Tumor Volume Measurement and Volume Measurement Comparison Plug-ins for ITK

DEVELOPER'S MANUAL
Rev.0.1

Imaging Science and Information Systems Center
Department of Radiology
Georgetown University
2115 Wisconsin Ave NW, Suite #603
Washington, DC 20007

1. Introduction

The volume mesurement and comparison library consist in two components:first part asses volume estimation using counting techinques and second part consist in volume comparisons that has the role of validating different segmentation techinques.

2. Volview Plug-in Architecture

The Volview plug-in architecture defines a mechanism to specify the number and types of parameters for the plug-in that are required from the user. Volview create GUI elements for those parameters and passes the corresponding values to the plug-in. This architecture is shown in Figure 1.

For developing plug-ins we had to implement three Volview methods:

-The Init() function. This function defines the fundamental characteristics of the plug-in. These characteristics include its name, group, terse documentation, the number of GUI items needed for passing parameters from the user, and an estimation of bytes per voxel required for processing the image.

Example:

```
extern "C" {
void VV_PLUGIN_EXPORT vvITKVolumeInit(vtkVVPluginInfo *info)
{
  vvPluginVersionCheck();
  // setup information that never changes
  info->ProcessData = ProcessData;
  info->UpdateGUI   = UpdateGUI;
  info->SetProperty(info, VVP_NAME, "Volume(ITK)");
  info->SetProperty(info, VVP_GROUP, "ISIS - Volume comparison");
  info->SetProperty(info, VVP_TERSE_DOCUMENTATION,
                            "Compute Volume");
  info->SetProperty(info, VVP_FULL_DOCUMENTATION,
    "This module calculate Volume.");
  info->SetProperty(info, VVP_SUPPORTS_IN_PLACE_PROCESSING, "0");
  info->SetProperty(info, VVP_SUPPORTS_PROCESSING_PIECES,   "1");
  info->SetProperty(info, VVP_NUMBER_OF_GUI_ITEMS,          "1");
 info->SetProperty(info,  VVP_REQUIRED_Z_OVERLAP,           "0");
  info->SetProperty(info, VVP_PER_VOXEL_MEMORY_REQUIRED,    "1");
}
```

-The UpdateGUI() function. This function defines all the properties of the GUI items. This includes their text labels, their type(scale,number,check box,etc) their default value, their range of values, and a short help message indicating the role of this parameter.

```
static int UpdateGUI(void *inf)
{
```

```
    vtkVVPluginInfo *info = (vtkVVPluginInfo *)inf;

    //info->SetProperty(info, VVP_REQUIRED_Z_OVERLAP, "0");
    info->OutputVolumeScalarType = info->InputVolumeScalarType;
    info->OutputVolumeNumberOfComponents = info-
                >InputVolumeNumberOfComponents;
    memcpy(info->OutputVolumeDimensions,info-
                >InputVolumeDimensions,
                3*sizeof(int));
    memcpy(info->OutputVolumeSpacing,info->InputVolumeSpacing,
                3*sizeof(float));
    memcpy(info->OutputVolumeOrigin,info->InputVolumeOrigin,
                3*sizeof(float));
    return 1;
}
```

-The ProcessData() function. This is where the ITK pipeline of the plug-in is actually executed. It usually involves creating the ITK pipeline, gathering all the parameters from the GUI, passing them to the ITK pipeline, and triggering the execution of the pipeline.

```
template <class InputPixelType>
void vvVolumeTemplate(vtkVVPluginInfo *info,
                vtkVVProcessDataStruct *pds,
                InputPixelType *)
{
  typedef itk::Image< InputPixelType, 3 > InputImageType;
  typedef  itk::StatisticsImageFilter<InputImageType>     FilterType;
  typedef  VolView::PlugIn::FilterModule< FilterType>   ModuleType;

  char tmp[1024];
  ModuleType  module;

  module.SetUpdateMessage("Computing Volume...");
  module.SetPluginInfo( info );
  module.InitializeProgressValue();
  module.ProcessData( pds  );
  InputImageType::SpacingType spacing =module.GetInput()->GetSpacing();
  double factor = 1.0;
  for(unsigned int d=0; d<3; d++)
   {
   factor *= spacing[d];
   }
  itk::StatisticsImageFilter<InputImageType>::RealType Volume=
module.GetFilter()->GetSum();
  sprintf(tmp,"%f mm^3",Volume*factor);
  info->SetProperty( info, VVP_REPORT_TEXT, tmp );
```

```
  InputImageType::ConstPointer outputImage= module.GetInput();

    typedef itk::ImageRegionConstIterator< InputImageType >
OutputIteratorType;

    OutputIteratorType ot( outputImage, outputImage->GetBufferedRegion() );
    ot.GoToBegin();
        InputPixelType * outData = (InputPixelType *)(pds->outData);

     while( !ot.IsAtEnd() )
      {
      *outData = ot.Get();  // copy output pixel
      ++outData;
      ++ot;
      }
}

//-------------------------------------------------------------

static int ProcessData(void *inf, vtkVVProcessDataStruct *pds)
{
 vtkVVPluginInfo *info = (vtkVVPluginInfo *)inf;

 try{
 switch (info->InputVolumeScalarType)
  {
  // invoke the appropriate templated function
  vtkTemplateMacro3(vvVolumeTemplate, info, pds,
              static_cast<VTK_TT *>(0));
  }
 }
 catch( itk::ExceptionObject & except )
 {
        info->SetProperty( info, VVP_ERROR, except.what() );
        return -1;
 }
 return 0;
}
```

3. Volume mesurement component

Following is a description of the module followed by its programming interface.

3.1.Voxel count

Description

It involves counting the number of voxels and then adjusting this number by the voxel volume. This method provides a coarse volume estimation.

3.2.AntiAliased Voxel count

Description

In this implementation, the AntiAliasBinaryImageFilter of ITK is used as a preprocesing stage. This step was necessary to reduce aliasing artifacts which result in visualization of binary partitioned surfaces. The resulting image from the antialias filter contains some border voxels that have a partial contribution to the total volume. The output of the anti-alias filter is thresholded to the mid-value of the range of voxel values to get rid of the voxels which fall below this threshold value. Then we use the StatisticsImageFilter of ITK to obtain the sum of all the voxel values. This sum is normalised with the maximum voxel value in the image and multiplied by the unit voxel volume. This second method attempts to compensate for partial volume effects in the evaluation of the full volume.

4. Volume comparison components

The volume comparison part of the library is implementing different metrics to mesure the differences of segmentation results from intra-rater,inter-rater and automatic-manual segmentations.Furher it is possible with the help of Volview enviroment to display volumetric images overlaid with segmentation with 3D distances.

4.1.Volumetric Overlap (true and false positives, true and false negatives)

Description

The SimilarityIndexImageFilter measures the spatial overlap between two segmentations, A and B target regions, and is defined as

$$S = \frac{2|A \cap B|}{|A| + |B|}$$

where A and B are respectively the set of non-zero pixels in the first and second input images. Operator $|\cdot|$ represents the size of a set and $\cap$ represents the intersection of two sets. (Fig 1).

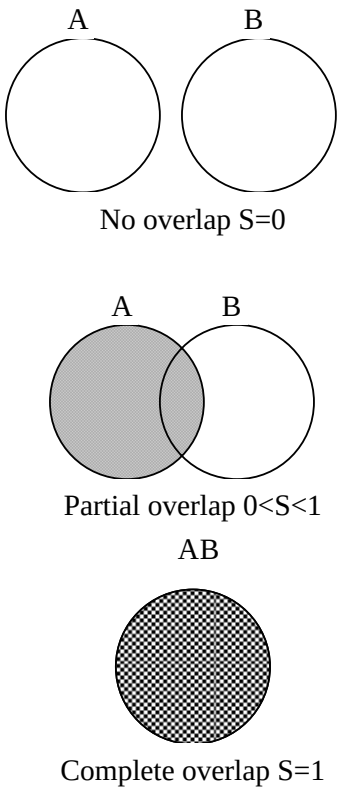

Fig 1. Similarity coefficient representing spatial overlap and reproducibility

It is comparing sets of non-zero pixels from two binary image segmentations by relative overlap.The measure is derived from a reliability measure known as the kappa statistic. It assume both image have the same number of dimensions.

SimilarityIndexImageFilter is sensitive to both differences in size and in location and have been in the literature for comparing two segmentation masks.

4.2. Maximum Surface Distance (Hausdorff distance)

Description

The Hausdorff-Chebyshev metric defines the largest difference between two contours.

If we have two sets of points $A = \{a_1,...,a_m\}$ and $B = \{b_1,...,b_n\}$

$$h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$

where h(A,B) is directed Hausdorff distance and *A* and *B* are respectively the set of non-zero pixels in the first and second input images. It identifies the point $a \in A$ that is farthest from any point of *B* and measures the distance from *a* to the nearest neighbor in *B*. Note that this function is not is not symmetric and hence is not a true distance. In

other words, the directed Hausdorff Distance can be viewed as the maximum distance from any point in A to its nearest point in B.

In particular, this filter uses the DanielssonDistanceMapImageFilter inside to compute distance map from all non-zero pixels in the second image
After that it use this 3D Euclidean distance map of the first object and overlay of the second object to efficiently calculate the measure and then find the largest distance (in pixels) within the set of all non-zero pixels in the first image.The measure is extremely sensitive to outliers and does not reflect properties integrated along the whole boundary or surface. In certain cases, however, where a procedure does have to stay within certain limits, this measure would be the metrics of choice.

The directed Hausdorff metric is nonsymmetric. A symmetric version is defined as:

$$H(A,B) = \max(h(A,B), h(B,A))$$

The full Hausdorff-Chebyshev metric defines the largest difference between two contours.

4.3. Mean absolute surface distance

Description

This filter extracts the contour of the second volume and computes the average distance between the first tumor and this surface.

The volumetric overlap metric compares sets of non-zero pixels from two binary image segmentations for relative overlap. This measure is derived from a reliability measure known as the kappa statistic. This reliability measure is implemented in an ITK filter called SimilarityIndexImageFilter. The SimilarityIndexImageFilter is sensitive to both differences in size and in location and has been described in the literature for comparing two segmentation masks. The measure gives a score of 1 for perfect agreement and 0 for complete disagreement. The overlap measure depends on the size and the shape complexity of the object.