Horus User Guide

Version 2.0 - Jan 2003

Dennis Koelma

And other ISIS members

Intelligent Sensory Information Systems
University of Amsterdam, Faculty of Science
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
koelma@science.uva.nl
http://www.science.uva.nl/~horus/

Contents

1 Introduction **2**

1.1 Introduction 2

1.2 Architectural overview 2

1.3 Conceptual overview 4

1.4 This document 5

2 Installation **6**

2.1 Installation 6

2.2 On Windows 98/2000/NT/XP 6

2.3 On Unix (SunOS and Linux) 7

3 Getting Started **10**

3.1 Getting Started 10

3.2 Starting applications 10

3.2.1 A C++ console application 10

3.2.2 Corba applications 11

3.3 A first application 11

3.4 Calling image operations 14

3.4.1 Reading images from file 14

3.4.2 An example of a binary pixel operation: Adding images 14

3.4.3 An example of a unary pixel operation: Thresholding images 18

3.4.4 An example of a generalized convolution: Uniform filter 20

3.4.5 An example of a generalized convolution: Convolution 22

3.4.6 An example of a generalized convolution: Erosion 24

3.5 Instantiating patterns 25

3.5.1 Instantiation of a binary pixel operation: Squared distance 26

3.5.2 Instantiation of a unary pixel operation: Tri state threshold 28

3.5.3 Instantiation of an in/out operation : Image to histogram 31

Chapter 1

Introduction

1.1 Introduction

This document gives an overview of the architecture and functionality of the current Horus implementation. The overview focusses on the implementation as is. It provides only a flavour of the design rationale. The aim is to provide a starting point for usage and extension of the functionality.

For more detailed information please refer to the reference documentation. And, of course, the answer is in the source code :-)

1.2 Architectural overview

The core of Horus is a C++ library for image processing. The library contains classes for images, image sequences, histograms, BSplines, etc. The functionality of the core Horus library can be employed in several ways.

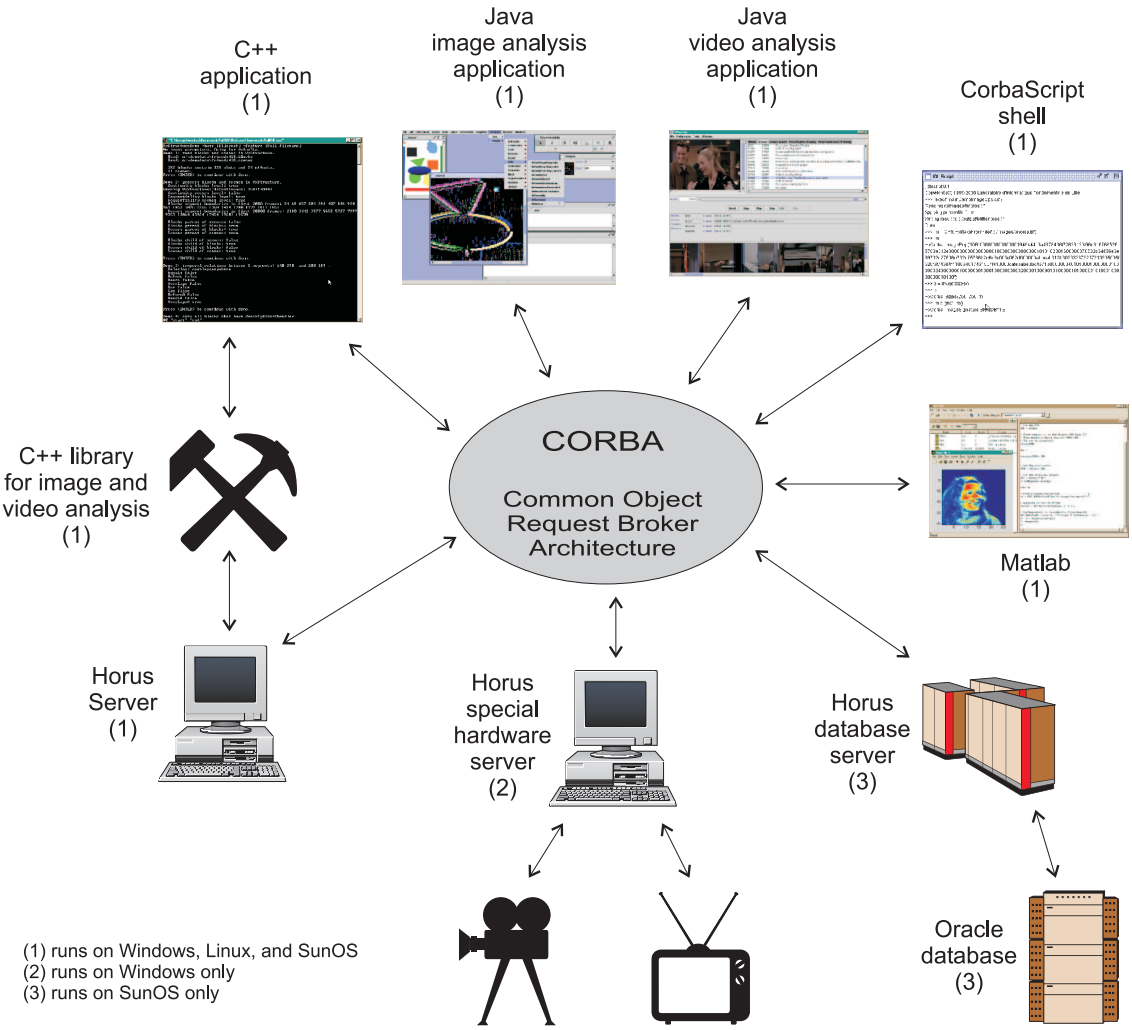

Figure 1.1: Horus architecture

The most direct way is calling the C++ functions from a console application. Within the C++ console application a user has complete access to the functionality but lacks flexibility in that changing the application requires recompilation. Also, there are no means for visualization of results.

Several graphical user interfaces are available that provide access to the C++ functionality in a more interactive way. The GUI's are implemented in Java to be portable and, in the end, suited for demonstrations via WWW. To facilitate development of these GUI's and new ones a set of common Java components is collected in a separate library. In the end, the components are to become Java beans to be used in a standard application development environment.

The Java Mdi application is specifically targeted towards researchers in the image processing community. It has a menu and dialogue interface to provide interactive access to most commonly used image processing operations and viewers to inspect images up to the pixel level.

The Video application is more targeted towards video processing. It has a simple interface that would also suit an end-user.

The Java GUI's communicate with the C++ library via CORBA (the Common Object Request Broker Architecture). To that end, the C++ image processing functionality is defined in CORBA's Interface Definition

Language (IDL) and an HxServer has been build to translate the IDL requests to the corresponding C++ functionality. Client applications, e.g. the Java GUI's, have access to the server in all language bindings supported by CORBA.

The Horus functionality may also be used from within third-party software. CorbaScript provides direct access to all CORBA-based functionality (without additional effort on the Horus side). Since Matlab 6 (release 12) has a Java interpreter, the C++ functionality of Horus is accessible directly via CORBA.

Examples of how to use the various Horus applications are given in the next chapter.

1.3 Conceptual overview

This is really conceptual, not all is implemented yet :-)

From an (end-)users' point of view it is essential to have a clear definition of concepts used in the Horus system in order to master the functionality provided by the system. The concepts provide the user with a set of handles for interaction with the Horus system. An overview of the concepts is given in the figure.

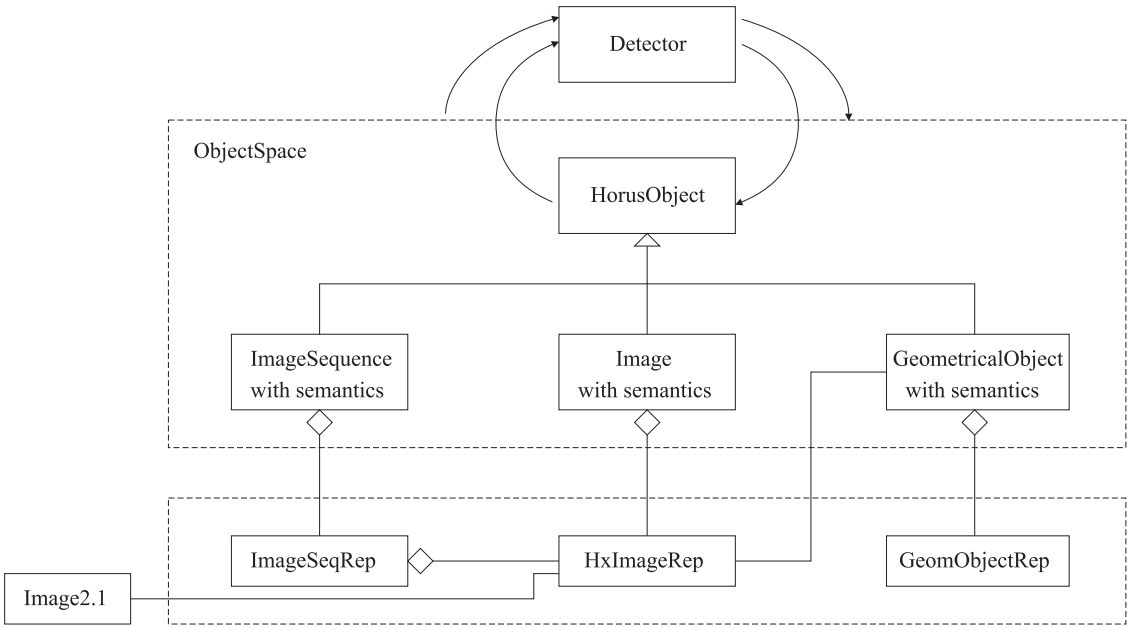

Figure 1.2: Conceptual overview

Functionality within the Horus system is divided into two categories: HorusObjects and Detectors. Each data-entity used within Horus is called a HorusObject. Horus objects include the sensory data itself and the spatial entities present within the sensory data. Transformations on (a set of) HorusObjects are preformed by detectors.

A key issue in the design of Horus is the separation between semantics, representation, and implementation of HorusObjects. Semantics, represenation, and implementation are defined and implement by different classes. Typically, the class defining the semantics contains (a reference to) the class defining the representation which contains (a reference to) the class defining the implementation. Each class has its own set of admissable operations.

As an example, consider a HorusObject defining the age (in years) of a person as a number. For the represenation we take a natural number, i.e. x is an element of **N**. The choice for **N** also defines the set of

admissable operations on x. The class defining the semantics of x, i.e. it is an age, typically modifies the set of allowable operations. Some operations may no longer be allowed, e.g. decrement, while others may be introduced, e.g. isAnAdult. For the implementation we might take an 8-bit integer if space is critical or a 16-bit integer to be on the safe side. Naturally, the situation is a bit more complicated when dealing with images instead of ages.

1.4 This document

The next chapters provides a handle to get install Horus and to get started with the various components of Horus. The remaining chapters describe each component in more detail.

Chapter 2

Installation

2.1 Installation

1. **On Windows 98/2000/NT/XP** (p. 6)
2. **On Unix (SunOS and Linux)** (p. 7)

2.2 On Windows 98/2000/NT/XP

Other software

Horus depends upon/coorporates with several other software packages. Whether you need to install these depends upon the way you use Horus.

Java 1.4 (and Java3D 1.3) Either from `http://www.science.uva.nl/~horus/third/java/` or the Java site (`http://java.sun.com`). In this document we assume Java to be installed in d:\jdk.

Visual C++ 6.0 You have to buy this one :-(If you don't have it, you can still run the applications by putting msvcp60.dll and msvcrt.dll from HXOBJ\lib\vc60 on your path.

Matlab 6 (release 12) In this document we assume Matlab to be installed in c:\matlabr12.

The release

The latest Horus release can be obtained from `http://www.science.uva.nl/~horus/release/` . There are 4 zip-files in a distribution:

horus_"number"_lib.zip Contains libraries and other essentials to run Horus. In this document we assume this zip-file to be extracted to d:\hx.

horus_"number"_src.zip Contains the source code. In this document we assume this zip-file to be extracted to n:\horus\src.

horus_"number"_third_lib.zip Contains third party libraries needed by Horus. In this document we assume this zip-file to be extracted to d:\hx.

horus_"number"_third_src.zip Contains third party include files needed by Horus. In this document we assume this zip-file to be extracted to n:\horus\src.

The ”number” of the third party software may not be in sync with the horus number. In that case, use the latest version of the third party software.

Configuration

You have to make the source directory known as a logical drive x with:

```
subst x: n:\horus\src
```

You can, for example, do this by putting the statement in a batch file which is executed at startup, i.e. put the batch file in the startup menu.

You have to set (or add to) the following environment variables:

```
set HXJDK=d:/jdk
set PLATFORM=gnu
set HXOBJ=d:\hx (pay attention to the ONLY backslash!)
set HXROOT=d:/hx
set HXSRC=x:
set HXUSER=x:/HxUser
set ORBACUS_CONFIG=x:/HxCorba/cfg/orbacus.cfg
set CLASSPATH=.;%HXROOT%/lib/classes;%HXROOT%/lib/third
set PATH=%HXJDK%/bin;%HXSRC%/HxBin;%HXROOT%/lib;%HXROOT%/lib/third
```

For CorbaScript the following are handy:

```
set CS_CONFIG_HOMEDIR=x:/HxCorba/cfg
set CSPATH=x:/HxSamples/Scripts
```

In case you want to recompile .idl or .java files you will need to configure GNU make (only once, e.g. at the command prompt):

```
md c:\cygwin
mount -u c:\cygwin /
sh -c "gnu_mkdir /bin"
mount d:\hx\lib\third /bin
```

In case you use Matlab you have to add

```
d:/hx/lib/classes
d:/hx/lib/third
```

to *c:\matlabr12\toolbox\local\classpath.txt* . Also handy :

```
copy x:\HxSamples\Matlab\startup.m c:\matlabr12\toolbox\local
```

Done!

If all went well you are now ready to run Horus. For some examples, read chapter **Getting Started** (p. 10).

Good luck.

2.3 On Unix (SunOS and Linux)

Other software

Horus depends upon/coorporates with several other software packages. The packages are installed via `softpkg`, so make sure you use `softpkg` (or find your own solution :-)

To find all packages, insert the following before the softpkg command is executed (note that this example is for .cshrc):

```
setenv PACKAGEPATH ${HOME}/pkg:/home/horus/pkg:/usr/local/pkg
```

Put the following packages in your .pkgrc. Make sure they are before the DEFAULT package otherwise you will end up with the wrong compiler.

```
gcc-3.2
jdk-1.4.0
orbacus405_410
jidlscript01
jpeg6b
horus
oracle817
resetclasspath
```

Linux users may want to install the software packages on their local disc. The source of all packages is in /home/horus/third.

The release

The latest Horus release can be obtained from `http://www.science.uva.nl/~horus/release/` or /home/horus/release. You need only to copy the src directory. In this document we assume you copied it to ${HOME}/horus/src.

Configuration

You have to set one environment variable:

```
setenv HXBUILD Release
```

In case you use Matlab you have to have a directory ${HOME}/matlab.

```
mkdir ${HOME}/matlab
```

Copy the *classpath.txt* file to ${HOME}/matlab. The classpath.txt file is located by typing "which classpath.txt" in Matlab. (typically something like /usr/local/matlab/toolbox/local/classpath.txt) Add the following to ${HOME}/matlab/classpath.txt:

```
/home/USER/horus/lib/classes
/home/horus/local/JOB-4.0.5/lib/OB.jar
/home/horus/local/JOB-4.0.5/lib/OBUtil.jar
```

Note that you cannot use environment variables in classpath.txt so make sure you adjust the USER value above to reflect your situation! And, you have to start matlab in the ${HOME}/matlab directory otherwise your classpath.txt is not used!

Also handy :

```
cp ${HXSRC}/HxSamples/Matlab/startup.m ${HOME}/matlab
```

Build

Go to the ${HOME}/horus/src directory, type `make`, and go get a cup of coffee, or two, or three, ...

Done!

If all went well you are now ready to run Horus. For some examples, read chapter **Getting Started** (p. 10).

Good luck.

Chapter 3

Getting Started

3.1 Getting Started

Horus contains several applications to use the core C++ library. The applications include a command line C++ application, a Java GUI interface, the CorbaScript interpreter, and Matlab. In this chapter, we povide a starting point to use the applications to do some image processing.

This chapter:

- **Starting applications** (p. 10)
- **A first application** (p. 11)
- **Calling image operations** (p. 14)
  - **Reading images from file** (p. 14)
  - **An example of a binary pixel operation: Adding images** (p. 14)
  - **An example of a unary pixel operation: Thresholding images** (p. 18)
  - **An example of a generalized convolution: Uniform filter** (p. 20)
  - **An example of a generalized convolution: Convolution** (p. 22)
  - **An example of a generalized convolution: Erosion** (p. 24)
- **Instantiating patterns** (p. 25)
  - **Instantiation of a binary pixel operation: Squared distance** (p. 26)
  - **Instantiation of a unary pixel operation: Tri state threshold** (p. 28)
  - **Instantiation of an in/out operation : Image to histogram** (p. 31)

3.2 Starting applications

3.2.1 A C++ console application

Well, starting a console application is easy : just type the name of the executable at the command prompt :-)

Windows The Horus distribution contains a sample workspace to compile and execute the first demo application in Microsoft Visual Studio 6. The workspace is located in *horus/worksp/hxconsoleUserDll*. You can execute the application via 'execute' from the 'Build' menu or via 'Ctrl-F5'. **End Windows**

Unix The Horus distribution contains a makefile to compile the first demo application. Go to *$HXSRC/Hx-Samples/ConsDemo* and type `make`. The application is executed via `mainDemoImageOps`. **End Unix**

3.2.2 Corba applications

The Java GUI's, CorbaScript, and Matlab are CORBA clients in that they execute C++ functions via CORBA. In general, CORBA clients require several CORBA applications to run before they are started : the Interface Repository (IR), the Naming Service (NS), and the server (HxServer) itself.

Windows The Horus distribution contains shortcuts to start the CORBA facilities with the proper parameters. The shortcuts are located in *horus/toolbar* . **End Windows**

Unix The Horus distribution contains a script called `ss` to start the CORBA facilities with the proper parameters. Type `ss ir` to start the Interface Repository, `ss ns` and `ss hxserver` for the other services. **End Unix**

Note : the Naming Service is based upon a database file that has to be created upon first use. You can do this by starting NSS instead of NS the first time you start the Naming Service.

Java GUI's

With the Interface Repository, the Naming Service, and the HxServer running we can start the Java applications by typing `java MidApp` or `java` **VideoApp** (p. **??**) at the command prompt.

CorbaScript

The Horus toolbar contains a shortcut to start the CorbaScript shell. You can also type `cssh` at the command prompt.

Matlab

For starting Matlab, consult the Matlab documentation (and make sure you followed the installation instructions in **Installation** (p. 6)).

3.3 A first application

In this section we shows how various applications are used to execute a first application.

C++ console application

An example of a C++ console application is given in *HxSamples/ConsDemo/mainDemoImageOps.c* :

```
/*
 *  Copyright (c) 2000, University of Amsterdam, The Netherlands.
 *  All rights reserved.
 *
 *  Author(s):
 *  Dennis Koelma (koelma@wins.uva.nl)
 *  Edo Poll (poll@wins.uva.nl)
 *  Marc Navarro (mnavarro@wins.uva.nl)
 *
 */

#include <cstdlib>
#include "HxImageRepGlobalFuncs.h"

#ifdef WIN32
const HxString inputDir = "//carol/horus/images/";
const HxString outputDir = "c:/output/";
#else
const HxString inputDir = "/home/horus/images/";
const HxString outputDir = HxString(getenv("HOME")) + "/output/";
#endif
```

```
int
main(int argc, char* argv[])
{
    HxString inputFileName = inputDir + "bnoise.tif";
    STD_COUT << "Reading [" << inputFileName << "]" << STD_ENDL;
    HxImageRep im = HxMakeFromFile(inputFileName);

    STD_COUT << "Applying percentile filter" << STD_ENDL;
    HxImageRep result = HxPercentile(im, 3, 0.5);

    HxString outputFileName = outputDir + "blittlenoise.tif";
    STD_COUT << "Writing result to [" << outputFileName << "]" << STD_ENDL;
    HxWriteFile(result, outputFileName);

    STD_COUT << "Done" << STD_ENDL;
    return 0;
}
```

See **A C++ console application** (p. 10) for more information on compilation and execution of this example.

Note : All images mentioned can be found in `http://www.science.uva.nl/~horus/images/` (or /home/horus/images). Output is written to 'c:\output' or '$HOME/output'. Make sure this directory exists before executing the demo.

MidApp

MidApp executes C++ functions via the Java binding of the IDL definitions. The IDL definitions of the Horus classes are in the **HxCorba** interface. The most important IDL interfaces are **HxCorba::Constructor** and **HxCorba::GlobalOps**. The Constructor interface is generally the starting point as it is used to make objects. The GlobalOps interface contains all global (C++) functions and is used to do most of the image processing stuff.

The MidApp has two ways to execute the application : via the menu & dialogue interface and via the builtin IDL script interpreter (jidlscipt).

In the menu & dialogue interface the Constructor and GlobalOps interfaces are accessible via the 'HxCorba' menu. GlobalOps also has its own menu.

To execute the three functions of the mainDemoImageOps application do this: (Note that reading the left-hand side of the dialogue box in top-down order gives an impression of the function call that will be executed)

- Select 'HxMakeFromFile' from the 'GlobalOps/Images/Generation' menu. Enter `im` for the 'ImageRep' (this will give the name 'im' to the resulting object) en `c:/images/bnoise.tif` for the 'fileName' parameter. Press 'OK'. (if all is well you will now see the image)
- Select 'HxPercentile' from the 'GlobalOps/Images/Filter' menu. Enter `result` for the name of the result object. Select 'im' for the 'ImageRep' parameter. Enter `3` for the 'neighSize' parameter and `0.5` for the 'perc' parameter. Press 'OK'
- Select 'HxWriteFile' from the 'GlobalOps/Images/Export' menu. Select 'result' for the 'ImageRep' parameter. Enter `c:/output/blittlenoise.tif` for the 'fileName' parameter. Press 'OK'.

Jidlscript is basically the same as the CorbaScript (cssh) but written in Java (so much slower). To open a jidlscipt window select 'IDL script' from the 'Tools' menu. Jidlscript is integrated with the Java application in that it also knows about the Constructor and GlobalOps interfaces. It has to objects (CTOR and OPS) to access the interfaces. You can check this by typing `global` in the IDL script window. (If you didn't quit the application since you executed the functions via the menu & dialogue interface you will also see the ImageRep objects).

The jidlscript code for our first sample application is given in *HxSamples/Scripts/mainDemoImageOps.cs* :

```
#  Copyright (c) 2001, University of Amsterdam, The Netherlands.
#  All rights reserved.
#
#  Author(s):
#  Dennis Koelma (koelma@wins.uva.nl)
#

println("Reading c:/images/bnoise.tif");
im = OPS.HxMakeFromFile("c:/images/bnoise.tif");

println("Applying percentile filter");
result = OPS.HxPercentile(im, 3, 0.5);

println("Writing result to c:/output/blittlenoise.tif");
OPS.HxWriteFile(result, "c:/output/blittlenoise.tif");

println("Done");
```

You can execute the script by typing

`exec("x:/HxSamples/Scripts/mainDemoImageOps.cs")`

in the IDL script window.

CorbaScript

Since cssh is a standalone application, the first thing we have to do is contact the HxServer. You can get the Constructor and GlobalOps objects by executing

`exec("hxInit.cs")`

Note : you have to specify the full path with the exec function unless you are in the 'x:/HxSamples/Scripts' directory since the environment variable CSPATH only seems to work for the 'import' command.

You can verify the existance of the objects by typing `global` . Execution of the sample application is done via

`exec("mainDemoImageOps.cs")`

Matlab

The Matlab code is in '$HXSRC/HxSamples/Matlab'. To avoid specifying the full path see **Installation** (p. 6).

In Matlab, the Constructor and GlobalOps object are obtained by executing `hxInit` . The sample application is implemented in *$HXSRC/HxSamples/Matlab/hxMainDemoImageOps.m* :

```
%hxMainDemoImageOps     Demo: shows the basics of how to use Horus images


%  Copyright (c) 2001, University of Amsterdam, The Netherlands.
%  All rights reserved.
%
%  Author(s):
%  Dennis Koelma (koelma@wins.uva.nl)


echo on

% Read c:/images/bnoise.tif
im = OPS.HxMakeFromFile('c:/images/bnoise.tif');

% Applying percentile filter
result = OPS.HxPercentile(im, 3, 0.5);
```

```
% Writing result to c:/output/blittlenoise.tif
OPS.HxWriteFile(result, 'c:/output/blittlenoise.tif');
```

and executed with `hxMainDemoImageOps` .

Horus images may be converted to arrays to manipulate the image data using Matlab functionality.

```
a = hxImToArray(im);
```

produces a Matlab array. The array is visualized via

```
imagesc(a)
```

`hxShow(im)` uses `hxImToArray` to show a Horus image in Matlab. The function also handles color images in RGB format. For other color models `hxGetRgb2d` can be used to display the image. However, `hxGetRgb2d` does not do array ordering conversion so you will see the image on its side.

3.4 Calling image operations

In Horus, image data is stored in an **HxImageRep**. The member functions of HxImageRep correspond to the generic image operations defined in **n_genImOp**. In the end, all image processing is based on these member functions.

Often used image processing functionality is typically implemented as a global (C++) function. The global functions are easier to use as they hide the general parameter handling mechanism and other complex issues associated with the calling of generic image operations via HxImageRep member functions. Global image functions provides an overview of the current set of global C++ functions.

Next, we will give some examples of how to use the global functions and indicate how they are implemented using generic functions so that you get an idea of how to do this yourself.

- **Reading images from file** (p. 14)
- **An example of a binary pixel operation: Adding images** (p. 14)
- **An example of a unary pixel operation: Thresholding images** (p. 18)
- **An example of a generalized convolution: Uniform filter** (p. 20)
- **An example of a generalized convolution: Convolution** (p. 22)
- **An example of a generalized convolution: Erosion** (p. 24)

3.4.1 Reading images from file

Reading images from file is done by calling **HxMakeFromFile** or HxMakeFromFileSI. Both use the **HxImageFactory** to construct an HxImageRep. HxMakeFromFile uses the Horus HxImgFileIo dll while HxMakeFromFileSI uses Image2.1.

As typing file names is cumbersome the MidApp also supports the typical opening of files via the 'File' menu. Currently, images can be read both via Horus ('Open Server Image') and Java ('Open Image Java'). Be aware that images read via Java are not present in C++. So, you can not use them in the demo functions unless you convert them to Horus images via 'Java image to HxImageRep' in the 'Object' menu.

3.4.2 An example of a binary pixel operation: Adding images

Addition of images is implemented by the global C++ function **HxAdd**:

```
/*
 *  Copyright (c) 2000, University of Amsterdam, The Netherlands.
 *  All rights reserved.
 *
 *  Author(s):
 *  Marc Navarro            (mnavarro@wins.uva.nl)
 *  Dennis Koelma           (koelma@wins.uva.nl)
 */

#include "HxAdd.h"
#include "HxPixMax.h"
#include "HxUnaryMax.h"

HxImageRep
HxAdd(HxImageRep im1, HxImageRep im2)
{
    HxString fname("HxAdd");

    if (im1.isNull())
    {
        HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVAI
        return HxImageRep();
    }
    if (im2.isNull())
    {
        HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVAI
        return HxImageRep();
    }

    if (im1.dimensionality() != im2.dimensionality())
    {
        HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_
        return HxImageRep();
    }
    if (im1.pixelDimensionality() != im2.pixelDimensionality())
    {
        HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionalities", HxGlobalError::HX_
        return HxImageRep();
    }

    if (im1.sizes().x() != im2.sizes().x())
    {
        HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAI
        return HxImageRep();
    }
    if (im1.sizes().y() != im2.sizes().y())
    {
        HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUA
        return HxImageRep();
    }
    if (im1.dimensionality() > 2)
    {
        if (im1.sizes().z() != im2.sizes().z())
        {
            HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNE
            return HxImageRep();
        }
    }

    // in case of byte, unsigned: generate warnings in case of potentially dangerous
    // situations.
    // Check if image is byte.

    if (((im1.signature().pixelType() == INT_VALUE) &&
         (im1.signature().pixelPrecision() == 8)) ||
```

```
        ((im2.signature().pixelType() == INT_VALUE) &&
         (im2.signature().pixelPrecision() == 8)))
    {
        if ((im1.pixelDimensionality() == 1) && (im1.pixelDimensionality() == 1))
        {
            if ((HxPixMax(im1).HxScalarIntValue() +
                 HxPixMax(im2).HxScalarIntValue()) > HxScalarInt(255))
            {
                HxGlobalError::instance()->reportWarning(fname,
                    im1.name()+HxString(" ")+im2.name(),
                    "possible overflow due to byte precision",
                    HxGlobalError::HX_GW_OVERFLOW);
            }
        }
        else if ((HxPixMax(HxUnaryMax(im1)).HxScalarIntValue() +
                  HxPixMax(HxUnaryMax(im2)).HxScalarIntValue()) > HxScalarInt(255))
        {
                HxGlobalError::instance()->reportWarning(fname,
                    im1.name()+HxString(" ")+im2.name(),
                    "possible overflow due to byte precision",
                    HxGlobalError::HX_GW_OVERFLOW);
        }
    }

    return im1.binaryPixOp(im2, "add");
}
```

As shown in the code, HxAdd is just a wrapper around the generic binary pixel operation. For an example of how to use a global function in a C++ application see **A first application** (p. 11).

MidApp

HxAdd can also be used via the menu and dialogue interface:

- Use 'File/Open Server Image' to read two images, e.g. trui.tif and cermet.tif.
- Select 'HxAdd' from the 'GlobalOps/Images/Arithmetic/Binary' menu.
- Select 'trui' for the first image and 'cermet' for the second.
- Press OK and you will see the result.

Note that at some points in the image overflow occurs due to the default display model. This is caused by the fact that an image representation (i.e. an instance of HxImageRep) does not know what the pixel values mean so all the default display model does is to map the pixel values directly onto the RGB values used in Java. You can change the display model of the image representation by right clicking on the image, selecting 'setDisplayMode' from the 'Run' menu and choosing a more appropriate display model, e.g. 'Stretch'. You can also set a default display model via 'Viewer/Default Display Mode'.

We can also execute the generic binary pixel operation directly:

- Select 'binaryPixOp' from the 'ImageRep' menu.
- Select 'trui' for the ImageRep object and 'cermet' for the ImageRep arg.
- Enter 'add' for the bpoName.

IDL script

Addition of images is demonstrated in *demoAdd.cs*

```
# Demo: shows how to add Horus images

println("Enter name of first image, e.g. c:/images/trui.tif");
```

```
name = getline();

println("Read image from disk");
a = OPS.HxMakeFromFile(name);

println("Enter name of second image, e.g. c:/images/cermet.tif");
name = getline();

println("Read image from disk");
b = OPS.HxMakeFromFile(name);

println("Now add the images using the global function HxAdd");
c = OPS.HxAdd(a, b);

println("Now add the images using the generic function");
d = a.binaryPixOp(b, "add", CTOR.emptyTagList());
```

and executed with `exec("x:/HxSamples/Scripts/demoAdd.cs");`.

Matlab

Addition of images is demonstrated in *demoAdd.m*

```
%demoAdd     Demo: shows how to add Horus images

echo on
clc

name = input('Enter name of first image, e.g. c:/images/trui.tif\n', 's');

% Read image from disk
a = OPS.HxMakeFromFile(name);

% Display the image
hxShow(a);

name = input('Enter name of second image, e.g. c:/images/cermet.tif\n', 's');

% Read image from disk
b = OPS.HxMakeFromFile(name);

% And display it
hxShow(b);

pause % Press any key to continue

% Now add the images using the global function HxAdd
c = OPS.HxAdd(a, b);

% And display it
hxShow(c);

pause % Press any key to continue

% Now add the images using the generic function
d = a.binaryPixOp(b, 'add', CTOR.emptyTagList);

% And display it
hxShow(c);
```

and executed with `demoAdd`.

3.4.3 An example of a unary pixel operation: Thresholding images

Thresholding of images is implemented by the function **HxThreshold**.

```
/*
 *  Copyright (c) 2000, University of Amsterdam, The Netherlands.
 *  All rights reserved.
 *
 *  Author(s):
 *  Marc Navarro            (mnavarro@wins.uva.nl)
 */

#include "HxThreshold.h"

HxImageRep
HxThreshold(HxImageRep im, HxValue level)
{
    HxString fname("HxThreshold");

    if (im.isNull())
    {
        HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALI
        return HxImageRep();
    }

    if (im.pixelDimensionality() == 1)
    {
        if ((level.tag() != HxValue::SI) && (level.tag() != HxValue::SD))
        {
            HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlo
        }
    }
    if (im.pixelDimensionality() == 2)
    {
        if ((level.tag() != HxValue::V2I) && (level.tag() != HxValue::V2D)
                                          && (level.tag() != HxValue::CPL))
        {
            HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlo
        }
    }
    if (im.pixelDimensionality() == 3)
    {
        if ((level.tag() != HxValue::V3I) && (level.tag() != HxValue::V3D))
        {
            HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlo
        }
    }

    HxTagList tags;
    HxAddTag(tags, "level", level);
    return im.unaryPixOp("threshold", tags);
}
```

This example introduces tags to pass parameters to the functor applied to each pixel in the image. Tags are used instead of parameters because they do not require changing the generic function signature. Tags are put in a **HxTagList** via the **HxAddTag** function. Be careful to give it the proper name, i.e. the same name as used in the functor.

The parameter is an **HxValue**. HxValue is able to represent all pixel types used in Horus.

MidApp

Read 'trui' from disk. Apply the function HxThreshold (menu 'GlobalOps/Images/Segmentation') with parameter 128. Set the display mode to 'Binary'.

In order to do thresholding via a generic function we first have to construct a taglist with the parameter. In general, Corba objects are constructed via a factory like mechanism since Corba doesn't have constructors. Once the taglist is constructed we can use it to add the parameter value and then pass it to the generic function.

- Select 'emptyTagList' from the 'HxCorba/Constructor/Taglist' menu. Enter 'tags' for the result name.
- Select 'addValue' from the 'HxCorba/TagList' menu. Enter 'level' for the name and '128' for the value.
- Select 'unaryPixOp' from the 'ImageRep' menu. Select 'trui' for the ImageRep object. Enter 'threshold' for the upoName.

IDL script

Thresholding of images is demonstrated in *demoThreshold.cs*

```
# Demo: shows how to threshold Horus images

println("Enter name of image, e.g. c:/images/trui.tif");
name = getline();

# Read image from disk
a = OPS.HxMakeFromFile(name);

# Get an empty taglist from the constructor
tags = CTOR.emptyTagList();

# Allocate an HxValue object that represents a scalar integer 128
value = HxCorba.PixValue(HxCorba.PixValueTag.SD, 128);

# Add parameter to the tag list
tags.addValue("level", value);

# Do the actual operation
b = a.unaryPixOp("threshold", tags);
```

and executed with `exec("x:/HxSamples/Scripts/demoThreshold.cs");`.

Matlab

Thresholding of images is demonstrated in *demoThreshold.m*

```
%demoThreshold  Demo: shows how to threshold Horus images

echo on
clc

name = input('Enter name of image, e.g. c:/images/trui.tif\n', 's');

% Read image from disk
a = OPS.HxMakeFromFile(name);

% Display the image
hxShow(a);

pause % Press any key to continue
```

```
% Get an empty taglist from the constructor
tags = CTOR.emptyTagList;

% Allocate an HxValue object that represents a scalar integer 128
value = HxCorba.PixValue;
value.scalarInt(128);

% Add parameter to the tag list
tags.addValue('level', value);

% Do the actual operation
b = a.unaryPixOp('threshold', tags);

% And display it
hxShow(b);
```

and executed with `demoThreshold`.

3.4.4 An example of a generalized convolution: Uniform filter

Uniform filtering is implemented by **HxUniform**. However, for this example we will use **HxUniformNonSep** since its code is a little simpler. HxUniformNonSep is a non separated version of HxUniform.

```
/*
 *  Copyright (c) 2001, University of Amsterdam, The Netherlands.
 *  All rights reserved.
 *
 *  Author(s):
 *  Dennis Koelma           (koelma@science.uva.nl)
 */

#include "HxUniformNonSep.h"
#include "HxMakeFromValue.h"

HxImageRep
HxUniformNonSep(HxImageRep im, HxSizes sizes)
{
    // An image signature for a 2D image with 64-bit real valued scalar pixels
    HxImageSignature sig(im.dimensionality(), 1, REAL_VALUE, 64);

    // The pixel value of the uniform kernel
    double val = 1.0 / (sizes.x() * sizes.y() * sizes.z());

    // Now construct the kernel image
    HxImageRep kernel = HxMakeFromValue(sig, sizes, val);

    // and apply the operation
    return im.generalizedConvolution(kernel, "mul", "addAssign");
}
```

Basically, a uniform filter is a convolution with a uniform kernel. A convolution is performed by supplying the generalized convolution with multiplication and addition as basic operations.

A uniform kernel image is constructed from the following parameters:

- An **HxImageSignature** to specify the type of image to be created,
- **HxSizes** to specify the extend of the image, and

- an **HxValue** to give the pixels in the image a value.

MidApp

Read 'flamingo' from disk. Apply the function HxUniformNonSep (menu 'GlobalOps/Images/Filter') with parameter '5 5'.

In order to do uniform filtering via a generic function we first have to construct the kernel image and then perform a convolution on the image via the generalized convolution.

- Select 'HxMakeFromValue' from the 'GlobalOps/Images/Generation' menu. Enter 'unif5x5' for the result name. Select 'SIG2DDOUBLE' for the signature. Enter '5 5' for the sizes and '0.04' for the value.
- Select 'generalizedConvolution' from the 'ImageRep' menu. Select 'flamingo' for the ImageRep object and 'unif5x5' for the kernel. Enter 'mul' for gMul and 'addAssign' for gAdd. Select 'SOURCE_PREC' for the precision and '_nil' for the taglist.

IDL script

Uniform filtering is demonstrated in *demoUniform.cs*

```
# Demo: shows how to apply a uniform 5x5 filter to Horus images

println("Enter name of image, e.g. c:/images/flamingo.tif");
name = getline();

# Read image from disk
a = OPS.HxMakeFromFile(name);

# An image signature for a 2D image with 64-bit real valued scalar pixels
sig = HxCorba.ImageSignature.SIG2DDOUBLE;

# Define the sizes of the neighbourhood
sizes = HxCorba.Sizes(5, 5, 1);

# The pixel value of the uniform kernel
val = HxCorba.PixValue(HxCorba.PixValueTag.SD, 1.0 / (sizes.x * sizes.y * sizes.z));

# Now construct the kernel image
kernel = OPS.HxMakeFromValue(sig, sizes, val);

# Apply the operation
empty = CTOR.emptyTagList();
prec = HxCorba.ResultPrecision.ARITH_PREC;
b = a.generalizedConvolution(kernel, "mul", "addAssign", prec, empty);
```

and executed with `exec("x:/HxSamples/Scripts/demoUniform.cs");`.

Matlab

Uniform filtering is demonstrated in *demoUniform.m*

```
%demoUniform  Demo: shows how to apply a uniform 5x5 filter to Horus images

echo on
clc

name = input('Enter name of image, e.g. c:/images/flamingo.tif\n', 's');

% Read image from disk
```

```
a = OPS.HxMakeFromFile(name);

% Display the image
hxShow(a);

pause % Press any key to continue

% An image signature for a 2D image with 64-bit real valued scalar pixels
sig = HxCorba.ImageSignature.SIG2DDOUBLE;

% Define the sizes of the neighbourhood
sizes = HxCorba.Sizes(5, 5, 1);

% The pixel value of the uniform kernel
val = HxCorba.PixValue;
val.scalarDouble(1.0 / (sizes.x * sizes.y * sizes.z));

% Now construct the kernel image
kernel = OPS.HxMakeFromValue(sig, sizes, val);

% Apply the operation
empty = CTOR.emptyTagList;
% Due to a bug in Matlab, the following statement crashes in a script.
%prec = HxCorba.ResultPrecision.ARITH_PREC;
prec = HxCorba.ResultPrecision.from_int(1);
b = a.generalizedConvolution(kernel, 'mul', 'addAssign', prec, empty);

% And display it
hxShow(b);
```

and executed with `demoUniform`.

3.4.5 An example of a generalized convolution: Convolution

To demonstrate the use convolutions with user-defined kernels we 'implement' a simple derivate in the x-direction (kernel is [-1 0 1]) and a Laplacian filter.

IDL script

Convolution is demonstrated in *demoConvolution.cs*

```
# Demo: shows how to do convolutions on Horus images

println("Enter name of image, e.g. c:/images/trui.tif");
name = getline();

# Read image from disk
a = OPS.HxMakeFromFile(name);

# Construct a 2D kernel with 32-bit integer valued scalar pixels
# initialized with the values -1, 0, and 1
data = [-1, 0, 1];
kernel = OPS.HxMakeFromIntData(1, 2, HxCorba.Sizes(3, 1, 1), data);

# Do a convolution
empty = CTOR.emptyTagList();
prec = HxCorba.ResultPrecision.ARITH_PREC;
b = a.generalizedConvolution(kernel, "mul", "addAssign", prec, empty);

# Construct a 2D kernel with 32-bit integer valued scalar pixels
# initialized with:
```

```
#  0 -1  0
# -1  4 -1
#  0 -1  0
data = [0, -1, 0, -1, 4, -1, 0, -1, 0];
kernel = OPS.HxMakeFromIntData(1, 2, HxCorba.Sizes(3, 3, 1), data);

# Do a convolution
c = a.generalizedConvolution(kernel, "mul", "addAssign", prec, empty);
```

and executed with `exec("x:/HxSamples/Scripts/demoConvolution.cs");`.

Matlab

Convolution is demonstrated in *demoConvolution.m*

```
%demoConvolution  Demo: shows how to do convolutions on Horus images

echo on
clc

name = input('Enter name of image, e.g. c:/images/trui.tif\n', 's');

% Read image from disk
a = OPS.HxMakeFromFile(name);

% Display the image
hxShow(a);

pause % Press any key to continue

% Construct a 2D kernel with 32-bit integer valued scalar pixels
% initialized with the values -1, 0, and 1
data = [-1 0 1];
kernel = OPS.HxMakeFromIntData(1, 2, HxCorba.Sizes(3, 1, 1), data);

% Do a convolution
empty = CTOR.emptyTagList;
% Due to a bug in Matlab, the following statement crashes in a script.
%prec = HxCorba.ResultPrecision.ARITH_PREC;
prec = HxCorba.ResultPrecision.from_int(1);
b = a.generalizedConvolution(kernel, 'mul', 'addAssign', prec, empty);

% And display it
hxShow(b);

pause % Press any key to continue

% Construct a 2D kernel with 32-bit integer valued scalar pixels
% initialized with:
%  0 -1  0
% -1  4 -1
%  0 -1  0
data = [0 -1 0 -1 4 -1 0 -1 0];
kernel = OPS.HxMakeFromIntData(1, 2, HxCorba.Sizes(3, 3, 1), data);

% Do a convolution
c = a.generalizedConvolution(kernel, 'mul', 'addAssign', prec, empty);

% And display it
hxShow(c);
```

and executed with `demoConvolution`.

3.4.6 An example of a generalized convolution: Erosion

The scripting languages may also be used to define functions that have no C++ counterpart (yet). The function 'myErosion' uses generalized onvolution to perform a classical erosion with a flat structuring element. The erosion is performed by supplying the generalized convolution with 'add' and 'minAssign' as basic operations.

IDL script

Erosion is demonstrated in *demoErosion.cs*

```
# Demo: shows how to do erosions on Horus images

# Definition of my own erosion function
#
# im is an ImageRep
# size is an int
#
proc myErosion(im, size)
{
    # Use the same signature as the input image
    sig = im.signature();

    # Define the sizes of the structuring element
    sizes = HxCorba.Sizes(size, size, 1);

    # The pixel value of the structuring element
    val = HxCorba.PixValue(HxCorba.PixValueTag.SI, 0);

    # Now construct the structuring element
    kernel = OPS.HxMakeFromValue(sig, sizes, val);

    # And apply the operation, using minimum
    empty = CTOR.emptyTagList();
    prec = HxCorba.ResultPrecision.ARITH_PREC;
    return im.generalizedConvolution(kernel, "add", "minAssign", prec, empty);

    # Could also use infimum, produces slightly different results.
    #return im.generalizedConvolution(kernel, "add", "infAssign", prec, empty);
}


println("Enter name of image, e.g. c:/images/flamingo.tif");
name = getline();

# Read image from disk
a = OPS.HxMakeFromFile(name);

# Apply my own erosion
b = myErosion(a, 5);
```

and executed with `exec("x:/HxSamples/Scripts/demoErosion.cs");`.

Matlab

Erosion is demonstrated in *demoErosion.m*

```
%demoErosion  Demo: shows how to do erosions on Horus images

echo on
clc
```

```
name = input('Enter name of image, e.g. c:/images/flamingo.tif\n', 's');

% Read image from disk
a = OPS.HxMakeFromFile(name);

% Display the image
hxShow(a);

pause % Press any key to continue

% Apply my own erosion
b = myErosion(a, 5);

% And display it
hxShow(b);
```

and executed with `demoErosion`.

The function myErosion is defined in *myErosion.m*

```
function r = myErosion(im, size)

%myErosion    Definition of my own erosion function
%    im is a Horus image (Java stub).
%    size is an int.

global CTOR
global OPS

% Use the same signature as the input image
sig = im.signature;

% Define the sizes of the structuring element
sizes = HxCorba.Sizes(size, size, 1);

% The pixel value of the structuring element
val = HxCorba.PixValue;
val.scalarInt(0);

% Now construct the structuring element
kernel = OPS.HxMakeFromValue(sig, sizes, val);

% And apply the operation, using minimum
empty = CTOR.emptyTagList;
% Due to a bug in Matlab, the following statement crashes in a script.
%prec = HxCorba.ResultPrecision.ARITH_PREC;
prec = HxCorba.ResultPrecision.from_int(1);
r = im.generalizedConvolution(kernel, 'add', 'minAssign', prec, empty);

% Could also use infimum, produces slightly different results.
%r = im.generalizedConvolution(kernel, 'add', 'infAssign', prec, empty);
```

3.5 Instantiating patterns

In this section the basics of instantiation of an image princessing pattern are explained with some examples.

- **Instantiation of a binary pixel operation: Squared distance** (p. 26)
- **Instantiation of a unary pixel operation: Tri state threshold** (p. 28)
- **Instantiation of an in/out operation : Image to histogram** (p. 31)

3.5.1 Instantiation of a binary pixel operation: Squared distance

The file *HxImageRep/Global/HxSquaredDistance.c* provides a complete overview of all that is needed to implement a function that computes the squared distance between corresponding pixels in two images:

```
/*
 *  Copyright (c) 1999, University of Amsterdam, The Netherlands.
 *  All rights reserved.
 *
 *  Author(s):
 *  Dennis Koelma (koelma@wins.uva.nl)
 *  Edo Poll (poll@wins.uva.nl)
 *  Marc Navarro (mnavarro@wins.uva.nl)
 *
 */


#include "HxSquaredDistance.h"
#include "HxImgFtorBpo.h"
#include "HxIncludedSigs.h"


/** Pixel functor for computation of squared distance
 */
template<class DstValT, class Src1ValT, class Src2ValT>
class HxBpoSqrDst
{
public:
    /** Functor is translation invariant */
    typedef HxTagTransInVar TransVarianceCategory;

                    /** Constructor : empty */
                    HxBpoSqrDst(HxTagList&)
                        { }

                    /** Actual operation : # return (x - y)^2 # */
    inline DstValT  doIt(const Src1ValT& arg1, const Src2ValT& arg2)
                        { return (arg1 - arg2) * (arg1 - arg2); }

                    /** The name : "sqrDst" */
    static HxString className()
                        { return HxString("sqrDst"); }
};


/** Instantiator for binary pixel operation with squared distance */
template<class ImgSigT>
struct HxInstantiatorSqrDst
{
    /** Instantiate image functor */
    HxImgFtorBpo<ImgSigT, ImgSigT, ImgSigT,
        HxBpoSqrDst<typename ImgSigT::ArithType, typename ImgSigT::ArithType,
                    typename ImgSigT::ArithType> > f;
};


// Put static variables in a namespace to avoid conflicts with file inclusion
namespace HxSquaredDistance_c {

/** Call instantiator for 2dByte images */
static HxInstantiatorSqrDst<HxImageSig2dByte>           f001;
/** Call instantiator for 2dShort images */
static HxInstantiatorSqrDst<HxImageSig2dShort>          f002;
/** Call instantiator for 2dInt images */
```

```
static HxInstantiatorSqrDst<HxImageSig2dInt>            f003;
/** Call instantiator for 2dFloat images */
static HxInstantiatorSqrDst<HxImageSig2dFloat>          f004;
/** Call instantiator for 2dDouble images */
static HxInstantiatorSqrDst<HxImageSig2dDouble>         f005;
/** Call instantiator for 2dVec2Byte images */
static HxInstantiatorSqrDst<HxImageSig2dVec2Byte>       f006;
/** Call instantiator for 2dVec2Short images */
static HxInstantiatorSqrDst<HxImageSig2dVec2Short>      f007;
/** Call instantiator for 2dVec2Int images */
static HxInstantiatorSqrDst<HxImageSig2dVec2Int>        f008;
/** Call instantiator for 2dVec2Float images */
static HxInstantiatorSqrDst<HxImageSig2dVec2Float>      f009;
/** Call instantiator for 2dVec2Double images */
static HxInstantiatorSqrDst<HxImageSig2dVec2Double>     f010;
/** Call instantiator for 2dVec3Byte images */
static HxInstantiatorSqrDst<HxImageSig2dVec3Byte>       f011;
/** Call instantiator for 2dVec3Short images */
static HxInstantiatorSqrDst<HxImageSig2dVec3Short>      f012;
/** Call instantiator for 2dVec3Int images */
static HxInstantiatorSqrDst<HxImageSig2dVec3Int>        f013;
/** Call instantiator for 2dVec3Float images */
static HxInstantiatorSqrDst<HxImageSig2dVec3Float>      f014;
/** Call instantiator for 2dVec3Double images */
static HxInstantiatorSqrDst<HxImageSig2dVec3Double>     f015;
/** Call instantiator for 3dByte images */
static HxInstantiatorSqrDst<HxImageSig3dByte>           f016;
/** Call instantiator for 3dShort images */
static HxInstantiatorSqrDst<HxImageSig3dShort>          f017;
/** Call instantiator for 3dInt images */
static HxInstantiatorSqrDst<HxImageSig3dInt>            f018;
/** Call instantiator for 3dFloat images */
static HxInstantiatorSqrDst<HxImageSig3dFloat>          f019;
/** Call instantiator for 3dDouble images */
static HxInstantiatorSqrDst<HxImageSig3dDouble>         f020;

}; // end of namespace


/* Global function to do squared distance.
 */
HxImageRep
HxSquaredDistance(HxImageRep im1, HxImageRep im2)
{
    HxString fname("HxSquaredDistance");

    if (im1.isNull())
    {
        HxGlobalError::instance()->reportError(fname, im1.name(), "null image", HxGlobalError::HX_GE_INVAI
        return HxImageRep();
    }
    if (im2.isNull())
    {
        HxGlobalError::instance()->reportError(fname, im2.name(), "null image", HxGlobalError::HX_GE_INVAI
        return HxImageRep();
    }

    if (im1.dimensionality() != im2.dimensionality())
    {
        HxGlobalError::instance()->reportError(fname, "unequal image dimensionalities", HxGlobalError::HX_
        return HxImageRep();
    }
    if (im1.pixelDimensionality() != im2.pixelDimensionality())
    {
        HxGlobalError::instance()->reportError(fname, "unequal pixel dimensionality", HxGlobalError::HX_GE
```

```
        return HxImageRep();
    }

    if (im1.sizes().x() != im2.sizes().x())
    {
        HxGlobalError::instance()->reportError(fname, "unequal image widths", HxGlobalError::HX_GE_UNEQUAI
        return HxImageRep();
    }

    if (im1.sizes().y() != im2.sizes().y())
    {
        HxGlobalError::instance()->reportError(fname, "unequal image heights", HxGlobalError::HX_GE_UNEQUA
        return HxImageRep();
    }
    if (im1.dimensionality() > 2)
    {
        if (im1.sizes().z() != im2.sizes().z())
        {
            HxGlobalError::instance()->reportError(fname, "unequal image depths", HxGlobalError::HX_GE_UNE
            return HxImageRep();
        }
    }

    // call HxImageRep member function to do the image processing
    return im1.binaryPixOp(im2, "sqrDst");
}
```

Instantiation of a new binary pixel operation basically includes the following steps:

- Define a functor that takes two pixel values and produces a result.

  In this example the function computes the squared distance between two pixel values. The functor is defined by class **HxBpoSqrDst**.
- Instantiate the binary pixel functor algorithm **HxImgFtorBpo** for all image types.

  The binary pixel functor will apply the user defined pixel operation to all pixels in an image. The functor must be instantiated for all image types separately so we use a helper class **HxInstantiator-SqrDst** to do instantiation for one image type, and ”call” the instantiator for each image type via declaration of a number of static variables (f001, f002, etc.).
- Write a global function to provide easy access to the new function.

  This is taken care of by **HxSquaredDistance**.

Extension of generic functions is to be based on arithmetic data types to ensure the operations are applicable to all image types. The set of operations defined for each data type is given in Pixels. For the exact syntax of the operations we refer to the class definitions: **HxScalarInt**, **HxScalarDouble**, **HxVec2Int**, **HxVec2Double**, **HxVec3Int**, and **HxVec3Double**.

3.5.2 Instantiation of a unary pixel operation: Tri state threshold

The file *HxImageRep/Global/HxTriStateThreshold.c* provides a complete overview of all that is needed to implement a function that performs a tri state threshold:

```
/*
 *  Copyright (c) 2000, University of Amsterdam, The Netherlands.
 *  All rights reserved.
 *
 *  Author(s):
```

```
 *  Dennis Koelma (koelma@wins.uva.nl)
 *
 */


#include "HxTriStateThreshold.h"
#include "HxImgFtorUpo.h"
#include "HxIncludedSigs.h"


/** Pixel functor for computation of tri state threshold
 */
template<class DstValT, class SrcValT>
class HxUpoTriStateThreshold
{
public:
    /** Functor is translation invariant */
    typedef HxTagTransInVar TransVarianceCategory;

                        /** Constructor : get parameters from taglist */
                        HxUpoTriStateThreshold(HxTagList&);

                        /** Actual operation. */
    DstValT             doIt(const SrcValT& x);

                        /** The name : "triStateThreshold" */
    static HxString     className();

private:
    SrcValT             _level;
    SrcValT             _v1;
    SrcValT             _v2;
    SrcValT             _v3;
};

template<class DstValT, class SrcValT>
HxUpoTriStateThreshold<DstValT, SrcValT>::HxUpoTriStateThreshold(HxTagList& tl)
{
    _level = HxGetTag<HxValue>(tl, "level");
    _v1 = HxGetTag<HxValue>(tl, "v1");
    _v2 = HxGetTag<HxValue>(tl, "v2");
    _v3 = HxGetTag<HxValue>(tl, "v3");
}

template<class DstValT, class SrcValT>
inline DstValT
HxUpoTriStateThreshold<DstValT, SrcValT>::doIt(const SrcValT& x)
{
    if (x < _level)
        return _v1;
    if (x > _level)
        return _v3;
    return _v2;
}

template<class DstValT, class SrcValT>
HxString
HxUpoTriStateThreshold<DstValT, SrcValT>::className()
{
    return HxString("triStateThreshold");
}


/** Instantiator for unary pixel operation with tri state threshold */
template<class ImgSigT>
```

```
struct HxInstantiatorTriStateThreshold
{
    /** Instantiate image functor */
    HxImgFtorUpo<ImgSigT, ImgSigT,
        HxUpoTriStateThreshold<
                typename ImgSigT::ArithType, typename ImgSigT::ArithType> > f;
};


// Put static variables in a namespace to avoid conflicts with file inclusion
namespace HxTriStateThreshold_c {

/** Call instantiator for 2dByte images */
static HxInstantiatorTriStateThreshold<HxImageSig2dByte>          f001;
/** Call instantiator for 2dShort images */
static HxInstantiatorTriStateThreshold<HxImageSig2dShort>         f002;
/** Call instantiator for 2dInt images */
static HxInstantiatorTriStateThreshold<HxImageSig2dInt>           f003;
/** Call instantiator for 2dFloat images */
static HxInstantiatorTriStateThreshold<HxImageSig2dFloat>         f004;
/** Call instantiator for 2dDouble images */
static HxInstantiatorTriStateThreshold<HxImageSig2dDouble>        f005;
/** Call instantiator for 2dVec2Byte images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec2Byte>      f006;
/** Call instantiator for 2dVec2Short images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec2Short>     f007;
/** Call instantiator for 2dVec2Int images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec2Int>       f008;
/** Call instantiator for 2dVec2Float images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec2Float>     f009;
/** Call instantiator for 2dVec2Double images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec2Double>    f010;
/** Call instantiator for 2dVec3Byte images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec3Byte>      f011;
/** Call instantiator for 2dVec3Short images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec3Short>     f012;
/** Call instantiator for 2dVec3Int images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec3Int>       f013;
/** Call instantiator for 2dVec3Float images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec3Float>     f014;
/** Call instantiator for 2dVec3Double images */
static HxInstantiatorTriStateThreshold<HxImageSig2dVec3Double>    f015;
/** Call instantiator for 3dByte images */
static HxInstantiatorTriStateThreshold<HxImageSig3dByte>          f016;
/** Call instantiator for 3dShort images */
static HxInstantiatorTriStateThreshold<HxImageSig3dShort>         f017;
/** Call instantiator for 3dInt images */
static HxInstantiatorTriStateThreshold<HxImageSig3dInt>           f018;
/** Call instantiator for 3dFloat images */
static HxInstantiatorTriStateThreshold<HxImageSig3dFloat>         f019;
/** Call instantiator for 3dDouble images */
static HxInstantiatorTriStateThreshold<HxImageSig3dDouble>        f020;

};


/* Global function to do tri state threshold.
 */
HxImageRep
HxTriStateThreshold(HxImageRep im, HxValue level, HxValue v1, HxValue v2, HxValue v3)
{
    HxString fname("HxTriStateThreshold");

    if (im.isNull())
    {
```

```
        HxGlobalError::instance()->reportError(fname, im.name(), "null image", HxGlobalError::HX_GE_INVALI
        return HxImageRep();
    }

    if (im.pixelDimensionality() == 1)
    {
        if ((level.tag() != HxValue::SI) && (level.tag() != HxValue::SD))
        {
            HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlo
        }
    }
    if (im.pixelDimensionality() == 2)
    {
        if ((level.tag() != HxValue::V2I) && (level.tag() != HxValue::V2D)
                                          && (level.tag() != HxValue::CPL))
        {
            HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlo
        }
    }
    if (im.pixelDimensionality() == 3)
    {
        if ((level.tag() != HxValue::V3I) && (level.tag() != HxValue::V3D))
        {
            HxGlobalError::instance()->reportError(fname, im.name(), "level is of wrong value type", HxGlo
        }
    }

    // Put all non-image parameters in a TagList
    HxTagList tags;
    HxAddTag(tags, "level", level);
    HxAddTag(tags, "v1", v1);
    HxAddTag(tags, "v2", v2);
    HxAddTag(tags, "v3", v3);

    // call HxImageRep member function to do the image processing
    return im.unaryPixOp("triStateThreshold", tags);
}
```

Tri state threshold assigns three values to output pixels instead of two (0 and 1) like the normal threshold does. All pixels below the threshold level will be set to the first value, pixels equal to the threshold level will be set to the second value, and all other pixel values to the third value.

Extension of the generic unary pixel operation is basically the same as extension of the generic binary pixel operation. So, we implement a functor **HxUpoTriStateThreshold** to do our operation on one pixel and we instantiate the unary pixel algorithm **HxImgFtorUpo** for all image types. However, as the tri state threshold function requires more parameters than the usual unary pixel operation (which has only one: the pixel value) we have to use the tag list to pass the additional parameters to the pixel functor.

The tag list is passed to the constructor of the pixel functor (HxUpoTriStateTreshold in this case) by the generic unary pixel operation HxImgFtorUpo. In the constructor we save the parameters in private member variables. As this is a functor, the parameters can be used again when the unary pixel operation calls the 'doIt' operation with a pixel value to be processed.

3.5.3 Instantiation of an in/out operation : Image to histogram

The file *HxDetector/Global/HxImageToHistogram.c* provides a complete overview of all that is needed to make a hisogram of an image:

```
/*
```

```
 *  Copyright (c) 2000, University of Amsterdam, The Netherlands.
 *  All rights reserved.
 *
 *  Author(s):
 *  Dennis Koelma (koelma@wins.uva.nl)
 *
 */

#include "HxImageToHistogram.h"
#include "HxImgFtorInOut.h"
#include "HxIncludedSigs.h"

template<class ValT>
class HxInOutHistogram
{
public:
    typedef HxTagPixOpOut       DirectionCategory;
    typedef HxTagTransInVar     TransVarianceCategory;
    typedef HxTag1Phase         PhaseCategory;

                        HxInOutHistogram(HxTagList&);

    void                doIt(const ValT& pixV);

    static HxString     className();

private:
    HxHistogram*        _hist;
    int                 _getDim;
};

template<class ValT>
HxInOutHistogram<ValT>::HxInOutHistogram(HxTagList& tags)
{
    _hist = HxGetTag(tags, "histogram", (HxHistogram*)0);
    _getDim = HxGetTag<int>(tags, "getDim");
}

template<class ValT>
inline void
HxInOutHistogram<ValT>::doIt(const ValT& pixV)
{
    if (_getDim <= 0)
        _hist->insertVal(pixV);
    else
        _hist->insertVal(((ValT) pixV).getValue(_getDim));
}

template<class ValT>
HxString
HxInOutHistogram<ValT>::className()
{
    return HxString("histogram");
}

template<class ImgSigT>
struct HxInstantiatorHistogram
{
    HxImgFtorInOut<ImgSigT, HxInOutHistogram<typename ImgSigT::ArithType> > f;
};

HxAlwaysInstantiateHistogram::HxAlwaysInstantiateHistogram()
{
}
```

```
namespace HxImageToHistogram_c {
static HxInstantiatorHistogram<HxImageSig2dByte>           f001;
static HxInstantiatorHistogram<HxImageSig2dShort>          f002;
static HxInstantiatorHistogram<HxImageSig2dInt>            f003;
static HxInstantiatorHistogram<HxImageSig2dFloat>          f004;
static HxInstantiatorHistogram<HxImageSig2dDouble>         f005;
static HxInstantiatorHistogram<HxImageSig2dVec2Byte>       f006;
static HxInstantiatorHistogram<HxImageSig2dVec2Short>      f007;
static HxInstantiatorHistogram<HxImageSig2dVec2Int>        f008;
static HxInstantiatorHistogram<HxImageSig2dVec2Float>      f009;
static HxInstantiatorHistogram<HxImageSig2dVec2Double>     f010;
static HxInstantiatorHistogram<HxImageSig2dVec3Byte>       f011;
static HxInstantiatorHistogram<HxImageSig2dVec3Short>      f012;
static HxInstantiatorHistogram<HxImageSig2dVec3Int>        f013;
static HxInstantiatorHistogram<HxImageSig2dVec3Float>      f014;
static HxInstantiatorHistogram<HxImageSig2dVec3Double>     f015;
static HxInstantiatorHistogram<HxImageSig3dByte>           f016;
static HxInstantiatorHistogram<HxImageSig3dShort>          f017;
static HxInstantiatorHistogram<HxImageSig3dInt>            f018;
static HxInstantiatorHistogram<HxImageSig3dFloat>          f019;
static HxInstantiatorHistogram<HxImageSig3dDouble>         f020;
};


HxHistogram
HxImageToHistogram(HxImageRep im, int getDim, double lowBin,
    double highBin, int nBin)
{
    int nDim = (getDim <= 0) ? im.pixelDimensionality() : 1;
    HxHistogram hist = HxHistogram(REAL_VALUE, nDim, lowBin, highBin, nBin,
                                                     lowBin, highBin, nBin,
                                                     lowBin, highBin, nBin);

    HxTagList tags;
    HxAddTag(tags, "histogram", &hist);
    HxAddTag(tags, "getDim", getDim);

    im.exportOp("histogram", tags);

    return hist;
}
```