Programming in C++ using RAVL

Bill Christmas

This presentation:

- includes a bit of the background to Ravl
- gives you an idea of some of the functionality available in RAVL;
- provides some examples of programming using RAVL;

..... *not* a comprehensive tour of RAVL

This presentation:

- includes a bit of the background to Ravl
- gives you an idea of some of the functionality available in RAVL;
- provides some examples of programming using RAVL;

..... *not* a comprehensive tour of RAVL

This presentation:

- includes a bit of the background to Ravl
- gives you an idea of some of the functionality available in RAVL;
- provides some examples of programming using RAVL;

..... *not* a comprehensive tour of RAVL

This presentation:

- includes a bit of the background to Ravl
- gives you an idea of some of the functionality available in RAVL;
- provides some examples of programming using RAVL;

..... *not* a comprehensive tour of RAVL

- AMMA C++ library — c. 1995?? — Radek Mařik
- New version — RAVL — c. 2000 — Charles Galambos

- AMMA C++ library — c. 1995?? — Radek Mařik

- New version — RAVL — c. 2000 — Charles Galambos

RAVL features

- C++ class library:
  - Basic containers
  - System interface
  - Maths, geometry inc. projective geometry - 2D & 3D
  - Image, video, 2D & 3D, audio and supporting tools
  - Pattern recognition
  - GUIs (GTK-based)
  - Applications (a few)
- Motivation:
  - lack of existing s/w (c. 1995)
  - C++ "difficult"
  - explicit memory management tedious
- Consistent programming interface
- Not much C++ knowledge required to use it
- RAVL and STL
- RAVL and OpenCV

RAVL features

- C++ class library:
  - Basic containers
  - System interface
  - Maths, geometry inc. projective geometry - 2D & 3D
  - Image, video, 2D & 3D, audio and supporting tools
  - Pattern recognition
  - GUIs (GTK-based)
  - Applications (a few)
- Motivation:
  - lack of existing s/w (c. 1995)
  - C++ "difficult"
  - explicit memory management tedious
- Consistent programming interface
- Not much C++ knowledge required to use it
- RAVL and STL
- RAVL and OpenCV

RAVL features

- C++ class library:
  - Basic containers
  - System interface
  - Maths, geometry inc. projective geometry - 2D & 3D
  - Image, video, 2D & 3D, audio and supporting tools
  - Pattern recognition
  - GUIs (GTK-based)
  - Applications (a few)
- Motivation:
  - lack of existing s/w (c. 1995)
  - C++ "difficult"
  - explicit memory management tedious
- Consistent programming interface
- Not much C++ knowledge required to use it
- RAVL and STL
- RAVL and OpenCV

RAVL features

- C++ class library:
  - Basic containers
  - System interface
  - Maths, geometry inc. projective geometry - 2D & 3D
  - Image, video, 2D & 3D, audio and supporting tools
  - Pattern recognition
  - GUIs (GTK-based)
  - Applications (a few)
- Motivation:
  - lack of existing s/w (c. 1995)
  - C++ "difficult"
  - explicit memory management tedious
- Consistent programming interface
- Not much C++ knowledge required to use it
- RAVL and STL
- RAVL and OpenCV

RAVL features

- C++ class library:
  - Basic containers
  - System interface
  - Maths, geometry inc. projective geometry - 2D & 3D
  - Image, video, 2D & 3D, audio and supporting tools
  - Pattern recognition
  - GUIs (GTK-based)
  - Applications (a few)
- Motivation:
  - lack of existing s/w (c. 1995)
  - C++ "difficult"
  - explicit memory management tedious
- Consistent programming interface
- Not much C++ knowledge required to use it
- RAVL and STL
- RAVL and OpenCV

RAVL features

- C++ class library:
  - Basic containers
  - System interface
  - Maths, geometry inc. projective geometry - 2D & 3D
  - Image, video, 2D & 3D, audio and supporting tools
  - Pattern recognition
  - GUIs (GTK-based)
  - Applications (a few)
- Motivation:
  - lack of existing s/w (c. 1995)
  - C++ "difficult"
  - explicit memory management tedious
- Consistent programming interface
- Not much C++ knowledge required to use it
- RAVL and STL
- RAVL and OpenCV

RAVL features

- C++ class library:
  - Basic containers
  - System interface
  - Maths, geometry inc. projective geometry - 2D & 3D
  - Image, video, 2D & 3D, audio and supporting tools
  - Pattern recognition
  - GUIs (GTK-based)
  - Applications (a few)
- Motivation:
  - lack of existing s/w (c. 1995)
  - C++ "difficult"
  - explicit memory management tedious
- Consistent programming interface
- Not much C++ knowledge required to use it
- RAVL and STL
- RAVL and OpenCV

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/RavI

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/RavI

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/RavI

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/RavI

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/RavI

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/RavI

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/Ravl

Support features

- Multi-platform: Linux, MS, ...
- Used for real products (Omniperception Ltd.)
- Source code available from SourceForge, under LGPL
- Simplified compilation system
- Recompiled and tested nightly
- Automatic documentation generation
- Full documentation on web:
  http://www.ee.surrey.ac.uk/CVSSP/Ravl

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :

RAVL C++ basics

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :

RAVL C++ basics

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

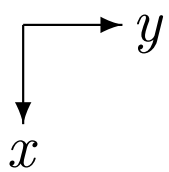
- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :

- RAVL naming conventions
- RAVL namespaces — easily overlooked
- RAVL primitive typedefs — for more consistent word lengths
- RAVL constants — under `RavlConstN` namespace:

```
RealT x = RavlConstN::pi;
```

```
using namespace RavlConstN;
RealT x = pi;
```

- `IndexC` - replacement for `int` that provides more consistent rounding behaviour
- RAVL coordinates :



Member function syntax

In general, C++ classes have 2 ways of accessing member functions:

- "struct" syntax:

```
a.b();
```

- "pointer" syntax:

```
a->b();
```

RAVL always uses the "struct" syntax, except where iterators are used.

Member function syntax

In general, C++ classes have 2 ways of accessing member functions:

- "struct" syntax:

```
a.b();
```

- "pointer" syntax:

```
a->b();
```

RAVL always uses the "struct" syntax, except where iterators are used.

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

$\longrightarrow$ "JAVA-like" class interfaces
$\longrightarrow$ (almost) no pointers
$\longrightarrow$ automatic destruction
$\longrightarrow$ no memory leaks (I believe)
$\longrightarrow$ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by inheritance of `RCHandleC` or `BufferAccessC` (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

⟶ "JAVA-like" class interfaces
⟶ (almost) no pointers
⟶ automatic destruction
⟶ no memory leaks (I believe)
⟶ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by inheritance of `RCHandleC` or `BufferAccessC` (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

⟶ "JAVA-like" class interfaces
⟶ (almost) no pointers
⟶ automatic destruction
⟶ no memory leaks (I believe)
⟶ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

⟶ "JAVA-like" class interfaces
⟶ (almost) no pointers
⟶ automatic destruction
⟶ no memory leaks (I believe)
⟶ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

$\longrightarrow$ "JAVA-like" class interfaces
$\longrightarrow$ (almost) no pointers
$\longrightarrow$ automatic destruction
$\longrightarrow$ no memory leaks (I believe)
$\longrightarrow$ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

$\longrightarrow$ "JAVA-like" class interfaces

$\longrightarrow$ (almost) no pointers

$\longrightarrow$ automatic destruction

$\longrightarrow$ no memory leaks (I believe)

$\longrightarrow$ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

$\longrightarrow$ "JAVA-like" class interfaces
$\longrightarrow$ (almost) no pointers
$\longrightarrow$ automatic destruction
$\longrightarrow$ no memory leaks (I believe)
$\longrightarrow$ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

$\longrightarrow$ "JAVA-like" class interfaces
$\longrightarrow$ (almost) no pointers
$\longrightarrow$ automatic destruction
$\longrightarrow$ no memory leaks (I believe)
$\longrightarrow$ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

$\longrightarrow$ "JAVA-like" class interfaces
$\longrightarrow$ (almost) no pointers
$\longrightarrow$ automatic destruction
$\longrightarrow$ no memory leaks (I believe)
$\longrightarrow$ thread-safe for SMP

Reference counting: big / small objects

All "big" objects:

- are generally objects of indeterminate size (e.g. containers)
- have a *handle* (a reference-counted pointer)
- identified by
  inheritance of `RCHandleC` or `BufferAccessC`
  (+ StringC)

"Small" objects behave like built-ins (int, double etc.)

$\longrightarrow$ "JAVA-like" class interfaces
$\longrightarrow$ (almost) no pointers
$\longrightarrow$ automatic destruction
$\longrightarrow$ no memory leaks (I believe)
$\longrightarrow$ thread-safe for SMP

Example: big object

Only the handle is copied in this example, *not* the contents (like a "C" array):

```
Array1dC<IntT>a(6);
a.Fill(27);
Array1dC<IntT>b = a;
b[4] = 55;
cout << a[4] << endl;
```

```
> myprog
55
```

Thus the value of a[4] is also changed.

Copying big objects


```
Array1dC<DListC<IntT> > a(6);
// then initialise object 'a' somehow

// just copies array handle:
Array1dC<DListC<IntT> > b = a;
```

```
// copies array of list handles:
Array1dC<DListC<IntT> > b = a.Copy();
```

```
// copies everything (maybe - best avoided 'til ready):
Array1dC<DListC<IntT> > b = a.DeepCopy()
```

Copying big objects

```
Array1dC<DListC<IntT> > a(6);
// then initialise object 'a' somehow

// just copies array handle:
Array1dC<DListC<IntT> > b = a;
```

```
// copies array of list handles:
Array1dC<DListC<IntT> > b = a.Copy();
```

```
// copies everything (maybe - best avoided 'til ready):
Array1dC<DListC<IntT> > b = a.DeepCopy()
```

Copying big objects

```
Array1dC<DListC<IntT> > a(6);
// then initialise object 'a' somehow

// just copies array handle:
Array1dC<DListC<IntT> > b = a;
```

```
// copies array of list handles:
Array1dC<DListC<IntT> > b = a.Copy();
```

```
// copies everything (maybe - best avoided 'til ready):
Array1dC<DListC<IntT> > b = a.DeepCopy()
```

Text and files

- `StringC` - for basic text processing. E.g.:

```
StringC a("Hello");  StringC b("World");
StringC c = a + ' ' + b;
cout << c.from('o') << endl;
```

generates "o World".

- `FilenameC`, `DirectoryC` — derived from `StringC` to provide information on and manipulate file properties and directories:

```
FilenameC f("myfile.txt");
if (f.Exists()) {
  f.SetPermissions(0444);
  f.Rename("yourfile.txt");
}
```

- Using "-" as a file name denotes stdin or stdout as appropriate.

Text and files

- `StringC` - for basic text processing. E.g.:

```
StringC a("Hello");  StringC b("World");
StringC c = a + ' ' + b;
cout << c.from('o') << endl;
```

generates "o World".

- `FilenameC`, `DirectoryC` — derived from `StringC` to provide information on and manipulate file properties and directories:

```
FilenameC f("myfile.txt");
if (f.Exists()) {
  f.SetPermissions(0444);
  f.Rename("yourfile.txt");
}
```

- Using "-" as a file name denotes stdin or stdout as appropriate.

Text and files

- `StringC` - for basic text processing. E.g.:

```
StringC a("Hello");  StringC b("World");
StringC c = a + ' ' + b;
cout << c.from('o') << endl;
```

  generates "o World".

- `FilenameC`, `DirectoryC` — derived from `StringC` to provide information on and manipulate file properties and directories:

```
FilenameC f("myfile.txt");
if (f.Exists()) {
  f.SetPermissions(0444);
  f.Rename("yourfile.txt");
}
```

- Using "-" as a file name denotes stdin or stdout as appropriate.

Text and files

- `StringC` - for basic text processing. E.g.:

```
StringC a("Hello");  StringC b("World");
StringC c = a + ' ' + b;
cout << c.from('o') << endl;
```

  generates "o World".

- `FilenameC`, `DirectoryC` — derived from `StringC` to provide information on and manipulate file properties and directories:

```
FilenameC f("myfile.txt");
if (f.Exists()) {
  f.SetPermissions(0444);
  f.Rename("yourfile.txt");
}
```

- Using "-" as a file name denotes stdin or stdout as appropriate.

Command-line options

`OptionC` handles processing of command-line options and parameters. This code fragment:

```
OptionC opt(argc, argv);
bool select     (opt.Boolean("bw",       "b&w (default: colour)"));
RealT gamma     (opt.Real   ("gamma", 1.0,       "display gamma"));
Index2dC size   (opt.Index2d("size",  576, 720,  "image size"));
StringC logFile (opt.String ("log",   "-",       "log file"));
StringC opName  (opt.String ("",      "out.ppm", "o/p image"));
opt.Compulsory("size");
opt.Check();
```

might handle this command line:

```
myprog file.pgm -bw -size 768 1024
```

- The default value of a boolean option is always `false`.
- Options without tags ("opName") *must* be handled last.
- File name "-" is used in RAVL for standard input & output.

Command-line options

`OptionC` handles processing of command-line options and parameters. This code fragment:

```
OptionC opt(argc, argv);
bool select     (opt.Boolean("bw",       "b&w (default: colour)"));
RealT gamma     (opt.Real   ("gamma", 1.0,       "display gamma"));
Index2dC size   (opt.Index2d("size",  576, 720,  "image size"));
StringC logFile (opt.String ("log",   "-",       "log file"));
StringC opName  (opt.String ("",      "out.ppm", "o/p image"));
opt.Compulsory("size");
opt.Check();
```

might handle this command line:

```
myprog file.pgm -bw -size 768 1024
```

- The default value of a boolean option is always `false`.
- Options without tags ("opName") *must* be handled last.
- File name "-" is used in RAVL for standard input & output.

Command-line options

`OptionC` handles processing of command-line options and parameters. This code fragment:

```
OptionC opt(argc, argv);
bool select     (opt.Boolean("bw",       "b&w (default: colour)"));
RealT gamma     (opt.Real   ("gamma", 1.0,       "display gamma"));
Index2dC size   (opt.Index2d("size",  576, 720,  "image size"));
StringC logFile (opt.String ("log",   "-",       "log file"));
StringC opName  (opt.String ("",      "out.ppm", "o/p image"));
opt.Compulsory("size");
opt.Check();
```

might handle this command line:

```
myprog file.pgm -bw -size 768 1024
```

- The default value of a boolean option is always `false`.
- Options without tags ("opName") *must* be handled last.
- File name `"-"` is used in RAVL for standard input & output.

Command-line options (cont.)

The command:

```
myprog -help
```

generates this o/p:

```
Usage: myprog [options]
   -bw (false) [false] b&w (default: colour)
   -gamma (1) [1] display gamma
   -size (576 720) [576 720] image size
   -log (-) [-] log file
   arg (out.ppm) [out.ppm] o/p image name
   -help (true) [false] Print usage information.

Dependencies:
   -size is compulsory.
```

Command-line errors produce a similar output.

Command-line options (cont.)

The command:

```
myprog -help
```

generates this o/p:

```
Usage: myprog [options]
   -bw (false) [false] b&w (default: colour)
   -gamma (1) [1] display gamma
   -size (576 720) [576 720] image size
   -log (-) [-] log file
   arg (out.ppm) [out.ppm] o/p image name
   -help (true) [false] Print usage information.

Dependencies:
   -size is compulsory.
```

Command-line errors produce a similar output.

- `IStreamC / OStreamC`: provide the usual text stream I/O.
- `BinIStreamC / BinOStreamC`: corresponding binary I/O - reads back in exactly what is written out (c.f. `fread / fwrite`).
- `StrIStreamC / StrOStreamC`: like `IStreamC / OStreamC`, but reads from / writes to a string (like `sscanf / sprintf`). Can also be used (with care) to initialise small arrays:

```
SArray1dC<RealT> coeffs(5);
StrIStreamC ("5 0.363 0.291 0.135 0.012 -0.030") >> coeffs;
```

- `XMLIStreamC / XMLOStreamC`: additional XML support for linear XML file handling.

- `IStreamC / OStreamC`: provide the usual text stream I/O.
- `BinIStreamC / BinOStreamC`: corresponding binary I/O - reads back in exactly what is written out (c.f. `fread / fwrite`).
- `StrIStreamC / StrOStreamC`: like `IStreamC / OStreamC`, but reads from / writes to a string (like `sscanf / sprintf`). Can also be used (with care) to initialise small arrays:

```
SArray1dC<RealT> coeffs(5);
StrIStreamC ("5 0.363 0.291 0.135 0.012 -0.030") >> coeffs;
```

- `XMLIStreamC / XMLOStreamC`: additional XML support for linear XML file handling.

- `IStreamC / OStreamC`: provide the usual text stream I/O.
- `BinIStreamC / BinOStreamC`: corresponding binary I/O - reads back in exactly what is written out (c.f. `fread / fwrite`).
- `StrIStreamC / StrOStreamC`: like `IStreamC / OStreamC`, but reads from / writes to a string (like `sscanf / sprintf`). Can also be used (with care) to initialise small arrays:

```
SArray1dC<RealT> coeffs(5);
StrIStreamC ("5 0.363 0.291 0.135 0.012 -0.030") >> coeffs;
```

- `XMLIStreamC / XMLOStreamC`: additional XML support for linear XML file handling.

- `IStreamC / OStreamC`: provide the usual text stream I/O.
- `BinIStreamC / BinOStreamC`: corresponding binary I/O - reads back in exactly what is written out (c.f. `fread / fwrite`).
- `StrIStreamC / StrOStreamC`: like `IStreamC / OStreamC`, but reads from / writes to a string (like `sscanf / sprintf`). Can also be used (with care) to initialise small arrays:

```
SArray1dC<RealT> coeffs(5);
StrIStreamC ("5 0.363 0.291 0.135 0.012 -0.030") >> coeffs;
```

- `XMLIStreamC / XMLOStreamC`: additional XML support for linear XML file handling.

Series of template classes that generate complex objects:

- arrays: `Array1dC`, `Array2dC`, `Array3dC`
- "simple" versions: `SArray1dC`, `SArray2dC`, `SArray3dC`
- dynamic 1D arrays: `DArray1dC`
- doubly linked lists: `DListC`
- hash tables: `HashC`, `RCHashC`
- trees
- queues
- .....

Iterators for containers

"Intelligent pointers": Each container class has one or more associated iterators to provide efficient indexing through the container object. So, to take square root of array values:

```
#include "Ravl/Array1d.hh"
#include "Ravl/Array1dIter.hh"

using namespace RavlN;

int main() {
  Array1dC<RealT>a(1,4);
  for (Array1dIterC<RealT> i(a); i; ++i) {
    *i = Sqrt(i.Index());
  }
  cout << a << endl;
}
```

Compiler options for array bound checking.

Iterators for containers

"Intelligent pointers": Each container class has one or more associated iterators to provide efficient indexing through the container object. So, to take square root of array values:

```
#include "Ravl/Array1d.hh"
#include "Ravl/Array1dIter.hh"

using namespace RavlN;

int main() {
  Array1dC<RealT>a(1,4);
  for (Array1dIterC<RealT> i(a); i; ++i) {
    *i = Sqrt(i.Index());
  }
  cout << a << endl;
}
```

Compiler options for array bound checking.

Iterators for containers

"Intelligent pointers": Each container class has one or more associated iterators to provide efficient indexing through the container object. So, to take square root of array values:

```
#include "Ravl/Array1d.hh"
#include "Ravl/Array1dIter.hh"

using namespace RavlN;

int main() {
  Array1dC<RealT>a(1,4);
  for (Array1dIterC<RealT> i(a); i; ++i) {
    *i = Sqrt(i.Index());
  }
  cout << a << endl;
}
```

☹ Compiler options for array bound checking.

Compound iterators

- You can iterate over more than one object using a single compound iterator.
  - E.g. to copy one array to another that is offset from the first:

```
#include "Ravl/Array1d.hh"
#include "Ravl/Array1dIter2.hh"
#include "Ravl/Index.hh"

using namespace RavlN;

int main() {
  Array1dC<RealT>a(4);
  Array1dC<IndexC>b(3,6);
  a.Fill(8.6);
  for (Array1dIter2C<RealT,IndexC> i(a,b); i; ++i) {
    i.Data2() = i.Data1();
  }
}
```

Compound iterators

- You can iterate over more than one object using a single compound iterator.
- E.g. to copy one array to another that is offset from the first:

```
#include "Ravl/Array1d.hh"
#include "Ravl/Array1dIter2.hh"
#include "Ravl/Index.hh"

using namespace RavlN;

int main() {
  Array1dC<RealT>a(4);
  Array1dC<IndexC>b(3,6);
  a.Fill(8.6);
  for (Array1dIter2C<RealT,IndexC> i(a,b); i; ++i) {
    i.Data2() = i.Data1();
  }
}
```

`ImageC`: 2-D array with extensions

- Can have any pixel type - byte, integer, real, boolean ....
- Variety of pre-defined pixel types
- Image border is object of class `ImageRectangleC`
- Comprehensive file and device I/O for images and video
- Image processing / computer vision algorithms

Filtering images

Example using user-specified mask:

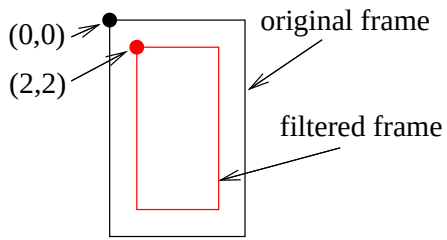
```
#include "Ravl/Image/ConvolveSeparable2d.hh"
#include "Ravl/IO.hh"
#include "Ravl/StrStream.hh"
using namespace RavlN;
using namespace RavlImageN;

int main (int argc, char* argv[]) {
  ImageC<RealT> im;  Load ("image.pgm", im);
  Array1dC<RealT> coeffs;
  StrIStreamC ("-2 2 1 3 5 3 1") >> coeffs;
  ConvolveSeparable2dC<RealT> filter(coeffs);
  im = filter.Apply(im);
  cout << im.Frame() << endl;
}
```

Note symmetric image shrinkage:

> myprog

2 297 2 197



Indexing images

Really bad:

```
for (IntT x(0); x<576; ++x)  for (IntT y(0); y<720; ++y)
  im[x][y] += 3;
```

Slightly better:

```
for (IntT x(0); x<im.Rows(); ++x)  for (IntT y(0); y<im.Cols(); ++y)
  im[x][y] += 3;
```

Better:

```
for (IndexC x(im.TRow()); x<=im.BRow(); ++x)
  for (IndexC y(im.LCol()); y<=im.RCol(); ++y)
    im[x][y] += 3;
```

Best (and fastest):

```
for (Array2dIterC<IntT>i(im); i; ++i)
  *i += 3;
```

Indexing images

Really bad:

```
for (IntT x(0); x<576; ++x)  for (IntT y(0); y<720; ++y)
  im[x][y] += 3;
```

Slightly better:

```
for (IntT x(0); x<im.Rows(); ++x)  for (IntT y(0); y<im.Cols(); ++y)
  im[x][y] += 3;
```

Better:

```
for (IndexC x(im.TRow()); x<=im.BRow(); ++x)
  for (IndexC y(im.LCol()); y<=im.RCol(); ++y)
    im[x][y] += 3;
```

Best (and fastest):

```
for (Array2dIterC<IntT>i(im); i; ++i)
  *i += 3;
```

Indexing images

Really bad:

```
for (IntT x(0); x<576; ++x)  for (IntT y(0); y<720; ++y)
  im[x][y] += 3;
```

Slightly better:

```
for (IntT x(0); x<im.Rows(); ++x)  for (IntT y(0); y<im.Cols(); ++y)
  im[x][y] += 3;
```

Better:

```
for (IndexC x(im.TRow()); x<=im.BRow(); ++x)
  for (IndexC y(im.LCol()); y<=im.RCol(); ++y)
    im[x][y] += 3;
```

Best (and fastest):

```
for (Array2dIterC<IntT>i(im); i; ++i)
  *i += 3;
```

Indexing images

Really bad:

```
for (IntT x(0); x<576; ++x)  for (IntT y(0); y<720; ++y)
  im[x][y] += 3;
```

Slightly better:

```
for (IntT x(0); x<im.Rows(); ++x)  for (IntT y(0); y<im.Cols(); ++y)
  im[x][y] += 3;
```

Better:

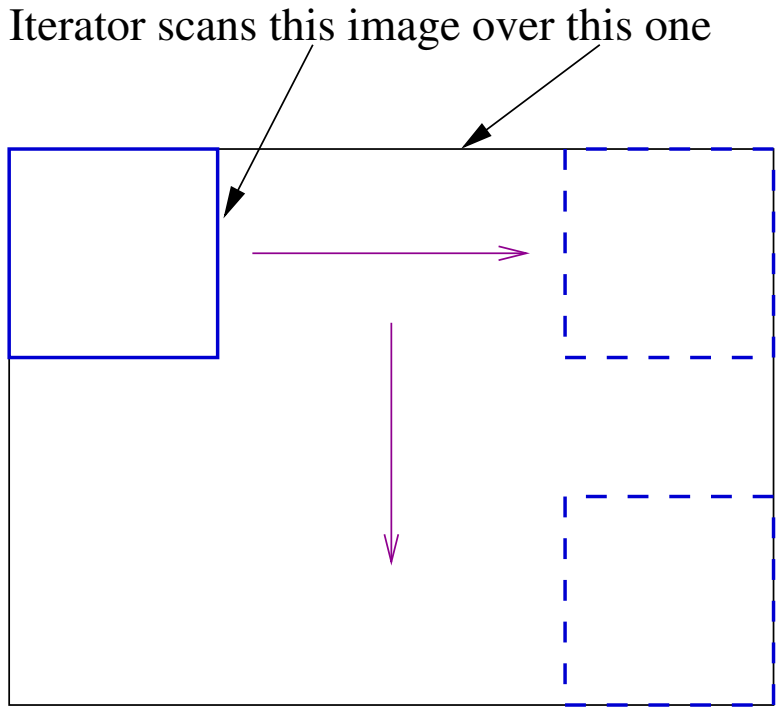
```
for (IndexC x(im.TRow()); x<=im.BRow(); ++x)
  for (IndexC y(im.LCol()); y<=im.RCol(); ++y)
    im[x][y] += 3;
```

Best (and fastest):

```
for (Array2dIterC<IntT>i(im); i; ++i)
  *i += 3;
```

Sliding one image around another

The `Rectangle2dIterC` iterator allows one image to scan over another larger one:



— e.g. for image convolution

Subimages

Construct new image from old:

```
ImageC<ByteT> a(576,720);
a.Fill(12);
ImageC<ByteT> b(a, IndexRange2dC(100,400,200,700));
b.Fill(135);
```

Data is *not* copied when constructing `b`. So `a` finally is:

Image I/O using `Load()` / `Save()`

- can be files or devices ("virtual files")
- conversion between pixel types is automatic
  E.g. conversion of RGB 8-bit file to grey-scale float object:

```
#include "Ravl/IO.hh"
#include "Ravl/Image/Image.hh"

using namespace RavlN;
using namespace RavlImageN;

int main (int argc, char* argv[]) {
  ImageC<RealT> im;
  if (! Load ("image.png", im))    cerr << "panic!!" << endl;
  Save ("image.pgm", im);
  Save ("@X:image", im); // write to screen with title "image"
}
```

Image I/O using `Load()` / `Save()`

- can be files or devices ("virtual files")
- conversion between pixel types is automatic
  E.g. conversion of RGB 8-bit file to grey-scale float object:

```
#include "Ravl/IO.hh"
#include "Ravl/Image/Image.hh"

using namespace RavlN;
using namespace RavlImageN;

int main (int argc, char* argv[]) {
  ImageC<RealT> im;
  if (! Load ("image.png", im))    cerr << "panic!!" << endl;
  Save ("image.pgm", im);
  Save ( "@X:image" , im); // write to screen with title "image"
}
```

Image I/O (cont.)

Corresponding `defs.mk` file:

```
MAINS= myprog.cc

PROGLIBS = RavlImageIO RavlExtImgIO RavlDPDisplay
```

Image sequence I/O — to read and write a sequence:

```
#include "Ravl/DP/SequenceIO.hh"
#include "Ravl/Image/Image.hh"
#include "Ravl/Image/ByteRGBValue.hh"
using namespace RavlN;
using namespace RavlImageN;

int main (int argc, char* argv[]) {
  if (argc < 3)  exit(-1);
  DPIPortC<ImageC<ByteRGBValueC> > in;
  if(!OpenISequence(in, argv[1]))  exit(-2);
  DPOPortC<ImageC<ByteRGBValueC> > out;
  if(!OpenOSequence(out, argv[2])) exit (-3);
  ImageC<ByteRGBValueC> im;
  while(in.Get(im))
    out.Put(im);
}
```

Video I/O (cont.)

Corresponding `defs.mk` file:

```
MAINS= seqio.cc

PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Run it like this to convert to sequence of 'ppm's:

```
seqio movie.mpeg tmp.ppm
```

Run it like this to display the file:

```
seqio movie.mpeg @X
```

[1]

Run it like this to display a camera o/p (include RavlImgIOV4L library):

```
seqio @V4L @X
```

[1] To run this example, you need to (a) be using xpdf to run the presentation, (b) have compiled up the "seqio" example, and (c) have "movie.mpeg" in your current directory.

Video I/O (cont.)

Corresponding `defs.mk` file:

```
MAINS= seqio.cc

PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Run it like this to convert to sequence of 'ppm's:

```
seqio movie.mpeg tmp.ppm
```

Run it like this to display the file:

```
seqio movie.mpeg @X
```

[1]

Run it like this to display a camera o/p (include RavlImgIOV4L library):

```
seqio @V4L @X
```

[1] To run this example, you need to (a) be using xpdf to run the presentation, (b) have compiled up the "seqio" example, and (c) have "movie.mpeg" in your current directory.

Video I/O (cont.)

Corresponding `defs.mk` file:

```
MAINS= seqio.cc

PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Run it like this to convert to sequence of 'ppm's:

```
seqio movie.mpeg tmp.ppm
```

Run it like this to display the file:

```
seqio movie.mpeg @X
```
[1]

Run it like this to display a camera o/p (include RavlImgIOV4L library):

```
seqio @V4L @X
```

[1] To run this example, you need to (a) be using xpdf to run the presentation, (b) have compiled up the "seqio" example, and (c) have "movie.mpeg" in your current directory.

Video I/O (cont.)

Corresponding `defs.mk` file:

```
MAINS= seqio.cc

PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Run it like this to convert to sequence of 'ppm's:

```
seqio movie.mpeg tmp.ppm
```

Run it like this to display the file:

```
seqio movie.mpeg @X
```
[1]

Run it like this to display a camera o/p (include `RavlImgIOV4L` library):

```
seqio @V4L @X
```

[1] To run this example, you need to (a) be using xpdf to run the presentation, (b) have compiled up the "seqio" example, and (c) have "movie.mpeg" in your current directory.

More general I/O

Automatic numbering of file sequences:

- `OpenOSequence(out,"a.ppm")` creates a0.ppm, a1.ppm, ..., a10.ppm, ...
- `OpenOSequence(out,"a%4d.ppm")` creates a0000.ppm, a0001.ppm, ..., a0010.ppm, ...
- `OpenISequence(in,"a.ppm")` will read either style.
- `Load()`, `Save()`, `OpenISequence`, `OpenOSequence` can also be used for reading writing arbitrary objects (including images) as text files:

```
ImageC<IntT>
0 1 0 2
32 35 45
33 39 42
```

More general I/O

Automatic numbering of file sequences:

- `OpenOSequence(out,"a.ppm")` creates a0.ppm, a1.ppm, ..., a10.ppm, ...
- `OpenOSequence(out,"a%4d.ppm")` creates a0000.ppm, a0001.ppm, ..., a0010.ppm, ...
- `OpenISequence(in,"a.ppm")` will read either style.
- `Load()`, `Save()`, `OpenISequence`, `OpenOSequence` can also be used for reading writing arbitrary objects (including images) as text files:

```
ImageC<IntT>
0 1 0 2
32 35 45
33 39 42
```

More general I/O

Automatic numbering of file sequences:

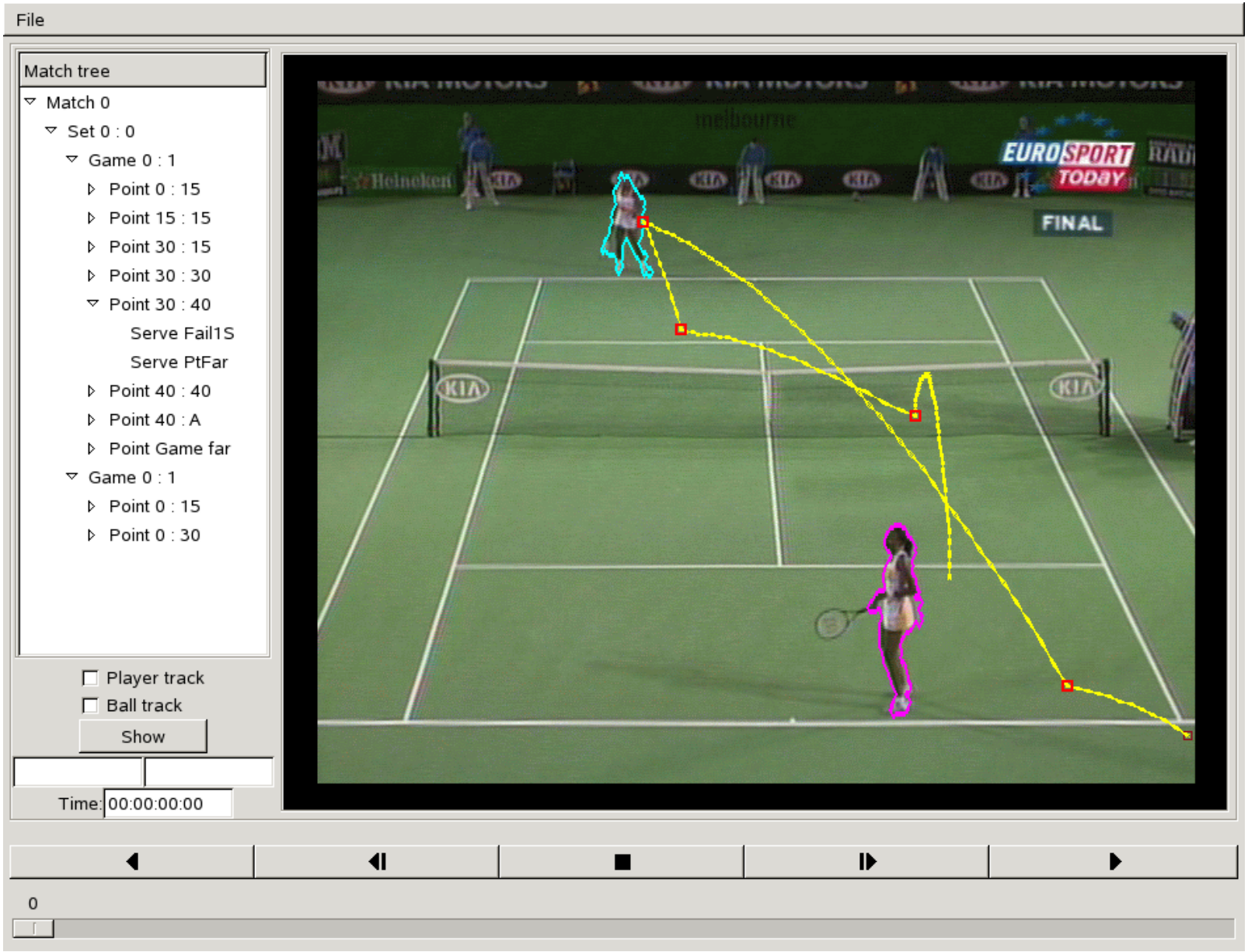
- `OpenOSequence(out,"a.ppm")` creates a0.ppm, a1.ppm, ..., a10.ppm, ...
- `OpenOSequence(out,"a%4d.ppm")` creates a0000.ppm, a0001.ppm, ..., a0010.ppm, ...
- `OpenISequence(in,"a.ppm")` will read either style.
- `Load()`, `Save()`, `OpenISequence`, `OpenOSequence` can also be used for reading writing arbitrary objects (including images) as text files:

```
ImageC<IntT>
0 1 0 2
32 35 45
33 39 42
```

Automatic numbering of file sequences:

- `OpenOSequence(out,"a.ppm")` creates a0.ppm, a1.ppm, ..., a10.ppm, ...
- `OpenOSequence(out,"a%4d.ppm")` creates a0000.ppm, a0001.ppm, ..., a0010.ppm, ...
- `OpenISequence(in,"a.ppm")` will read either style.
- `Load()`, `Save()`, `OpenISequence`, `OpenOSequence` can also be used for reading writing arbitrary objects (including images) as text files:

```
ImageC<IntT>
0 1 0 2
32 35 45
33 39 42
```



Maths

- linear algebra
- Euclidean geometry
- projective geometry
  - mosaicking:

- .....

Pattern recognition

Some standard classifiers:

- SVM
- GMM
- ANN
- KNN
- .....

Optimisers:

- conjugate gradient
- .....

O/S interfaces: threads

1. Define the function to be launched as a thread, e.g.:

```
bool PrintNumber(int &i) {
  cout << "PrintNumber called with value " << i << endl;
  return true;
}
```

2. Create a "trigger":

```
TriggerC myTrigger = Trigger(&PrintNumber, 53);
```

3. Then launch it as a separate thread:

```
LaunchThreadC thread = LaunchThread (myTrigger) ;
```

4. and wait for it to finish:

```
thread.Wait()
```

QMake: a compilation tool

Compiling main programs

To compile (1 or more) main program(s) in a directory:

- Create a file called `defs.mk` in that directory, consisting of one line to tell QMake what the files are called:

```
MAINS = main1.cc main2.cc main3.cc
```

- Execute the command `qm`

If your program(s) need RAVL libraries that QMake cannot find automatically at link time (mainly I/O libs), you will need to add them to the `defs.mk` file. E.g.:

```
MAINS = main1.cc main2.cc main3.cc
PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Compiling main programs

To compile (1 or more) main program(s) in a directory:

- Create a file called `defs.mk` in that directory, consisting of one line to tell QMake what the files are called:

```
MAINS = main1.cc main2.cc main3.cc
```

- Execute the command `qm`

If your program(s) need RAVL libraries that QMake cannot find automatically at link time (mainly I/O libs), you will need to add them to the `defs.mk` file. E.g.:

```
MAINS = main1.cc main2.cc main3.cc
PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Compiling main programs

To compile (1 or more) main program(s) in a directory:

- Create a file called `defs.mk` in that directory, consisting of one line to tell QMake what the files are called:

```
MAINS = main1.cc main2.cc main3.cc
```

- Execute the command `qm`

If your program(s) need RAVL libraries that QMake cannot find automatically at link time (mainly I/O libs), you will need to add them to the `defs.mk` file. E.g.:

```
MAINS = main1.cc main2.cc main3.cc
PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Compiling main programs

To compile (1 or more) main program(s) in a directory:

- Create a file called `defs.mk` in that directory, consisting of one line to tell QMake what the files are called:

```
MAINS = main1.cc main2.cc main3.cc
```

- Execute the command `qm`

If your program(s) need RAVL libraries that QMake cannot find automatically at link time (mainly I/O libs), you will need to add them to the `defs.mk` file. E.g.:

```
MAINS = main1.cc main2.cc main3.cc
PROGLIBS = RavlVideoIO RavlLibFFmpeg RavlDPDisplay
```

Adding extra libraries

You can including your own class and header files:

```
MAINS = main1.cc main2.cc main3.cc
HEADERS = header1.hh header2.hh
SOURCES = class1.cc class2.cc
```

And much more...

Adding extra libraries

You can including your own class and header files:

```
MAINS = main1.cc main2.cc main3.cc
HEADERS = header1.hh header2.hh
SOURCES = class1.cc class2.cc
```

And much more...

Compiling with QMake

- `qm` is just an alias for `make`:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - `qm shared`, `qm debugshared` → run-time array bound checking

- `$PROJECT_OUT`, `/tmp/<user>/qm` can always be deleted:

  `qm distclean`

Compiling with QMake

- `qm` is just an alias for `make`:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - `qm shared`, `qm debugshared` → run-time array bound checking

- `$PROJECT_OUT`, `/tmp/<user>/qm` can always be deleted:

  `qm distclean`

Compiling with QMake

- `qm` is just an alias for `make`:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - `qm shared`, `qm debugshared` → run-time array bound checking

- `$PROJECT_OUT`, `/tmp/<user>/qm` can always be deleted:

  `qm distclean`

Compiling with QMake

- qm is just an alias for make:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - qm shared, qm debugshared → run-time array bound checking

- $PROJECT_OUT, /tmp/<user>/qm can always be deleted:

  `qm distclean`

Compiling with QMake

- qm is just an alias for make:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - qm shared, qm debugshared → run-time array bound checking

- $PROJECT_OUT, /tmp/<user>/qm can always be deleted:

  `qm distclean`

Compiling with QMake

- qm is just an alias for make:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - qm shared, qm debugshared → run-time array bound checking

- $PROJECT_OUT, /tmp/<user>/qm can always be deleted:

  `qm distclean`

Compiling with QMake

- qm is just an alias for make:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - `qm shared`, `qm debugshared` → run-time array bound checking

- $PROJECT_OUT, /tmp/<user>/qm can always be deleted:

  `qm distclean`

Compiling with QMake

- qm is just an alias for make:

```
> alias qm
gmake -f /vol/vssp/localsoft/Auto/linux/Ravl/share/RAVL/QMake/QMake.mk USERBUILD=1 !* && rehash
```

- For faster linking:

  `qm shared`

- For faster runtime:

  `qm optshared`

- Debugging using ddd / gdb:

  `qm debugshared`

  - a bit of help in debugging RAVL
  - `qm shared`, `qm debugshared` → run-time array bound checking

- $PROJECT_OUT, /tmp/<user>/qm can always be deleted:

  `qm distclean`

Compiling with QMake (cont.)

Static or shared (dynamic) libraries?

- static libs (use `qm`, `qm debug`, `qm opt`)
  – good for long-running jobs (libs can't change)
- shared libs (use `qm shared`, `qm debugshared`, `qm optshared`)
  – good for development (fast linking)

"Complete" list of `qm` commands:

```
qm help
```

Compiling with QMake (cont.)

Static or shared (dynamic) libraries?

- static libs (use `qm`, `qm debug`, `qm opt`)
  – good for long-running jobs (libs can't change)
- shared libs (use `qm shared`, `qm debugshared`, `qm optshared`)
  – good for development (fast linking)

"Complete" list of `qm` commands:

```
qm help
```

Compiling with QMake (cont.)

Static or shared (dynamic) libraries?

- static libs (use `qm`, `qm debug`, `qm opt`)
  – good for long-running jobs (libs can't change)
- shared libs (use `qm shared`, `qm debugshared`, `qm optshared`)
  – good for development (fast linking)

"Complete" list of `qm` commands:

```
qm help
```

Documentation

- Class documentation automatically generated from header files
- Background documentation in branches, not in class leaves
- – while examples *are* usually in class leaves
- Search engine
- 2 lots of documentation: internal, public
  In each there is:
  - the 'blue users' version, and
  - the 'green developers' one

Documentation

- Class documentation automatically generated from header files
- Background documentation in branches, not in class leaves
- – while examples *are* usually in class leaves
- Search engine
- 2 lots of documentation: internal, public
  In each there is:
  - the 'blue users' version, and
  - the 'green developers' one

Documentation

- Class documentation automatically generated from header files
- Background documentation in branches, not in class leaves
- - while examples *are* usually in class leaves
- Search engine
- 2 lots of documentation: internal, public
  In each there is:
  - the 'blue users' version, and
  - the 'green developers' one

Documentation

- Class documentation automatically generated from header files
- Background documentation in branches, not in class leaves
- - while examples *are* usually in class leaves
- Search engine
- 2 lots of documentation: internal, public
  In each there is:
  - the 'blue users' version, and
  - the 'green developers' one

Documentation

- Class documentation automatically generated from header files
- Background documentation in branches, not in class leaves
- - while examples *are* usually in class leaves
- Search engine
- 2 lots of documentation: internal, public
  In each there is:
  - the blue users' version, and
  - the green developers' one

Documentation

- Class documentation automatically generated from header files
- Background documentation in branches, not in class leaves
- - while examples *are* usually in class leaves
- Search engine
- 2 lots of documentation: internal, public
  In each there is:
  - the blue users' version, and
  - the green developers' one

Documentation

- Class documentation automatically generated from header files
- Background documentation in branches, not in class leaves
- - while examples *are* usually in class leaves
- Search engine
- 2 lots of documentation: internal, public
  In each there is:
  - the blue users' version, and
  - the green developers' one

Finally....

- Contributions always welcome (including C):
  - for RAVL itself
  - for use within CVSSP
- Go to
  http://www.ee.surrey.ac.uk/CVSSP/Ravl/RavlIntro.html
  and read all about it.
- Tell us what is missing

Finally....

- Contributions always welcome (including C):
  - for RAVL itself
  - for use within CVSSP

- Go to
  http://www.ee.surrey.ac.uk/CVSSP/Ravl/RavlIntro.html
  and read all about it.

- Tell us what is missing

Finally....

- Contributions always welcome (including C):
  - for RAVL itself
  - for use within CVSSP

- Go to
  http://www.ee.surrey.ac.uk/CVSSP/Ravl/RavlIntro.html
  and read all about it.

- Tell us what is missing

We are here to help