A Label Geometry Image Filter for Multiple Object Measurement [1]

Release 0.00

Dirk Padfield, James Miller

August 14, 2008

GE Global Research, One Research Circle, Niskayuna, NY, 12309
{padfield,millerjv}@research.ge.com

Abstract

The itkLabelGeometryImageFilter is a new ITK filter that enables the measurement of geometric features of labeled objects. It calculates features similar to the “regionprops” command of Matlab. It is related to the itkLabelStatisticsImageFilter in that both filters measure features of labeled masks. It differs, however, in that it measures geometric features of the objects themselves rather than statistics of image intensities under the masks defined by the objects. This document describes the mathematical background of the geometric features measured by this filter and describes the framework of the code, which is structured to enable easy expandability as new object features are desired.

Contents

1 Introduction **2**

2 Image Moments **2**

3 Hyper-Ellipsoid Fitting **6**
- 3.1 Covariance Matrix (ND) . . . 7
- 3.2 Eigenvalues and Eigenvectors (ND) . . . 7

4 Calculated Object Features **8**
- 4.1 Volume and Centroid (ND) . . . 8
- 4.2 Axes Lengths (ND) . . . 8
- 4.3 Eccentricity (2D) . . . 8
- 4.4 Elongation (2D) . . . 8
- 4.5 Object Orientation (2D) . . . 8
- 4.6 Bounding Box (ND) . . . 9
- 4.7 Oriented Bounding Box Vertices (ND) . . . 9
- 4.8 Oriented Image Region (ND) . . . 9

[1]This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.

5 Implementation **10**

5.1 Implementation Structure 10

5.2 Inputs and Feature Accessor Methods 11

6 Conclusions **12**

1 Introduction

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 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 such as min, max, and mean intensity, intensity standard deviation, and bounding boxes. The itkLabelGeometryImageFilter, however, measures the geometry of the labeled regions themselves. It measures features similar to the “regionprops” command of Matlab. The set of measurements that it enables along with their definitions are given in Table 1. The first set of features in this table are based solely on the labeled mask itself, whereas the second set, including the integrated intensity and the weighted centroid, are measured on an intensity image under the labeled mask. While the majority of features are measured in ND, some are restricted to 2D by definition (these are explicitly marked as 2D). Much of the notation this table and other Sections is given in 2D for notational simplicity. The definitions and their derivations are given in greater detail in Sections 4. The features listed in this table 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. For example, since the calculation of the eigenvalues/eigenvectors and covariance matrices are already implemented, these measures can be used as the basis for other relevant calculations.

The rest of this paper is organized as follows. Most of the features currently implemented are based on image moments, so Section 2 gives an overview of the relevant mathematical equations. Section 3 introduces the framework of calculations based on a hyper-ellipsoid fitted to the data, and Section 4 describes in detail the calculation of the object features. Section 5 describes some implementation details of the filter and lists the feature accessor methods currently available. Finally, Section 6 lists the conclusions.

2 Image Moments

Image moments are particular averages of either binary objects (unweighted) or their pixel intensities (weighted). They are useful to describe objects and form the building blocks of many useful features of the objects. 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

$$M_{p,q} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)\,dx\,dy \tag{1}$$

where x and y are indices of the first and second dimensions of the function. The discrete counterpart of this

Table 1: **Definitions of label geometry features.** For the shape features, the *I* implicit in the equations represents a binary value set to 1 inside the object and 0 outside. For the shape & intensity features, the *I* represents an intensity value inside the object and 0 outside. All features are calculated in ND except those listed as otherwise. Notational clarifications and expanded definitions are given in Section 4.

Feature Name	**Definition**
Shape Features (I = binary)	
Volume	M_{00}
Centroid	$\left[\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right]$
Eigenvalues	$\lambda_1, \lambda_2, \ldots, \lambda_N$
Eigenvectors	$[\overline{v_0} \quad \overline{v_1} \quad \ldots \quad \overline{v_N}]$
Axes length	$4\sqrt{\lambda_i}$, i = 0,...,D-1
Eccentricity (2D)	$\sqrt{\frac{\lambda_1 - \lambda_0}{\lambda_1}}$
Elongation (2D)	$\frac{\lambda_1}{\lambda_0}$
Orientation (2D)	$\tan^{-1}\left(\frac{\overline{v_1}(1)}{\overline{v_1}(0)}\right)$
Bounding box	[min(X), max(X), min(Y), max(Y), ...]
Bounding box volume	(max(X)-min(X)+1) * (max(Y)-min(Y)+1) * ...
Bounding box size	[(max(X)-min(X)+1), (max(Y)-min(Y)+1), ...]
Oriented bounding box vertices	Bounding box along the major axis of the object
Oriented bounding box volume	Bounding box volume in rotated space
Oriented bounding box size	Bounding box size in rotated space
Rotation matrix	Eigenvectors organized to obey right-hand rule
Shape & Intensity Features (I = intensity)	
Integrated Intensity	M_{00}
Weighted centroid	$\left[\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right]$

function is

$$M_{p,q} = \sum_{y=0}^{Y-1}\sum_{x=0}^{X-1} x^p y^q I(x,y) \tag{2}$$

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 $\left[\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right]$ of the function from the indices. For a 2D continuous function, the central moments are defined as

$$\mu_{p,q} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\overline{x})^p (y-\overline{y})^q f(x,y)dxdy \tag{3}$$

and the discrete version is

$$\mu_{p,q} = \sum_{y=0}^{Y-1}\sum_{x=0}^{X-1} (x-\overline{x})^p (y-\overline{y})^q I(x,y) \tag{4}$$

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

$$\mu_{00} = M_{00} \tag{5}$$
$$\mu_{01} = 0 \tag{6}$$
$$\mu_{10} = 0 \tag{7}$$
$$\mu_{11} = M_{11} - \overline{x}M_{01} = M_{11} - \overline{y}M_{10} \tag{8}$$
$$\mu_{20} = M_{20} - \overline{x}M_{10} \tag{9}$$
$$\mu_{02} = M_{02} - \overline{y}M_{01} \tag{10}$$

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 below.

$$\begin{aligned}\mu_{01} &= \int\int (x-\overline{x})^0 (y-\overline{y})^1 f(x,y)dxdy = \int\int y f(x,y)dxdy - \overline{y}\int\int f(x,y)dxdy \\ &= M_{01} - \overline{y}M_{00} = M_{01} - \frac{M_{01}}{M_{00}}M_{00} = \mathbf{0}\end{aligned} \tag{11}$$

$$\begin{aligned}\mu_{11} &= \int\int (x-\overline{x})^1 (y-\overline{y})^1 f(x,y)dxdy = \int\int (xy - \overline{x}y - x\overline{y} + \overline{x}*\overline{y}) f(x,y)dxdy \\ &= \int\int xy f(x,y)dydy - \overline{x}\int\int y f(x,y)dxdy - \overline{y}\int\int x f(x,y)dxdy + \overline{x}*\overline{y}\int\int f(x,y)dxdy \\ &= M_{11} - \frac{M_{10}}{M_{00}}M_{01} - \frac{M_{01}}{M_{00}}M_{10} + \frac{M_{10}}{M_{00}}\frac{M_{01}}{M_{00}}M_{00} = M_{11} - \overline{x}M_{01} = \mathbf{M_{11}} - \overline{\mathbf{y}}\mathbf{M_{10}}\end{aligned} \tag{12}$$

$$\begin{aligned}\mu_{20} &= \int\int (x-\overline{x})^2 (y-\overline{y})^0 f(x,y)dxdy = \int\int (x^2 - 2x\overline{x} + \overline{x}^2) f(x,y)dxdy \\ &= \int\int x^2 f(x,y)dxdy - 2\overline{x}\int\int x f(x,y)dxdy + \overline{x}^2\int\int f(x,y)dxdy \\ &= M_{20} - 2\frac{M_{10}}{M_{00}}M_{10} + \left(\frac{M_{10}}{M_{00}}\right)^2 M_{00} = \mathbf{M_{20}} - \overline{\mathbf{x}}\mathbf{M_{10}}\end{aligned} \tag{13}$$

The rest of the equations above follow from similar derivations.

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

$$\mu'_{20} = \frac{\mu_{20}}{\mu_{00}} = \frac{M_{20}}{M_{00}} - \bar{x}^2 \tag{14}$$

$$\mu'_{02} = \frac{\mu_{02}}{\mu_{00}} = \frac{M_{02}}{M_{00}} - \bar{y}^2 \tag{15}$$

$$\mu'_{11} = \frac{\mu_{11}}{\mu_{00}} = \frac{M_{11}}{M_{00}} - \bar{x} * \bar{y} \tag{16}$$

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

$$\begin{bmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{bmatrix} \tag{17}$$

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*D. To generalize to ND, we introduce the notation $\mathcal{M}_N^D(v_1, v_2, ..., v_N)$ for raw moments and $\mathcal{C}'^D_N(v_1, v_2, ..., v_N)$ 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, μ'_{00} in the previous notation is $\mathcal{C}'^2_0()$, and μ'_{020} is $\mathcal{C}'^3_2(1,1)$. This notation is more convenient for ND because it doesn't require the explicit listing of a subscript value for each dimension as in μ'_{020} in 3D. This notation is used in Algorithm 1 to calculate the normalized second order central moments.

Algorithm 1 ND Normalized Second Moment Calculation. D = image dimension. N = moments order.

```
1: for i = 0:D-1 do
2:     for j = 0:D-1 do
3:         $\mathcal{C}'^D_2(i,j) = \frac{\mathcal{M}_2^D(i,j)}{\mathcal{M}_0^D()} - \frac{\mathcal{M}_1^D(i)}{\mathcal{M}_0^D()} * \frac{\mathcal{M}_1^D(j)}{\mathcal{M}_0^D()}$
4:         if i == j then
5:             $\mathcal{C}'^D_2(i,j) = \mathcal{C}'^D_2(i,j) + \frac{1}{12}$
6:         end if
7:     end for
8: end for
```

Since the resulting matrix is symmetric, in practice only half of the values need to calculated.

This algorithm simplifies the task of calculating second order moments in ND. Using this algorithm, the covariance matrix of the normalized second order central moments becomes (using the simpler notation using subscript indices)

$$\begin{bmatrix} \mu'_{200} & \mu'_{110} & \mu'_{101} \\ \mu'_{110} & \mu'_{020} & \mu'_{011} \\ \mu'_{101} & \mu'_{011} & \mu'_{002} \end{bmatrix} \tag{18}$$

Notice in line 5 of Algorithm 1 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. The normalized second order central moment of a pixel with unit length of a binary image can be found as (where p means a pixel)

$$M_{00}(p) = \int_0^1 \int_0^1 dxdy = \int_0^1 [x]_0^1 dy = \int_0^1 dy = [y]_0^1 = 1$$

$$M_{10}(p) = \int_0^1 xdx = \left[\frac{1}{2}x^2\right]_0^1 = \frac{1}{2}$$

$$M_{11}(p) = \int_0^1 \int_0^1 xydxdy = \int_0^1 y\left[\frac{1}{2}x^2\right]_0^1 dy = \frac{1}{2}\left[\frac{1}{2}y^2\right]_0^1 = \frac{1}{4}$$

$$M_{20}(p) = \int_0^1 \int_0^1 x^2dxdy = \left[\frac{1}{3}x^3\right]_0^1 = \frac{1}{3}$$

$$\bar{x}(p) = \frac{M_{10}(p)}{M_{00}(p)} = \frac{1}{2} = \frac{M_{01}(p)}{M_{00}(p)} = \bar{y}(p)$$

$$\mu'_{11}(p) = \frac{M_{11}(p)}{M_{00}(p)} - \bar{x}(p) * \bar{y}(p) = \mathbf{0} \tag{19}$$

$$\mu'_{20}(p) = \frac{M_{20}(p)}{M_{00}(p)} - \bar{x}^2(p) = \mathbf{\frac{1}{12}} \tag{20}$$

Thus, the second order cross moment is 0, and the second order moment is $\frac{1}{12}$. A straightforward analysis holds for other dimensions. This explains why it is necessary to add a scalar $\frac{1}{12}$ to each normalized second order central moment when the indices are the same dimension and it is not necessary to add anything for normalized second order central moments consisting of cross moments.

In the next section, these moment calculations will be used as the foundation for many of the object features.

3 Hyper-Ellipsoid Fitting

Several useful features are calculated based on the features of a hyper-ellipsoid fit to each object. A hyper-ellipsoid 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 is at locations (0,b) and (0,-b), and the line connecting the foci to these locations have length a. From the triangle thus formed, it is clear that $f^2 = a^2 - b^2$.

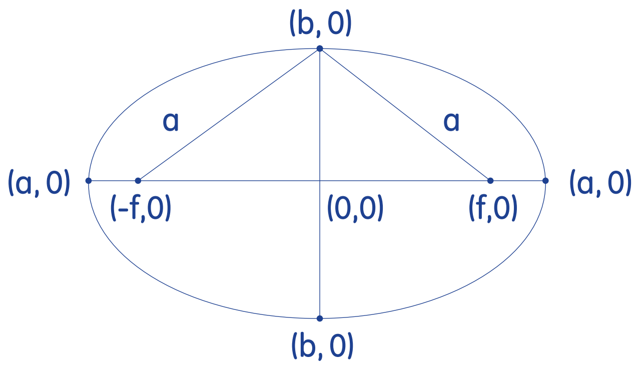

Figure 1: **Ellipse notation illustration.**

These definitions can be expanded to 3D 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 defined the eigenvalues/eigenvectors.

3.1 Covariance Matrix (ND)

The covariance matrix is constructed directly from the normalized second order central moments in ND using the loop defined in Algorithm 1. For example, in 2D, this becomes

$$cov\left(I(x,y)\right) = \begin{bmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{bmatrix} \tag{21}$$

and in 3D it becomes

$$cov\left(I(x,y)\right) = \begin{bmatrix} \mu'_{200} & \mu'_{110} & \mu'_{101} \\ \mu'_{110} & \mu'_{020} & \mu'_{011} \\ \mu'_{101} & \mu'_{011} & \mu'_{002} \end{bmatrix} \tag{22}$$

3.2 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
- $\overline{v_i} =$ eigenvector corresponding to λ_i

For example, in 2D, λ_1 is the eigenvalue along the main axis, and λ_0 is the eigenvalue along the axis perpendicular to the main axis.

Note that, in 2D, the calculation of the eigenvalues can be simplified as

$$\lambda_i = \frac{\mu'_{20} + \mu'_{02}}{2} \pm \frac{\sqrt{4\mu'^2_{11} + (\mu'_{20} - \mu'_{02})^2}}{2} \tag{23}$$

4 Calculated Object Features

4.1 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}, Centroid = $\left[\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right]$. When calculating the unweighted centroid, I is a binary object, and when calculating the weighted centroid, I is the intensity image.

4.2 Axes Lengths (ND)

The length of the axes of the ND hyper-ellipsoid can be found directly from the eigenvalues as

$$4\sqrt{\lambda_i} \tag{24}$$

where i is the index of the eigenvalue from $i = 0, \ldots, D-1$. If the eigenvalues are in increasing order, the axes lengths will correspond to the increasing lengths of the hyper-ellipsoid 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 $b = 2\sqrt{\lambda_0}$.

4.3 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 $\frac{f}{a}$. Since $a = 2\sqrt{\lambda_1}$, $b = 2\sqrt{\lambda_0}$, and $f^2 = a^2 - b^2$

$$\frac{f}{a} = \frac{\sqrt{\text{MajorAxisLength}^2 - \text{MinorAxisLength}^2}}{\text{MajorAxisLength}} = \frac{\sqrt{(4\sqrt{\lambda_1})^2 - (4\sqrt{\lambda_0})^2}}{4\sqrt{\lambda_1}} = \sqrt{\frac{\lambda_1 - \lambda_0}{\lambda_1}} \tag{25}$$

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

4.4 Elongation (2D)

The elongation feature is defined as the ratio of the major axis length to the minor axis length

$$\frac{\text{MajorAxisLength}}{\text{MinorAxisLength}} = \frac{4\sqrt{\lambda_1}}{4\sqrt{\lambda_0}} = \sqrt{\frac{\lambda_1}{\lambda_0}}. \tag{26}$$

4.5 Object Orientation (2D)

In 2D, the eigenvectors $\overline{v}_1$ corresponding to the largest eigenvalue λ_1 corresponds to the major axis of the object, so the orientation can be extracted from the angle between this eigenvector and the origin

$$\theta = \tan^{-1}\left(\frac{\overline{v_1}(1)}{\overline{v_1}(0)}\right) \tag{27}$$

In 2D, this θ can also be simply calculated without calculating the eigenvalues/eigenvectors as

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\mu'_{11}}{\mu'_{20} - \mu'_{02}}\right) \tag{28}$$

4.6 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 [min(X), max(X), min(Y), max(Y)], and in 3D it is [min(X), max(X), min(Y), max(Y), min(Z), max(Z)].

4.7 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 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. The bounding box cannot be transformed directly back to the original space 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 [minX,minY], [minX,maxY], [maxX,minY], [maxX,maxY]. In ND, there will be 2^N 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.

4.8 Oriented Image Region (ND)

The rotation matrix calculated for 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 label image or the intensity image and results in cropped images.

5 Implementation

5.1 Implementation Structure

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, it 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 labels in the input label image to a LabelGeometry structure with the features listed above 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 next loops through all of the labels to calculate the rest of the features, which can be derived from the ones above as described in Section 4. In this iteration, the following values are calculated

- Centroids
- Second order central moments
- Normalized second order cen tral moments
- 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)

5.2 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 the `itkRelabelComponentImageFilter` that has been applied to the connected components of a binary image.

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

All that was needed was to define the input image and call `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.

```
labelGeometryFilter->SetIntensityInput( intensityReader->GetOutput() );
labelGeometryFilter->CalculatePixelIndices();
labelGeometryFilter->CalculateOrientedBoundingBox();
labelGeometryFilter->CalculateOrientedLabelRegions();
labelGeometryFilter->CalculateOrientedIntensityRegions();
```

Here `intensityReader` is the output of a reader of an intensity image.

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

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

The features described in Section 4 can be accessed using the `Get()` methods in Table 2.

The features that are calculated internally that do not have accessor methods are

- First order raw moments
- First order weighted raw moments (if intensity image defined)
- Second order raw moments
- Second order central moments

6 Conclusions

The itkLabelGeometryImageFilter is a filter for measuring features of objects in a labeled image. Several core features are implemented, and the code was designed so that these features can be easily extended as measurements of other features are desired.

Table 2: **Feature Accessor Methods**

unsigned long GetVolume (LabelPixelType)	Return the number of pixels for a label. This is the same as the volume and the zero order moment.
RealType GetIntegratedIntensity (LabelPixelType)	Return the integrated intensity for a label.
LabelPointType GetCentroid (LabelPixelType)	Return the unweighted centroid for a label.
LabelPointType GetWeightedCentroid (LabelPixelType)	Return the weighted centroid for a label.
VectorType GetEigenvalues (LabelPixelType)	Return the eigenvalues as a vector.
MatrixType GetEigenvectors (LabelPixelType)	Return the eigenvectors as a matrix.
AxesLengthType GetAxesLength (LabelPixelType)	Return the axes length for a label.
RealType GetMinorAxisLength (LabelPixelType)	Return the minor axis length for a label. This is a convenience class that returns the shortest length from GetAxesLength.
RealType GetMajorAxisLength (LabelPixelType)	Return the major axis length for a label. This is a convenience class that returns the longest length from GetAxesLength.
RealType GetEccentricity (LabelPixelType)	Return the eccentricity for a label.
RealType GetElongation (LabelPixelType)	Return the elongation for a label.
RealType GetOrientation (LabelPixelType)	Return the orientation for a label defined in radians.
BoundingBoxType GetBoundingBox (LabelPixelType)	Return the computed bounding box for a label. This is organized in min/max pairs as [min(X), max(X), min(Y), max(Y),...]
RealType GetBoundingBoxVolume (LabelPixelType)	Return the volume of the bounding box.
LabelSizeType GetBoundingBoxSize (LabelPixelType)	Return the size of the bounding box.
LabelIndicesType GetPixelIndices (LabelPixelType)	Return all pixel indices for a label.
BoundingBoxVerticesType GetOrientedBoundingBoxVertices (LabelPixelType)	Return the oriented bounding box vertices. The order of the 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 [minX,minY], [minX,maxY], [maxX,minY], [maxX,maxY].
RealType GetOrientedBoundingBoxVolume (LabelPixelType)	Return the volume of the oriented bounding box.
LabelPointType GetOrientedBoundingBoxSize (LabelPixelType)	Return the size of the oriented bounding box.
MatrixType GetRotationMatrix (LabelPixelType)	Return the rotation matrix defined by the eigenvalues/eigenvectors.
RegionType GetRegion (LabelPixelType)	Return the region defined by the bounding box.
TLabelImage *GetOrientedLabelImage (LabelPixelType)	Return the label region defined by the oriented bounding box.
TIntensityImage *GetOrientedIntensityImage (LabelPixelType)	Return the intensity region defined by the oriented bounding box.