Tina Memo No. 2004-013
Internal

Incorportating Optical Flow into Tinatool.

J.L.Barron.

Last updated
20 / 1 / 2005

Imaging Science and Biomedical Engineering Division,
Medical School, University of Manchester,
Stopford Building, Oxford Road,
Manchester, M13 9PT.

Project	OSMIA: Open Source Medical Image Analysis EU 5^{th} Framework Programme
Project Number	IST-2001-34512
Deliverable Numbers	D4.3, D4.4ii, D4.5ii
Report Title	The Integration of Optical Flow into Tinatool
Prepared by	Prof. John Barron Dept. of Computer Science Middlesex College MC379 The University of Western Ontario London, Ontario, Canada, N6A 5B7 barron@csd.uwo.ca www.csd.uwo.ca/faculty/barron
Date	September 29^{th}, 2003
Technical Report	UWO-CSD-TR-601-2003

1 Introduction to Tinatool

Tinatool is a software package that provides a platform for the development of Computer Vision software [2]. It allows the user to integrate any code from existing TINA modules with his or her own code. Tinatool provides a hierarchy of image processing, display and interactive manipulation modules specifically designed for the recovery and representation of the 2D and 3D geometrical primitives required for the development and evaluation of Computer Vision systems. The software environment is X-windows based (running on, for example, a Sun Sparc Solaris platform or a PC Linux platform) with tinatool. Tinatool code is open source and is implemented in C. We use version 4.0.2 for the work described here, but note that this code has been fully integrated into NeatVision [10] using version 5.0 with only minimal changes (required by Microsoft C++).

We have implemented 2D and 3D Lucas and Kanade [4] and Horn and Schunck [3] optical flow algorithms in Tinatool. The 3D algorithms are simple extensions of the 2D algorithms as described in [1]. We tested our 2D algorithms using image sequences from the anonymous ftp site, ftp.csd.uwo.ca[1], as used in Barron et al. [5]. For the versions of the 3D algorithms, we use a gated MRI heart sequence (made at Robarts Research Institute, UWO). This data consists of 20 volumes of volumetric data for 1 heart beat, which each 3D volume dataset consisting of $256 \times 256 \times 31$ voxels (unsigned shorts) in the range $[0 - 4095]$ (12 bits). The resolution is 1.25mm in the X and Y dimensions and 5.0mm in the Z axis. The slices are axial images (head to feet). As such, the 3D optical flow was used to measure 3D volumetric optical flow for this data (the local 3D heart motion) and this project fits nicely into the OSMIA (Open Source Medical Image Analysis) project.

2 Optical Flow in Tinatool

Figure 1 shows the initial tinatool window, obtained by typing tinatool. (It is recommended that one uses a call script generated by running **tinatool -s** the first time one runs tinatool. Then place all the windows as you like, press any tinatool button as needed and then press the **close** button (this may not be necessary) on the **tinatool** window and finally end the tinatool session (press `^C` in the X window where **tinatool** was launched from or press the **Exit tinatool** buttons on either of the 2D and 3D flowtool windows. This generates a call script named **tinatool.cls**). Then type **tinatool -r** and the **tinatool.cls** file just created will place all the windows as you last left them with all the operations performed for the buttons pressed. If one wishes one can use **tinatool -f 2DtinatoolSUN** or **tinatool -f 2DtinatoolPC** for the 2D setups or **tinatool -f 3DtinatoolSUN** or **tinatool -f 3DtinatoolPC** for the 3D setups to use my previously saved **.cls** files, **2DtinatoolSUN.cls**, **3DtinatoolSUN.cls**, **2DtinatoolPC.cls** or **3DtinatoolPC.cls**. Different call scripts were used because the SUN had a 21 inch monitor and the PC (a laptop) had a 14.1 inch screen. (Note that there is no **.cls** at the end of the call script filenames when using the **-f** option.) Now 3 buttons for **2D flow**, **3D flow** and **3D movie** in the tinatool window that can be used to invoke the 2D and 3D optical flow tools and the 3D Movie tool.

Before running an optical flow tool or the movie tool, **Mono Tool** and **Imcalc Tool** must be instantiated and two TV installed. **Mono Tool** provides generic file manipulation, including controlling the TV tools and accessing the tinatool stack via pop and push operations. It is always necessary to activate this tool. A rendering of it is shown in Figure 2. The optical flow portions of this program also needs access to the image calculator, **Imcalc Tool**, a stack based image processing tool. We use it here to install two TV tools. This tool must also be activated each time the program is run. Figure 3 shows a rendering of it. To create two TV tools holding the top and second from top images on the imcalc stack, we simply click twice on the new TV button in the **tinatool** window. To install the TVs we select **imcalc** or **imcal2** on the image calculator window and the click **install** on the selected TV tool. Remember, if this was done while **tinatool -s** was being run, it will automatically be done again when **tinatool -r** is executed using that **tinatool.cls** file. Figures 4a and 4b show two TV tools that have been initiated (so far the stack is empty so no images are present). Unfortunately, even if an image is displayed in a TV, we cannot use an X program like **xv** to grab it as the image's grayvalues are treated like pointers into X windows' current 8-bit colour lookup table, resulting in useless (but pretty) "modern-art-like" looking colour images.

2.1 Computing 2D Flow

At this point we can activate the 2D optical flowtool by clicking on the **2D flow** button on the **tinatool** window. The 2D flowtool window is shown in Figure 5. The tool allows one to compute Lucas and Kanade and/or Horn and Schunck optical flow on an 2D image sequence (by default compute the flow for **yos.9** using the 7 images from **yos.6** to **yos.12**). Of course the image stem name **yos.** can be changed (for example to **rubic.**). While the

[1] cd pub/vision/TESTDATA.

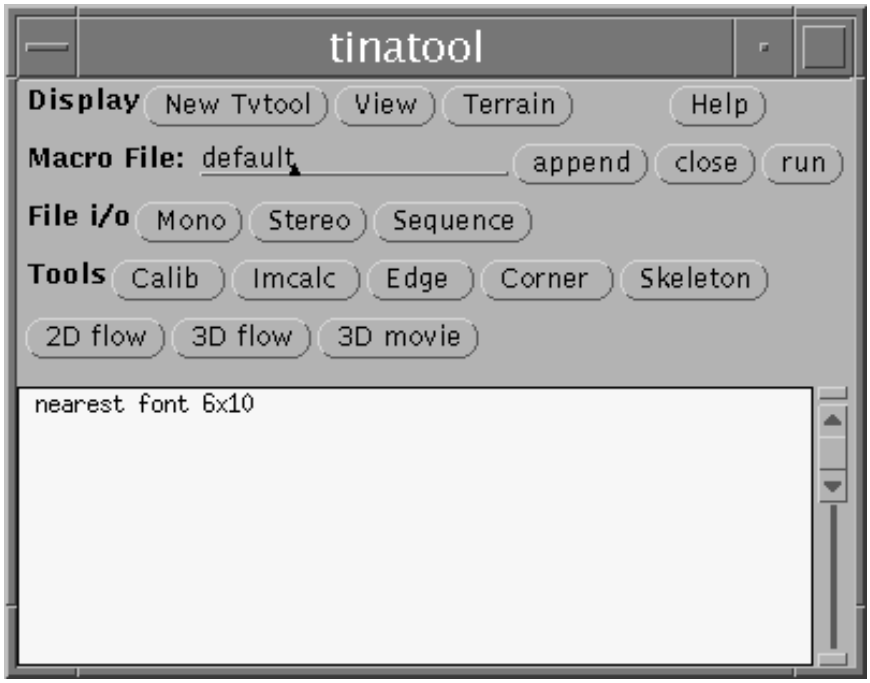


Figure 1: The main tinatool window, showing buttons for 2D and 3D optical flow and 3D Movietool.

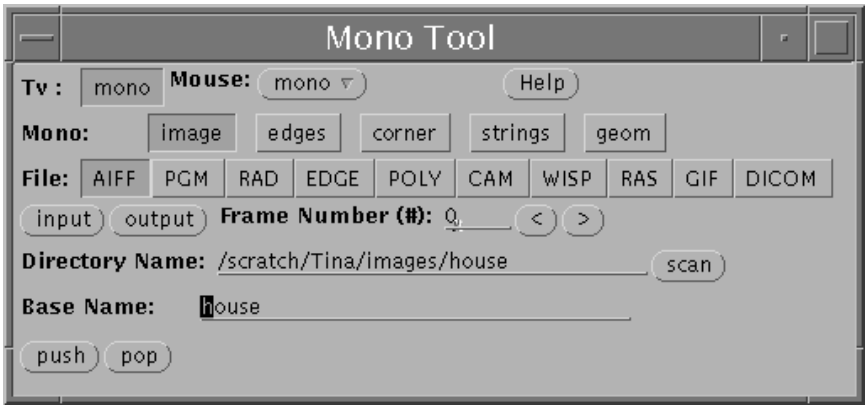


Figure 2: The monotool for accessing the tinatool stack.

images in a particular sequence must all be the same size (and 7 are always required) the image size for different sequences can vary. The program does this by de-allocating all the **Imrects** used for the previous image sequence (if there was one) and re-allocating them with the new size information when a new sequence is input. Image 9 is defaulted as the middle image and **DATA2D** is defaulted as the directory where all data/images are read from or written to. Both can be changed.

Two **tw_choice** buttons allow one to indicate whether your program is running on a SUN workstation or on a PC and whether your input data was made for a SUN or for a PC. These buttons enable/disable swapping the integers in the image headers (the images are assumed to be SUN grayvalue images) or SUN unsigned shorts in the MRI data (see below). Of course, a single **tw_choice** button would suffice here, if one could aways remember when to swap and when not to swap! The default assumption is that the data was made for a SUN and your program is running on a SUN. If one choice is a PC and the other choice is a SUN then byte swapping will be enabled. If both choices are SUN (or PC) no byte swapping is done. Note that 3D derivative data and 3D velocity data is, by default, read or written for the machine tinatool is running on.

The images used here were generated on a SUN workstation, to run this program and properly read the image data one must click either of these buttons if you are using a PC. Note that this means any output images are suitable for viewing a SUN workstation only (unless the button is clicked a second time after the images have

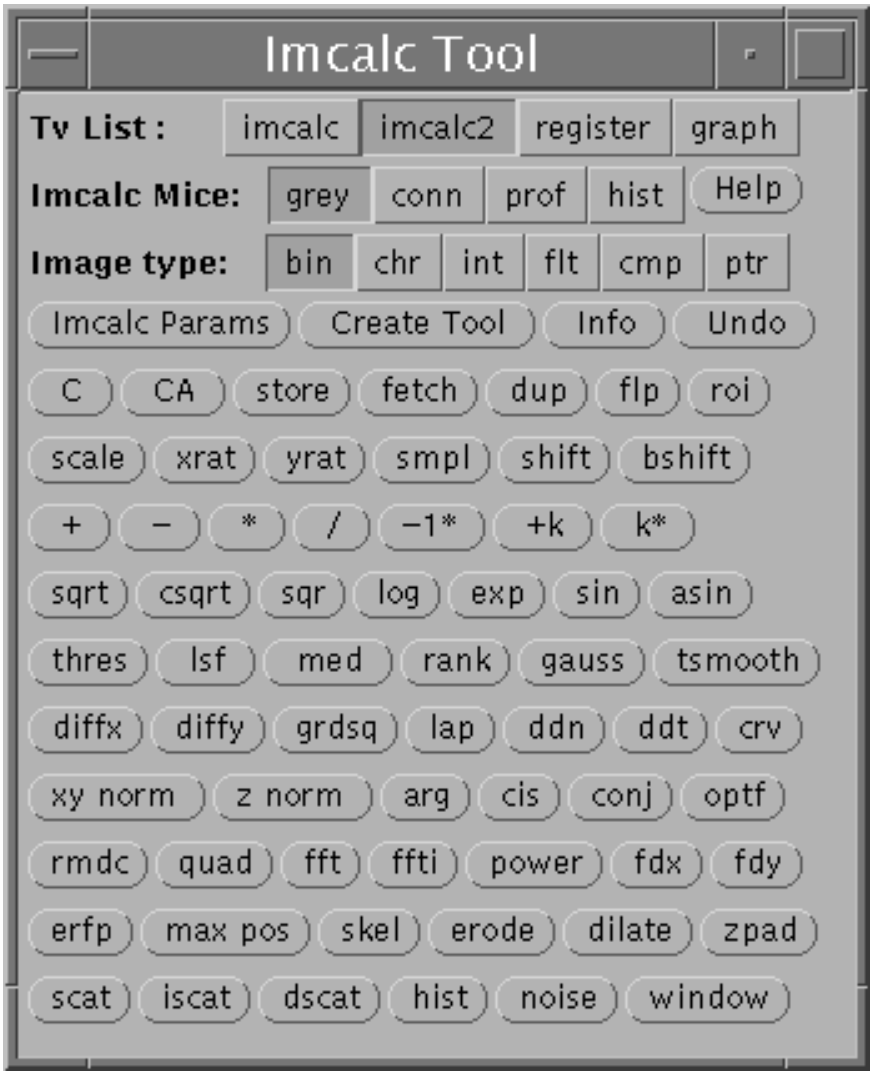


Figure 3: The image calculator tool.

been read to make the output images suitable for a PC or for an X program (such as **xview**), which automatically understands the difference between big/little Indian Architecture). When the **Read and Differentiate** button is pressed, the specified 7 images are read and differentiation is performed. The differentiation routine, **apply_Simoncelli_filters_2D**, also prints out rasterfiles of the derivative data for the central image. Figures 6a to 6d show the original 9^{th} Yosemite Fly-Through image, its I_t derivative image and its I_x and I_y derivative images (floating point values scaled to be in the range $[0-255]$ using the appropriate tinatool functions)[2]. Once the images have been differentiated, clicking the **Compute L&K** button will perform the 2D Lucas and Kanade optical flow calculation while clicking the **Compute H&S** button will perform the 2D Horn and Schunck optical flow calculation. These programs will use the parameters as set (or the defaults) in this window in their calculation. For example, the smallest eigenvalue threshold for Lucas and Kanade, τ_D, and the α value for Horn and Schunck are both 1.0 while the number of iterations and spatial gradient thresholds for Horn and Schunck are 50 and 0.0 respectively. Note that the **imcalc** TV should hold the image (with the flow superimposed on it) on the top of the stack while the **imcalc2** TV will hold the second image (with the flow superimposed on it) on the stack. Thus, if one first computes Lucas and Kanade and clicks the **Save and Display L&K** button, the image and superimposed flow will appear in the **imcalc** TV, if then Horn and Schunck is run and **Save and Display H&S** is pressed, the Horn and Schunck flow replaces the Lucas and Kanade flow in the **imcalc** TV while the Lucas and Kanade flow moves to the **imcalc2** TV (it is now the second image on the stack). This allows the flow for the two algorithms to be compared side by side. Pressing **Save and Display L&K** or **Save and Display H&S** also save the image

[2]The function in **calc2Dflow.c.c** that computes the derivatives, **apply_Simoncelli_filters_2D**, also computes 2^{nd} order derivatives, I_{xx}, I_{xy}, I_{yx}, I_{yy}, I_{xt} and I_{yt}, but these derivatives are not used in the algorithms described here. 2^{nd} order derivatives are computed by convolving the Simoncelli kernels, p_5 or d_5, on the 1^{st} order derivatives in the appropriate dimensions.

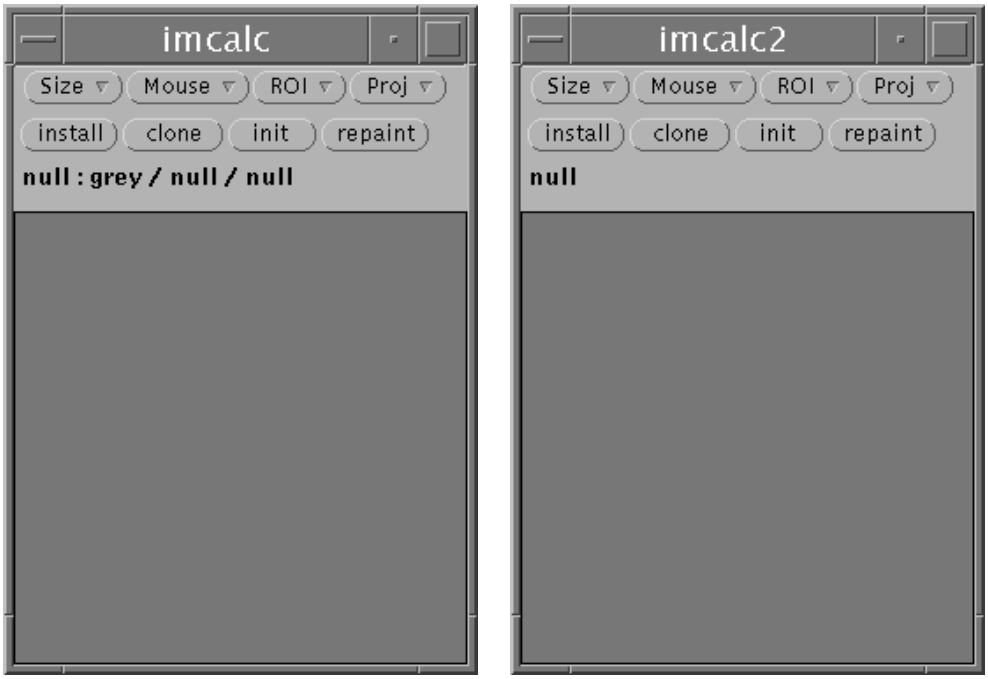


Figure 4: Two blank installed TVs.

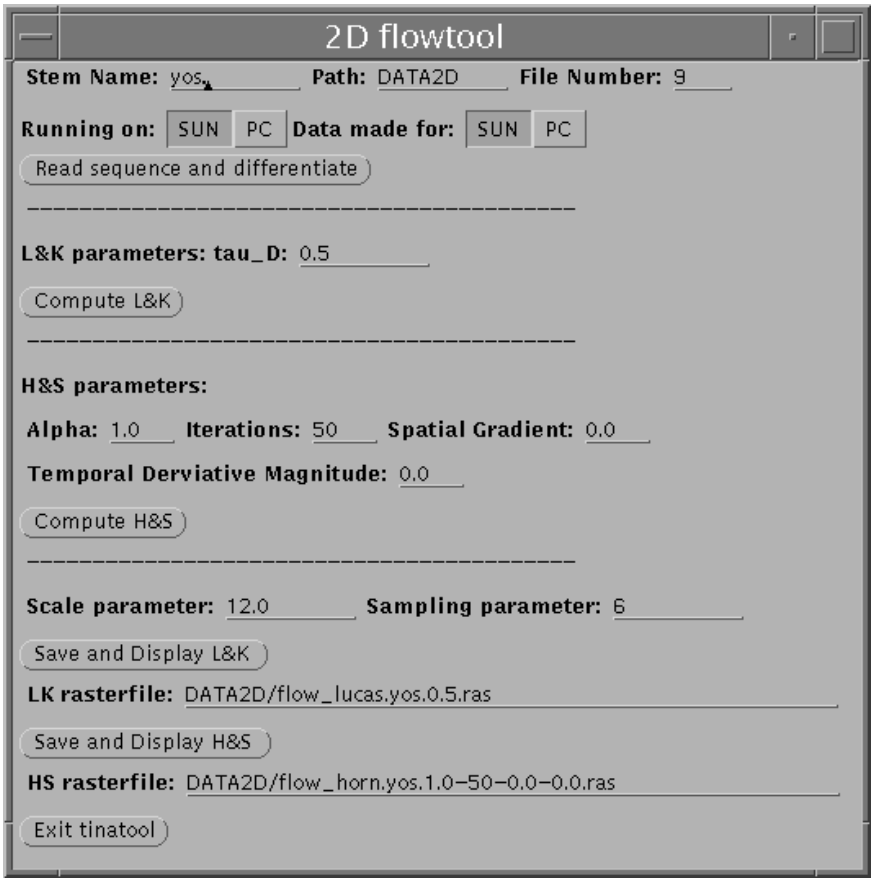


Figure 5: The 2D optical flow tool.

on the top of the stack as a SUN rasterfile. Also note that the flow fields are scaled and sampled by the current scaling and sampling values (which can be changed at any time). The **LK rasterfile** and **HS rasterfile** labels

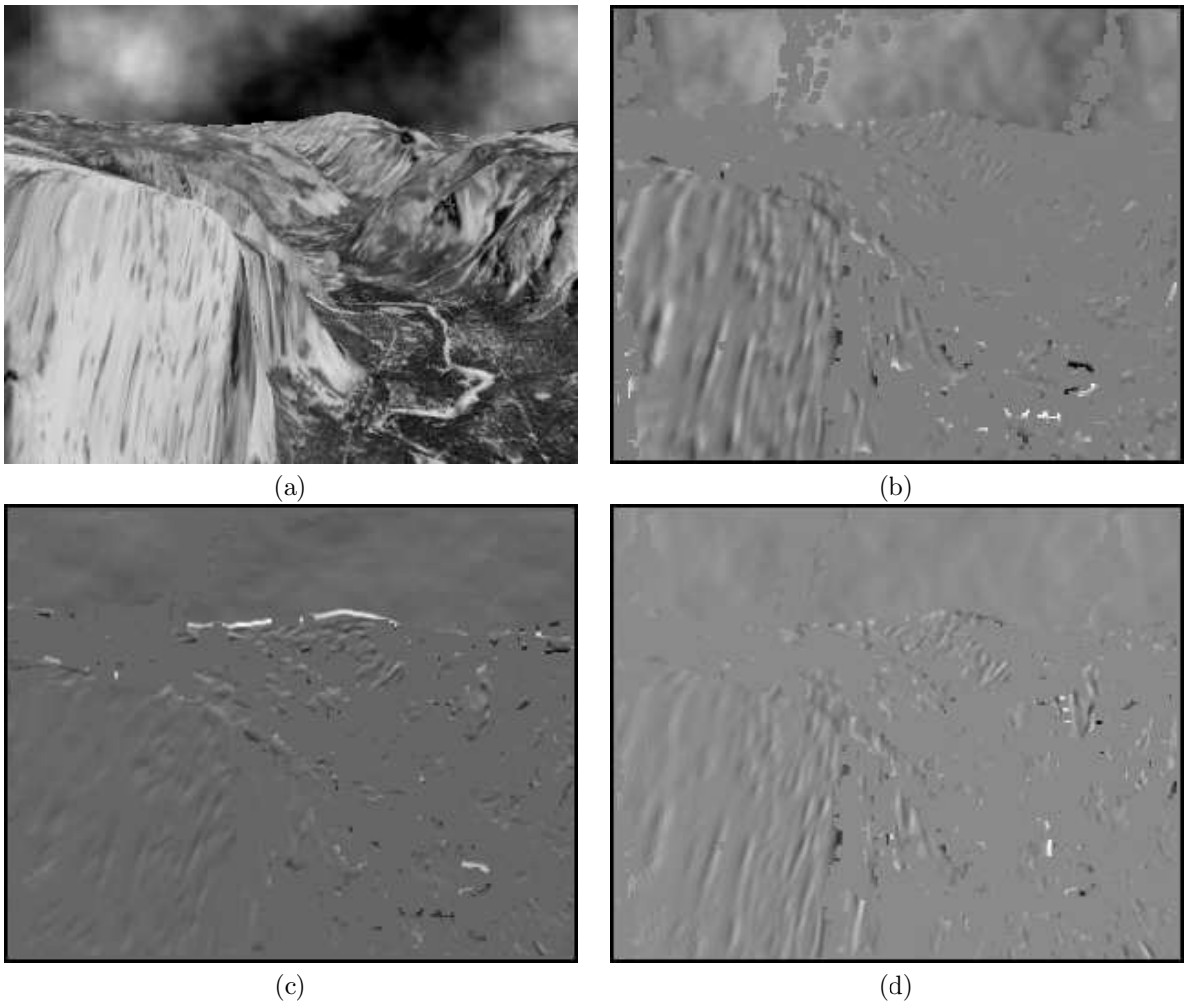
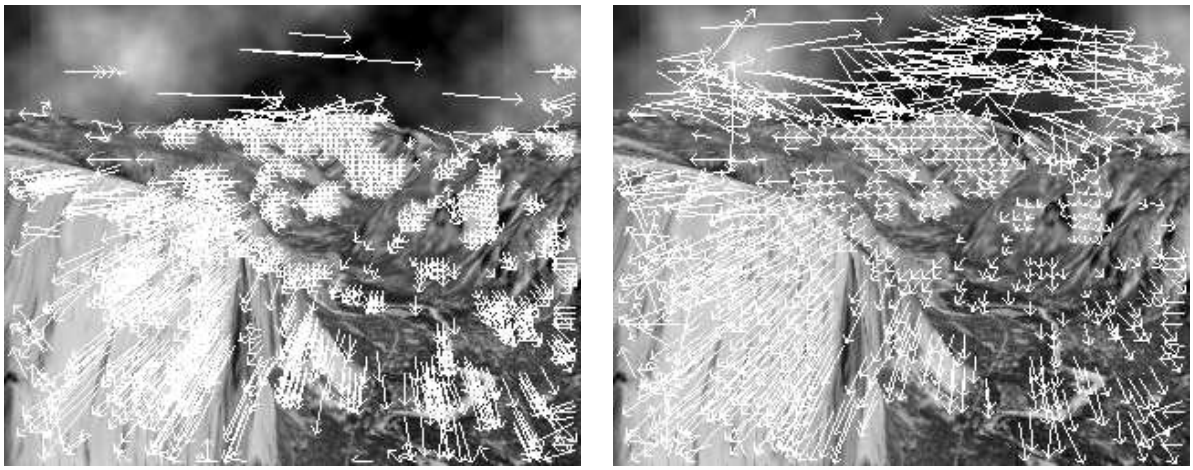


Figure 6: (a) The original 9^{th} image of the Yosemite Fly-Through sequence (yos.9) and (b) the I_t, (c) the I_x and (d) the I_y derivative images of yos.9.

give the current Lucas and Kanade and Horn and Schunck flow rasterfile names that an image would be saved to. These names are instantly updated using **tw_sglobal_reset** when parameter values are changed and a save operation is performed. Lastly, an **Exit tinatool** button terminates the current tinatool session. Figures 7 and 8 show some Lucas and Kanade and Horn and Schunck flow fields computed for the Yosemite Fly Through sequence [These 316×252 images were generated by Lynn Quam at SRI in the late 1980's]. When **save and display L&K** or **save and display H&S** is clicked the image with the superimposed flow is saved as a SUN grayvalue rasterfile with the appropriate name. The byte order of the integers in the rasterfile's header is determined by whether **SUN** or **PC** has been chosen via the two choice buttons.

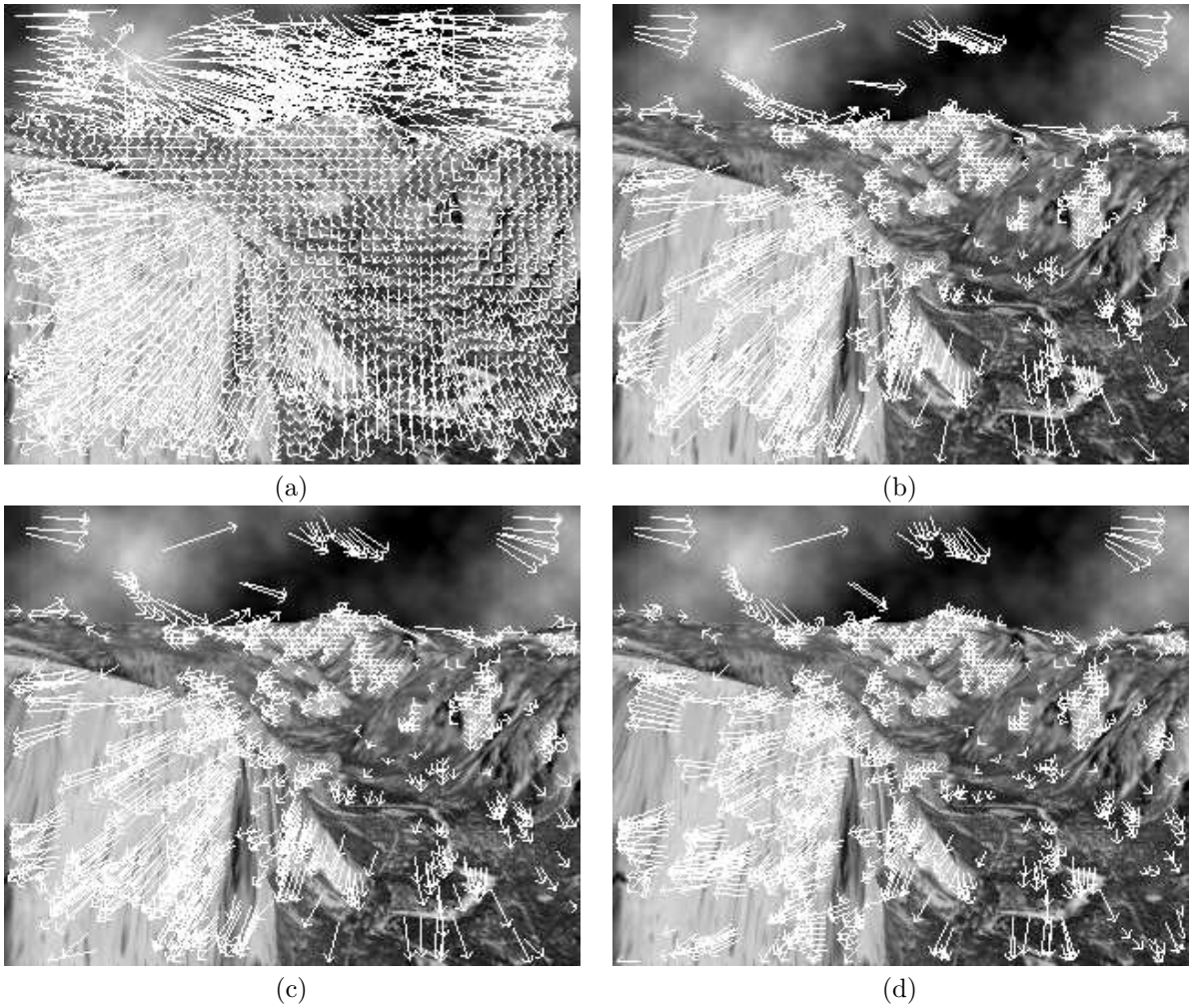
The flows for Lucas and Kanade in Figure 7 show that a smaller value of τ_D results in a denser flow field and flow being computed in the fractal cloud area (the clouds are non-rigid and consequently yield poor flow). The flows for Horn and Schunck in Figure 8 show the various effects of the parameters. When $\alpha = 1.0$, 100 iterations and $||\nabla I||_2 \leq 0.0$ are used we obtain an almost 100% dense flow field (Figure 8a), when $\alpha = 1.0$, 100 iterations and $||\nabla I||_2 \leq 5.0$ are used, we obtain a more sparse flow field (Figure 8b). When $\alpha = 1.0$, 50 iterations are used instead of 100 and $||\nabla I||_2 \leq 5.0$ (Figure 8c) the flow is slightly less smooth than the flow in (Figure 8b) while when $\alpha = 10.0$, 50 iterations are used and $||\nabla I||_2 \leq 5.0$ (Figure 8d) the flow field is slightly more smooth than the flow fields in Figures 8b 8c.

The program can also be run with other images of different sizes. For example, if the stem name is changed to **rubic.** we get an image of a Rubik's Cube rotating on a microwave turntable [made by Richard Szeliski, then at Digital Equipment Corporation]. The image is 240 rows by 256 columns. To handle the different image size, the space for the **yos.** images is first de-allocated and then new space (**Imrects**) with the correct sizes are allocated. Figure 9a shows this middle image in the sequence while 9b shows its gradient image (automatically computed and output by the program at the end of the differentiation step). Figure 10 shows the flow fields for Lucas and



(a) (b)

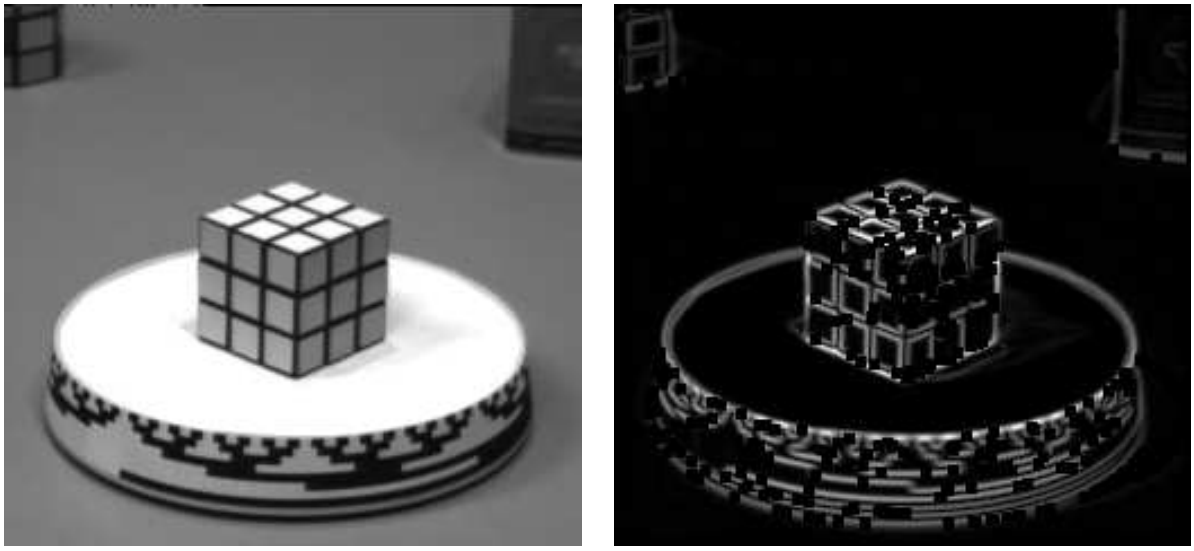
Figure 7: The Lucas and Kanade 2D flow fields using the smallest eigenvalue threshold (a) $\tau_D = 1.0$ [the default value in the 2D optical flowtool window] and (b) $\tau_D = 0.1$.



(a) (b)

(c) (d)

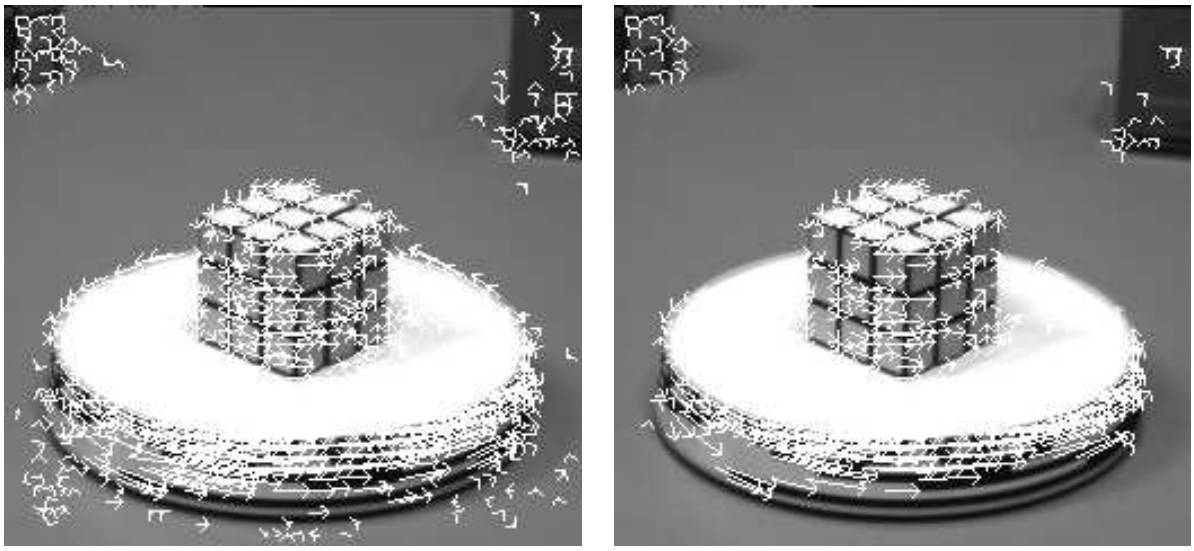
Figure 8: The Horn and Schunck 2D flow field using (a) the default parameters in the 2D optical flowtool window, i.e. $\alpha = 1.0$, 100 iterations and the spatial gradient threshold being $||\nabla I||_2 <= 0.0$, (b) with with the spatial gradient changed to $||\nabla I||_2 <= 5.0$, (c) with the number of iterations changed to 50 and (d) with the value of $\alpha = 10.0$.



(a) (b)

Figure 9: (a) The original middle image of the Rubik's Cube sequence and (b) its gradient image $||\nabla I||_2 = \sqrt{I_x^2 + I_y^2}$.

Kanade when the smallest eigenvalue threshold, τ_D, is 0.1 and 0.5 respectively. Note that for the higher τ_D value the flow is less dense and many erroneous flows have been removed. Finally Figure 11 shows the flow fields for



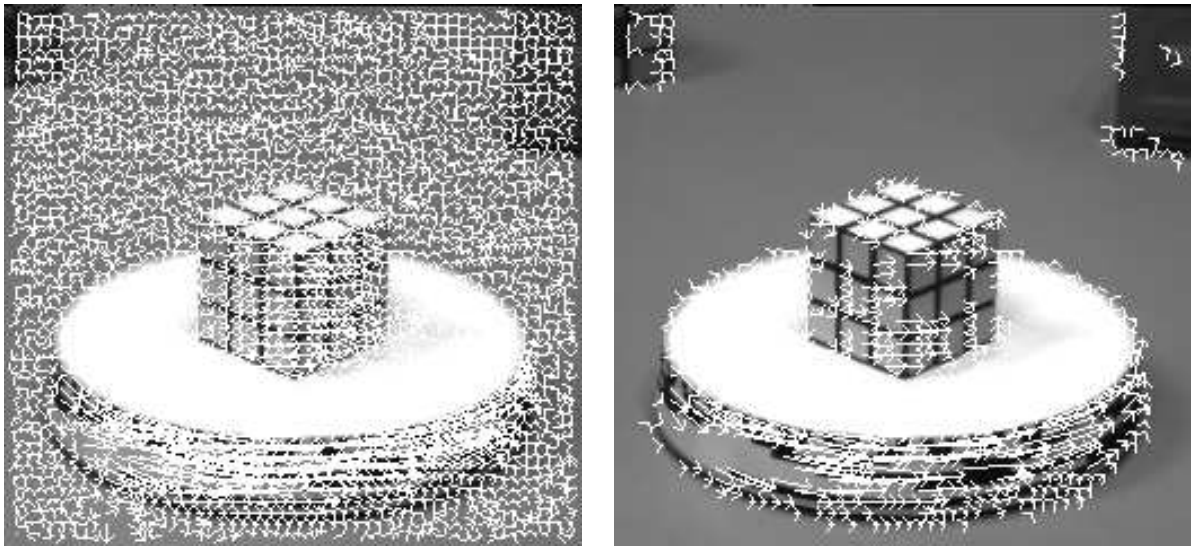
(a) (b)

Figure 10: The Lucas and Kanade flow fields for the Rubik's Cube sequence (a) when the smallest eigenvalue threshold, τ_D, is 0.1 and (b) when it is 0.5.

Horn and Schunck for 100 iterations when the spatial gradient threshold ($||\nabla I||_2 = \sqrt{I_x^2 + I_y^2}$) is either 0.0 or 5.0. Note that the result for 5.0 has all the small background flows removed (correctly as there is no image motion here).

2.2 Computing 3D Flow

The **3D flow** button activates the 3D optical flowtool, **3Dflowtool**, as shown in Figure 12. Again, the Lucas and Kanade and Horn and Schunck algorithms were simply extended to 3D as described above. This tool can read in 7 volumetric sets of data and differentiate them (to compute the 3D spatiotemporal derivatives) and then compute either the 3D Lucas and Kanade flow or 3D Horn and Schunck flow.



(a) (b)

Figure 11: The Horn and Schunck flow fields for the Rubik's Cube sequence (a) when $\alpha = 1.0$, the number of iterations was 100 and the spatial gradient threshold was $||\nabla I||_2 = \sqrt{I_x^2 + I_y^2} \leq 0.0$ and (b) when the spatial gradient threshold was changed to $||\nabla I||_2 = \sqrt{I_x^2 + I_y^2} \leq 5.0$.

Currently, we have analyzed two 3D volumetric datasets, one a 3D sinusoid moving with a constant velocity of $(V_x, V_y, V_z) = (3.0, 2.0, 1.0)$ which we used to check the correctness of our optical flow computations and the other a low resolution gated MRI sequence of cardiac motion made by Robarts Research Institute at the University of Western Ontario. Both datasets are comprised of 20 sets of $256 \times 256 \times 31$ unsigned shorts (2 bytes). We call this raw data as no file format is used.

Owing to the large size of the 3D volume datasets, the **3D flowtool** has been designed differently than **2D flowtool** in that it allows one to read the Sinusoid or MRI volume data only via:

Read 3D data for viewing only,

differentiate the 3D volume data via:

Read 3D data (if not read) and differentiate,

re-read the 3D volume data and derivative data via:

Read 3D data and previously computed 3D derivative data,

compute Lucas and Kanade or Horn and Schunck optical flow via:

Compute 3D L&K Velocities

or

Compute 3D H&S Velocities

or read the previously computed 3D derivative and 3D optical flow via:

Read previously computed L&K Velocities and Derivatives

or

Read previously computed H&S Velocities and Derivatives.

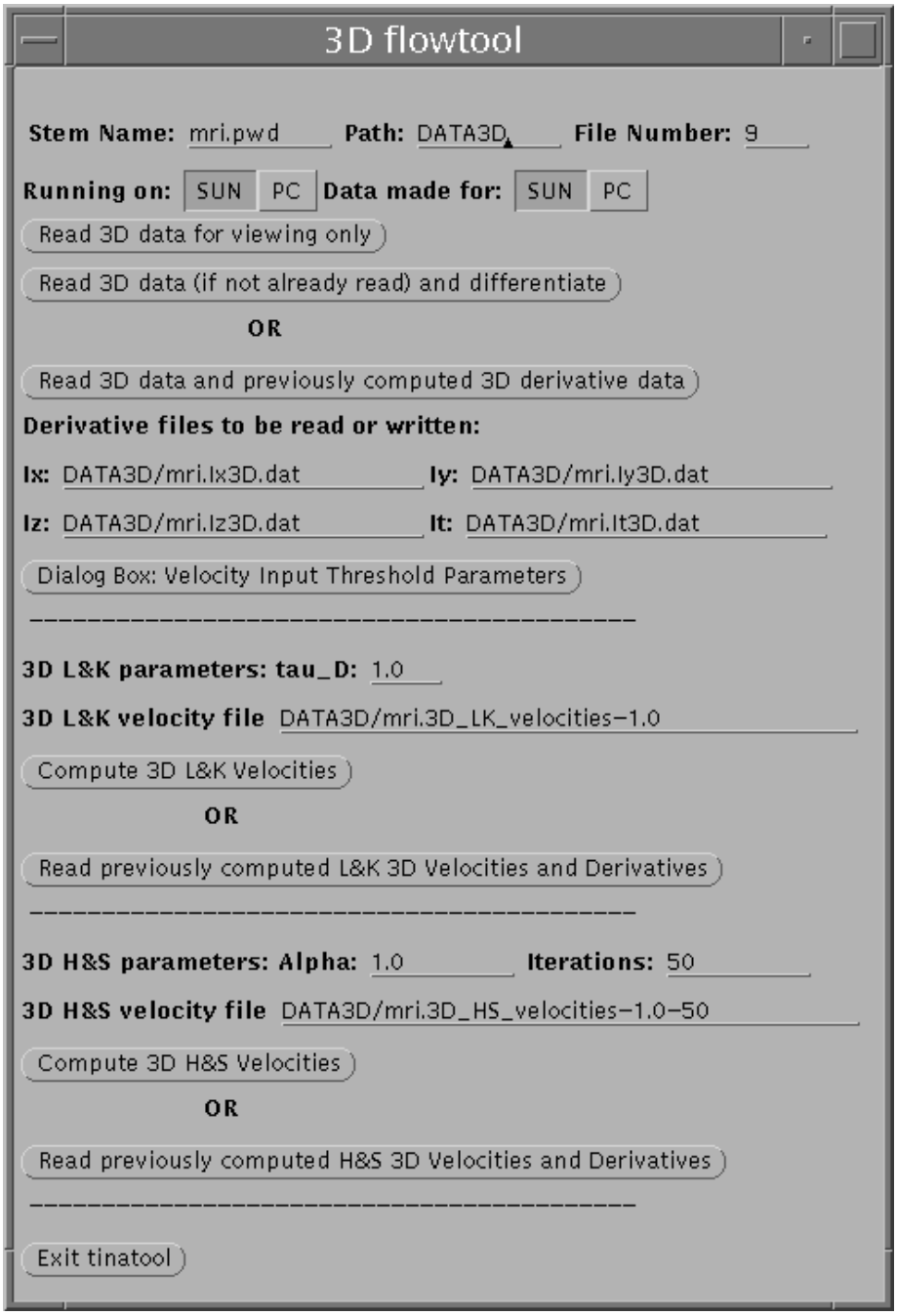


Figure 12: The 3D optical flow tool.

A full computation on a 750MHz laptop running Linux 7.2 takes about 5 minutes for differentiation and 10 minutes for a Lucas and Kanade optical flow computation! The amount of time required for a Horn and Schunck optical flow computation can run into the 10's of hours depending on the number of iterations chosen (20 minutes for 50 iterations). Currently, these algorithms are not real-time!!!

Again the default stem name is **mri.**, the default data path (where files are read from or written to) is **DATA3D** and the default middle image (for which flow is computed) is 9. Thus files **mri.6** to **mri.12** must be available in directory **DATA3D** for reading and be in raw data format. Again **tw_choice** buttons allows one to select a the type of machine you are running on and the machine the data was made on (each either a **SUN** or a **PC**). The sinusoid data consists of unsigned shorts made on for PC (so no swapping is necessary if running on a PC). The MRI data consists of unsigned shorts made on a SUN and the bytes of each short must be swapped if running on a PCs. The **Read 3D data for viewing only** lets one read the data in and explore it using the 3D movietool. If the **Read 3D data (if not read) and differentiate** is pressed then the 3D MRI data is read (unless it has already

been read) and the 3D I_x, I_y, I_z and I_t derivatives are computed. After the differentiation calculation, some rasterfile images of the derivatives at the middle time (now renumbered 3) and middle slice (15) are automatically saved. If, on the other hand, the **Read 3D data and previously computed 3D derivative data** is pressed then the 4 derivative files with default names (these can be changed by changing the file stem name or the output directory and execution of a **tw_sglobal_reset** command):

- **DATA3D/mri.Ix3D.dat**,
- **DATA3D/mri.Iy3D.dat**,
- **DATA3D/mri.Iz3D.dat** and
- **DATA3D/mri.It3D.dat**

are read (these file must exist). Note that as well as reading the derivative data the MRI data is also read. All this data is maintained in dynamically allocated tinatool **Imrects**.

Again there are default values for the Lucas and Kanade τ_D parameter (of 1.0) and Horn and Schunck α and number of iterations parameters (of 1.0 and 50 respectively). There are 2 file default labels for 3D velocity file names:

- **DATA3D/mri.3D_LK_velocities-1.0** and
- **DATA3D/mri.3D_HS_velocities-1.0-50**.

These are created using the stem name (**mri.**), the data path name (**DATA3D**) and either the value of τ_D (1.0 is the default for Lucas and Kanade) or the values of α and the number of iterations (1.0 and 50 are the default values for Horn and Schunck). If these values are changed than the label values will also be changed to reflect the new names via execution of a **tw_sglobal_reset** command.

If either the **Read previously computed L&K 3D Velocities and Derivatives** or **Read previously computed H&S 3D Velocities and Derivatives** buttons is pressed, the corresponding velocity name is read (this assumes it exists). Also if the 3D Sinusoid or MRI data and 3D derivative data has not been read it is assumed to have been previously computed and is re-read. Indeed, the same function to perform the **Read 3D data and previously computed 3D derivative data** is used by those functions. If one reads a previously computed 3D velocity file one can also perform thresholding on it (see below). Indeed, this is the style adapted to allow the 3D optical flow fields of various computations to be viewed under different thresholding conditions without a redundant velocity computation. If a velocity computation is made and the 3D velocity written, it can then be immediately re-read with the input thresholds as set in a dialog box. The **Compute 3D L&K Velocities** or **Compute 3D H&S Velocities** buttons do a new 3D velocity computation (CPU intensive). Note that when a second 3D optical flow calculation is done or re-read, the first optical flow result is overwritten: another reason to allow 3D velocity files to be saved and re-read. [None of this is necessary in the 2D case as those computations are very fast.]

When reading in a previously computed 3D optical flow field, 4 types of thresholding are available (their values can be set via a dialog box):

1. The minimum acceptable velocity magnitude can be specified: all velocities below this are eliminated (typically these small velocities are due to noise),
2. The maximum acceptable velocity magnitude can be specified: all velocities above this are eliminated (good for eliminating outliers),
3. The spatial gradient threshold $||\nabla_I||_2 = (\sqrt{I_x^2 + I_y^2 + I_z^2})$ can be specified: all velocities computed at positions with a spatial gradient less than this are eliminated and
4. The temporal derivative magnitude threshold $|I_t|$ can be specified: all velocities computed at positions with a temporal derivative magnitude less than this are eliminated.

Figure 13 shows this popup dialog box with the default values.

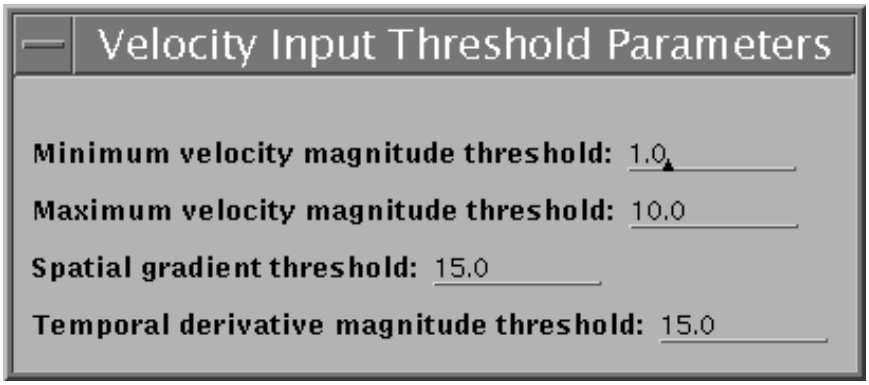


Figure 13: The 3D dialog tool for 3D velocity thresholds.

2.3 Display the 3D Optical Flow

Lastly, the **3D movie** button allows one to examine/explore the 3D volume data in movie mode as well as look at the XY and XZ components of the 3D flow superimposed on the images. The 3D flowtool must be run first, as this function used global variable set by that function.

After reading (and, perhaps differentiating the 3D volume data) or re-reading existing volume and derivative data, one can use this tool to examine the data. One can select a volume number (between 0 and 6) and a slice number (between 0 and 30) and generate a movie for the selected volume (one sees a movie of all 31 slices for that volume number) or by slice number (one sees a movie of all 7 volume images for that slice number). Lastly, one can look at a specific volume image (for the current slice and volume numbers). Buttons allow the slice and volume numbers to be incremented or decremented by 1 (with the resulting image being displayed each time) and these allow one to move through the data. Again a **3D Rasterfile** label has the current filename, which is immediately updated when the slice or volume number are changed (this is also true if the data path or stem names are changed). Initially, the 3D rasterfile label is **DATA3D/labelled_mri.3.15.ras**. Note that the "labelled" part of the name means the volume and slice name are drawn directly on the image (by some functions in **calc2Dflow.c**).

Once a 3D optical flow calculation has been made (or re-read) one can view a movie of the XY and XZ components of the flow superimposed on the corresponding image. Since the MRI images are made from an axial view (sometimes called the long axis, it is top-down from the head to the feet), the XY flows correspond to this orientation. The XZ images show the motion along the sigital direction (side view) and these images are not currently displayed, there are 256 of them and they each have an awkward size of 256×31. Rather, the XZ flow is displayed on the axial images as well on the 2^{nd} TVtool. The movie is made by pushing pairs of XY and XZ images with the corresponding superimposed flow onto the stack in a loop; the effect is that the **imcalc** TV shows the XY movie while the **imcalc2** TV shows the XZ movie at the same time. Buttons also exist to allow one to move through the slice data and examine each flow field in turn. Of course, since the flow is only computed for the middle volume one cannot decrement or increment the volume number (it always stays at 3). Again, via use of the **tw_sglobal_reset** command, the **XY Rasterfile** and **XZ Rasterfile** names stay current with slice number, stem name and data path. **DATA3D/labelled_lucas.mri.XY.3.15.ras** and **DATA3D/labelled_lucas.mri.XZ.3.15.ras** are the current default names.

Figures 15a and 15b show the XY and XZ flow fields for the sinusoid datasets for Lucas and Kanade while Figures 16a and 16b show the XY and XZ flow fields for the sinusoid datasets for Horn and Schunck. These flows are quite accurate as shown by the numerical results in Tables 1 and 2. The large standard deviations in the tables,

Velocity Component	Correct Value	Absolute Error	Absolute St. Dev.	Percent Error	Percent St. Dev.
V_x	3.0	3.003015	0.509606	0.874446%	50.7658596%
V_y	2.0	2.002197	0.240405	0.333465%	24.0648365%
V_z	1.0	0.914160	0.364763	4.340410%	40.2264871%

Table 1: The correct velocity component values and their average absolute and percentage errors and standard deviations for the three 3D optical flow components computed by the 3D Lucas and Kanade algorithm for the sinusoid datasets. $\tau_D = 1.0$ was used.

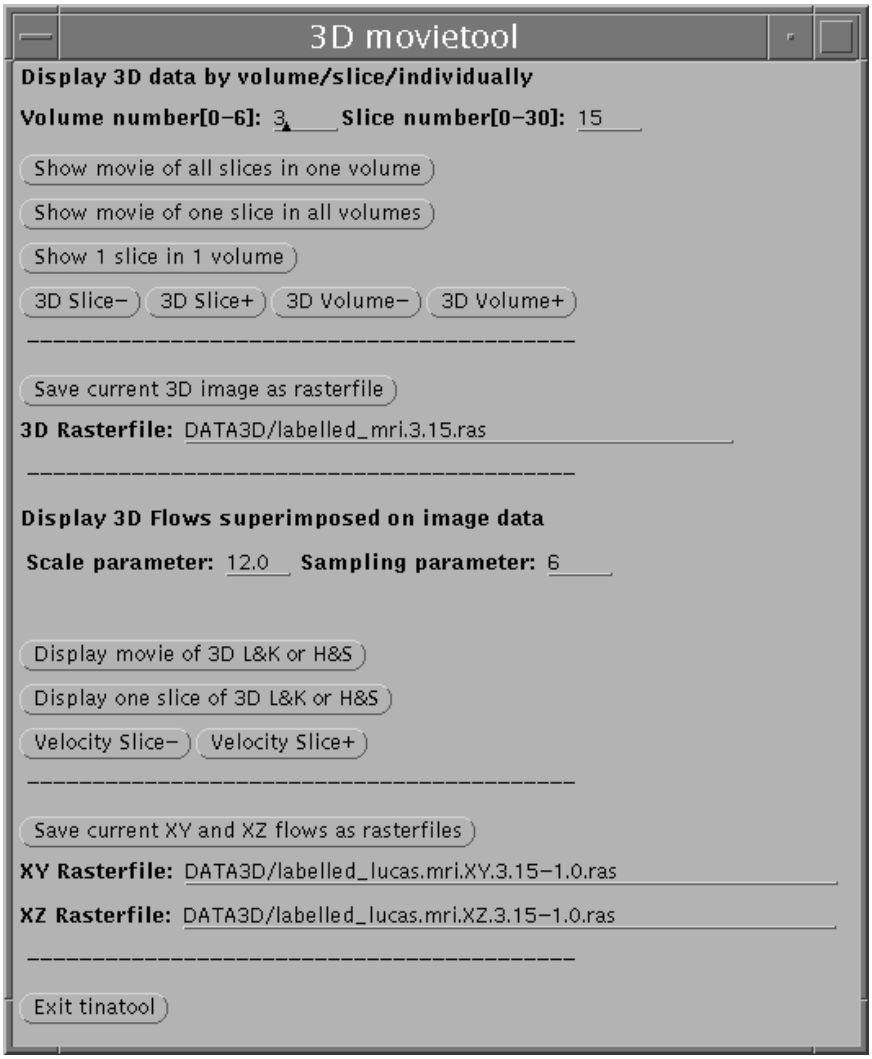


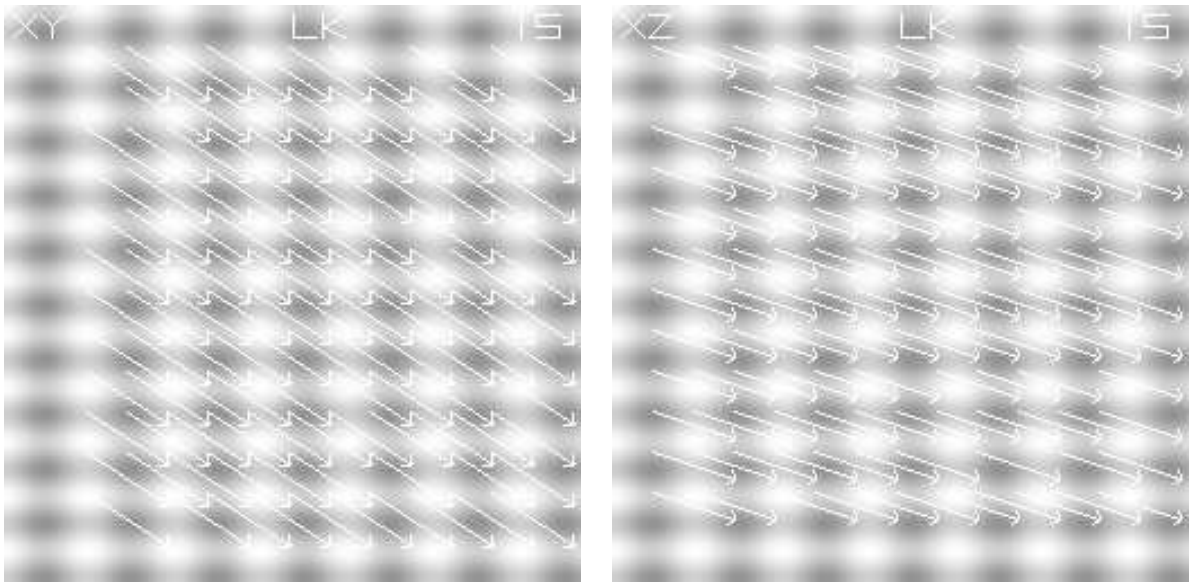
Figure 14: The 3D movietool.

Velocity Component	Correct Value	Absolute Error	Absolute St. Dev.	Percent Error	Percent St.Dev
V_x	3.0	2.999351	0.006280	0.593660%	1.234306%
V_y	2.0	2.000206	0.004270	0.333465%	0.730525%
V_z	1.0	1.044922	0.003901	4.340410%	8.689922%

Table 2: The correct velocity component values and their average absolute and percentage errors and standard deviations for the three 3D optical flow components computed by the 3D Horn and Schunck algorithm for the sinusoid datasets. $\alpha = 1.0$ and 200 iterations were used.

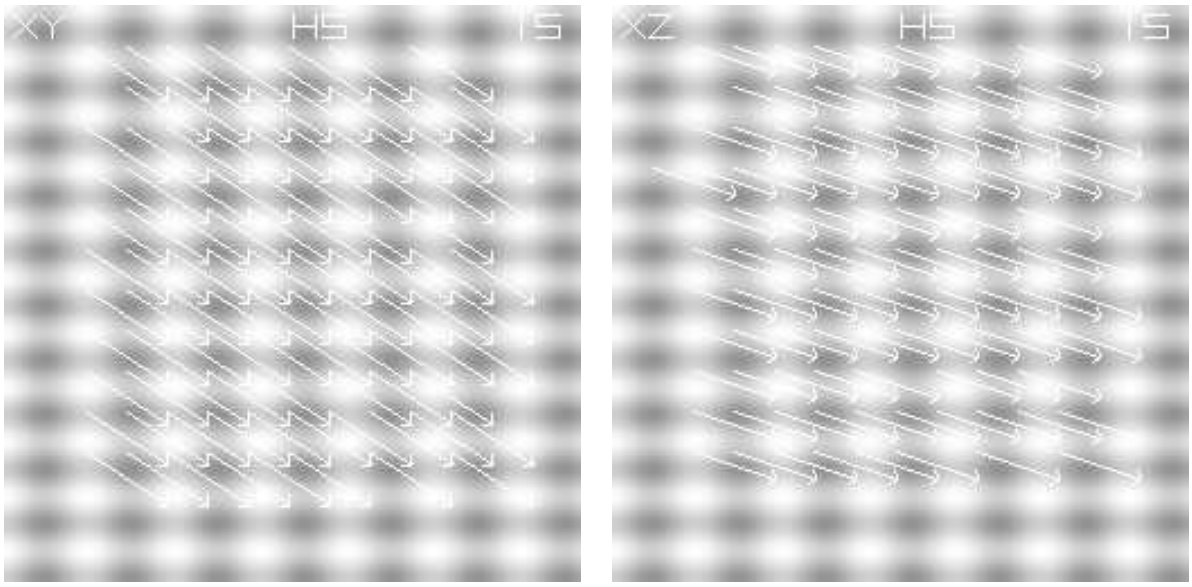
especially in Table 1 and to a less degree in Table 2 result from poor derivatives calculations in slices 2-4 and 26-29, an artifact of the construction of the sinusoid data sequence. The velocity was 1.0 in the z dimension and with only 31 images in that dimension, good differentiation was not always possible everywhere for the chosen sinusoid wavelength. Of course, for images 5-25 the flow was quite good. These artifacts are smoothed out by Horn and Schunck, as we can see from the considerably lower standard deviation values in Table 2, especially for V_x and V_y. Program **make3Dsin.c** was used to compute sin.1 to sin.20. The main purpose, of course, in using this sinusoid data was to show the correctness of the two 3D algorithms and this has been accomplished.

Figures 17, 18 and 19 show the Lucas and Kanade XY and XZ flow fields superimposed on the appropriate images for the 10^{th}, 13^{th}, 15^{th}, 17^{th} and 20^{th} slices of volume 3. [Flow is only computed for volume 3, which is the middle volume MRI dataset, **mri.9**.] Note that the volume number and slice number are labels on the images. Label "LK" indicates Lucas and Kanade while label "HS" indicates Horn and Schunck. The smallest eigenvalue threshold, τ_D



LK-XY-Sinusoid LK-XZ-Sinusoid

Figure 15: The Lucas and Kanade XY and XZ flow fields superimposed on the 15^{th} slice of the 3^{rd} volume of the sinusoid data. Scaling is 12 and sampling is 18. $\tau_D = 1.0$.

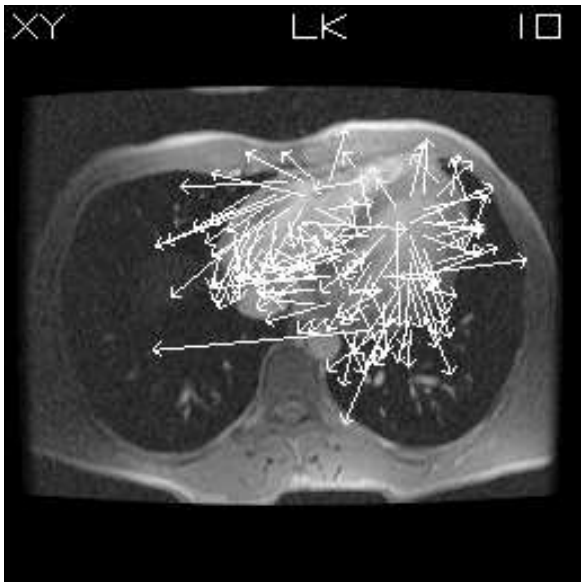


HS-XY-Sinusoid HS-XZ-Sinusoid

Figure 16: The Horn and Schunck XY and XZ flow fields superimposed on the 15^{th} slice of the 3^{rd} volume of the sinusoid data. Scaling is 12 and sampling is 18. 200 iterations were used.

is 1.0 for all the flow fields. Finally, Figure 20 shows the flows for τ_D values of 20.0. The larger τ_D is, generally the fewer but more reliable flow vectors there are.

Figures 21, 22 and 23 show the XY and XZ Horn and Schunck flow fields superimposed on the appropriate images for the 10^{th}, 13^{th}, 15^{th}, 17^{th} and 20^{th} slices of volume 3. Figures 24 and 25 show the effect of using 10, 25 and 50 iterations for the XY and XZ flows for the 15^{th} slice. There is a slight difference (improvement) between the flows for 50 and 100 iterations over those for 10 and 25 iterations but for 200, 500 and 1000 iterations all the flows were effectively the same. Finally, Figure 26 shows the effect of α. The flow fields are for slice 15 and 100 iterations with $\alpha = 10.0$ instead of $\alpha = 1.0$ as above. They seem similar to the flows in Figure 22 [HS-XY-3-15 and HS-XZ-3-15] but more a larger weight has been to the smoothing term in the Horn and Schunck functional in Figure 26. Some obvious outliers have been suppressed.



LK-XY-3-1o

LK-XZ-3-10

LK-XY-3-13

LK-XZ-3-13

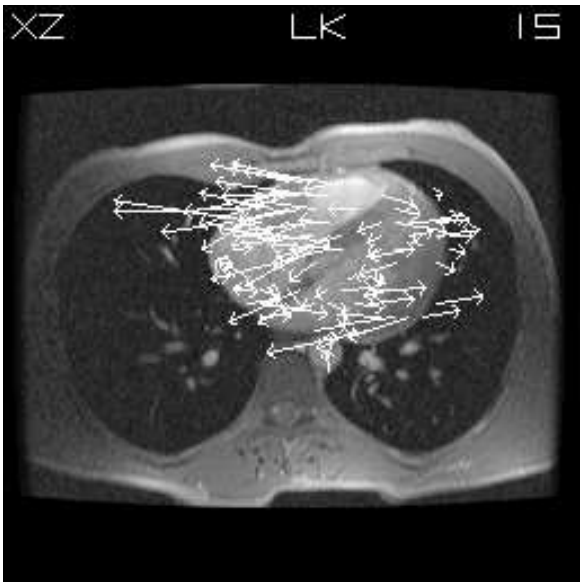
Figure 17: The Lucas and Kanade XY and XZ flow fields superimposed on the 10^{th} and 13^{th} slices of the 3^{rd} volume of MRI data. $\tau_D = 1.0$.

The quality of the MRI flow obtained by the 2 algorithms is still under investigation. Subjectively, the 3D Horn and Schunck flows often look better than the 3D Lucas and Kanade flows. One problem is that the quality of the flow is directly dependent on the quality of the derivatives. The coarse sampling nature of the data and the registration mis-alignments in adjacent slices of the data probably cause serious problems for differentiation. I believe a spline based approach to differentiation may overcome these problems. Another problem with the MRI data is that the 3D motion is discontinuous at places in space and time (after all the heart is beating). A 3D algorithm, based on Nagel's 2D optical flow algorithm [6, 7, 8, 9], where a Horn and Schunck smoothing is used but where the smoothing is inhibited across intensity discontinuities may better be able to handle discontinuous optical flow fields.

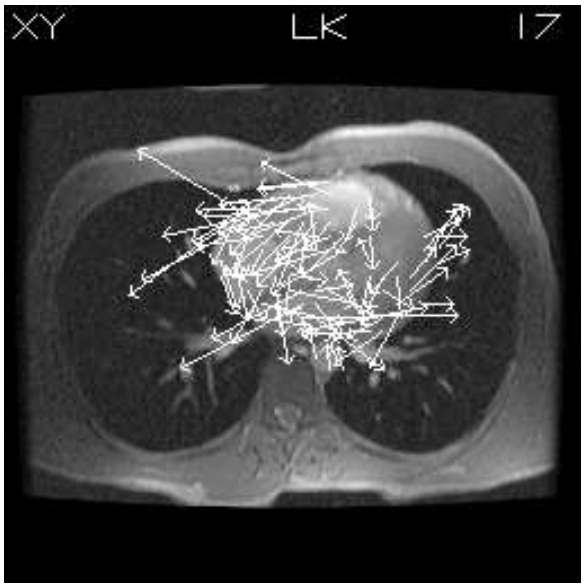
3 The Program Structure

The program is structured into 3 main files: **calc2Dflow.c**, which contains all the 2D optical flow code plus some extra code for superimposing flow on raster images, drawing simple text and numbers, etc. **calc3Dflow.c**, which

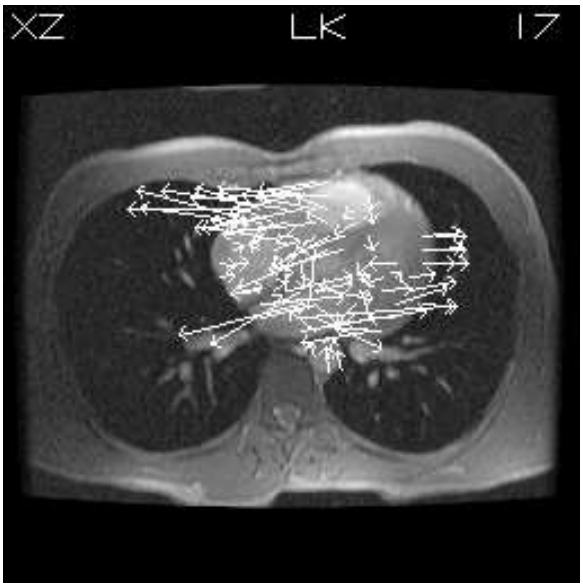
LK-XY-3-15



LK-XZ-3-15



LK-XY-3-17



LK-XZ-3-17

Figure 18: The Lucas and Kanade XY and XZ flow fields superimposed on the 15^{th} and 17^{th} slices of the 3^{rd} volume of MRI data. $\tau_D = 1.0$.

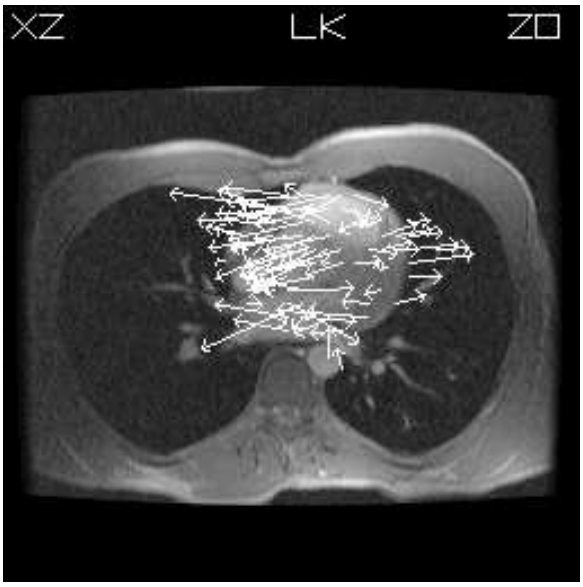
contains the 3D optical flow code, and **flowtool.c**, which contains all the GUI code.

We outline below the main functions contained in each of the files. In **calc2Dflow.c** we have:

- Functions to do lowpass (smoothing) and high pass (differentiation) processing on image sequences:

```
void diff_in_x(Imrect *Ix,Imrect *putting_y,float d_kernel[FIVE],
               int pic_x,int pic_y,int n);
void diff_in_y(Imrect *Iy,Imrect *putting_x,float d_kernel[FIVE],
               int pic_x,int pic_y,int n);
void diff_in_t(Imrect *It,Imrect *putting_t[FIVE],float d_kernel[FIVE],
               int pic_x,int pic_y,int n);
void prefilter_in_x(Imrect *putting_x,Imrect *pre_in_t,
                    float p_kernel[FIVE],
                    int pic_x,int pic_y,int n);
void prefilter_in_y(Imrect *putting_y,Imrect *pre_in_x,
                    float p_kernel[FIVE],
```

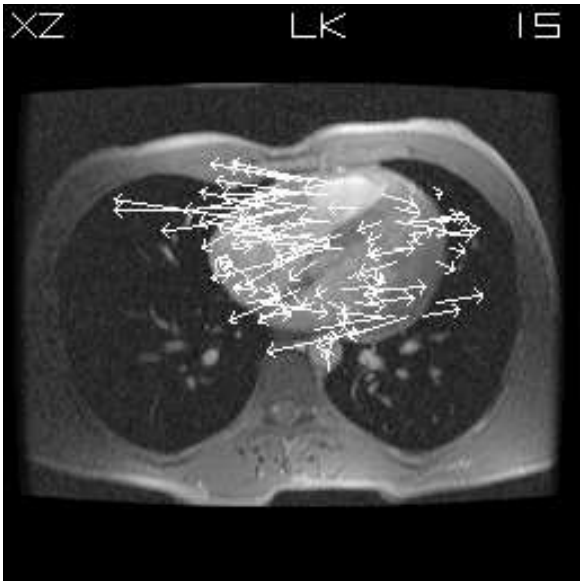
LK-XY-3-20



LK-XZ-3-20

Figure 19: The Lucas and Kanade XY and XZ flow fields superimposed n the 20^{th} slice of the 3^{rd} volume of MRI data. $\tau_D = 1.0$.

LK-XY-3-15 $\tau_D = 20.0$



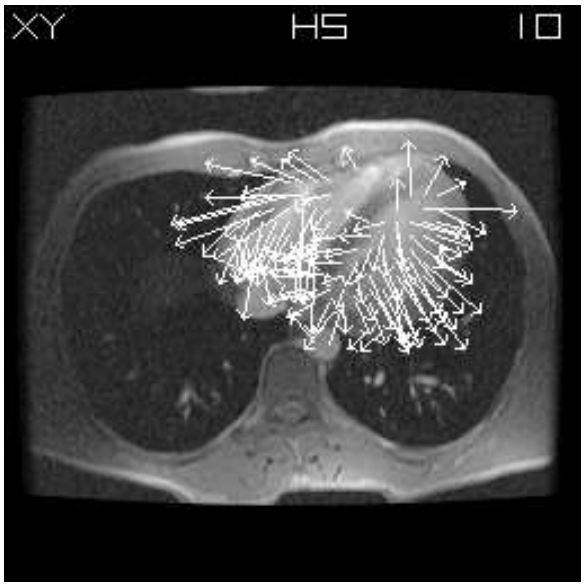
LK-XZ-3-15 $\tau_D = 20.0$

Figure 20: The Lucas and Kanade XY and XZ flow fields superimposed on the 15^{th} slice of the 3^{rd} volume of MRI data for τ_D having a value 20.0.

```
                    int pic_x,int pic_y,int n);
void prefilter_in_t(Imrect *putting_t,Imrect *floatpic[SEVEN],
                    float p_kernel[FIVE],int pic_x,int pic_y);
void calc_lowpass_kernel(float p_kernel[FIVE]);
void calc_highpass_kernel(float d_kernel[FIVE]);
```

- A function to apply these filters to the 2D images to obtain the 1^{st} and 2^{nd} order derivatives, I_x, I_y, I_t, I_{xx}, I_{xy}, I_{yx}, I_{yy}, I_{xt} and I_{yt}. (Again, the 2^{nd} order derivatives were not used here.)

```
void apply_Simoncelli_filters_2D(char stem[MAXPATHLEN],
                         char path[MAXPATHLEN],
                         int start,int middle,int end,
```



HS-XY-3-10

HS-XZ-3-10

HS-XY-3-13

HS-XZ-3-13

Figure 21: The Horn and Schunck XY and XZ flow fields superimposed on the 10^{th} and 13^{th} slices of the 3^{rd} volume of MRI data. $\alpha = 1.0$ and 100 iterations were used.

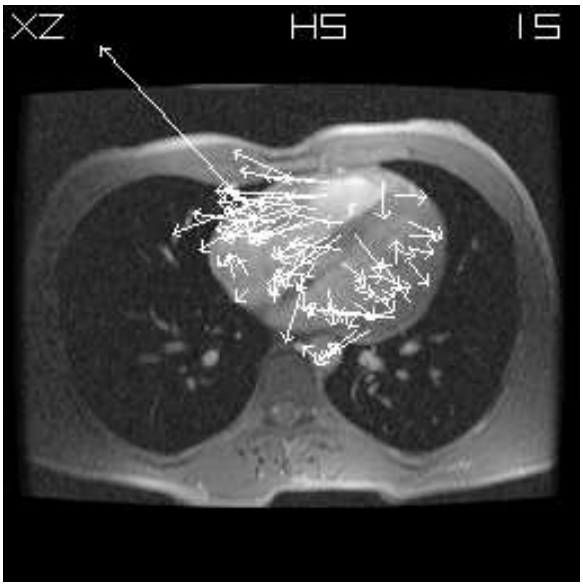
```
Imrect *images[SEVEN],Imrect *Ix,Imrect *Iy,
Imrect *It,Imrect *Ixx,Imrect *Iyy,Imrect *Ixy,
Imrect *Iyx,Imrect *Ixt,Imrect *Iyt,
int pic_x,int pic_y,int n);
```

- A function to compute 2D Lucas and Kanade optical flow. Note that both **raw** and **least squares** 2D normal velocity is also computed in `norm_vels1` and `norm_vels2`. Our program does not display these fields.

```
void compute_lucas_optical_flow_2D(Imrect *Ix,Imrect *Iy,Imrect *It,
                    Imrect *full_vels[2],Imrect *norm_vels1[2],
                    Imrect *norm_vels2[2],int pic_x,int pic_y,int n,
                    float tau_D,int flag,
                    int flow_number);
```

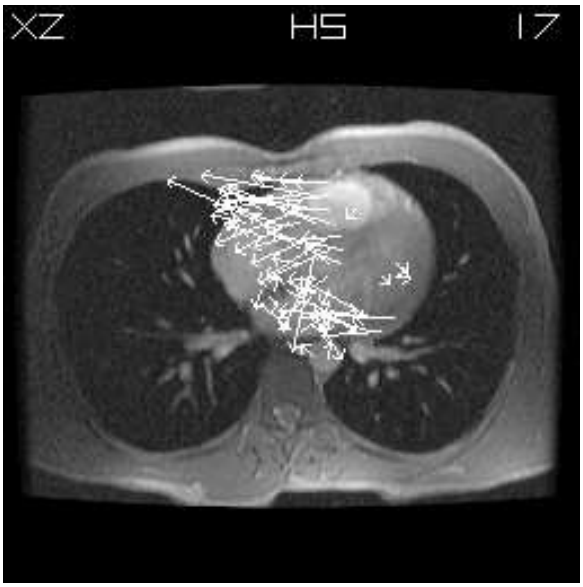
- Three functions to compute iterative Horn and Schunck optical flow, to perform two iterations at a time and to compute 2D velocity averages at each iteration.

HS-XY-3-15



HS-XZ-3-15

HS-XY-3-17



HS-XZ-3-17

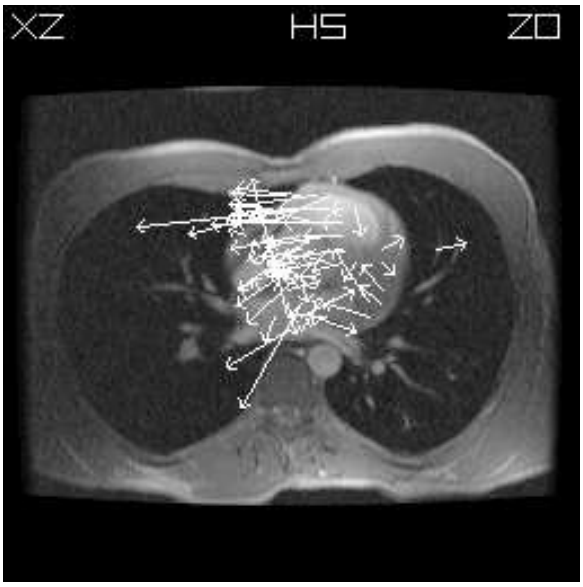
Figure 22: The Horn and Schunck XY and XZ flow fields superimposed on the 15^{th} and 17^{th} slices of the 3^{rd} volume of MRI data. $\alpha = 1.0$ and 100 iterations were used.

```
void compute_horn_optical_flow_2D(Imrect *full_vels[2],int pic_x,int pic_y,
                                  Imrect *Ix,Imrect *Iy,Imrect *It,
                                  int offset,int numpass,float alpha);
void calc_horn_averages_2D(Imrect *vels[2],Imrect *ave[2],
                           int pic_x,int pic_y,int offset);
void perform_horn_iteration_2D(Imrect *vels[2],Imrect *vels1[2],
                               Imrect *Ex,Imrect *Ey,Imrect *Et,
                               float alpha,int pic_x,int pic_y,
                               int offset,int it_no);
```

- Functions to draw lines, arrows, digits, labels, numbers, etc in rasterfiles and a function to superimpose the flow field on a rasterfile image. These are used to label images and draw the computed flow fields superimposed on the appropriate images.

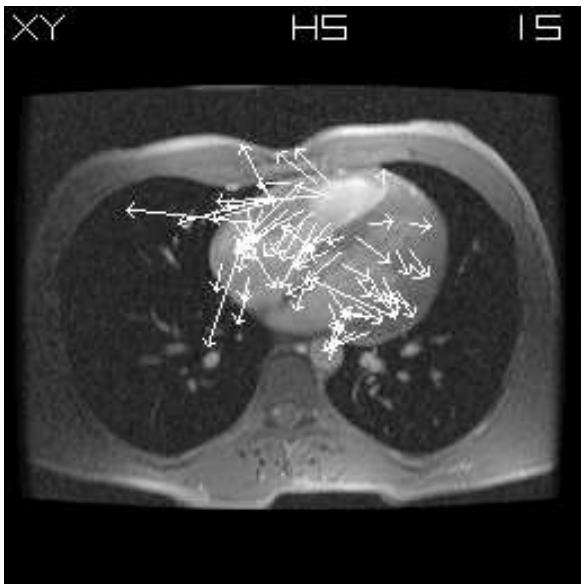
```
void draw_line(float x1,float y1,float x2,float y2,
               int pic_x,int pic_y,Imrect *flow_image,int offset);
void draw_arrow(float x1,float y1,float x2,float y2,
```

HS-XY-3-20

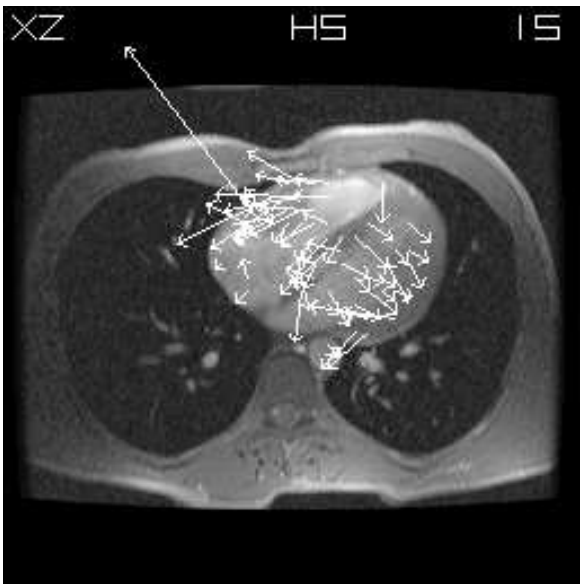


HS-XZ-3-20

Figure 23: The Horn and Schunck XY and XZ flow fields superimposed on the 20^{th} slice of the 3^{rd} volume of MRI data. $\alpha = 1.0$ and 100 iterations were used.



HS-XY-3-15 (10 iterations)

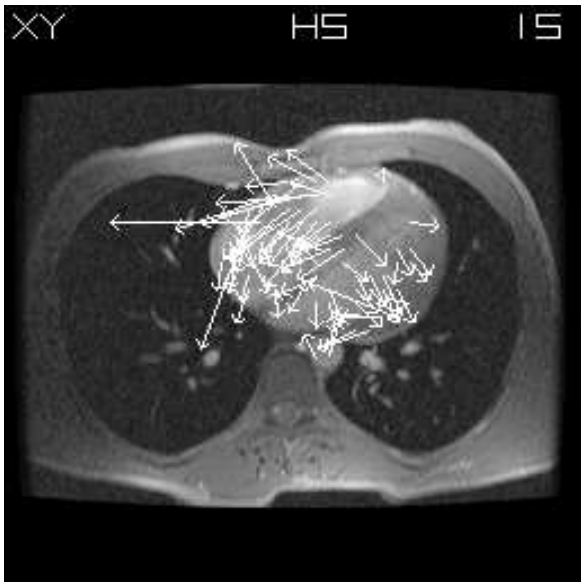


HS-XZ-3-15 (10 iterations)

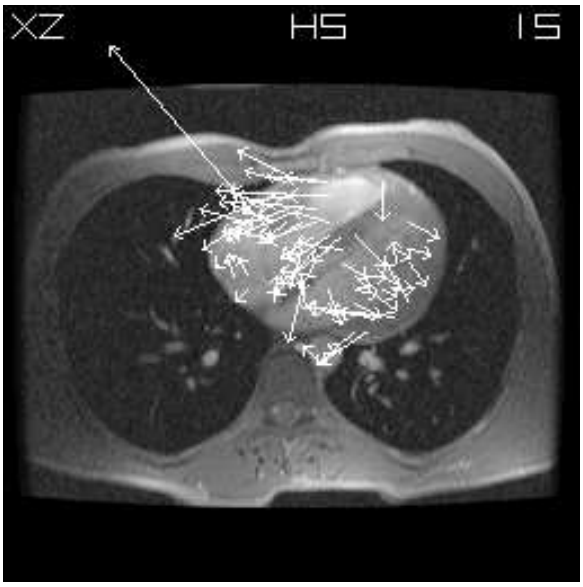
Figure 24: The Horn and Schunck XY and XZ flow fields superimposed on the 15^{th} slice of the 3^{rd} volume of MRI data 10 iterations for $\alpha = 1.0$.

```
                int pic_x,int pic_y,Imrect *pic,int offset);
void draw_label(Imrect *pic,int pic_x,int pic_y,float lx,float ly,
                float length,char label[10]);
void draw_digit(Imrect *pic,int pic_x,int pic_y,
                float lx,float ly,float length,int digit);
void draw_number(Imrect *pic,int pic_x,int pic_y,
                 float lx,float ly,float length,int number);
void arrow_rotate(float x,float y,float theta,float *a,float *b);
void superimpose(Imrect *vels[2],Imrect *pic,int pic_x,int pic_y,
                 int sample,float scale,int offset);
```

The **calc3Dflow.c** file contains all the 3D optical flow code. Both **calc2Dcode.c** and **calc3Dcode.c** use Numerical

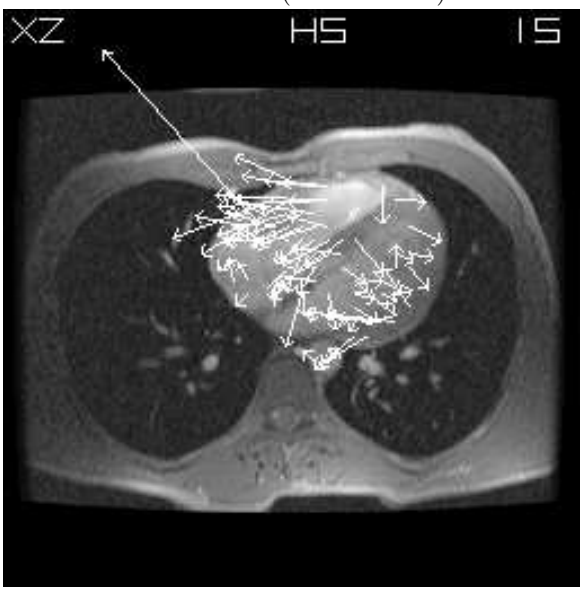


HS-XY-3-15 (25 iterations)



HS-XZ-3-15 (25 iterations)

HS-XY-3-15 (50 iterations)



HS-XZ-3-15 (50 iterations)

Figure 25: The Horn and Schunck XY and XZ flow fields superimposed on the 15^{th} slice of the 3^{rd} volume of MRI data for 25 and 50 iterations for $\alpha = 1.0$.

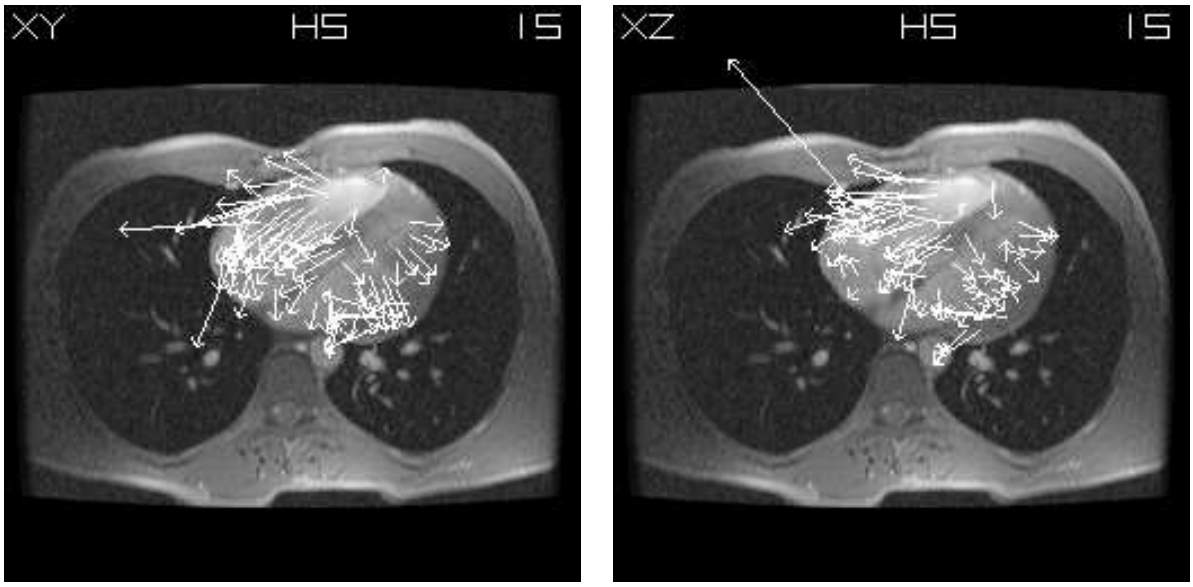
Recipes code for eigenvalue/eigenvector analysis and **calc3Dflow.c** uses some of the code in **calc2Dflow.c** (for example for labelling rasterfile images). The main functions in **calc3Dflow.c** are:

- Functions to read previously computed and write currently computed 3D derivative data.

```
void read_volume_derivatives(Imrect *Iders3D[DEPTH],int *pic_x,int *pic_y,
                int *pic_z,char in_filename[MAXPATHLEN]);
void write_volume_derivatives(Imrect *Iders3D[DEPTH],int pic_x,int pic_y,
                int pic_z,char out_filename[MAXPATHLEN]);
```

- Functions to lowpass (smooth) and highpass (differentiate) 3D volume data (actually 4D data in x, y, z and t).

```
void diff_in_x_3D(Imrect *Ix3D,Imrect *putting_y[DEPTH],
          float d_kernel[FIVE],int pic_x,int pic_y,int pic_z,int n,int k);
void diff_in_y_3D(Imrect *Iy3D,Imrect *putting_x[DEPTH],
          float d_kernel[FIVE],int pic_x,int pic_y,int pic_z,int n,int k);
```



HS-XY-3-15 $\alpha = 10.0$ HS-XZ-3-15 $\alpha = 10.0$

Figure 26: The Horn and Schunck XY and XZ flow fields superimposed on the 15^{th} slice of the 3^{rd} volume of MRI data for 100 iterations and $\alpha = 10.0$.

```
void diff_in_z_3D(Imrect *Iz3D,Imrect *putting_z[DEPTH],
          float d_kernel[FIVE],int pic_x,int pic_y,int pic_z,int n,int k);
void diff_in_t_3D(Imrect *It3D,Imrect *putting_t[FIVE][DEPTH],
          float d_kernel[FIVE],int pic_x,int pic_y,int pic_z,int n,int k);
void prefilter_in_x_3D(Imrect *putting_x,
                       Imrect *pre_in_t,float p_kernel[FIVE],
                       int pic_x,int pic_y,int pic_z,int n);
void prefilter_in_y_3D(Imrect *putting_y,
                       Imrect *pre_in_x,float p_kernel[FIVE],
                       int pic_x,int pic_y,int pic_z,int n);
void prefilter_in_z_3D(Imrect *putting_z,
                       Imrect *pre_in_y,float p_kernel[FIVE],
                       int pic_x,int pic_y,int pic_z,int n);
void prefilter_in_t_3D(Imrect *putting_t,
                    Imrect *floatpic[SEVEN][DEPTH],
                    float p_kernel[FIVE],int pic_x,int pic_y,int pic_z,int k);
```

- A function to compute 3D 1^{st} order derivatives, I_{x3D}, I_{y3D} and I_{t3D}.

```
void apply_Simoncelli_filters_3D(char stemname[MAXPATHLEN],
                                 char pathname[MAXPATHLEN],
                                 int start,int middle,int end,
                                 Imrect *images3D[SEVEN][DEPTH],
                                 Imrect *Ix3D[DEPTH],Imrect *Iy3D[DEPTH],
                                 Imrect *Iz3D[DEPTH],Imrect *It3D[DEPTH],
                                 int pic_x,int pic_y,int pic_z,int n);
```

- A function to compute 3D Lucas and Kanade optical flow. τ_D is the smallest eigenvalue threshold. Plane and line normal velocities are not computed.

```
void compute_lucas_optical_flow_3D(Imrect *Ix3D[DEPTH],
                                   Imrect *Iy3D[DEPTH],
                                   Imrect *Iz3D[DEPTH],
                                   Imrect *It3D[DEPTH],
                                   Imrect *full_volume_vels[DEPTH][3],
                                   int pic_x,int pic_y,int pic_z,int n,
                                   float tau_D,int flag,
                                   int flow_number,
                                   char vels_name[MAXPATHLEN]);
```

- Three functions to compute iterative 3D Horn and Schunck optical flow, to perform two iterations at a time and to compute 3D velocity averages.

```
void compute_horn_optical_flow_3D(Imrect *full_volume_vels[DEPTH][3],
                                  int pic_x,int pic_y,int pic_z,
                                  Imrect *Ix3D[DEPTH],
                                  Imrect *Iy3D[DEPTH],
                                  Imrect *Iz3D[DEPTH],
                                  Imrect *It3D[DEPTH],
                                  int offset,int numpass,float alpha,
                                  char vels_name[MAXPATHLEN]);
void perform_horn_iteration_3D(Imrect *full_volume_vels1[DEPTH][3],
                               Imrect *full_volume_vels2[DEPTH][3],
                               Imrect *Ex[DEPTH],Imrect *Ey[DEPTH],
                               Imrect *Ez[DEPTH],Imrect *Et[DEPTH],
                               float alpha,int pic_x,int pic_y,int pic_z,
                               int offset,int it_no);
void calc_horn_averages_3D(Imrect *full_volume_vels[DEPTH][3],
                           Imrect *ave[DEPTH][3],
                           int pic_x,int pic_y,int pic_z,int offset);
```

The last file is **flowtool.c**, which contains all the 2D and 3D tinatool GUI routines. Although, this was built using tinatool, version 4.0.2, it follows the spirit of tinatool, version 5.0, in that no functions in **calc2Dflow.c** or **calc3Dflow** call any function defined in **flowtool.c**: the GUI and library function use is completely separated. This code has been integrated into NeatVision[10] using tinatool 5.0 with minor modifications (due to using Microsoft C++). In more detail, **flowtool.c** contains:

- Functions for allocating/de-allocating 2D and 3D data structures, updating **tw_sglobal** pointers for updating fields on the GUI windows, and callback routines for reading 3D volume data, reading previously differentiated data and reading previously computed Lucas and Kanade or Horn and Schunck optical flow fields (extra thresholding can be performed on these flows when they are read in as outlined above).

```
void deallocate2D(void);
void deallocate3D(void);
void update_ptrs1(void);
void update_ptrs2(void);
void read_volume_image_sequence1(void);
void read_image_sequence1(void);
void read3Dimages1(void);
void read3Dders1(void);
void read3Dvels1(void);
void read3Dvels2(void);
void compute_lucas_optical_flow1(void);
void compute_lucas_optical_flow2(void);
void compute_horn_optical_flow1(void);
void compute_horn_optical_flow2(void);
```

- Functions to display individual slices in individual volumes in a TV, to display a slice movie for a particular volume and to display a volume movie for a particular slice.

```
void display_image(int volume_number,int slice_number);
void slice_movie(int slice_number);
void volume_movie(int volume_number);
void display_all_LK_HS_3D(void)
void display_one_LK_HS_3D(void)
```

- A function to run the 2D optical flow tool.

```
int flowtool2D_main(int argc, char **argv) /* 2Dflowtool */
```

- Functions to run the 3D optical flow and 3D movie tool windows, including a dialog box in the 3D optical flow tool to modifying input velocity thresholds.

```
int flowtool3D_main1(int argc, char **argv) /* 3Dflowtool */
int flowtool3D_main2(int argc, char **argv) /* 3Dmovietool */
void parameter_dialog(void) /* Dialog box */
```

- Functions to save and display 2D optical flow superimposed on rasterfiles.

```
void save_and_displayLK_2D(void)
void save_and_displayHS_2D(void)
```

- Functions to save an individual slice of the current 3D volume set, to save the XY and XZ flow superimposed on current slice and to manipulate the current slice/volume numbers (these allow a user to browse through the data, including the data with the superimposed XY and XZ flow fields).

```
void save_current_XY_and_XZ_flowimages(void)
void save_current_volume_image(void)
void add_1_to_slice()
void subtract_1_from_slice()
void add_to_volume()
void subtract_1_from_volume()
void add_1_to_slice_with_velocities()
void subtract_1_from_slice_with_velocities()
```

The **tinatool.c** code was also modified to have 3 additional buttons for 2D and 3D optical flow and 3D movie display, which call the appropriate functions in **flowtool.c**. Makefiles for both linux (Redhat 7.2) and Unix (Solaris) are included in the tarball, along with the *.cls files for 2D and 3D optical flow setups for both SUNs and PCs. Two additional files, **envPC** and **envSUN** give two sets of typical environment variables used for PC and SUN installations of tinatool (these might be useful if one wishes to see what environment variable values I used in my tinatool instalment). A gzipped tar file of all the data, programs and this writeup is available at

www.eeng.dcu.ie/whelanp/osmia/JohnBarron.tar.gz.

Tinatool source code and documentation can be downloaded from:

www.niac.man.ac.uk/Tina/tina_soft.html

4 Tinatool: An Assessment from a User's Point of View

From an user's, rather than a developer's point of view, tinatool proved quite useful for getting a C program running with a nice Graphics User Interface in X windows. The program described here was compiled on a Sparc 10 workstation using both the Xview and Motif versions of tinatool and on a PC (laptop) running Redhat 7.2 Linux and the Xview version of tinatool. The use of Tinatool's interface capacities allowed a very fast, but non-trivial, implementation of an interface. The use of dialog boxes, action buttons and labels made the interface easy to design. It would have been nice if pop up menus were available (my understanding is that it could be made to work under Motif and not Xview and unless a feature worked under both versions of X windows it was not included).

The dynamic storage capabilities, including the stack and **imcalc** proved easy to use. Dynamic image/volume structures were maintained using **Imrect**s. In addition to the GUI library routines, I also used a small number of simple tinatool library functions, including **im_alloc**, **im_free**, **stack_push**, **stack_pop**, **stack_inspect**, **im_copy_inplace**, **im_sqrt**, **imcalc_draw** and **im_scale_range_inplace**. I did not need to use any of tinatool's algorithms, such as stereo, edge, camera calibration, corner detection, matching/correlation, etc. in my coding. As suggested by Neil Thacker at our last research meeting in Dublin, I used **im_sqrt** to perform a gamma-like correction on the MRI images for display purposes. Tinatool has a gamma correction function which I could not get to work (but then I couldn't get my own gamma correction code to work either [and it has always worked for other images before!!!]) on the MRI data as the dynamic range has grayvalues too skewed towards dark.

I found I had to write 3 functions:

```
void my_copy_ushort_to_uchar(Imrect *out,Imrect *in);
void my_copy_float_to_uchar(Imrect *out,Imrect *in);
void my_copy_uint_to_uchar(Imrect *out,Imrect *in);
```

because if I used tinatool's **im_copy_inplace**/**im_copy** functions with different data type I got casting warnings. With these 3 simple functions, my tinatool code compiles without any warnings.

Rather then use existing Computer Vision algorithms implemented in Tinatool, I saw my task as adding new functionality to tinatool. Hence, my main contributions are the 2D and 3D differentiation routines and the 2D and 3D optical flow routines. To my knowledge, no one is doing 3D optical flow for gated MRI cardiac data in quite the way I am (and I believe I ultimately will be more successful that the current approaches [ever the optimist!!!]). This work falls nicely under the OSMIA umbrella.

I can make one major criticism: the documentation was lacking. It was often difficult to find out what functions were available (one had to search libraries in the **src** directory for examples). Only later in the project did I start using the web search tool for tinatool[3], which made my task much easier. I also found the 4 research meetings we had (in Dublin, Manchester, Edinburgh and again in Dublin) to be quite useful; my current problems were always solved at those meetings. I realize extensive documentation requires a lot of time and effort (and frankly is probably more than a bit boring to do) but without it things can be difficult to figure out. The learning curve could be significantly reduced with good documentation. The documentation provided also needs a glossary of terms with page numbers where they are described in the text.

Now that I have made the effort in learning Tinatool I will continue to use it. The program described here will be extended to have a 3D version of Nagel's oriented smoothness optical flow [6, 7, 8, 9] and new, spline-based methods of computing 3D intensity derivatives will be added over the coming year.

My results on computing 3D optical flow using tinatool will be the subject of 1-2 conference papers. At those conference, I will disseminate information about tinatool. I could not do so before as I was still learning tinatool myself!

References

[1] J.L.Barron, Tutorial: Computing 2D and 3D Optical Flow. Tina Memo 2004-012.

[2] S. Pollard, J. Porrill and N. Thacker (1999), "TINA Programmer's Guide", Medical Biophysics and Clinical Radiology, University of Manchester, UK. (`www.tina-vision.net programmers_guide`).

[3] B.K.P. Horn and B.G. Schunck, "Determining Optical Flow", *Artificial Intelligence*, **17**, 1981, pp185–204.

[4] B.D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", DARPA *Image Understanding Workshop*, 1981, pp121–130 (see also *IJCAI'81*, pp674–679).

[5] J.L. Barron, D.J. Fleet and S.S. Beauchemin, "Performance of Optical Flow Techniques", *Int. Journal of Computer Vision*, **12**, 1994, pp. 43–77.

[6] Nagel H.H. (1983), "Displacement vectors derived from second-order intensity variations in image sequences", *CGIP* 21, pp85-117.

[7] Nagel H.-H. (1987), "On the estimation of optical flow: Relations between different approaches and some new results", *AI* 33, pp299-324.

[8] Nagel H.-H. (1989), "On a constraint equation for the estimation of displacement rates in image sequences", *IEEE Trans. PAMI* 11, pp13-30.

[9] Nagel H.H. and Enkelmann W. (1986), "An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences", *IEEE Trans. PAMI* 8, pp565-593.

[10] Whelan P.F. and Molloy D. (2000), "Machine Vision Algorithms in Java: Techniques and Implementation", Springer (Visit www.NeatVision.com/ for details on downloading and installing the software).

[3] www.niac.man.ac.uk/Tina/, click on Documentation and then click on online. Use the 3 search facilities from there.