Image Registration
Lecture 11: VXL

Prof. Chuck Stewart

VXL - History

- Early 1990's:
  - GE's TargetJr
  - DARPA's Image Understanding Environment (IUE)
  - Both are complicated and cumbersome
- VXL (1999-present):
  - Rewrite / retain best of TargetJr and IUE
  - Stresses simplicity and discipline

VXL - Warning

- High-quality code, but not professional
  - Documentation, while good, is somewhat spartan
  - Not as many examples as ITK
  - No significant notion of releases
- Outside the *core*, development code is mixed with stable code
  - We will not touch this development code, except in the actual registration library

3

VXL Documentation

- Homepage
  - `http://vxl.sourceforge.net`
- VXL “Book”
  - Linked from the homepage
  - `http://paine.wiau.man.ac.uk/pub/doc_vxl/books/core/book.html`

4

VXL Layout - Overview

- src/vcl -
  - Fixes for “non-standard” compiler
- src/core
  - Main vxl libraries, fairly stable
- src/v3p
  - 3rd party code, such as jpeg, mpeg2, netlib…
- src/contrib
  - Libraries written at contributing sites
  - Code has not yet been “promoted” to the core
  - We will use a restricted subset of the contributed libraries

5

VXL – Download

- Checkout the source into a different directory from the Insight (ITK) source!
- You can rename the source direction to anything you wish (as long as it is meaningful to you)

6

Configuring VXL

- Out of tree build is recommended, just as with ITK
  - In my course software directory I have four primary subdirectories:
    - `itk_src, itk_bin, vxl_src, vxl_bin`
- I will use the names `vxl_src` and `vxl_bin` generically to refer to the main source and binary directories

7

Configuring VXL - MS-Windows

- Run CMake, just as you did with ITK
- Select the SOURCE directory, the BINARY directory, and your compiler
- Disable BUILD_EXAMPLES
- Disable BUILD_SHARED_LIBS
- Disable BUILD_TESTING
- Click “Configure”
- Click “Ok”

8

Configuring VXL - Unix

- Create the BINARY directory
- Change directory to the BINARY directory
- Set the environment variables for the compiler: CC and CXX
- Type ccmake with argument the SOURCE directory

9

Configuring VXL - Unix

- Disable BUILD_EXAMPLES
- Enable BUILD_SHARED_LIBS
- Disable BUILD_TESTING

- Type “c” to configure
- Type “g” to generate (ends ccmake)
- Type “make” (in the top-level bin directory)

10

Building VXL - MS-Windows (VC++)

- Open vxl.sln in the Binary Directory
- Select ALL_BUILD project
- Build it

…It will take more than 15 minutes…

11

Building VXL - MS-Windows (VC .NET)

- Open vxl.sln in the Binary Directory
- Select ALL_BUILD project
- Build it

…It will take more than 15 minutes …

12

Verifying the Build

- Libraries will be found in
- In MS-Windows
  - `vxl_bin / lib / { Debug, Release }`
- In UNIX
  - `vxl_bin / lib`

Verifying the Build

- Many libraries
  - From the core: `vcl, vcl, vgl, vgl_algo, vidl, vnl, vnl_algo, vsl, vul`
  - Third-party: `netlib`, ….
    - The number of entries here depends on how many "built-in" libraries were found
  - From contrib: `mbl, rrel, rsdl`

VXL - Hello World Example

```
#include <vcl_iostream.h>

int
main()
{
  vcl_cout << "VXL Hello world!" << vcl_endl;
  return 0;
}
```


Hello World Compilation

- Two options:
  - Follow the instructions for writing your own CMakeLists.txt files from Lecture 6 (recommended).
  - Create code under the main source in `vxl_src/contrib/rpl`
- We'll follow the latter, for now

Hello World - CMake

- Create subdirectory `vxl_src/contrib/rpl/course` and place the source code in a file called `hello_world.cxx` in the new subdirectory
- Edit `vxl_src/contrib/rpl/CMakeLists.txt`
  - Add line

```
SUBDIRS( course )
```

- Create `rpl/course/CMakeLists.txt` with just two lines in it:

```
ADD_EXECUTABLE( hello_world hello_world.cxx )
TARGET_LINK_LIBRARIES( vcl )
```

17

Building Hello World - MS-Windows

- Run CMake again (click “Configure”, click “Ok”)
- Open vxl.dsw (or .sln) generated by CMake
- Select hello_world project
- Build it
- Locate the file hello_world.exe in

```
vxl_bin/contrib/rpl/courses/{ Debug, Release }
```

18

Building Hello World - Unix

- Change into `vxl_bin/contrib/rpl`
- Run `make`
- This creates the `courses` subdirectory and compiles `hello_world.cxx`
- The executable is in the new `courses` subdirectory.

VXL - src/vcl

- Written to allow use of compilers that aren't (weren't at the time) standards-compliant
- Combination of
  - Slightly restricted subset of C++
  - Compiler work-arounds, gathered into one directory.
  - No use of namespaces
- Instead of using `std:` every name from the standard library is pre-fixed as `vcl_`

```
#include <vcl_iostream.h>
int main()
{
  vcl_cout << "VXL Hello world!" << vcl_endl;
  return 0;
}
```

vcl - another example

```
#include <vcl_string.h>    //string
#include <vcl_iostream.h>  // cout
#include <vcl_vector.h>    // vector
#include <vcl_algorithm.h> // copy
#include <vcl_iterator.h>  //ostream_iterator
int main()
{
     vcl_vector<vcl_string> strings;
     strings.push_back("Hello, ");
     strings.push_back("World.");
     vcl_copy(strings.begin(), strings.end(),
            vcl_ostream_iterator<vcl_string>(vcl_cout));
     return 0;
}
```

21

vcl Header Files

- Within directory `vcl` under top-level source directory
  - At same level as core and contrib
- Each time you would like to type the name of a std library header file, such as `foo`, instead type

  `vcl_foo.h`

- This is annoying at first, but after a while you become accustomed to it

22

VXL Naming Conventions

- File names end with
  - .h for header files
  - .cxx for source files
  - .txx for templated source files
- We'll discuss more about templates soon

`vxl_src/core`

- Many libraries. We are primarily interested in four:
  - `vbl` - basic utilities libraries, including arrays, sparse arrays, smart ptr, etc.
  - `vgl` - geometric objects libraries
  - `vil` - image I/o, representation, views and access
  - `vnl` - numerics library
- Algorithm libraries sit as subdirectories:
  - vgl/algo, vil/algo, vnl/algo

vxl_src/core

- Other libraries we may touch
  - `vsl` - binary I/0 stream library
  - `vul` - utilities for timing, I/o, command-line parameter parsing, etc.
  - `vgui` - graphical user-interface
- There are still others that we will not touch

25

vbl

- A library of basic structures:
  - Arrays in 1d, 2d, and 3d
  - Smart pointer
  - Bounding box
  - Sparse arrays
- We'll discuss the first two

26

vbl_array_2d

```
#include <vcl_iostream.h>
#include <vbl/vbl_array_2d.h>

int main()
{
  vbl_array_2d<int> example( 3, 5 );

  example.fill( 1 );

  for ( int i = 0; i < example.rows() && i < example.cols(); ++i )
    example(i, i ) = 10-i;

  vcl_cout << "The example array has " << example.rows()
           << " rows and " << example.cols() << " columns\n"
           << "Here are the contents: \n" << example << vcl_endl;
  return 0;
}
```

Note paths of header files

Templated object

27

Changes to CMakeLists.txt

- Placed in same directory as my hello world program
- My CMakeLists.txt file is now

```
ADD_EXECUTABLE( hello_world hello_world.cxx )
ADD_EXECUTABLE( vbl_array_example main_vbl_array.cxx )

TARGET_LINK_LIBRARIES( hello_world vcl )
TARGET_LINK_LIBRARIES( vbl_array_example vcl vbl )
```

- For MS-Windows you need to re-run Cmake, which will detect the changes.
- For UNIX you do not need to re-run ccmake explicitly. Running `make` in the bin directory will re-run cmake automatically, as necessary.

28

Naming Conventions in VXL

- File names tend to be the same as the single class the file defines
  - `vbl_array_2d.h, vbl_array_2d.txx`
- All objects and files in directory are prefaced with the name of the library
  - Avoids name clashes
  - Makes finding objects simple
  - It's a bit annoying as well

29

Templates and Template Instantiation

- Template classes and functions are specified in both .h and .txx files
  - Vbl_array_2d.txx is pretty small, but this isn't always the case
- Template instantiation is done explicitly rather than implicitly (as in ITK)
- Each .txx has a C++ instantiation macro at the end. Here is the one from vbl_array_2d.txx

```
#undef VBL_ARRAY_2D_INSTANTIATE
#define VBL_ARRAY_2D_INSTANTIATE(type) \
template class vbl_array_2d<type >;\
template vcl_ostream& operator<< (vcl_ostream&,\
   vbl_array_2d<type > const& )
```

30

Template, continued

- Source directories will often have subdirectories called Templates, where templated classes and functions are instantiated explicitly.
  - CMake finds and compiles these instantiation files automatically, without having to specify the file names in CMakeLists.txt

31

Templates, continued

- For example, there are six files for instantiating vbl_array_2d objects in vbl/Templates:

```
vbl_array_2d+bool-.cxx
vbl_array_2d+double-.cxx
vbl_array_2d+int-.cxx
vbl_array_2d+short-.cxx
vbl_array_2d+unsigned-.cxx
vbl_array_2d+unsignedchar-.cxx
```

- Here is the interior of vbl_array_2d+int-.cxx

```
#include <vbl/vbl_array_2d.txx>
VBL_ARRAY_2D_INSTANTIATE(int);
```

- Note: if you were instantiating a templated object to contain a class that you had defined, you would have to #include that class's .h file as well

32

Where to Put Template Instantiations?

- Generic, widely used instantiations are placed where the templated class is declared
  - E.g. `vbl/Templates`
- More specialized instantiations involving user-defined classes are placed with the user-defined class.
  - For example, if you created a class myl_foo in directory myl, then an instantiation file would be placed in myl/Templates:
    - vbl_array_2d+myl_foo-.cxx

33

Templates, Final Comments

- Remember: you do not need to change anything in vxl_src/vcl, vxl_src/core or vxl_src/contrib to use templated classes declared there.
- We don't anticipate that you will be writing your own templated classes or functions
- You may need to write your own template instantiations
- Work from examples and look at Appendix C of the vxl book.

34

Smart Pointers in VXL

- Automatic deletion of reference counted objects so that you don't need to keep track of pointers
- Never invoke `delete` on a smart pointer!
- What's needed to use smart pointers in vxl:
  - Class pointed to from a smart pointer must inherit from `vbl_ref_count`
  - Smart pointer class `vbl_smart_ptr` is templated

Simple Smart Pointer Example

```
#include <vcl_iostream.h>
#include <vbl/vbl_ref_count.h>
#include <vbl/vbl_smart_ptr.h>


class big_foo : public vbl_ref_count {
public:
  big_foo( double dflt=0.0 )
  { for ( int i=0; i<1024; ++i ) data[i] = dflt; }

  double get( int i ) const { return data[i]; }
  void set( int i, double x ) { data[i] = x; }

private:
  double data[1024];
};

//  Create a typedef for the big_foo smart pointer
typedef vbl_smart_ptr<big_foo> big_foo_sptr;
```

Simple Smart Pointer Example

```
int
main()
{
  big_foo_sptr p = new big_foo( 1.0 );
  {
    big_foo_sptr q = p;  // alias
    q->set( 0, 23.0 );
  }  // q is out of scope so it is gone

  //  p remains, but its data has been changed.
  vcl_cout << "p's first value = "
           << p -> get( 0 ) << vcl_endl;
  return 0;
}

//  Instantiating here.  This isn't the common practice
#include <vbl/vbl_smart_ptr.txx>
VBL_SMART_PTR_INSTANTIATE(big_foo);
```

37

vil

- Image representation and access library
- 2d images only and no associated physical coordinates
  - Reflects difference between medical images and video images
  - A 3d image library exists in `vxl_src/contrib/mul`
- Image processing and feature detection algorithms exist in subdirectory `vil/algo`

38

vil - Basic Operations

- Load images using vil_load
- Save images using vil_save
- Access pixels using a vil_image_view<T>

vil - Threshold Example

```
#include <vxl_config.h>
#include <vil/vil_load.h>
#include <vil/vil_save.h>
#include <vil/vil_image_view.h>

int main(int argc, char **argv)
{
  vil_image_view<vxl_byte> img;
  img = vil_load(argv[1]);
  for (unsigned j = 0; j < img.nj(); ++j)
     for (unsigned i = 0; i < img.ni(); ++i)
        if (img(i,j) < 200 )
          img(i,j) = vxl_byte(0);
  vil_save(img, argv[2]);
  return 0;
}
```

vil/algo

- Image analysis algorithms are in a subdirectory of vxl_src/core/vil called algo
  - Examples include morphological openings and closings, smoothing operators, Sobel edge detectors, etc.
- Placing algorithms in an algo subdirectory is also used in the numerics and geometry libraries

41

vil/algo - Example

```
#include <vcl_iostream.h>
#include <vxl_config.h>  // generated file
#include <vil/vil_load.h>
#include <vil/vil_save.h>
#include <vil/vil_image_view.h>
#include <vil/algo/vil_line_filter.h>
#include <vnl/vnl_math.h>   // vnl_math_rnd

int main(int argc, char **argv)
{
  vil_image_view<vxl_byte> img;
  img = vil_load(argv[1]);

  vil_image_view<vxl_byte> line_dir;   // results storage
  vil_image_view<float> line_str;

  // Apply line finder
  vil_line_filter<vxl_byte> line_filter;
  line_filter . dark_lines_5x5( line_dir, line_str, img );
```

42

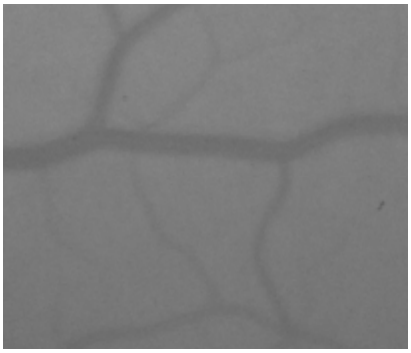
vil/algo - Example

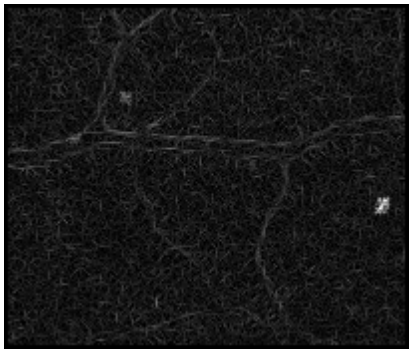
```
  // Find the maximum strength line output
  float max_str = 0.0;
  for ( unsigned int i=0; i<line_str.ni(); ++i )
    for ( unsigned int j=0; j<line_str.nj(); ++j )
      if ( line_str(i,j) > max_str ) max_str = line_str(i,j);

  //  Create scaled image of results
  vil_image_view<vxl_byte> scaled_results( line_str.ni(), line_str.nj()
);
  for ( unsigned int i=0; i<line_str.ni(); ++i )
    for ( unsigned int j=0; j<line_str.nj(); ++j )
      scaled_results(i,j) = vnl_math_rnd( line_str(i,j) / max_str * 255
);
  // Can be handled easily with functions defined in vil_math

  vil_save(scaled_results, argv[2]);
  return 0;
}
```


vil/algo - Example Results





vil vs. vil1

- You will see references for the library vil1
  - The imaging library was completely rewritten and improved significantly last year
  - Most libraries have been rewritten, but a few, such as vidl have not.
  - This is not a major concern, but don't let yourself be confused by it

`vnl`

- Numerics library
  - Shared with ITK, although it is usually buried inside higher-level classes
- `vnl/` contains
  - Vectors and their (compile-time) fixed specializations
  - Matrices and sparse matrices
  - `vnl_math.h` - handy things like `vnl_math_rnd` which you wish `math.h` had
  - Quaternions
- `vnl/algo`
  - SVD and other matrix decompositions
  - Various optimization techniques

vnl and vnl/algo - SVD Example

```
#include <vcl_iostream.h>
#include <vnl/vnl_math.h>
#include <vnl/vnl_vector.h>
#include <vnl/vnl_matrix.h>
#include <vnl/algo/vnl_svd.h>

int main()
{
  vnl_vector<double> p(4);   // 4-component vector

  p(0) = 1.0;
  p(1) = 0.3;
  p(2) = -0.5;
  p(3) = 2.0;

  //  4x4 matrix, filled with 0.0
  vnl_matrix<double> m(4, 4, 0.0);
  m(0,0) = 1.0;   m(0,1) = 5.0;   m(0,2) = -2.0;  m(0,3) = 0.5;
  m(1,0) = 0.0;   m(1,1) = 1.0;   m(1,2) = 3.0;   m(1,3) = -3.0;
  m(2,0) = -2.0;  m(2,1) = 15.0;  m(2,2) = 4.0;   m(2,3) = 0.5;
  m(3,0) = 1.0;   m(3,1) = -0.2;  m(3,2) = 1.0;   m(3,3) = 9.5;
```

47

vnl and vnl/algo - SVD Example

```
  vcl_cout << "p = " << p << "\n"
           << "m = \n" << m << "\n" << vcl_endl;

  //  The computation of the svd is computed in the constructor.
  vnl_svd<double> svd(m);

  vcl_cout << "SVD of m:\n" << "U = " << svd.U() << "\n"
          << "V = " << svd.V() << vcl_endl;

  vcl_cout << "Singular values:  ";
  for ( unsigned int i=0; i<4; ++i )
    vcl_cout << svd.W(i) << " ";
  vcl_cout << vcl_endl;

  return 0;
}
```

48

vgl

- Geometric library
- Represent points, lines, planes, polygons, ellipses, etc…
- Very few numerical techniques in `vgl`
  - Prevents dependence between `vgl` and `vnl`
- Numerical techniques are in `vgl/algo`

Command Lines and GUIs

- Many programs can be written and run from the command-line
  - Use other programs - xv, matlab, etc. to view images and results
- vgui
  - Graphical interface
  - See xcv and the vxl book

vxl_src/contrib

- Contributed libraries
- Some excellent code, but “use at your own risk”
- We will concentrate on the Rensselaer library
  - `rgrl` - the generalized registration library
  - `rrel` - the robust estimation library
  - `rsdl` - the spatial data structures library
- Lecture 12 will focus on `rgrl`

Summary

- Read the book!
- Look at the .h files and the documentation