomier for his help in deciding not to use the itk::SpatialObject class as base class of the itk::LabelObject class. The author would also like to thank Dr. Pierre Adenot and MIMA2 confocal facilities (mima2.jouy.inra.fr) for providing the 3D test image, and Dr. Maria Ballester for providing the image used in the python example.

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 the Insight Toolkit, including several packages in the mathematical morphology field and WrapITK, the enhanced language support for ITK.

VTKEDGE
VTKEdge has grown rapidly over the past few months, with several significant functionality areas under active development. The following is a brief description of three such functionality areas.

A framework has been added to VTKEdge to facilitate the serialization and deserialization of vtkObjects. Serialization "helpers" provide the ability to handle vtkObjects in general (such as existing classes in VTK; enabling an existing VTK class requires only a small amount of work), while vtkKWE-SerializableObject (a thin subclass of vtkObject) provides a simple serialization framework to its subclasses.

The ObjectTree structure was added to VTKEdge to allow a general but flexible method of associating properties (such as visual properties), transforms and other attributes to nodes (each containing a vtkObject) in the tree. Implemented as a subclass of vtkKWE-SerializableObject, an ObjectTree can easily be serialized. Some potential applications include a structure for holding the state of the objects in an application (so that it can be saved and restored), an aid to undo/redo functionality and a more efficient method for rendering a scene of what would otherwise be potentially thousands of vtkActors (nodes in the ObjectTree represent the different pieces of geometry which a specialized mapper could handle as single entity).

A framework for a special class of “paintbrush” widgets has been added to VTK to facilitate 2- and 3D image data editing. A paintbrush is a shape (which may be interactively sized and positioned) that modifies the image through a customizable operation. This framework allows not only simple “painting” or erasing of properties, but also more complex functionality such as region growing constrained by the underlying image. The framework provides widget grouping to facilitate collections of paintbrushes working on the same data shown in multiple views. For example, editing medical data by interacting with the axial, coronal and sagittal slices. Undo / redo functionality is also provided by the framework.

Figure 1: A screenshot of Merlin, a Maverick application, illustrating the use of paintbrush widgets in VTKEdge to edit label maps. Here, a brush is used to perform constrained smoothing of a label on a voxelized segmentation of Duke (Virtual Family).
The first official release of VTKEdge will occur in conjunction with the VTK 5.4 release which is expected to be complete by Summer 2009.

KITWARE NEWS

KITWARE SPONSORS CAMP KDE
Kitware sponsored the first Camp KDE (http://camp.kde.org). Camp KDE 2009 was KDE’s first annual developer conference taking place West of the Atlantic. The conference is a by-product of the very successful KDE 4 Release Event held in Mountain View, California. It was intended to ensure that KDE’s presence in the world is not simply seen as being Euro-centric.

Camp KDE 2009 was held at the Travellers Beach Resort in Negril, Jamaica January 17 - 23, 2009.

RPI OPEN SOURCE SOFTWARE PRACTICE COURSE
Drs. William J. Schroeder and Luis Ibáñez of Kitware teach an undergraduate level course at Rensselaer Polytechnic Institute entitled “Open Source Software Practice”. The course was established as part of Rensselaer’s Center for Open Software. The purpose is to foster an awareness of open source concepts, issues and opportunities. In addition to providing students with practical engagement in open-source communities through course projects, the course covers open source licensing, open source business models, and case studies of successful open source systems. Several prestigious speakers also present at the course. This year the following guests addressed the course:

- Michael Tiemann, Vice President of Open Source Affairs at Red Hat, Inc.
- Chris DiBona, Open Source Programs Manager at Google, Inc.
- Daniel Frye, Vice President of Open Systems Development at IBM
- Open source marketing experts Sandro Groganz (Init Marketing) and James Cherkoff (Collaborate Marketing)
- Kent Quirk from Linden Lab (i.e., Second Life)

Several other Kitware personnel also taught portions of the course, on the following topics: Bill Hoffman and Dave Cole—software process, Niki Russell—marketing and Ken Martin—business models.

More information about the course, including notes, can be found by searching “Open Source Software Practice” on the course Wiki which is hosted at public.kitware.com.

KITWARE, OSA RELEASE SOFTWARE FOR INTERACTIVE SCIENTIFIC PUBLISHING
Kitware is pleased to announce the launch of OSA’s Interactive Science Publishing (ISP) system. ISP is the first open-access scientific publication system that allows authors to distribute imaging datasets and advanced visualizations over the Internet for interactive display. The dissemination of information and data using novel software technology and systems falls directly in line with Kitware’s organizational goals and its efforts to advance publishing systems for open science. OSA ISP is a unique publishing system that enables researchers to create active publications supporting direct interaction with the data referred to in the publication.

“Kitware is delighted to have partnered with the Optical Society of America and the National Library of Medicine to develop and launch the Interactive Science Publishing system,” said Kitware’s Director of Medical Applications, Rick Avila. "This first of its kind open-access system represents an important milestone in scientific publishing that will significantly improve the dissemination of scientific findings and ultimately accelerate scientific enquiry, all of which are at the core of Kitware’s long-standing commitment to open science."

ISP was built on top of Kitware’s MIDAS multimedia server and VolView visualization application, which is an intuitive, interactive system for volume visualization. ISP allows authors to submit a manuscript including large three-dimensional data sets and the visualization settings needed to reproduce figures in the manuscript. Readers are then able to load and interactively explore the author’s datasets using advanced 2D and 3D display methods and tools.

Clicking on a figure in an ISP paper launches ISP, which obtains the requested dataset from Kitware’s Midas server and displays the data as an interactive rendering.

The ISP software is the result of an ISP initiative between The Optical Society (OSA) and the National Library of Medicine (NLM). Kitware collaborated with OSA, a leader in providing educational resources that support technical and professional development in the field of optics, to develop the ISP software.
ISP provides the opportunity for researchers, scientists and engineers to evaluate new research results more thoroughly. OSA staff published the first two ISP articles in their all-electronic journal for optics, Optics Express, on October 15, 2008.

For more information on ISP, visit http://www.opticsinfobase.org/isp.cfm.

OPEN-SOURCE TOOLS FOR MEDICAL RESEARCH AND APPLICATIONS
The field of medical image research and application is undergoing an explosion of scientific advancement, due in part to the spread of open source software. In a workshop presented at RSNA 2008, Kitware’s Director of Medical Applications, Rick Avila and Chief Medical Scientist, Stephen Aylward presented on two leading open-source, freely available software libraries for medical image analysis and visualization. The first is the Insight Toolkit (itk.org) which was funded by the National Institutes of Health, particularly the National Library of Medicine, to provide state-of-the-art medical image segmentation and registration algorithms. The second is the Visualization Toolkit (vtk.org) which provides leading medical image visualization techniques. Their presentation covered the rigorous software practices which assure the functionality and stability of those toolkits and explored segmentation and registration algorithms and visualization methods for medical research and applications.

If you missed this workshop at RSNA and are interested in finding out more about Kitware’s open-source tools for medical research and applications, please contact us at kitware@kitware.com. Kitware training courses offer the best opportunity to gain an in-depth understanding of our open source products. To find out about upcoming training courses, workshops and tutorials please contact us at courses@kitware.com.

Kitware is pleased to announce the release of VolView 3.0. VolView is an intuitive, interactive system for volume visualization that allows researchers to quickly explore and analyze complex 3D medical or scientific data on Windows, Mac and Linux computers. Users can easily load and interactively explore datasets using 2D and 3D display methods and tools. 3D tools include volume rendering, maximum intensity projections, and oblique reformating. The ability to save an entire visualization session allows users to easily stop and start sessions. Advanced users can perform custom data processing using a simple plug-in API.

This release is a major upgrade of the VolView platform with several new capabilities including:

• A simplified user interface including a toolbar for quick access to commonly used tools
• Accelerated volume rendering including GPU acceleration on the Nvidia platform
• Easy application, creation, and management of 2D and 3D display presets
• The ability to load and display multiple datasets
• Improved image measurements including a Bezier contour tool
• Significantly improved DICOM file loading

VolView is a stand alone application, but is also a platform for developing more advanced applications for use in clinical trials, by medical device manufacturers and in applications such as the ISP Software (referenced on page 14 of the Source).

We’re working on adding advanced segmentation algorithms to VolView—the first of which is an advanced algorithm for CT Lung Lesion Sizing. These new algorithms will be available with the next release of VolView 3.

To evaluate this product please visit kitware.com/volview and be sure to let us know what you think. Submit your feedback to kitware@kitware.com.

BRAD DAVIS WINS DISSERTATION AWARD
Brad Davis was one of four graduate students selected to receive The University of North Carolina, Chapel Hill 2009 Graduate Dean’s Distinguished Dissertation Award. The Department of Computer Science at UNC nominated Brad Davis for the award based on his dissertation “Medical Image Analysis via Fréchet Means of Diffeomorphisms” which defined new methods for statistical analysis of medical images in non-linear spaces to measure anatomic change and define geometry based averaging and regression of anatomic structures.

The Award is designed to recognize the scholarly contributions of UNC-Chapel Hill doctoral students as revealed through their dissertation projects and to highlight the timely completion of doctoral training. Each graduate program may nominate one doctoral student who has completed his doctoral degree during the academic year. The selection criteria include: originality, innovativeness, scholarly excellence, methodological sophistication and significant contribution to the discipline. His outstanding academic achievement will be celebrated at the Graduate School’s annual Graduate Student Recognition Event on April 20, 2009.

NEW EMPLOYEES
Michel Audette
Dr. Michel Audette joined Kitware in November 2008 as an R&D Engineer and is currently contributing to the IGSTK and ITK projects. Dr. Audette is also conceptualizing several novel
applications which utilize Kitware technology as methods to advance surgical research and development in human and animal subjects. Prior to joining Kitware Dr. Audette researched surgery simulation and model-based surgical guidance at the Innovation Center Computer Assisted Surgery (ICCAS) in Leipzig, Germany. Dr. Audette received his Bachelor's in Electrical Engineering from McGill University, his Master's in Electrical Engineering from École Polytechnique, Montreal and his PhD in Biomedical Engineering from McGill University.

**Arslan Basharat**

Arslan Basharat joined Kitware in January 2009 as an R&D Engineer where he is currently contributing to the VIRAT contract. Prior to joining Kitware Arslan worked in research and development at the Computer Vision Lab at UCF under the supervision of Dr. Mubarak Shah. Arslan received his BS in Computer System Engineering from the GIK Institute (GIKI), his MS in Computer Science from the University of Central Florida (UCF) and is currently completing his PhD in Computer Science from UCF. His research interests lie in the areas of computer vision, machine learning, video processing and activity analysis.

**Naresh Cuntoor**

Dr. Naresh Cuntoor joined Kitware in October 2008 as an R&D Engineer in the Computer Vision group where he is currently contributing the VIRAT contract. Prior to joining Kitware, he worked for the Signal Innovations Group in Durham, North Carolina. He received his BE from the Karnataka Regional Engineering College, Surathkal and his MS and PhD from the University of Maryland, College Park. His research interests are in computer vision, image and video processing, machine learning, topology and differential geometry.

**Jean-Christophe Fillion-Robin**

Jean-Christophe Fillion-Robin joined Kitware’s North Carolina office in January 2009 as an R&D Engineer. Currently J-Chris is contributing to Midas and working as the systems administrator for the North Carolina office. Prior to joining Kitware J-Chris interned at Scynexis, a drug discovery and development company, where he developed a web-based Drug Research information management application to support scientists and facilitate collaboration in their daily work. J-Chris received his Bachelor's in Computer Science from the University Claude Bernard, Lyon and his Master's in Electrical Engineering and Information Processing from ESCPE Lyon.

**EMPLOYMENT OPPORTUNITIES**

Kitware is actively seeking talented software professionals to work with leaders in the fields of computer vision, medical imaging, visualization, 3D data publishing and technical software development. Candidates will actualize and implement leading-edge software solutions in the context of an open-source development model in order to foster extended, collaborative communities and provide flexible, low-cost technical solutions.

By joining our team you will participate in a dynamic work environment with exceptionally talented coworkers and collaborate with esteemed researchers from around the world by:

- providing technical expertise to research, development, software creation and customer support
- designing and developing new visualization, graphics, image processing and computational sciences software
- implementing, documenting, analyzing, creating and modifying computer systems or programs.

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

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](http://www.vtk.org))
- The Insight Segmentation and Registration Toolkit ([www.itk.org](http://www.itk.org))
- ParaView ([www.paraview.org](http://www.paraview.org))
- The Image Guided Surgery Toolkit ([www.itk.org](http://www.itk.org))
- CMake ([www.cmake.org](http://www.cmake.org))
- CDash ([www.cdash.org](http://www.cdash.org))
- KWWidgets ([www.kwwidgets.org](http://www.kwwidgets.org))
- BatchMake ([www.batchmake.org](http://www.batchmake.org))
- VTKEdge ([wwwvtkedge.org](http://wwwvtkedge.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.