Abstract

The goal of the project is to implement an Open Source toolbox for multivariate image analysis to enable dynamic and flexible mapping between pixel and statistical domain plots. The simplest approach to mapping is brushing where corresponding data object in different plots are highlighted. However more general mappings are necessary, for instance where the underlying latent model (from e.g. a partial least squares regression or principal component analysis) is modified interactively to see how this affect various statistical plots. A simple example is dynamically changing the number of latent variables used in the model.

Various filtering methods (f.x. noise filtering, region weighting, non-linear transforms) should also be implemented for changing the original hyperspectral images interactivly and analyzed as to how they affect the final results.

Preface

This paper was written as my final Master Thesis to complete my sivilingeniør, or Master of Technology, grade at the Norwegian University of Technology and Science (NTNU).

I would like to thank my advisor Professor Richard Blake. I would especially like to thank my co-advisor professor Bjørn Alsberg who guided me through this work. Sadly, due to comminucation issues and the author of this paper spending a lot of time sick, neither Richard Blake nor Bjørn Alsberg got much input into the actual writing of this paper.

I would also like to thank the company NEO in Oslo for sharing rawdata with us and thus making it actually possible to test what was made.

As all foreigners, my language is a weird mix between British and American spelling. I have strived to write this paper using British spelling, and I apologise for any mispellings. Furthermore, I have strived to only provide citations where they are relevant and where the text in question actually refers to the citation.

Lastly, at the time of writing this preface I am living Moscow, Russia. It truely is a beautifull country.

Vennlig hilsen
Reidar Strand Hagen

Contents

List of Figures

Chapter 1

Introduction

This chapter contains a verbose introduction, an overview of the task and the motivation, aswell as a rundown of the final results.

1.1 Verbose Introduction

By definition, an image is a reproduction of a scene or an object. A reproduction does only preserve some qualities of its original, f.x. the statue of Jean D'Arc in

Paris may preserve form, however it does not preserve texture or colour, and most likely not smell. More relevant to this paper, a digital reproduction, ie a computer image, will preserve lighting, and to some degree colour and form. Colour is only partially preserved due to needing to have the exact same wavelenght spread of the original colour to behave exactly similar under different lighting conditions. Using regular three datapoint-images this is very unlikely to happen.

Figure 1.1: Joan D'Arc

With light containing many different wavelengths with varying intensities, there are very, very many different combinations of light spreads. Our eyes, on the other hand, only contain 3 different types of colour-receptors, cones, each giving of one single response. By consequence, by using our eyes as the standard one only needs three datapoints to properly distinguish colours. Since we can only measure about 80 amounts of grey-levels on a computer screen, these three don't even need all that much resolution to be indistingishuable from eachother either. Essentially, by using our

eyes as the standard for how much colour information to include, we're setting the bar quite low when doing data-analysis.

While special equipment is needed for gathering full spectral information, equipment is fast dropping in price. Keeping light-spread information increases the amount of information rather drastically though, and the mathemathical tools for analysis and transformation become much more expensive to use. Methods which scale f.x. $O(N^3)$ fast become infeasible. Treatment of large images is however what Multivariate Image Analysis is all about.

1.2 MultiVariate Image Analysis

There are several good introductions for Multivariate Image Analysis. The aptly named paper 'Multivariate Image Analysis (MIA)'[13] and the book 'Multivariate Image Analysis'[3] are good starting points.

Multivariate Imaging stem from a variate of researching fields, all having the common factor of handling large amounts of data with a planar position. The criterium for what is considered multivariate dataset is thus having atleast 3 dimensions of some size with two being spatial. Multivariate Imaging is thus concerned with how to understand, visualize and analyse large data volumes, aswell as the number crunching algorithms needed for reducing the datasets.

As listed in [13], multivariate images can come from a variety of sources, images from f.x. satelites, or other sources not having anything to do with light at all (secondary ion mass spectroscopy, SIMS). Almost any physical unit can be used to make images and multivariate images: temperature, gravitational field, impedance, magnetic field, electrical field, mass, wavelength, ultrasound wavelength, polarization, electron energy etc. As such, it has been a field of research for quite some time, however some appliances used, such as wavelets, are rather new.

1.3 Problem Description

Excerpt from the Abstract:

> The end goal of the project is to implement an Open Source toolbox for multivariate image analysis to enable dynamic and flexible mapping between pixel and statistical domain plots.
> ...
> Various filtering methods (f.x. noise filtering, region weighting, nonlinear transforms) should also be implemented for changing the original hyperspectral images interactivly and analyzed as to how they affect the final results.

It was decided early on to integrate the toolkit with Scicraft. Scicraft is a middle-tier program for visualization and glueing methods together, developed

at Chemometrics and Bioinformatics Group (CBG) in the Department of Chemistry at Norwegian University of Science and Technology (NTNU). As such, the tasks at can be summarised as:

1. Creating a toolkit with the most common methods required for multivariate image analysis.
2. Set up basic datastructures, including support for mappings between datasets and images
3. Integrate this toolkit with Scicraft, and where necessary improve Scicraft to handle what's offered from the toolkit.

1.3.1 Needed Features

File Input / Output ; Ability to read various scientific and regular image formats is needed. Some of this is present in Scicraft already.

Various Transforms ; Mathemathical transforms such as PCA, PLSR etc are already available through Scicraft. More Imaging specific methods such as filters need to be implemented.

Visualisation ; methods involving the preparation of data for visualisation, aswell as the actual visualisation

Mapping ; methods for setting the relationship between plots, aswell as visualising them

1.4 Problem Motivation

The gist of the motivation is quite simple. There exists, as far as the author of this paper is aware, currently no open implentations of Multivariate Imaging toolkits. There exists freely available toolkits and there are commercial appliances, however neither of these offer the flexibility of actually being able to read and change yourself what is done with your data.

Addionally, multivariate image analysis on regular workstations are only just becomes feasible, due to larger amounts of memory becoming the norm.

1.4.1 Computer Requirements

As dataset grows large, the CPU-time needed for transformations will usually increase drastically. However, this can be offset by either using simpler tranforms or ones that scale better. Either way, it simply means you have to wait longer.

What is harder to adjust for however, is memory requirements. When making transforms, one need at the barest minimum atleast enough memory to hold two copies of the dataset, preferably a lot more. While swap-memory basically gives a computer unlimited memory, it is impossible to efficiently switch

memory when the amount of memory being activly used actually exceeds the amount of physical memory. As such, excessive memory usage will cause severe slowdowns due to harddrive-IO. A 512x512 image with 512 image bands using shorts (2byte) will by itself use 256MB of memory just for storing one copy.

While mainframes and dataclusters usually do have copious amounts of memory normal Workstations have not. However, with memory sizes +1GB becoming more and more common on scientific Workstations Multivariate Imaging becomes more and more feasable.

1.4.2 Existing Software

While several packages and related software exists, the two most important are

TNT ; commercial software primarly for Geo-spatial analysis. Offers a variety of methods for handling multivariate image data. [7]

CenSSIS Solutionware: Hyperspectral Image Analysis Toolbox ; Offered freely, but only precompiled sources offered. [6]

1.5 Results

This section contains a short overview of what has accomplished.

Regular Images : Basic operations needed, such as reading/saving images, folding/unfolding data, and plot-transforms such as autoscale and density-transform have been implemented. Dynamic mapping between transformed and original images is working.

Multivariate Images : Methods for creating composite 2D-images have been created. Methods for transforming 3D-datasets with wavelets have been implemented, aswell as subsequently reducing/restoring them.

General Imaging : Various filters have been implemented, including preset methods for fixing lightness / contrast, blurring and edge-analysis.

Various : A simple, stand-alone tool for viewing, cropping and resizing datasets have been created.

1.6 Conclusion: Summary

The features in the toolkit in the toolkit are working as intended. The ground work for visualisation mappings and relationships between datasets have been finished. Examples of use have been provided, and results visualised. Notably, using wavelets for decompression of large dataset prior to other analysis work.

Multivariate Image Analysis is viable on regular Workstations.

Chapter 2

Methodology

This chapter outlines the preliminary choices made, aswell as general methology concerning the implementation of the toolkit.

2.1 Scicraft

Scicraft was early chosen as a basic platform for development. Scicraft is a middle-tier program meant for unifying different mathemathical tools into one managable GUI.

> [Exerpt from Scicraft.org]
> SciCraft is an open source data analysis software which solves these problems through an intuitive and user friendly framework where existing methods written in any programming language can easily be combined. It provides integration of a large number of methods from multiple sources such that the user does not need to be concerned with problems related to data imports/exports, file formats and automation. The user can concentrate on the scientific aspects of data analysis without technical distractions.
>
> SciCraft is also an advantage to method developers as their code can rapidly be made available and user friendly without the need for excessive construction of advanced graphical user interfaces. This will significantly increase the turnover rate of new data analytical methods in the scientific community.

From a practical perspective, what it is actually offers is access to several libraries of pre-defined matemathical methods, aswell as a GUI for putting together combinations of methods and decent plotting capabilities.

2.2 Dependencies

Scicraft itself depends on quite a few libraries. These are all available for Linux, Microsoft Windows and *BSD. However installation in Microsoft Windows ends up being quite alot more work than Microsoft Windows users are used to, so bundling everything into an executable has been preferred.

Python ; In order for any python program to be ran, a python interpreter needs to be available

QT ; QT[8] is a cross-platform GUI toolkit created by Trolltech. It is primarly intended for C.

PyQT ; PyQT is bindings which makes it possible to use QT through Python.

PyQWT ; PyQWT is a collection of various visualisations methods. Binaries for this is hard to come by for the various platforms, so it might be needed to manually compile.

Addionally, the toolkit uses the Python Imaging Library (PIL). This library is commonly installed along with Python, and is easily available. The PIL features used in the toolkit are however also supplied by QT, and reprogramming the parts using PIL to use QT in order to remove PIL as a dependency is thus an alternative.

2.3 Programming Langauge

Since Scicraft is cross-platform, this was also a prerequisite for any toolboxes written for it. While Scicraft itself is true cross-platform with Python, any language with cross-platform compiles was deemed acceptable.

After the initial discussion, there were two main choices for programming language:

Python

C++

While recent Virtual Machines for Java are becoming very fast, python was deemed more appropriate due to the fact that Scicraft is python, pythons better readability, and most importantly, former developer experience with Python and its excellent mathemathical library Numeric.

Along the same notes, C and Fortran were not considered due to having few to none advantages over C++. C# and the .NET languages were not considered due to being Microsoft Windows only.

Ultimately, Python was choosen. The reason for this was largely Numeric[5]. It is an low-level array implementation, exposing C internals to Python. With the ease of embedding C-code into Python if really necessary, it was deemed that C++ offered very few substantial advantages over the Python/Numeric combination, and incurred a few disadvantages, most notably the need to compile the code seperatly for each platform.

2.4 Testing Regime

While considered, no formal test regime has been used for this project.

There are a couple of reasons for this; firstly the focus being on prototyping and exploration of possibilities the need for stability and reliability was not that great.

Secondmost, it was very hard to finalize the design. This meant the toolkit was a moving target and as such, the price of keeping a formalized testing regime would have been quite high compared to the benefits. Formalisation is in general a good thing when one knows what one wants, unfortunatly, this did not hold here.

2.5 Tests

All timed tests were run on the same Dell Inspiron 8200 Laptop.

2.5.1 Computer Specification

2.0 GHz Intel P4 Processor
768 MB 133MHZ RAM
ATI RADEON MOBILITY 9000 Graphic Cards
Unknown Motherboard

2.5.2 Operating Systems

Linux

Mandrake 10.2 Testing was used, running KDE 3.2. All options kept at standard from install, except enabling DMA[1]. No services such as webserver or ftpserver were running in the background. All timed tests were run under normal load, specifically with Open Office Writer, Gimp-2.0, an mp3-player and several terminals open.

For the graphic card, the open source 'ATI' driver was used and not the binary drivers provided by ATI.

Microsoft Windows

Microsoft Windows NT 5.1 (specifically Windows XP Home Edition), fully patched and Service Pack 1 installed. Service Pack 2 was not installed due to problems with the special Dell motherboard drivers.

[1] Direct Memory Access enabling Input/Output devices direct access to the system memory thus substancially increasing their speed. Enabled by default in Microsoft Windows operating systems, however disabled by default on most Linux distributions due to increased risk of illegal filesystem states after unclean poweroffs

Comparison

No timed tests were done using Microsoft Windows. However, no speed differences were noticed under normal usage.

It is however of interest to note Microsoft Windows superior performance under high memory load. While Mandrake would slow to a crawl with massive harddisk-activity, eventually forcing the user to end the Scicraft-process manually, Microsoft Windows would gracefully slow down and eventually actually finish. This was not unexpected, as the Microsoft Windows pagefile algorithm is heavily optimized for single-computer usage, while Linux and Unix/BSD both use much simpler and cleaner architectures.

2.6 Test Images: Copyright

The images in Figures 5.5 and 5.10 are copyrighted Kristine Strand Harbek, used here with permission and can be located at http://harbek.deviantart.com.
The image in Figures 5.2 is copyrighted the owners of http://www.freeimages.co.uk/, restrictions apply on usage.
The different illustrative images in this paper are copy-left and thus freely available, and were taken from http://wikipedia.org
The test 3D dataset was provided by the company NEO located in Oslo, Norway.

2.7 Image Formats

As PNG offers the best lossless compression widely supported, all images created have been saved in the PNG format. For images aquired in the lossy JPG format, the original has been kept as JPG and all alterations has been saved as PNG. Image formats that change the image are not acceptable for scientific usage.

Chapter 3

Background

This chapter explains many of the features implemented in the toolkit.

3.1 Multivariate Image Analysis

Multivariate Image Analysis primarely deal with the interpretation and understanding of the results when reducing large datasets into smaller ones through methods such as Principal Component Analysis.

In this paper, Principal Component Analysis(PCA) has also been used on regular 2D-images. As there initially only are three colour values pr position in a regular image, one needs more variables for reduction to be possible. Fortunately, there are plenty to choose from, most obvious adding X and Y as variables, since PCA does not use implicit positional information.

3.1.1 Principal Component Analysis

Principal Component Analysis is a technique that can be used to reduce a dataset to its principal components. More specifically, it is a linear transformation which works by choosing a new-coordinate system in such a way as to minimize the sum of squared distance between the new axis and the datapoints for each axis added.

From a practical perspective, it works by first finding the axis through the dataset with the least squared distance between itself and the datapoints. Subsequently, the original dataset is shifted to remove the positioning explained by this new axis. After this, another axis can be found by searching anew.

1. Given $[x, y, z]$ in $DatasetD$
2. Set $ResidueR = DatasetD$
3. Do until Residue is 0 or wanted number of components is discovered:

(a) Find the $LineL$ going through Origo with the least squared distance between itself and the datapoints in $ResidueR$

(b) The point slope for $LineL$ is the loadings for the new axis.

(c) For all points in $DatasetD$, find closest $PointP$ to $LineL$. The score for this point on this axis is the distance to Origo from this Point P.

(d) Subtract $Loading*Score(x)$ for this axis from all points on $ResidueR$

The 'Score', ie the new values, will for each point and each axis be the value found in each iteration on step 3.3.

The 'Loadings', ie the relationship between the new co-ordinate system and the old, will be the list of vectors found in step 3.2.

The 'Residue', ie the unexplained part of data, will be the rest left in Residue R. The number of variables needed for R to be 0 is known as the rank of the matrix, and will i no cases exceed the number of variables in the original matrix.

There are however many ways of implementing Principal Component Analysis. Scicraft uses the NIPALS algorithm.

3.1.2 Wavelets

The wavelet transform is not one specific transform per se. Rather, it refers to the principle of representing a signal by finite length or fast decaying waveforms. These waveforms are scaled and translated in order to match the input signal. As such, both temporal and frequency information is retained. From a practical perspective, their usage is quite similar to the Fourier Transform. However, unlike wavelets, the Fourier transform has no locality and subsequently the same resolution is used globally for all parts of the dataset.

For the details on wavelets, these, among others, are good readings: 'Wavelets for kids: A tutorial introduction'[11] 'An introduction to Wavelets', [4] 'Wavelets and their applications in computer graphics'[2] 'Wavelets: an Elementary Introduction and Examples'[9].

3.1.3 Wavelet: Compression and Decompression

The paper 'Utilizing three-dimensional wavelet transforms for accelerated evaluation of hyperspectral image cubes'[12] describes using wavelets for compression in order to speed up computation. It was used as the frame of reference for this implementation.

The values in the results from a wavelet transform are directly proportional to how much they contribute to the original image. Utilizing this, it is possible to remove the least significant data from a dataset. While the compression itself is nowhere near real compression algorithms, the resulting dataset is still usable for data-analysis. Reducing a 512x512x512 dataset to f.x. 200x200x200 (a reduction to 40%, which means the original image is mostly retained[12]) means the runtime of a $(O(N^2))$ method is reduced to $1/280$.

Wavelet Compression

Wavelets compression consists of two steps:

1. Convert data using the wavelet-transform
2. Find the least significant slices in the cube of data and simply remove them

In this case, least signicant slices means the slices whose removal alter the origal image the least. As suggested by Vogt et[12] a good arimethic for finding these is simply summing the slices, and selecting the smallest ones' sums. Alternativly, if for example the goal for datacompression was to retain as much detail as possible in one part of the image while blurring, but keeping the rest, a different aritmethic heavily weighting the relevant areas could be used.

The number of slices removed in this toolkit has been determined by a fix-percentage for each axis given to the compression-function. A number of different algorithms, such as discarding all slices with sums lower than Mean - 2 * Standard Deviance could be used (which would ensure only sums very small compared to the rest would be removed).

Wavelet Decompression

Wavelet decompression consists of the exact opposite steps:

1. Add slices of zeros where the slices of data were formerly removed
2. Convert wavelet back using inverse wavelet-transform

In order for step 1 to be actually possible, the positions of the slices removed in compression has to be known.

Compressed Wavelet Storage

In order to save a compressed wavelet, two sets of information needs to be stored;

Decompression information ; Number of slices removed on each axis, and their position

Data ; The dimensions of the cube aswell as the actual cube.

The data needed is basically 3 vectors of slices removed aswell as the cubes, and ideally, it should have been possible to simply save these through Scicraft. Sadly, the only binary format supported by Scicraft at the time of writing is the Matlab-fileformat, and its implementation does not support 3 dimensional data. A non-binary format is not an alternative due to the size of the datasets.

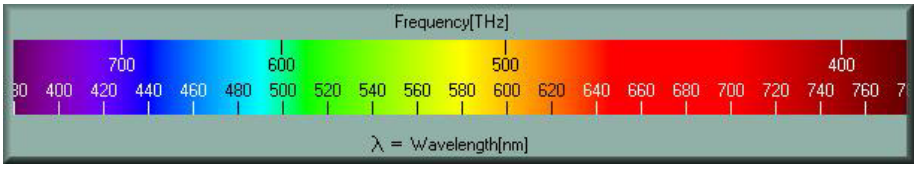

Figure 3.1: Colour Spectrum

3.2 Physics

3.2.1 Light and Vision

Light is electromagnetic radition. It has wavelength, amplitude and frequency. However, due to the speed of light being constant, wavelength and frequency can be interchanged.

What is usually referred to as light in an everyday settings however, is light with frequencies from 400nm to 700nm (Fig 3.1). Moreover, it is not light with one specific wavelength, however it is rather a combination of many different wavelengths.

Light itself has three physical properties:

intensity ; the amplitude of the lightwave, perceived by humans as brightness

wavelength ; (or frequency) the length of one oscillation, perceived by humans as colour

polarisation ; angle of vibration, not visible for humans under normal circumstances [1]

The human retina has three types of colour-receptors, cones. The forth type of photoreceptor, the rod, is mainly concerned with illuminance at low illuminance levels. Because there are exactly three types of colour-receptors each giving off one single signal, three numerical components are needed to adequatly describe colour. Each colour-receptor reacts with different strength to different wavelengths (fig 3.2). Naturally, there are many different light-spreads that will produce the exact same cone-response. For our perception of a colour to be exactly similar from time to time, the surroundings will also need to be similar as colour and lightness are relative measures. [ref BOKA JEG KJØPTE]

3.3 Computer Science

3.3.1 Colour Representations

The toolkit has implemented two colour representations. Firstmost, standard 24-bit RGB is be used. Three values, range 0-255, depict the amount of re-

[1] Unless f.x. wearing polaroid sunglasses

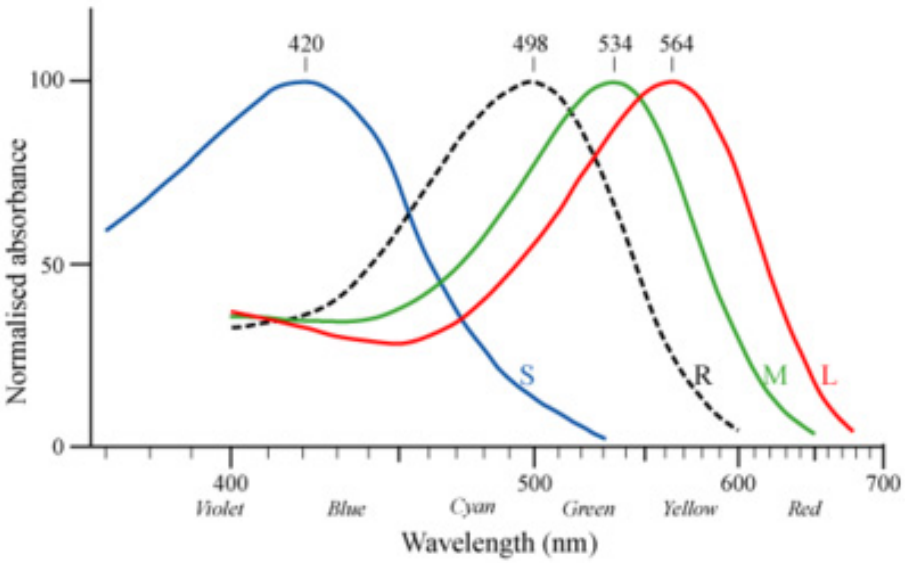

Figure 3.2: Normalized Cone Response

spectivly Red, Green and Blue. Numerical differences in the RGB system does however translate extremely badly to our perceived colour differences[1], and as such, the relationships found when f.x. using Principal Component Analysis does not correspond well to our expectations of similarity.

Secondly, a convertion to HLS has been supplied. HLS colour-space consists of three values, Hue being an angle in the colour circle, Lightness being the amount of light in the image and Saturation being the strength of the colour. While numerical differences in HLS and the related HLB correspond better to our interpretation of colour and lightness there are several problems. Firstly, lightness is computed by (R+G+B)/3, which means it computes Yellow to be about 6 times more 'intense' than Blue when the actual luminance is equal. Secondly, Hue is next to always computed by dividing the colour circle into 60'. This introduces visible discontinuities in the colour space. Lastly, the discontinuity between 360' and 0' makes it impossible for use with regular arithmethics [10].

Chapter 4

Implementation

This chapter documents the implementation of the methods contained in the toolkit.

4.1 Overview

Initially, the toolkit has been divided into three[1] parts:

- General : Containing methods for reading images, and the methods which are meant to work on any dataset.
- 2D : Contains methods which work on matrix or cubes with dimensions $[XxY]$ or $[XxYx3]$.
- 3D : Contains methods which works exclusivly on 3D datasets, such as creating composite 2D-images or the 3D wavelet transform.

The 2D part of the toolkit deals almost exclusivly with colour images, and is generally aimed towards data-massaging prior to Visualisation. The 3D parts of the toolkit deals with Multivariate Images.

4.2 Programming Techniques

Some changes were made to Scicraft to improve performance on large datasets. While any middle-tier program meant for 'glueing together' applications means incurring extra costs, some optimisation was needed.

[1] On some screenshots, one may notice a fourth category wavelets. This was used internally for methods concerning wavelets, as there was alot of trouble finalizing the design on this part of the toolkit

4.2.1 Lazy initalization

With standard install, a node/method in Scicraft initalizes and computes all its output-data regardless of wether or not it is requested. In the cases where a node can output several different sets of data which may be computed seperatly, it would be common sense to only compute the ones actually needed. Initially, Scicraft did not support this. However, since methods in Python are first-class citizens[2], it was possible to overwrite the getData()-method which fetches the data from the container object with the method that actually computes the data. Obviously, in order to avoid calculating the same results several times, the method inserted has to save its own result and return this directly after the first computation. This ensured that the data would not be computed prematurely, and that change could be introduced seamlessly without breaking other functionality. A few changes to eliminate premature and unnecessary getData()-calls was also needed.

4.2.2 Data duplication

Some methods and transforms duplicate data. With large datasets, the overhead in memory usage, and ultimatly swap-memory usage can get excessivly large. The original python-plugin used a pipe to transfer data over to a seperate python-process. Furthermore, The data was truncated using the Pickle module before transfer. This meant that, at worst, there were 5 seperate addional copies of the data existing on the system. With datasets +100 Mb, this naturally had quite an adverse affect on overall system speed.

To solve this, a simple python-plugin utilizing the same python-process was written. While this is against the Scicraft-philosophy of keeping the GUI-process as seperate as possible from the processing, the overhead was simply too high. Several changes was also made to the ImagePlot-class responsible for generating plots of images.

4.2.3 Dynamic Programming

Dynamic Programming is the basic principle of dividing an operation in to many similar smaller operations to save computing. This, combined with simple result-storing for potentially recurring queries have been used where appropriate. For converting between RGB and HLS, the speed gain was substantial.

4.2.4 Fault Tolerance

Besides aiming for a clean documentation, no special techniques for fault tolerance has been used. When a crash occurs, the methods will not result in a

[2]ie Methods/Functions are treated the same as variable names; it is possible to reference and assign them

usuable result anyways, and as such, there is no need for state-restoration or alternate actions on failure. The results already calculated will thus be available, and an as sane as possible error-message will be provided by Scicraft.

4.3 General Imaging

4.3.1 Principal Compenent Analysis

The general PCA routine already found in Scicraft has been used.

Usage

It reduces a XxY matrix into a XxZ matrix, where Z is given as a parameter to the function. See 3.1.1

Implementation

See 3.1.1.

4.3.2 Normalisation

Normal Z-normalisation has been implemented. In this case, the matrixes will be divided by their standard deviation and subsequently mean-centered.

Usage

Any number of matrix may be supplied to the function. The function will however merge these before calcaluting standard deviance and mean. The internal differences between datasets supplied to this function is thus retained. This is advantegous in cases where the data is obviously in the same format to begin with, such as the three values in RGB. In order to seperatly normalize variables this function have to be invoked indepently.

Implementation

Standard Deviation is calculated using the std method contained in MLab module bundled with Numerical.

1. Data = Data / Standard_Deviation(Data) * New_Standard_Deviation
2. Data = Data - Mean(Data) + New_Mean

4.3.3 ImageRead

This function reads an imagefile and converts it into a datamatrix. The node uses the Python Imaging Library which support all regular image formats.

Usage

A filename/path is supplied to the node and this file is read.

Implementation

Data is read using the Python Imaging Library, and subsequently converted to Numeric. However, PIL uses custom list-objects, and in order for the convertion to Numeric to work, the list has to be transformed to regular Python Objects. While this solution is somewhat slow, and not very satisfactory aestithically, it was found to be better than using QT to read the images and copy the pixels out seperatly. PyQT[3] does not support copying the QImage data in one operation, and thus contained just as much unneseccary copying. While writing own image reading methods working directly on Numeric arrays was a possibility, this was decided against due to the cost being incurred was only one-time and scaled fairly well.

4.3.4 FileRead

This function reads a .HDR file containing image information. It is included because the data we had available were in that format. This node was initially meant to read alot of different formats, but this was not implented due to it never being actually needed.

Useage

A file name to a .HDR file must be supplied. An .IMG file containing the actual data must be in the same catalog with the same name.

4.4 2D-Imaging

The toolkit has two methods for visualizing score plots from PCA tranforms. The Density-Plot transform can be used on dataset reduced to two variables. It works by simply viewing the two variables as co-ordinates in a new plot and retaining the original colour. On the other hand, the regular colour-plot works exactly the opposite, it retains positional information, and uses the new variables as colours directly.

The latter way is the common way of visualising score plots. The variables as positional information approach does have the advantes of very easily visualising the relationships between the pixels and discerning the defining features of an image (as far as PCA goes that is). Combined with a mapping between the original image, allowing one easily to find the original positions of a group of pixels this seems to be a very usefull tool.

[3]The Python bindings for QT

4.4.1 Density-Plot transform

The Density-Plot works by simply viewing the supplied variables as co-ordinates, and places them with their original colour into the new image.

Usage

The data is normalized, and projected into a new image. Image colour is given by the original colour of the pixel, while intensity is given by the number of pixels placed there. A parameter is used to determine the background colour of pixels which aren't mapped to. Tables depicting which pixels were placed where were created to utilize the Dynamic Mappings implemented. Some magic numbers[4] have been used. For positions with several pixels placed in them, the average colour is used. Positions with many pixels placed into them will be brightened up, while positions with few pixel placed will be darkened.

Implementation

1. Let $ListL1$ and $ListL2$
2. For each pair of co-ordinates in original image (X, Y) do:
   (a) Find values (X', Y') in new image on position (X, Y)
   (b) Find co-ordinates on new Image (X', Y') and find placement (X'', Y'') using linear interpolation with new Image size and max/min values of (X', Y').
   (c) Add entries for original pixel (X, Y) and new pixel (X'', Y'') in tables
   (d) # creating lookuptables for the relationship between the pixels
   (e) lookuptable_from_ori[original_x, original_y] = (x,y)
   (f) lookuptable_from_new[x,y] += (original_x, original_y)
3. For each pixel in new image do:
   (a) Set colour to the colour original pixel had. Where several pixels was placed, use the average.
   (b) Scale colour by multiplying with 0.6 + 0.8 * number of pixels here / max number of pixels at any pixel.

4.4.2 Dynamic Mapping

The imageviewing capabilities of Scicraft has been improved to enable dynamic mappings between plots. Provided the 'lookupdata' property is set, if an area is selected in the image, a window will pop up showing the corresponding data in the linked-to image.

[4] Magic numbers are numbers chosen 'just because they work' and are thrown directly into the code without explanation or justification

Usage

Use a method which sets the lookupdata property, plot the data in a '2D-Plot: Image'. To see the source of a portion of the data, simply select it with the mouse.

Implentation

1. For each pixel in 'list of pixels selected in image':
   (a) Use 'lookupdata' to find out which pixels was placed here
   (b) Plot the pixels found, with position and colour taken from the target image

4.4.3 Autoscaling

A simple method which ensures that all values are between 0 and 255. Lowest value is set to 0, and the spectrum is linearly stretched so maximum is 255.

This is the most common way of visualising score plots. ie simply transforming data down to three variables, scaling these and using these directly as colour values.

Implementation

1. Input = Input - min(array)
2. Onput = Input * 255 / (Max(array)-Min(array))

4.4.4 Fold / Unfold

Principal Component Analysis can not be done on cubes. However, by collapsing/folding the cube into a matrix PCA can be used, and the result can be expanded/unfolded back into a cube. Since PCA doesn't concern itself with the order of pixels, any information from pixel placement was lost anyways and collapsing the matrix will thus not make any difference.

The Fold / Unfold functions provided in the toolkit folds and unfolds a cube.

Usage

The Fold function takes a cube, and outputs the converted matrix and the former dimensions of the cube. The Unfold function takes a matrix and a set of dimensions and converts them into a cube.

Implementation

data.shape = new_shape

4.5 3D-Imaging

4.5.1 Wavelets

See 3.1.3 and 'Utilizing three-dimensional wavelet transforms for accelerated evaluation of hyperspectral image cubes'[12]. The steps explained in these can be summaried to:

1. Preprocessing: Add zeros to the dataset until it is of the format $XxXxX$ where X is the power of two.
2. Wavelet: Use the wavelet transform
3. Removal: Remove the least significant slices of data in all three directions (ie, the slices whose removal will affect the inverse-transform the least)
4. Processing: One now has a reduced orthogonal dataset which one may transform in any way one wishes
5. Adding: Insert the slices removed earlier, only now with zeros.
6. Inverse Wavelet: Use the inverse wavelet transform
7. Postprocessing: Crop the dataset down until its original size

4.5.2 Wavelet transform

Using the wavelet transform for cubes supplied originally belong to the freely avaiable wavelab package, Copyright (c) 1993. David L. Donoho. It was modified for use on 3D Matrix by Vicki Yang and Brani Vidakovic, ISyE, GaTech 2002.

All results were done using Van Den Haar basic transform. However other transforms available were also tried. For comparisons on different basic transforms for compression please refer to 'Utilizing three-dimensional wavelet transforms for accelerated evaluation of hyperspectral image cubes'[12]

Usage

Called with a cubic dataset, meaning $XxXxX$, most likely $128x128x128$ or $256x256x256$.

4.5.3 Shrinking Wavelets

Using the algorithm discussed in 3.1.3 and [12] for selecting and removing slices. The inverse of this functions expands the dataset back by adding 0's where data was removed.

Chapter 5

Results: 2D

This chapter contains examples of usage of the methods implemented in the '2D' package. Please refer to the Implentation chapter for details on the workings of methods described and the Background chapter for theory on regular Multivariate Image analysis and Principal Compenent Analysis.

5.1 Example 1

This simple example is meant to show how the methods are used, and how they can be combined. Figure 5.1 shows what is called a ModuleDiagram in Scicraft. Each box, called a node, showed there represents a method. Most the methods used in this toolkit have been implemented in Python, however some are only avaiable through Octave[1]. The methods can be arranged in any, and by double-clicking on the nodes you get an option box where you can set optional parameters. The 'ImageRead' node for example, allows one to specify filename to be read.

5.1.1 Setup

The workflow of the Module Diagram shown in figure 5.1 is as following;

1. ImageRead reads the imagefile specified by a preset parameter, resulting in a 400x600x3 matrix representation of the image.
2. CreateVariables transforms supplies several transformed images, however these are not actually computed until requested. These are of the format 400x600.
3. MergeVariables request variables from CreateVariables, in this case, it was set to request Hue, Lightness and Saturation. These are re-combined into a 400x600x3 cube.

[1] Open Source clone of Matlab

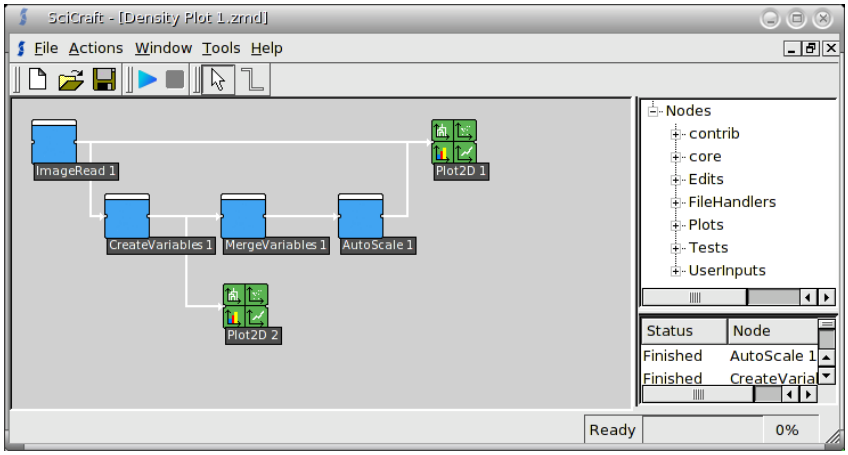

Figure 5.1: Module Diagram Example 1 in Scicraft

4. AutoScale linearly scales the data it receives from MergeVariables, ensuring that all variables in the 400x600x3 are between 0-255. This is a prerequisite for using the ImagePlot.

5. Plot 2D 1, when opened, shows the original Image, aswell as the merged cube containing Hue, Lightness, Saturation instead of R, G, B.

6. Plot 2D 2, when opened, shows the 9 different transforms created from CreateVariables. The six transforms not yet computed are computed at this time.

5.1.2 Original Image and Merged Image

Figure 5.2 contains the resulting images after running the example. The first picture is simply the original image. In the second picture, Hue, Lightness and Saturation was used directly as the RGB Colour components. While the resulting image from this transform is not especially interesting, it shows how transforms work and how any $XxYx3$ matrix can be used as an image.

5.1.3 Plots from CreateVariables

Figure 5.3 contains the resulting images from the CreateVariables method. While this method was created for building variables for later reduction through Principal Component Analysis(PCA), the components are useable in other settings.

The different images in Figure 5.3 are respectivly:

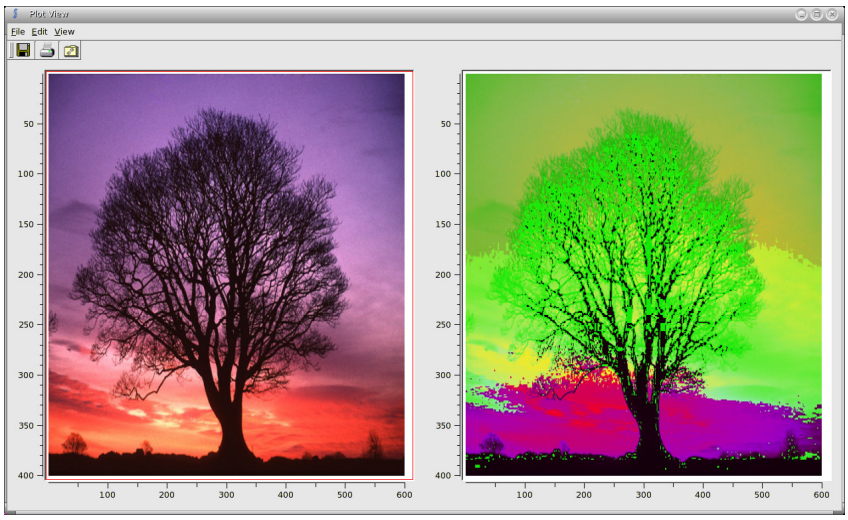

Figure 5.2: Hue, Lightness and Saturation used directly as the colour components

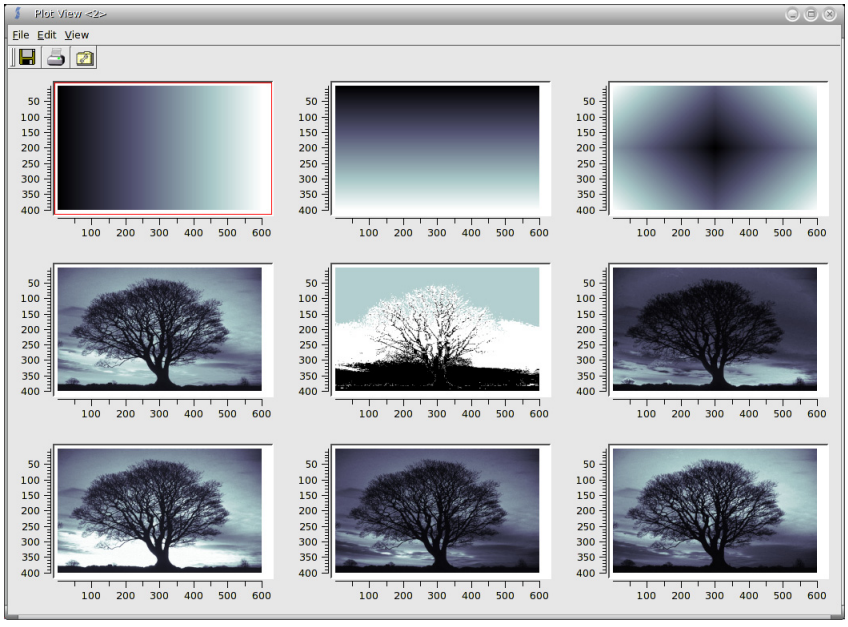

Figure 5.3: Outputs from CreateVariables

X-Position	Y-Position	Distance to Sentrum
Lightness	Hue	Saturation
Red	Green	Band

The top three elements are naturally only dependant image size. Their use is for re-introducing position information into PCA (see 3.1.1). The middle variables stem from a conversion to HLS Colour space(see 3.3.1). Lightness is amount of background lighting in the colour, hue is an angle in the colour circle, while saturation is the purity, or strength, of the colour. The last three are merely the regular RGB values seperated into slices.

5.1.4 Speed

These results were made using Mandrake 10.2 Testing on a Dell Inspiron 8200. Please refer to 2.5.2 for complete System specification.

Name	Speed(S)
ImageRead	4.8506500721
CreateVariables	0.0136461257935
Plot2DNode	0.000693798065186
MergeVariables	6.68285107613

5.2 Example: PCA on 2D-Image

Two main methods for visualiszing PCA-Score values have been implemented. The most common way is to reduce a dataset down to three variables which are used directly as colour components. This means keeping pixel positions the same as in the original image. Alternativly, one may do the exact opposite, retain original pixel colours, and use the new variables as co-ordinates. This example is an example of the former.

Module Diagram

This example is fairly similar to the previous one. However, after merging the variables into a cube, the cube is folded into a matrix, reduced using Principal Component Analysis and subsequently unfolded back into a cube. The two red nodes are merely nodes which show their inputs in a popup box on-screen.

1. 'ImageRead' reads the image specificed: 420x300x3
2. 'CreateVariables' supplies variables to 'MergeVariables', which requests and merges X, Y, Distance to Sentrum, Hue, Lightness and Saturation. (420x300x3 to 6 * 420x300 to 420x300x6)
3. 'Fold' transforms the cube 420x300x6 into a matrix 1260 x 6 .
4. 'PCA' reduces the 1260x6 matrix to a 1260x3 matrix.
5. 'Unfold' unfolds the 1260x3 matrix back into a 420x300x3 cube

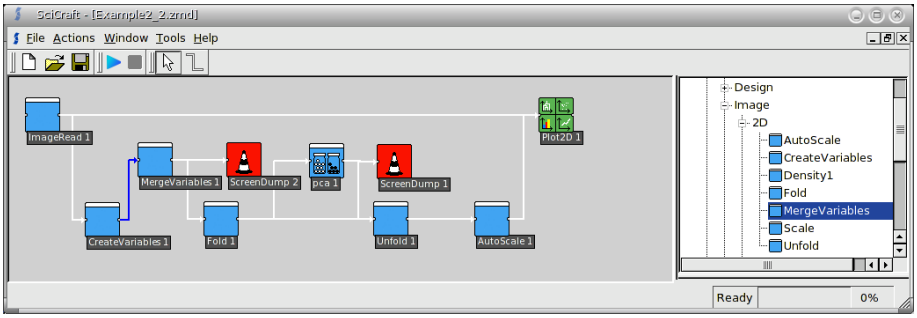

Figure 5.4: Module Diagram Example 2

6. 'Autoscale' linearly scales the values in the cube to 0-255.

7. 'Plot2D 1' can be opened to show the originalimage and the new cube.

Scores

Variabel	Used as	Dist.	Y	X	Satur.	Light	Hue
1	Red	0.040	0.973	-0.007	-0.005	-0.013	0.227
2	Green	-0.626	0.181	0.322	0.043	0.235	-0.643
3	Blue	0.250	-0.046	0.944	-0.039	-0.107	0.175

As explained in 3.1.1 the scores are the new variables weightings of the old variables.

Looking at Red, the first value computed, one can conclude there is a relationship between Y-Position and Hue. As the darker variants of red in the upper parts of the picture carries the same hue as the lighter parts further down, this certainly seems correct.

Looking at Green, the second most determining feature of the image was the relationship between distance to centrum, hue (ie; lack of red) and lightness. Lastly, the third variable created shows a relationship between the right (and to some degree bottom) part of the picture with hue and darkness.

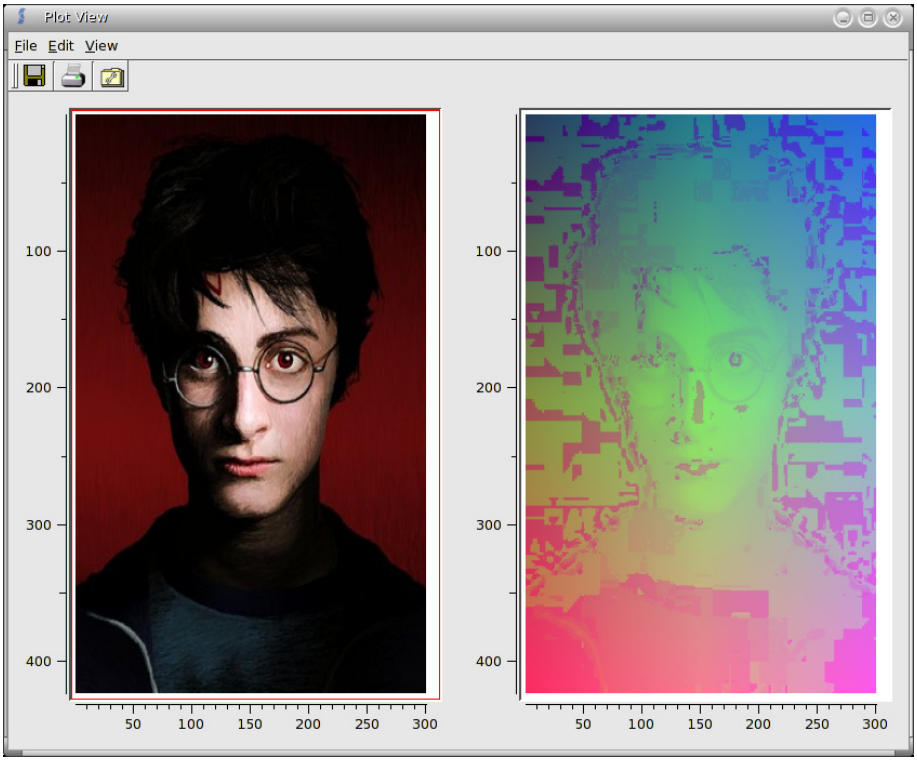

Figure 5.5: Resulting Images Example 2

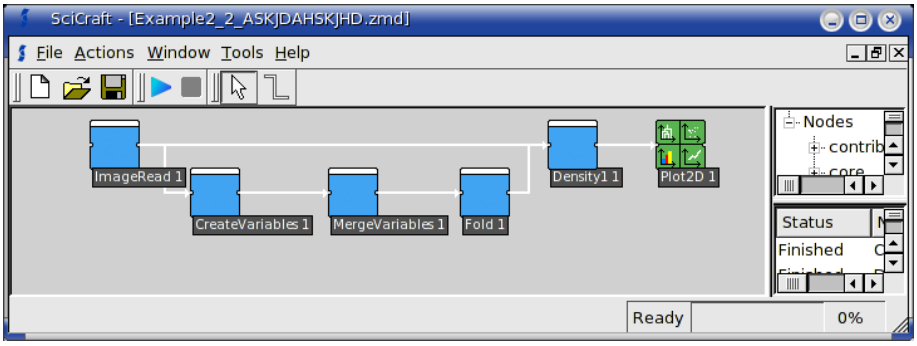

Figure 5.6: Module Diagram Example 3

Speed

Name	Speed(S)
AutoScale	0.107710838318
Plot2DNode	0.000708103179932
ImageRead	2.48364019394
CreateVariables	0.0123908519745
MergeVariables	4.14734101295
Fold	0.0204799175262
ScreenDumpNode	0.002925157547
pca	67.8035509586
Unfold	0.0199151039124
ScreenDumpNode	0.015310049057

5.3 Example: Density Plot

The first example in this chapter used the new values supplied as colours while using the original image for pixelpositions. The Density Plot simply does the opposite; it accepts a list of paired numbers which it uses for co-ordinates while using the pixels original colour when plotting.

Since not all pixels in the new image may have pixels set, background colour may be chosen as a parameter. For the new positions where several pixels are set the average colour will be used (taken linearly using the seperate RGB-components). In order to visualize pixels placements, positions where several pixels have been placed will be brightened up, and positions with few darkened. Please refer to [implementation] for specific details.

5.3.1 Module Diagram

1. 'ImageRead' reads image.

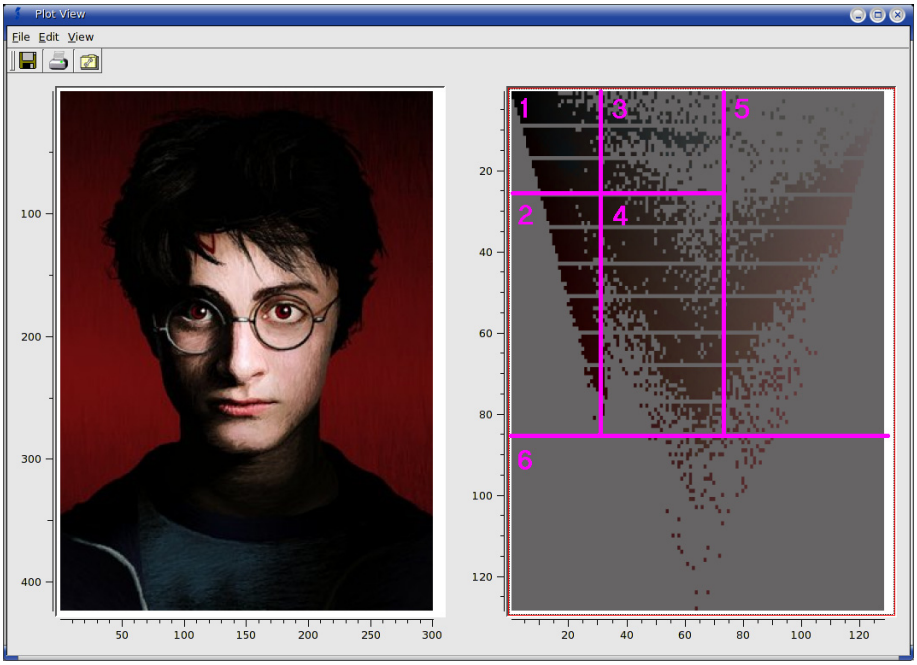

Figure 5.7: Resulting Images Example 3. Pink lines and numbers are inserted. The different areas were selected and their origin plotted in Figure 5.8

2. 'CreateVariables' supplies Lightness and Saturation to 'MergeVariables' which are merged into a cube.
3. 'Fold' folds the cube into a long list of numberpairs
4. 'Density 1' creates a Density Plot of the Image, and sets the appropriate 'Lookupdata' properties in order to preserve the relationship between the images.
5. Plot 2D shows the Image. By selecting parts of the image, a pop-up will appear showing which part of the other image the selected part corresponds to.

Resulting Images

The resulting images can be found at figure 5.7. By using the Dynamic Mapping implemented, the images corresponding to the six areas in the second picture on 5.7 can be found on 5.8

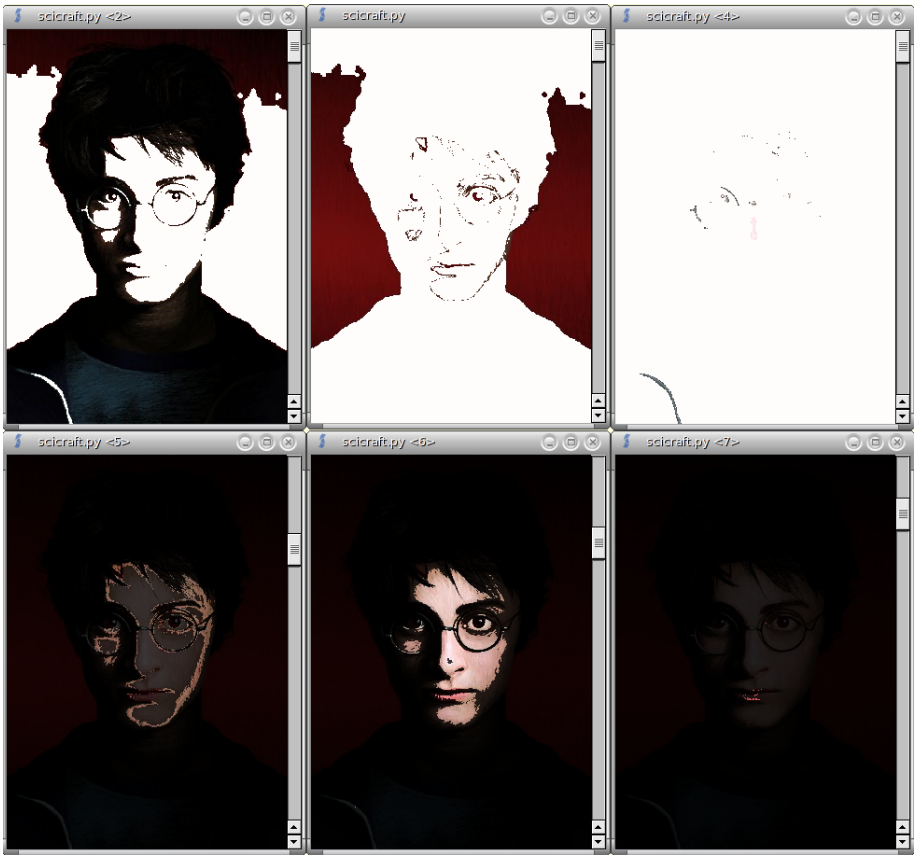

Figure 5.8: The different areas selected in image 2 on Figure 5.7

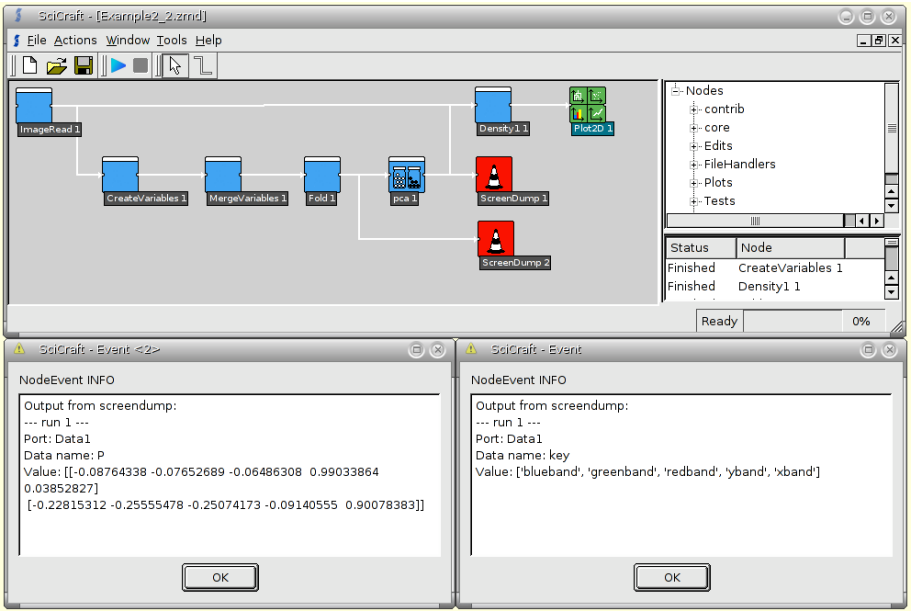

Figure 5.9: Module Diagram: No normalisation example

Speed

Name	Speed(S)
ImageRead	7.43504214287
CreateVariables	0.0262589454651
MergeVariables	2.23696804047
Fold	0.00427508354187
ScreenDumpNode	0.00141596794128
pca	107.233451843
ScreenDumpNode	0.0392458438873
Density1	25.0737512112
Plot2DNode	0.000715017318726

5.4 Examples: Normalisation

5.4.1 Example without Normalisation

Fig 5.9 shows the Module Diagram used and the popups from the two red nodes showing the data they received. What happens is:

1. ImageRead reads image and CreateVariables extract the R,G and B colour values aswell as the X and Y positional information for each pixel.

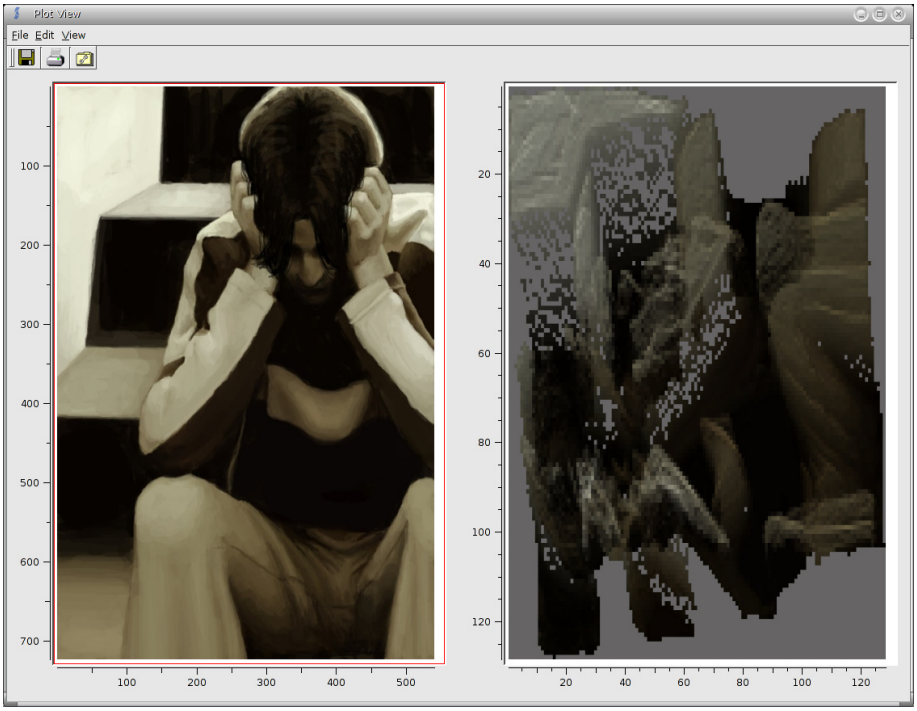

Figure 5.10: Example without normalisation: Result 1

2. MergeVariable builds a matrix where each item consists of R,G,B,X and Y for each pixel. Fold folds this into a list of R,G,B,X and Y values.

3. Principal Component Analysis reduces the dataset down to two new variables. Their scores are shown in the left box on fig 5.9 and the name of the variables are on the right box.

4. Density Plot uses the new values as co-ordinates and builds a new image using these new co-ordinates and their original colour.

5.4.2 Results

Figure 5.10 shows the two resulting images. The scores for the variables are (as shown in fig 5.9:

Variabel	Used as	Blue.	Green	Red	Y	X
1	X	-0.087	-0.077	-0.065	0.990	0.038
2	Y	-0.228	-0.256	-0.251	-0.091	0.901

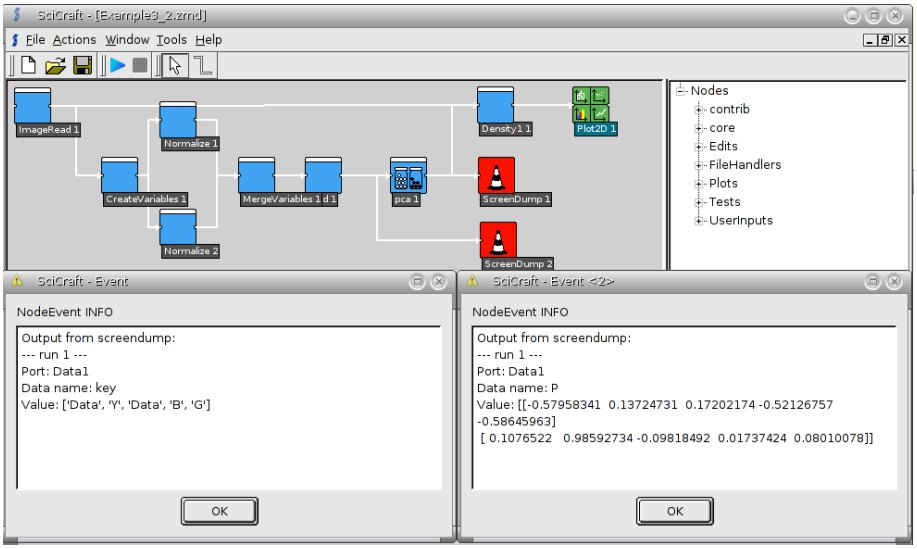

Figure 5.11: Module Diagram: No normalisation example

As one can see from both the resulting image and the score plot the dataset is really dominated by the X and Y components. Part of the reason for this is the maximum and minimum values were simply larger than the colour values. Naturally, this is like comparing apples to pears, as there is no way to tell how two absolute values with different connotations should be compared. The common answer to this problem is of course normalisation, ensuring that all variables are treated somewhat equally.

5.4.3 Example with Normalisation

Fig 5.11 shows the modified ModuleDiagram. In this case, the R,G and B values are normalised together, meaning standard deviance and mean is computed for them together. The reason for this is that the relative values really do contain information between the colour bands. Very low R and generally high G and B values simply mean there is little red in the picture and the values should reflect this. The X and Y values are also normalised together.

By normalised, it is here meant z-normalisation, ie setting standard deviance and the mean is set to a specific amount.

5.4.4 Results

Figure 5.12 shows the two resulting images. The scores for the variables are (as shown in fig 5.11:

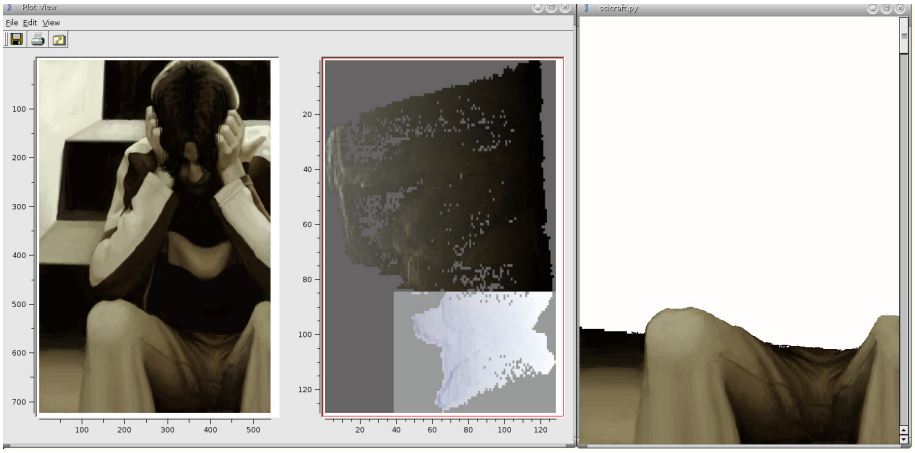

Figure 5.12: Example with normalisation: the rightmost picture is what the selected area in the middle image correspond to

Variabel	Used as	X	Y	Red	Blue	Green
1	X	-0.579	0.137	0.172	-0.521	-0.586
2	Y	0.107	0.986	-0.098	0.017	0.070

As one can see from the score plot, the first variable is just as much dependant on Blue and Green as X now. A possible interpretation is as one goes left, there is a distinct lack of green and blue. However, looking at the source image this does not seem to make much sense since the image is in Black and White. To some extent this example highlights the problems with normalisation: when there is very little differences the small, somewhat random, differences that are will be magnified.

In essence, the non-normalised image made more sense in this case. This was not the case with other images.

Chapter 6

Results; 3D

This chapter contains examples using storing, loading datasets converted by Wavelets. It also contains examples of using PCA on said wavelets and summaries of time and storage-savings.

6.1 Example Data

The image used as example in this chapter was provided by NEO. It is a $760x362x160$ satelite image, with 160 pictures taken from wavelengths 410NM to 997NM. As this is a satelite image, there is sadly no original image ot use a frame of reference. The data was provided in a XXX large file, with each value being a short. Furthermore, a text-file containing meta-information and datastructure details was provided.

6.2 Composite Images

When dealing with actual lightdata, it is a natural for the goal of any visualisation to be as realistic as possible. Since we use our own vision as the frame of refence for similarity, this goal can be paraphrased into making sure the light produced by the output device (be it a printer, computer-screen or a TV) results in the same cone-responce as the original light. While a perfect result is impossible, as colour-reproduction on computer screens actually do vary quite alot depending on both colour-settings, manufacturer and type of device, they are atleast fairly similar. As the different cone-responses have been quite accuratly measured a good, although not perfect, approach is to calculate the cone-responses generated from the light-spread in question, and choose the RGB values which (provided RGB was translated into pure colours) would generate the exact same cone-response.

The approach of using seperate slices for the colour components is simple and fast, and can produce good results most of the time. Packages like TNT[7] automatically selects the most noise-free bands. However, this does mean throwing

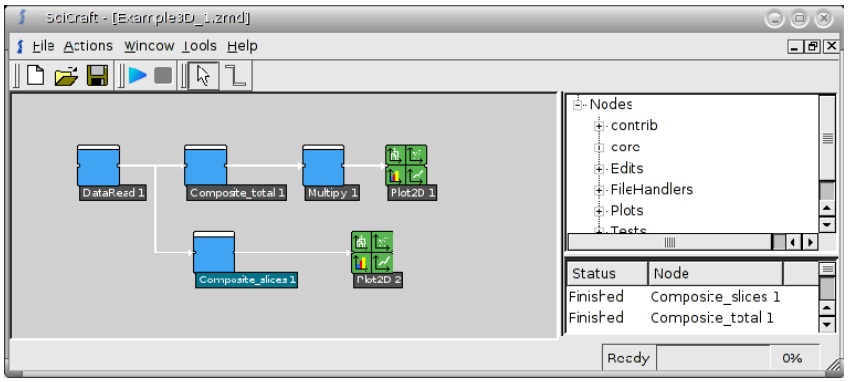

Figure 6.1: Example set up in Scicraft

away alot of colour-information that our eyes are capable of interpreting, and as such you will always loose details and shadings. The amount lost will differ depending on the materials and the type of picture though.

6.2.1 Example

This example reads the IMG file set by 'DataRead 1', and the two different methods for creating composite images are used. The image from composite_total transform is also brightened up by multiplying the matrix by 1.2.

1. DataRead reads the image file.
2. Composite_total and Composite_slices simultaniously transform the original dataset into images[1]
3. Multiply takes the output from Composite_totals and multiply it by 1.2.

6.2.2 Comparison

Quality

Upon close inspections, there seem to be some differences in the level of detail on the images. The little shack right of the lowermost house is perceivably more fine-grained on the transform using the whole picture. The roads and some features on the grass also seem to be different.

[1] X x Y x 3

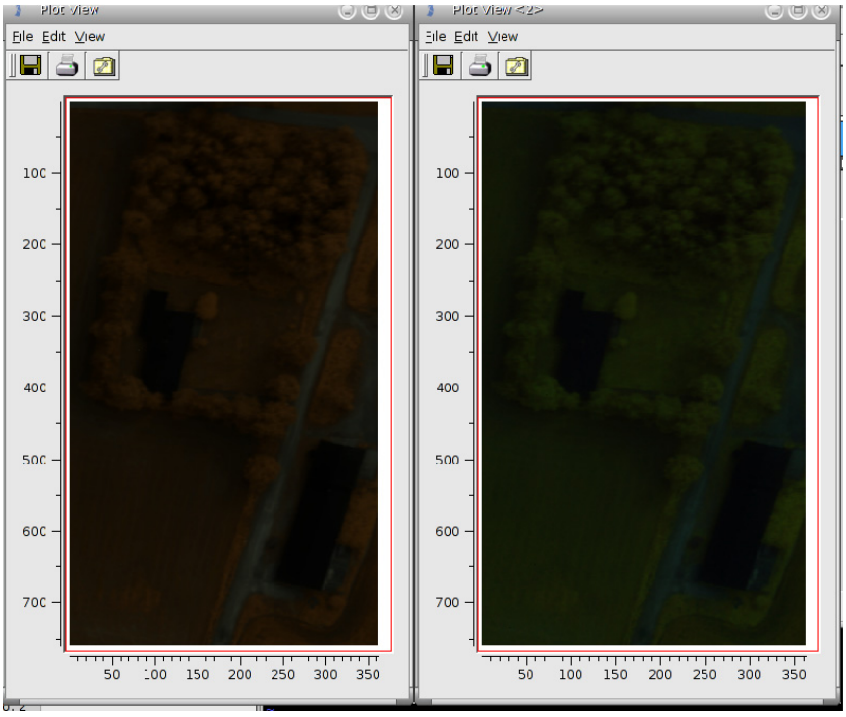

Figure 6.2: Resulting Images

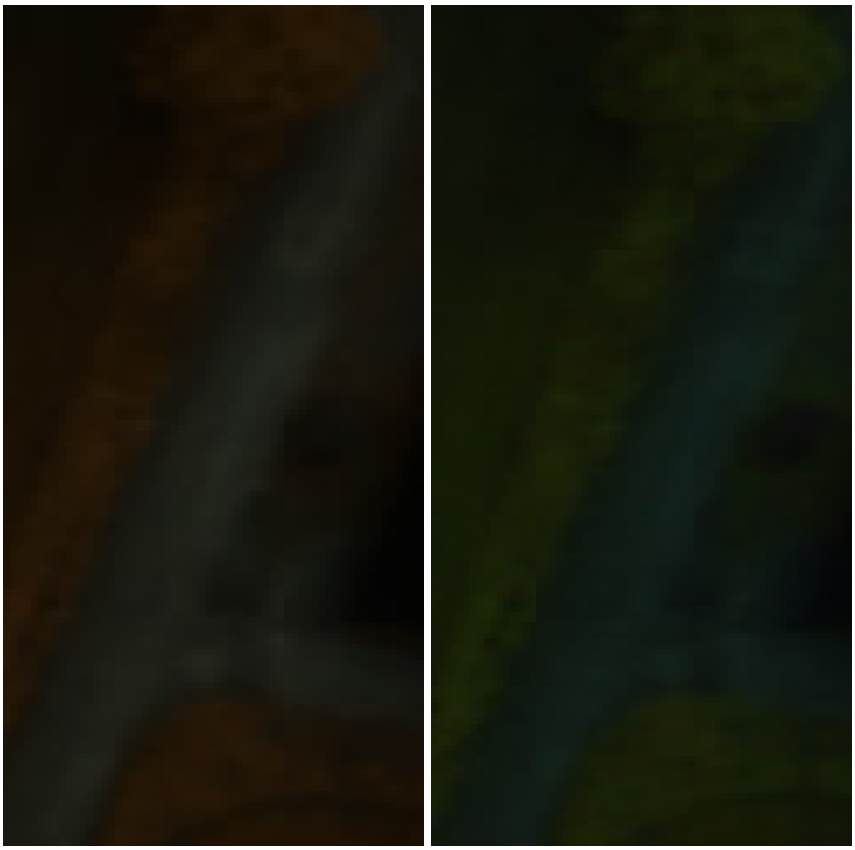

Figure 6.3: Blown up images of Fig 6.2

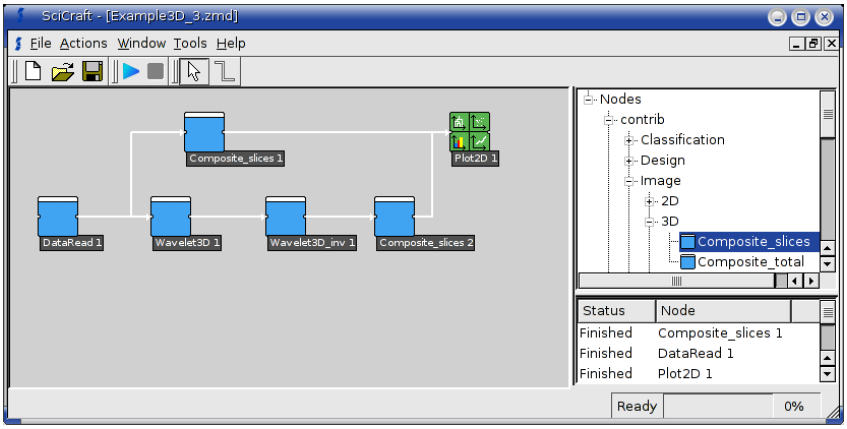

Figure 6.4: Example set up in Scicraft

Colouring

The colouring on the image from the transform seems to be somewhat off. The colouring will however only be as good the transfer-function is, and the one used here was not tuned and quite crude to begin with. The image created from the slices does seem to be overly green.

6.3 Wavelets

Please refer to Background 3.1.3 for explanations.

6.4 Wavelet Examples

This section contains several different examples using wavelets.

6.4.1 Transform and Inverse Transform

This examples shows the forward and the inverse wavelet transform used on a dataset.

The ModuleDiagram can be seen in fig 6.4

1. XXX Reads data
2. 3D Wavelet transforms data
3. 3D Wavelet Inv transforms data back

Result

[include graphic]

As one can see, although the wavelet transform is lossy, there is no perceivable difference between the image.

Speed

Name	Speed(S)
Wavelet3D	210.61946702
shrink_wavelet	2.47157406807
shrink_wavelet_inv	0.191494941711
wavelet3D_inv	212.955876112

As one can see, the Wavelet transform run in octave on a 128x128x128 matrix takes time. Comparably, as long as one chooses sound algorithmal approaches [2] , the tim e spent on the rest will be inconsequental.

6.4.2 Wavelet Compression

Please refer to 3.1.3.

This example compresses and decompresses the dataset to different sizes and compares the results with original image.

Fig 6.5 shows the ModuleDiagram used.

1. DataRead 1 reads data.
2. Wavelet3D 1 carries out the wavelet transform
3. Shrink_Wavelet 1 reduces the size of the cube by a percentage set in a parameter
4. Shrink_Wavelet_inv 1 increases returns the cube to the original size by inserting 0's.
5. Wavelet3D_inv carries out the inverse wavelet transform.

Results

Figure	Name	Dimensions	Size
	Original	$128x128x128$	4MB
6.6	75%	$96x96x96$	1.69MB
6.7	50%	$64x64x64$	0.5MB
6.8	25%	$32x32x32$	0.06MB

[2]As an example the shrink_wavelet method used in the next example took 185 seconds with regular double For statements and pixel for pixel operations, which was slashed to under a second with regular Numeric arithmethic

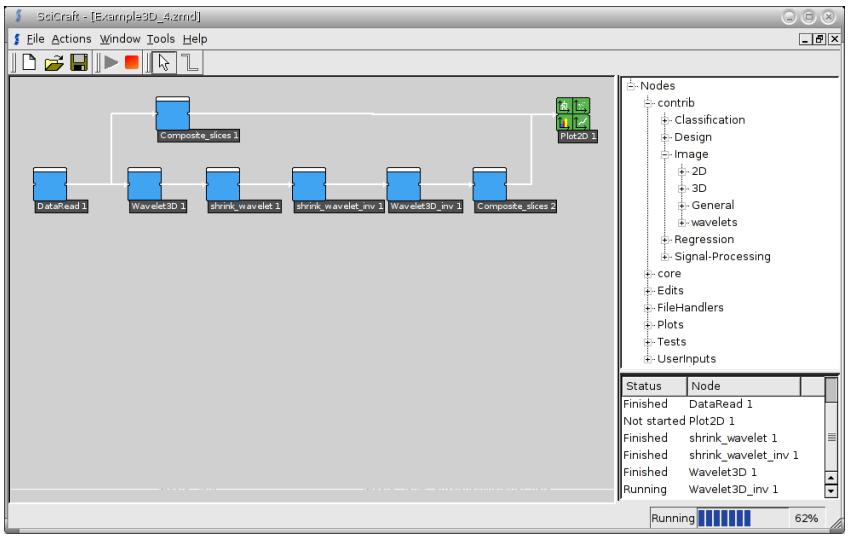

Figure 6.5: Example set up in Scicraft

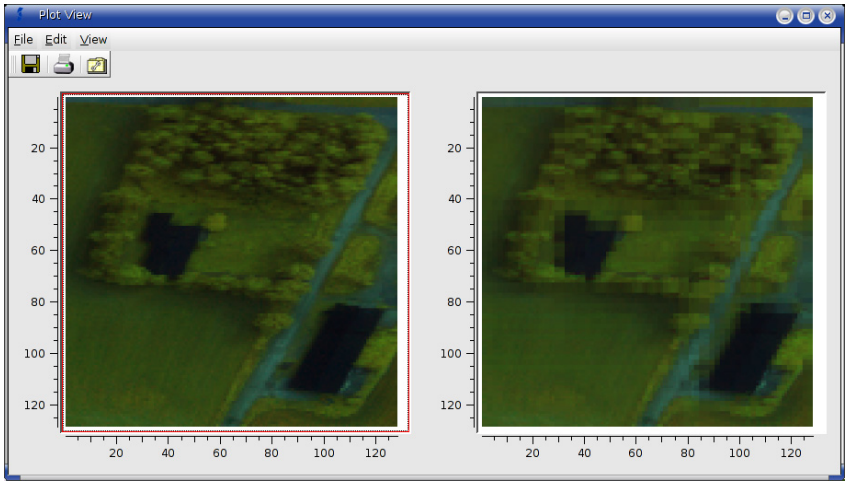

Figure 6.6: Result of compression to 75% pr axis

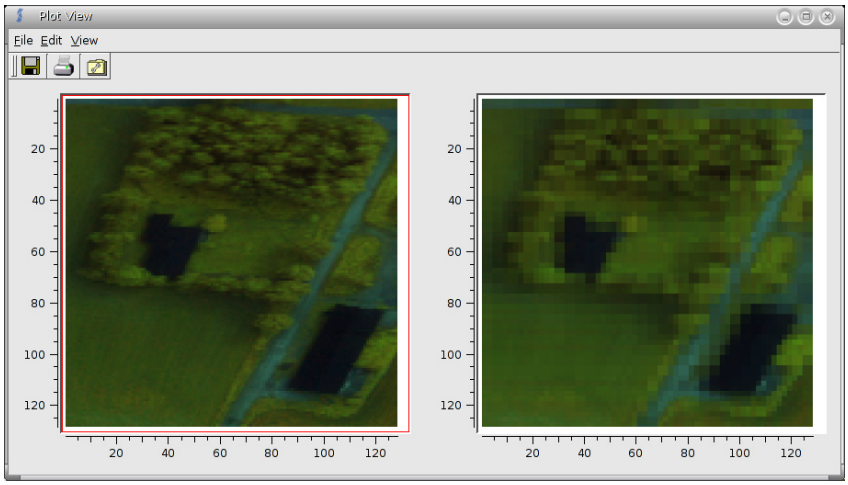

Figure 6.7: Result of compression to 50% pr axis

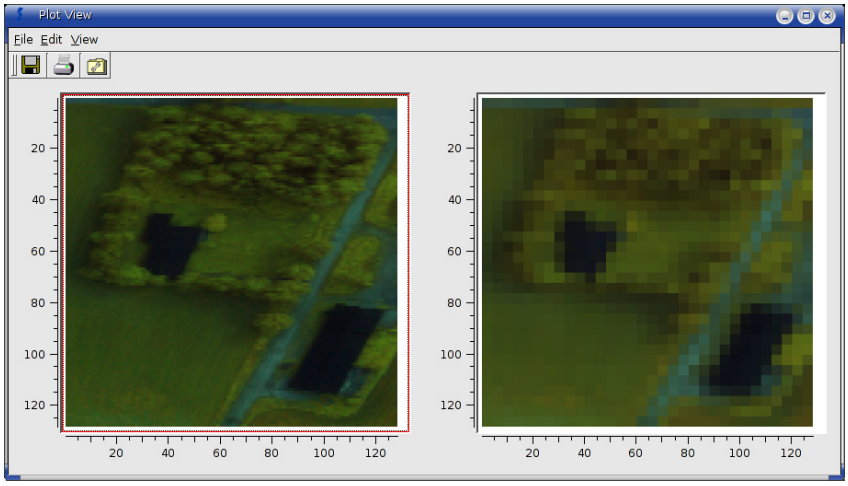

Figure 6.8: Result of compression to 25% pr axis

Chapter 7

Conclusion

7.1 Summary

The features in the toolkit in the toolkit are working as intended. The ground work for visualisation mappings and relationships between datasets have been finished. Examples of use have been provided, and results visualised. Notably, using wavelets for decompression of large dataset prior to other analysis work.

Multivariate Image Analysis is viable on regular Workstations.

7.2 Evaluation

The majority of features simply work. Contrary to what was believed at the start, all methods were implemented cleanly, and there are no special preconditions or 'hacks' needed for the methods to work.

- Using Wavelets for Compression/Decompression prior to other dataanalysis work.
- Using PCA and Folding/Unfolding of the datasets needed work.
- Visualising data in a variety of forms work.
- Some Image enhancement methods have been implemented.

Speed is, somewhat surprisingly (and actually contrary to what the first draft of this document said [1]) not an issue; all 'For sentences' iterating pixel

[1] At the time, some pure Python code was present which has now been replaced with Numeric code

by pixel in Python have been replaced by fairly efficient Numeric operations. While it most certainly is possible to speed up the Python code, the bottleneck is the Principal Component Analysis and Wavelet methods written in Octave. Rewriting these in a faster language, aswell as using a faster algorithm for Principal Component Analysis will help alot more than any code optimization could ever do on the rest of the code.

Dynamic Mappings are working as intended, and with very few changes to the Scicraft base. Lazy initialisation for some methods have introduced notable speed increasements. In this regard, Python really showed its flexibility. The solutions created were simple, clean and efficient. While further interactivity and live-transformations would be preferable, it is not currently possible with Scicraft.

Some features work, like the Filters are inadequatly implemented. They are not as flexible and fault-tolerant as they should be. As such, the toolkit is not by any means a mature product. Addionally, some methods like the Density Plot use Magic Numbers (numbers which cannot be changed easily and were chosen without any other justification than 'they work'). These should either by justified or moved out as parameters for the functions.

While the toolkit is stable for anyone familiar with its workings, it is doubtfull it will hold long against a novice user. Good error messages are mainly lacking.

Lastly, good datastructures for Meta-information were not created (or rather, they were not able to be integrated cleanly with Scicraft). The lack of f.x. wavelength information on the images passed around is notable.

While it was a goal for the toolkit to work stand-alone aswell as with Scicraft, this was not accomplished.

7.3 Discussion / Further Work

Initially, the 2D part of the toolkit was only meant for visualising and data-massaging pre-visualisation. However, as smaller datasets were wanted for testing, regular images was expanded with various extra data. The author of this paper has not seen this done anywhere else [2]. Under testing, interesting results were found on some images, however the results seemed a bit random. While it may not appear usefull at first sight, there may be possibilities of creating more usefull and specialised transforms by further experimentation.

While still offering the possibility of visualising score plots one and one by using a colour scale for the values, two new possibilities have been offered. However using three different score plots directly as the three colour componenets seemed a bit confusing for the eye. On the other hand, the Density Plot transform really does seem usefull, and is also something the author has never seen in Multivariate Image Analysis before.

[2] For that matter, the author has neither ever seen anyone jump of a building

Bibliography

[1] LeRoy E. DeMarsh and Edward J. Giorgianni. Color science for imaging systems. *Physics Today*, September 1989, 44-52.

[2] Alain Fournier. Wavelets and their applications in computer graphics. 1994.

[3] Paul Geladi and Hans Grahn. *Multivariate Image Analysis*. John Wiley & Sons, 1997.

[4] A. Graps. An introduction to wavelets. *IEEE Computational Science & Engineering*, Summer, 1995.

[5] http://sourceforge.net/projects/numpy.

[6] http://www.censsis.neu.edu/software/hyperspectral/hyperspectral.html.

[7] http://www.microimages.com/getstart/hypanly.htm.

[8] http://www.trolltech.no.

[9] Ueda Masami and Suresh Lodha. WAVELETS: AN ELEMENTARY INTRODUCTION AND EXAMPLES. Technical Report UCSC-CRL-94-47, University of California, Santa Cruz, Jack Baskin School of Engineering, January 1995.

[10] Charles A. Poynton. Frequently asked questions about color. 1997-03-02a.

[11] Brani Vidaković and Peter Müller. Wavelets for kids: A tutorial introduction. 1994.

[12] Frank Vogt, Soame Banerji, and Karl Booksh. Utilizing three-dimensional wavelet transforms for accelerated evaluation of hyperspectral image cubes. *JOURNAL OF CHEMOMETRICS*, J. Chemometrics 2004; 18: 350-362.

[13] Barry M. Wise and Paul Geladi. Multivariate image analysis. *Wiley, Wilchester*, 1996.