In-Class Lab Activity: Introduction to ITK / VTK Code Structure

Andrew Hahn (adhahn@wisc.edu)

Set up

- Will need virtual box to import the virtual machine with the itk and vtk code and compiler
  - https://www.virtualbox.org/wiki/Downloads
- ITK-SNAP free toolbox with some itk functionality in an easy to use GUI
  - http://www.itksnap.org/pmwiki/pmwiki.php
- ITK is very well documented and has an extensive library of examples and a Wiki
  - https://itk.org/ITK/help/documentation.html

VirtualBox

- Make sure to install the VirtualBox Extension Pack
  - Should be prompted to install when you open VirtualBox
  - Otherwise, it can be downloaded from the page linked on the previous slide
    - It can be manually installed by selecting File->Preferences… and selecting the Extensions tab

- Import the MP574 VM appliance
  - File->Import Appliance…
  - Select MP574.ova
  - Click Next, and then Import

Sharing Between the Host and Guest

- Create a folder on the host (the machine running VirtualBox) called vbox_shared
  - This will be the host-side shared directory
- Boot up the virtual machine, then select Machine->Settings...
  - Select the Shared Folders tab
- Select Transient Folders and add a new share
  - Browse to the host-side directory under Folder Path:
  - Enter vbox_shared for Folder Name and hit “OK”

Sharing Between the Host and Guest

- Open the Terminal Program (on the icon bar to left of screen)

- Type: sudo mount –t vboxsf vbox_shared ~/shared
  - When prompted for password, enter $mpuser

- Screenshots
  - You can use the screenshot application to record the output of the ITK exercises
  - You can search for it using the Show Applications functions (button on lower right of the screen).

Text Editor

- Editing text can be done with the provided Text Editor program or with vim from within the terminal
- You are welcome to install a text editor of your choice as well

C++ Basics

Write Main Function

- Every C++ program requires a **main** function, which defines the starting point of the program
- The structure of the main function is always the same
- It includes a list of command line arguments (argc = number of input arguments, argv = argument list)

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

        // Enter program here

        return 0;
}
```

Classes in C++

- Classes describe properties and methods of an object
- Objects are instances of classes
- Objects of the same class can have different property values

```
class MyClass
{
  int PrivateMemberVariable;
private:
  // Private Member Variables and Functions
protected:
  // Protected Member Variables and Functions
public:
  float PublicMemberVariable;
  // Constructor
  MyClass(int value1, float value2)
    : PrivateMemberVariable( value1 )
  {
    PublicMemberVariable = value2 + float(value1);
  }
  int GetPrivateVariable() const
  {
        return PrivateMemberVariable;
  }
};
```

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

  MyClass instance(5, 1.3f);

  cout << instance.GetPrivateVariable() << endl;
  cout << instance.PublicMemberVariable << endl;

  return 0;
}
```

Header (.h) and Source (.cpp) Files

Headers contain descriptions of classes and there member variables and functions

```
class MyClass
{

  int var;

public:

  // Constructor declaration
  MyClass(int value);

  // Member function declaration
  int GetValue() const;
};
```

Source files contain the implementation of the classes member functions

```
#include "MyClass.h"

// Constructor implementation
MyClass::MyClass(int value)
: var(value)
{
}

// Member function implementation
int MyClass::GetValue() const
{
return this->var;
}
```

Namespaces

- Namespaces define an area of code in which all identifiers are guaranteed to be unique

```
namespace MyFunctions
{
  int add(int a, int b) { return a + b; }
  int sub(int a, int b) { return a - b; }
}
```

```
int main(int argc, const char *argv[])
{
  int res = MyFunctions::add(1, 2);
  return 0;
}
```

- Access functions in a namespace using the “::” operator
- … or “using” statement

```
using MyFunctions::sub;
using namespace MyFunctions;

int main(int argc, const char *argv[])
{
  int res = add(1, 2);
  int res2 = sub(2, 1);
  return 0;
}
```

Standard Template Library (STL)

Abstraction of Types and Behaviors

```
#include <vector>

std::vector< T >
std::vector< int >
std::vector< double >
std::vector< char * >
std::vector< Point >
std::vector< Image >
```

Typedef / Using

Typedef / **using** allows the programmer to create an alias for a data type, and use the aliased name instead of the actual type name

```
typedef std::vector< int > VectIntType;
using VectIntType = std::vector<int>;


VectIntType vec;
vec.push_back(5);
```

Dynamic memory allocation

```
int *pnValue = new int[2];      // dynamically allocate an array of integers

pnValue[0] = 7;                 // assign 7 to first element
pnValue[1] = 3;                 // assign 3 to second element

delete pnValue;                 // unallocate memory assigned to pnValue
```

When we are done with a dynamically allocated variable, we need to explicitly tell C++ to free the memory for reuse.

Smart pointer

```
FilterType::Pointer filter = FilterType::New();
ImageType::Pointer image = filter->GetOutput();
```

NO NEED FOR

```
filter->Delete();
```

Do not call Delete() in an ITK smart pointer

ITK Basics

ITK Basics

- Strongly templated
  - Use typedef
- Common classes in ITK
  - itk::Object
    - Base class implements smart pointer
  - itk::Image
    - Represents an image or volume
  - itk::ImageToImageFilter
    - Filters are classes which operate on images

```
include "itkImage.h“

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

  typedef shortPixelType;
  const   unsigned int    Dimension = 2;
  typedef itk::Image< PixelType, Dimension >
        ImageType;

  ImageType::Pointer image = ImageType::New();

}
```

ITK Basics: Filters

1. Typedef filter type
2. Create filter object
3. Define input(s)
4. Set additional parameters
5. Call member function Update()

- Input of a filter can be the output of a different filter
- Update() only has to be called on the last filter in a filter chain

Reading an RGB Image

```
#include <itkImageFileReader.h>
#include <itkPNGImageIO.h>
#include <iostream>

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

  typedef shortPixelType;
  const   unsigned int        Dimension = 2;
  typedef itk::Image< PixelType, Dimension >
          ImageType;
  typedef itk::ImageFileReader< ImageType >
          ReaderType;
  typedef itk::PNGImageIO ImageIOType;

…
```

```
…

  ImageIOType::Pointer imageIO =
         ImageIOType::New();
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetImageIO(imageIO);

  const char * inputFilename = argv[1];
  reader->SetFileName(inputFilename);
  try
  {
    reader->Update();
  }
  catch (const itk::ExceptionObject &ex)
  {
    std::cout << ex.what() << std::endl;
  }
}
```

Reading Dicom Image

```
#pragma warning(disable : 4996)
#include <itkImageFileReader.h>
#include <itkGDCMImageIO.h>
#include <iostream>

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

  typedef shortPixelType;
  const   unsigned int        Dimension = 2;
  typedef itk::Image< PixelType, Dimension >
          ImageType;
  typedef itk::ImageFileReader< ImageType >
          ReaderType;
  typedef itk::GDCMImageIO ImageIOType;

  ImageIOType::Pointer imageIO =
          ImageIOType::New();
  ReaderType::Pointer reader = ReaderType::New();
          reader->SetImageIO(imageIO);

…
```

```
…

  const char * inputFilename = argv[1];
  reader->SetFileName(inputFilename);
  try
  {
    reader->Update();
  }
  catch (const itk::ExceptionObject &ex)
  {
    std::cout << ex.what() << std::endl;
  }
  ImageType::SpacingType spacing = reader->
          GetOutput()->GetSpacing();
  std::cout << spacing[0] << ", " << spacing[1] <<
          std::endl;
}
```

Display Image Using VTK

```cpp
#pragma warning(disable : 4996)
#include <vtkAutoInit.h>
VTK_MODULE_INIT(vtkRenderingOpenGL2)
VTK_MODULE_INIT(vtkInteractionStyle)
#include <itkImageToVTKImageFilter.h>
#include <vtkImageViewer2.h>
#include <vtkRenderWindowInteractor.h>

…

typedef itk::ImageToVTKImageFilter< ImageType>
         ConnectorFilterType;

ConnectorFilterType::Pointer connector =
         ConnectorFilterType::New();

vtkImageViewer2 * viewer = vtkImageViewer2::New();
vtkRenderWindowInteractor * renderWindowInteractor =
         vtkRenderWindowInteractor::New();
```

```cpp
…

connector->SetInput(reader->GetOutput());

viewer->SetSize(256, 256);
viewer->SetupInteractor(renderWindowInteractor);
connector->Update();

viewer->SetInputData(connector->GetOutput());
viewer->SetColorWindow(192);
viewer->SetColorLevel(96);

viewer->Render();
renderWindowInteractor->Start();
```

Compiling Code

- Each exercise has the same directory structure
  - /home/mpuser/MP574/X, where X is the exercise name, e.g. 01_ImageViewer
    - This folder contains the exercise code and the CMakeLists.txt file (used to configure the build)
  - /home/mpuser/MP574/X/build
    - This is were the executable will be built
- Building the executable
  - Open the terminal
  - >> cd /home/mpuser/MP574/X/build
  - >> cmake ..
  - >> make
  - Ignore warnings

Running Executables

- >> cd /home/mpuser/MP574/X/build
- >> ./Exercise_N xx yy
  - N is the exercise number
  - xx and yy are the inputs (some only have 1 input)

Exercise 1

1. Read a DICOM image
   - /home/mpuser/MP574/ITK_Images/lungImage2.dcm
2. Display the image using VTK
3. Print meta information
   - Uses MetaDataDictionary
4. Display another image
   - /home/mpuser/MP574/ITK_Images/lungImage.dcm
   - Can you determine why it looks different and how the code can be changed to fix it?
5. Modify and recompile the code to read in a PNG image filetype
   - Will have to comment out the reading of meta information
   - Use code in 03_MorphologicalWatershed for refererence