TIM for Windows Demo Program

Welcome at the TIM for Windows Demo program. This program will give you an overview of TIMWIN's features: a self running command file program, written in TIMWIN's powerful new language.

Before continuing, you may want to look around in the Demo Program's Help, in order to learn what can be expected.

To inform yourself, please consult the following items:

- System setup
- Running the Demo
- Using Help

If you want to start the demo, press the **OK** button in the Command File Specific Help Message Box.

System setup

You system configuration determines what can be seen during the demos. It can be one of the following:

You have a frame grabber:

In this case most of the image processing takes place in four images (**p, q, r** and **s**), located in the frame grabber window. This gives you a detailed insight in what happens (you can observe the pixels as they are modified), and also provides the fastest display method.

You don't have a frame grabber and you chose the standard image set-up:

In this case image processing takes place invisibly, but from time to time results are copied to two Windows images (**h** and **i**). On display systems with only 16 colours (e.g. standard VGA) grey values are simulated using dithering. Due to the nature of some functions the system may seem asleep now and then. This mode takes longer than the frame grabber mode, described above, and doesn't provide as much insight in what happens.

You don't have a frame grabber and you chose the extended image set-up:

In this case the work images in the demo (**p, q, r** and **s**) have a display window attached, so every action in these images will be displayed.
This setup is useful only if you have a Super VGA video adapter (at least 800x600, better yet 1024x768) at 256 colours, and sufficient memory.
Since display involves copying if the image, this mode is slower than the frame grabber mode, while you see less operational details. For example, when producing the skeleton of an object, you only see the end result, while, in a frame grabber image, you'll see layer after layer being removed.
Nevertheless, this setup is recommended if you have no frame grabber and the hardware requirements are satisfied.

See also: setting up images

Running the Demo

The demo consists of a TIM program (a set of compiled command files). You start the program by entering `*demo1` on the command line in the main window.

The Demo program gives you a glimpse of TIMWIN's facilities for processing images and deriving data from them.

Every demo is accompanied by a help page, that describes the demo.

Continue by selecting one of the following:

- Configuration
- Organising your screen
- User interaction
- Image display
- Interrupting the demo
- Debug mode

Configuration

Depending on your system's capabilities, you may want to select or disable features during the demo. See the following items :

- Available memory
- Setting up images
- Frame grabber
- Screen resolution
- Floating point coprocessor

Available memory

The standard TIMDEMO allocates about 628 KBytes of memory for images if a frame grabber is present, and 1,2MBytes without a frame grabber.

If your system complaines about insufficient memory during startup, the current setup may be inappropriate to run the demo. In this case, see setting up images and select one of the **imgmin** files.

Setting up images

In TIMWIN, image organisation and set up is determined by a file: **images.tim.**
The demo package has several prototype files for various situations. Changing image set up is a matter of copying one of the **images...** file to **images.tim**
To change the image setup, do the following:

- Exit TIMWIN: from the **File** menu click **Exit**
- From the MS-DOS shell or from Windows' File Manager copy one of the files **IMAGES.xxx** or **IMGMIN.xxx** to **IMAGES.TIM**. (xxx stands for one of the available versions). To find out which file to use, select a file from the table below.
- Restart TIMWIN. It will come up in the new configuration.

Image Description Files Table

If you have FG	**little memory**	**sufficient memory**	**high resolution display**
Cortex-I	IMGMIN.CX1	IMAGES.CX1	idem
PCVision	IMGMIN.PCV	IMAGES.PCV	idem
PCVisionPlus *)	IMGMIN.FGS	IMAGES.FGS	idem
PCVisionPlus *)	IMGMIN.FGD	IMAGES.FGD	idem
VS100	IMGMIN.VFG	IMAGES.VFG	idem
VFG	IMGMIN.VFG	IMAGES.VFG	idem
No frame grabber	IMGMIN.NOF	IMAGES.NOF	IMAGES.PQR

*) The PCVisionPlus can be either in Single store or in Dual store mode. See the PCVisionPlus manual (page 2-14). TIMWIN recommends using Single store mode.

Frame grabber

A frame grabber is an extension board in your computer. It contains memory, which is used by TIMWIN to store images. The content of this memory can be observed by connecting a separate colour monitor, and also a video camera can be connected to grab images. Frame grabbers have useful features like display look up tables, hardware zoom, etc.

On systems having a frame grabber, image processing can be followed in detail. If desired, the results can be copied to windows images as well.

If no frame grabber is present, image processing takes place invisibly in computer memory, and only now and then some intermediate results are copied to an image window.

Screen resolution

Your Windows display has the following properties:

- Horizontal and vertical resulution (number of pixels)

  The higher the horizontal and vertical resolution, the better. If you have only standard VFG (640x480) you may find it difficult to arrange all windows on the screen. The use of Super VGA (800x600 or 1024x768) is strongly recommended. See your adapter's documentation for possible resolutions.

- Pixel resolution (number of colours)

  The pixel resolution may be 16 colours (standard) or 256 colours (recommended). If you have a 16 colour display system, grey values are displayed by using a dithering technique. On a 256 colour display system, display quality of windows images is the same as frame grabber's.

Floating point coprocessor

Some functions in TIMWIN internally use floating point arithmetic. For these operations there will be a speed advantage for systems that have a floating point coprocessor.

There are a few other TIMWIN functions that are very floating point intensive. Executing these functions on a system without a coprocessor is not recommended for reasons of speed.

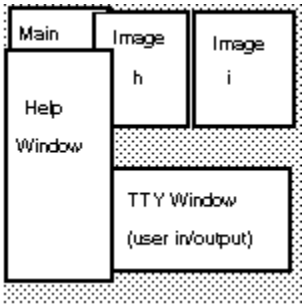
The demos that contain floating point intensive operations are separated from the rest in a special menu selection. They include: FFT, geometric transformations and the generation of Mandelbrot fractals.

Organising your screen

The large amount of windows can make your screen look a little cluttered. It is easy to get lost in an ill-organized screen. Therefore we advise the following:

- keep the windows as small as possible
- reduce windows to icons, if they are temporarily not of interest to you
- close applications that are not in use. For example, closing the Status Window will speed execution of the program

The following figure gives a suggestion for a useful screen set-up.



Since the Main Window is not needed during the demo, its space can be used by another window (here: Help).

Usually the windows overlap each other. Avoid overlapping with image windows, since this may lead to time consuming repainting of windows images.

User interaction

For command file input and output a single window is used: the TTY window. This is a simple text oriented window. This system is chosen to maintain compatibility with TIM for DOS. In a later version menu's, dialog boxes etc. will be supported.

In the Demo program you specify your choices by entering a number at the prompt. To inform you that the system expects a reaction from you, the menu caption shows the text "Waiting for input"

Image display

The images, that result from the operations can be shown as follows:

- In a frame grabber image. This selection is the default if you have a frame grabber.
- In a Windows image. This selection is default if you don't have a frame grabber. However, you may want to select this option if you do have a frame grabber.

If you use a Windows image for display, you can select either 16- or 256 colour display.

- 16 colours in this mode grey value images are dithered
- 256 colours in this mode images are displayed without any transformation

In both cases colours are used to enhance image properties, if necessary.

For further information, see

- Screen size
- Grey value display

Interrupting the demo

To interrupt the demo program, do the following:

1. Select the TTY window by clicking on it.

2. Press the Escape key. A dialog box comes up which offers you to:
    - Continue in debug mode
    - Quit the current file
    - Quit all
    - Continue without any change

3. Select an item and press OK

Or, in the TTY window, click the Break menu item

Debug mode

TIMWIN has a strong debugging facility, which allows you to test a program by running it line by line, while observing images, variables, etc.

You can run the Demo program in debug mode, to get more insight in the internals of a TIMWIN program.

To start TIMWIN Demo in debug mode from the beginning, enter

```
*demo1 debug
```

on the command line in the main window.

To continue in debug mode from any stage in the running Demo program, do the following:

1. Select the TTY window by clicking on it.
2. Press the Escape key.
3. In the pop up dialog box select the Debug radio button
4. Press OK

Or click the **Debug** menu item in the TTY window

For details on using the debug facility, consult the appropriate titles in TIMWIN Help

Using Help

In any stage of the Demo program you'll see these help windows popping up, that contain information about the current demo. After reading the information, you can press the **OK** button in the message box.

In the Demo Set Up part you can choose not to be prompted for new help pages. This will make the demo to continue without interruptions.

You are not limited to the selected part of Help. The entire help file is accessible all the time. You can use all the features of Windows Help for accessing the file: keywords, the index, browse through the information, etc.

The Demo Help system consists of two parts:

- TIMDEMO, which you are looking at right now. It contains help about the TIM demo program
- the standard TIMWIN Help. This contains general information about TIMWIN. To invoke it, or press the help button in one of the TIM windows.

You can freely switch between the two help files.
To consult the standard help file:

- select the **Help** menu in TIMWIN's main window,
- Or press a help button in any of the TIM windows.

To consult the TIM Demo Help:

- continue the demo program by selectin an item in the TTY window. Soon a new help page will come up. Or, press F1 while a demo is running.

Notice that the TIMDEMO help window disappears when the TIM Demo Program finishes. TIMDEMO Help is not accessible without running TIMDEMO.

Screen size

The following screen sizes are common:

- Standard VGA 640x480
- Super VGA 800x600
- Super VGA 1024x768

The recommended screen resolution for TIMWIN is 800x600 and higher.
A small resolution (like standard VGA) can make it difficult to arrange the necessary windows.

Tip: you can reduce the size of the window images to 128x128 by clicking on them and selecting another resolution. However, some details may become invisible in a reduced resolution.

Grey value display

Generally, the images are 8-bits grey value images, that are capable of showing 256 grey values. To display an image correctly (showing 256 grey values) in a window, you must have an appropriate graphics adapter. Also, a correct driver must be installed.

If a grey value image displays crude, with only four grey values visible, then you have a 16-colour adapter or -driver. You can then select an option that simulates grey values using dithering (see the description of the **dot** and **sdot** commands)

See the documentation of your graphics adapter for information on grey value resolution.

Note: some display adapters can be very slow in 256 colour mode, especially when scrolling text. For this reason an 'accellerated' adapter is recommended.

Systems with a frame grabber

Most of the image processing demonstrations run on a system that has a frame grabber. In these systems observation of intermediate results is efficiently achieved, which makes them suitable for a demonstration.

In the demo the correct functioning of the frame grabber is assumed. Please perform the tests that are described in the installation directives to check whether your frame grabber functions correctly.

Systems with no frame grabber

The image processing demonstrations on a system without a frame grabber are somewhat limited, because of the less efficient image display: images are copied to the display after the operation. Not all intermediate results are shown.

The TIM image set-up includes two windows images of size 256x256. You need a proper display adapter to be able to usefully display images.

See also:

- grey value display
- window size

Windows in TIM

TIMWIN has the many windows, each window performing a task. Only one window can be active at a time. An active window is recognised by the coloured caption (top of the window containing the name).

Depending on the flow of control the active window changes from time to time. The user can make a window active by clicking on it.

Sometimes the system expects a response in a window. When the user made another window active, the system seems to stop reacting. You can continue by selecting the correct window and enter the required response.

The following windows exist in TIMWIN:

- TIM's Main window
- The TTY window
- The Status window
- The Statistics window
- The Graphics window
- The Image edit window
- The Ibuf edit window
- The Filter window
- The Debug Window
- The Watch window

TIM's Main window

The main window functions as an input window for entering commands.
It also contains the main menu bar, that gives you access to TIM's menus and dialog boxes.
System messages and function return values are written into this window.

The TTY window

The TTY window is the interface between command file programs and the user. There are several functions to write to this window or to get input from it.

The Status window

The status bar contains several controls that allow you to select images, look up tables, observe and change cursor positions, sub image size, etc.

You can use also the status bar to connect the mouse from Windows to the image cursor.

- In the Status Window, press the Cursor button to control the windows cursor.
- To switch back to Windows, press the mouse's right button

Note: controlling an image's cursor can only be done with frame grabber images.

The Statistics window

The statistics window shows several statistic values, derived from an image's histogram. You can choose to update the statistic window's content automatically with each image processing operation by setting the Update flag in the Update menu.

The Graphics window

The graphic window shows the following data graphically:

- the content of Ibuf
- the histogram of an image

You can select either of these, or both.
You can choose to update the graphic window automatically with each image processing operation that changes the involved data by setting the Update flag in the Update menu.

The Image edit window

The image edit window shows a selected part of an image numerically. You can edit the values and change the selected area.
You can choose to update the image edit window automatically with each image processing operation that changes the involved data by setting the Update flag in the Update menu.

The Ibuf edit window

The image edit window shows a selected part of the Ibuf buffer. You can edit the values and change the selected area.
You can choose to update the Ibuf edit window automatically with each image processing operation that changes the involved data by setting the Update flag in the Update menu.

The Filter window

The filter window allows you to enter values for convolution kernels. You can indicate horizontal and/or vertical symmetry to reduce the number of items to add.

The Debug Window

Use the debug window when debugging command files. It contains functions to single step the program, watch variables, set breakpoints, etc.

The Watch window

When in debug mode, this window shows the value of selected variables.

Finally ...

This is the end of the TIMWIN demo program. If you have questions or comments on this program, don't hesitate to contact us.

DIFA bv.
PO Box 3132
4800 DC Breda
the Netherlands
tel.: (31)76-710144
FAX: (31)76-711953

If you want to discuss technical matters with the developers, please use Email or fax.

ekkers@ph.tn.tudelft.nl
fax no. (31)15 626752, attn. TIM development team

Holo

This demo uses a holographic interferometry image of an aeroplane part under stress. The fringes show the distribution of force. In the demo you'll see:

- how shading can be removed from an image
- further pre-processing using binary and cellular logic operations

Removing shading

Shading is an uneven distribution of foreground and background grey values in an image. Shading prevents the image from being correctly segmented by thresholding. This is shown in the first part.

When the circumstances during image acquisition don't permit shadingless recordings, removing of shading must take place afterwards. In this demo minimum- and maximum filters are used for this purpose.

First a sequence of threshold operations using several threshold values demonstrates the effect shading has. Then correction takes place.

1. A minimum filter operation, followed by a maximum filter produces an accurate estimation of the background shading (which has an *additive* character)

2. A maximum filter operation, followed by a minimum filter, produces an image which, after subtraction the result of 1. , produces an estimation of the local contrast. Differences in local contrast are caused by *multiplicative* shading.

Finding fringes

After pre-processing the information in the image must be extracted. This information is hidden in the fringes. In this sequence the goal is:

- to segment the image using thresholding
- to remove noisy image parts outside the object
- to fill small holes by growing line parts together

Parts

In this demo a bunch of electronic components are recognised. Recognition takes place for each object individually.

First some pre-processing is done in order to be able to access individual objects. Then the objects are analysed. The algorithm looks at:

- object size, by eroding it a fixed number of times. If something is left, it is a big object.
- number of end points. This is a nice measure to distinguish electronic components (end point = leg)

After retrieving these values, classifying the objects is a matter of elimination.

This demo shows the power of CLP operations for determining properties of objects.

Rnoise

In this demo the percentile filter is demonstrated. This filter removes certain types of noise, without blurring the image too much.

First some shot noise is added to a standard image: 20% of the pixels are replaced by either 0 or 255.

Then the percentile filtering is performed, with window size 3. Most noise pixels are replaced by better ones, but some noise pixels remain. This is because in these places the noise is too much concentrated.

To get better results two methods can be used:

- perform the 3x3 filter on the first result
- use a larger filter, e.g. 5x5

The last method is used in image **r**.

Label

This demo gives an example of object analysis based on shape.

In the firs stage the image is segmented by thresholding and labelled, to be able to access the objects individually.

The following properties of each object are measured

- area (= number of pixels)
- perimeter (producing the contour and then measuring its length)
- maximum diameter (the two contour points with the largest mutual distance)

Area and perimeter are also combined in a shape parameter, using the formula

$$\frac{perimeter*perimeter}{4*pi*area}$$

This value is a minimum (1.0) for circular objects.

Switch

This demo program performs a good/bad test on industrial objects. The program consists of the following two main parts:

- finding the exact location of an object in the image, and isolating it
- performing the test on the object

The test is done on the inverse of the image - in fact the holes are analysed. This is more efficient, because the holes have a much simpler form. In this demo, the number of holes is sufficient for analysis.

The object to be tested is located between the four "balls". These can be found rather easy by labelling. A sub image containing the entire object is determined by the first and the last object (upper left and lower right ball).

Labelling the resulting image returns the number of objects (holes). The first run is performed on a known good object. The resulting number is then used to check the other objects.

An image containing all 8 objects shows the result of the test: green is good, red is bad.

Colour

TIM has several possibilities of showing real colour images.

- on an 8-bits frame grabber with look-up tables the available 256 colours can be mapped so that a useful number of intensities for each colour is created. This is, for example: 6 levels for red, 7 levels for green and 6 levels for blue (6x7x6 = 252)
- on a 12-bits frame grabber the same technique can be used, but here 16 levels are available for each colour (16x16x16 = 4096)
- in a Windows image with the possibility of showing 256 grey levels (Super VGA) the same technique can be used
- in a Windows system with the standard number of colours (16) another system must be used - dithering. Here pixels are written using full intensity, but varying the space between the pixels is used to simulate intensity.

256 colours (8 bits frame grabbers)

Each colour image consists of three: a red, green and blue one.

First the respective images are reduced to the available number of levels (6 or 7) and scaled to the grey values corresponding with the colour map. This is:

- for blue: 0, 1, 2, 3, 4, and 5
- for green: 0, 6, 12, 18, 24, 30, and 36.
- for red: 0, 42, 84, etc.

Then the images are added.

4096 colours (16 bits frame grabbers)

Each colour image consists of three: a red, green and blue one.

First the respective images are reduced to the available number of levels (16) and scaled to the grey values corresponding with the colour map. This is:

- for blue: 0, 1, 2, 3, 4, etc. up to 15
- for green: 0, 16, 32,etc. up to 240
- for red: 0, 256,512 etc.

Since TIMWIN only handles 8 bits images, the red part is made by writing a 4-bits image into the overlay image of the frame grabber.

The blue and green image are added as usual.

256 colours (Super VGA)

Each colour image consists of three: a red, green and blue one.

First the respective images are reduced to the available number of levels (6 or 7) and scaled to the grey values corresponding with the colour map. This is:

- for blue: 0, 1, 2, 3, 4, and 5
- for green: 0, 6, 12, 18, 24, 30, and 36.
- for red: 0, 42, 84, etc.

Then the images are added.

Dithering (all systems)

Each colour image consists of three: a red, green and blue one.

The individual images are dithered - grey values are simulated by placing dots of a fixed intensity on varying distances. By giving the dots of the three images the primary colours, and OR-ing the three images together, a reasonable colour image is achieved.

TIMWIN uses a unique dithering algorithm, that avoids the annoying patterns that often show up with dithering.

Fill

Filling holes in objects is often used in image processing. This demo shows two methods to do this:

- filling each object individually
- filling all the objects in the image in one pass.

In both cases the propagation operation is the main operation.

Distort

This demo shows how you can break up an image and put it together again.

You can take the image apart using several line drawing techniques. Then, you store the pixels laying 'under' the line in Ibuf and write the buffer into another image.

In the first example a line is moved through the image.

In a second example a vector is drawn repeatedly from a single starting point.

Important operations are: **rdln, rdvec, ihis**

Maze

This demo shows a surprising application of the CLP skeleton operation: finding a path through a maze.

The skeleton operation thins the path globally, until a single pixel line remains. The version used in these demo, also 'eats' the end pixels. Since the end pixels represent dead paths, these are automatically removed. The result is the single path, that connects start and finish.

Distrans

The distance transform operation has some unique properties. In this demo, the operation is used for segmentation.

The goal is to segment a number of Dutch coins. The total amount of money must be determined. The distance transform is fast, insensitive to objects touching or overlapping each other, and can also be used for measuring.

FFT

The Fast Fourier Transform is used to transform images into the frequency domain and vv. In the frequency domain analysis and filtering can take place, that have no equivalent in space domain.

In this demo an image is transformed, the frequency image is shown, it is filtered and transformed back.

This demo needs a complex floating point image (64 bit/pixel). In the standard image set-up this image is called: f.
Press the Change button in the status window to see the available images. To bring up the status window, press ALT-S.

Note: This demo is not recommended if your system does not have a mathematic coprocessor (80X87)

1. Converting the pixels to complex floating point

First the 8 bits integer pixel representation must be changed to 2x32-bits complex floating point.

2. Transforming the image into the frequency domain

The pixels representation is complex now, but the image still represents a space domain image. This operation performs the transformation to the frequency domain. Even on systems equipped with a coprocessor this is a time consuming task. It may take about a minute on a reasonable fast system.

3. Showing the modulus of the FFT image

Now the transformation is ready, it may be nice to look at the result. Since the FFT image has a very large range of values, we'll compress the values using a logarithmic function.

You'll see an image where each pixel represents a frequency.
The centre represents frequency 0. To the outside of the image the frequencies increase.

In the example you see 'arms', that represent the fringes in the original image.

4. Filtering in the FFT domain

You can suppress or enhance selected frequencies by creating a mask, and multiplying the Fourier image with it. In this example we'll use the displayed result for this purpose.

After some thinning of this pattern we'll multiply this image with the fft-image. The operation used accepts a standard 8-bits image.

5. Transforming the filtered image back

After the filtering the FFT image must be transformed back to the space domain. Again this will take some time ...

6. Displaying the filtered image

Now the image is transformed back to space domain, we can show the result. However, since the floating point image now contains values in the original range, no compression is necessary. We'll use the unmodified modulus.

In the image you see an enhancement of the majority of the fringes, and a suppression of the deviating patterns.

You are encouraged to play with different filter patterns to obtain better results.

Fractals

Mandelbrot fractals produce fascinating images. In this demo a random set is chosen. You can explore this world by experimenting with other coordinates, magnification factors and display look-up tables.

You can benefit from the multi tasking capacities of Windows by having TIMWIN calculate a fractal for you, while doing something else.

Note: This demo is not recommended if your system does not have a mathematic coprocessor (80X87)

Abingdon Cross Benchmark

The Abingdon Cross benchmark is developed by prof. Kendall Preston Jr. in order to have a method to compare image processing systems. Although everybody is aware of the limitations of benchmarks, they may be useful to supply a coarse platform of comparison.

The source image consists of a noisy cross. Goal is to retrieve the skeleton of the cross.

The image is created during the first part of this demo. Gaussian noise is obtained by adding random noise images (as produced by the **noise** operation) together in a 16-bits image. After scaling back the noise image into an 8-bits image, the cross is created by adding 32 to the pixels in a horizontal and a vertical sub-image.

The benchmark itself starts when you finish reading this text (press the OK button). It consists of three operations:

```
unif 33      ;a uniform filter with a size of 33x33 effectively removes the noise, and leaves a 'ridge',
             that is already close to a skeleton. The size 33 seems to be optimal in relation to the size
             of the cross (which is 32 pixels wide)
thre 156     ;the threshold operation just leaves the top of the ridge
lsk 1        ;the skeleton is produced in bitplane 1 (red)
```

It is performed twice: once in display memory (slower) and once in fast (but invisible) memory. The actual benchmark takes place between the two beeps.