# Seamless VTK-ITK pipeline connection for image data handling

*Release 1.00*

Andinet Enquobahrie and Luis Ibanez

March 27, 2007

Kitware Inc.
andinet.enqu@kitware.com,luis.ibanez@kitware.com

**Abstract**

Often times, developers face the need to connect ITK and VTK pipelines to handle image data. For this purpose, image data importer and exporter classes have been implemented both in ITK and VTK. However, as evident from frequent questions in the ITK and VTK users and developers mailing list, usage of these classes is not straight forward. Furthermore, the difficulty gets more challenging with the need to handle different pixel types. For these reasons, an effort was made to write Image IO classes that encapsulate all the intricacies and provide users with easy to use API. This article describes these Image IO classes and present an example to demonstrate how to use these classes.

## Contents



## 1 Introduction

Application developers often need to connect VTK and ITK piplelines for their image analysis and visualization needs. Both ITK and VTK toolkits provide importer and exporter filters for this purpose. However, their usage is not straight forward. Furthermore, users of ITK toolkit are often challenged by the need to instantiate image reader classes without knowing first the pixel type of the image data.

## 2 Class descriptions

To solve the above issues, two classes were implemented: vtkKWImage and vtkKWImageIO.

### 2.1 vtkKWImage

This class is a container class of an image object. The class associates an internal ITK image and a VTK importer in such a way that the image data is available in both ITK and VTK image data format.

### 2.2 vtkKWImageIO

This class reads the image data using ITK object factory pattern and stores it in a vtkKWImage image object. The image reading process is broken down to three steps. In the first step, the pixel type is determined. Next, an appropriate ITK image reader is instantiated using the determined pixel type. Finally, the image data is read and stored into a vtkKWImage container object. From the vtkKWImage container object, both the ITK and VTK image data can be easily accessed.

## 3 Example

This program is a modified version of the example application in InsightApplications. The program demonstrates how easily vtkKWImage and vtkKWImageIO classes can be used to load an image and render it using VTK filters without the need to instantiate importer, and exporter filters and without the need to know the pixel type to instantiate the appropriate image reader.

```
#include "itkCommand.h"
#include "itkImage.h"
#include "vtkImageData.h"

#include "vtkKWImageIO.h"
#include "vtkKWImage.h"

#include "vtkImageActor.h"
#include "vtkRenderer.h"
#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkInteractorStyleImage.h"

int main(int argc, char * argv [] )
{

  if( argc < 2 )
    {
    std::cerr << "Missing parameters" << std::endl;
    std::cerr << "Usage: " << argv[0] << " inputImageFilename " << std::endl;
    return 1;
```

```
  }

vtkKWImageIO * reader = vtkKWImageIO::New();

reader->SetFileName( argv[1] );

try
  {
  reader->ReadImage();
  }
catch( itk::ExceptionObject & excp )
  {
  std::cerr << excp << std::endl;
  return EXIT_FAILURE;
  }

vtkKWImage * kwImage = reader->HarvestReadImage();
vtkImageData * vtkImage = kwImage->GetVTKImage();
vtkImageActor* actor = vtkImageActor::New();
actor->SetInput( vtkImage );

vtkInteractorStyleImage * interactorStyle = vtkInteractorStyleImage::New();

// Create a renderer, render window, and render window interactor to
// display the results.
vtkRenderer* renderer = vtkRenderer::New();
vtkRenderWindow* renWin = vtkRenderWindow::New();
vtkRenderWindowInteractor* iren = vtkRenderWindowInteractor::New();

renWin->SetSize(500, 500);
renWin->AddRenderer(renderer);
iren->SetRenderWindow(renWin);
iren->SetInteractorStyle( interactorStyle );

// Add the vtkImageActor to the renderer for display.
renderer->AddActor(actor);
renderer->SetBackground(0.4392, 0.5020, 0.5647);

// Bring up the render window and begin interaction.
renWin->Render();
iren->Start();

// Release all VTK components
actor->Delete();
interactorStyle->Delete();
renWin->Delete();
renderer->Delete();
iren->Delete();
```

```
  return 0;
}
```

## References

[1] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.

[2] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit, An Object Oriented Approach to 3D Graphics*. Kitware Inc, 1998.