OSMIA

Deliverable D3.5

Tina Evaluation Report

Prepared by Ian Poole for Voxar Ltd.

September 2003

Voxar Ltd
Bonnington Bond
2 Anderson Place
Edinburgh EH6 5NP
T : +44 131 472 4792
F : +44 131 472 4799

email: ipoole@voxar.com
http://www.voxar.com

1 Introduction

Having dealt with the technical difficulties of building the Tina libraries in a Windows environment, and interfacing to them from our codebase [1], Voxar turned its attention to evaluating the usefulness of the library through "in anger" use.

Our aim throughout has been to use the OSMIA project as a means to add new segmentation functionality to our flagship product *Voxar3D* (formerly known as *Plug'n'View*). In particular, we seek to demonstrate this new functionality to prototype standard, at RSNA 2003, which takes place in November in Chicago, USA.

We have worked on 3 particular areas of new segmentation functionality, selected for their relevance to our product, and overlap with Tina. These are:

- Tissue selection
- "Live wire" boundary tracing.
- Boundary fitting by active shape models

This goal driven approach gave us an insight into the strengths and weaknesses of the Tina libraries, in a commercial software engineering environment, far better than any academic analysis. We have distilled these findings into a critique of the libraries which we hope will inform their future development. Voxar has not been just a passive user of the libraries; we have collaborated with ISBE to address some of issues raised.

The report concludes with consideration of how OSMIA/Tina has benefited Voxar, and how it might continue to do so in the future.

2 Summary of Tina functionality

The Tina libraries are divided into 6 sections:

- **Sys** – low level infra-structure for allocation and manipulation of lists, vectors etc.
- **Math** – mathematical routines particularly for vector and matrix algebra and statistics. Some of this section is based on Numerical Recipes, so permission would be required for commercial use (usually a formality, granted free of charge).
- **File** – sequential representation of Tina structures.
- **Image** – image manipulation, filters, edge detectors.
- **Geometry** – modelling of points, lines, curves and planes, including regression fitting, spatial transforms...
- **Medical** – higher level medical applications – perfusion analysis, active shape models and MR coil correction.
- **Vision** – higher level vision applications, particularly 3D range from stereo (unlikely to be of interests to Voxar).

The sections of most interest to Voxar are **math**, **image**, **geometry** and **medical**. We have assisted ISBE in adding file level summaries in Doxygen[1] format, and these for the **image** section are listed in appendix 1.

[1] Doxygen is an Open Source system for extracting documentation from source code into a variety of formats, including web pages. See http://sourceforge.net/projects/doxygen/.

3 Using Tina for tissue segmentation by example

Currently, Voxar3Ds segmentation facilities are based on thresholding CT density (or MR value), with threshold values taken from the colour/opacity table which parameterises the 3D visualisation. This is ineffective in some situations, since the density (or MR value) of adjacent tissues may differ less than the variation due to noise, or the discrimination may depend on textural features. Even if discrimination by density alone is possible, it can be tedious for the user manually to set the thresholds.

This new technique breaks the link between region selection (= segmentation) 3D visualisation settings. Instead, the user indicates on the side MPR navigation views a few examples of the tissue required, as well as some examples of neighbouring tissue *not* required. The method is summarised as follows:

1. Consider a small neighbourhood of voxels around the indicated points to provide +ve and -ve examples of the required tissue.
2. Compute various local features for these points. Features currently include:
    - Local average in 3x3 window
    - Local average in 5x5 window
    - Tangential smooth in 3x3 window
    - Standard deviation in 3x3
    - Coil corrected intensity
3. Select a sub-set of the available features which best discriminate +ve and -ve cases.
4. Compute these selected features for *all* voxels in the VOI
5. Run the EM algorithm, taking the distribution from each of the user's indications as the seeds to the algorithm. These lead to a fitting of the observed distribution by a mixture of multivariate Gaussian distributions.
6. Use the converged distributions to compute a probability image (assuming equal priors).
7. Threshold this probability image at a level which achieves the proportion of +ve voxels predicted by the EM algorithm.
8. Apply morphological volume filtering [5] to remove any positive material at the interface between two negative regions.
9. Identify the selection as those +ve thresholded voxels connected to the user's +ve indicated points. This connectivity requirement may or may not be desirable, depending on the clinical application.

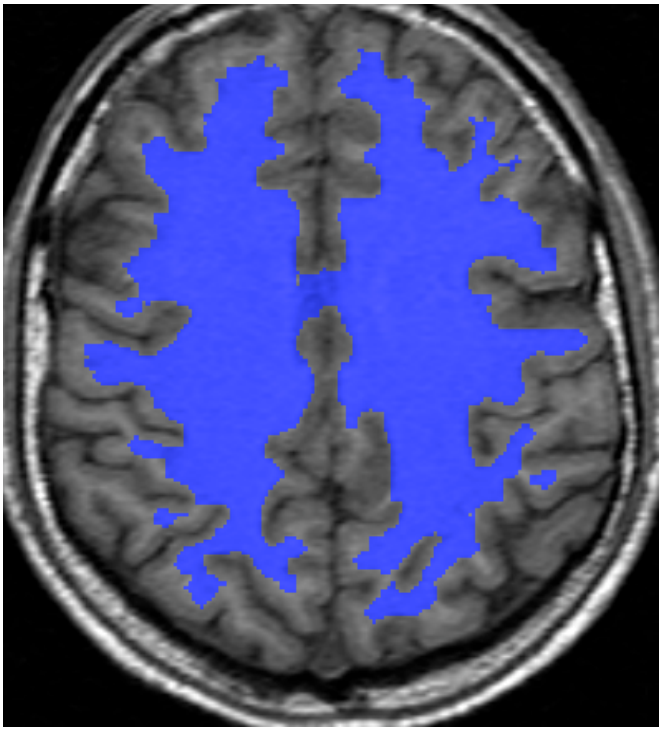
Implementation made use of the following features of Tina:

- Tangential smooth – `im_tsmooth()` in `imgPrc_smooth.c`. This achieves smoothing (noise reduction) while preserving sharp edges.
- Coil correction[2] – `xy_norm()` in `medNorm_base.c`
- Matrix inversion and determinant – `mat_invert()` and `mat_det()` in `mathMatv_invert.c`.

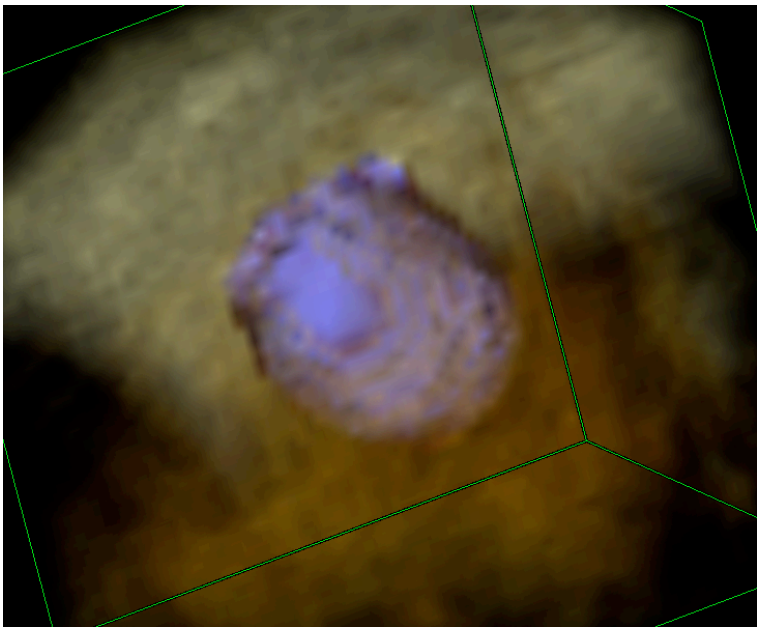
[2] At the time of writing we have had to disable the coil correction feature because of problems with the Tina implementation.

We coded the EM algorithm ourselves, after studying the Tina version, and reading and the ISBE paper [4].

Following are screen captures from the Voxar3D prototype.



Segmentation by example applied to grey/white matter discrimination



Segmentation by example applied to isolating a meningioma in MR (shown as volume rendering)

4 Using Tina for "LiveWire" boundary tracing

Many segmentation problems are too difficult to accomplish automatically, and so are best tackled by semi-automated/interactive means. Tasks such as segmenting the liver, or delineating a highly heterogeneous tumour, fall into this category.

Prior to OSMIA, Voxar3D provided a facility we call "MPR shape selection" which allows a user to define a 3D shape by manually tracing around the boundary on several slices, by the placement of poly-line segments. Not every slice needs to be traced – shape interpolation is used to fill in intermediate slices. Never-the-less, the tracing of the boundary is tedious, time consuming and subject to inaccuracy.

The inspiration to experiment with LiveWire arose during discussions with Tony Lacey (ISBE) and Ovidiu Ghita (VSL) during an OSMIA workshop visit to Manchester. Tony mentioned that a colleague, Graham Vincent, of iMorphics, was using LiveWire in one of their products. Tony arranged for a demonstration by Graham, which convinced us that the technique would be a very useful addition to Voxar3D. He was also able to point us at key papers, and tell us that, as far as he was aware, there were no patents which would prevent us from using the technique (we have since confirmed this with our own searches).

LiveWire [3][4] is a method for accelerating the boundary tracing process. The user clicks on a boundary point, and drags to another boundary point, as before. However, instead of a straight line 'rubber band' being shown, the line follows a path along any 'edge' feature, between the two points. If the user is happy with the path taken, he/she clicks and moves on; if not, they back up to define a shorter segment. Thus, we are dividing the work of boundary tracing between human and computer, according to their respective strengths – humans are good at *recognition* – i.e. which edge should be followed, while the compute is better at accurate *delineation*..

Adding LiveWire to Voxar3Ds MPR shape selection was convenient to prototype, since most of the UI infra-structure was already in place, and we could concentrate on the analysis and use of Tina.

In brief, the analysis involves finding the minimum cost path between two points – the fist being the previously confirmed anchor, the second being the current mouse position. The problem is tractable in linear time (w.r.t to number of image pixels) using a dynamic programming algorithm, only if the cost of a path is defined as the *sum* of the cost for each segment (between adjacent pixels).

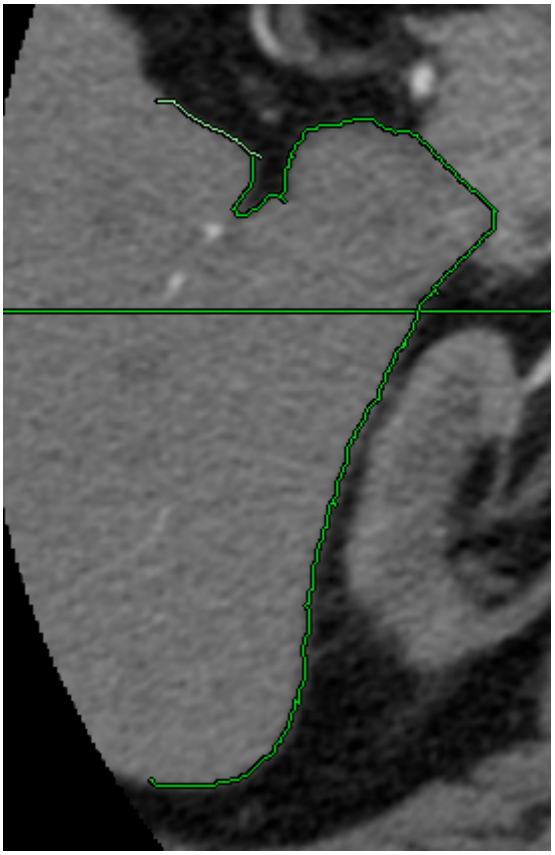
The finesse, is in defining an appropriate per-pixel cost function, such that the path selected by its minimisation turns out to be the subjectively desirable boundary. The cited papers describes a comprehensive cost function which we have, as yet, only partially implemented. The most important components of the cost appear to be:

1. Gradient magnitude (at various spatial scales)
2. Canny edges (at various spatial scales)

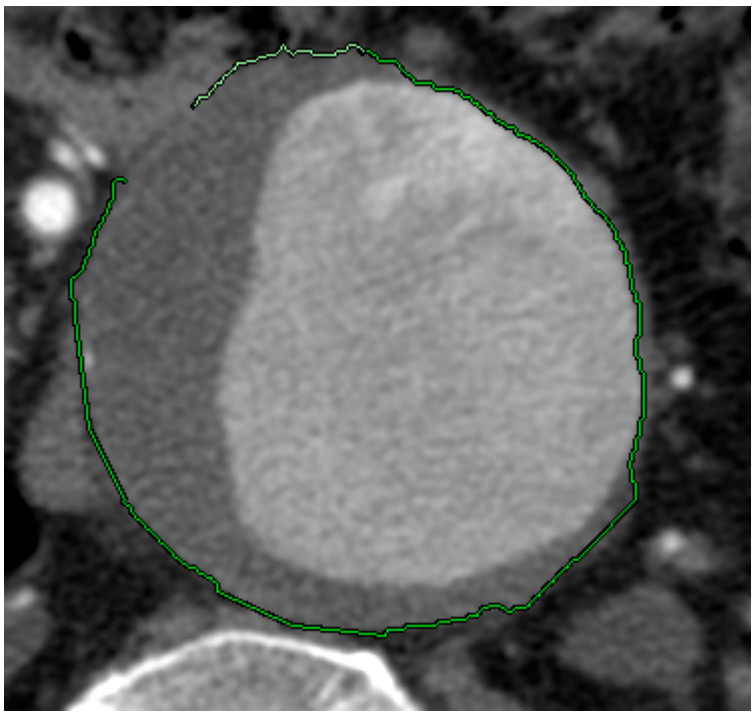
These routines are available in Tina. The first of these is comparatively straight-forward (we have a version in our own libraries). However the Canny edge detector is a complex piece of code which would likely have taken us several weeks to implement. Furthermore, it seems to be the most important component of the cost function – when left out, the tool performs poorly. The code below shows how we interface from our codebase.

```
/* Wrapper for the Tina implementation of the Canny edge detector.  'sigma' is the
 * spread of the gradient operator, in pixels (e.g. 1.0).   'noiseSD' the standard
 * deviation of the noise in the image – it will be estimated if zero is passed.
 * The result is a raster image (short) with 1 indicating an edge pixel, 0 otherwise.
 */
VyImagePtr VyTinaWrappers::Canny (
                      const  VyImagePtr& im,
                      double sigma,
                      double noiseSD) // if <= 0.0 will be estimated
{
       VyImagePtr floatIm = vyToFloatData(im);
       Imrect* tinaIm = VyTinaConversions::CopyToTinaImrect(floatIm);

       /* Scale the image range so that the SD of the noise is 1.0
        */
       if (noiseSD <= 0.0)
              noiseSD = imf_diffx_noise (tinaIm, tinaIm->region);

       /* These constants were recommended by Tony, modified by trial and error.
        */
       double  noiseScale = 0.25;
       dobble lowthresh = 3.0;
       double upthresh = 5.0;

       if (noiseSD > 0.0)
              imf_const_mul(tinaIm, 1.0/(noiseSD*noiseScale));

       Imrect* tinaCanny = canny (tinaIm, sigma, 0.01, lowthresh, upthresh, 5);

       /* The Tina result is a "ptr" image, of pointers to edge structures.   Here,
        * we discard the edge orientation information, and simply set 1 everywhere
        * the result is not NULL, else 0.
        */
       VyImagePtr cannyIm = VyTinaConversions::CopyToVyImage(tinaCanny);
       im_free (tinaIm);
       im_free (tinaCanny);
       VYSCANMACRO_1(cannyIm, {if (*p1 != 0) *p1 = 1; p1++;});
       return vyConvertToType(cannyIm, DatUShort);
}
```

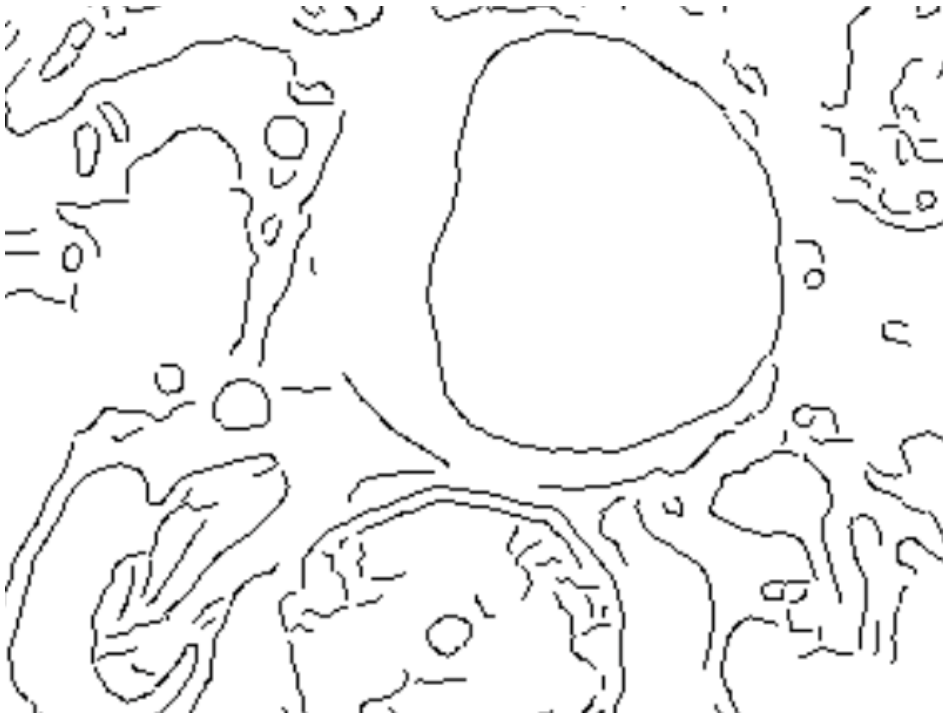
Following are screenshots from the Voxar3D prototype, showing LiveWire in use. The process is highly dynamic and interactive, and is best demonstrated 'live'…



LiveWire used for tracing the outline of liver in a CT dataset



LiveWire used for tracing the outline of thrombus in a CTA dataset



Output from Tina's Canny edge detector (CTA dataset)

5 Active shape models for organ segmentation

At the time of writing, this is an incomplete work in progress. We wish to experiment with the use of active shape models (ASM) [6] as a further acceleration of MPR shape selection. The hope is that after tracing the boundaries manually (with the help of LiveWire) on a few slices, an active shape model could be trained on the fly to fit subsequent slices automatically, requiring only confirmation by the user.

The Tina code for active shape models was not at first included in the conversion to Tina 5, so Voxar volunteered to take the unconverted code and make the necessary modifications. On studying the code we found it made much use of hidden state, through static data (see discussion below) which made it difficult for us to use even in a prototyping context. We thus set about re-factoring the code to remove the static data and expose all state in explicitly passed structures. This we have done, as far as the compilation and linking stage. The testing and debug was taking too much time, however, and so we decided to put this work to one side, and move on. The re-factored code is committed to the CVS repository, to be taken up by ourselves, or others, at a later date.

6 Critique of Tina from a commercial software engineering perspective

Programming language

C is not a modern language, making it more difficult (but not impossible) to solve some of the issues discussed below. C being a subset of C++, it would be possible, trivially, to 'declare' the library to be C++. This would then allow the gradual introduction of C++ features to solve problems - such as namespaces, classes to package state, use of Standard Template Library (STL) containers for better compatibility and readability, variable declaration and

initialization at the point of use… Of course, such a step would lock out any users who for whatever reason, can only work with C.

Imaging structures

Based around "Imrect" with an "Imregion" which allows specification of a sub - VOI. This two level structuring - to allow sub-images with data sharing – is common in image analysis libraries. However, inclusion of the Imrect fields 'width' and 'height' (intended for use in display only) causes confusion and would be best removed.

Tina uses a style of image data access which involves copying a line at a time into a buffer of type double, processing, and then copying back. Although this has a cost (a small one) in efficiency, it does allow for processing code to be written just once for all image data types.

Programming infra-structure - lists, matrices, vectors etc.

A significant proportion of the library is devoted to providing these. Being a home-grown system there is a cost to others learning the library or interfacing with it. In C++ these facilities are well provided for, in a type safe way, via STL. Reasonably well adopted libraries also exist for C - e.g. 'GLib', see http://developer.gnome.org/doc/API/2.0/glib/.

Memory management

Freeing of Tina structures must be done explicitly by the programmer. In C it's impossible to avoid this, whereas C++ allows for automatic destructors, and Java does its own garbage collection. Reference counting (as used by Woolz[3], also in C) is not used, making it difficult to manage the sharing of structures. As a result, the sharing of data by sub-images is little used in practice. Also, where a function can in some situations return an identical image (e.g. im_square(), when the given image is already square) the tendency is to return a *copy* of the image (costing space and time) rather than a shared reference which might end up getting freed twice.

Namespace cleanliness

There is substantial pollution of the global namespace, with e.g., `Imrect, im_free(), str_length(), Vec2`. These are highly likely to conflict if used in conjunction another (similarly polluting) mathematical or imaging library. Apart from the risk of conflicts, it makes the code difficult to read because one cannot easily tell whether a given function/structure is part of Tina and not, perhaps, a standard C library function. In C++ this problem can be solved either by:

a) Using namespaces,
b) Placing global functions inside a wrapping class as static functions,
c) By use of a prefix.

In C only the final option - a prefix - is available. For example GLib uses "G", Woolz uses "Wlz", VyLib[4] uses "Vy" or "vy", VoxarLib[5] uses "Vxr" (even though the latter 2 are in C++).

Program comments

There are very few program comments in the library - typically all one has to go on is the routine's short name. Comments on *how* a function works are not important (C code is a good

[3] Woolz is an image analysis library developed at the MRC Human Genetics Unit, Edinburgh, in the early 1980s and still in use today.
[4] VyLib is Voxar's in house image analysis library.
[5] VoxarLib is Voxar's core in house library, aimed predominantly at 3D volume rendering.

language for this), but commentary on *what* each routine does, including it's assumptions and side-effects, is essential. Trying to infer the *what* from the detailed *how* is very difficult. If, further, the routine has a flaw, then without any statement of what is *intended*, a developer is powerless to provide a fix and will typically program around the problem - re-writing of the enclosing functionality to avoid the problematic routine.

At a higher level, the absence of file level comments make it difficult for a new user even to assess what the library provides.

Reentrancy – hidden state through static data

There is extensive use of static data throughout the library as a means of providing globally persistent state. Following is a count of lines declaring static data, broken down by library section:

Sys	12
Math	26
Image	23
Geometry	53
File	40
Medical	22
Vision	100+

As a result, funcitons cannot be used reentrantly in more than one context, whether via multiprocessor threads or not. The software made difficult to understand since routines can access (for read or write) data at any time, they need/do not declare the data on which they act in their parameters, so giving little away as regards their purpose. Module testing is hampered because a result may depend on the un-encapsulated history of previous calls – so for example it might work the first time, but not the second.

Some use of static data is benign – for example to hold the value of computed constants, lazily initialized by an accessor, then never modified.

Compilation warnings

With the Visual Studio compiler set to the highest level of warning (level 4 - the default used at Voxar) Tina compiles with over 1400 warnings! Most of these are probably benign, but many probably are not. Indeed we have found examples of actual errors which had indeed generated a warning. Even if all the current warnings were benign, they serve to mask new problems as the library is modified.

Working with ISBE we have noted that the Visual Studio and gcc compilers (used on Windows and Unix platforms respectively) generate a substantially different set of warnings. For example, gcc seems not to detect “variable not [or may not be] initialized before use”, while Visual Studio does not warn about missing enumerations in case statements. Both of these categories are useful. Clearly, obtaining warning free compilation across all platforms is difficult, but it is a laudable goal.

Test harness

There is an absence (within the CVS committed version of the library, at least) of any test harness for the library, except for the “Tinatool” application[6]. There is also a scarcity of internal checks on correctness (assertions) - for example there is no facility, even in a debug build, to check for out-of-bound accesses to image data.

[6] Tinatool is an X11 based interactive application, based on the Tina libraries. It is problematic to build for the Windows platform (though it is possible). Voxar has made only minimal use of Tinatool..

Voxar makes extensive use of automated component testing, and has found it to be of great benefit, detecting errors much more cheaply and quickly than allowing them to surface in the integrated product. There are two broad categories of automated testing:

1. **Assertion tests** – modules (functions, classes..) are exercised on a combination of randomly generated and contrived data, and the results are tested for expected invariant properties. For example, an isotropic image filter such as edge detection or Gaussian convolution should be invariant to image transposition or shift; morphological opening should be idempotent; a median filter should not introduce new values to the image….
2. **Regression tests** – these are also based on random or contrived inputs, with outputs saved to file. Subsequent execution of the tests, performed after each commit, are compared with the previous saved results and any differences reported. (At Voxar, the failure of any regression test is automatically emailed to the developer who made the commit!).

In both cases, tests also implicitly exercise assertions included as part of the code – allowing the code often exposes its own errors.

Each type of test has it's strengths and weaknesses. Regression tests are more difficult to manage because of the persistent state (the previous results), but are more useful for higher-level functionality where invariant properties cannot be easily devised (the result of perfusion analysis, for example).

It is important that any test harness runs as a simple text based, console application (no user input, no display of image results). Output should be something like:

```
name_of_a test1: pass/FAIL!
name_of_a test2: pass/FAIL!
…..
All tests passed / n TESTS FAILED!
```

Thus, any new user of Tina can easily build and execute the tests, so gaining confidence that their installation is correct. Seeing a particular routine included in the automated test gives a user confidence that it is 'live' and believed to work. This is important, since in any open source library such as Tina, there will inevitably be some code that is unfinished work in progress, or has been superseded.

Further, it gives users confidence to dive in and fix code they believe to be wrong/inefficient/poorly factored, knowing the tests will at least detect any gross blunders.

Reliability

The visible issues identified above would lead any software engineer to *suspect* the library to have tangible errors. But by dent of sheer effort and/or genius a particular piece of software might triumph over these structural problems to in fact be highly reliable.

How about Tina? Although we have used only a small part of the library, we have encountered several of coding errors. E.g., tangential smoothing (im_tsmooth()) seems to have an error (the central pixel does not contribute); im_median() does not support double images (the recommended type for analysis); im_square() and thence the coil correction code fails; attempt at re-factoring the ASM ("Sroi") functionality identified code options which are not in fact supported. So it seems that the quality issues identified above are not simply academic.

Portability

Because of it's use of C the Tina 5 libraries have proved to be reasonably portable across OS platforms, though some issues remain - e.g. use of the header file `dirent.h` which is not available on the Windows platform.

Ease of installation

Much improved with Tina-5, via use of auto-config and the tidy arrangement of header files. Glitches remain however, and when the auto-config approach fails, it's difficult for a user to resolve the problem. As described in D3.3, at Voxar we decided to side-step these problems by providing a Windows specific (Visual Studio) build project.

Source control

Possibly the greatest improvement to the usability of the libraries, achieved in the scope of the OSMIA project, is the adoption by ISBE of formal source control, through the establishment of a CVS[7] repository. This has been working very well, and has enabled Voxar, ISBE and VSL all to contribute to improving the libraries without conflicts or confusion.

Critique conclusions

Tina has been developed in academia where priorities differ greatly from those of industry. It is no great surprise (to ourselves, or to ISBE) that the library falls short of commercial software engineering standards. The above critique was carried out in June of this year, and it should be said was well received by ISBE. It has resulted in steps being taken to address some of the issues raised here, to which will return below.

7 Conclusions

How has the OSMIA project benefited Voxar?

Voxar has derived real commercial benefit from OSMIA. It was interactions, particularly with Neil Thacker and Tony Lacey (ISBE) that inspired the tissue segmentation and LiveWire prototype work. The latter in particular has been well received within Voxar, and will almost certainly be presented at RSNA 2003. A decision on the timing of introduction into Voxar3D product is pending. Tissue selection also shows promise and may be demonstrated at RSNA, though more work is needed before considering it for product. This interaction with leading academics in the medical imaging field has been as valuable to us as the Tina libraries themselves..

How has Tina library benefited Voxar?

The implementation of these prototypes was facilitated by the Tina libraries. True, the time spent dealing with interface issues, familiarisation, fixing errors etc. likely equals or exceeds the time it would have taken us to code the required routines ourselves (*given* Tina as a guide). But that misses the point. Only by being involved in the project and being good open source citizens, have we been able to interact with leading academics, and to discover the possibilities that Tina offers. Thus, the time invested in familiarisation, having already more-or-less paid for its self (quite apart from OSMIA project funding) will pay dividends in the future as we make further use of the library.

[7] CVS – Concurrent Versioning System – see e.g. http://www.cvshome.org/

Our use of Tina has been significant, but not extensive. We have found it is the 'middle' layer of the library (roughly, routines in the image and geometry sections) which have been most useful to us. At the outset of the project we had expected to use higher level functionality – perfusion analysis and active shape models, for example. As described above, we did attempt to use the ASM code, but this proved difficult, because of the scarcity of programme comments, and the use of static data which was difficult to work with in the context of Voxar3D, even for a prototype.

These issues not withstanding, it will always be difficult to package and parameterise such high level functionality so that it can be used 'as is' in a commercial setting. We have come to realise that a library such as Tina can provide useful building bricks from which complex analysis can be constructed, and even *demonstrate* that analysis through "Tinatool" or a web server interface. However, provision of the functionality in a commercial product will inevitably involve re-implementation at the higher level, to meet the specific needs of the product. That re-implementation requires a thorough understanding of the method involved, best obtained by studying published papers and, crucially, talking to the people who developed the analysis.

Could the Tina library be used directly in a Voxar product?

The quality issues identified in section 6 of this report prevent us from using the libraries, in their present form, directly in a Voxar *product* (as apposed to prototype for demonstration only). The minimum issues that need to be resolved before this would be possible are (refer to section 6 for details):

- Fix namespace pollution
- Removal of static state
- Fix most warnings
- Module and function level comments.
- Basic module test harness

Note that there has already been good progress on this wish-list. Compiler warnings have been reduced, particularly in the important "Image" section of the library and some use of static data has been removed. Voxar has already made a small contribution to add module level comments, and ISBE are in the process of doing more.

The question will become of material significance to us very soon, with the likely decision to include LiveWire in Voxar3D. As discussed in section 4 , this crucially depends on Tina's Canny edge detector. There are three broad approaches we could take:

1. Re-implement ourselves based on published papers.
2. "Cut-and-paste" the relevant Tina code into our own codebase and then fix the above problems, likely replacing list structures etc. with STL equivalents.
3. Select the minimum set of Tina modules (.c/.h pairs) used by Canny; fix the above listed problems for these modules; commit these changes back to the Tina CVS repository; build a mini-tina library from these modules alone.

We would *like* to take the 3rd route, since this would benefit the Tina community in general, and make the libraries more useful to us the next time we need them.

Some of these issues – such as the namespace – are structural and difficult for any one user of the library to fix. We hope that the proposed "OSMIA Foundation" will provide a forum within which these issues can be tackled decisively.

How will Voxar continue to benefit from OSMIA after the project ends?

Concrete future benefits have already been highlighted – productising the LiveWire addition to MPR shape selection, and possibly the tissue segmentation by example.

In the longer term, Voxar expects to

- Exploit more of Tina, in particular:
    - coil correction for improved visualisation and segmentation in MR datasets
    - active shape models, as described earlier
    - 'iso-canny' – a variation on the Canny edge detector, developed at ISBE and recently added to Tina. This might find application in improving threshold based segmentation.
- Continue to improve the quality of the Tina libraries – particularly the modules we wish to use, by directly committing changes via CVS; discuss and collaborate with other key users where deeper structural changes are required.
- Continue the relationship with the OSMIA members, including occasional meetings, in order to continue the cross-fertilisation of ideas. This is particularly relevant to a small company like Voxar, where the internal group working on segmentation and analysis problems, is small.

Clearly, the OSMIA Foundation would be an ideal structure to coordinate our involvement.

Acknowledgement

We are grateful to all of the OSMIA project members for their generous collaboration, with particular thanks to ISBE for their patience and forbearance in helping us get the most out of their Tina libraries.

Appendix 1 – Tina library module summaries ("image" section)

Following are module summaries relating to one of the key sections of the library. These were entered by Voxar (hence some are flagged as "UNCERTAIN". ISBE are in the process of providing module summaries for the other sections, and all will be viewable in Doxygen documentation, accessible at http://www.tina-vision.net/doxygen/html-libs/index.php.

Module	Purpose
imgEM_estep.c	EM Algorithm: Expectation (E) step
imgEM_max.c	EM Algorithm: Maximisation (M) step
imgEM_mix.c	EM Algorithm: Gaussian distribution model.
imgEM_probs.c	EM Algorithm: Computation of probability images from model parameters.
imgGen_alloc.c	Allocation and copying of Imrect - Tina's generic image structure.
imgGen_get.c	Pixel level read access to Imrect data.
imgGen_get_interp.c	Image pixel access by bi-cubic interpolation.
imgGen_getrast.c	Sample a straight line through an image, using bi-linear interpolation.
imgGen_getvec.c	Extract data from image row or col in vector format
imgGen_put.c	Pixel level write access to Imrect data.
imgGen_region.c	Allocation and manipulation of Imregion - structure for rectangular regions of interest.
imgPrc_add.c	Image add
imgPrc_apply.c	Applying per-pixel functions to images, including complex pixel types.
imgPrc_aratio.c	Image re-samping in x or y.
imgPrc_bshift.c	Image (integer) shift by pixel copying.
imgPrc_bskel.c	Morphological thinning (skeleton).
imgPrc_combine.c	Combine of two images via an arbitrary per-pixel function.
imgPrc_complexify.c	Generate a complex image from two scalar images representing real and imaginary components
imgPrc_connect.c	UNCERTAIN - (connectivity analysis?)
imgPrc_conv_1d.c	Separable convolution functions.
imgPrc_conv_prof.c	Compute 1D Gaussian convolution profiles
imgPrc_convolve.c	Convolve two images
imgPrc_create.c	Generate various image patterns, for test purposes.
imgPrc_deriv.c	Derivative operations on images - 1st and second partial derivatives, Laplacian etc.
imgPrc_erf.c	Apply the erf() (cummulative normal error) function to image values, including complex types.

imgPrc_filter.c	One dimensional generic filtering.
imgPrc_fireburn.c	Fireburn alorithm - grow regions of known pixels values
imgPrc_fourier.c	Fast Fourier Transform (FFT) and inverse.
imgPrc_gabor.c	Gabor filter and image.
imgPrc_gauss.c	Gaussian filter and image with arbitrary spread.
imgPrc_grad.c	Simple image gradient, horizontal and vertical.
imgPrc_hist.c	UNCERTAIN
imgPrc_log.c	Log and exp transforom of scalar and complex images.
imgPrc_lsf.c	Linear sequential filter functions
imgPrc_median.c	Median (3x3) filter
imgPrc_morph.c	Morphological dilation and erosion by arbitrary structuring element.
imgPrc_poly.c	UNCERTAIN
imgPrc_prof1.c	Allocation, free and reverse for generic profiles
imgPrc_ptr.c	Handing of pointer images; operations Vec2 and Mat2.
imgPrc_quad.c	Create a square image by reflecting out the original data.
imgPrc_quad.c	Create a square image from im by reflecting out the original data.
imgPrc_rank.c	Rank filter in arbitrary square window.
imgPrc_rot.c	Image rotation, with bi-linear interpolation.
imgPrc_sample.c	Re-sample an image at given scale, with bi-linear interpolation.
imgPrc_scale.c	Finding min and max and scaling (e.g. normalising) images values
imgPrc_scat.c	Scatter plot of complex image values
imgPrc_shade.c	UNCERTAIN
imgPrc_sin.c	Sine and inverse sine transformation of scalar and complex images.
imgPrc_smooth.c	Tangential (edge preserving) smoothing.
imgPrc_smooth_1d.c	One dimensional smoothing function
imgPrc_spiral.c	Generate spiral images.
imgPrc_sqrt.c	Square root transformation of scalar and complex images.
imgPrc_suptype.c	UNCERTAIN
imgPrc_surf.c	UNCERTAIN
imgPrc_warp.c	General image warping
imgPrc_window.c	Clamping image values to a given range of values.
imgPrc_zeropad.c	Create a larger image, padded with zeros.
imgSeq_alloc.c	Sequence Structure Allocation, copying and removal.
imgSeq_frame.c	UNCERTAIN
imgSeq_imstack.c	Image stack handling
imgSeq_logic.c	Functions associated with the Sequence structure - logical stuff to get and set.

imgSeq_slice.c	Functions associated with the Sequence structure (UNCERTAIN)
imgSeq_voi.c	Volume of interest (defined by splines? UNCERTAIN) handling functions

References

[1] OSMIA Deliverable D3.3 – Tina interface system for Voxar3D.

[2] E.N. Mortensen & W.A.Barrett, *Interactive Segmentation with Intelligent Scissors*, Graphical Models and Image Processing 1998.

[3] A.X. Falcao *et al*, *User-Steered Image Segmentation Paradigms: Live Wire and Live Lane,* Graphical Models and Image Processing 60, 233-260 (1988)

[4] M. Pokric, N. A. Thacker, M.L.J. Scott and A. Jackson, *Multi-dimensional Medical Image Segmentation with Partial Voluming,* Proceedings of MIUA Conference, Birmingham, 2001 (http://www.cs.bham.ac.uk/research/proceedings/miua2001/papers/pokric.pdf)

[5] UK patent GB2374774, *Region boundary identification in 2d representation of 3d subject*.

[6] T.F. Cootes C.J. Taylor, D.H. Cooper abd J Graham, *Active Shape Models – Their Training and Application*, Vision and Image Uderstanding 61, 1995.