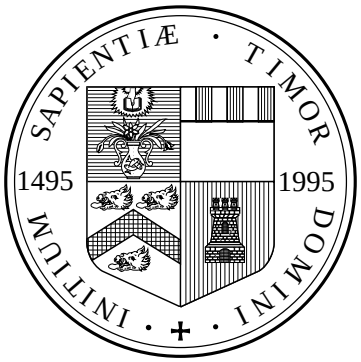
UNIVERSITY OF ABERDEEN
DEPARTMENT OF BIO-MEDICAL PHYSICS AND BIO-ENGINEERING



Comparison of Simulator and Portal Radiographs

Jonathan A. Buzzard

B.Sc. (Honours) Physics with Computer Applications
Heriot-Watt University

This report is submitted in partial fulfilment of the requirements for the degree of M.Sc. Information Technology (Medical Physics) at Aberdeen University and is a record of work carried out during the last four months of the Instructional Course for that degree in Session 1995/96.

Abstract

The problem of accurately setting up patients has been understood for some time, and it is now common practice to take portal films during a course of external beam radiotherapy as part of a complete quality assurance programme.

A method for registration of digitized portal and simulator films was developed based on the identification of anatomical fiducial points on a simulator image and its corresponding portal image to provide a method of patient position verification.

Experiments conducted on pelvic phantom images to check the consistency and accuracy of the registration process indicate that single intra-user accuracy was 0.5° for rotations and 1mm for shifts. Multiple inter-user variation was also examined and found to be 1° for rotations and 1.2mm for shifts.

The technique appears to be suitable as a portal image verification system for radiotherapy treatment.

Declaration

This report is my own composition and has not been accepted in any previous application for a degree. I have carried out the work, of which this is a record, at the Department of Bio-Medical Physics and Bio-Engineering, University of Aberdeen and the Radiotherapy Department of Aberdeen Royal Infirmary during the summer of 1996. All verbatim extracts have been distinguished by quotation marks and the sources of all information specifically acknowledged.

Jonathan A. Buzzard

August 1996

Acknowledgements

I would like to thank the following people for their assistance in this project, Mr. A. G. W. Robertson my project supervisor for his assistance, guidance and the encouragement he gave me throughout this project, Dr. P. E. Kopp for pointing me in the right direction on how to solve overdetermined linear systems, Lorna Dewar for her assistance in taking the phantom images and all those people, who gave up their spare time to register the image set. Finally, I want to express my special thanks to my mother. Without her constant encouragement and support over the past two decades, none of this would have been possible.

Contents

1 Introduction

The aim of this project is to provide a computer program to aid the routine comparison of portal and simulator radiographs for checking the accuracy of patient and beam setups in radiotherapy. Portal radiographs taken on a high energy treatment machine lack the detail of simulator films taken using much lower energy X-rays and require some image processing to enhance the visible features. In addition the two sets of films usually have different magnifications, rotations and translations and will require registering before a comparison can be made.

The goal of radiotherapy is to deliver a prescribed dose of radiation to a tumour as accurately as possible while minimizing the dose to normal surrounding tissues. Therefore, keeping the irradiated volume to a minimum is desirable. This is accomplished by keeping the treatment volume as close as possible to the tumour volume. However, this has to be balanced against the possibility of undertreating cancerous tissue due to positional uncertainties.

Studies of portal images have shown that variations in a patient setup greater than 1 cm are common (Byhardt, Cox, Hornburg & Liermann 1978, Rabinowitz, Broomberg, Goitein, McCathy & Leong 1985, Ding, Shalev & Gluchev 1993). Positional errors of this size mean that the full benefits from improvements in treatment planning by using computers for dose calculation

and CT scans for tumour localization are not being realized. It is increasingly becoming the case that the positional accuracy to which the dose is delivered is the most significant and avoidable factor in treatment complications and failures (Williams & Thwaites 1993). It has been theoretically estimated that reductions in cure rates due to positional errors could be as high as 20% (Goitein 1975, McParland 1993).

Unfortunately the positional accuracy to which the prescribed dose is delivered is also the most difficult to control. This relates to the difficulties involved in setting up the patient, accurately and reproducibly from day to day. The problem of accurately setting up patients has been understood for some time (Marks, Haus, Sutton & Griem 1976), and it is now common practice to take portal films during a course of external beam radiotherapy as part of a complete quality assurance programme. The portal films can then be compared with the simulator film to ensure that patient alignment matches that at the simulation stage, and remains consistent throughout the treatment.

The use of portal images also provides a valuable tool for checking the placement of shielding blocks.

A significant difficulty encountered in the use of portal images for treatment verification is the low contrast that they exhibit. Bone, producing contrast of 18% at photon energies of 50keV, will only show a 2% contrast

when imaged at an energy of 6MeV (Williams & Thwaites 1993). In addition at this higher treatment energy, there are large amounts of scatter further reducing contrast in the image. Unfortunately suitable grids are not available to overcome this. The resultant poor quality of portal images makes their comparison with the prescription (simulator) images a difficult and time-consuming process. A further complication is that the two images are usually at different magnifications, and commonly have rotations and translations with respect to each other, often as a result of film placement as much as than patient setup. These factors combine generally to limit the use of portal images in the modern hospital environment to one check film on the first fraction, which is far from the ideal of verification at every treatment.

Advances in megavoltage imaging systems mean that treatment machines can now be fitted with on-line portal imaging, allowing the acquisition of digital portal images in the first few seconds of treatment. If techniques for the rapid comparison of these images to those acquired at the simulation stage can be developed, a potential for the pro-active detection and correcting of poor patient alignment exists. In this scenario, at the beginning of each treatment fraction, a portal image will be taken and compared with the simulator image, and any problems in the patient setup can then be resolved before the radiation dose is delivered. This is favorable to the current practice of retrospectively correcting the patient setup based on a portal image taken

at a previous treatment fraction, with the possibility of re-simulation in more difficult cases. The improvements in patient setups this would bring may well lead to a reduction in treatment complications and failures and clearly if this were the case would represent a major step forward in accurate patient setups.

For the time being, the requirements of the program are to display portal and simulator images digitized on a scanner. After image processing to enhance the contrast, registration will be carried out. This will correct differences in magnification and bring the two images into alignment, thus making comparisons between the images much simpler. Finally a facility to compare the two images either qualitatively or quantitatively will be required.

2 Mathematical Background

As discussed previously the two radiographs are at different magnifications, rotated with respect to one another, and the images are at different offsets in the film. Therefore before the two images can be compared, they must be brought into alignment by the application of a spatial transformation. A spatial transformation is a mapping function that establishes a relationship among all points in an image and its warped counterpart. To register the two images, a method of determining the spatial transformation involved is required.

In this section we discuss the mathematics behind the spatial transformations involved, and the possible methods available to infer the parameters of the transformation involved for two particular images.

2.1 The Affine Transformation

The three possible transformations of the radiograph (rotation, scale and translation) are special forms of the Euclidean Affine transformation, which is a transformation $w : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, such that the transformation of the point (x, y) is described by the equation

$$w(x, y) = (ax + by + e, cx + dy + f) \tag{1}$$

where a, b, c, d, e and f are real numbers. This can be more conveniently

represented in matrix notation as

$$\begin{aligned} w(x,y) &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \mathbf{x} + \begin{bmatrix} e \\ f \end{bmatrix} \\ \mathbf{W} &= \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \end{aligned}$$

where $\mathbf{A}$ represents a linear transformation and $\mathbf{b}$ a translation.

Lemma 1 *The affine transformation has an important property, that is if* $w_1 : \mathbb{R}^2 \to \mathbb{R}^2$ *and* $w_2 : \mathbb{R}^2 \to \mathbb{R}^2$ *are both affine transformations then* $w_3 = w_1 \circ w_2$ *is also an affine transformation.*

As the three translations that we wish to apply to our radiograph are particular forms of affine transformations, then by lemma 1 (Barnsley & Hurd 1993) there must exist a single affine transformation $\mathbf{W}$ that when applied to the radiograph will apply all three desired transformations in one operation. This reduces the problem of registering the two radiographs to finding the general affine transformation $\mathbf{W}$ that will apply a suitable scaling, rotation and translation to our image.

The affine transformation can be characterized by it's action on three points, as shown in figure 1. If we have three points (x_1, y_1), (x_2, y_2), (x_3, y_3), and the corresponding points $(\tilde{x}_1, \tilde{y}_1)$, $(\tilde{x}_2, \tilde{y}_2)$, $(\tilde{x}_3, \tilde{y}_3)$ after the affine transformation $\mathbf{W}$ has been applied. Then if we substitute these into equation 1, we get a system of linear equations (2) that when solved will give us the

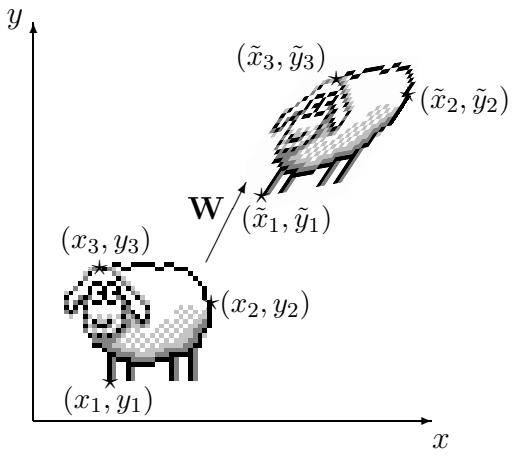


Figure 1: An affine transformation is determined by its action on three points.

coefficients a, b, c, d, e and f of $\mathbf{W}$.

$$\begin{aligned} x_1a + y_1b + e &= \tilde{x}_1 \\ x_2a + y_2b + e &= \tilde{x}_2 \\ x_3a + y_3b + e &= \tilde{x}_3 \\ x_1c + y_1d + f &= \tilde{y}_1 \\ x_2c + y_2d + f &= \tilde{y}_2 \\ x_3c + y_3d + f &= \tilde{y}_3 \end{aligned} \tag{2}$$

Thus by selecting three points in one image and the corresponding points in the other image, by then solving the resultant system of linear equations we have a means by which we can calculate the correct general affine transformation required to register the two images.

Using this approach however has a drawback in that a general affine

transformation has more degrees of freedom than we require. A general affine transformation is composed of a rotation, a magnification of different amounts in the directions of the x and y axis, a shear and a translation. We know however that any difference in the magnifications between the two images will be the same in both directions, and that is also unlikely for there to be any significant shearing between the two images.

Therefore what we require is to find a more specialized form of the affine transformation that only involves a translation (e, f), a rotation θ and a scaling r, uniform in all directions. This special case form of the affine transformation is known as a *similitude* and has the form

$$w(x,y) = \begin{bmatrix} r\cos\theta & -r\sin\theta \\ r\sin\theta & r\cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{3}$$

substituting $\alpha = r\cos\theta$ and $\beta = r\sin\theta$ into equation 3 gives

$$\begin{aligned} w(x,y) &= \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \\ &= (\alpha x - \beta y + e, \beta x + \alpha y + f) \end{aligned}$$

This is now an equation in four unknowns, just two control points (x_1, y_1), (x_2, y_2), and the corresponding transformed points $(\tilde{x}_1, \tilde{y}_1)$, $(\tilde{x}_2, \tilde{y}_2)$ will provide sufficient equations to allow the solution for the four unknowns α, β, e

and f.

$$\begin{aligned} x_1\alpha - y_1\beta + e &= \tilde{x}_1 \\ x_2\alpha - y_2\beta + e &= \tilde{x}_2 \\ x_1\beta + y_1\alpha + f &= \tilde{y}_1 \\ x_2\beta + y_2\alpha + f &= \tilde{y}_2 \end{aligned} \tag{4}$$

It would also be useful if we could determine from the overall affine transformation what the specific rotation, magnification and translation were. Determining the translation is trivial as the two components are separate unknowns in the simultaneous equations (e and f), however the rotation and magnification are more complicated as the rotation θ and the magnification r are combined to give the two unknowns α and β, as

$$\alpha = r\cos\theta \tag{5}$$

$$\beta = r\sin\theta \tag{6}$$

It is possible to separate r and θ by solving this system of simultaneous equations. First re-arranging equation 5 to give

$$r = \frac{\alpha}{\cos\theta} \tag{7}$$

then by substituting equation 7 back into equation 6 we get

$$\beta = \frac{\alpha}{\cos\theta}\sin\theta$$

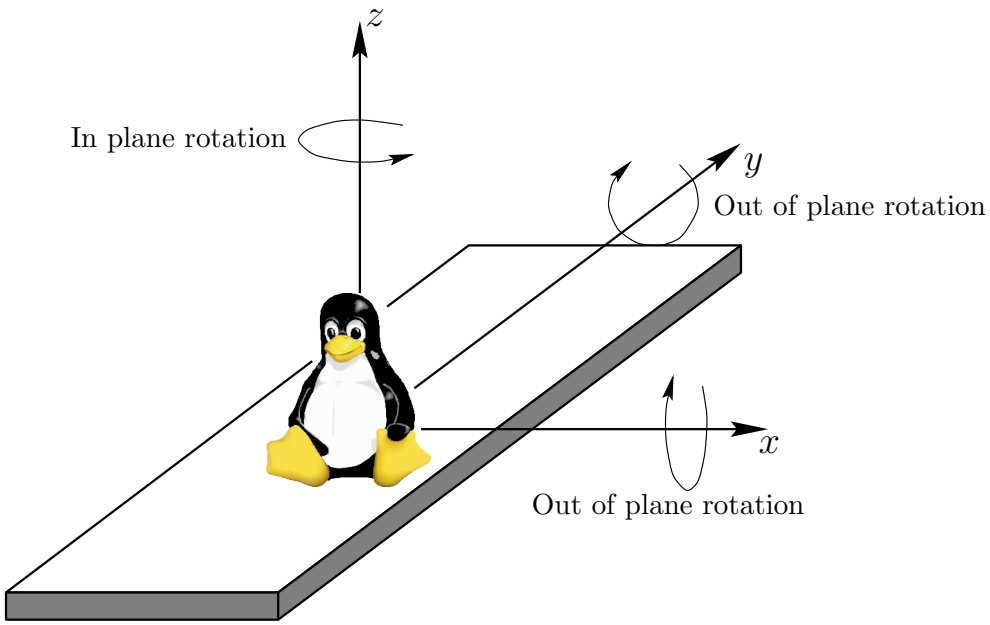


Figure 2: The geometry of the patient and treatment couch.

$$\begin{aligned} \frac{\beta}{\alpha} &= \frac{\sin\theta}{\cos\theta} \\ \tan\theta &= \frac{\beta}{\alpha} \end{aligned} \tag{8}$$

Hence using equation 8 we can work out the angle of rotation θ, which can then be substituted into equation 7 to workout the magnification r. It should be noted that care has to be taken when working out the inverse tangent, due to the limited range of the inverse tangent function ($\tan^{-1}\theta$ has the range $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$). When carrying this out, using an inverse tangent function that takes the numerator and denominator of the fraction θ as separate arguments is preferable. Inverse tangent functions of this form have the full range $[-\pi, \pi]$

One final point to consider is the inability of affine transformations in the

Euclidean plane to cope with off-axis rotations. This is shown by figure 2. From this we can see how movements produce the various transformations seen in the images. Shifts in the xy plane, result in translations in the image, movement along parallel to the z-axis will produce changes in magnification, and rotations around the z-axis will produce rotations in the image.

2.2 Overdetermined Linear Systems

As discussed in the previous section to find the correct transformation required to register the two images we are required to solve a system of simultaneous linear equations. After two pairs of fiducial points have been identified in each image, the coordinates can be substituted into equation 4 and the systems solved.

This however overlooks the fact that each point identified in the second image is unlikely to be in the true corresponding position. In reality when identifying a fiducial point there is likely to be some uncertainty $(\Delta x, \Delta y)$ in identifying it's position. Additionally this uncertainty in the location of the fiducial point will vary between different attempts at identification and between different operators. So while in theory a set of two fiducial points is sufficient to calculate the correct transformation to register the two images, in reality the errors associated with their positions means that the

transformation calculated will not be optimal.

A method to overcome the uncertainties in the placement of the fiducial points is therefore required. The positional uncertainty associated with each individual point will be normally distributed, randomly around it's *true* point.

One method then to overcome these positional uncertainties is simply to increase the number of points used in calculating the transformation. If the uncertainties are truly random, then this will reduce the overall error. This is because as the number of points is increased, the system moves from a random regime to a stochastic one. In the stochastic regime the errors balance themselves out, with an equal number of positive and negative errors. This is analogous to radioactive decay and as the number of fiducial points tends towards infinity the error tends to zero as shown in equation 9.

$$\lim_{n\to\infty} \sum_{i=1}^{n} (\Delta x_i, \Delta y_i) = 0 \tag{9}$$

Unfortunately if we have more than two sets of fiducial points, we will have a system of linear algebraic equations that has more equations than unknowns. Usually such systems have no unique solution, and the set of equations are said to be *overdetermined*.

2.2.1 The Normal System

The system of linear algebraic equations whose solution will yield the transformation required to register the two images is overdetermined, that is we have m equations with n unknowns with $m > n$. If all the fiducial points were perfectly placed a unique solution would exist, however due to the uncertainties in the placement of the points, a unique solution does not exist. What is required then is to seek a "compromise" solution to the system of equations $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, which comes closest to satisfying all of the equations simultaneously.

The solution first proposed by Gauss, is to seek the parameters $\mathbf{x}$, which would have the property that the sum of the squares of the differences between the left- and right-hand sides of the equation are minimized. This is known as the *linear least-squares* problem. The solution proposed by Gauss is to solve a reduced set of n equations in n unknowns, given by equation 10

$$(\mathbf{A}^{\mathrm{T}} \cdot \mathbf{A}) \cdot \mathbf{A} = (\mathbf{A}^{\mathrm{T}} \cdot \mathbf{b}) \tag{10}$$

where $\mathbf{A}^{\mathrm{T}}$ denotes the transpose of the matrix $\mathbf{A}$. The system of equations 10 are known as the *normal system*, whose solution will yield a solution vector $\mathbf{x}$ that minimizes the sum of the squares of the residuals.

Unfortunately the condition number of matrix $\mathbf{A}^{\mathrm{T}} \cdot \mathbf{A}$ is the square of

the condition number of the matrix $\mathbf{A}$. This means that if the residuals are small then any numerical solution of the normal system, will suffer a serious decline in accuracy due to round off errors. So while it may be tempting to solve the normal system directly by Cholesky decomposition, exploiting the symmetrical and positive definite properties of the matrix $\mathbf{A}^{\mathrm{T}} \cdot \mathbf{A}$, this will lead to a numerically unstable answer.

2.2.2 Singular Value Decomposition

The method of choice for solving a linear least squares problem, without prior knowledge of the system is *Singular Value Decomposition* or SVD. This is a powerful technique for dealing with equations which are singular, or close to. This is exactly what we have in our normal system. The technique of singular value decomposition relies on the theorem of linear algebra, shown in lemma 2 (Press, Teukolsky, Vetterint & Flannery 1992), the proof of which is beyond the scope of this document.

Lemma 2 *Any $m \times n$ matrix $\mathbf{A}$ whose number of rows m is greater than or equal to its number of columns n, can be written as the product of an $m \times n$ column-orthogonal matrix $\mathbf{U}$, an $n \times n$ diagonal matrix $\mathbf{W}$ with positive or zero elements (the* singular values*), and the transpose of an $n \times n$ orthogonal matrix $\mathbf{V}$.*

The technique works by avoiding the direct computation of the normal system. Considering our set of simultaneous equations

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{11}$$

where $\mathbf{A}$ is a m by n matrix, and $\mathbf{x}$ and $\mathbf{b}$ are vectors. Then if we then perform singular value decomposition on the matrix $\mathbf{A}$ to give us the matrixes $\mathbf{U}$, $\mathbf{W}$ and $\mathbf{V}$ and substitute these into equation 11 according to lemma 2 we get

$$\mathbf{U} \cdot [\,\mathrm{diag}(w_j)\,] \cdot \mathbf{V}^{\mathrm{T}} \cdot \mathbf{x} = \mathbf{b} \tag{12}$$

Now as both $\mathbf{U}$ and $\mathbf{V}$ are orthogonal, then the products $\mathbf{U} \cdot \mathbf{U}^{\mathrm{T}}$, and $\mathbf{V} \cdot \mathbf{V}^{\mathrm{T}}$ are both equal to the identity matrix $\mathbf{I}$. Additionally as $\mathbf{W}$ is a diagonal matrix, then its inverse is the diagonal matrix whose elements are the reciprocals of the elements w_j. From equation 12 it follows that the solution vector $\mathbf{x}$ is given by

$$\mathbf{x} = \mathbf{V} \cdot \left[\mathrm{diag}\left(\frac{1}{w_j}\right)\right] \cdot \mathbf{U}^{\mathrm{T}} \cdot \mathbf{b} \tag{13}$$

This "solution" vector $\mathbf{x}$ constructed by equation 13, will not exactly solve equation 11. However it will be the solution, from the set of all possible solutions that minimizes the sum of the squares of the residuals between the left- and right-hand side, ie. the least squares solution.

Therefore by calculating the singular value decomposition of the matrix **A** formed from the set of linear simultaneous equations, resulting from the coordinates of the fiducial points we have a means by which we can calculate the "best" transformation to register the two images.

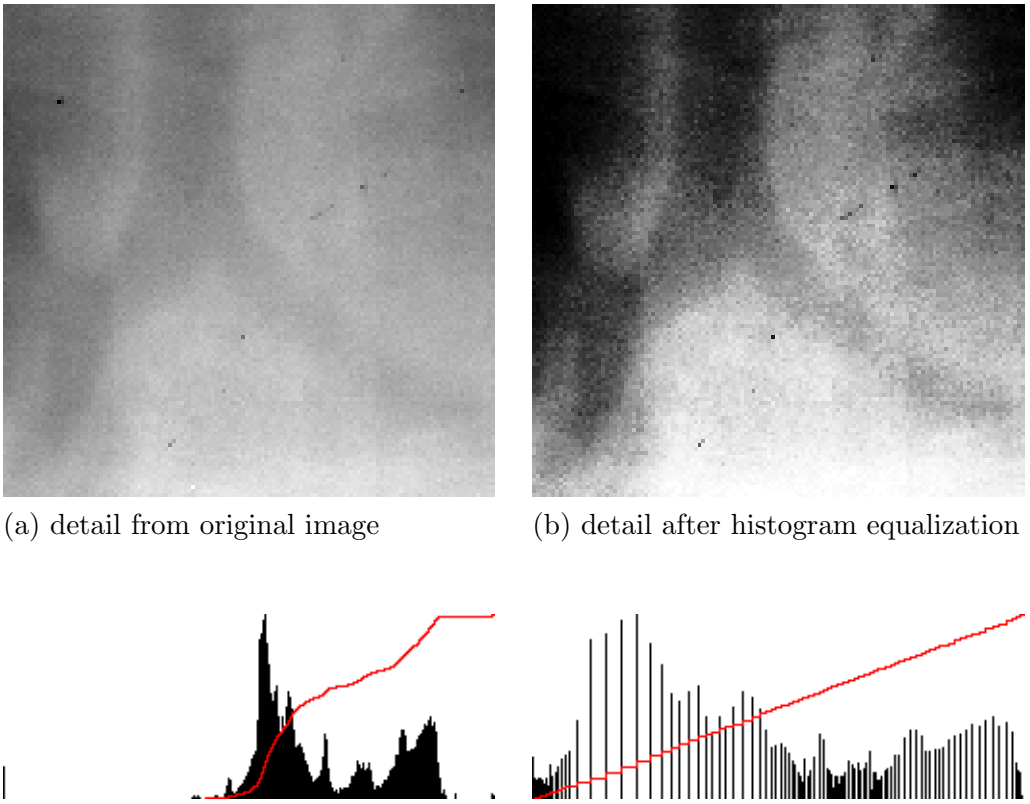
3 Image Processing

The portal images suffer from poor contrast, primarily due to the much higher energies of the photons in comparison to standard X-rays. This makes the identification of suitable anatomical points in the images for use in registration a difficult and often impossible process. Therefore before the selection of fiducial points takes place, processing the portal image to improve the contrast is necessary.

3.1 Histogram Equalization

The commonest form of contrast enhancement is a 1–to–1 pixel transformation where the output image is formed by using a lookup table to convert the brightness of the pixel in the input image to a new brightness value in the output image.

The most frequently used technique of this type is known as *histogram equalization*, and is based on the assumption that a good grey-level assignment scheme should have equally distributed brightness levels over the whole brightness scale. Individual pixels retain their brightness order (i.e. they remain brighter or darker than other pixels). However, the values are shifted so that they are equally distributed over the brightness scale. The result of the brightness transformation should be that the cumulative histogram becomes



(a) detail from original image

(b) detail after histogram equalization

(c) histogram of original image

(d) histogram after equalization

Figure 3: The effect of a histogram equalization

a straight line.

As a digital image has only a finite number of grey scales, an ideal equalization is not possible. This causes some pixels with initially different brightness values to be assigned the same value, and other values to be missing altogether. The result of a histogram equalization on a typical portal radiograph can be seen in figure 3.

Formalizing this, if we have a pixel of brightness p_b in the original image from a range $[p_0, p_k]$, then if t_b is the number of pixels in the image at a

brightness level b or less, defined as

$$t_b = \sum_{i=0}^{b} p_i \tag{14}$$

then if there are a total of n pixels in the image, the new brightness value q is defined to be

$$q = \frac{t_b(k-1)}{n} - 1$$

The histogram equalization enhances the contrast for brightness values close to maxima in the histogram and decreases contrast near the minima. That is, it improves the contrast in the image in areas of poor contrast at the expense of those areas where there is already good contrast.

A limitation of histogram equalization is that large peaks in the histogram can also be caused by large areas of similar brightness. Frequently these correspond to areas of background, and are essentially uninteresting. The effect of histogram equalization on these areas is to enhance the visibility of noise.

A simple method by which this can be overcome for either background or foreground areas is to display the histogram and allow the user to select a cutoff value. All pixels in the input image of the same or lower brightness are mapped to the minimum brightness level, and remaining brightness levels are equalized. Similarly a cutoff value can be defined for foreground areas, where

all pixels equal or above a selected brightness are mapped to the maximum brightness level.

Another limitation of the technique is that it does not adapt to local contrast requirements; minor contrast differences can be missed entirely when the number of pixels falling in a particular brightness range is small. This weakness can be overcome by using an adaptive histogram equalization technique, which optimizes image contrast locally (Pizer, Amburn, Austin, Cromartie, Geselowitz, ter Haar Romeny, Zimmerman & Zuiderveld 1987). In particular a technique known as contrast limited adaptive histogram equalization (CLAHE), which seeks to reduce the noise produced in homogeneous areas by basic adaptive histogram equalization, and was originally developed for medical imaging, has been successful for the enhancement of portal images (Rosenman, Roe, Cromartie, Muller & Pizer 1993).

3.2 Colour Schemes

A further method to improve the contrast perceived by the operator is to optimize the representation of the brightness levels to maximize the contrast.

The representation of brightness by grey shades such that as the brightness increases it moves from black through grey to white are by far the most common. Unfortunately, all the shades used in this scheme are different satu-

rations of the same hue (i.e. grey). This presents a problem as only a limited number of grey levels are perceivable as different to the human eye.

To maximize the ability to distinguish between adjacent levels of brightness, changing both the hue and the saturation is necessary. Two colour schemes that use the extended range offered by using colour coding while retaining the ability for the observer to see adjacent brightness levels as possibly relating to the same image feature, are the Blue/White and Hot Body.

In the blue/white scheme, the coding of the brightness levels moves from black through shades of blue ending in white. The hot body scheme varies from black through shades of red, orange, yellow, ending in white and is designed to follow the emission spectra of a black body as it is heated up. Both these schemes offer improvements in the perception of contrast changes and hence localization. It has to be born in mind however that there is often resistance among medical personnel to view images in anything other than grey scale.

4 Implementation

The chosen language for the implementation of the algorithms previously discussed was Interactive Data Language or IDL (Research Systems Incorporated). IDL is an interpreted array-oriented language combined with many graphical display and mathematical analysis routines. It also incorporates routines to allow the writing of programs with graphical user interfaces. Programs written in IDL can be run on all supported platforms (Unix, VMS, Microsoft Windows, Macintosh) with only small modifications.

4.1 User Interface

The user interface for the program was implemented using the IDL widgets. Widgets are simple reusable graphical objects, such as buttons, menus, scrollbars or display areas, that allow user interaction via a pointing device and a keyboard. A special class of widgets, mangers, exists that control the layout of other widgets.

The user interface consists of two scrolled draw widgets placed horizontally next to each other in which the two images to be compared are loaded. Below each scrolled draw widget is a row of buttons to enable the image to be histogram equalized, flipped and rotated in 90 degree increments. A menu bar for the common operations such as loading images, clearing control

points, and performing transformations runs along the top of the window. Along the base of the window is a status bar for passing messages to the user. Control points are selected by clicking at the appropriate places in the two scrolled draw widgets (see figure 4).

4.2 Image Acquisition

The images were obtained by taking both standard portal and simulator radiographs and digitizing them, using a scanner. The scanner used, was a Lumiscan™ 100. This laser scanning device digitizes a radiograph by scanning a focused spot of light across the film, as it is moved perpendicular to the scan. The transmitted light is then converted to an electrical signal using a photo-multiplier tube and digitized. The Lumiscan™ 100 can digitize a radiograph up to 14" × 17" using 12 bits for the intensity, and up to a resolution of 256 pixels per inch.

The image format that the Lumiscan™ device saves digitized images in is proprietary, so before the scanned images could be used a custom routine to phrase the image file and load in the data had to be written. The scanner digitizes the image using 12 bits for the intensity, so when this is stored in the image file the intensity level for each pixel is stored in two bytes. This means that any routine to read in the image has to be aware of the endianes

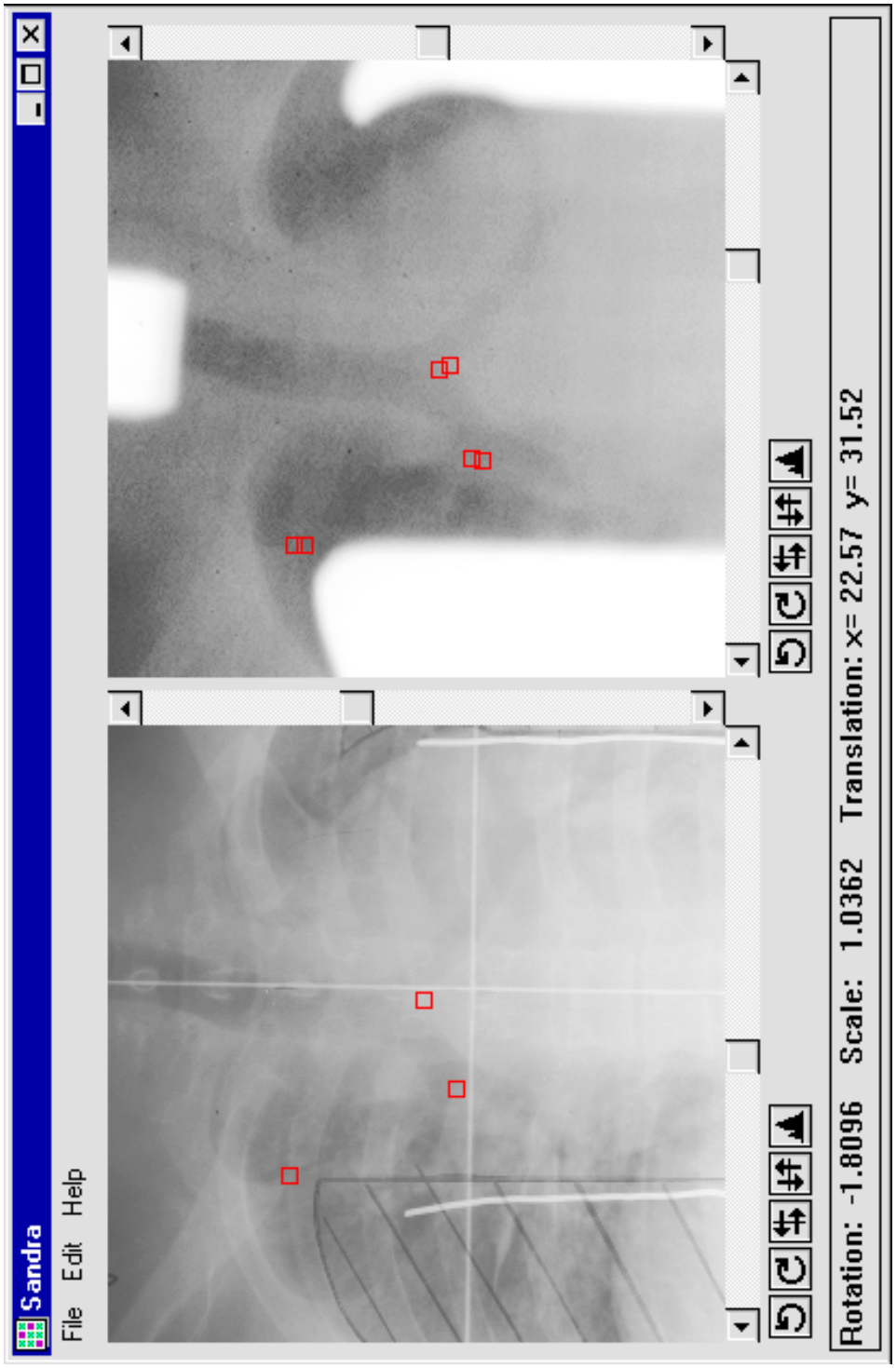


Figure 4: Screen shot of program after registering two images.

(byte order) of the machine on which it is running, or the loaded image will be gibberish. The endianes of the machine on which the program is running is tested, and if it differs from the endianes in which the image is stored (always little endian) a byte swap is done on the image data. For Intel x86 machines this is not a problem as they are little endian, but many Unix machines and Macintoshes are big endian. The code for this routine can be seen in Appendix B.

4.3 Image Display

The images are copied from their byte arrays into the draw widgets using IDL's `TVSCL` routine. This procedure takes a two dimensional array of numbers and outputs the data to the display, scaling the data to use the maximum number of available colours. A blue/white colour lookup table was used to enhance the contrast.

The image display facilities of IDL are much simpler than using the underlying windowing system. However, a limitation of IDL is that it only uses a maximum of 256 colours. An additional drawback is that each time the image is displayed it has to be converted into the window systems device independent bitmap format. A program written for the native window system would manipulate this bitmap directly, therefore reducing memory overheads

and speeding up image display.

4.4 Histogram Equalization

The built in histogram equalization routine in IDL (`HIST_EQUAL`) is severely limited. Only basic global equalization with cut-offs is provided, and even this has the unfortunate side effect of converting all output images to 8 bits. Additionally the minimum and maximum cut-off values only work on byte input data. The interpretive nature of the IDL language means that improved adaptive histogram equalization techniques that have shown promise in the area of portal images (Rosenman et al. 1993) would be prohibitively slow. The histogram equalization as implemented brings up a modal dialog box that displays a histogram of the image. The user is then free to interactively select minimum and maximum cutoff values using two sliders to optimize the histogram equalization to produce the maximum contrast improvement (see figure 5).

4.5 Fiducial Point Selection

Pairs of fiducial points are selected by clicking on both images to identify the coordinates in each image of a structure present in it. This causes a small square, the centre of which is the position of the point selected, to be drawn

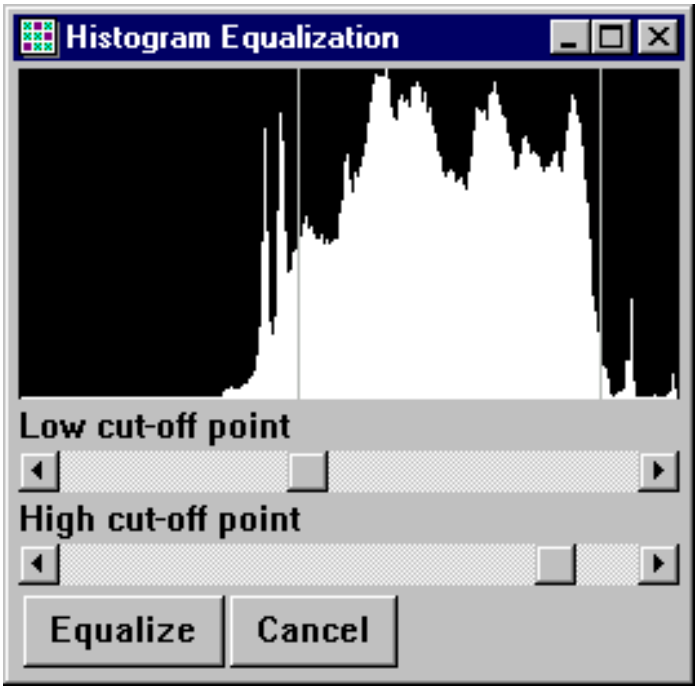


Figure 5: Histogram equalization dialog box

on the image. The points have to be selected in their corresponding pairs, i.e. after the selection of a point in the left image the corresponding point must be chosen in the right image before another point can be selected in the left image. The status bar is used to display the coordinates of the selected fiducial point and how many pairs of points have been selected. The status bar is also used to pass suitable error messages to the user if they click out of sequence on an image.

Although the automatic selection of fiducial points using image processing techniques is theoretically possible, no guarantee would exist that the selected points would correspond to invariant features in the images. The selection of

suitable fiducial points represents the slowest part of the procedure for image alignment.

A facility is provided for the user to clear the selected fiducial points if an error has been made. Also, rotating, flipping and loading a new image causes all the selected fiducial points to be automatically cleared.

4.6 Image Transformation

The built in routine `POLY_2D` was used to transform the image. This is a general purpose routine that geometrically transforms any two dimensional array of numbers using an arbitrary polynomial. The resulting array is defined by

$$v(x, y) = u(x', y') = u(a(x, y), b(x, y))$$

where $v(x, y)$ represents the pixel in the output image at coordinate (x,y), and $u(x', y')$ is the pixel at (x', y') in the input image used to derive $v(x, y)$. The functions $a(x, y)$ and $b(x, y)$ are polynomials in x and y of degree n, whose coefficients are given by U and V and specify the spatial transformation

$$\begin{aligned} x' &= a(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} U_{i,j} x^j y^i \\ y' &= b(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} V_{i,j} x^j y^i \end{aligned}$$

The `POLY_2D` routine uses inverse mapping, this means that the brightness of pixels in the transformed image is calculated by looking at which point

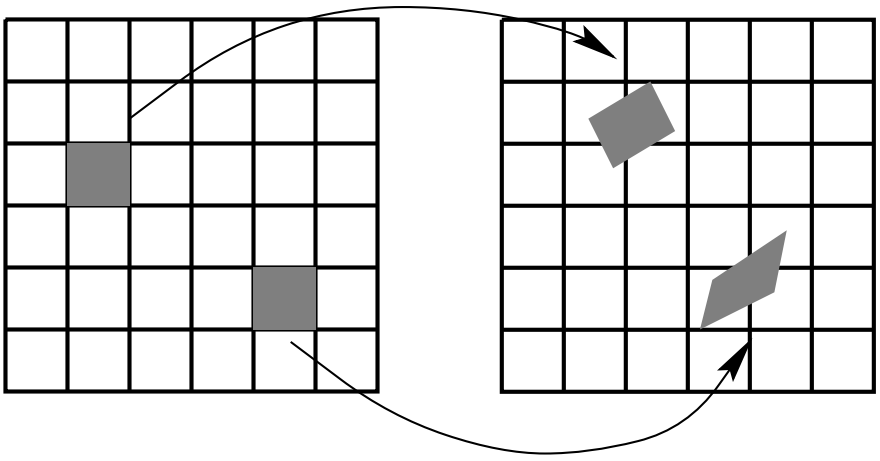


Figure 6: Transformed pixels do not lie on an integer lattice

in the input image, the output pixel corresponds to. Various sampling and interpolation techniques can be used to calculate the brightness value as it is unlikely that the input pixels will exactly correspond to an output pixel (see figure 6).

The transformation passed to the `POLY_2D` routine is that required to map the transformed image onto the input image. This is counter intuitive, but is accomplished by simply swapping the points around in the system of linear equations (4). The singular value decomposition is calculated using IDL's `SVDC` routine. This is an implementation of the routine found in Press et al. (1992). The individual components that make up the transformation are displayed on the status bar.

The `POLY_2D` routine can use either nearest neighbour, bilinear or cubic

interpolation. Cubic interpolation was used as it produces the best results, although it is much slower than bilinear or nearest neighbour interpolation.

To provide some feedback to the operator on the accuracy of the selected fiducial points, the points for the transformed image were transformed as well, and plotted alongside those from the reference image. Using this method the better the pairs of points match up in the two images, the closer the two points when plotted. Poorly selected, points are immediately apparent, as they will show significant separation from one another.

4.7 Image Comparison

To allow the two images to be compared after alignment, a fader dialog is provided. This superimposes the two images upon each other. The fraction of each image displayed is controlled by means of a slider. This allows the user to fade from one image to the other, by which means deciding whether any mismatch between the simulator and portal images is possible.

4.8 Summary

Several drawbacks to IDL were encountered while conducting this project. The range of widget types provided is limited, in particular the geometry management provided does not extend to resize events. The behaviour of the

widgets was found to vary significantly under different platforms, particular problems existing in the creation of scrolled drawing area widgets.

Medical images are often 10 and 12 bit, especially in the field of digital radiography. The current version of IDL (4.0.1) provides little support for images with more than 8 bits, routines often scaling the output of 10 and 12 bit images to 8 bits. The image display routines use a maximum of 256 colours despite 16/24 bit graphic cards being the norm. I would contend that those problems make IDL less than ideal for the processing of many medical images.

The interpreted virtual machine architecture of IDL slows down any processing of large datasets outside the built in routines. Although provision is made for writing external routines in languages such as C or Fortran, it is not supported on all platforms and adds an extra layer of complexity. The provision of a *just in time* compiler for the virtual machine would lessen this problem. The just in time compiler converts the byte code of the virtual machine into machine code immediately before execution. This technique can provide performance similar to compiled C code and would eliminate the speed problems.

For rough and ready implementations of algorithms to test their effectiveness, IDL is powerful and useful tool. However factors such as cost, speed, flexibility, and robustness means that for writing programs that would be

used on a day to day basis in a clinical environment, traditional languages like ‘C’ remain the first choice over tools such as IDL.

5 Results

In order to estimate the reliability and accuracy of the image registration algorithms, a series of pelvic phantom images with known shifts, rotations and magnification changes was taken. The results detected by the algorithms were then compared with the known values. Both the intra-user and inter-user variation in image registration were studied. As the images were obtained by digitizing films, errors in alignment between the images existed due to movement of the film in the cassette, and on the scanner. Therefore before alignment took place these errors were corrected by digitally transforming the images such that the bottom field edge was horizontal and the field centre was in the same position in all images.

Table 2 shows the results of the intra-user variation for the alignment of image pairs using six pairs of fiducial points on pelvic phantom images. Each image pair was aligned on five separate occasions, and the shifts have been converted into millimetres using pixel size and image magnification. The shifts produced by the affine transformation are in the xy plane as opposed to shifts along and perpendicular to the line of rotation that the simulator/treatment machines use. The figures given here have been converted into machine geometry. Table 3 shows the average inter-user variation for the alignment of image pairs, again using six pairs of fiducial points.

Image pair	Rotation (°)	Magnification (%)	Long (mm)	Lateral (mm)
1–2	2.0	1.0000	10	0
1–3	-1.1	1.0000	10	5
1–4	-0.6	1.0714	-8	5
1–5	-1.6	1.0357	-8	-10
1–6	-0.9	1.0357	-4	-7
2–3	-3.1	1.0000	0	5
2–4	-2.6	1.0714	-18	5
2–5	-3.6	1.0357	-18	-10
2–6	-2.9	1.0357	-14	-7
3–4	0.5	1.0714	-18	0
3–5	-0.5	1.0357	-18	-15
3–6	0.2	1.0357	-14	-12
4–5	-1.0	0.9615	0	-15
4–6	-0.4	0.9615	4	-12
5–6	0.7	1.0000	4	3

Table 1: Differences between images as set on machine.

Image pair	Rotation (°)	Magnification (%)	Long (mm)	Lateral (mm)
1–2	-0.14 ± 0.12	0.09 ± 0.40	1.2± 0.14	0.6± 0.31
1–3	-0.01 ± 0.13	-0.14 ± 0.18	0.9± 0.18	-0.4± 0.19
1–4	0.28 ± 0.10	0.34 ± 0.20	0.1± 0.19	0.6± 0.13
1–5	0.09 ± 0.09	0.30 ± 0.14	-0.6± 0.16	1.1± 0.18
1–6	0.01 ± 0.10	-0.11 ± 0.16	0.3± 0.23	0.2± 0.15
2–3	0.05 ± 0.09	-0.01 ± 0.18	0.5± 0.13	0.3± 0.14
2–4	0.32 ± 0.14	0.63 ± 0.33	-0.9± 0.28	1.4± 0.13
2–5	0.20 ± 0.20	0.28 ± 0.09	-1.1± 0.12	2.0± 0.11
2–6	0.11 ± 0.09	0.13 ± 0.19	-0.3± 0.14	0.8± 0.12
3–4	0.25 ± 0.06	0.28 ± 0.10	-0.5± 0.25	0.7± 0.28
3–5	-0.03 ± 0.09	0.00 ± 0.28	-1.4± 0.23	1.5± 0.18
3–6	0.09 ± 0.17	-0.05 ± 0.25	-0.4± 0.22	0.3± 0.11
4–5	-0.06 ± 0.11	0.27 ± 0.18	-0.0± 0.22	0.8± 0.17
4–6	-0.11 ± 0.22	0.05 ± 0.25	0.8± 0.32	-0.1± 0.15
5–6	-0.02 ± 0.22	-0.09 ± 0.25	0.9± 0.21	-0.7± 0.30
Average	0.07 ± 0.19	0.13 ± 0.31	-0.0± 0.80	0.6± 0.71

Table 2: Average errors and standard deviations for intra-user registration

Image pair	Rotation (°)	Magnification (%)	Long (mm)	Lateral (mm)
1–2	-0.10 ± 0.06	-0.06 ± 0.42	1.1± 0.22	0.6± 0.45
1–3	0.06 ± 0.08	-0.33 ± 0.18	0.9± 0.33	-0.4± 0.12
1–4	0.28 ± 0.18	0.07 ± 0.40	0.1± 0.25	0.7± 0.28
1–5	0.17 ± 0.07	0.12 ± 0.20	-0.6± 0.22	0.8± 0.32
1–6	0.12 ± 0.08	-0.23 ± 0.11	0.4± 0.38	-0.9± 1.82
2–3	0.33 ± 0.54	0.11 ± 0.41	0.2± 0.78	-0.0± 0.41
2–4	0.26 ± 0.24	0.19 ± 0.25	-0.9± 0.21	1.1± 0.40
2–5	0.39 ± 0.09	0.21 ± 0.24	-1.0± 0.40	1.7± 0.49
2–6	0.02 ± 0.10	-0.05 ± 0.46	-0.5± 0.17	0.8± 0.14
3–4	0.11 ± 0.07	0.35 ± 0.32	-0.7± 0.27	0.9± 0.21
3–5	0.01 ± 0.12	-0.07 ± 0.26	-1.7± 0.22	1.3± 0.48
3–6	0.16 ± 0.17	-0.23 ± 0.21	-0.3± 0.39	0.4± 0.06
4–5	0.02 ± 0.22	0.42 ± 0.43	-0.0± 0.43	0.3± 0.35
4–6	-0.01 ± 0.23	0.44 ± 0.12	0.5± 0.11	-0.1± 0.26
5–6	0.06 ± 0.17	-0.35 ± 0.11	0.9± 0.18	-0.6± 0.21
Average	0.13 ± 0.24	0.04 ± 0.39	-0.1± 0.86	0.4± 0.91

Table 3: Average errors and standard deviations for inter-user registration

6 Discussion

The phantom studies indicate that the accuracy of intra-user alignment for the method is less than 0.5° for the rotation and approximately 1mm for shifts. The inter-user variation is greater, with an accuracy of 1° for the rotation and approximately 1.2mm for the shifts.

How much of this variation relates to the unfamiliarity of the users with the program and the alignment procedure is unknown. The majority of the users in the inter-user study were using the program for the first time. Given that the results for the inter-user study are still acceptable however, suggests that the algorithms are robust.

The greater variation in the inter-user alignment indicates that before image registration software could be use in a clinical environment users would require proper training to ensure proficiency. This could take the form of a series of test images of various anatomical sights with known offsets. The user could then attempt alignment of the images. When a suitable level of accuracy on each image and across the whole image set they could be deemed competent.

7 Summary

7.1 Future Work

A quantitative approach for assessing the accuracy of the alignment process is needed. The further the selected points are from the ideal (i.e. when the sum of the squares of the residuals is zero), the further apart the points appear when plotted on the transformed image. However this does not help the user to decide when to reject the transformation due to bad fiducial point placement and try again. A simple numerical measure, independent of the number of points used, that showed the uncertainty in the transformation would probably be sufficient. One possibility is to use the sum of the squares of the residuals from the solution of the normal system divided by the number of points used. Whatever measure is adopted, work would be required to assess the normal variation in the figure and at what point to flag a bad transformation.

The use of a tablet as the input method rather than a mouse should be explored. In general people have more control over a pen than a mouse. For example preventing a cursor moving one or two pixels while clicking on a button is hard, whereas while pressing down on a tablet with a stylus this is not a problem.

The selection of fiducial points in the portal image, after histogram equal-

ization has been applied is still difficult. The inclusion of more advanced portal image processing techniques such as contrast limited adaptive histogram equalization to improve the visibility of portal anatomy should be considered. This should make the selection of fiducial points easier, faster and more accurate.

Before using the program to correct patient alignment pro-actively, the registration transformation would need converting into machine geometry. Besides knowing the pixel size and the magnification of the two images, this also requires the coordinates of the centre of rotation of the image. One method of identifying the centre of rotation, which is also the beam centre would be to get the user to identify the four corners of the beam on the image, from which the centre could be calculated. However this method would not work on beams where the corners are obscured by shielding blocks or a multi-leaf collimator. Alternatively this could be assumed to be in the centre of the image, and the alignment of the on-line portal imaging system incorporated into the quality assurance programme.

The time required to align an image file is dependent on the experience of the user. Currently it takes about two minutes for a proficient user to align two images. This includes the loading and image processing times. If an image registration tool of this nature was to be use clinically for the pro-active correction of poor patient alignment, this would need to be reduced.

Two areas that could yield significant reductions in the alignment time are a larger display and more processing power (or a reduction in current demands by increasing computational efficiency).

A larger display (i.e. screen resolution) enabling both the portal and simulator images to be displayed simultaneously in their entirety, would save the user form having to scroll the two images around looking for suitable fiducial points.

Another major overhead is the time required to do the simple image processing such as rotations/flips, histogram equalization and the transformation. Either the introduction of a workstation class machine or rewriting of the program in a compiled language such as C or C++ would reduce the time spent waiting for the computer to finish processing, reducing the registration time.

7.2 Conclusions

The results of the phantom studies show that the accuracy of the technique is similar to reports in the literature, though in clinical practice the accuracy of the alignment is unlikely to be this high. The size of both intra and inter-user variation in the alignment process is small enough to offer improvements in patient alignment, that show errors of more than 10mm. Studies of por-

tal images have shown that these could account for as many as 15% of all treatments (Lam, Partowmah, Lee, Wharam & Lam 1987).

The technique appears suitable for the verification of patient position in external beam radiotherapy treatment. The small size of the variations in the alignment transformation indicate that the algorithms are robust, and the small size of the absolute errors offers the possibility of being able to improve patient setup.

References

Barnsley, M. F. & Hurd, L. P. (1993) *Fractal Image Compression*, 1st ed., chap. 3, pp. 47–74. AK Peters Ltd.

Byhardt, R. W., Cox, J. D., Hornburg, A. & Liermann, G. (1978) Weekly localization films and detection of field placement errors. *Int. J. Radiat. Oncol. Biol. Phys.* **4**, 881–887

Ding, G. X., Shalev, S. & Gluchev, G. (1993) A $\rho-\theta$ technique for treatment verification in radiotherapy and its clinical applications. *Med. Phys.* **20**, 1135–1143

Goitein, M. (1975) Immobilization error: Some theoretical considerations. *Radiology* **117**, 407–412.

Lam, W. C., Partowmah, M., Lee, D. J., Whatam, M. D. & Lam, K. S. (1987) On-line measurement of field placement errors in external beam radiotherapy. *Br. J. Radiol.* **60**, 361–367

Marks, J. E., Haus, A. G., Sutton, H. G. & Griem, M. L. (1976) The value of frequent treatment verification films in reducing localization error in the irradiation of complex fields. *Cancer* **37**, 2755–2761.

McParland, B. J. (1993) Uncertainty of field placement errors in digital portal images. *Phys. Med. Biol.* **35**, 299–323.

Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., ter Haar Romeny, B., Zimmerman, J. B. & Zuiderveld, K. (1987) Adaptive histogram equalization and its variations. *Comput. Vis. Graph. Image Process.* **39**, 355–368

Press, W. H., Teukolsky, S. A., Vetterint, W. T. & Flannery, B. P. (1992) *Numerical Recipes in C The Art of Scientific Computing*, 2nd ed. , Cambridge University Press.

Rabinowitz, I., Broomberg, J., Goitein, M., McCathy, K. & Leong, J. (1985) Accuracy of radiation field alignment in clinical practice. *Int. J. Radiat. Oncol. Biol. Phys.* **11**, 1857–1867

Research Systems, Inc. (1995) *IDL Reference Guide Version 4.*

Rosenman, J., Roe, C. A., Cromartie, R., Muller, K. E. & Pizer, S. M. (1993) Portal film enhancement: Technique and clinical utility. *Int. J. Radiat. Oncol. Biol. Phys.* **25**, 333–338

Williams, J. R. & Thwaites, D. I. (1993) *Radiotherapy Physics in practice*, 1st ed., chap. 10, pp. 227–251. Oxford University Press.

Appendix A

The source to the main section of the program is contained in this appendix. The complete program also has a routine to load a Lumiscan™ image into memory (see appendix B) and the bitmaps files containing the graphics used for button labels (see appendix C).

<sandra.pro>

```
;=====================================================================
;
; Sandra -- a portal image processing system written in IDL
;
; Written by Jonathan Buzzard. (last update 12/08/96)
;
;    S - System for
;    A - Analysing
;    N - Normal
;    D - Displacements in
;    R - Radiotherapy
;    A - Alignment
;
;=====================================================================


;
; The next three routines take care of the events being dispatched by
; XMANAGER, performing all the necessary actions.
;

;
; Handle the events generated by the main application
;
PRO SandraEventHdlr, event

    WIDGET_CONTROL, GET_UVALUE=control, event.id

    IF (STRPOS(control, "LEFT") EQ 0) THEN BEGIN
        iWhich = 1
        sAction = STRMID(control, 4, STRLEN(control)-4)
        control = "ACTION"
    ENDIF
    IF (STRPOS(control, "RIGHT") EQ 0) THEN BEGIN
        iWhich = 2
        sAction = STRMID(control, 5, STRLEN(control)-5)
        control = "ACTION"
    ENDIF

    CASE control OF

        "ABOUT": BEGIN
```

```
    WIDGET_CONTROL, event.top, GET_UVALUE=State, /NO_COPY
    WIDGET_CONTROL, State.InfoText, SET_VALUE="System for Analysing"$
                +" Normal Displacements in Radiotherapy Alignment"$
                +"        Version 1.1 (09/08/1996)"
    WIDGET_CONTROL, event.top, SET_UVALUE=State, /NO_COPY
ENDCASE

"EXIT": WIDGET_CONTROL, event.top, /DESTROY

"ACTION" : CALL_PROCEDURE, sAction+"_EVENT", event.top, iWhich

"IMAGE_LEFT": BEGIN


    ; Marking control points on the images
    IF event.press NE 0 THEN RETURN

    WIDGET_CONTROL, event.top, GET_UVALUE=State, /NO_COPY

    ; if mouse press in the correct image add a control point
    IF (NOT (State.ControlPtCount AND 1)) THEN BEGIN
        WSET, State.LeftImgHdl
        PLOTS, event.x, event.y, /DEVICE, COLOR=!D.TABLE_SIZE-1, PSYM=6
        InfoTxt = STRING(FORMAT='("Fiducial Pair:",I2," Left Point:",I4,",",I4)',$
                    1+State.ControlPtCount/2, event.x, event.y)
        WIDGET_CONTROL, State.InfoText, SET_VALUE=InfoTxt
        WIDGET_CONTROL, State.LeftCtrlPts, GET_UVALUE=leftpts
        leftpts = [leftpts, event.x, event.y]
        State.ControlPtCount = State.ControlPtCount+1
        WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=leftpts
    ENDIF ELSE BEGIN
        WIDGET_CONTROL, State.InfoText, SET_VALUE="Click in the right-hand"$
                    +" image to select the corresponding point"
    ENDELSE

    WIDGET_CONTROL, event.top, SET_UVALUE=State, /NO_COPY
ENDCASE

"IMAGE_RIGHT": BEGIN

    ; Marking control points on the images
    IF event.press NE 0 THEN RETURN
    WIDGET_CONTROL, event.top, GET_UVALUE=State, /NO_COPY

    ; if mouse press in the correct image add a control point
    IF (State.ControlPtCount AND 1) THEN BEGIN
        WSET, State.RightImgHdl
        PLOTS, event.x, event.y, /DEVICE, COLOR=!D.TABLE_SIZE-1, PSYM=6
        InfoTxt = STRING(FORMAT='("Fiducial Pair:",I2," Right Point:",I4,",",I4)',$
                    1+State.ControlPtCount/2, event.x, event.y)
        WIDGET_CONTROL, State.InfoText, SET_VALUE=InfoTxt
        WIDGET_CONTROL, State.RightCtrlPts, GET_UVALUE=rightpts
        rightpts = [rightpts, event.x, event.y]
        State.ControlPtCount = State.ControlPtCount+1
        WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=rightpts
    ENDIF ELSE BEGIN
        WIDGET_CONTROL, State.InfoText, SET_VALUE="Click in the left-hand"$
                    +" image to select a primary fiducial point"
    ENDELSE

    WIDGET_CONTROL, event.top, SET_UVALUE=State, /NO_COPY
ENDCASE
```

```
        ELSE: CALL_PROCEDURE, control+"_EVENT", event.top

    ENDCASE
END

;
; Handle the events generated by the histogram dialog
;
PRO HistEventHdlr, event

    WIDGET_CONTROL, GET_UVALUE=control, event.id

    CASE control OF

        "CANCEL": WIDGET_CONTROL, event.top, /DESTROY

        "EQUALIZE" : BEGIN
            WIDGET_CONTROL, event.top, GET_UVALUE=State, /NO_COPY
            WIDGET_CONTROL, State.ImageStore, GET_UVALUE=iImage

            ; do the actual histogram equalization
            iImage = HIST_EQUAL(TEMPORARY(iImage), MINV=State.LowCut,$
                                MAXV=State.HighCut)
            WSET, State.ImageDraw
            TVSCL, iImage

            WIDGET_CONTROL, State.ImageStore, SET_UVALUE=iImage
            WIDGET_CONTROL, event.top, SET_UVALUE=State, /NO_COPY

            WIDGET_CONTROL, event.top, /DESTROY
        ENDCASE

        "SLIDE_LOW": BEGIN
            WIDGET_CONTROL, event.top, GET_UVALUE=State, /NO_COPY
            iNewCut = event.value
            ; stop low cutoff from becoming greater than high cutoff
            IF (iNewCut GE State.HighCut) THEN BEGIN
                iNewCut=State.HighCut-1
                WIDGET_CONTROL, event.id, SET_VALUE=State.HighCut-1
            ENDIF
            bTemp = State.HistImage(iNewCut,*)
            State.HistImage(iNewCut,0:127) = 200
            State.HistImage(State.LowCut,*)=State.LowCutStore
            State.LowCut = iNewCut
            State.LowCutStore = bTemp
            WSET, State.HistDraw
            TV, State.HistImage
            WIDGET_CONTROL, event.top, SET_UVALUE=State, /NO_COPY
        ENDCASE

        "SLIDE_HIGH": BEGIN
            WIDGET_CONTROL, event.top, GET_UVALUE=State, /NO_COPY
            iNewCut = event.value
            ; stop high cutoff from becoming less than low cutoff
            IF (iNewCut LE State.LowCut) THEN BEGIN
                iNewCut=State.LowCut+1
                WIDGET_CONTROL, event.id, SET_VALUE=State.LowCut+1
            ENDIF
            bTemp = State.HistImage(iNewCut,*)
            State.HistImage(iNewCut,0:127) = 200
            State.HistImage(State.HighCut,*)=State.HighCutStore
```

```
            State.HighCut = iNewCut
            State.HighCutStore = bTemp
            WSET, State.HistDraw
            TV, State.HistImage
            WIDGET_CONTROL, event.top, SET_UVALUE=State, /NO_COPY
        ENDCASE

    ENDCASE
END

;
; Handle the events generated by the Fader dialog
;
PRO FaderEventHdlr, event

    WIDGET_CONTROL, GET_UVALUE=control, event.id

    CASE control OF

        "DISMISS": WIDGET_CONTROL, event.top, /DESTROY

        "SLIDE_FADE" : BEGIN
            WIDGET_CONTROL, event.top, /HOURGLASS
            WIDGET_CONTROL, event.top, GET_UVALUE=State, /NO_COPY

            ; get the two images
            WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iRightImage, /NO_COPY
            WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iLeftImage, /NO_COPY

            ; combine the two images at the appropriate fading
            WSET, State.ImageDraw
            TVSCL, event.value*FIX(iRightImage(0:state.XSize-1,0:State.YSize-1))+$
                (8-event.value)*FIX(iLeftImage(0:state.XSize-1,0:State.YSize-1))

            ; store the two images back in there user values
            WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iRightImage, /NO_COPY
            WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iLeftImage, /NO_COPY

            WIDGET_CONTROL, event.top, SET_UVALUE=State, /NO_COPY
        ENDCASE
    ENDCASE

END

;
; Clean up when Sandra exits; ie. restore colour table
;
PRO CleanUpSandra, wSandraWindow

    ; Get the color table saved in the window's user value
    WIDGET_CONTROL, wSandraWindow, GET_UVALUE=SandraState
    TVLCT, SandraState.ColourTable

END

;
; calculate the affine transformation and apply it
;
PRO affine_event, child

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY
```

```
; need same number of points in each window and at least 2 pairs
IF (State.ControlPtCount LT 4) THEN BEGIN
    WIDGET_CONTROL, State.InfoText,$
        SET_VALUE="Need at least two fiducial points to calculate tranformation"
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY
    RETURN
ENDIF
IF (State.ControlPtCount AND 1) THEN BEGIN
    WIDGET_CONTROL, State.InfoText,$
        SET_VALUE="Need same number of fiducial points in each window"
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY
    RETURN
ENDIF

WIDGET_CONTROL, State.LeftCtrlPts, GET_UVALUE=leftpts, /NO_COPY
WIDGET_CONTROL, State.RightCtrlPts, GET_UVALUE=rightpts, /NO_COPY

; the transformation calculated is the one to bring the
; right hand image into alignment with the left and image.
; IT DOES NOT represent the changes to be made on the treatment or
; simulator machines to bring the patient into the correct position.
A = DBLARR(4,State.ControlPtCount)
FOR iLoop=0,State.ControlPtCount-1,2 DO BEGIN
    a(0,iLoop) = DOUBLE(leftpts(2+iLoop))
    a(1,iLoop) = -1.0D*DOUBLE(leftpts(3+iLoop))
    a(2,iLoop) = 1.0D
    a(3,iLoop) = 0.0D
    a(0,iLoop+1) = DOUBLE(leftpts(3+iLoop))
    a(1,iLoop+1) = DOUBLE(leftpts(2+iLoop))
    a(2,iLoop+1) = 0.0D
    a(3,iLoop+1) = 1.0D
ENDFOR

B = DBLARR(State.ControlPtCount)
FOR iLoop=0, State.ControlPtCount-1 DO BEGIN
    b(iLoop) = DOUBLE(rightpts(2+iLoop))
ENDFOR

SVDC, A, w, u, v, /DOUBLE
n = N_ELEMENTS(w)
wp = DBLARR(n, n)
FOR iLoop=0,n-1 DO $
    IF (ABS(w(iLoop)) GE 1.0D-7) THEN wp(iloop, iloop) = 1.0D/w(iloop)

x = v ## wp ## TRANSPOSE(u) ## b

; decompose the transformation into its seperate components.
; ONLY the rotation represents the change needed to patient
; setup. If you know the magnification of the two images you
; can work out the vertical movement needed to the table.
; If you know the coordinates of the beam centre (isocentre),
; and the pixel size, you can workout the lateral and
; longditude movements of the table to correct patient setup.
radeg = -57.29577951D
theta = ATAN(x(1),x(0))
AffText1 = STRING(FORMAT='("Rotation: ",F8.4,"    Scale: ",F8.4)',$
                  radeg*theta,SIN(theta)/x(1))
AffText2 = STRING(FORMAT='("    Translation: x=",F6.2,"  y=",F6.2)',$
                  x(2), x(3))
WIDGET_CONTROL, State.InfoText, SET_VALUE=AffText1+Afftext2
```

```
    ; generate the matrix holding the transformation polynomial
    p = [x(2), -x(1), x(0), 0.0D]
    q = [x(3), x(0), x(1), 0.0D]

    ; transform the image using cubic interpolation
    WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iImage, /NO_COPY
    iImage = POLY_2D(TEMPORARY(iImage), p, q, 2, MISSING=0)
    WSET, State.RightImgHdl
    TVSCL, iImage
    WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iImage, /NO_COPY

    ; transform the control points (first need to invert matrix)
    invtran = INVERT([[x(0),-x(1)], [x(1),x(0)]], /DOUBLE)
    FOR iLoop=0,State.ControlPtCount-1,2 DO BEGIN
        xx = rightpts(2+iLoop)
        yy = rightpts(3+iLoop)
        rightpts(2+iLoop) = xx*invtran(0)+yy*invtran(1)-x(2)
        rightpts(3+iLoop) = xx*invtran(2)+yy*invtran(3)-x(3)
    ENDFOR

    ; redraw the control points (add left points for comparison)
    FOR iLoop=0,State.ControlPtCount-1,2 DO BEGIN
        PLOTS, rightpts(2+iLoop), rightpts(3+iLoop), /DEVICE,$
            COLOR=!D.TABLE_SIZE-1, PSYM=6
        PLOTS, leftpts(2+iLoop), leftpts(3+iLoop), /DEVICE,$
            COLOR=!D.TABLE_SIZE-1, PSYM=6
    ENDFOR

    WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=rightpts, /NO_COPY
    WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=leftpts, /NO_COPY
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

END

;
; Histogram equalise an image
;
PRO histequ_event, child, iWhich

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY

    IF (iWhich EQ 1) THEN BEGIN
        hImageDraw = State.LeftImgHdl
        hImageStore = State.LeftImgStore
    ENDIF ELSE BEGIN
        hImageDraw = State.RightImgHdl
        hImageStore = State.RightImgStore
    ENDELSE

    WIDGET_CONTROL, hImageStore, GET_UVALUE=iImage, /NO_COPY

    ; generate the histogram as an image
    iHist = HISTOGRAM(iImage, BINSIZE=1)
    fFact = 127.0/FLOAT(MAX(iHist(1:254)))
    iHistImg = BYTARR(256,128)
    FOR iLoop=1,254 DO BEGIN
        iHistImg(iLoop,0:iHist(iLoop)*fFact)=255
    ENDFOR

    ; put the information stored in user values back
    WIDGET_CONTROL, hImageStore, SET_UVALUE=iImage, /NO_COPY
```

```
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

    ; create dialog box for histogram equalization
    wHistWindow = WIDGET_BASE(TITLE="Histogram Equalization")
    wHistBase = WIDGET_BASE(wHistWindow, /COLUMN)
    wHistDraw = WIDGET_DRAW(wHistBase, XSIZE=256, YSIZE=128, RETAIN=2)
    wHistLowLabel = WIDGET_LABEL(wHistBase, VALUE="Low cut-off point",$
                                /ALIGN_LEFT)
    wHistLow = WIDGET_SLIDER(wHistBase, /SUPPRESS_VALUE, MINIMUM=0,$
                            MAXIMUM=255, VALUE=0, UVALUE="SLIDE_LOW")
    wHistHighLabel = WIDGET_LABEL(wHistBase, VALUE="High cut-off point",$
                                /ALIGN_LEFT)
    wHistHigh = WIDGET_SLIDER(wHistBase, /SUPPRESS_VALUE, MINIMUM=0,$
                            MAXIMUM=255, VALUE=255, UVALUE="SLIDE_HIGH")
    wHistButtonBase = WIDGET_BASE(wHistBase, /ROW)
    wHistEqual = WIDGET_BUTTON(wHistButtonBase, VALUE="Equalize",$
                                UVALUE="EQUALIZE")
    wHistCancel = WIDGET_BUTTON(wHistButtonBase, VALUE="Cancel",$
                                UVALUE="CANCEL")

    ; realize the histogram dialog box
    WIDGET_CONTROL, /REALIZE, wHistWindow

    ; display the histogram
    WIDGET_CONTROL, wHistDraw, GET_VALUE=hHistDraw
    WSET, hHistDraw
    TV, iHistImg

    ; store the histogram state inforamtion in user value of the dialog
    HistState = CREATE_STRUCT('LowCut', 0)
    HistState = CREATE_STRUCT(HistState, 'LowCutStore', BYTARR(128))
    HistState = CREATE_STRUCT(HistState, 'HighCut', 255)
    HistState = CREATE_STRUCT(HistState, 'HighCutStore', BYTARR(128))
    HistState = CREATE_STRUCT(HistState, 'HistImage', iHistImg)
    HistState = CREATE_STRUCT(HistState, 'HistDraw', hHistDraw)
    HistState = CREATE_STRUCT(HistState, 'ImageStore', hImageStore)
    HistState = CREATE_STRUCT(HistState, 'ImageDraw', hImageDraw)
    WIDGET_CONTROL, wHistWindow, SET_UVALUE=HistState, /NO_COPY

    ; register dialog box with XMANAGER as modal
    XMANAGER, "Histogram", wHistWindow, EVENT_HANDLER="HistEventHdlr",$
            /MODAL

END

;
; Pop up a dialog box to fade between the two images
;
PRO fade_event, child

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY

    WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iRightImage, /NO_COPY
    WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iLeftImage, /NO_COPY

    ; find the sizes of the two images
    rightSize=SIZE(iRightImage)
    leftSize=SIZE(iLeftImage)

    ; select the smaller x and y sizes for image addition
    IF (rightsize(1) LE leftsize(1)) THEN xx=rightsize(1) ELSE xx=leftsize(1)
```

```
    IF (rightsize(2) LE leftsize(2)) THEN yy=rightsize(2) ELSE yy=leftsize(2)

    ; create dialog box for fading between two images
    wFadeWindow = WIDGET_BASE(TITLE="Fader")
    wFadeBase = WIDGET_BASE(wFadeWindow, /COLUMN)
    wFadeDraw = WIDGET_DRAW(wFadeBase, XSIZE=xx, YSIZE=yy,$
                            X_SCROLL_SIZE=400, Y_SCROLL_SIZE=400,$
                            RETAIN=2, /SCROLL)
    wFadeControlBase = WIDGET_BASE(wFadeBase, /ROW)
    wFadeSlide = WIDGET_SLIDER(wFadeControlBase, MINIMUM=0,$
                            MAXIMUM=8, VALUE=4, UVALUE="SLIDE_FADE")
    wFadeDismiss = WIDGET_BUTTON(wFadeControlBase, VALUE="Dismiss",$
                                UVALUE="DISMISS")

    ; realize the fader dialog box
    WIDGET_CONTROL, /REALIZE, wFadeWindow

    ; display the two images equally mixed
    WIDGET_CONTROL, wFadeDraw, GET_VALUE=hFadeDraw
    WSET, hFadeDraw
    TVSCL, FIX(iRightImage)+FIX(iLeftImage(0:xx-1,0:yy-1))

    ; store the fader state inforamtion in user value of the dialog
    FadeState = CREATE_STRUCT('XSize', xx)
    FadeState = CREATE_STRUCT(FadeState, 'YSize', yy)
    FadeState = CREATE_STRUCT(FadeState, 'LeftImgStore', State.LeftImgStore)
    FadeState = CREATE_STRUCT(FadeState, 'RightImgStore', State.RightImgStore)
    FadeState = CREATE_STRUCT(FadeState, 'ImageDraw', hFadeDraw)
    WIDGET_CONTROL, wFadeWindow, SET_UVALUE=FadeState, /NO_COPY

    ; restore the state inforamtion and images in their widget user values
    WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iRightImage, /NO_COPY
    WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iLeftImage, /NO_COPY
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

    ; register dialog box with XMANAGER as modal
    XMANAGER, "Fader", wFadeWindow, EVENT_HANDLER="FaderEventHdlr",$
            /MODAL

END

;
; Clear the control points from screen and memory
;
PRO clear_pts_event, child

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY

    ; redraw the left image
    WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iImage, /NO_COPY
    WSET, State.LeftImgHdl
    TVSCL, iImage
    WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iImage, /NO_COPY

    ; redraw the right image
    WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iImage, /NO_COPY
    WSET, State.RightImgHdl
    TVSCL, iImage
    WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iImage, /NO_COPY

    ; reset the control points
```

```
    WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=[-1, -1]
    WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=[-1, -1]
    State.ControlPtCount = 0

    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

END

;
; Load an image, resetting control points
;
PRO load_event, child, iWhich

    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY
    file = PICKFILE(FILE="", GROUP=child, FILTER="*.img", /READ)

    IF (file EQ "") THEN BEGIN
        WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY
        RETURN
    ENDIF

    WIDGET_CONTROL, child, /HOURGLASS

    ; redraw the other image as necessary
    IF (iWhich EQ 1) THEN BEGIN
        IF (State.ControlPtCount GT 0) THEN BEGIN
            WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.RightImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
        lImgWdg = State.LeftImgWdg
        hImageDraw =State.LeftImgHdl
        hImageStore = State.LeftImgStore
    ENDIF ELSE BEGIN
        IF (State.ControlPtCount GT 0) THEN BEGIN
            WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.LeftImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
        lImgWdg = State.RightImgWdg
        hImageDraw =State.RightImgHdl
        hImageStore = State.RightImgStore
    ENDELSE

    ; reset the control points
    WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=[-1, -1]
    WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=[-1, -1]
    State.ControlPtCount = 0

    read_lum, file, iImage
    iImage(WHERE(iImage GT 4094)) = 4094
    iImage = BYTSCL(TEMPORARY(iImage))

    iDim = SIZE(iImage)
    WIDGET_CONTROL, lImgWdg, XSIZE=iDim(1), YSIZE=iDim(2)

    ; store the image size
    IF (iWhich EQ 1) THEN BEGIN
        State.LeftImgX = iDim(1)
        State.LeftImgY = iDim(2)
```

```
    ENDIF ELSE BEGIN
        State.RightImgX = iDim(1)
        State.RightImgY = iDim(2)
    ENDELSE

    WSET, hImageDraw
    TVSCL, iImage

    WIDGET_CONTROL, hImageStore, SET_UVALUE=iImage, /NO_COPY
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

END

;
; Routine to rotate an image counter clockwise 90 degrees
;
PRO rotccw_event, child, iWhich

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY

    IF (iWhich EQ 1) THEN BEGIN
        lImgWdg = State.LeftImgWdg
        hImageDraw = State.LeftImgHdl
        hImageStore = State.LeftImgStore
        iTemp = State.LeftImgX
        State.LeftImgX = State.LeftImgY
        State.LeftImgY = iTemp
        WIDGET_CONTROL, lImgWdg, XSIZE=State.LeftImgX, YSIZE=State.LeftImgY
        IF (State.ControlPtCount GT 1) THEN BEGIN
            WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.RightImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDIF ELSE BEGIN
        lImgWdg = State.RightImgWdg
        hImageDraw = State.RightImgHdl
        hImageStore = State.RightImgStore
        iTemp = State.RightImgX
        State.RightImgX = State.RightImgY
        State.RightImgY = iTemp
        WIDGET_CONTROL, lImgWdg, XSIZE=State.RightImgX, YSIZE=State.RightImgY
        IF (State.ControlPtCount GT 0) THEN BEGIN
            WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.LeftImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDELSE

    WIDGET_CONTROL, hImageStore, GET_UVALUE=iImage, /NO_COPY

    iImage = ROTATE(TEMPORARY(iImage), 1)

    WSET, hImageDraw
    TVSCL, iImage

    ; reset the control points
    WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=[-1, -1]
    WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=[-1, -1]
    State.ControlPtCount = 0
```

```
    WIDGET_CONTROL, hImageStore, SET_UVALUE=iImage, /NO_COPY
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

END

;
; Routine to rotate an image clockwise 90 degrees
;
PRO rotcw_event, child, iWhich

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY

    IF (iWhich EQ 1) THEN BEGIN
        lImgWdg = State.LeftImgWdg
        hImageDraw = State.LeftImgHdl
        hImageStore = State.LeftImgStore
        iTemp = State.LeftImgX
        State.LeftImgX = State.LeftImgY
        State.LeftImgY = iTemp
        WIDGET_CONTROL, lImgWdg, XSIZE=State.LeftImgX, YSIZE=State.LeftImgY
        IF (State.ControlPtCount GT 1) THEN BEGIN
            WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.RightImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDIF ELSE BEGIN
        lImgWdg = State.RightImgWdg
        hImageDraw = State.RightImgHdl
        hImageStore = State.RightImgStore
        iTemp = State.RightImgX
        State.RightImgX = State.RightImgY
        State.RightImgY = iTemp
        WIDGET_CONTROL, lImgWdg, XSIZE=State.RightImgX, YSIZE=State.RightImgY
        IF (State.ControlPtCount GT 0) THEN BEGIN
            WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.LeftImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDELSE

    WIDGET_CONTROL, hImageStore, GET_UVALUE=iImage, /NO_COPY

    iImage = ROTATE(TEMPORARY(iImage), 3)
    Dim = SIZE(iImage)
    WIDGET_CONTROL, lImgWdg, XSIZE=Dim(1), YSIZE=Dim(2)
    WSET, hImageDraw
    TVSCL, iImage

    ; reset the control points
    WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=[-1, -1]
    WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=[-1, -1]
    State.ControlPtCount = 0

    WIDGET_CONTROL, hImageStore, SET_UVALUE=iImage, /NO_COPY
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

END
```

```
;
; Routine to flip the portal image horizontally
;
PRO fliph_event, child, iWhich

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY

    IF (iWhich EQ 1) THEN BEGIN
        hImageDraw = State.LeftImgHdl
        hImageStore = State.LeftImgStore
        IF (State.ControlPtCount GT 1) THEN BEGIN
            WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.RightImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDIF ELSE BEGIN
        hImageDraw = State.RightImgHdl
        hImageStore = State.RightImgStore
        IF (State.ControlPtCount GT 0) THEN BEGIN
            WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.LeftImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDELSE

    WIDGET_CONTROL, hImageStore, GET_UVALUE=iImage, /NO_COPY

    iImage = REVERSE(TEMPORARY(iImage),1)
    WSET, hImageDraw
    TVSCL, iImage

    ; reset the control points
    WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=[-1, -1]
    WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=[-1, -1]
    State.ControlPtCount = 0

    WIDGET_CONTROL, hImageStore, SET_UVALUE=iImage, /NO_COPY
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

END

;
; Routine to flip the portal image vertically
;
PRO flipv_event, child, iWhich

    WIDGET_CONTROL, child, /HOURGLASS
    WIDGET_CONTROL, child, GET_UVALUE=State, /NO_COPY

    IF (iWhich EQ 1) THEN BEGIN
        hImageDraw = State.LeftImgHdl
        hImageStore = State.LeftImgStore
        IF (State.ControlPtCount GT 1) THEN BEGIN
            WIDGET_CONTROL, State.RightImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.RightImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.RightImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDIF ELSE BEGIN
```

```
        hImageDraw = State.RightImgHdl
        hImageStore = State.RightImgStore
        IF (State.ControlPtCount GT 0) THEN BEGIN
            WIDGET_CONTROL, State.LeftImgStore, GET_UVALUE=iImage, /NO_COPY
            WSET, State.LeftImgHdl
            TVSCL, iImage
            WIDGET_CONTROL, State.LeftImgStore, SET_UVALUE=iImage, /NO_COPY
        ENDIF
    ENDELSE

    WIDGET_CONTROL, hImageStore, GET_UVALUE=iImage, /NO_COPY

    iImage = REVERSE(TEMPORARY(iImage),2)
    WSET, hImageDraw
    TVSCL, iImage

    ; reset the control points
    WIDGET_CONTROL, State.LeftCtrlPts, SET_UVALUE=[-1, -1]
    WIDGET_CONTROL, State.RightCtrlPts, SET_UVALUE=[-1, -1]
    State.ControlPtCount = 0

    WIDGET_CONTROL, hImageStore, SET_UVALUE=iImage, /NO_COPY
    WIDGET_CONTROL, child, SET_UVALUE=State, /NO_COPY

END

;
; Create the widgets that make up the point and click interface to SANDRA,
; and then register them with XMANAGER
;

PRO sandra

    ; get the current colour vectors to restore when application is exited.
    TVLCT, savedR, savedG, savedB, /GET

    ; build colour table from colour vectors
    colourTable = [[savedR],[savedG],[savedB]]

    ; create the top level window for the application
    wSandraWindow = WIDGET_BASE(TITLE="Sandra", MBAR=wMenuBar)

    wFileMenu = WIDGET_BUTTON(wMenuBar, VALUE="File", /MENU)
    wOpen1Item = WIDGET_BUTTON(wFileMenu, VALUE="Open Left Image...",$
                               UVALUE="LEFTLOAD")
    wOpen2Item = WIDGET_BUTTON(wFileMenu, VALUE="Open Right Image...",$
                               UVALUE="RIGHTLOAD")
    wExitItem = WIDGET_BUTTON(wFileMenu, VALUE="Exit", UVALUE="EXIT")


    wEditMenu = WIDGET_BUTTON(wMenuBar, VALUE="Edit", /MENU)
    wClearPtItem = WIDGET_BUTTON(wEditMenu, VALUE="Clear Points",$
                               UVALUE="CLEAR_PTS")
    wAffineItem = WIDGET_BUTTON(wEditMenu, VALUE="Affine Transform",$
                               UVALUE="AFFINE")
    wFadeItem = WIDGET_BUTTON(wEditMenu, VALUE="Fade Images...",$
                               UVALUE="FADE")

    wHelpMenu = WIDGET_BUTTON(wMenuBar, VALUE="Help", /HELP, /MENU)
    wHelpItem = WIDGET_BUTTON(wHelpMenu, VALUE="About", UVALUE="ABOUT")

    wSandraBase = WIDGET_BASE(wSandraWindow, /COLUMN)
```

```
wDrawBase = WIDGET_BASE(wSandraBase, /ROW)
wLeftBase = WIDGET_BASE(wDrawBase, /COLUMN)
wRightBase = WIDGET_BASE(wDrawBase, /COLUMN)

; Determine hardware display size.
DEVICE, GET_SCREEN_SIZE = screenSize
IF (screenSize(0) GT 640) THEN iScroll = 370 ELSE iScroll = 285

; create the two draw widgets to hold the images
wLeftDraw = WIDGET_DRAW(wLeftBase, XSIZE=512, YSIZE=512,$
                        X_SCROLL_SIZE=iScroll, Y_SCROLL_SIZE=iScroll,$
                        RETAIN=2, /BUTTON_EVENTS, /SCROLL,$
                        UVALUE = "IMAGE_LEFT")
wRightDraw = WIDGET_DRAW(wRightBase, XSIZE=512, YSIZE=512,$
                        X_SCROLL_SIZE=iScroll, Y_SCROLL_SIZE=iScroll,$
                        RETAIN=2, /BUTTON_EVENTS, /SCROLL,$
                        UVALUE= "IMAGE_RIGHT")

; load the bitmaps for the buttons
READ_X11_BITMAP, "rotccw.xbm", bRotateCCW
READ_X11_BITMAP, "rotcw.xbm", bRotateCW
READ_X11_BITMAP, "fliph.xbm", bFlipH
READ_X11_BITMAP, "flipv.xbm", bFlipV
READ_X11_BITMAP, "histoequ.xbm", bHistoEqu

; create the left image buttons
wLeftButtonBase = WIDGET_BASE(wLeftBase, /ROW)
wLeftRotateCCW = WIDGET_BUTTON(wLeftButtonBase, VALUE=bRotateCCW,$
                UVALUE="LEFTROTCCW")
wLeftRotateCW = WIDGET_BUTTON(wLeftButtonBase, VALUE=bRotateCW,$
                UVALUE="LEFTROTCW")
wLeftFlipH = WIDGET_BUTTON(wLeftButtonBase, VALUE=bFlipH,$
                UVALUE="LEFTFLIPH")
wLeftFlipV = WIDGET_BUTTON(wLeftButtonBase, VALUE=bFlipV,$
                UVALUE="LEFTFLIPV")
wLeftHistEqu = WIDGET_BUTTON(wLeftButtonBase, VALUE=bHistoEqu,$
                UVALUE="LEFTHISTEQU")

; now create the right image buttons
wRightButtonBase = WIDGET_BASE(wRightBase, /ROW)
wRightRotateCCW = WIDGET_BUTTON(wRightButtonBase, VALUE=bRotateCCW,$
                UVALUE="RIGHTROTCCW")
wRightRotateCW = WIDGET_BUTTON(wRightButtonBase, VALUE=bRotateCW,$
                UVALUE="RIGHTROTCW")
wRightFlipH = WIDGET_BUTTON(wRightButtonBase, VALUE=bFlipH,$
                UVALUE="RIGHTFLIPH")
wRightFlipV = WIDGET_BUTTON(wRightButtonBase, VALUE=bFlipV,$
                UVALUE="RIGHTFLIPV")
wRightHistEqu = WIDGET_BUTTON(wRightButtonBase, VALUE=bHistoEqu,$
                UVALUE="RIGHTHISTEQU")

; Create a text widget to display information
wInfoText = WIDGET_TEXT(wSandraBase, XSIZE=32, YSIZE = 1,$
          VALUE=STRING(REPLICATE(32B,32)) )


; realize the window
WIDGET_CONTROL, /REALIZE, wSandraWindow

WIDGET_CONTROL, wLeftDraw, GET_VALUE=hLeftDraw
WIDGET_CONTROL, wRightDraw, GET_VALUE=hRightDraw
```

```
    ; save the previous colour table in the user value to restore on exit
    SandraState = CREATE_STRUCT('ColourTable', colourTable)

    SandraState = CREATE_STRUCT(SandraState, 'LeftImgHdl', hLeftDraw)
    SandraState = CREATE_STRUCT(SandraState, 'LeftImgWdg', wLeftDraw)
    SandraState = CREATE_STRUCT(SandraState, 'LeftImgStore', wLeftBase)
    SandraState = CREATE_STRUCT(SandraState, 'LeftImgX',512)
    SandraState = CREATE_STRUCT(SandraState, 'LeftImgY',512)
    SandraState = CREATE_STRUCT(SandraState, 'RightImgHdl', hRightDraw)
    SandraState = CREATE_STRUCT(SandraState, 'RightImgWdg', wRightDraw)
    SandraState = CREATE_STRUCT(SandraState, 'RightImgStore', wRightBase)
    SandraState = CREATE_STRUCT(SandraState, 'RightImgX',512)
    SandraState = CREATE_STRUCT(SandraState, 'RightImgY',512)
    SandraState = CREATE_STRUCT(SandraState, 'RightCtrlPts', wRightButtonBase)
    SandraState = CREATE_STRUCT(SandraState, 'LeftCtrlPts', wLeftButtonBase)
    SandraState = CREATE_STRUCT(SandraState, 'ControlPtCount', 0)
    SandraState = CREATE_STRUCT(SandraState, 'InfoText', wInfoText)

    WIDGET_CONTROL, wLeftButtonBase, SET_UVALUE=[-1, -1]
    WIDGET_CONTROL, wRightButtonBase, SET_UVALUE=[-1, -1]
    LOADCT, 1  ; 0-greyscale 1-blue/white 3-hotbody

    ; load two blank 512x512 images into draw widgets this means we
    ; don't have to track whether images are loaded.
    WIDGET_CONTROL, wLeftBase, SET_UVALUE=BYTARR(512,512)
    WIDGET_CONTROL, wRightBase, SET_UVALUE=BYTARR(512,512)

    WIDGET_CONTROL, wSandraWindow, SET_UVALUE=SandraState, /NO_COPY

    ; register the application with XMANAGER
    XMANAGER, "Sandra", wSandraWindow, EVENT_HANDLER="SandraEventHdlr",$
            CLEANUP="CleanUpSandra"

END
```

Appendix B

Because a routine to load a Lumiscan™ image into IDL would be a useful routine in other programs, it was written as a separate procedure. The comments contained in the header can be extracted using the `DOC_LIBRARY` routine to provide documentation on the how to use the routine consistent with the IDL library routines.

<read_lum.pro>

```
PRO READ_LUM, cFile, iImage
;+
; NAME:
;       READ_LUM
;
; PURPOSE:
;       Read the contents of a LUMISCAN(tm) format image file and return
;       the image in the form of an IDL variable.
;
; CATEGORY:
;       Input/Output.
;
; CALLING SEQUENCE:
;       READ_LUMI, cFile, image
;
; INPUTS:
;       file:   Scalar string giving the name of the LUMISCAN file.
;
; OUTPUTS:
;       image:  The 2D byte array to contain the image.
;
; SIDE EFFECTS:
;       None.
;
; EXAMPLE:
;       To open and read the LUMISCAN image file named "foo.img" in the
;       current directory, store the image in the variable IMAGE1 enter:
;
;               READ_LUM, "foo.img", image1
;
; MODIFICATION HISTORY:
;       Written May, 1996, Jonathan Buzzard.
;-

    ON_IOERROR, BAD_IO
    ON_ERROR, 1

    OPENR, unit, cFile, /GET_LUN, /BLOCK
```

```
    iWidth = 0
    iHeight = 0
    POINT_LUN, unit, 806

; test for the machines endian'es and load with byte swapping if big endian
; note: it should return 1 for a little endian machine and 0 for a big endian

    IF (BYTE(1, 0, 2))(0) EQ 1B THEN BEGIN

        READU, unit, iWidth
        READU, unit, iHeight
        POINT_LUN, unit, 2048
        iImage = INTARR(iWidth, iHeight)
        READU, unit, iImage

    ENDIF ELSE BEGIN

        READU, unit, width
        READU, unit, height
        BYTEORDER, iWidth, iHeight
        POINT_LUN, unit, 2048
        iImage = INTARR(iWidth, iHeight)
        READU, unit, iImage
        BYTEORDER, iImage

    ENDELSE

    FREE_LUN, unit
    RETURN

BAD_IO: MESSAGE, 'Error occured accessing Lumiscan file : ' + cFile

END
```

Appendix C

The five buttons below each image each have a bitmap label to convey their function. These bitmaps are stored in the X11 Bitmap format, which consists of small fragments of C code, providing an array of bits with symbolic constants giving the width and height. These are read in using the `read_x11_bitmap` function from the IDL library before being used as the values for the button widgets. The five bitmap files used for button labels are shown below.

<rotccw.xbm>

```
#define rotateCCW_width 16
#define rotateCCW_height 16
static unsigned char rotateCCW_bits[] = {
   0x00, 0x00, 0x0c, 0x00, 0xec, 0x03, 0xfc, 0x0f, 0x3c, 0x1c, 0xfc, 0x18,
   0xfc, 0x30, 0x00, 0x30, 0x00, 0x30, 0x00, 0x30, 0x06, 0x30, 0x06, 0x18,
   0x1c, 0x1c, 0xf8, 0x0f, 0xe0, 0x03, 0x00, 0x00};
```

<rotcw.xbm>

```
#define rotateCW_width 16
#define rotateCW_height 16
static unsigned char rotateCW_bits[] = {
   0x00, 0x00, 0x00, 0x18, 0xe0, 0x1b, 0xf8, 0x1f, 0x1c, 0x1e, 0x8c, 0x1f,
   0x86, 0x1f, 0x06, 0x00, 0x06, 0x00, 0x06, 0x00, 0x06, 0x30, 0x0c, 0x30,
   0x1c, 0x1c, 0xf8, 0x0f, 0xe0, 0x03, 0x00, 0x00};
```

<fliph.xbm>

```
#define flipH_width 16
#define flipH_height 16
static unsigned char flipH_bits[] = {
   0x90, 0x00, 0x98, 0x00, 0x9c, 0x00, 0xfe, 0x1f, 0xfe, 0x1f, 0x9c, 0x00,
   0x98, 0x00, 0x90, 0x04, 0x80, 0x0c, 0x80, 0x1c, 0xfc, 0x3f, 0xfc, 0x3f,
   0x80, 0x1c, 0x80, 0x0c, 0x80, 0x04, 0x00, 0x00};
```

<flipv.xbm>

```
#define flipV_width 16
#define flipV_height 16
static unsigned char flipV_bits[] = {
   0x00, 0x00, 0x00, 0x0c, 0x18, 0x1e, 0x18, 0x3f, 0x98, 0x7f, 0x18, 0x0c,
   0x18, 0x0c, 0xff, 0x7f, 0x18, 0x0c, 0x18, 0x0c, 0xff, 0x0c, 0x7e, 0x0c,
   0x3c, 0x0c, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00};
```

<histoequ.xbm>

```
#define histEqu_width 16
#define histEqu_height 16
static unsigned char histEqu_bits[] = {
   0x00, 0x00, 0x00, 0x01, 0x00, 0x03, 0x80, 0x03, 0x80, 0x03, 0xc0, 0x07,
   0xc0, 0x07, 0xc8, 0x0f, 0xd8, 0x0f, 0xd8, 0x0f, 0xfc, 0x1f, 0xfc, 0x1f,
   0xfc, 0x3f, 0xff, 0x7f, 0x00, 0x00, 0x00, 0x00};
```

Appendix D

The translations and rotations produced by the affine transformation do not correspond to those used on a treatment machine. The treatment and simulator machines use a frame of reference in which the operations of rotation and translation commute (i.e. it does not matter in which order they are carried out). Additionally the centre of rotation for the treatment and simulator machines is the isocentre. A method is required to convert the affine transformation into the coordinate system used by the treatment machines.

Take two images, image one being the simulator image and image two the portal image, and a transformation (r, θ, x_t, y_t) that maps pixels in the version of image two registered with image one, onto corresponding pixels in image two (i.e. the transformation passed to `POLY_2D`). If (x_c, y_c) is the position of the centre of rotation (assumed to be the same in both images), and image two has an initial magnification m and the pixel size of both images is p (assumed to be square), then the position of the centre of rotation before the transformation was applied (x_a, y_a) is given by

$$\begin{aligned} x_a &= \frac{x_c r \cos\theta + y_c r \sin\theta - x_t \cos\theta - y_t \sin\theta}{\cos^2\theta - \sin^2\theta} \\ y_a &= \frac{y_c r \cos\theta - x_c r \sin\theta - y_t \cos\theta + x_t \sin\theta}{\cos^2\theta - \sin^2\theta} \end{aligned}$$

The distance between the two points (x_c, y_c) and (x_a, y_a) using Pythagoras' theorem is

$$h = \sqrt{(x_a - x_c)^2 + (y_a - y_c)^2}$$

and the angle is given by

$$\tan \phi = \frac{y_a - y_c}{x_a - x_c}$$

The longitudinal movement of the table (i.e. along its length) is given by

$$\frac{p}{m} \times h \cos\left(\frac{\pi}{2} - \theta - \phi\right)$$

and the lateral movement of the table is given by

$$\frac{p}{m} \times h \sin\left(\frac{\pi}{2} - \theta - \phi\right)$$

These movements will have the same units as the pixel size p. The rotation is the same in both systems, and knowing the table height and magnification of image one, working out the vertical table movement is trivial.