Lecture 22
Misc. ITK Stuff + GUI

Methods in Medical Image Analysis - Spring 2015
BioE 2630 (Pitt) - 16-725 (CMU ECE)
18-791 (CMU ECE) - 42-735 (CMU BME)
Dr. John Galeo2

Based on Shelton’s slides from 2006

Part 1: Generic Programming Revisited

- Generic programming may be loosely defined as “programming with concepts” (David Musser)
- A lot of what I’ll be talking about I’ve mentioned in passing before; hopefully you have a different perspective after having written some code.
- What concepts have we looked at so far?

Concept of an image

- An image is a rectilinear container in N-space which holds regular samples of some physical space
- Each of these regular samples is called a pixel

Concept of a pixel

- A pixel is a sample of data in N-space, and may be represented by a variety of data types depending on the modality used to acquire the data
- Remember, in ITK it is perfectly valid to use an arbitrary class as a pixel type.
- Ex: each pixel is a C++ object containing two floats, one integer, and a linked list of indices.

Concept of an iterator

- An iterator is a way to move over an image; it provides a method that mimics “sequential” access regardless of the actual access method or dimensionality

Concept of a pipeline

- There are two main types of objects in the world, data objects and process objects
- Typically, we feed a data object to a process object and get a new data object as a result
- A sequential chain of process objects is called a pipeline
Writing generic code

- Successful generic programming means that you "ignore" concerns that would be specific to a particular image
  - pixel type (i.e. the actual data type)
  - dimensionality
- The first way you do this is with templating

Questions to ask yourself

- Am I making tradeoffs between:
  - Speed of coding and reusability?
  - Level of generality and execution speed?
  - Compactness and clarity?
- ITK seems to lean towards reusable, generic, and clear code
  - Depending on your needs this may be a criticism or a point in favor of the toolkit
- SimpleITK seems to lean toward rapid coding, somewhat generic, and clarity.

When generic programming fails

- As much as we'd like, not all algorithms extend to N-d or to all pixel types
- But... Don't assume that things won't work in higher (or lower) dimensions
  - Surprisingly discovered that code written to do medial axis analysis in 3D worked correctly on a 4D hypersphere

Part 2: Open Source & Cross Platform Development

- Successfully managing any software project is not an easy job
- There are a lot of tools out there that make your life easier
- Some we've talked about, some we haven't

Tool 1: CMake

- You should now be very familiar with CMake, what it does, and why it's important
- CMake is a pretty unique tool—be sure to keep it in mind for future projects
Tool 2: Revision control

- You're now familiar with basic SVN / CVS operations
- Allows many developers to change a common code base simultaneously
- Revision control of ASCII text files is easy; binary files are harder

CVS

- CVS server configuration:
  - pserver - relatively easy to configure on the client side, low security
  - SSH - high security, client & server configuration a bit trickier
- Benefits:
  - Traditional, or “reference,” version control software—many different client applications
- Problems:
  - Security
  - Some odd side effects of versioning—can’t delete directories

Subversion (SVN)

- A newer CVS alternative, core syntax is identical
- Server configuration:
  - Apache web server - can use https for security
  - Enables several special features
  - svnserve - custom server
  - svnserve & ssh - custom server with encryption layer
  - Local file access - useful when everyone has easy access to the file system. Common for personal use when stored in your home directory.

Subversion (SVN)

- Benefits:
  - More capable versioning core - directory deletion a legal operation
  - More flexible configuration options
  - Fewer security issues
- Problems:
  - Fewer clients available (especially as part of integrated build environments)
    - Windows
    - Mac
    - Command line (most other Unixes)

Version control interaction via the web

- Use CVSWeb to provide HTML access to a CVS repository
- Subversion supports native HTML access to the repository
- Useful for viewing differences between file versions

Git

- “Biggest and Best” open-source revision control
- Allows Branches in the revision “tree,” e.g.:
  - Start with a master branch (think of a tree trunk)
  - One group works on a branch for updating DICOM
  - Another group works on a branch for GPU updates
  - Someone else works on a branch for Deconvolution
- Each branch can be merged into “master” when it is ready
  - Much cleaner merging and conflict management for large project chunks
  - Master can be kept very stable
Tool 3: Code testing

- How to keep track of multiple OS's and continuous revision of a common codebase?
- Option 1... don't worry
- Option 2... use a testing framework

DART

- Dashboard generator
- Coordinates builds and tests from multiple sources
- Client produces build/test results in XML
- Server converts XML to HTML web pages
- Supports continuous build process

DART, cont.

<table>
<thead>
<tr>
<th>Tool 4: Graphics Toolkits/Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Low level: OpenGL</td>
</tr>
<tr>
<td>• Scientific visualization: VTK</td>
</tr>
<tr>
<td>• Other toolkits exist which are a bit more geared towards game development</td>
</tr>
<tr>
<td>• Plib</td>
</tr>
<tr>
<td>• SDL</td>
</tr>
<tr>
<td>• Fonts: Freetype &amp; FTGL</td>
</tr>
</tbody>
</table>

GUI toolkits

- GLUT - an older but still useful very lightweight toolkit for GL development
- FLTK - my old favorite (LGPL, unless statically linked); not the prettiest, but does what you need
- Mac users should only use FLTK versions ≥ 2.3.0
- QT - the luxury swiss army approach to cross platform GUIs (but LGPL license restrictions)
- wxWidgets - Very cross-platform (also available as wxPython). I may switch to this (versions 3.x when available or possibly ≥ 2.9.3 now). Essentially LGPL, but without restrictions on "binary object code" based on it. Be sure to also grab wxVTK, and consider getting a GUI builder from wiki.wxwidgets.org/Tools. |
- kwWidgets - also very cross-platform, and also supports Python, but may have small issues on Macs; no graphical GUI-building tool; C++ layer on top of Tcl/Tk; http://www.kwwidgets.org/ |

Tool 5: Documentation

- Doxygen is the gold standard
- Easy to integrate with your web server to automatically generate fresh documentation each night
Tool 6: Bug tracking

- May or may not be useful depending on the number of users you have
- First line of defense is a mailing list
- If that's not adequate, try JIRA

GitHub

- An online web system for handling git
  - Git hosting
  - Programmer collaboration
  - Code review—is this git branch ready?
  - Other code management
  - Can be tied to Jira: which bugs does this branch “fix”?
- Used by ITK, Linus Torvalds (Linux), etc.

What level to buy in at?

- Solo project - CVS/SVN, Doxygen documentation
- Small lab
  - SVN web access (or possibly Git)
  - Manual build tests
  - Fix problems via a mailing list
- Large lab/several labs
  - Git (possibly GitHub)
  - Automated build testing
  - JIRA bug tracking

Unless you have a good reason, always…

- Program with multiple OS's in mind
- Test on alternative OS's if practical
- Document, document, document
- Store your code using revision control

Part 3: Visualization

- I’ll present a brief summary of how VTK works
  - It’s very likely that some of you will want to use VTK in stand-alone mode, e.g. ITK + VTK, but no menus, buttons, etc.
- I’ll also go over the highlights of using ITK+VTK +FLTK
  - Using VTK with MFC, Qt, wxWidgets, etc. is similar.

The Visualization Toolkit (VTK)

- A very large open source project run by Kitware
- Many visualization tools: surface extraction, volume rendering, data display, etc.
- We’re interested in its ability to integrate with ITK to display 3D images
- Remember, ITK was intentionally designed without visualization tools
Rendering images in VTK

- There are numerous ways to show 3D image data in VTK
- One of my favorites is `vtkImagePlaneWidget`.

Linking ITK to VTK

- Both ITK and VTK use pipelines
- You can interconvert ITK and VTK update calls so that modifying your image (in ITK) will cause the visualization to update (in VTK)
- Easier with ITKv4:
  - When configuring an ITK build, include `ITKVtkGlue` module
  - Use `itk::ImageToVTKImageFilter` to adapt an ITK image into a VTK image from an ITK image

Easier with ITKv4:

- Fully interconnects ITK and VTK data pipelines
- http://www.itk.org/Doxygen/html/classitk_1_1ImageToVTKImageFilter.html
- Be sure to look at the referenced examples

itk::ImageToVTKImageFilter

- Adapts an ITK image into a VTK image from an ITK image
- Fully interconnects ITK and VTK data pipelines
- http://www.itk.org/Doxygen/html/classitk_1_1ImageToVTKImageFilter.html
- Be sure to look at the referenced examples

Actors in VTK

- The basic “thing” that can be displayed in VTK is an `Actor`
- Actors use `Mappers` to convert raw data
- For example:
  - boundary points in ITK → VTK pointset → VTK polygon data mapper → VTK actor

Rendering layout of VTK

- `vtkRenderWindow` defines the window that is displayed on your monitor
- `vtkRenderer`
  - `vtkRenderer`s are attached to windows
  - They are responsible for converting `vtkActors` into displayed images

vtkRenderWindowInteractors

- `Interactors` pass mouse and keyboard events back and forth
- Work with `RenderWindow`s
- VTK has different Render-Window Interactors for many common platforms:
  - Win32
  - Mac Cocoa
  - iOS
  - Android
  - OpenGL
  - QT (QVTKInteractor)
  - ...
To summarize: process flow

Widgets

- Widgets are more complicated objects that combine some of the functionality of Interactors and Actors
- The ImagePlaneWidget is a mouse-controlled object that provides an arbitrary slice through a 3D volume

Adding “stuff” to your images

- It’s easier than you might think to render additional objects along with the image plane widgets (boundary points for instance)
- Starting with some sort of object in ITK, you would do the following...

Arbitrary object visualization

1. Figure out what type of primitive you have (points/lines/etc.)
2. Create VTK data representing your primitives
3. Convert this to poly data
4. Map this to an actor

Data representation in VTK

- For geometric data, you may be interested in the following classes:
  - vtkPoints stores a list of 3D points
  - vtkUnstructuredGrid stores a collection of arbitrary cells, where a cell is a primitive such:
    - vtkVertex
    - vtkLine
    - vtkTriangle
    - vtkPolyhedron
    - _

Data representation cont.

- This may seem a bit convoluted, but in practice it’s pretty simple once you get the hang of it
- VTK has a pipeline, similar to that of ITK, so changing the data/mapper/etc. will affect downstream filters but not upstream ones
Cool side effects

- An “added bonus” of using VTK is the ability to export scenes to files
- Since data and rendering are abstracted away from each other, it’s pretty easy to, for example, dump your entire rendering setup to a Pixar RenderMan format file (vtkRIBExporter) or a PovRay format file (vtkPOVExporter).
- Have a look at the child classes of vtkExporter to find all available export formats.

The Fast Light Toolkit (FLTK)

- What we have so far is “naked” image processing and rendering code
- We typically want a GUI wrapped around it to facilitate human interaction
- FLTK is an easy-to-use cross platform GUI

Designing a GUI in FLTK

- You can write GUI’s completely by hand, but it’s easier to use FLTK’s GUI-builder, Fluid
- Fluid lets you design an interface graphically and has a native file format, .fl
- It generates two files, a .h and a .cxx containing the GUI code

Program layout in FLTK

- The order in which classes are derived in a FLTK based program may seem non-intuitive
- It arises because the GUI code is generated automatically (by Fluid), and because the core functionality should be abstracted away from the GUI

Program layout in FLTK, cont.

Top: MyAppBase - The base class for your application, the majority of your code goes here.
Middle: MyAppGUI - Generated by Fluid, contains only GUI related code.
Bottom: MyApp - Contains very little code, typically only functions concerning both GUI and core functionality.

How to link FLTK and VTK

- vtkRenderWindowInteractor allows you to place a VTK render window inside a FLTK based GUI
- Mouse events, key presses, etc. are passed to the VTK window
Linking FLTK and ITK

- Although ITK does not render anything directly, you can use FLTK to monitor the progress of pipeline updates
- FltkLightButton changes colors to show modified status
- FltkProgressbar moves to show filter progress between 0 and 100% completion

myITKgui

- myITKgui is “basecode” that you can use or extend
- It is on the class website
- It provides an application framework that allows you to rapidly add new functionality without getting wrapped up in GUI related issues
- Uses a combination of ITK+VTK+FLTK
  - The same combination used by SNAP (from assignment 2)
  - BUT: SNAP has a different structure
- myITKgui is currently not fully compatible with recent versions of ITK+VTK+FLTK (especially on mac).

myITKgui high-level layout

1. Create a RenderWindow
2. Create a Renderer
3. Create a FlRenderWindowInteractor
4. Load an image
5. Create 3 image plane widgets, attach them to the interactor
6. Enter a message loop to run the program

Classes

A. migApp
- migAppBase - contains the pipeline
- migAppGUI - created by Fluid
- migApp - virtual implementation class

B. migWindow
- migWindowBase - renders a 3D image
- migWindowGUI - created by Fluid
- migWindow - doesn’t do much

Execution walkthrough

1. Application starts
2. The code in main() looks like:
   ```c++
   int main(int, char**)
   {
     migApp theApp;
     theApp.Show();
     Fl::run();
     return 0;
   }
   ```
   Create a new app
   Make the app visible
   Start the FLTK event loop

App creation

- The app initializes two render windows, one for the “before pipeline” image and one for the “after pipeline” image

```c++
m_BeforeWindow = new migWindow;
m_BeforeWindow->renderWindow->
  label("myITKgui - 'Before' Image");
m_BeforeWindow->Show();
```
Window creation

- Window creation isn't too interesting
- Member variables are initialized and a renderer / renderWindow pair is created

Some terminology

- Event loop - a program executes in a timed loop and checks each cycle to see if anything interesting has happened
- Callback - a function (generically) that is called when an event occurs

1. If the loop is timed, then it is a polling loop. In multithreaded operating systems, other methods are used that are more efficient. You don't need to worry about this.

After initialization

- Now we have:
  - One migApp
  - Two migWindows
  - Each migWindow has a VTK renderWindow and renderer tucked inside it
  - No image is present
  - FLTK sits in its event loop waiting for something to happen

The user pushes “load image”

- The callback for the load image button is set to migAppBase::ReadImage - this is possible since the GUI derives from the Base
- You can see where the callback is set by editing migAppGUI.fl in Fluid

migAppBase::ReadImage()

1. Get a .mha filename from a dialog box
2. Load this file
3. Set the input image window’s image to what we just loaded and create image plane widget objects
4. Construct the ITK pipeline (what happens here depends on what you’re doing)

Now what?

- We have everything we did before, plus:
  - An image in memory
  - An ITK pipeline
  - The input image is displayed using 3 image planes
  - The output image does not exist yet
  - FLTK goes back into “waiting” mode
User pushes “Update Pipeline”

- The callback for the update button is set to `mAppBase::UpdatePipelineCallback`
- Exactly what happens here is app dependent, but the intent is to call `Update()` on the last object (filter) in the pipeline
- Finally, the output image window has its image set, and image planes are created

That’s it

- That’s the limit of myITKgui’s functionality at this point
- Some easy extensions:
  - Add additional FLTK controls and use their callbacks to specify pipeline parameters in `mAppBase`
  - Add additional VTK objects to the output window (points/lines/etc.)

Error handling

- There really isn’t any - doh!
- Don’t read more than one image
- Don’t push the update button more than once
- We handle these cases in our research code; it’s probably useful for you to solve them on your own (it’s not hard, esp. case 2)

Extending myITKgui

- Replace `CreateITKPipeline()` with code that does something more interesting than binary thresholding
- Edit `UpdatePipeline()` so that your pipeline is updated correctly