A LON publication

GScnd v1.24 User Manual

Written by J.Romaya W.D.C.N.

23rd October 2002

Terms and conditions

Cogent Graphics is provided to you free of charge under the following conditions:-

1) Acknowledgement

Acknowledgement from users helps us justify the time we are spending further developing and maintaining this free software. Therefore we request that, when you use Cogent Graphics for your experiments, you include the following statement in your publication:

"This experiment was realised using Cogent Graphics developed by John Romaya at the LON at the Wellcome Department of Imaging Neuroscience."

2) Copying

This package may be distributed freely as long as it is distributed in its original form. It may not be sold without permission.

3) Liability

The package is distributed as it is, without any warranty implied. No liability is accepted for any damage or loss resulting from the use of these routines.

Cover - The Three Graces (1814-1817) by Antonio Canova (1757-1822).

In Greek mythology the Three Graces, Euphrosyne, Aglaia and Thalia, bestowed joy, beauty and elegance on mankind. Please try to bear this in mind when you design your experiments.

Table of Contents

Introduction

Known bugs and limitations

1/ When using more than two monitors try to use the rightmost and bottommost screen for the cogent display. Otherwise the cgMouse() function may behave strangely.

2/ The cgMouse() function has not yet been fully integrated with the rest of cogent. In particular there may be undesirable interactions between this function and cogent logging of mouse events. Use this function with caution.

3/ Multiple displays have not yet been tested with the rest of Cogent. It is possible that there could be some problems.

4/ The playing of movies is not supported in palette mode.

5/ When running a tight animation loop on a dual display system Ctrl+C does not always terminate the script. If this happens it is necessary to invoke the task manager with Ctl+Alt+Del to clear the hang.

Update history

Changes v1.23 to v1.24

1/ A bug has been fixed in the cgFlip() function. This is now synchronized properly with the display. In previous versions the display lagged behind this command by a period of one frame. The bug has been fixed in the underlying GPrim library. The command has also had another change made which makes it much more reliable with regard to dropped frames. See the new section in this manual on “Display timing”.

2/ New instructions in the manual explain how you can use the cgopen command to select high refresh rates up to 160Hz. It is possible to select refresh rate directly through the cgopen command providing your graphics card supports it. See the revised section on "Direct X and Refresh Rate" and the "Opening and closing graphics" item in the "Direct colour tutorial" section.

3/ A recording function has been added to the eyetracker commands, you can now use 'start' and 'stop' to retrieve a sequence of values. You can also set the background and foreground colours of the calibration screen.

4/ The sample scripts have been reworked and are now useful tools for testing timing. I recommend that you download the new samples when you download the toolbox.

5/ There was a bug in the cgEllipse and cgArc functions; a memory leak used up the system brush resources after a long period. This has now been fixed.

6/ A bug has been fixed in the cgopen command. Previously the Windows desktop appeared in the initial cogent display in certain circumstances. The display is now correctly cleared to black.

7/ A bug has been fixed in sub-window mode. When playing movies the movie is properly fixed in the display.

8/ A memory leak has been fixed in the movie-playing functions.

9/ The cgloadbmp function now accepts BMP files formatted to 1 and 4 bits per pixel.

10/ Audio and visual synchronisation problems fixed for playing of some movies.

Changes v1.22 to v1.23

1/ A new function has been added; cgTracker() communicates with the ASL Model 5000 eyetracker control unit. You can now read eye position in cogent screen co-ordinates in real time.

2/ A bug has been fixed in the serial interface code for the photometer interface. The cgPhotometer function should now function properly on all systems.

3/ A bug has been fixed in sub-window mode graphics. The cgFlip() command now swaps background and foreground buffers in sub-window mode.

Changes v1.21 to v1.22

1/ The cgopen function now accepts a bits per pixel (bpp) value of zero. When this is passed the display is opened in 32-bpp mode. If this is unavailable the display is opened in 24-bpp mode and if this is also unavailable the display is opened in 16-bpp mode. This gives a mechanism of opening a display when we are not sure what bpp values are available.

2/ The rgb sample scripts have been modified to utilise bpp=0. They first check that we are using v1.22 or later.

3/ The cgloadlib unload function crashed with Matlab v12.1; it of course unloaded itself (cgloadlib.dll) and then failed to return. The unload function does NOT now unload cgloadlib.

4/ A small bug has been fixed in gprim('gRectFill'). When called with no arguments it now functions correctly.

5/ A message-processing loop in gprim('gFlip') has been added so that graphic animations work better with Matlab v12.1. Specifically Alt+Tab can now be used to flip to the matlab console and some occasional flashes of white after a few seconds of running scripts that don't poll the keyboard or mouse no longer occur.

Changes v1.20 to v1.21

1/ The cgloadlib function which was previously implemented as a matlab script (.m or .p file) has now been implemented as a library (.dll file). This is because older versions of Matlab would not run the pcode file. The function can now also unload the libraries if requested. There should not be any conflict with previous undeleted copies of cgloadlib.m and cgloadlib.p because the .dll file takes precedence.

Changes v1.19 to v1.20

1/ A new function, cgBlitSprite(), has been added which can copy arbitrary rectangles from one sprite to another.

2/ A bug has been fixed in the underlying gprim libraries so that cgFlip() now operates correctly in immediate mode.

Changes v1.18 to v1.19

1/ A new function, cgArc() has been added which can draw hollow arcs or filled sectors.

2/ Minor bug fix - back buffer is cleared to black or palette index 0 when graphics are opened.

Changes v1.17 to v1.18

1/ Bug fix – there was a memory leak in the cgloadarray() function. This meant that memory resources were depleted each time the function was called. If the function was called many hundreds of times it would eventually use up all available memory resources and cause the system to seize up. This bug has now been fixed in v1.18.

2/ Two new functions have been added to the samples; the utility functions equil, which matches equiluminant colours by a flicker method, and PerfTest, which measures the graphics performance of your PC when using Cogent Graphics.

Changes v1.16 to v1.17

1/ Bug fix - the time in microseconds as returned by cgFlip() and cgNewPal() wrapped round to a negative value after 35' 47", an unacceptably short interval. These functions now return a floating point 'double' in units of seconds.

Changes from v1.15 to v1.16

1/ Bug fix - There was a bug in the functions that draw multiple lines and ellipses. This bug meant that the sample scripts starspal.m and starsrgb.m did not work under Windows 98. This bug has now been fixed.

2/ The cgLoadBMP() function has been changed. You may now set the required sprite width or height to be zero. The aspect ratio of the original bitmap is used with the other dimension to calculate the required size.

Changes from v1.14 to v1.15

1/ Bug fix - Sample script dartboard.m did not work on dual-monitor setup when both displays were set to direct colour mode (although it did work when they were set to palette mode). This bug has been fixed by changing CGLib_DD7::SetMode() although I do not see why the original version caused a problem. It may be that this bug crops up again on different systems.

2/ Bug fix - cgMouse was not working properly on dual-monitor systems. This has been fixed by adding a couple of variables to CGLib::m_Data (MouseOffsetX & MouseOffsetY) and by modifying CGLib::Mouse(), CGLib_DD7::MouseProc() and CGLib_DD7::SetMode().

Changes from v1.13 to v1.14

1/ Bug fix - cgRect had some problems since v1.10; now fixed.

Changes from v1.12 to v1.13

1/ New function - cgScrDmp() added to allow screen dumps of the display screen.

Changes from v1.11 to v1.12

1/ Bug fix - 16 bit graphics had some colour problems which have been sorted out now.

Changes from v1.10 to v1.11

1/ Cogent Graphics now supports the playing of movies.

Changes from v1.09 to v1.10

1/ The cgRect() function now follows the alignment regime set up by cgAlign().

2/ The cgDrawSprite() function can now take a width and a height so that sprites can be scaled when drawn.

3/ The cgMouse() function now detects when the control and shift buttons are down with a mouse button.

4/ A new function has been added, cgKeyMap(), which allows the user to quickly and easily scan the keyboard for keys pressed. At the moment this function has not been fully integrated with the rest of cogent and there may be some problems concerning the use of this function while cogent mouse logging is going on. Use this function with caution.

5/ The samples have been modified so that pressing the escape key terminates the script. This is necessary because Ctrl+C doesn't always work when running a tight animation loop on a dual-monitor system.

Changes from v1.08 to v1.09

1/ The cgOpen() command has been updated so that multiple displays can be used. This manual now contains a section on "Multiple Displays".

Changes from v1.07 to v1.08

1/ The functions cgDraw(), cgRect() and cgEllipse() have been modified so that they accept array arguments. This provides a huge increase in speed when drawing large numbers of items.

2/ A new function has been added, cgMouse(), which allows the user to quickly and easily obtain a mouse pointer position in the current co-ordinate system. At the moment this function has not been fully integrated with the rest of cogent and there may be some problems concerning the use of this function while cogent mouse logging is going on. Use this function with caution.

Changes from v1.06 to v1.07

1/ New function cgGetData() allows access to virtually all internal data values from the Matlab workspace.

2/ New function cgPhotometer() communicates with the PhotoResearch PR-650 spectra-colorimeter. Colorimetric and spectral measurements can be obtained.

Changes from v1.05 to v1.06

The main change has been additional support for 8-bit graphics and the completion of various previously missing functions. The cgLoadArray function has changed substantiallyand you should consult this manual in further detail about this function.

1/ The cgColTab matlab call now accepts a single (n x 3) RGB argument as well as separate R, G and B arguments.

2/ The cgFlip command has been extended so that you can simply wait until the start of the next display frame without doing a pageflip.

3/ The cgLoadArray command now takes a single PixVal argument rather than the previous separate R, G and B arguments. It is also possible to define a palette based array as well as a direct mode rgb array and arrays can now be loaded in palette display mode.

4/ The cgLoadBMP command now works in palette mode.

5/ This manual now has a new section of sample matlab scripts.

Changes from v1.04 to v1.05

The main change has been the addition of support for 8-bit graphics.

1/ The cgopen() command has been changed. In v1.04 the format was cgopen(mod,ref,mon). Now the format is cgopen(res,bpp,ref,mon). The graphics mode (mod) argument has been replaced by separate resolution (res) and bits per pixel (bpp) arguments.

2/ The cgpen() command has been replaced by two new commands; cgpencol() and cgpenwid().

3/ The cgmakesprite() and cgloadarray() commands have been changed. The transparent colour (TCol) argument has been dropped.

4/ There is a new command, cgtrncol(). This can be used to set the transparent colour of a sprite.

5/ The following commands can accept a colour as an RGB triplet when in normal graphics mode, or as a palette index when in 8-bit mode; cgflip(), cgmakesprite(), cgpencol().

6/ cgflip() now has an immediate mode of operation in palette mode where it does not wait for the VBL. This is to allow cgnewpal and cgflip to occur during the same frame. cgflip does not return a timestamp in immediate mode.

7/ The cgloadarray() and cgloadbmp() commands are only available in direct colour mode.

General description

The GScnd suite of graphics functions allows you to present graphic stimuli from within a Matlab program. It is provided so that you can design your own experiments for use in clinical testing of patients, psychophysics, scanning and eeg. You should be able to run your experiment on any PC running the Windows operating system.

You may present images, lines, rectangles, ellipses and polygons with accurate timing and fully integrated within the Matlab environment.

Installation

The GScnd suite can be installed on any PC running the Windows operating system. This includes Windows 95, Windows 98 and Windows 2000. In addition, DirectX, Matlab and Cogent Graphics must be installed on the PC.

DirectX Installation

The PC should also have DirectX installed. DirectX allows the program to work with the huge variety of graphics cards that are available and that you may have on your PC. DirectX is a Microsoft product and is distributed free of charge from the following web address:-

http://www.microsoft.com/downloads

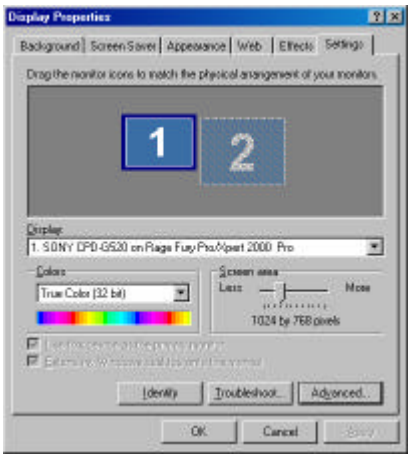
If you have Windows 2000 installed on your PC it should not be necessary to install DirectX. However, you will need it if you have Windows95 or Windows98. When you reach the Microsoft download URL mentioned above select the operating system you have on your PC (either Windows 95 or Windows98) and then select the "Keyword Search" selection button. Type "DirectX" into the Keywords box and start the search. A bewildering array of possibilities will appear for you to choose from such as those shown below:-

Date	Title	Version	Size/Time
7 Mar 2000	DirectX 7.0a Update DirectMusic Producer	7.0a	334 kb / 3min
3 Jun 1999	DirectX 5.0 DDK (Driver Developers Kit) Release 3	5	7,497 kb / 37min
20 Sep 1999	**DirectX 7.0**	**7.0**	**6,164 kb / 30min**
11 Jan 2000	DirectX 7.0 Update: 7.0 to 7.0a Patch	7.0a	234 kb / 2 min
17 Dec 1999	**DirectX 7.0a**	**7.0a**	**6,341 kb / 31 min**
15 Feb 2000	DirectX DDK (Driver Developers Kit) 7.0 for Windows 98	7.0	5,251 kb / 26 min
9 Jun 1999	DirectX Fix: DirectPlay 6.1a	6.1a	632 kb / 4 min
30 Aug 1999	DirectX Media Runtime 6.0	6.0	4,679 kb / 23 min
30 Aug 1999	DirectX Media Runtime 6.0 Patch for DirectAnimation	6.0	1,020 kb / 6 min
26 Oct 1999	DirectX SDK 3.0 Sample: Render Triangle in Direct3D Immediate Mode (D3debtri.exe)	1	54 kb / 1 min
22 Sep 1999	DirectX SDK 7.0	7.0	128,892 kb / 10h 12min
13 Oct 1999	DirectX SDK 7.0 API: documents, headers and library files	7.0	5,441 kb / 27min
22 Dec 1999	DirectX SDK 7.0 library for Borland Cbuilder	7.0	65 kb / 1min
20 Dec 1999	DirectX SDK 7.0a	7.0a	128,090 kb / 10h 8min

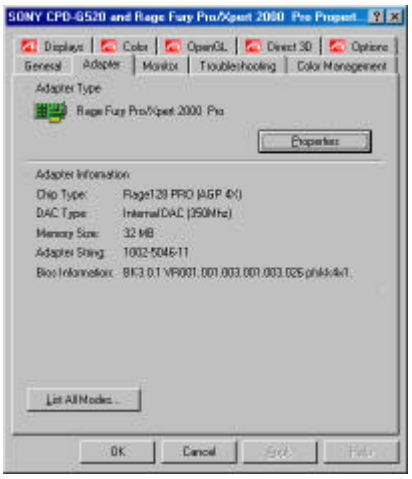
You should select the most recent of the items named simply DirectX n.nn (shown in bold lettering above). In the case shown above you would select **DirectX 7.0a**. You should then follow the installation procedure described on the website for that component.

DirectX and refresh rate

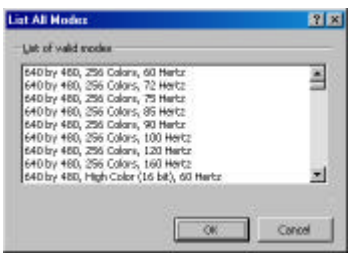
You may request a specific refresh rate in the **cgopen** command providing it is supported by your graphics card. If you want a refresh rate of 100Hz you may use **cgopen(1,0,100,1)**. You may select refresh rates in this way that are not normally available using the Direct X control panel or even under Windows. You should take care that your monitor does not blow up when you select a refresh rate that it is too high. You may want to arrange some sandbags for this purpose. To find out which refresh rates are supported by your graphics card, open the "Display" control panel and then click on the "Advanced" button:-



A control panel for your graphics card should appear:-



Click on the "Adapter" tab and then click the "List all modes" button. You should see a list showing which refresh rates are supported at different display resolutions:-

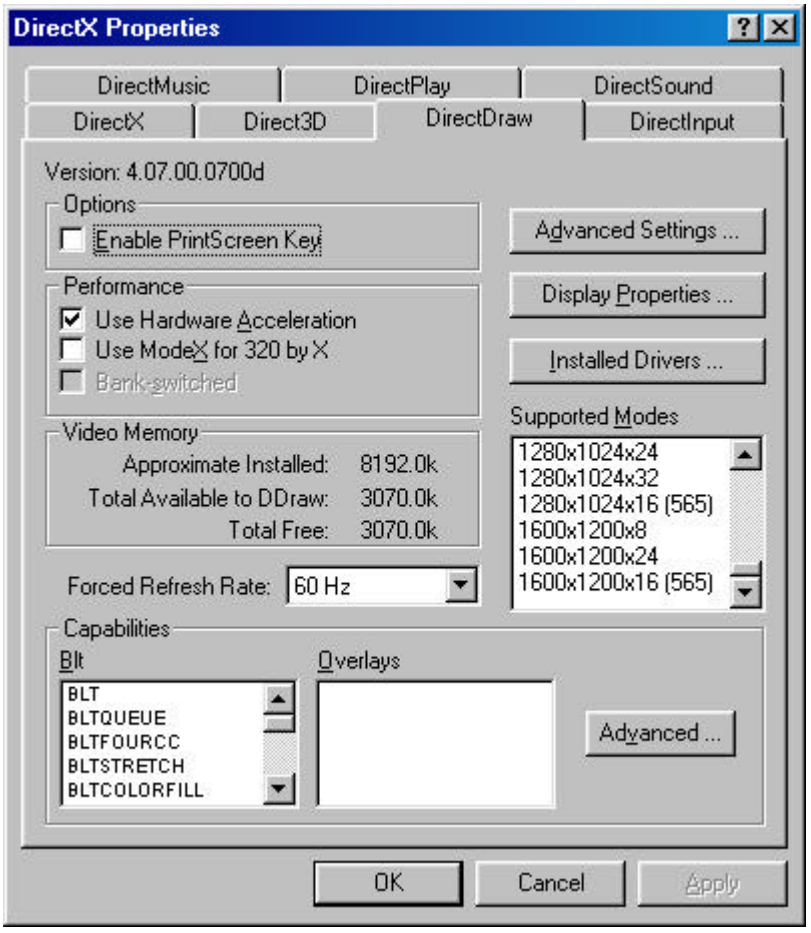


If you do not see your required refresh rate then there are still a couple of things you can do. First step is to go to the website of the manufacturer of your graphics card and install the most up-to-date driver for it. Then reboot your machine and look again. I have also found that with some graphics cards such as the Matrox G450 you can set up custom refresh rates which then become available. In this way you can push the G450 up to 160Hz even though initially it appears to only go up to 85Hz. You will perhaps have to tinker with these display settings to get the most out of the graphics card you have on your system.

However, I have found that this method does not work on some operating systems. If this is the case, you have another way to select your refresh rate; use the Direct X control panel (DirectX.cpl). This may be downloaded from the Microsoft website as part of the DirectX SDK (Software developer's kit) and it should be copied into your WINNT\System32 folder (Windows 2000) or Windows\System folder (Windows 98).

Once it has been installed in this way it will appear with all your other control panels (Start Menu-Settings-Control panel):-

When you run the control panel you should select the "DirectDraw" tab and then set the "Forced Refresh Rate" to the value you require.



You may find that your requested refresh rate is not supplied. For example, you may select 90Hz and only receive 85Hz. This means that DirectX cannot supply that refresh rate and has instead given the fastest refresh rate it can for your hardware.

Matlab Installation

The PC should have Matlab installed. Either the full version of Matlab can be used, or if you want to use your own personal computer you may prefer to purchase the student version of Matlab which is available at a discounted price. Please refer to the Matlab documentation for installation instructions.

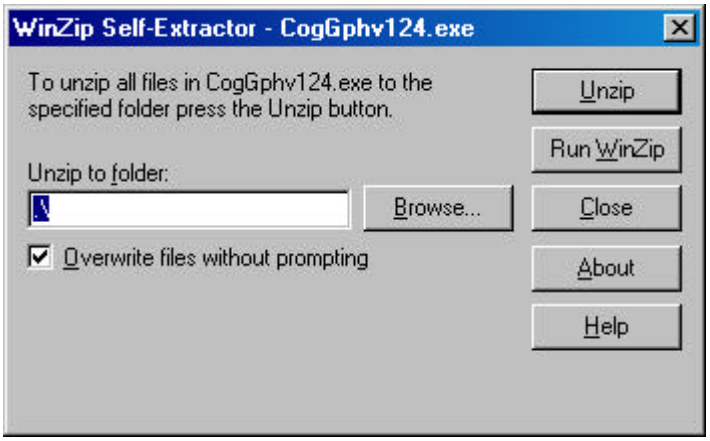
Cogent Graphics Installation

The Cogent Graphics website is to be found at the following URL:-

http://www.vislab.ucl.ac.uk/CogentGraphics.html

From that site you may download the Cogent Graphics package. This is downloaded in compressed form to your PC and must be first decompressed and then copied into the Matlab toolbox directory. Finally Matlab must be set up to search for commands in the new toolbox.

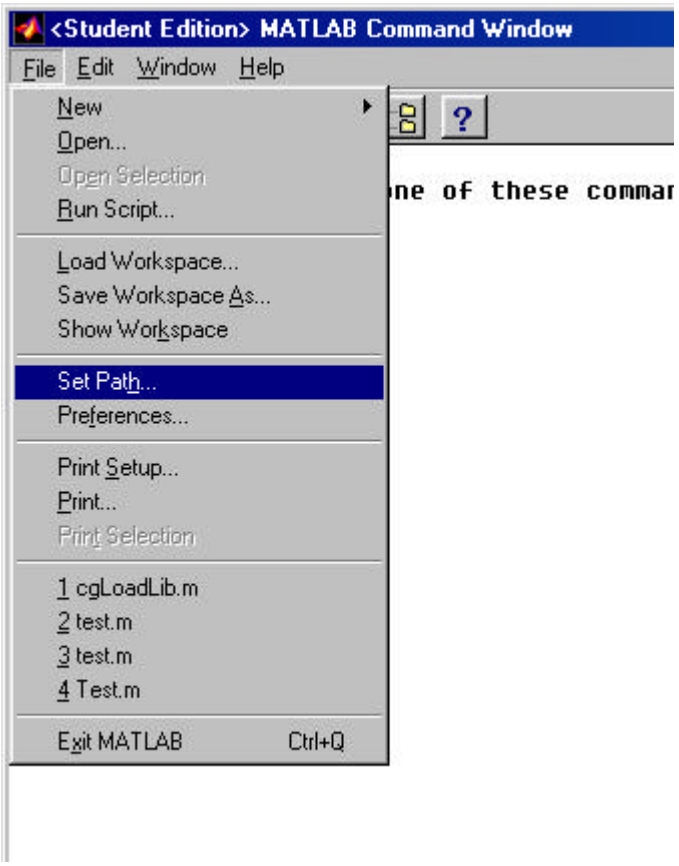
Cogent Graphics is downloaded from the website in compressed format in a file named CogGphvNNNx.exe where NNN represents the version number of the release. When you double click this file the following dialog box will appear:-



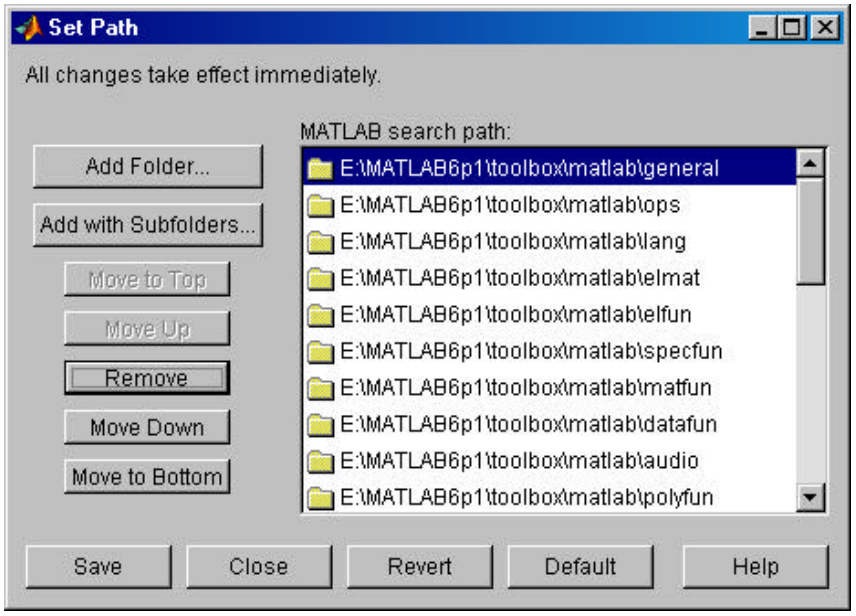
When you click on the Unzip button the toolbox will be uncompressed into the location in the "Unzip to folder:" box. A new folder will be created named "CogGph". This in turn contains three folders; "CogGphTB", which contains the matlab toolbox files for Cogent Graphics, "Doc", which contains documentation and "Samples", which contains example Matlab scripts. You should then copy the "CogGphTB" folder and its contents into your Matlab\Toolbox folder.

Finally you have to instruct Matlab to search this new folder for commands.

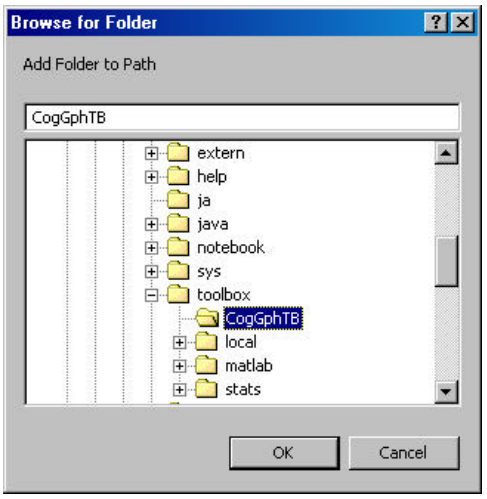
Run matlab and then select “Set Path” from the “File” menu:-



The MATLAB Path dialog will appear. The exact appearance will vary depending on your version of Matlab:-



Click on the "Add Folder…" button, select the MATLAB\toolbox\CogGphTB folder and click "OK":-



You should make sure it is at the top of the list (if necessary select it and then click on the "Move To Top" button) and finally click the "Save" button. This will complete the Cogent Graphics installation on your PC.

GPrim and CogStd

The GScnd suite of commands uses two underlying libraries named GPrim.dll and CogStd.dll. It should not be necessary for you to use these libraries.

Palette mode

Most of the time you will probably write Cogent Graphics scripts using "direct" colour mode, where colours are defined in red, green and blue colour components. However, for some applications you may elect to use "palette" or "8-bit" mode. In this mode you have a fixed number of "palette indices" (usually 256) and once you have drawn something in a particular "palette index" you can then set that palette index to any colour you want.

This mode allows you to achieve some particular animation effects very efficiently. You can for example set all colours to black to blank the whole screen. You can "cycle" colours to achieve movement. You can also achieve effects of changing illumination.

Another reason for using 8-bit mode is increased speed. Any drawing operation including clearing the screen will be up to four times faster in palette mode than in direct mode. This is because each pixel on the screen is represented by a single byte in palette mode whereas it may occupy up to four bytes in direct mode.

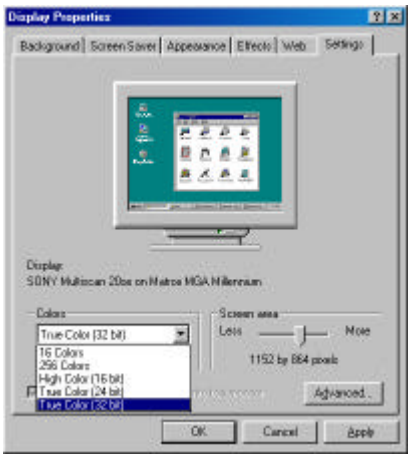
Similarly, palette mode conserves memory, sprites will be up to four times smaller in this mode than in direct colour mode.

However, using palette mode means that you have to manage the palette yourself, and there are only 256 palette entries to play with so this can be a difficult task. For this reason you should only use palette mode if you are forced to because of one of the reasons mentioned above.

The first tutorial section describes "direct" colour mode and then there is a separate tutorial on palette mode. It is important to set the display on your PC correctly for each tutorial because the

sub-window mode used in many of the examples cannot change the display mode for you. You should select "Control Panel" from the Start menu of Windows and then double click the "Display" control panel:-

Then choose the "Settings tab". In the colours menu you can select from 32 bit or 24 bit which result in direct colour modes or you can select "256 Colors" [sic] for palette mode. The 16 bit setting can be selected too and will result in a direct colour mode but the colour definition is inferior to 24 and 32 bit modes. When choosing between 24 and 32 bit colour modes you may find that 32 bit mode is (perhaps unexpectedly) faster than 24 bit mode. In general my advice is to select "32 bit" for direct colour mode and "256 Colors" for palette mode.



Display timing

There are three subjects to consider regarding display timing; display complexity, dropped frames and synchronization. Please read all three following sections if your experiment requires timing accuracy of greater than 0.1 seconds or if you are synchronising your display to external control or measuring equipment.

Display complexity

If you are running an animation at a certain frame rate, say 60Hz, you will have a fixed length of time available to draw each frame, in this case 1/60 second or 16.7 mS. If your stimulus is too complicated to draw in this time period you will find that your display will no longer run at 60Hz but will run at a slower frame rate. If this is unacceptable you should first try to optimize your script so that it runs more efficiently. Try to replace any **for** loops with matlab matrix operations, try to draw multiple objects, use palette animation if applicable and try drawing some complex elements of the stimulus beforehand if possible. You could even consider preparing all your frames beforehand in off-screen sprites.

Dropped frames

The following sample scripts can be used to test the timing accuracy of Cogent Graphics on your PC:-

BallsRGB/BallsPAL	ChessRGB/ChessPAL	Dartboard
ScrollRGB/ScrollPAL	StarsRGB/StarsPAL	TennisRGB/TennisPAL

These sample scipts are described briefly in a later section of this manual. Each one provides the following timing information:-

```
ProgName vN.NN P:NORMAL Tim:00:00:05 Frm:199 Av:75.12Hz Drp:0
```

The components are as follows:-

`ProgName vN.NN`	Program name and version number
`P:NORMAL`	Current priority class (explained below)
`Tim:00:00:05`	Time elapsed in hours minutes and seconds since script was started
`Frm:199`	Number of frames since script was started
`Av:75.12Hz`	Average frame rate in Hertz
`Drp:0`	The number of frames dropped since script was started

The sample scripts time each frame and calculate the average frame rate in hertz. This gives an average frame time. Each frame time is compared to the average and if it differs by more than 2 mS (0.002 seconds) it is counted as a "Dropped Frame".

There are two checks that you should make on these timing statistics. The first is to test the elapsed time counter against a stopwatch. This will check that the PC's inbuilt millisecond timer is accurate. It should be accurate to at least one second per hour. The second is to monitor the frequency of dropped frames. You should aim for a rate of one dropped frame per one million frames. You must run the sample for several hours to check the statistics to this level of accuracy.

When a frame is dropped your cogent animation will appear to hesitate momentarily and the dropped frames counter will increase by one. You will especially notice this if you do not use the

full cogent distribution commands "**start_cogent**" and "**stop_cogent**" before and after you run the sample script. These dropped frames are obviously undesirable and they could have serious consequences for your experiment if they occur at a critical point in your stimulus. The dropped frames cannot be eliminated altogether but by following the methods described in this section it should be possible to reduce them to a frequency of about one dropped frame per million frames. If your experiment is sensitive to dropped frames you should always monitor the timestamps of your **cgflip** and **cgnewpal** commands and you should be prepared to ignore any trials where a frame has been dropped.

Cogent Graphics has been designed to work with Windows 2000, a multi-tasking operating system. In such an operating system there are many tasks competing for your PCs resources, even though you may not be aware of them. There may be anti-virus software running on your PC and there will be operating system tasks running to monitor the keyboard, mouse and network and to control disk activity when you save data to files. All of these other tasks can potentially interfere with your Cogent Graphics script. Practically this can result in a "Dropped Frame". Steps have been taken in the larger cogent distribution to eliminate this effect; when you issue the "**start_cogent**" command you should no longer see these dropped frames. By issuing the "**start_cogent**" command your matlab session is given a higher priority than all these other tasks. When you use the "**stop_cogent**" command the priority is returned to normal. If you are using the wider cogent distribution this should be all you have to do; if you find that you are experiencing significant numbers of dropped frames while using the full cogent distribution you should contact a member of the computer support staff to discuss the problem.

If you are not using the full cogent distribution you can still raise the priority of your matlab session to a higher class using the following command:-

cogstd('spriority','high')

You can return the priority to normal using the following command:-

cogstd('spriority','normal')

Typically you would use the command in the following way to run the sample script "ballsrgb" at a high priority:-

```
cogstd('spriority','high')
ballsrgb
cogstd('spriority','normal')
```

If you still find there are significant numbers of frames being dropped you should contact a member of the computer support staff. They should then examine your PC and go through the steps described later under "System Tuning" in order to rectify the problem.

Synchronisation

When using a CRT (Cathode Ray Tube or standard monitor) display, the **cgflip** and **cgnewpal** commands should be synchronized to the top left corner of the display. You can check this by running a script to flash the screen black and white and also generate a simultaneous external signal on the parallel port. A dual beam oscilloscope and photodiode circuit can then be used to check the synchronization directly. You should ask a member of the computer support staff to assist you if you want to do this.

You should also be aware that using different graphics cards and different displays may disrupt synchronization. In particular, the use of LCD screens and projectors may introduce a constant timelag of one frame between issuing a **cgflip** or **cgnewpal** command and the results appearing on the display. If this would affect your experiment you should consult the computer support staff who will advise you how to proceed further.

System Tuning

On a typical PC (1.4GHz Pentium IV, 256Mb RAM) running Windows 2000, Matlab v6.1 and Cogent Graphics v1.24 running in the "high" priority class the sample scripts should exhibit a dropped frame rate of about one in a million. If you find that you are getting significantly more dropped frames than this you should take steps to improve the situation. The system configuration described above is the currently recommended minimum configuration; Cogent Graphics will work on a machine with lower specifications, but the performance may be so poor on a slower machine as to render the program unusable.

Problems with dropped frames are due to two possible reasons; hardware conflicts and software conflicts.

Hardware conflicts may occur when a card in your PC is causing an interrupt which interferes with Cogent Graphics. The most likely suspect here is your network card. Try unplugging your PC from the network. This may well cure the dropped frames. If this is the case then you should remember to unplug your PC from the network whenever you run a time-critical experiment. You may also be able to cure the problem by changing the process priority as described below for dealing with software conflicts, or by using a different driver for your network card.

Software conflicts may occur between Cogent Graphics and other software that has been installed on the system. Usually these conflicts can be resolved by increasing the priority class to "high", either by using the **CogProcess** command as employed in **start_cogent** and **stop_cogent** or by using the **CogStd('spriority','high')** command. If this fails, you could try increasing the priority class to "realtime" using the **CogStd('spriority','realtime')** command. This may solve the problem but it may have undesirable consequences regarding system performance. It is an option which should be tried as a simple quick fix but checks should also be made to make sure that other system processes such as keyboard, mouse and serial communications and disk operations are not affected. In particular a possible side-effect may be that timestamps for communications events become tied to the vertical blanking interrupt of the screen.

If increasing the priority class to realtime does not work, the next step is to try to ascertain which process is causing the conflict. You should do this with the aid of the task manager, and you should be prepared to uninstall programs if necessary. Possible candidates for conflict are anti-virus software, the windows update utility, X-terminal servers, or any program which accesses the network. You could also try disabling items from the taskbar such as the system clock and CD-writer software icons. Potentially any program may be causing the conflict and you should try to

eliminate possible programs by reference to the task manager, one by one, until the conflict is resolved.

My final suggestion is to create a clean operating system devoid of unnecessary programs, either by reinstalling, or by creating a bootable clean operating system on a separate partition. If you use a separate partition then you can also disable any hardware that is causing a problem by using the device manager in the System control panel.

Direct colour tutorial

Getting started

This first section checks that the installation has gone correctly and that all the components of the program are compatible. From the Matlab console, type the following:-

>> **cgloadlib**

If you type the command in correctly and get an error message...

>> **cgloadlib**
??? Undefined function or variable 'cgloadlib'.

…it means that there is something wrong with your installation. Use the "File" menu "Set Path…" item to invoke the "MATLAB Path" dialog box. Double-check that the Cogent Graphics toolbox folder is one of the listed paths and that it does contain a file named cgloadlib.dll. You may want to repeat the Cogent Graphics toolbox installation at this point.

The **cgloadlib** command prepares all the program components for execution. You must start each Cogent Graphics session with this command. The command also checks that all the necessary components are present. If something is missing you will get an error message such as:-

>> **cgloadlib**
??? Undefined function 'cgellipse'.

Error in ==> E:\matlabR12\toolbox\CogGphTB\cgLoadLib.dll

The message above means that a file named cgellipse.dll is missing from the Cogent Graphics toolbox. If this happens you must exit matlab and re-install Cogent Graphics from the web distribution.

The **cgloadlib** command also has another form, **cgloadlib('U')**, which unloads all the graphics library files. You should not need to use this form of the command under normal circumstances.

Next type the **cgvers** command:-

```
>> cgvers
CogStd  v1.24 Compiled:Oct 23 2002
GPrim v1.24 Compiled:Oct 23 2002 (GLib not set by ginit)
GScnd:cgData  v1.24 Compiled:Oct 23 2002
GScnd:cgVe rs  v1.24 Compiled:Oct 23 2002
GScnd:cgAlign  v1.24 Compiled:Oct 23 2002
GScnd:cgColTab  v1.24 Compiled:Oct 23 2002
GScnd:cgDraw  v1.24 Compiled:Oct 23 2002
GScnd:cgDrawSprite  v1.24 Compiled:Oct 23 2002
GScnd:cgEllipse  v1.24 Compiled:Oct 23 2002
GScnd:cgFlip  v1.24 Compiled:Oct 23 2002
GScnd:cgFont  v1.24 Compiled:Oct 23 2002
GScnd:cgFreeSprite  v1.24 Compiled:Oct 23 2002
GScnd:cgLoadArray  v1.24 Compiled:Oct 23 2002
GScnd:cgLoadBMP  v1.24 Compiled:Oct 23 2002
GScnd:cgMakeSprite  v1.24 Compiled:Oct 23 2002
GScnd:cgNewPal  v1.24 Compiled:Oct 23 2002
GScnd:cgOpen  v1.24 Compiled:Oct 23 2002
GScnd:cgPenCol  v1.24 Compiled:Oct 23 2002
GScnd:cgPenWid  v1.24 Compiled:Oct 23 2002
GScnd:cgPolygon  v1.24 Compiled:Oct 23 2002
GScnd:cgRect  v1.24 Compiled:Oct 23 2002
GScnd:cgScale  v1.24 Compiled:Oct 23 2002
GScnd:cgSetSprite  v1.24 Compiled:Oct 23 2002
GScnd:cgShut  v1.24 Compiled:Oct 23 2002
GScnd:cgText  v1.24 Compiled:Oct 23 2002
GScnd:cgTrnCol  v1.24 Compiled:Oct 23 2002
GScnd:cgPhotometer  v1.24 Compiled:Oct 23 2002
GScnd:cgGetData  v1.24 Compiled:Oct 23 2002
GScnd:cgMouse  v1.24 Compiled:Oct 23 2002
GScnd:cgKeyMap  v1.24 Compiled:Oct 23 2002
GScnd:cgOpenMovie  v1.24 Compiled:Oct 23 2002
GScnd:cgShutMovie  v1.24 Compiled:Oct 23 2002
GScnd:cgPlayMovie  v1.24 Compiled:Oct 23 2002
GScnd:cgScrDmp  v1.24 Compiled:Oct 23 2002
GScnd:cgArc  v1.24 Compiled:Oct 23 2002
GScnd:cgBlitSprite  v1.24 Compiled:Oct 23 2002
GScnd:cgLoadLib  v1.24 Compiled:Oct 23 2002
```

The **cgvers** command checks that all the components are present and also that the version numbers of all components match. If one of the components is not present you will get the following message:-

```
>> cgvers
DLL load failed for mex file E:\matlabR12\toolbox\CogGphTB\cgVers.dll
The specified module could not be found.
??? Invalid MEX-file
```

If this happens you must again re-install Cogent Graphics from the web distribution.

If one of the components has the wrong version number you will get a warning message informing you of the fact. Your experiment may nevertheless run but it would be safer to re-install Cogent Graphics from the web distribution.

```
>> cgvers
CogStd v1.03 Compiled:Jun 13 2000
WARNING – INCONSISTENT VERSION NUMBER
GPrim v1.03 Compiled:Jun 13 2000 (Glib not set by ginit)
WARNING – INCONSISTENT VERSION NUMBER
GScnd:cgVers   v1.24 Compiled:Oct 23 2002
        .       .       .
GScnd:cgBlitSprite   v1.24 Compiled:Oct 23 2002
```

If you want to see a short description of any command type the command with a question mark string as its argument to see a usage guide:-

```
>> cgvers('?')
ERR GScnd:cgVers Usage:-
ERR GScnd:cgVers    cgVers<('U')> ('U' prints usage description too)
```

In this case you can see that cgvers can optionally be called in the form **cgvers('U')**. This command will print the usage guide for every possible command in the GScnd suite of functions. It is rather lengthy so won't repeat it here. If you make a mistake typing in any command try looking at the usage guide (use the **('?')** format) to see where you have gone wrong.

Opening and closing graphics

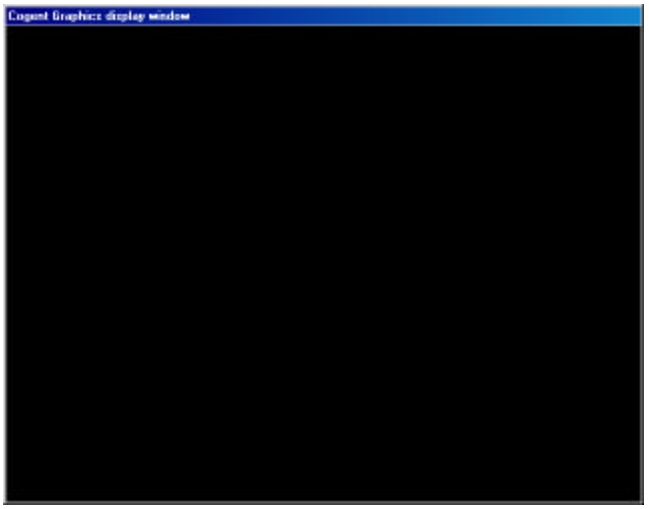
This tutorial is for direct colour graphics mode. You should set your Windows desktop to 32 bit mode using the display control panel (see the "Palette mode" section in the introduction).

First of all prepare for Cogent by typing the **cgloadlib** command:-

```
>> cgloadlib
```

Now open a graphics screen using the **cgopen** command:-

```
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 75.24Hz
```



The Cogent Graphics display window should open on your desktop showing a black screen and a two line message should appear on the matlab console. The first line gives the version number of the underlying "GPrim" library that is being used and the second line tells you some information about the display that has been opened; 640x480x32 denotes a display that is 640 pixels wide by 480 pixels high with pixels that are 32 bits big and 75.24Hz tells you the refresh rate of the screen; in this case 75.24 hertz. You can close the graphics screen with the **cgshut** command:-

```
>> cgshut
```

The window that you opened above was a sub-window on the desktop. This is useful for developing and debugging your experiment but when the time comes to run your experiment in earnest you will want the display to take over the whole screen. This is because the program runs very slowly in sub-window mode. You can do this using a slightly different command:-

```
>> cgopen(1,0,0,1)
```

This time the whole display goes black. At this point you can minimise the black full-screen window by holding down the Alt key and pressing the tab key. The minimised window is now represented by a rectangle on the system toolbar. You can restore it by clicking on the rectangle.

Furthermore, if you have a PC with two monitors you can open the window on the second monitor using the following command:-

>> **cgopen(1,0,0,2)**

The command above will only work if you have a second monitor, otherwise you will get an error message. If you want to know what your available device options are, try the command below:-

```
>> cgopen(1,0,0,-1)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Available devices:-
    1:display (Primary Display Driver)
```

If you want to list the available resolutions, try the command below:-

```
>> cgopen(1,0,0,-2)
Available display resolutions:-
  1: 640 x 480
  2: 800 x 600
  3: 1024 x 768
  4: 1152 x 864
  5: 1280 x 1024
  6: 1600 x 1200
```

You may also explicitly request a specific refresh rate such as 120 Hz by using the command **cgopen(1,0,120,1)**. You can only request refresh rates that are supported by your graphics card. See the section "Direct X and Refresh rate" to discover how to find out which refresh rates are supported.

Colours

Colours are represented as a mixture of red, green and blue intensities. The intensities can take values from zero (off) to one (maximum intensity). Thus:-

0,0,0	represents black
1,0,0	represents maximum red
0,1,0	represents maximum green
0,0,1	represents maximum blue

Other colours can be expressed as mixtures of red, green and blue:-

1,1,0	represents maximum yellow
1,0,1	represents maximum magenta
0,1,1	represents maximum cyan
1,1,1	represents maximum white

Varying intensities can be expressed using decimal fractions:-

0.2,0.2,0.2	represents dark grey
0.5,0.5,0.5	represents mid-grey
0.8,0.8,0.8	represents light grey

Page flipping

When you execute a drawing command there is no change visible on the screen. This is because all drawing commands are executed on an offscreen area of graphics memory. This offscreen area may be visualised as a rectangle exactly the same size as the screen floating somewhere invisibly in space. The act of copying this invisible offscreen area to the visible monitor screen is called page-flipping. When you execute the **cgflip** command the offscreen area is copied to the display. You can also clear the offscreen area to a flat colour at the same time:-

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgflip(0,0,1)
```

Here the **cgflip(0,0,1)** command clears the offscreen area to maximum blue. However, nothing appears on the monitor screen until you call **cgflip** again:-

```
>> S=cgflip

S =

   19.0860
```

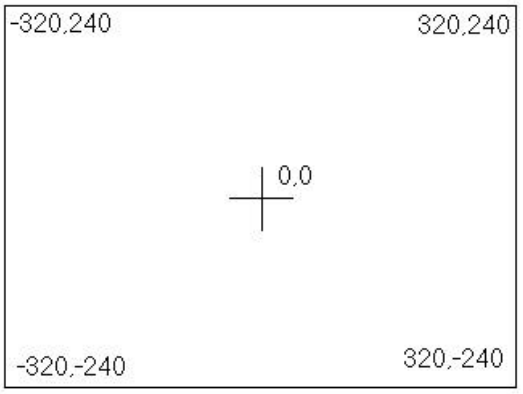
The screen now turns blue. This time we have used the **S=cgflip** form of the command. This form of **cgflip** returns a timestamp indicating precisely when the offscreen area appeared on the monitor. The timestamp is a value in seconds since the **cgopen** command was executed. At the moment the second timer increments in steps of 0.001, i.e. it is accurate to 1 millisecond. The value returned above represents a time of 19.0860 seconds since the **cgopen** command as issued. Prior to v1.17 this function returned the timestamp in units of microseconds but there was a bug in these values which meant that the timestamp became negative after 35 minutes. This bug has been fixed from v1.17 onwards where the return value is in units of seconds.

Another form of this command allows you to wait for the start of the next display frame. This facility is useful when presenting stimuli for short periods of time measured in individual frames. To wait until the next display frame (without flipping) use the command **cgflip('v')**.

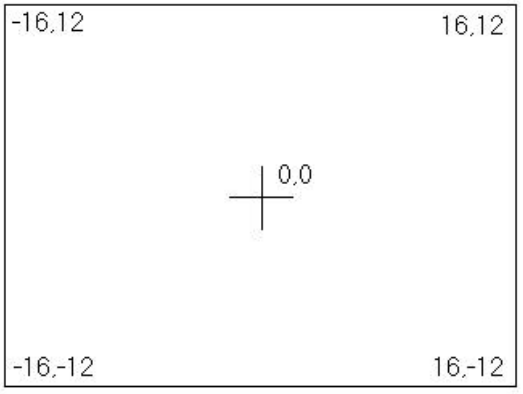
Co-ordinate system

Cartesian (x,y) co-ordinates are used with the origin (the point 0,0) at the centre of the screen. Horizontal (x) co-ordinates increase to the right and vertical (y) co-ordinates increase upwards. Two types of units can be used; pixels and visual angle.

The default is to use pixel co-ordinates. Consider a screen which is 640 pixels wide and 480 pixels high. The co-ordinates of the centre and corners of the screen will be:-



However, you may want to use co-ordinates with visual angle as the units. One advantage of doing this is that it is easier to ensure that the subject of you experiment sees the same sized stimuli on different occasions, perhaps using different displays. If you know the screen width in degrees for the set up you want to use you can set that directly or you can set the screen width and observer distance in millimetres. Suppose that the screen is 32 degrees wide and 24 degrees high. The co-ordinates of the centre and corners of the screen will now be:-



You use the **cgscale** command to set the co-ordinate system:-

cgscale On its own selects pixel co-ordinates.

cgscale(32) Sets visual angle co-ordinates with the screen width set to 32 degrees.

cgscale(400,600) Sets visual angle co-ordinates with screen width 400mm and observer distance 600mm.

Drawing pen

The **cgpencol** and **cgpenwid** commands set the colour and line width respectively for subsequent drawing operations:-

cgpenwid(5) Sets the line width to 5 units. The units are either in pixels or in degrees depending on which co-ordinate system is being used.

cgpencol(1,0,0) Sets the current drawing colour to maximum red.

Points and lines

You can use the **cgdraw** command to draw points and lines. The command can take two forms;

cgdraw(x,y) Draws a point at co-ordinate x,y. The point is drawn in the current drawing colour.

cgdraw(x1,y1,x2,y2) Draws a line between co-ordinates x1,y1 and x2,y2. The line is drawn in the current drawing colour. The width of the line may be set with the **cgpen** command.

Try the following exercise…

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgpencol(1,1,1)
```

So far we have opened the graphics and then set the current drawing colour to maximum white. Now let us draw some points…

```
>> cgdraw(100,100)
>> cgdraw(-100,100)
>> cgdraw(-100,-100)
>> cgdraw(100,-100)
```

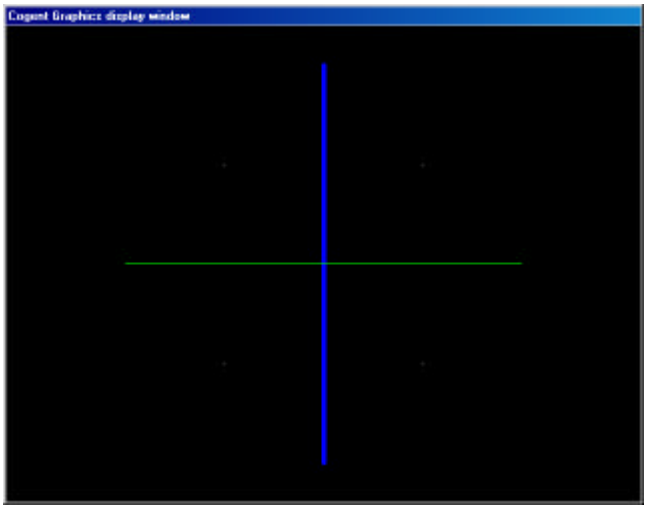
Of course, nothing appears on screen yet because it is all being drawn on the offscreen area. Anyway, for the moment trust that the points have been drawn. Now let us add some lines…

```
>> cgpencol(0,0,1)
>> cgpenwid(5)
>> cgdraw(0,200,0,-200)
>> cgpencol(0,1,0)
>> cgpenwid(1)
>> cgdraw(-200,0,200,0)
```

First we set the drawing colour to maximum blue and the line width to 5 pixels. Next we draw a vertical line from 0,200 to 0,-200. Then we set the drawing colour to maximum green and the line width to 1 pixel and draw a horizontal line from –200,0 to 200,0. To see what it all looks like we must execute the page-flipping command…

>> **cgflip**

You should see something similar to the window shown below…



You should also read "Drawing multiple items' later on in this section so you know how to draw many points or lines efficiently with a single command.

Rectangles, scaling and alignment

You can use the **cgrect** command to draw filled rectangles. The rectangle is filled with the current drawing colour as set by **cgpen**. The command has two forms…

cgrect The command on its own fills the whole destination with the current drawing colour.

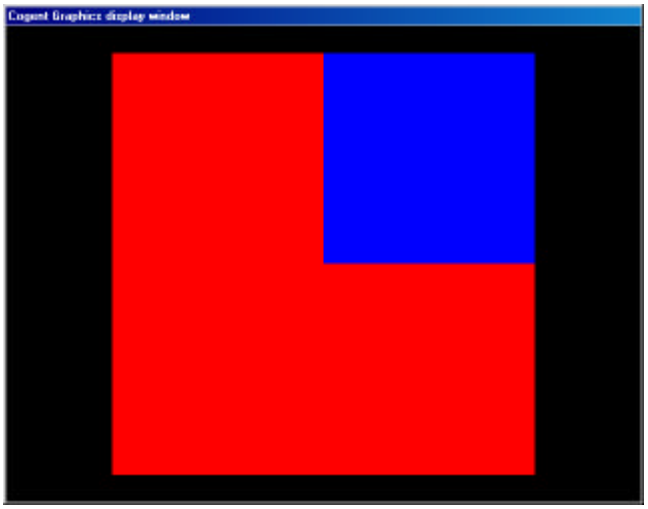
cgrect(x,y,w,h) This command fills a rectangle with the current drawing colour. The centre of the rectangle is at co-ordinate x,y and the rectangle width and height are defined by w and h.

In this example we will use **cgscale** to use a visual angle co-ordinate system. We open the graphics and then set the screen width to 30 degrees. We set the drawing colour to maximum red and then draw a rectangle, centred on the middle of the screen with width 20 degrees and height 20 degrees – a square. Nothing appears on the display yet as it is all being drawn on the offscreen area.

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgscale(30)
>> cgpencol(1,0,0)
>> cgrect(0,0,20,20)
```

We next set the drawing colour to blue and draw another square, side 10 degrees, centred on a point 5 degrees up and 5 degrees right of the middle of the screen. These co-ordinates have been chosen so that the top right corners of both squares coincide. We then use th flip command to see what we have drawn:-

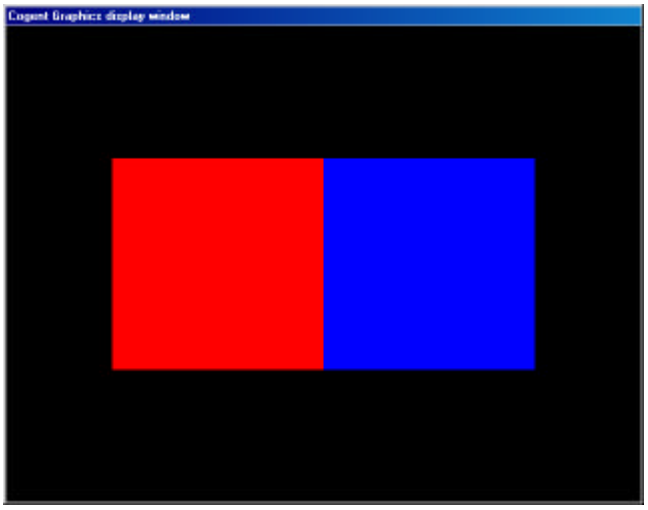
```
>> cgpencol(0,0,1)
>> cgrect(5,5,10,10)
>> cgflip
```



The rectangles just drawn were centred on the x and y co-ordinates. This is because the default alignment mode when we call **cgopen** is to centre horizontally and vertically. However, we can alter the alignment of the rectangles using the **cgalign** command:-

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgscale(30)
>> cgalign('r','c')
>> cgpencol(1,0,0)
>> cgrect(0,0,10,10)
>> cgalign('l','c')
>> cgpencol(0,0,1)
>> cgrect(0,0,10,10)
>> cgflip
```

Here we set the alignment mode to 'right centre' with the **cgalign('r','c')** command and draw a red square with its right edge on 0,0 (the centre of the screen). Then we set the alignment mode to 'left centre' with the **cgalign('l','c')** command and draw a blue square aligned so that its left edge on 0,0. The vertical alignment is still centred on the centre of the screen:-



You should also read "Drawing multiple items' later on in this section so you know how to draw many rectangles efficiently with a single command.

Ellipses and circles

You can use the **cgellipse** command to draw ellipses and circles.

cgellipse(cx,cy,w,h) This command draws a hollow ellipse with the current drawing colour and the current line width as set by **cgpen**. The centre of the ellipse is at co-ordinate cx,cy and the ellipse width and height are defined by w and h.

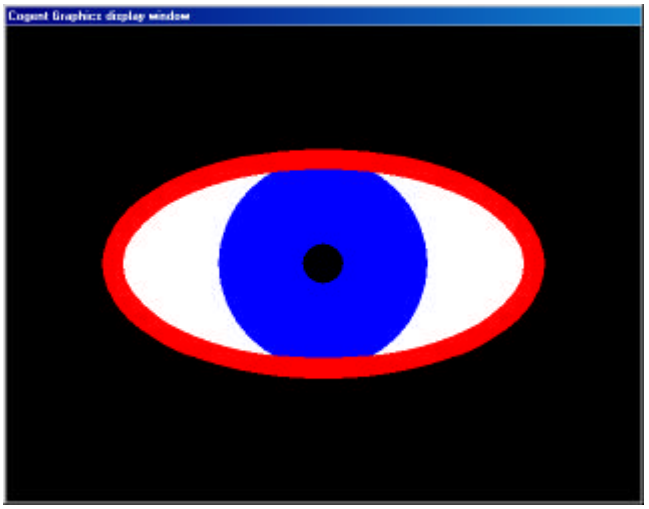
cgellipse(cx,cy,w,h,'f') This command draws a filled ellipse.

In this example we set the screen width to 30 degrees, set the drawing colour to white and draw a filled ellipse, centred on the screen with width 20 degrees and height 10 degrees.

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgscale(30)
>> cgpencol(1,1,1)
>> cgellipse(0,0,20,10,'f')
```

Then set the pen to blue and draw a filled circle of diameter 10 degrees. Next, set the pen to black and draw a filled circle of diameter 2 degrees. Then we set the pen to red and the line width to 1 degree and draw a hollow ellipse with the same dimensions as the first ellipse. Finally we display what we have drawn…

```
>> cgpencol(0,0,1)
>> cgellipse(0,0,10,10,'f')
>> cgpencol(0,0,0)
>> cgellipse(0,0,2,2,'f')
>> cgpencol(1,0,0)
>> cgpenwid(1)
>> cgellipse(0,0,20,10)
>> cgflip
```



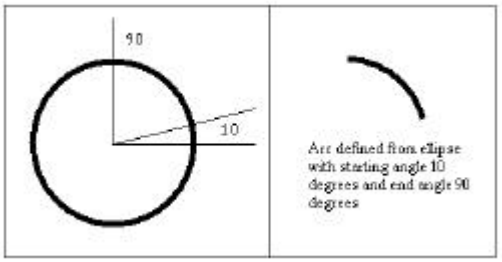
You should also read "Drawing multiple items' later on in this section so you know how to draw many ellipses efficiently with a single command.

Arcs and sectors

You can use the **cgarc** command to draw hollow arcs and filled sectors:-

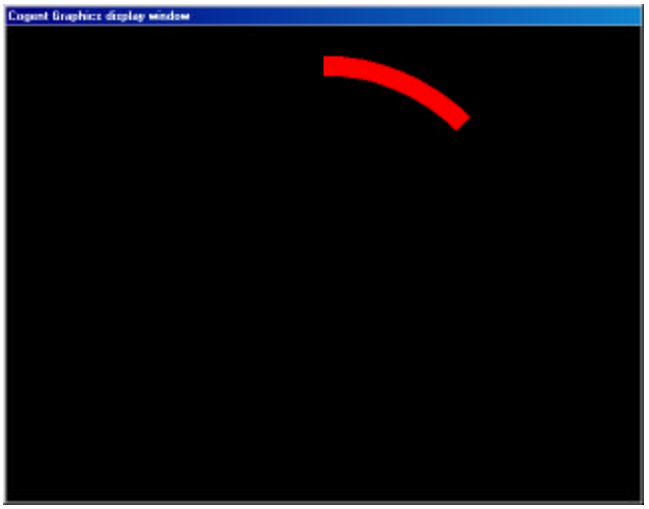
cgarc(cx,cy,w,h,a1,a2,Typ) This command draws an arc using the current drawing colour and the current line width as set by **cgpen**. The arc is drawn around the circumference of an ellipse with centre at co-ordinate cx,cy and the ellipse width and height are defined by w and h. The starting and ending angles of the arc are defined by a1 and a2 (see below). The **Typ** argument can take the value 'A' (the default) to draw hollow arcs or 'S' to draw filled sectors.

The a1 and a2 arguments define the start and end angles of the arc. These are specified in degrees with zero degrees being a horizontal direction to the right. Angles increase anti-clockwise so that 90 is a vertical upward direction, 180 is horizontally to the left and 270 is vertically down. The figure below shows an arc defined by a1 = 10, a2 = 90:-



Now try the following example:-

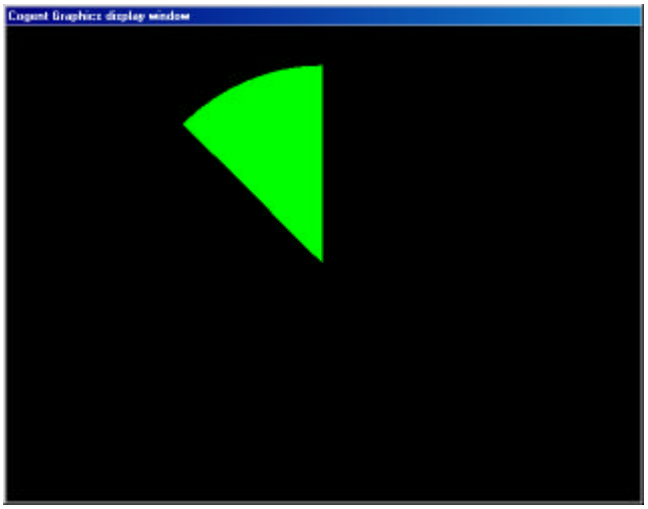
```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgpencol(1,0,0)
>> cgpenwid(20)
>> cgarc(0,0,400,400,45,90)
>> cgflip(0,0,0)
```



We have used **cgpencol** to set the drawing colour to red and **cgpenwid** to set the line width to 20. Then we used **cgarc** to draw an arc of an ellipse. The ellipse was centred on (0,0) with width 400 and height 400 (i.e. a circle of radius 200). We drew an arc from starting angle 45 degrees to ending angle 90 degrees.

In a similar way we can draw a filled sector:-

```
>> cgpencol(0,1,0)
>> cgpenwid(20)
>> cgarc(0,0,400,400,90,135,'S')
>> cgflip(0,0,0)
```



You should also read "Drawing multiple items' later on in this section so you know how to draw many arcs or sectors efficiently with a single command.

Polygons

You can use the **cgpolygon** command to draw filled polygons. The current drawing colour is used as set by **cgpen**. The command has two forms…

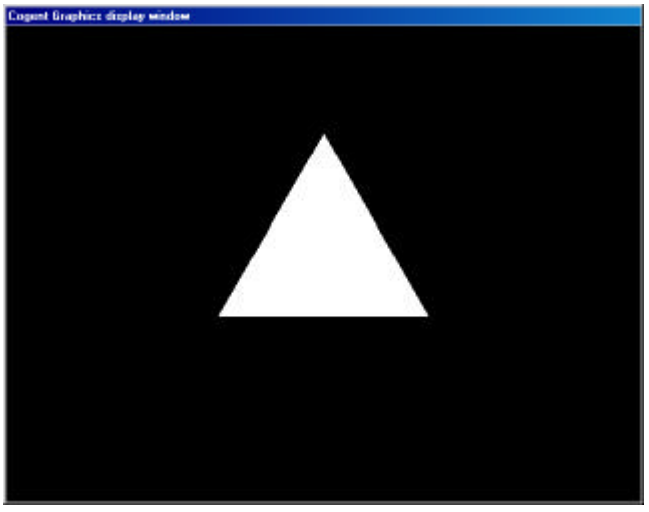
cgpolygon(x,y) This command draws a filled polygon in the current drawing colour as set by **cgpen**. The vertices (corners) of the polygon are held in arrays x and y.

cgpolygon(x,y,Xoffset,Yoffset) This command draws a filled polygon. The meanings of x and y are the same as in the previous form. The Xoffset and Yoffset terms allow you to move the polygon by an offset without having to redefine all the x and y values.

In this exercise we define a triangle in the arrays x and y…

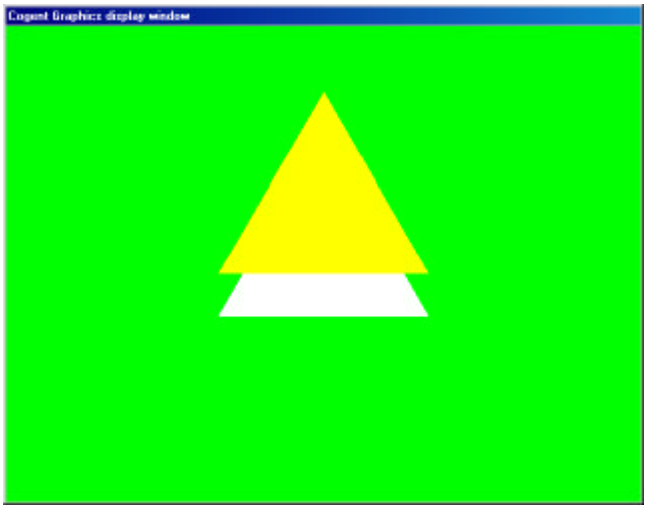
```
>> x = [    0    5     -5];
>> y = [6.15  –2.5 –2.5];
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgscale(30)
>> cgpencol(1,1,1)
>> cgpolygon(x,y)
>> cgflip(0,1,0)
```

After opening the graphics and scaling the screen at 30 degrees wide we set the drawing pen to maximum white, draw the triangle on the offscreen area and then display it with the **cgflip** command, clearing the offscreen area to maximum green.



>> **cgpolygon(x,y)**
>> **cgpencol(1,1,0)**
>> **cgpolygon(x,y,0,2)**
>> **cgflip**

We then draw the white polygon again, set the drawing colour to maximum yellow and then draw the polygon again, offset by 0,2 i.e. two degrees higher than the white polygon. The **cgflip** command simply displays the new scene.



Fonts

A font defines the typeface in which your text will appear. A particular font defines the size of the text as well as the exact shape of the letters.

To view fonts on your computer

2. Click **Start**, point to **Settings**, click **Control Panel**, and then double-click **Fonts**.

2. To look at a sample of a font, click the icon for the font.

You can select a particular font using the **cgfont** command…

cgfont('Fontname',FontHeight) This command selects the font called "Fontname" at a size specified by FontHeight for subsequent text drawing operations.

Text

You can draw text using the **cgtext** command…

cgtext('Text',x,y) This command draws the given text at the point x,y using the currently selected font as set by **cgfont** in the current drawing colour as set by **cgpen**. The current alignment mode is used to position the text with respect to x,y.

The following example illustrates the use of the **cgtext** command…

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgscale(30)
>> cgfont('Arial',4)
>> cgpencol(1,1,0)
```

We open the graphics, define the screen width to be 30 degrees and then load 'Arial' font with the letters 4 degrees high. We then set the drawing colour to be yellow.

```
>> cgtext('All that glisters',0,2)
>> cgtext('is not gold',0,-2)
>> cgflip
```

Next we draw two lines of text, the first is 2 degrees above the centre of the screen, the second is 2 degrees below. Then we make the offscreen area visible with **cgflip**.

There are two points to note in this example, both concerning the alignment of the text:-

Firstly, we selected a font that was 4 degrees high. When we drew the two lines of text we drew the first 2 degrees above the midline of the screen and the second 2 degrees below the midline. The vertical separation of 4 degrees (the same as the font height) gives a nice vertical spacing between the two lines of text.

Secondly, the text was drawn centred vertically and horizontally on the co-ordinates in the **cgtext** command. This is the default text drawing mode but this can be changed by setting a different alignment mode with the **cgalign** command.

Text alignment

You can set the alignment mode with the **cgalign** command:-

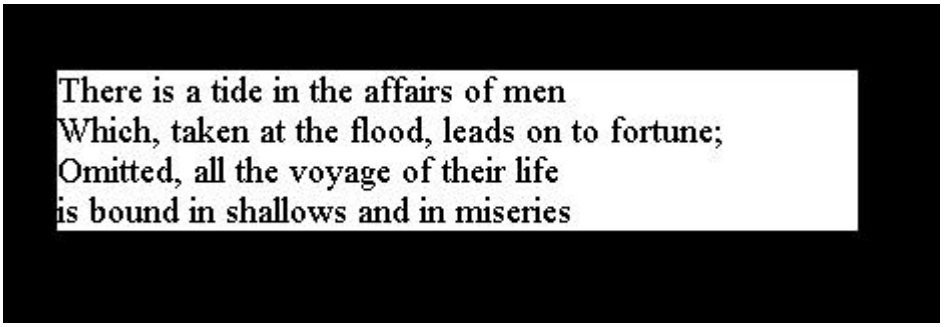
cgalign('l/c/r','t/c/b') This command sets the alignment mode for the horizontal and for the vertical. The horizontal mode can either be 'l' for left alignment, 'c' for centering or 'r' for right alignment. The vertical mode can either be 't' for top alignment, 'c' for centering or 'b' for bottom alignment.

This command has already been used to align rectangles but it is also used to align text. In the following example we shall use pixel co-ordinates and draw a rectangle on the screen upon which we shall draw some text…

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgpencol(1,1,1)
>> cgrect(0,0,400,80)
>> cgfont('Times',20)
>> cgpencol(0,0,0)
```

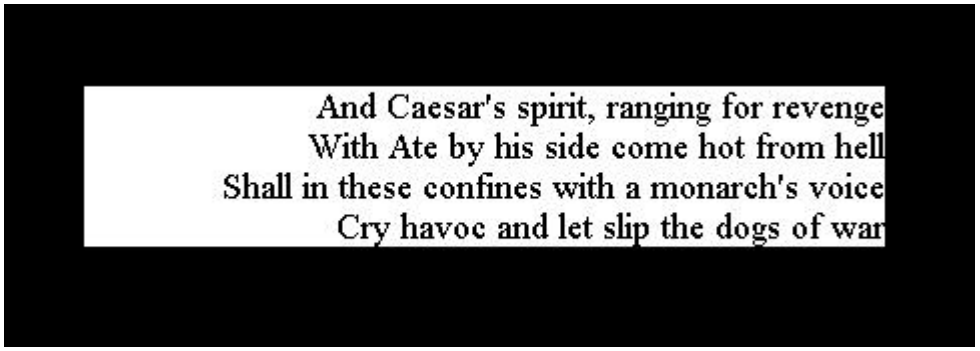
We have opened the graphics and drawn a 400 x 80 pixel white rectangle centred on the centre of the screen. Then we loaded 'Times' font at a height of 20 pixels and set the drawing colour to black. Of course, nothing appears on the display yet because it is all on the offscreen area.

```
>> cgalign('l','t')
>> cgtext('There is a tide in the affairs of men,',-200,40)
>> cgtext('Which, taken at the flood, leads on to fortune;',-200,20)
>> cgtext('Omitted, all the voyage of their life',-200,0)
>> cgtext('is bound in shallows and in miseries',-200,-20)
>> cgflip(0,0,0)
```



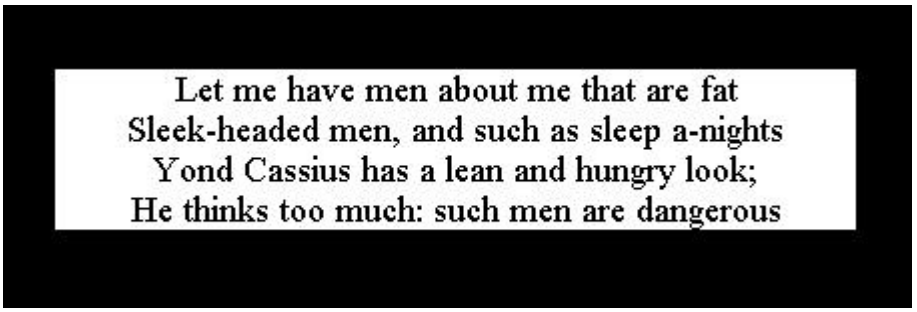
You should see that the text is all neatly aligned with the left hand side of the white box along the vertical line x = -200. This is because we selected left horizontal alignment. The vertical alignment mode was top alignment so the top of the first line was on the y = 40 line, the top of the white box. Now let us try something else…

```
>> cgpencol(1,1,1)
>> cgalign('c','c')
>> cgrect(0,0,400,80)
>> cgpencol(0,0,0)
>> cgalign('r','t')
>> cgtext('And Caesar''s spirit, ranging for revenge',200,40)
>> cgtext('With Ate by his side come hot from hell,',200,20)
>> cgtext('Shall in these confines with a monarch''s voice',200,0)
>> cgtext('Cry havoc and let slip the dogs of war',200,-20)
>> cgflip(0,0,0)
```



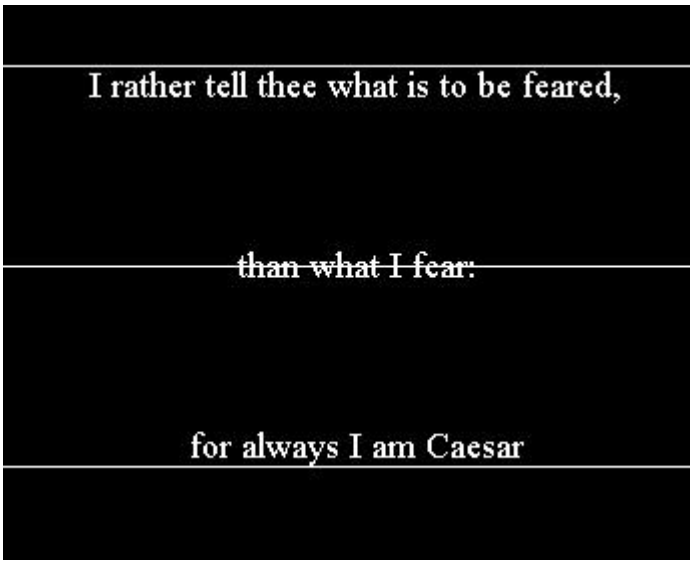
This time you should see that the text is all neatly aligned with the right hand side of the white box along the vertical line x = 200. This is because we selected right horizontal alignment. Notice especially above that the apostrophes within the string must be typed twice as in **Caesar''s** and in **monarch''s**. This is to differentiate them from the sign for the end of the string. If you make a mistake here and just put a single apostrophe, Matlab snivels and prints out an error message. Alright, now let us try centering…

```
>> cgpencol(1,1,1)
>> cgalign('c','c')
>> cgrect(0,0,400,80)
>> cgpencol(0,0,0)
>> cgalign('c','t')
>> cgtext('Let me have men about me that are fat,',0,40)
>> cgtext('Sleek-headed men, and such as sleep a-nights',0,20)
>> cgtext('Yond Cassius has a lean and hungry look;',0,0)
>> cgtext('He thinks too much: such men are dangerous',0,-20)
>> cgflip(0,0,0)
```



The next example shows vertical alignment. Three horizontal lines are drawn and then three lines of text. The top of the first text is aligned with the first line, the centre of the second text with the second and the bottom of the third text with the third…

```
>> cgpencol(1,1,1)
>> cgdraw(-320,100,320,100)
>> cgdraw(-320,0,320,0)
>> cgdraw(-320,-100,320,-100)
>> cgalign('c','t')
>> cgtext('I rather tell thee what is to be feared,',0,100)
>> cgalign('c','c')
>> cgtext('than what I fear:',0,0)
>> cgalign('c','b')
>> cgtext('for always I am Caesar',0,-100)
>> cgflip(0,0,0)
```



Sprites

A sprite is a rectangular graphics area that you can create, draw into and then manipulate. If you have a complicated piece of graphics that you want to use again and again you can create a sprite for it and then use the sprite each time rather than reconstructing the graphics over and over again. One example of this might be a background for a scene that you want to present a stimulus on. If the background is complicated then create it once in a sprite and then draw it each time you need to redraw your graphics. Another use for sprites is to hold images, which will be described in a later section.

The command to make a sprite is rather lengthy…

cgmakesprite(Key,Width,Height <,R,G,B>)

The **R,G,B** arguments are optional.

Key Your identification number for this sprite. This can be any number from 1 to 10,000. You use this Key to identify this sprite. If a sprite with this identification number already exists then this new sprite will replace it.

Width,Height The width and height of the rectangular area. This is in whatever units you are currently using, either pixels or degrees of visual angle as set by the **cgscale** command.

R,G,B If you want to initialise the rectangular area to a flat colour, you can specify the colour here.

The command to delete a sprite is quite succinct…

cgfreesprite(Key)

Key The identification number of the sprite you want to delete.

The command to draw a sprite is also straightforward…

cgdrawsprite(Key,x,y<,w,h>)

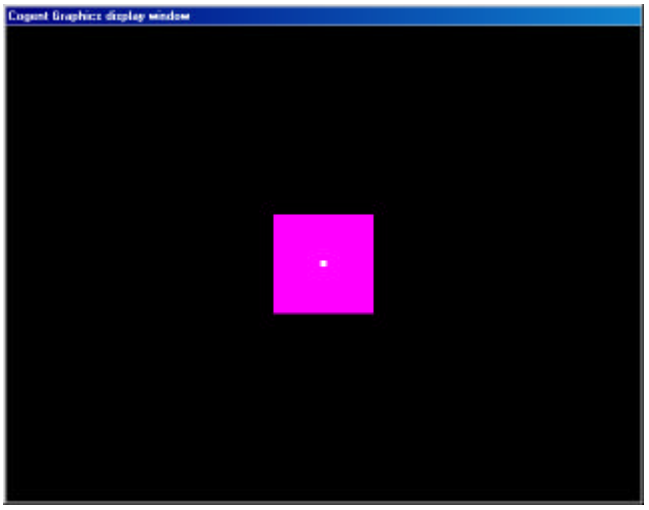
Key Defines the sprite you want to draw.

x,y Where you want to draw it. The current alignment mode as set by **cgalign** is used to position the sprite relative to these co-ordinates.

w,h You may optionally choose to scale the sprite to a new size on the destination. These values specify the width and height of the copied sprite.

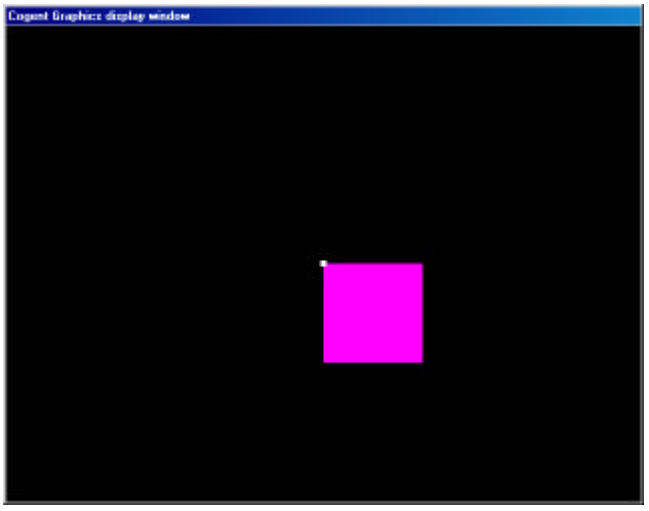
The next example creates a sprite and copies it onto the screen…

>> **cgloadlib**
>> **cgopen(1,0,0,0)**
GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> **cgpencol(1,1,1)**
>> **cgmakesprite(123,100,100,1,0,1)**
>> **cgdrawsprite(123,0,0)**
>> **cgrect(0,0,6,6)**
>> **cgflip(0,0,0)**



We open the graphics as usual and set the drawing colour to white (1,1,1). Then we make a sprite with Key 123, 100 pixels square, initialised to colour 1,0,1 (magenta). Then we draw the sprite onto the centre (co-ordinates 0,0) of the offscreen area. We also mark the centre of the offscreen area with a small white square of side 6 pixels before the **cgflip** command. You should see that the square magenta sprite is centred under the white square. This is because the default alignment is horizontal and vertical centering. Now set the alignment mode to left and top…

>> **cgalign('l','t')**
>> **cgdrawsprite(123,0,0)**
>> **cgalign('c','c')**
>> **cgrect(0,0,6,6)**
>> **cgflip(0,0,0)**



Drawing into sprites

You can select a sprite as the destination for drawing commands using the **cgsetsprite** command. From then on, all drawing commands will occur in that sprite. Remember to reset the destination back to the offscreen area though when you are finished by issuing a **cgsetsprite(0)** command.

cgsetsprite(Key) This command selects the sprite with ID 'Key' for subsequent drawing operations. If 'Key' is zero then the offscreen area will receive drawing operations.

This next exercise creates and selects a sprite that is one quarter the size of the screen…

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgmakesprite(78,320,240,0,0,0)
>> cgsetsprite(78)
```

Then we perform some graphics commands…

```
>> x = [-50   0   50];
>> y = [-25 61 –25];
>> cgpencol(1,0,0)
>> cgrect(0,0,160,120)
>> cgpencol(0,1,0)
>> cgellipse(0,0,160,120,'f')
>> cgpencol(1,1,0)
>> cgpolygon(x,y)
>> cgpencol(0,0,1)
>> cgdraw(-160,120,160,-120)
>> cgdraw(-160,-120,160,120)
>> cgpencol(1,1,1)
>> cgdraw(-90,0)
>> cgdraw(90,0)
>> cgdraw(0,-70)
>> cgdraw(0,70)
>> cgpencol(0,0,0)
>> cgfont('Arial',14)
>> cgtext('To sleep, perchance to dream',0,0)
```

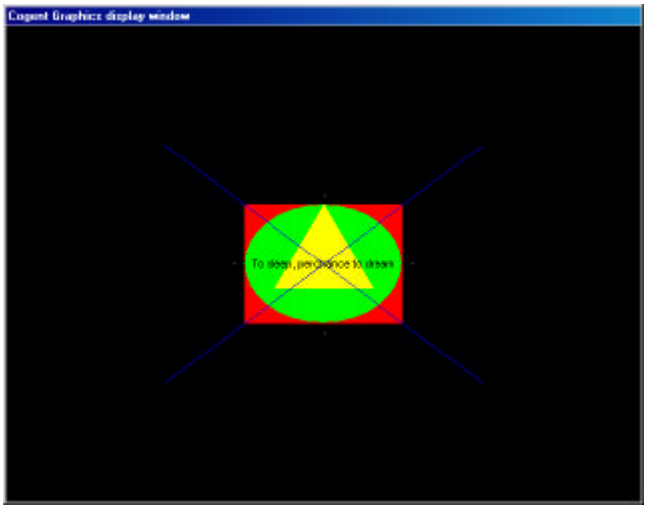
The lines above employ all the drawing commands described up to now apart from one, the **cgdrawsprite** command. We start by drawing a red rectangle with **cgrect** then draw a green filled ellipse with **cgellipse**. Then we draw a yellow triangle with **cgpolygon**. All these figures are centred on the centre of the sprite. Next we draw blue lines across the principal diagonals of the sprite using **cgdraw** and we use **cgdraw** again to draw four white dots. Finally we draw some text using **cgtext**. The point of this rather tedious procedure is to demonstrate that all these commands can be redirected to the sprite we have selected (sprite 78)

Now select the offscreen area as the destination and clear it with a **cgflip** command.
The screen should remain black at this point. This is just to convince you that the offscreen buffer really is clear at this point.

>> **cgsetsprite(0)**
>> **cgflip(0,0,0)**

Now we draw sprite 78 into the offscreen area with a **cgdrawsprite** and display it…

>> **cgdrawsprite(78,0,0)**
>> **cgflip(0,0,0)**



Now for the **cgdrawsprite** command. Let us create and select a new sprite the same size as the screen…

>> **cgmakesprite(79,640,480)**

You may find that you get a warning message when you try to execute the **cgmakesprite** command:-

>> **cgmakesprite(79,640,480)**
WRN GPrim:gAddRAS System rather than video memory used

This warning occurs because your graphics card does not have sufficient memory resources to create the sprite. Instead, system memory has been used. This will still work although the performance may be compromised.

However, you may be able to get round this if you make your screen smaller from the Display control panel. Try resizing your screen to 800 x 600 pixels and start again. The section immediately below tells you how you can do this:-

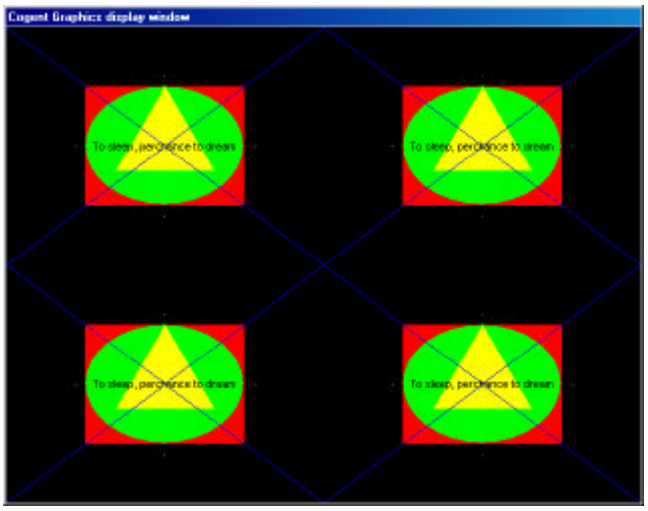
> You can open the **Display Properties** dialog box at the **Settings** tab by clicking **Start**, pointing to **Settings**, clicking **Control Panel**, double-clicking **Display**, and then clicking the **Settings** tab. Then, in **Screen area**, click the desktop size you require (800 x 600).

Once you have successfully created the new sprite you can select it and copy the previous sprite into it four times…

>> **cgsetsprite(79)**
>> **cgdrawsprite(78,-160,120)**
>> **cgdrawsprite(78,160,120)**
>> **cgdrawsprite(78,-160,-120)**
>> **cgdrawsprite(78,160,-120)**

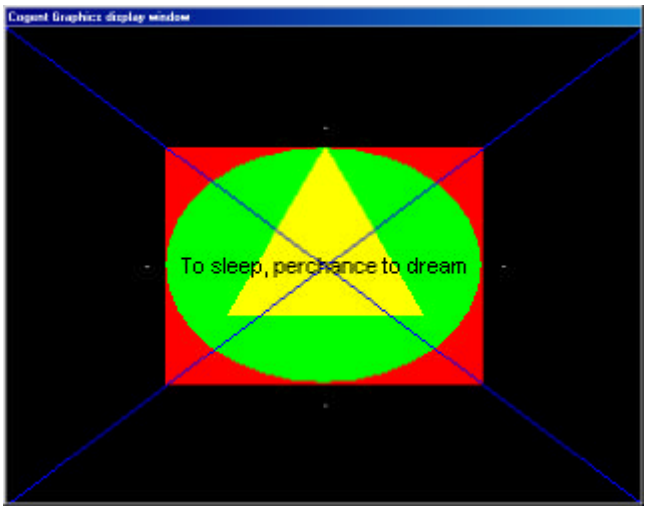
Select the offscreen area as the destination again and clear it with a **cgflip** command just to make sure that the offscreen buffer really is clear at this point. Then draw sprite 79 into the offscreen area and display it…

>> **cgsetsprite(0)**
>> **cgflip(0,0,0)**
>> **cgdrawsprite(79,0,0)**
>> **cgflip(0,0,0)**



Finally, we can also scale the sprite if we want:-

>> **cgsetsprite(0)**
>> **cgdrawsprite(78,0,0,640,480)**
>> **cgflip**



Transparency

Transparency allows us to create a sprite with an irregular outline. In order to do this we associate a 'transparent colour' with the sprite. You may choose the transparent colour from the following; black, red, green, yellow, blue, magenta, cyan, white. The **cgtrncol** command selects the transparent colour:-

cgtrncol(Key,TCol)

Key The identification number of the sprite whose transparent colour you want to set.

TCol Transparent colour to use for this sprite. If you omit the TCol value transparency is switched off for the sprite. You may choose from the following values:-

'n'	selects black	(0,0,0)
'r'	selects maximum red	(1,0,0)
'g'	selects maximum green	(0,1,0)
'y'	selects maximum yellow	(1,1,0)
'b'	selects maximum blue	(0,0,1)
'm'	selects maximum magenta	(1,0,1)
'c'	selects maximum cyan	(0,1,1)
'w'	selects maximum white	(1,1,1)

Consider the following example…

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgpencol(0,0,1)
>> cgrect
>> cgpencol(1,1,0)
>> cgfont('Arial',240)
>> cgtext('SPQR',0,0)
>> cgfont('Arial',40)
>> cgpencol(1,0,0)
```

We open the graphics, clear the offscreen area to blue and then draw some yellow text on the screen. We then select a smaller font and set the pen to red.

```
>> cgmakesprite(1,600,40,0,1,0)
>> cgsetsprite(1)
>> cgtext('Senatus Populusque Romanus',0,0)
>> cgtrncol(1,'g')
>> cgsetsprite(0)
>> cgdrawsprite(1,0,-50)
>> cgtrncol(1)
>> cgdrawsprite(1,0,50)
>> cgflip
```

Here we make sprite 1. This sprite is initialised to a green background and we draw some red text into it. Then we set the transparent colour for the sprite to green. We then copy it into the offscreen buffer. Then we switch off the transparent colour for the sprite and copy sprite 1 again, above. Then we display what we have drawn with the **cgflip** command…

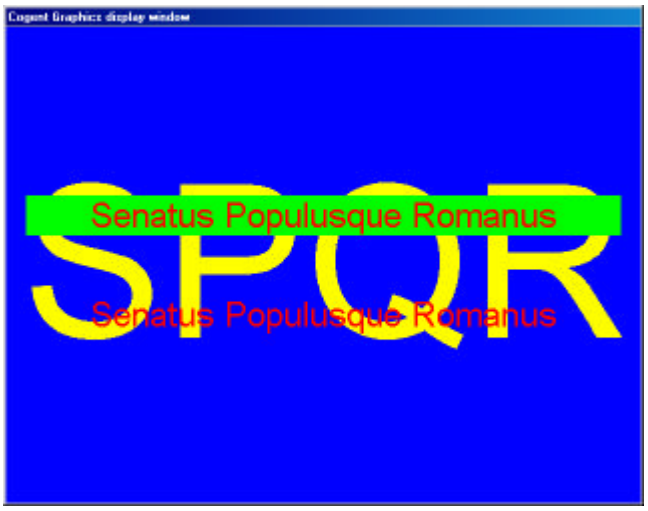


Image files

You may load up an image file into a sprite using the **cgloadbmp** command. The image file must be in uncompressed .BMP format.

cgloadbmp(Key,'Filename'<,Width,Height>)

The **Width** and **Height** arguments are optional.

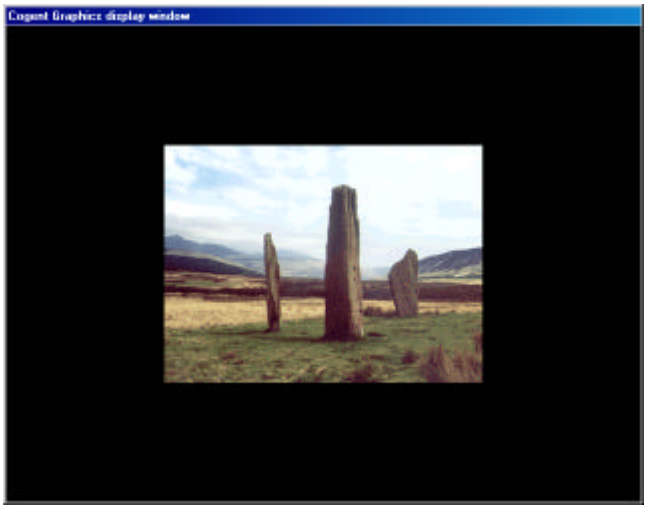
Key is your ID number for the sprite which will be created.

Filename is the full filename of the BMP file to load

Width,Height are the dimensions of the sprite you want to create. The image will be scaled to fit these dimensions exactly. By choosing different dimensions to the image you can achieve some special effects such as squeezing or stretching the image. You can also enlarge the image. However, if you try to make the image smaller the result can be very poor. You may set either **Width** or **Height** to be zero. If you do this, the aspect ratio of the image is used to calculate the width or height from the other, supplied dimension.

You may download sample BMP files named Demo.bmp and Demo2.bmp from the website for use in the following examples. You should copy them into the same directory that you run matlab from…

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgloadbmp(1,'Demo.bmp')
>> cgdrawsprite(1,0,0)
>> cgflip(0,0,0)
```



The Demo.bmp image is 320 x 240 pixels, one quarter of the size of the screen. However, we can scale it to full screen size if we require…

>> **cgloadbmp(1,'Demo.bmp',640,480)**
>> **cgdrawsprite(1,0,0)**
>> **cgflip(0,0,0)**

Notice here that we re-used sprite number 1 for this example. The old, smaller image was discarded when we reloaded sprite 1 with this bigger image.

We could also have used either of the commands below to scale the bitmap to full screen width or height; supplying a zero for width or height means that the aspect ratio of the image should be maintained when calculating the size of the image:-

>> **cgloadbmp(1,'Demo.bmp',640, 0)**
>> **cgloadbmp(1,'Demo.bmp', 0,480)**

You may find that you get a warning message with the **cgloadbmp** command:-

>> **cgloadbmp(1,'Demo.bmp',640,480)**
WRN GPrim:gAddRAS System rather than video memory used.

This warning occurs because your graphics card does not have sufficient memory resources to create the sprite. Instead, system memory has been used. This will still work although the performance may be compromised. Now let us try an example using transparency…

>> **cgloadbmp(2,'Demo2.bmp')**
>> **cgdrawsprite(1,0,0)**
>> **cgdrawsprite(2,0,0)**
>> **cgflip(0,0,0)**

We have loaded a new image file; Demo2.bmp into a new sprite, number 2 and then drawn sprite 1 and then sprite 2. You can see that the new image is of a face on a red background. Now let us try that again, setting the transparent colour to red…

>> **cgtrncol (2,'r')**
>> **cgdrawsprite(1,0,0)**
>> **cgdrawsprite(2,0,0)**
>> **cgflip(0,0,0)**

Loading an image from the matlab workspace

You can also create and display an image using the matlab workspace. The image may be defined as a direct mode rgb image or as a palette defined image.

cgloadarray(Key,ImgWid,ImgHgt,PixVal<,PalRGB><,SprWid,SprHgt>)

The **PalRGB** argument is required for a palette based image.
The **SprWid** and **SprHgt** arguments are optional

Key is your ID number for the sprite which will be created.

ImgWid,ImgHgt are the width and height of the image in matlab.

PixVal Matlab array that defines the colour of each pixel in the matlab workspace image. There are 'n' (=ImgWid x ImgHgt) pixels. In the case of an RGB image this array has dimension (n x 3); n rows and three columns. The three columns represent the red, green and blue components of each pixel respectively and take values from 0 to 1. In the case of a palette based image this array has just n values. Each value represents an index into the palette (0 to 255) which is defined by the PalRGB argument.

PalRGB Matlab array defining the palette of a palette based image. The appropriate type of PixVal argument must be supplied. This array has dimension (m x 3); m rows and three columns. There may be up to 256 rows in the array. The three columns represent the red, green and blue components respectively of each palette entry.

SprWid,SprHgt are the width and height of the sprite to create. The image will be stretched in the x and y dimensions to exactly fit the sprite

If the image is w pixels wide and h pixels high there will be (w x h) pixels in the image and so there must be (w x h) elements in each of the R, G and B arrays. For example, if you want to create an image that is 80 pixels wide by 70 pixels high there must be (80 x 70 = 5600) pixel values in the PixVal array.

The first value in PixVal corresponds to the top left pixel in the image and as you go on through the array elements you move horizontally right along each row of pixels. When you get to the right hand side of a row you then move to the leftmost pixel of the next row down in the image and then continue horizontally to the right. The diagram below shows how array elements 1 to 9 represent pixels in a three by three pixel image:-

1	2	3
4	5	6
7	8	9

Now for an example:-

>> **PixVal = [0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1; 0 0 0];**

Here we have created our 9x3-element RGB arrays which we will use to define a three by three pixel image. So the 9 elements in turn have RGB components and colours as follows:-

Array Element	R	G	B	Colour
1	0	0	0	Black
2	1	0	0	Red
3	0	1	0	Green
4	1	1	0	Yellow
5	0	0	1	Blue
6	1	0	1	Magenta
7	0	1	1	Cyan
8	1	1	1	White
9	0	0	0	Black

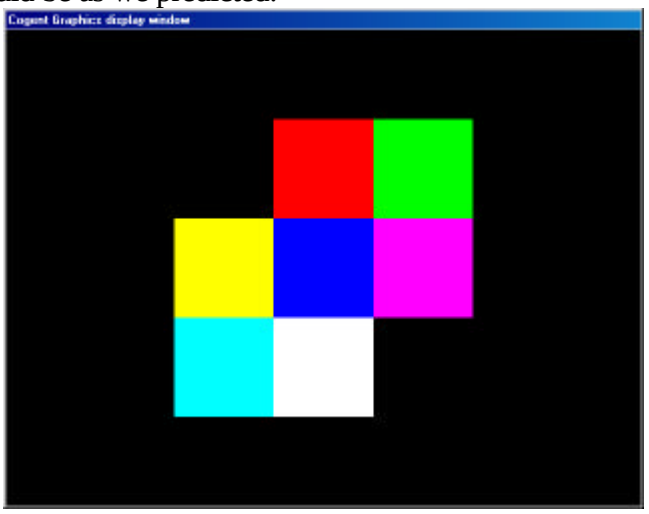
Translating back to pixels in an image we get:-

Black	Red	Green
Yellow	Blue	Magenta
Cyan	White	Black

So let us now create this image:-

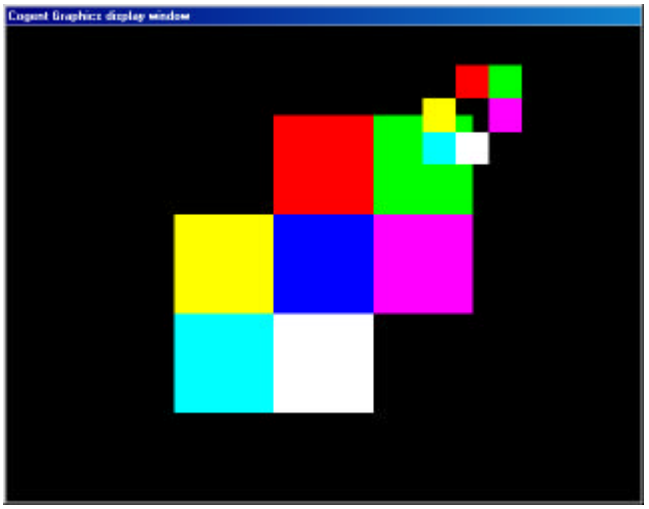
>> **PixVal = [0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1; 0 0 0];**
>> **cgloadlib**
>> **cgopen(1,0,0,0)**
GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> **cgloadarray(1,3,3,PixVal,300,300)**
>> **cgdrawsprite(1,0,0)**
>> **cgflip(0,0,0)**

The arrays we created were for a 3 by 3 pixel image which is rather tiny, so we used the **SprWid** and **SprHgt** arguments of the cgloadarray command to blow the image up to a 300 by 300 pixel sprite. The results should be as we predicted:-



To demonstrate transparency we can create another smaller sprite from our array but we shall make the central blue square transparent. We can then draw it over the image above to illustrate the transparency of the central square…

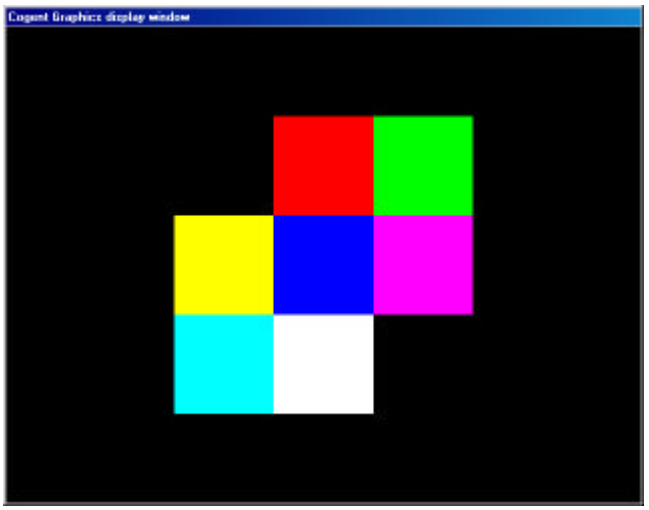
>> **cgloadarray(2,3,3,PixVal,100,100)**
>> **cgtrncol(2,'b')**
>> **cgdrawsprite(1,0,0)**
>> **cgdrawsprite(2,150,150)**
>> **cgflip(0,0,0)**



It is also possible to define the same image using a palette based system…

>> **PixVal = [0 1 2 3 4 5 6 7 8];**
>> **PalRGB = [0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1; 0 0 0];**
>> **cgloadlib**
>> **cgopen(1,0,0,0)**
GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> **cgloadarray(1,3,3,PixVal,PalRGB,300,300)**
>> **cgdrawsprite(1,0,0)**
>> **cgflip(0,0,0)**

Here we have defined our pixel values as palette indices 0 to 8 and we have also defined a palette which contains the 9 colours defined previously. The result should be the same.



Blitting sprites

You can copy a complete sprite using the **cgdrawsprite** command described earlier. However, you may want to copy just a rectangular section from one sprite to another. This process is called 'blitting'. 'Blit' is an abbreviation for '**BL**ock **I**mage **T**ransfer' which simply means copying a rectangular block from one image to another.

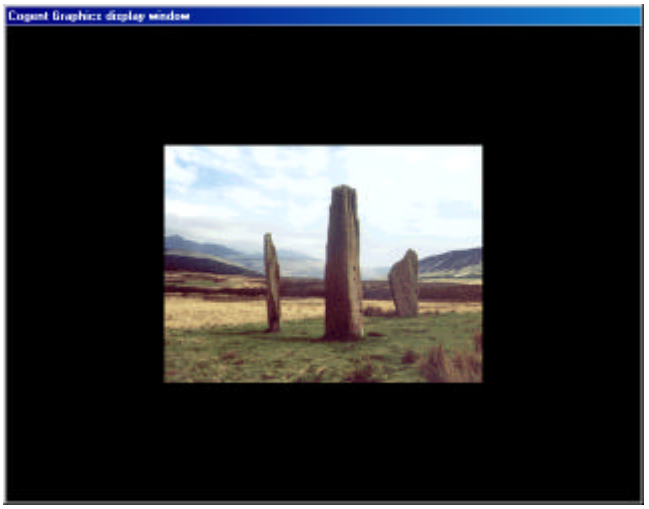
You can use the **cgblitsprite** command to copy an arbitrary rectangle from a sprite to the current destination:-

cgblitsprite(Key,srcx,srcy,srcw,srch,dstx,dsty<,dstw,dsth>)

Key	The identification number for the source sprite.
srcx,srcy	The source position for the rectangle. This is modified by the current alignment mode as set by **cgalign**.
srcw,srch	The width and height of the rectangle to copy.
dstx,dsty	The destination position for the rectangle. Again this is modified by the current alignment mode as set by **cgalign**.
dstw,dsth	You may optionally set the destination width and height if it is different from the source width and height. In this way you can scale the rectangle if you require.

Consider the following example:-

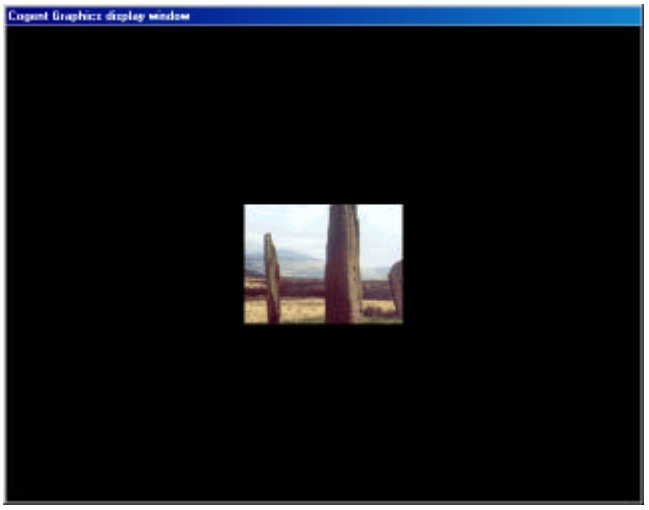
```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgloadbmp(1,'Demo.bmp')
>> cgdrawsprite(1,0,0)
>> cgflip(0,0,0)
```



Here we have loaded a familiar image into sprite 1 and copied it onto the screen using the **cgdrawsprite** command.

Now let us use the **cgblitsprite** command to copy just the central rectangle. The Demo.bmp image is 320x240 pixels so we shall set the alignment mode to horizontal and vertical centering and then blit a rectangle 160 x 120 pixels:-

>> **cgalign('c','c')**
>> **cgblitsprite(1,0,0,160,120,0,0)**
>> **cgflip(0,0,0)**

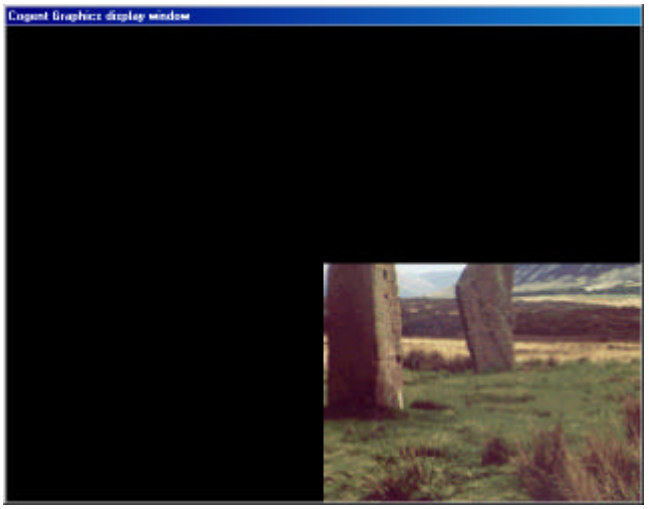


We can also use the **cgblitsprite** command to scale the rectangle up to the full screen size of 640x480 pixels:-

>> **cgblitsprite(1,0,0,160,120,0,0,640,480)**
>> **cgflip(0,0,0)**

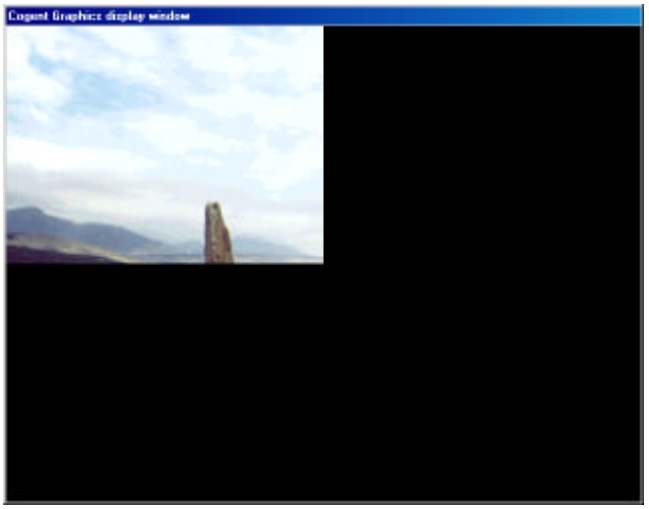
The **cgblitsprite** command obeys the **cgalign** setting. The previous examples demonstrate how the command operates when alignment is set to horizontal and vertical centering. However, we can also demonstrate other alignment modes. Here we shall set the alignment mode to left/top alignment and then blit the lower right corner of the image into the lower right corner of the screen, scaling the rectangle by a factor of 2 from 160x120 pixels to 320x240 pixels:-

>> **cgalign('l','t')**
>> **cgblitsprite(1,0,0,160,120,0,0,320,240)**
>> **cgflip(0,0,0)**



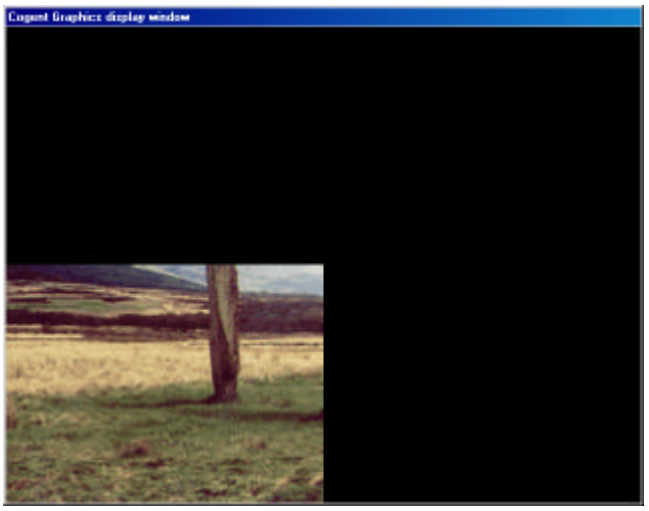
Next we can set the alignment mode to right/bottom and copy the upper left corner using exactly the same **cgblitsprite** settings once we have reset the alignment mode:-

>> **cgalign('r','b')**
>> **cgblitsprite(1,0,0,160,120,0,0,320,240)**
>> **cgflip(0,0,0)**



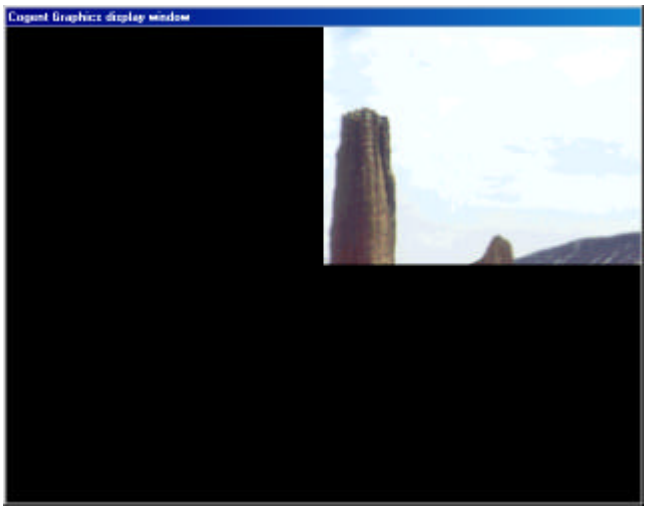
We can copy the lower left corner by setting the alignment mode to right/top:-

>> **cgalign('r','t')**
>> **cgblitsprite(1,0,0,160,120,0,0,320,240)**
>> **cgflip(0,0,0)**



Finally copy the upper right corner by setting the alignment mode to left/bottom:-

>> **cgalign('l','b')**
>> **cgblitsprite(1,0,0,160,120,0,0,320,240)**
>> **cgflip(0,0,0)**



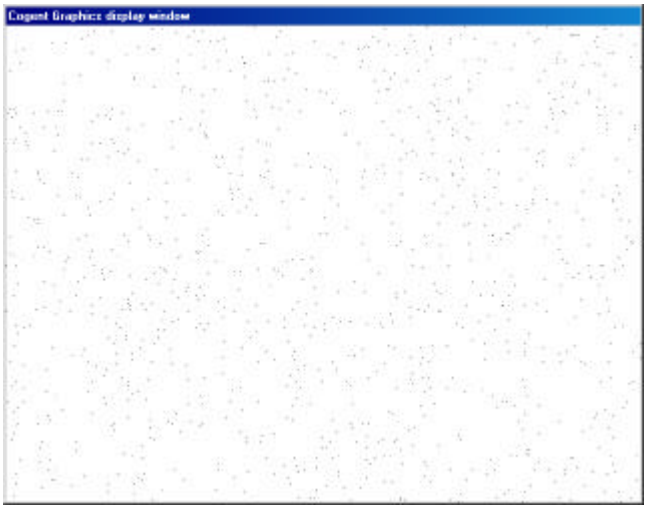
Drawing multiple items

Sometimes you may wish to draw many hundreds of items. You could do this by drawing each item individually in a loop but matlab is rather slow at loop processing and you will get much faster drawing speeds if you can draw all your items in a single function call. The following functions accept arrays for their arguments to make this possible:-

cgdraw **cgrect** **cgellipse** **cgarc**

Consider the following example:-

```
>> x = rand(1,1000)*640 - 320;
>> y = rand(1,1000)*480 - 240;
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgflip(1,1,1)
>> cgpencol(0,0,0)
>> cgdraw(x,y)
>> cgflip
```



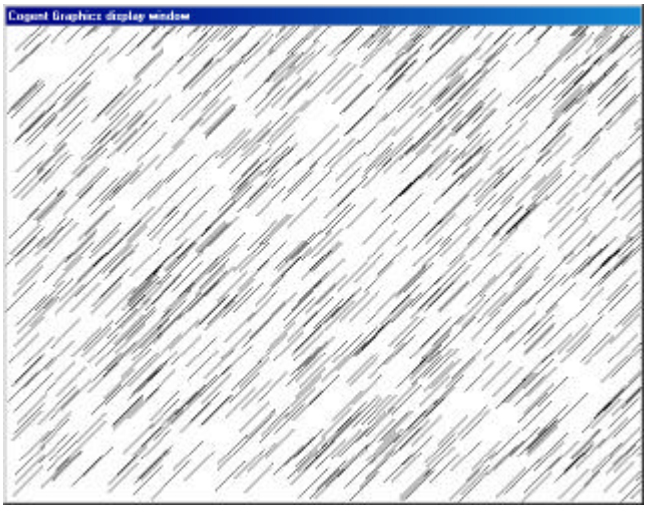
The first two lines set up two arrays, each containing 1000 elements. The 'x' array contains random values from –320 to 320 and the 'y' array contains random values from –240 to 240. We then open a 640x480 pixel screen, clear the background to white (1,1,1) and set the drawing colour to black (0,0,0). We then use the **cgdraw** command to draw our array of 1000 randomly positioned pixels with a single call.

We can use the same method to draw lines. Let us now set up two further arrays; x2 and y2 which form short lines extending the single pixels into short diagonal lines, 30 pixels long:-

```
>> x2 = x + 30;
>> y2 = y + 30;
```

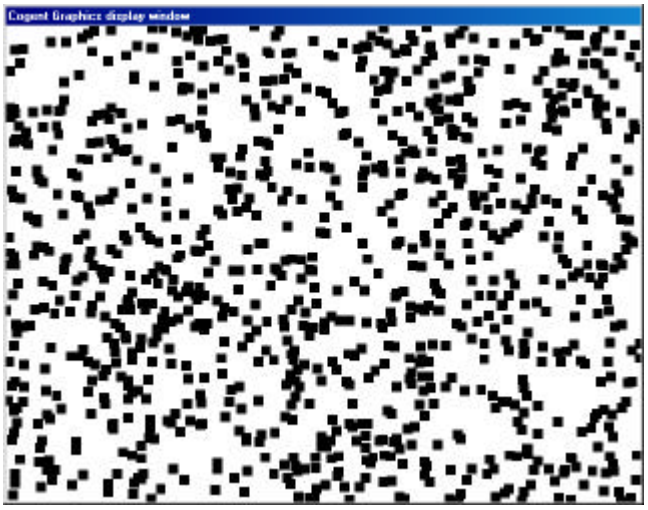
We can now draw these 1000 lines in a very simple way…

>> **cgflip(1,1,1)**
>> **cgpencol(0,0,0)**
>> **cgdraw(x,y,x2,y2)**
>> **cgflip**



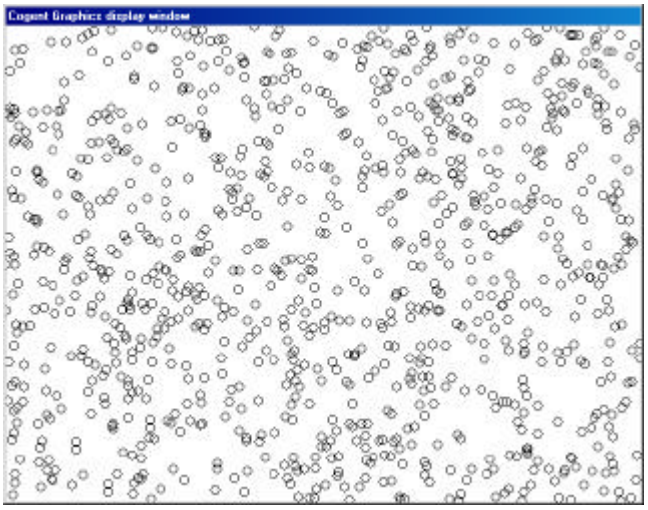
In a similar way we can draw rectangles. Let us draw 1000 squares of width 10 pixels:-

>> **w(1:1000) = 10;**
>> **h(1:1000) = 10;**
>> **cgflip(1,1,1)**
>> **cgpencol(0,0,0)**
>> **cgrect(x,y,w,h)**
>> **cgflip**



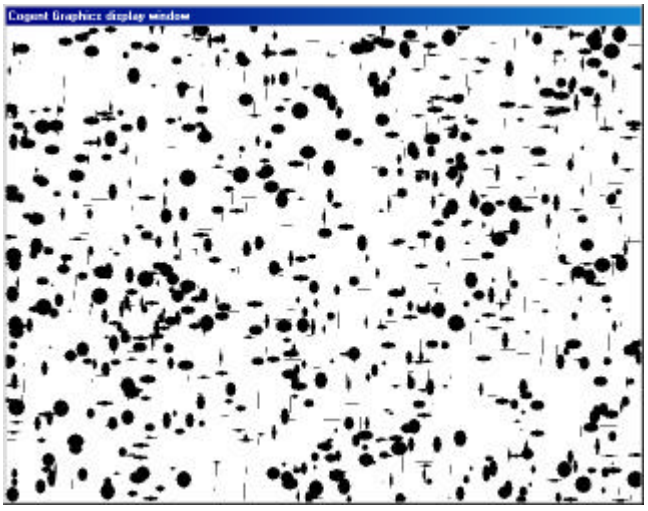
Or we can draw hollow circles instead of squares using the **cgellipse** command:-

```
>> cgflip(1,1,1)
>> cgpencol(0,0,0)
>> cgellipse(x,y,w,h)
>> cgflip
```



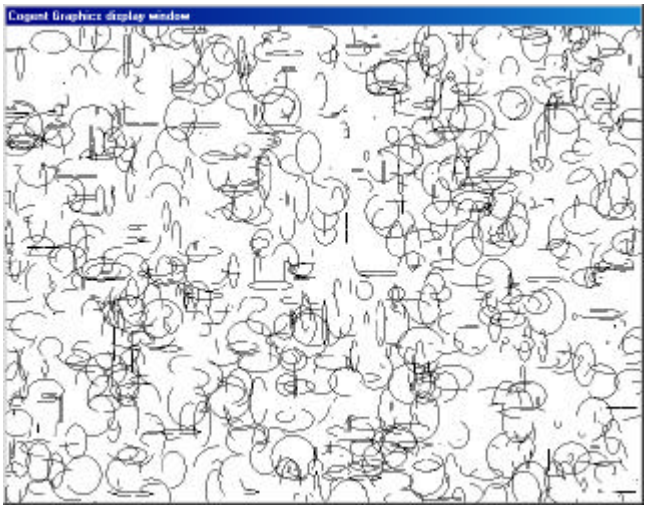
Of course, we do not need to have the dimensions all the same. Let us set the widths and heights to random values between 1 and 20 pixels and draw filled ellipses instead:-

```
>> w(1:1000) = 1 + rand(1,1000)*19;
>> h(1:1000) = 1 + rand(1,1000)*19;
>> cgflip(1,1,1)
>> cgpencol(0,0,0)
>> cgellipse(x,y,w,h,'f')
>> cgflip(1,1,0)
```



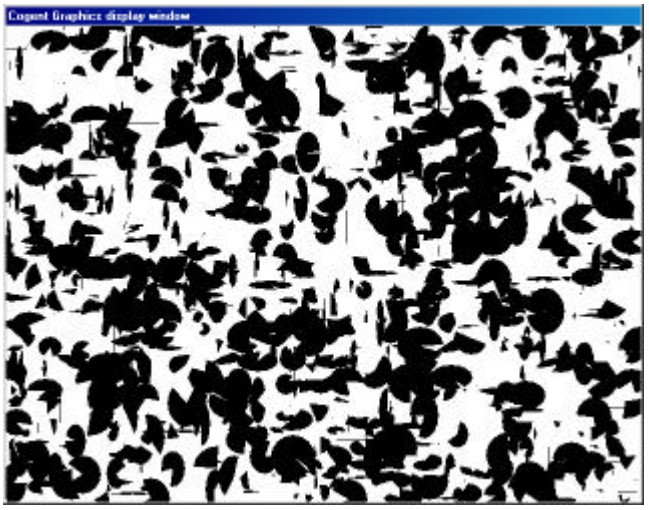
We can do a similar thing with the **cgarc** command (this time with larger ellipses):-

```
>> w(1:1000) = 1 + rand(1,1000)*49;
>> h(1:1000) = 1 + rand(1,1000)*49;
>> a1(1:1000) = rand(1,1000)*360;
>> a2(1:1000) = rand(1,1000)*360;
>> cgpencol(0,0,0)
>> cgpenwid(1)
>> cgarc(x,y,w,h,a1,a2)
>> cgflip(1,1,1)
```



Or with filled sectors:-

```
>> cgarc(x,y,w,h,a1,a2,'S')
>> cgflip(1,1,1)
```



It is also possible to specify a different colour for each item. To do this we must define a colour array with the same number of entries as there are items to be drawn. When we are in direct colour mode, a colour is defined as an RGB triplet:- [1 0 0] represents red for example.

So, to define an array of 'n' colours we must define an (nx3) matrix ('n' rows and 3 columns):-

[1 0 0; 0 0 1; 0 1 0; 1 1 1]

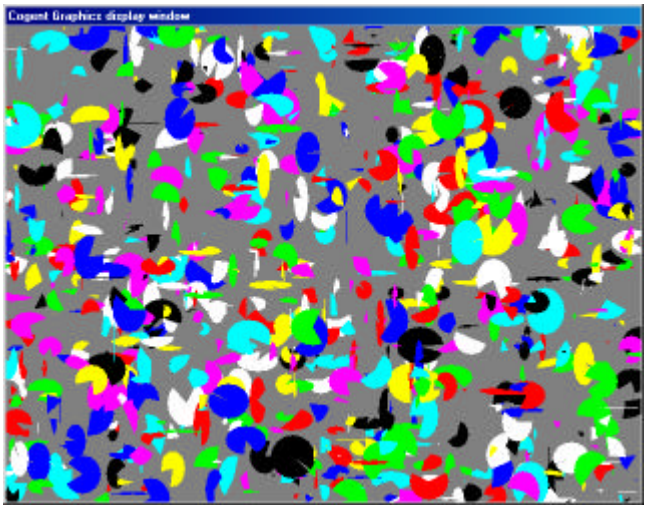
The above matrix defines four colours;- 1 0 0 (red), 0 0 1 (blue), 0 1 0 (green) and 1 1 1 (white).

We can define an rgb array with 1000 values where r, g and b are individually either 0 or 1 using the following lines:-

>> **rgb(1:1000,1) = fix(rand(1000,1)+.5);**
>> **rgb(1:1000,2) = fix(rand(1000,1)+.5);**
>> **rgb(1:1000,3) = fix(rand(1000,1)+.5);**

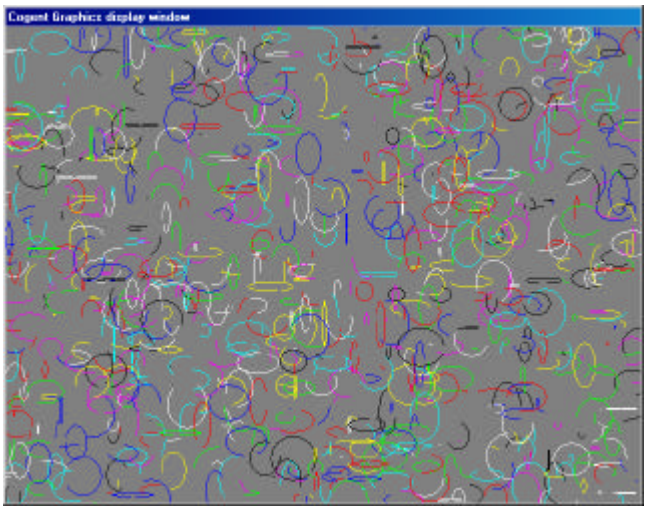
Now let us draw those filled sectors again, this time in different colours and on a grey background:-

>> **cgflip(.5,.5,.5)**
>> **cgarc(x,y,w,h,rgb,'S')**
>> **cgflip(.5,.5,.5)**



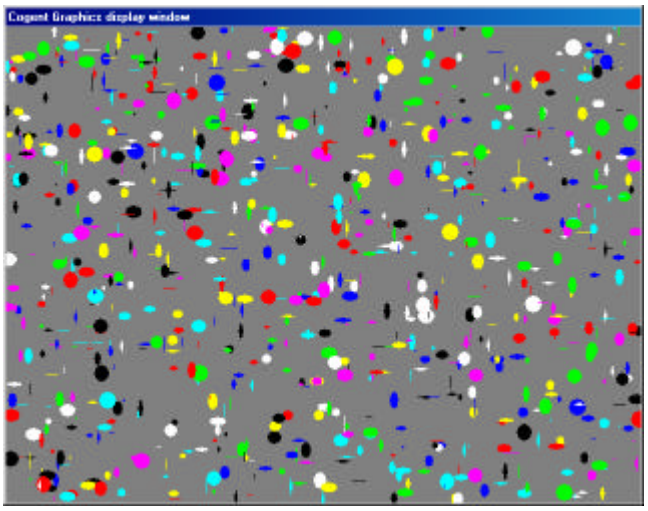
We can use this method to draw all the other figures in different colours too. For example, hollow arcs:-

>> **cgarc(x,y,w,h,rgb)**
>> **cgflip(.5,.5,.5)**



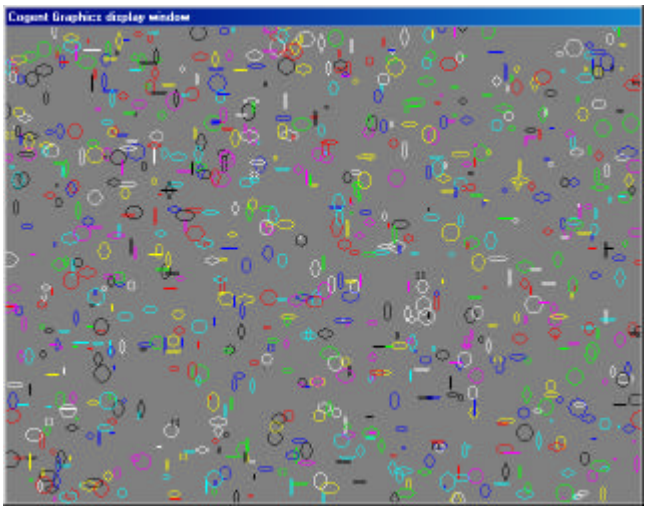
Filled ellipses (resetting the ellipse size to small again):-

>> **w(1:1000) = 1 + rand(1,1000)*19;**
>> **h(1:1000) = 1 + rand(1,1000)*19;**
>> **cgellipse(x,y,w,h,'f')**
>> **cgflip(.5,.5,.5)**



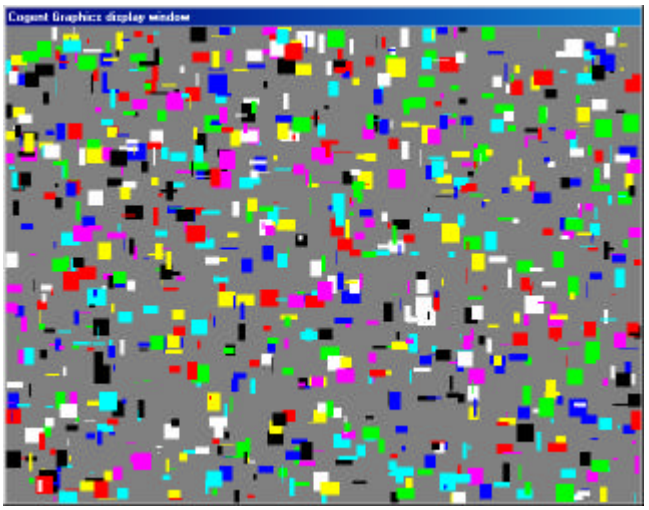
Hollow ellipses:-

>> **cgellipse(x,y,w,h,rgb)**
>> **cgflip(.5,.5,.5)**



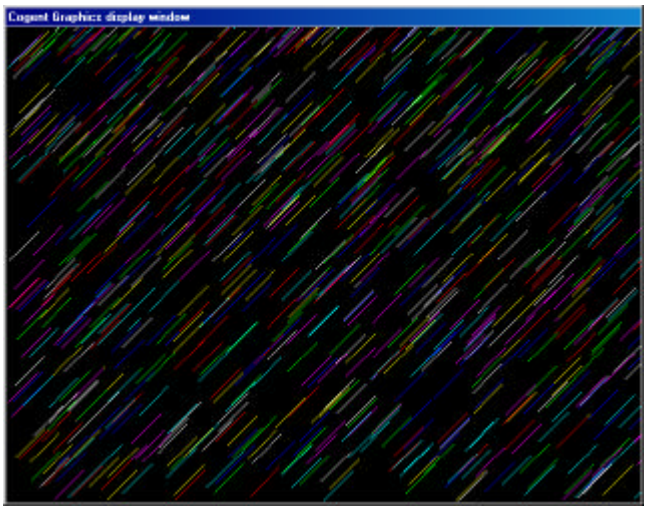
Rectangles:-

>> **cgrect(x,y,w,h,rgb)**
>> **cgflip(.5,.5,.5)**



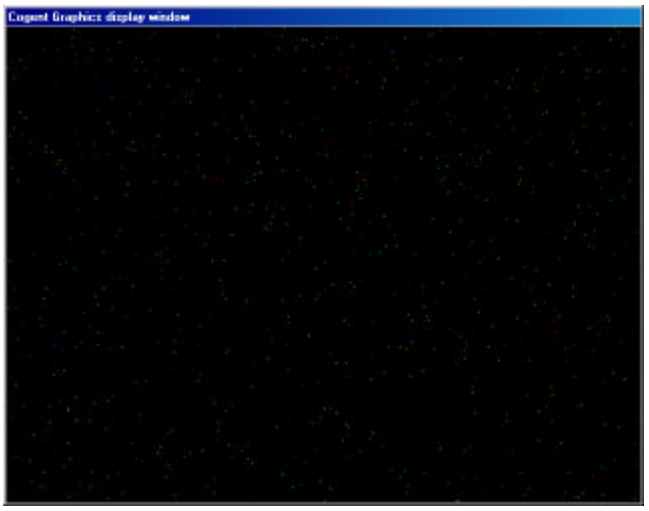
Lines (these show up better on a black background):-

>> **cgflip(0,0,0)**
>> **cgdraw(x,y,x2,y2,rgb)**
>> **cgflip(0,0,0)**



And individual points (these also show up better on a black background):-

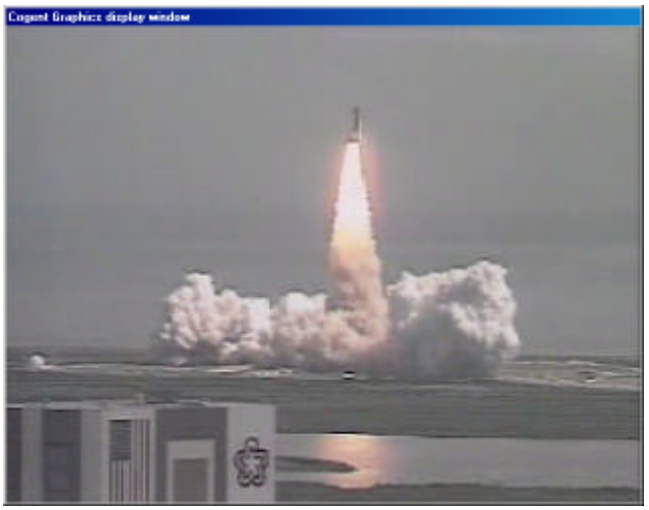
>> **cgflip(0,0,0)**
>> **cgdraw(x,y,rgb)**
>> **cgflip(0,0,0)**



Movies

Movie files should be in Microsoft .avi format. You should download the sample movie file "movie.avi" from the website and then follow the commands below:-

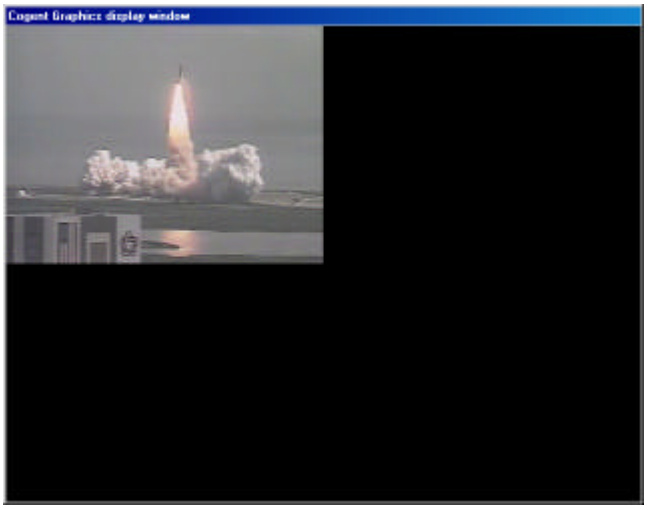
```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 59.96Hz
>> cgopenmovie(1,'movie.avi')
>> cgplaymovie(1)
```



We first open a graphics window. and then load the movie file 'movie.avi' as movie number 1. Then we play the movie. The movie plays moving graphics and audio. While the movie is playing nothing else can be done; we cannot "interrupt" the movie until it has stopped.

The destination rectangle can be positioned and scaled as well:-

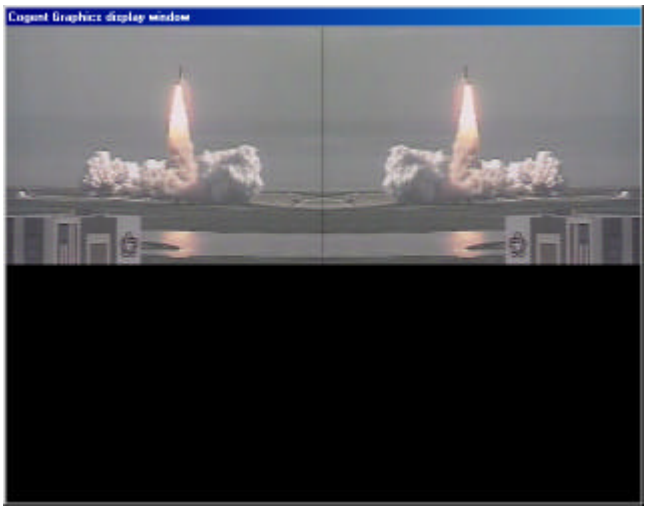
```
>> cgalign('r','b')
>> cgflip(0,0,0)
>> cgplaymovie(1,0,0,320,240)
```



Here the alignment mode is set to 'right', 'bottom' and the movie is scaled to 320x240 pixels (half the normal size). The **cgflip** command clears the previous screen.

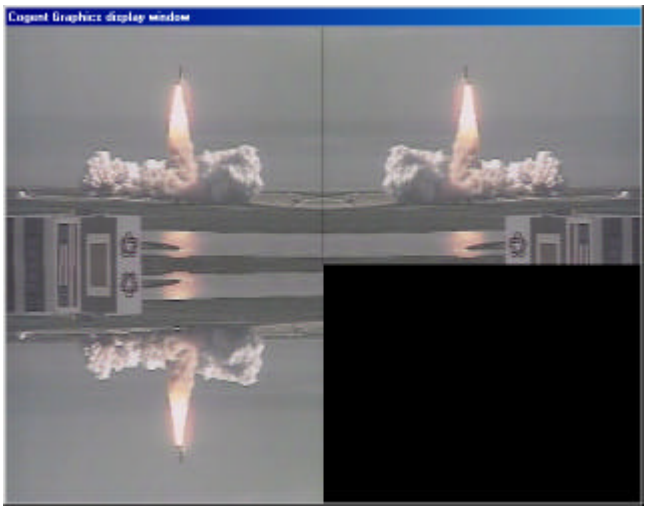
We can even flip the image horizontally by setting the width to be -320 instead of 320:-

>> **cgalign('l','b')**
>> **cgplaymovie(1,0,0,-320,240)**



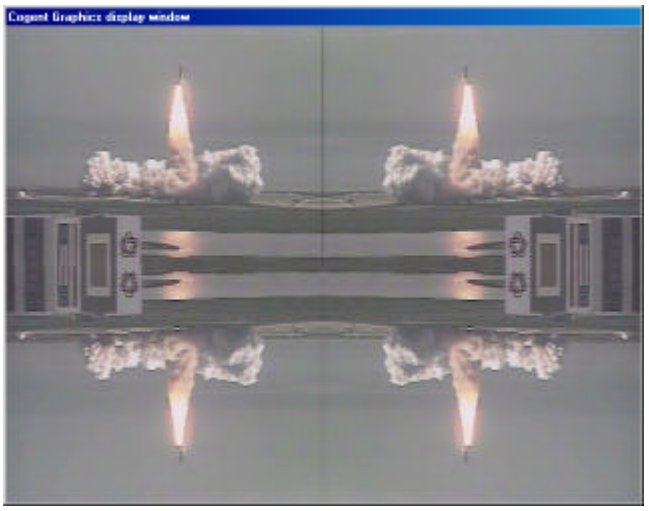
In a similar way we can flip vertically by using -240 for the height, rather than 240:-

>> **cgalign('r','t')**
>> **cgplaymovie(1,0,0,320,-240)**



Finally, to complete our sequence, we can flip horizontally and vertically at the same time. This is the same as a 180 degree rotation:-

>> **cgalign('l','t')**
>> **cgplaymovie(1,0,0,-320,-240)**



When we have finished we should free up the memory taken up by the movie by issuing a **cgshutmovie(1)** command. You may open more than one movie at a time if you want to play them in quick succession but it is probably better to just keep one open at a time if you can.

Palette mode tutorial

Opening and closing graphics

This tutorial is for palette mode graphics. You should set your Windows desktop to 256 colour mode using the display control panel (see the "Palette mode" section in the introduction).

Assuming you have correctly set up your Windows desktop in 256 colour mode you should open Cogent Graphics as before:-

```
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 85.03Hz
```

The "Display:640x480x8 85.03Hz" should now indicate that we are in 8-bit or palette mode (the 8 in 640x480x***8*** indicates this). You can close the graphics screen with the **cgshut** command:-

```
>> cgshut
```

The window that you opened above was a sub-window on the desktop. To open a full-screen window in palette mode, use the command below:-

```
>> cgopen(1,8,0,1)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 75.00Hz
```

This time the whole display goes black. Remember that at this point you can do one of two things:-

1/ Minimise the black full-screen window by holding down the Alt key and pressing the tab key. The minimised window is now represented by a blank rectangle on the system toolbar. You can restore it by clicking on this blank toolbar. You can issue the **cgshut** command from the matlab console to shut the window. The blank rectangle on the system toolbar may not disappear until you select it.

2/ Exit the program by holding down the Ctrl key and pressing the C key.

At this point you may want to try altering the refresh rate. You can use the DirectX control panel to set the fixed refresh rate, as described in the Introduction under the heading "DirectX and refresh rate". Close your graphics screen first with a **cgshut** command, set the forced refresh rate with the DirectX control panel and then repeat the **cgopen(1,8,0,1)** command above. You may find that your requested refresh rate has not been delivered. For example you may select 90Hz and only receive 85Hz. If this happens it means that DirectX has "done the best it can" and set the fastest refresh rate for you.

The colour table

In palette mode there are 256 palette entries which can be set to whatever colour you require. When you first open a screen it is filled with palette entry number 0. Initially, all the colour entries are set to black.

```
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 85.03Hz
```

You can set the colour for palette entry number 0 using the **cgcoltab** command. Here we use it to set palette entry 0 to colour 1,0,1 (magenta).

```
>> cgcoltab(0,1,0,1)
```

However, the screen does not change until we issue the command to make the current colour table visible, **cgnewpal**:-

```
>> cgnewpal
```

Preparing the colour table with **cgcoltab** and displaying it with **cgnewpal** is analogous to preparing graphics with the drawing commands and then displaying them with **cgflip**. You can obtain a timestamp for displaying the new palette in the following way:-

```
>> S = cgnewpal
```

The **cgcoltab** command can also accept an array to define the r, g and b values…

```
>> rgb = [1 1 1];
>> cgcoltab(0,rgb)
```

Is equivalent to:-

```
>> cgcoltab(0,[1 1 1])
```

You can also set several colours at the same time with the **cgcoltab** command:-

```
>> r = [0 1 0 1 0 1 0 1];
>> g = [0 0 1 1 0 0 1 1];
>> b = [0 0 0 0 1 1 1 1];
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 85.03Hz
>> cgcoltab(5,r,g,b)
>> cgnewpal
```

Or, using a single array…

```
>> rgb = [0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1];
>> cgcoltab(5,rgb)
```

In both of the above examples we have set up eight palette entries using the r, g and b arrays starting at entry number 5:-

```
 5 = 0,0,0 = black
 6 = 1,0,0 = red
 7 = 0,1,0 = green
 8 = 1,1,0 = yellow
 9 = 0,0,1 = blue
10 = 1,0,1 = magenta
11 = 0,1,1 = cyan
12 = 1,1,1 = white
```

Now let us use the **cgflip** command to set the display surface to each of the palette indices in turn. Note that in palette mode the cgflip command takes a single palette index rather than the RGB triplet which it uses in direct colour mode. Remember that the **cgflip** command clears the offscreen buffer to the index so the display actually lags behind the command by one command as shown below in italics:-

```
>> cgflip(5)    Onscreen is now 0 (black) offscreen is 5 (black)
>> cgflip(6)    Onscreen is now 5 (black) offscreen is 6 (red)
>> cgflip(7)    Onscreen is now 6 (red) offscreen is 7 (green)
>> cgflip(8)    Onscreen is now 7 (green) offscreen is 8 (yellow)
>> cgflip(9)    Onscreen is now 8 (yellow) offscreen is 9 (blue)
>> cgflip(10)   Onscreen is now 9 (blue) offscreen is 10 (magenta)
>> cgflip(11)   Onscreen is now 10 (magenta) offscreen is 11 (cyan)
>> cgflip(12)   Onscreen is now 11 (cyan) offscreen is 12 (white)
>> cgflip(5)    Onscreen is now 12 (white) offscreen is 5 (black)
```

At this point the display is filled with palette index 12 which is currently set to white. Let us now change the colour to yellow:-

```
>> cgcoltab(12,1,1,0)
>> cgnewpal
```

cgnewpal and cgflip synchronisation

In order to avoid flickering, the **cgnewpal** and **cgflip** commands are usually both synchronised with your monitor display. Each command waits until the end of the current display frame before changing the palette or copying the offscreen area to the visible screen respectively. This could cause a problem if you wanted to complete both actions on the same display frame. To get around this, the **cgflip** command can operate in *immediate* mode in which case it does not wait for the end-of-frame period. You can therefore issue the following command sequence:-

```
S = cgnewpal
cgflip('I')
```

The **cgnewpal** command above is synchronised to the end-of frame period and the timestamp for the display is taken at that command. The **cgflip** command takes place immediately afterwards. It is not possible to obtain a timestamp for the **cgflip** command in immediate mode.

However, your graphics card may not support immediate mode for **cgflip**. If that is the case you will receive an error message when you issue the command. If that happens your only alternative is to run cgnewpal in immediate mode instead:-

```
S = cgflip
cgnewpal('I')
```

This is generally undesirable because the **cgflip** command will take some time to execute and so when the **cgnewpal** command is issued you may get some flicker at the top of the screen. The previous configuration is much better because the **cgnewpal** command executes quickly in comparison to **cgflip**.

If you still have problems with flicker you must redesign your code.

Drawing commands

You have just seen how the **cgflip** command takes a palette index rather than an RGB triplet when in palette mode. Similarly the **cgpencol** command also takes a palette index when in palette mode:-

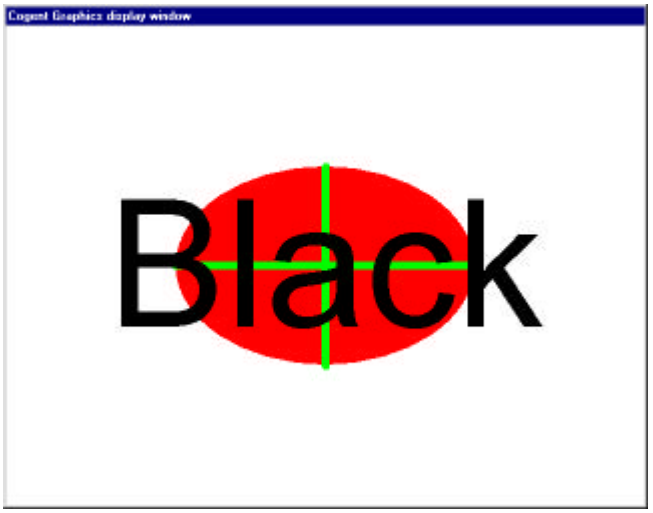
```
>> rgb = [0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1];
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 85.03Hz
>> cgcoltab(10,rgb)
>> cgnewpal
```

Here we have set up the following palette index colours:-

10 = black	12 = green	14 = blue	16 = cyan
11 = red	13 = yellow	15 = magenta	17 = white

Now let us issue some drawing commands:-

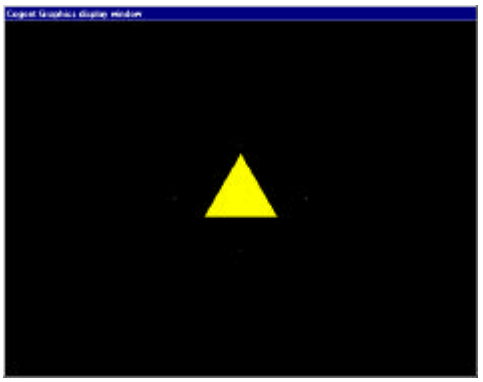
```
>> cgpencol(17)
>> cgrect
>> cgpencol(11)
>> cgellipse(0,0,300,200,'f')
>> cgpenwid(8)
>> cgpencol(12)
>> cgdraw(-150,0,150,0)
>> cgdraw(0,-100,0,100)
>> cgpencol(10)
>> cgfont('Arial',200)
>> cgtext('Black',0,0)
>> cgflip(10)
```



We have cleared the whole display to palette index 17 (white) and then drawn an ellipse in palette index 11 (red). Then we drew a cross in palette index 12 (green) and then some text in palette index 10 (black). Finally we issued a **cgflip** command to display our graphics and set the offscreen area to palette index 10 (black).

The drawing commands **cgdraw**, **cgrect** and **cgellipse** are similar whether in palette mode or in direct mode as seen above, but there is a difference when it comes to drawing multiple items with different colours for each item. You should read "Drawing multiple items" later on in this section to find out how the commands change when in palette mode. Now let us continue with some other drawing commands…

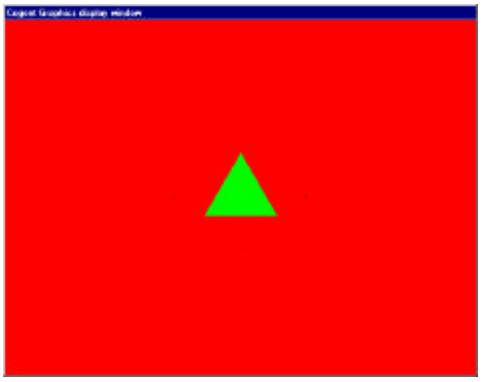
```
>> x = [-50  0  50];
>> y = [-25 61 -25];
>> cgpencol(13)
>> cgpolygon(x,y)
>> cgpencol(17)
>> cgdraw(-90,0)
>> cgdraw(90,0)
>> cgdraw(0,-70)
>> cgdraw(0,70)
>> cgflip
```



First we set up a triangular polygon and draw it in palette index 13 (yellow) and then we draw four dots in palette index 17 (white). Now let us change these colours…

```
>> cgcoltab(10,1,0,0)
>> cgcoltab(13,0,1,0)
>> cgcoltab(17,0,0,0)
>> cgnewpal
```

We have changed palette index 10 (the background) to 1,0,0 (red), palette index 13 (the triangle) to 0,1,0 (green) and palette index 17 (the dots) to 0,0,0 (black):-

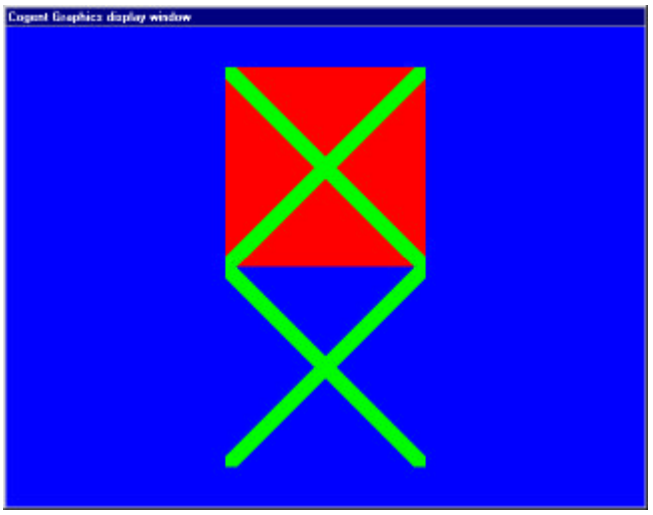


Sprites and transparency

The command to make a sprite in palette mode is similar to that for direct colour mode but the RGB colour is replaced by a palette index. The command to set the transparent colour for a sprite also takes a palette index rather than a TCol value.

Consider the example below:-

```
>> rgb = [0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1];
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 85.03Hz
>> cgcoltab(10,rgb)
>> cgnewpal
>> cgflip(14)
>> cgmakesprite(1,200,200,11)
>> cgsetsprite(1)
>> cgpencol(12)
>> cgpenwid(16)
>> cgdraw(-100,-100,100,100)
>> cgdraw(-100,100,100,-100)
>> cgsetsprite(0)
>> cgdrawsprite(1,0,100)
>> cgtrncol(1,11)
>> cgdrawsprite(1,0,-100)
>> cgflip(10)
```



We load our palette with the colours black, red, green, yellow, blue, magenta, cyan and white starting at palette index 10 and then use the **cgflip** command to set the offscreen area to palette index 14 (blue). Then we make our sprite setting a background palette index of 11 (red) and draw a cross in palette index 12 (green). We then copy this sprite to the offscreen area. Then we set the transparent palette index for our sprite to be 11 and copy the sprite again, underneath. The upper sprite is drawn with the red background, but the lower sprite appears without a red background as the red background was palette index 11, which we made transparent.

Image files

Images are now supported in palette mode. However, the image must be palette based and an extra argument, StartIndex, is required for the cgloadbmp command, shown here in its full form:-

[RASKey,RGBPal] = cgloadbmp(Key,'Filename',StartIndex,<,Width,Height>)

The new variables are:-

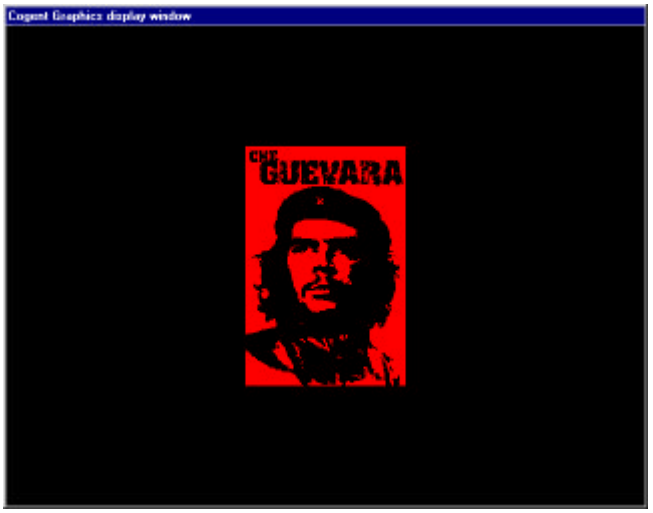
RASKey This is the key for the underlying raster for the sprite to be created. This variable is only of use if you want to use some of the lower level graphics commands.

RGBPal This contains the palette for the image in an array of size (nx3) where there are 'n' rows (one for each palette entry in the image) and 3 columns (red, green and blue palette entry values respectively).

StartIndex You can choose where the image colours will be in your display palette; this value (0 to 255) defines the starting palette entry for the image colours.

Now for an example (You can download sample BMP file Demo3.bmp from the website):-

>> **cgopen(1,0,0,0)**
GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 74.59Hz
>> **[r,rgb]=cgloadbmp(1,'Demo3.bmp',10);**
>> **cgdrawsprite(1,0,0)**
>> **cgflip(0)**
>> **cgcoltab(10,rgb)**
>> **cgnewpal**



We loaded file Demo3.bmp into sprite 1 with the colours starting at palette index 10. The image palette colours were returned in array **rgb**. We then drew the sprite and executed a flip to make it visible. Then we loaded the palette array **rgb** into our display palette with the **cgcoltab** command and displayed the colours with **cgnewpal**.

We can check the contents of the **rgb** array:-

```
>> rgb
rgb =
    1    0    0
    0    0    0
```

Here we can see that there are two colours in the Demo3.bmp palette; the first is maximum red (1,0,0) and the second is black (0,0,0). Now let us set up the following colours in the palette:-

10 = 1 0 0 = red 11 = 0 0 0 = black
12 = 0 1 0 = green 13 = 0 0 0 = black
14 = 1 1 0 = yellow 15 = 0 0 0 = black
16 = 0 0 1 = blue 17 = 0 0 0 = black

```
>> rgb = [1 0 0; 0 0 0; 0 1 0; 0 0 0; 1 1 0; 0 0 0; 0 0 1; 0 0 0];
>> cgcoltab(10,rgb)
>> cgnewpal
```

Now make sprites 1 to 4, all from the Demo3.bmp file but respectively starting at palette indices 10, 12, 14 and 16:-

```
>> cgloadbmp(1,'Demo3.bmp',10);
>> cgloadbmp(2,'Demo3.bmp',12);
>> cgloadbmp(3,'Demo3.bmp',14);
>> cgloadbmp(4,'Demo3.bmp',16);
```

Now draw the sprites evenly spaced horizontally and display them:-

```
>> cgdrawsprite(1,-240,0)
>> cgdrawsprite(2,-80,0)
>> cgdrawsprite(3,80,0)
>> cgdrawsprite(4,240,0)
>> cgflip(0)
```

The same Demo.bmp file has now been transposed to four different positions in the palette and each position has been given a different colour.

Loading an image from the matlab workspace

When in palette mode an extra argument, StartIndex, is required when using the cgloadarray command which is shown here in its full form:-

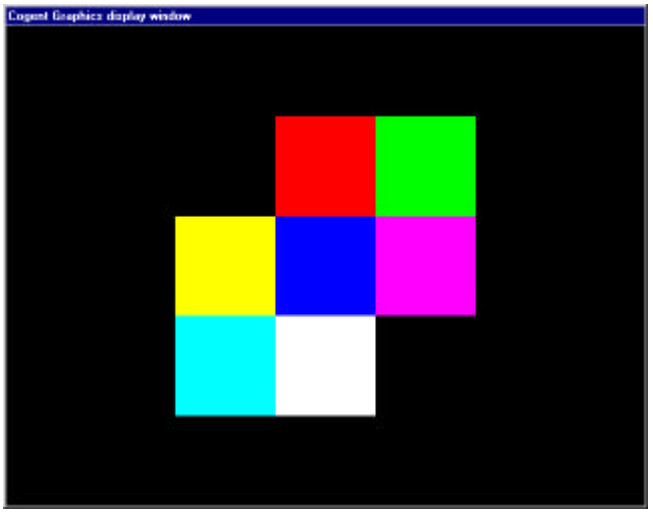
RASKey=>cgloadarray(Key,aw,ah,PixVal<,PalRGB<,StartIndex>><,sw,sh>)

StartIndex You can choose where the image colours will be in your display palette; this value (0 to 255) defines the starting palette entry for the array image colours.

The array must use a palette to define the colours, i.e. the PalRGB argument must be present. This is useful because, once you have defined your PixVal array of palette indices going from 0 to 'n' where 'n' is the size of the PalRGB palette, you can then load that image into a sprite starting at any palette entry you may require, without having to change the individual PixVal elements.

Consider the example below:-

```
>> PixVal = [0 1 2 3 4 5 6 7 8];
>> PalRGB = [0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1; 0 0 0];
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 72.85Hz
>> cgloadarray(1,3,3,PixVal,PalRGB,20,300,300)
>> cgflip(0)
>> cgdrawsprite(1,0,0)
>> cgflip(0)
>> cgcoltab(20,PalRGB)
>> cgnewpal
```



Once again the familiar 3 x 3 colour square appears, this time on a palette mode display. Although the original array uses palette entries 0 to 8 to define the image, we have read it in starting at palette entry 20 so it uses entries 20 to 28 in our display palette. We then use the cgcoltab and cgnewpal commands to set the display colours to match the array palette.

Drawing multiple items

You should first work through the "Drawing multiple items" in the Direct Colour section before starting this description which specifically addresses differences between direct and palette mode when drawing multiple items.

Drawing multiple items in palette mode is very similar to the method used in direct colour mode. If the items are all to be drawn in the same colour there is essentially no difference. Consider the following example:-

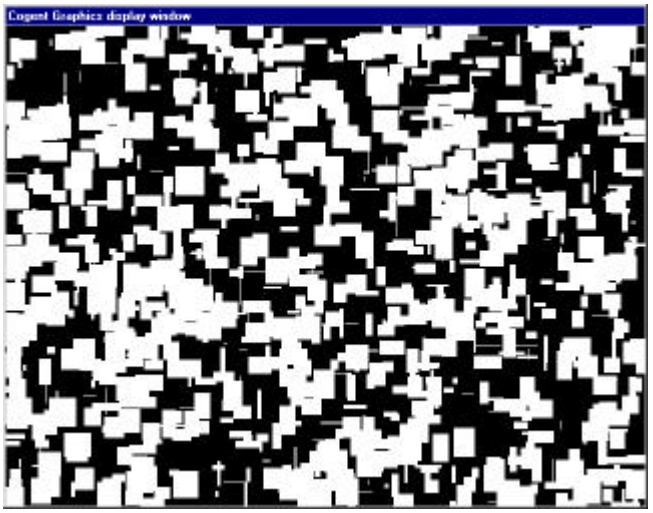
```
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 74.59Hz
>> cgcoltab(0,[0 0 0; 1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1])
>> cgnewpal
```

Here we have opened a palette mode window and set the palette colours 0 to 7 to be red, green, yellow, blue, magenta, cyan and white respectively. Next, let us set up some co-ordinates:-

```
>> x = rand(1,1000)*640 - 320;
>> y = rand(1,1000)*480 - 240;
>> w = 1 + fix(rand(1,1000)*29);
>> h = 1 + fix(rand(1,1000)*29);
```

The x and y arrays each contain 1000 random co-ordinates from (–320 to 320) and (–240 to 240) respectively. The w and h arrays each contain 1000 values from 1 to 30. Now let us use these co-ordinates to draw 1000 rectangles in palette index 7 (white) on a background of palette index 0 (black):-

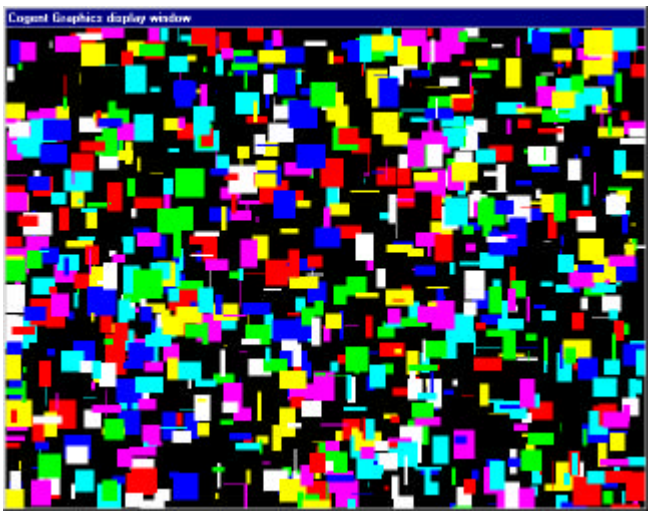
```
>> cgflip(0)
>> cgpencol(7)
>> cgrect(x,y,w,h)
>> cgflip(0)
```



The **cgrect** command used above is identical to that used in direct colour mode. The same holds true for all the other commands for **cgdraw** and **cgellipse** *provided the items are all drawn in the current drawing colour*. If the items are all to be drawn in different colours there is an important difference in palette mode.

Instead of defining a colour as an rgb triplet as we do in direct colour mode, we must define the colours as a single column array (n x 1) of palette indices as shown in the example below:-

```
>> PalInd = 1 + fix(rand(1000,1)*6.99);
>> cgflip(0)
>> cgrect(x,y,w,h,PalInd)
>> cgflip(0)
```

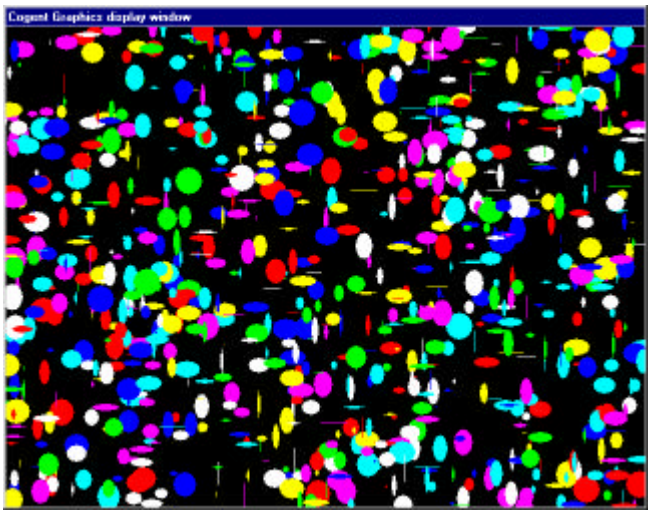


Here we have defined an array, PalInd, which contains a random sequence of 1000 numbers taking values from 1 to 7. This is used to define the palette indices for the rectangles. In a similar way we can draw hollow ellipses:-

```
>> cgflip(0)
>> cgellipse(x,y,w,h,PalInd)
>> cgflip(0)
```

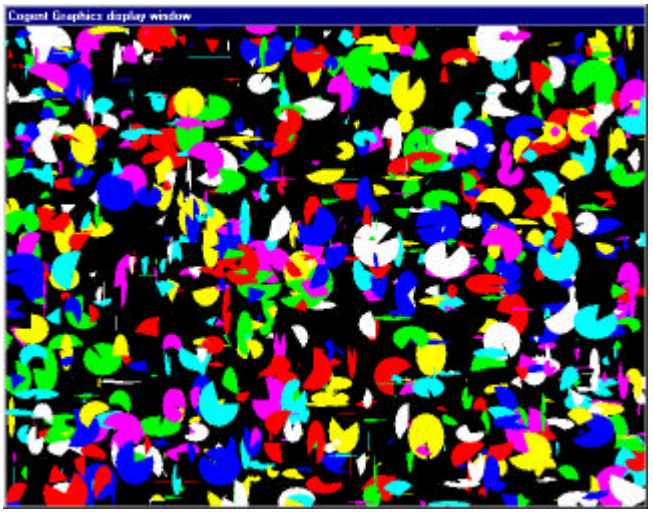
Filled ellipses:-

```
>> cgflip(0)
>> cgellipse(x,y,w,h,PalInd,'f')
>> cgflip(0)
```



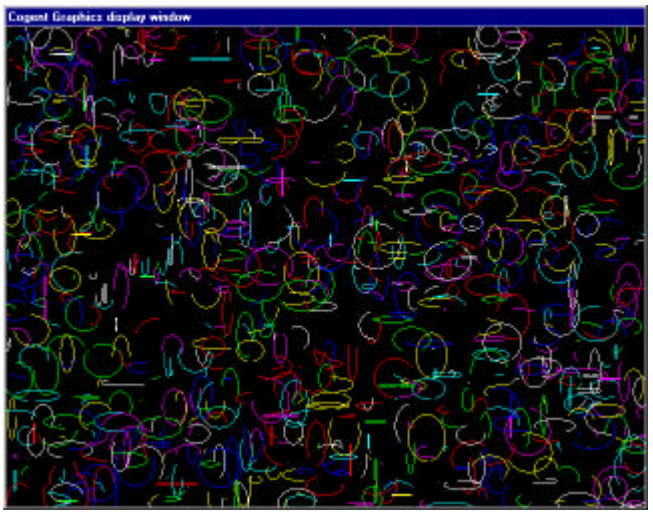
Filled sectors (use larger width and height):-

```
>> w(1:1000) = 1 + rand(1,1000)*49;
>> h(1:1000) = 1 + rand(1,1000)*49;
>> a1(1:1000) = rand(1,1000)*360;
>> a2(1:1000) = rand(1,1000)*360;
>> cgflip(0)
>> cgarc(x,y,w,h,a1,a2,PalInd,'S')
>> cgflip(0)
```



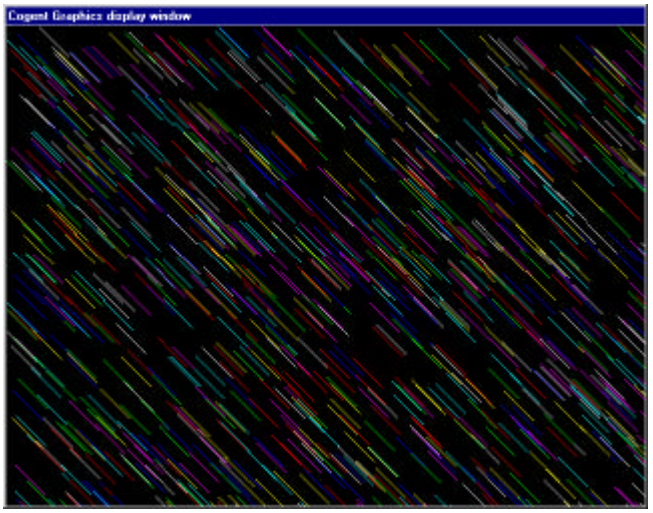
Holow arcs:-

>> **cgflip(0)**
>> **cgarc(x,y,w,h,a1,a2,PalInd)**
>> **cgflip(0)**



Lines:-

>> **x2 = x + 30;**
>> **y2 = y – 30;**
>> **cgflip(0)**
>> **cgdraw(x,y,x2,y2,PalInd)**
>> **cgflip(0)**



Or dots:-

>> **cgflip(0)**
>> **cgdraw(x,y,PalInd)**
>> **cgflip(0)**

Movies

Movies are not currently supported in palette mode.

Further Tutorials

Using the mouse

*Currently the **cgmouse()** function has not been fully integrated with the rest of Cogent. In particular the use of this function may interact undesirably with the Cogent mouse-event logging service. Consequently use this function with caution.*

The **cgmouse()** function provides an efficient and simple way to read and set the mouse pointer position. Because of the nature of this function it is simplest to begin by describing how to set the mouse position. Try the following commands:-

```
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x8 75.31Hz
>> cgmouse(0,0)
```

When you issue the **cgmouse(0,0)** command, the mouse pointer on the screen should jump to the centre of the cogent display window.

Now make it jump to the top right corner:-

```
>> cgmouse(320,240)
```

You can use this form of the command to make the mouse pointer move to the given co-ordinates. The co-ordinates are the local co-ordinates as set up by the **cgscale()** command.

Reading the mouse is also simple:-

```
>> [x,y,bs,bp] = cgmouse;
```

The call above returns the mouse position **(x,y)** in **cgscale()** co-ordinates and the button state **(bs,bp)**. The **bs** value gives the mouse buttons that were down at the instant **cgmouse** was called. The **bp** value gives the mouse buttons that had been pressed (and possible subsequently released) since the previous time **cgmouse** was called.

Mouse buttons are coded as the arithmetic sum of the following values:-

```
1 = left mouse button
2 = middle mouse button
4 = right mouse button
```

In the case of the bs value (buttons down at the time of the call to **cgmouse**) the sum can also include the following components:-

```
 8 = Control button also pressed
16 = Shift button also pressed
```

So if the middle and right buttons were pressed the coded value would be 2 + 4 = 6.

You may extract a particular button value using the matlab **bitand()** function:-

bitand(bp,1) will be non-zero if the left button has been pressed
bitand(bp,2) will be non-zero if the middle button has been pressed
bitand(bp,4) will be non-zero if the right button has been pressed

You do not have to read all four variables each time, the following are all valid alternatives:-

>> **cgmouse;**
>> **x = cgmouse;**
>> **[x,y] = cgmouse;**
>> **[x,y,bs] = cgmouse;**
>> **[x,y,bs,bp] = cgmouse;**

This function is best demonstrated in a loop and so an example script named Mouse.m has been included with the samples which you can download from the website. Here is a listing of Mouse.m:-

```
function Mouse
%
% Move the cursor into the display.
%
% Hit a mouse button to exit
%
fprintf('\nMove the cursor into the display.\n\n');
fprintf('Hit a mouse button to exit\n\n')

cgloadlib
cgopen(1,0,0,0)

gsd = cgGetData('GSD');
if gsd.ScreenBits == 8
   cgcoltab(1,1,1,1);
   cgnewpal
   cgpencol(1)
else
   cgpencol(1,1,1)
end

bp=0;
while ~bp
   [x,y,bs,bp]=cgmouse;
   cgellipse(x,y,100,100,'f')

   if gsd.ScreenBits == 8
      cgflip(0)
   else
      cgflip(0,0,0)
   end
end

cgshut
return
```

The script starts off with some help information which is also printed out as a matter of course when the script is run. Then a window is opened and we use the **`cgGetData`** command to find out whether we are in palette mode **`(gsd.ScreenBits == 8)`** or not. In either case the drawing colour is set to maximum white. Then the value of **bp** is set to zero. The value of **bp** will reflect the state of the mouse buttons and the statement **while ~bp** sets up a loop which will continue until a mouse button is pressed. Inside this loop we read the mouse position and button state and draw a filled ellipse to follow the mouse position. We then issue the **`cgflip`** command, clearing the background to black (the form is slightly different depending on whether or not we are in palette mode). If a mouse button has been pressed we drop out of the loop, close the graphics and return.

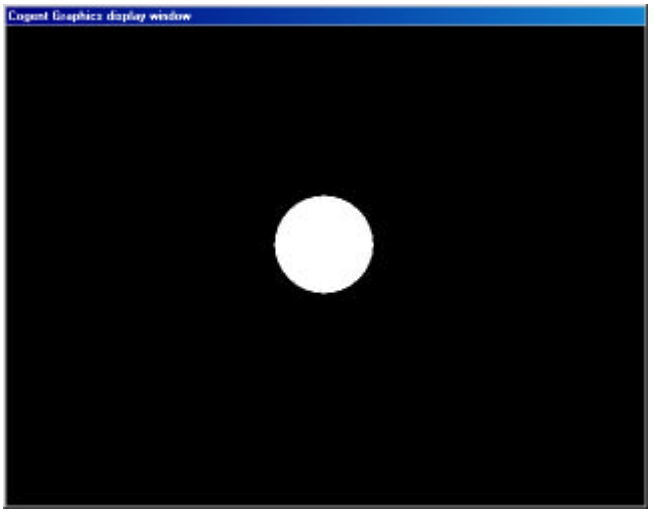
When the script is run you should see the following:-

>> **Mouse**

Move the cursor into the display.

Hit a mouse button to exit

GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 60.14Hz
>>



The white circle should follow the mouse when it is over the display window.

Using the keyboard

*Currently the **cgKeyMap()** function has not been fully integrated with the rest of Cogent. In particular the use of this function may interact undesirably with the Cogent keyboard logging service. Consequently use this function with caution.*

The **cgKeyMap()** function provides an efficient and simple way to read the keyboard. The format of the command is as follows:-

>> **[ks,kp] = cgKeyMap;**

The command returns two arrays; **ks** and **kp**. Both array contain 95 elements which represent the keys on the keyboard. If a particular key ; say key 'n', is currently pressed, then **ks(n)** will equal 1, otherwise it will equal 0. The **kp** arrays indicate those keys that have been pressed (and possibly released) since the previous call to **cgKeyMap**. The key numbers for the keys on the keyboard are given below:-

Key	No.
ESC	1
1!	2
2"	3
3£	4
4$	5
5%	6
6^	7
7&	8
8*	9
9(	10
0)	11
-_	12
=+	13
BK SP	14
TAB	15
Q	16
W	17
E	18
R	19
T	20
Y	21
U	22
I	23
O	24
P	25

Key	No.
[{	26
]}	27
ENTER	28
ENTER£	28
L CTRL	29
R CTRL£	29
A	30
S	31
D	32
F	33
G	34
H	35
J	36
K	37
L	38
;:	39
'@	40
#~	41
L SHIFT	42
|	43
Z	44
X	45
C	46
V	47
B	48

Key	No.
N	49
M	50
,<	51
.>	52
/?	53
GREY/£	53
R SHIFT	54
PRT SCR	55
L ALT	56
R ALT£	56
SPACE	57
CAPS	58
F1	59
F2	60
F3	61
F4	62
F5	63
F6	64
F7	65
F8	66
F9	67
F10	68
F11	87
F12	88
NUM	69

Key	No
SCROLL	70
HOME	71
HOME£	71
UP	72
UP£	72
PGUP	73
PGUP£	73
GREY-	74
LEFT	75
LEFT£	75
CENTRE	76
RIGHT	77
RIGHT£	77
GREY+	78
END	79
END£	79
DOWN	80
DOWN£	80
PGDN	81
PGDN£	81
INS	82
INS£	82
DEL	83
DEL£	83

Keys marked with a following £ symbol are only available on extended keyboards.

This function is best demonstrated in a loop and so an example script named KeyMap.m has been included with the samples which you can download from the website. Here is a listing of KeyMap.m:-

```
function KeyMap
%
% Click in the display window to activate it
% and then press any key to see its keycode.
%
% Hit Esc to exit
%
fprintf('\nClick in the display window to activate it\n');
fprintf('and then press any key to see its keycode.\n\n');
fprintf('Hit Esc to exit\n\n')

cgloadlib
cgopen(1,0,0,0)
kd(1)=0;
while ~kd(1)
   [kd,kp]=cgkeymap;
   kp=find(kp);
   if length(kp)
      fprintf('Key:%d \n',kp)
   end
end
cgshut
return
```

The script starts off with some help information which is also printed out as a matter of course when the script is run. Then a window is opened and the value of **kd(1)** is set to zero. The value of **kd(1)** will reflect the state of the escape key on the keyboard and the statement **while ~kd(1)** sets up a loop which will continue until the escape key is pressed. Inside this loop we obtain the state of the keyboard with the command **`[kd,kp]=cgkeymap;`** and then we find all the keys that have been pressed since the last call using the command **`kp=find(kp);`** which selects all non-zero values in the **kp** array. If there are any keys that have been pressed then the statement **`if length(kp)`** will be true and we print out the number of the key with the **`fprintf('Key:%d \n',kp)`** command. The two **`end`** statements close the **`if length(kp)`** and **`while ~kd(1)`** statements respectively and finally the graphics are closed with the **`cgshut`** command before the function returns. You should get output like this:-

```
>>Keymap

Click in the display window to activate it
and then press any key to see its keycode.

Hit Esc to exit

GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 60.17Hz
Key:36
Key:37
...
...
Key:1
>>
```

Screen dumps

The **cgscrdmp** command has three forms:-

1/ **cgscrdmp**
2/ **cgscrdmp(Filename)**
3/ **cgscrdmp(Defname,Defnumber)**

The first form (on its own, without arguments) creates a screen dump file and saves it with the current default name. The default name has two parts; Defname and Defnumber. Defname is initially set to 'CGSD' (a pseudo-acronym for **C**o**G**ent **S**creen **D**ump) and Defnumber is initially set to 1. The default name is then CGSD00001.BMP. Defnumber is incremented each time the default name is used so you can create a whole series of screen dump files in a simple way. Such a sequence might be used to create a movie using a program such as Adobe Premier. You can reset the values of Defname and Defnumber using form 3/ of this command.

The second form, with the Filename argument creates a screen dump file named Filename.BMP. The current values of Defname and Defnumber are not changed.

The third form, with the Defname and Defnumber arguments, does not create a file but sets the default name components Defname and Defnumber as used in form 1/ described above. You may use this command to put all screen dump files in a subfolder by setting Defname appropriately. For example if you set Defname to 'ScreenDumps/AA' and Defnumber to 1, then files created using form 1/ of the command above will be in subfolder ScreenDumps and will be named AA00001.BMP, AA00002.BMP etc...

Consider the following sequence...

```
>> cgloadlib
>> cgopen(1,0,0,0)
GPrim  v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Feb  6 2002)
Display:640x480x8 59.91Hz
>> mkdir ScreenDump
```

A graphics screen is opened and a new subfolder is created, named 'ScreenDump'.

```
>> cgscrdmp('ScreenDump/aa',1)
```

The default filename and filenumber for screen dumps are set to 'ScreenDump/aa' and 1.

```
>> cgscrdmp
>> cgscrdmp
```

Two files are created in the 'ScreenDump' sub-folder named aa00001.BMP and aa00002.BMP. The command is slow and takes a moment or two to execute.

```
>> cgscrdmp('ExtraSD')
```

A file named ExtraSD.BMP is created in the top directory.

```
>> cgscrdmp
```

Another file is created in the 'ScreenDump' sub-folder named aa00003.BMP.

Animation

I include here a short section on how to program animations. This is a very varied topic and I intend here just to "get you started". Once you have completed this sections you could have a look at some of the sample scripts mentioned in this manual and try to see how they have been done.

If you want to program animated graphics you must think of the graphics display in the same terms as movie film; i.e. an animated sequence is composed of a sequence of still frames. So in order to create movement on the screen you must draw each frame in the sequence, one after the other. This is obviously not a job you can accomplish by typing out successive commands into Matlab using the keyboard and so invariably you must write a Matlab script for your animation. Consider the Matlab script shown below:-

```
 1 function Example
 2
 3 cgloadlib
 4 cgopen(1,0,0,1)
 5
 6 kd(1) = 0;
 7 while kd(1) == 0
 8    kd = cgkeymap;
 9 end
10
11 cgshut
12 return
```

You should create the matlab script shown above using the matlab editor and save it as a file named "Example.m". In general you should name your script files exactly as the function name. Please note that the numbers 1-12 at the left should NOT be included. They simply indicate the line numbers for descriptive purposes.

When you run this script (type "Example" from the Matlab console) the screen will go black because a full-screen Cogent Graphics display has been opened. When you press the escape key the screen will be closed.

Line by line the script does the following:-

1	Defines this script as a Matlab function named "Example".
2	
3	Initializes Cogent Graphics
4	Opens a full-screen graphics display, resolution 640x480 pixels.
5	
6-9	In these lines a "while" loop is executed. On line 8 the keyboard state is read into an array named "kd". When the escape key is pressed the value of kd(1) will change from zero to one and the loop will be broken. Line 6 simply initializes kd(1) to zero.
10	
11	The full-screen display is closed
12	The function has ended and control returns to the Matlab console.

The above example is all very well but it is somewhat lacking in excitement. We can now proceed to add an animated element...

```
 1 function Example
 2
 3 cgloadlib
 4 cgopen(1,0,0,1)
 5
 6 siz = 0;
 7 cgpenwid(10)
 8
 9 kd(1) = 0;
10 while kd(1) == 0
11    kd = cgkeymap;
12
13    cgellipse(0,0,siz,siz,[1 1 1])
14    cgflip(0,0,0)
15
16    siz = siz + 2;
17    if siz > 640
18       siz = 0;
19    end
20
21 end
22
23 cgshut
24 return
```

This time when you run the script you should see a simple animation. You can press the escape key as before to quit the demonstration.

The extra lines that have been added accomplish the following...

6	Sets the initial value of "siz" to zero.
7	Sets the drawing pen width to ten units.
13	Draws a circle of size "siz" in the centre of the screen in full white in the back-buffer.
14	Flips the display to make the latest circle visible and clears the back-buffer to black.
16	Makes "siz" greater by 2.
17-19	If "siz" is greater than 640 (the screen size), reset it to zero.

We now have an "animation loop" in the script (the "while" loop from lines 10 to 18) in which we draw each successive "movie frame" of our animation. Each frame consists of a white circle of variable size. As the animation progresses the circle increases in size until it reaches the edge of the screen and then it starts again at a point. For each frame the size of the circle is defined by the variable "siz" which is modified as necessary during each pass of the loop. This loop carries on ad-infinitum until the user presses the escape key.

This example now shows the general principles of animation using Cogent Graphics:-

1/ Create a Matlab script for your animation.
2/ Do some initial preparation for the animation; initialize some variables, prepare some graphics etc...
3/ Have an "animation loop" in your script.
4/ During each pass of the loop write a new movie "frame" and then make it visible. Make sure that each "frame" differs from the previous "frame" in the appropriate way.
5/ Have some mechanism for ending the loop and returning control back from your script.

Using the photometer

Cogent Graphics can communicate with the departmental photometer, the PhotoResearch PR-650 Spectra-Colorimeter.

The photometer should be connected to a serial port on the PC. The PR-650 comes with a black communication cable which plugs into a circular 12 pin socket on the PR-650 and has a D-type 25 way socket on the other end. This 25 way socket requires an adapter so that you can connect it to the 9-pin serial port plug on your PC. The 25 wat socket also has a toggle switch which has two positions; 'CTRL' and 'XFER'. The switch must always be in the 'CTRL' position.

When you have connected the PR-650 to the PC you should then switch it on by pressing the red '0/I' button on the front of the photometer. The power light should come on on the photometer. You are now ready to start communications.

To open communications with the photometer, use the command below:-

```
>> cgphotometer('open','pr650',1)
```

This command assumes that serial port COM1 is being used. The photometer display should illuminate indicating that communication has been successful.

You can check the photometer identification using the command below:-

```
>> idstr=cgphotometer('id')

idstr =

GScnd:cgPhotometer   v1.24 Compiled:Oct 23 2002
PhotoResearch Spectra-Colorimeter Model PR-650 SN:60954201
```

To make a measurement, use the 'XYZ' command:-

```
>> xyz=cgphotometer('XYZ')

xyz =

3.7690    3.6290    1.1300
```

This returns the CIE (1931) XYZ values for the measurement.

You can then also download the radiant spectrum of the measurement:-

```
>> spc=cgphotometer('SPC')

spc =

  380.0000    0.0000
  384.0000    0.0000
     . . .       . . .
     . . .       . . .
  776.0000    0.0000
  780.0000    0.0000
```

The returned matrix is an (n x 2) array of values. The first column gives the wavelngth of the measurement in nanometres. The second column gives the radiant power in $Wm^{-2}sr^{-1}nm^{-1}$.

Finally you should close down communications with the 'Shut' command and then disconnect and pack away the photometer.

```
>> cgphotometer('shut')
```

A whole set of colorimetry utilities are available which allow you to perform display calibrations and analyze them as well as plot CIE diagrams and radiant spectra. You can download these utilities from the Cogent Graphics website.

Using the eyetracker

The eyetracker must be correctly set up as described in the section "Eyetracker Setup" and you must be familiar with the operation and calibration of the eyetracker before you attempt to use it. Please consult a member of the computer support staff before you use the instrument so that you are happy about using it correctly. There is a sample script available named "Tracker" which is described elsewhere in this manual which gives a realtime demonstration of using the eyetracker.

To open communications with the eyetracker, you can use the command below:-

cgtracker('open',TrackerID,PortNum,Mode,BaudRate<,c1,c2,c3,c4>)

The following arguments must be supplied:-

TrackerID Currently the only supported eyetracker is 'ASL5000'

PortNum You must tell cogent where the serial connection to the ASL5000 comes in. This should be one of the serial ports, usually COM1 or COM2. A number between 1 and 8 is expected here.

Mode The eyetracker can be set up to send eye position only when requested ("On Demand") or in "Continuous" mode. You must let cogent know how the eyetracker is set up. Specify Mode = 1 for "On Demand" or Mode = 0 for "Continuous". Currently "On Demand" is recommended (Mode = 1). There is currently a bug in the ASL software which means that "Continuous" mode does not work properly when you do a subject eye calibration. See "Connecting to your Cogent PC" in the "Eyetracker setup" section for a bit more detail.

BaudRate Here you tell cogent the speed of the serial connection to the ASL5000. This should be the number 57600.

c1,c2,c3,c4 These values are optional. If you don't put them in then the values for eye position that you receive will be in arbitrary units. However, if you put the correct values in here then you will read the eye position in the same co-ordinates that you are using for your screen. You need to go through the calibration procedure to get the numbers that you should use. You get these numbers when you do the "Set Target Points" part of the calibration as described below in the section "Setting the Target Points". This part of the calibration only needs to be done once even if you use different subjects.

So typically you might use the command as shown below:-

>> **cgtracker('open', 'ASL5000',1,1,57600,46,55,219,214)**

When you have finished using the eyetracker it is important that you close the connection using the command below:-

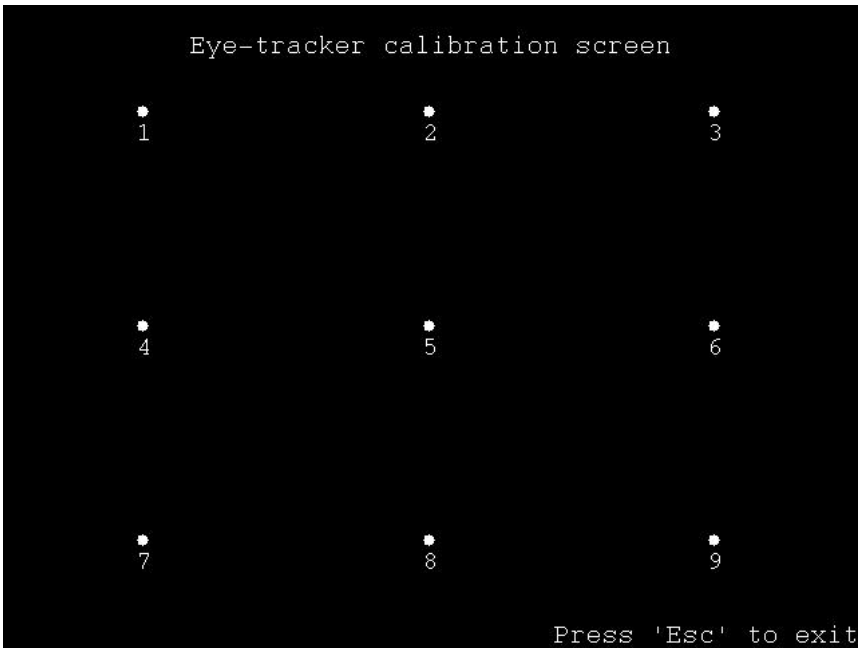
>> **cgtracker('shut')**

Eyetracker calibration screen

In order to use the eyetracker correctly you must calibrate it for your subject. There are two steps to this; first you must let the ASL software know where your calibration points are and secondly you must ask your subject to look at each calibration point in turn and tell the ASL software when they are doing so. There is a command to let you do the calibration. Try the lines below:-

```
>> cgtracker('open','ASL5000',1,1,57600)
>> cgtracker('calibrate')
GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 59.96Hz
```

The screen should appear as shown below:-



This screen can be used to perform both parts of the calibration procedure and it will stay on the display until you press the 'Esc' key. You may use it at any point, even during a Cogent Graphics experiment, and it will appear in the current display and when you exit the display should be restored to its original condition. You may use this feature to recalibrate during an experiment at any time.

You may want to use different colours for the calibration screen which more closely resemble the colours of your stimulus. This is because your subject's pupil diameter will change according to the overall brightness of the scene they are observing. You can specify different colours for the calibration screen in the following way:-

```
>> cgtracker('calibrate',[1 0 0],[0 0 1])
```

Here we specify that the background of the calibration screen should be red **[1 0 0]** and the figures should be blue **[0 0 1]**.

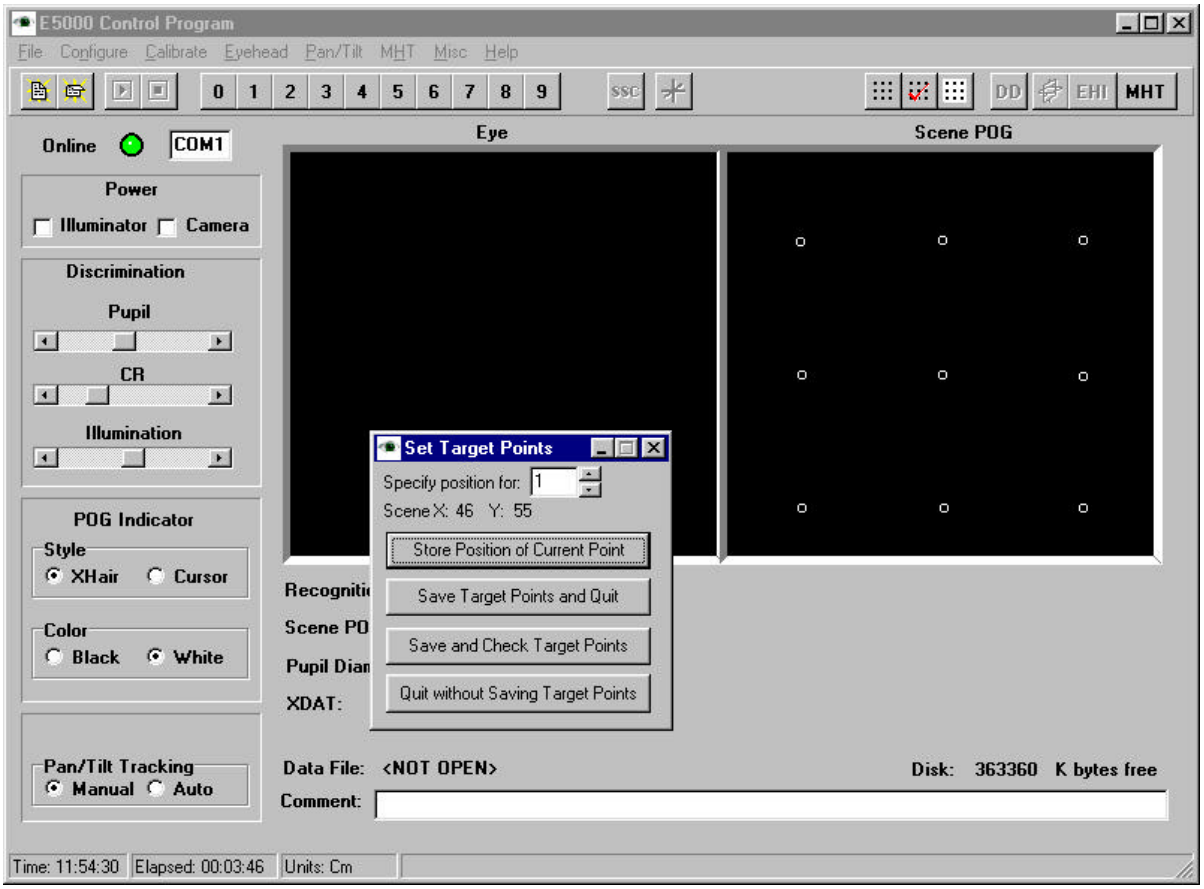
Remember to shut down the eyetracker when you are finished:-

```
>> cgtracker('shut')
```

Setting the Target Points

As mentioned previously there are two parts to the calibration procedure.

The first part is to tell the ASL software where the calibration points are. Generally this only needs to be done once so that the ASL software knows the general configuration of the target points on the cogent calibration screen. You should not be monitoring a subject when you do this. To do this part of the calibration you must run the eyepos software on the eyetracker controller PC and then select "Set Target Points" from the "Calibration" menu:-



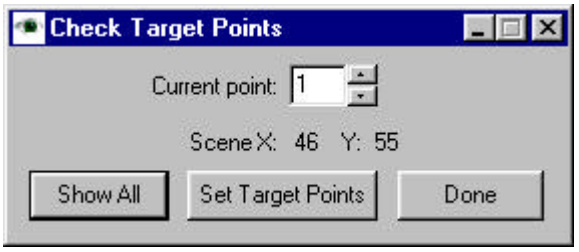
Depending on your setup you should be able to see the cogent calibration screen on the "Scene" monitor of your eyetracker equipment. If you can, then you should move your mouse over the "Scene POG" window so that the crosshairs in the "Scene" monitor lie over each point in turn and then click the left mouse button to store the point and move onto the next one until all nine points have been entered. If you do not have a "Scene" monitor or you cannot see the cogent calibration screen in the "Scene" monitor then you can still enter suitable calibration points "blind". Move the mouse over the "Scene POG" window and watch the "Scene X: Y:" values and enter the following points:-

Target point 1: 40, 40	Target point 2: 130, 40	Target point 3: 220, 40
Target point 4: 40,120	Target point 5: 130,120	Target point 6: 220,120
Target point 7: 40,200	Target point 8: 130,200	Target point 9: 220,200

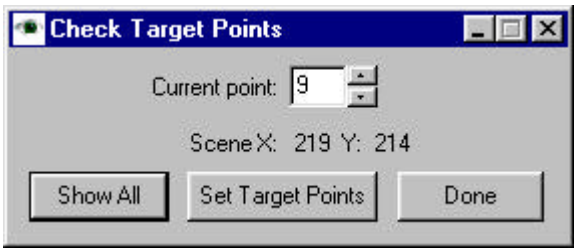
At any rate, this part of the calibration gives you the **c1,c2,c3,c4** calibration co-ordinates that you need to enter into the "open" command to receive the eye position in screen co-ordinates. The values for **c1,c2,c3,c4** are the co-ordinates for target points 1 and 9 in sequence:-

Target point 1: **c1,c2** Target point 9:**c3,c4**

You can check what the co-ordinates are for points 1 and 9 by using the "Check Target Points" item from the "Calibration" menu of the eyepos program:-



Here you can see that the co-ordinates for point 1 are (46,55) i.e. **c1** = 46 and **c2** = 55. You can get the values for **c3** and **c4** in a similar way from the co-ordinates for point 9:-



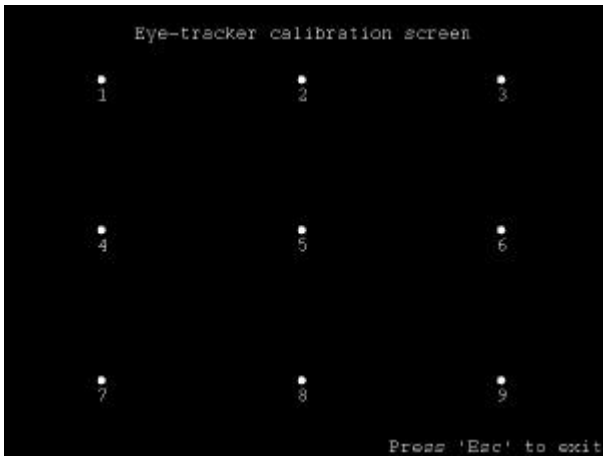
Here we get **c3** = 219 and **c4** = 214. So the "open" command we would use in this case would be:-

>> **cgtracker('open', 'ASL5000',1,1,57600,46,55,219,214)**

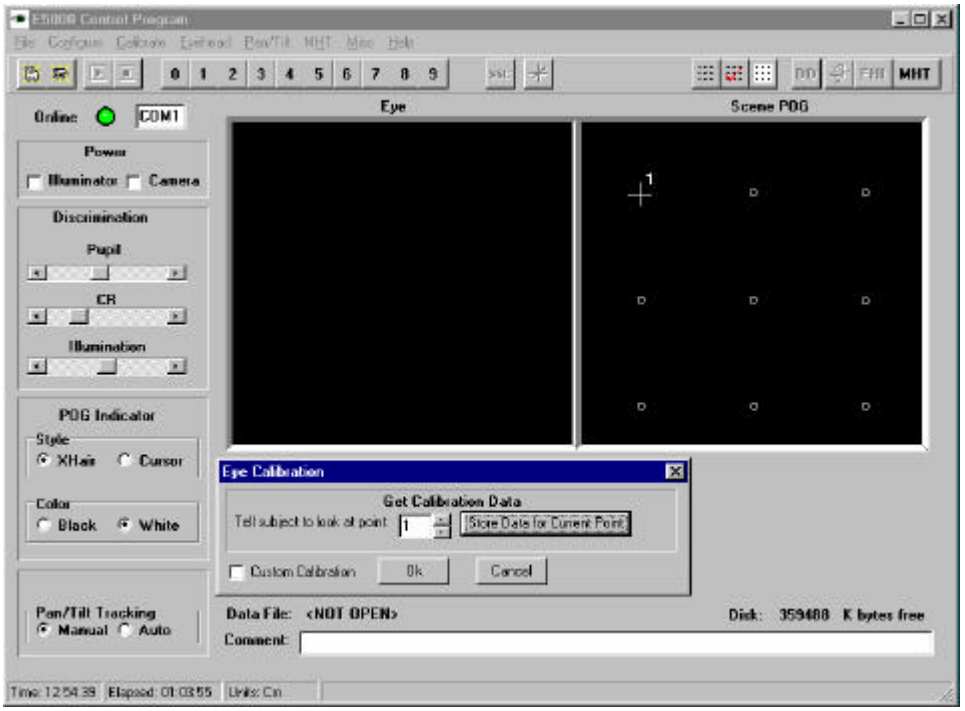
Calibrating each subject

You only really need to "Set Target Points" once; the values you set up should be fine for all subsequent cogent experiments. However, you also need to carry out an individual calibration for each subject or before each experiment. You may even feel that you want to recalibrate during an experiment. This should not be a problem because this calibration can usually be carried out in less than a minute.

To do this part of the calibration you must have your subject looking at the cogent display exactly as if he were doing your experiment. You may find it useful to use some kind of head support or chin rest although this is not absolutely necessary. You must run the ASL eyetracker software and adjust it correctly for your subject so that the pupil and corneal reflection is being correctly discriminated. Then display the cogent calibration screen:-



On the ASL control PC, select "Eye Calibration…" from the "Calibrate" menu:-



You should then ask your subject to look at each of the points in turn on the cogent display and when they look at each one click on the "Store Data for Current Point" so that all nine points are stored. That completes the calibration.

Eyetracker eye data

You can obtain eye data from the eyetracker using the "eyedat" command:-

```
>> cgtracker('open', 'ASL5000',1,1,57600,46,55,219,214)
>> eyedat = cgtracker('eyedat')

eyedat =

           X: 280
           Y: 176
   Timestamp: 17.6532
       Pupil: 7
      Status: 0

>> cgtracker('shut')
```

The "eyedat" structure contains the following elements:-

X,Y	The X and Y co-ordinates of the subject's point of gaze. If the calibration co-ordinates were supplied in the **cgtracker('open'…)** call then the co-ordinates will be in cogent screen co-ordinates, otherwise they will be in arbitrary units.
Timestamp	This gives the exact time when the eyedata was received. The eyetracker system calculates a new position for each frame taken by the eye camera and so if the camera is operating at 50Hz there will be a new eye position every 20mS.
Pupil	This is the pupil diameter as calculated by the eyetracker. A value of zero means that the pupil was not discriminated; i.e. the subject had turned away or blinked.
Status	This value is returned by the eyetracker. A value of zero means normal operation. Other values may be used to indicate special conditions. Consult the ASL manual for details.

Timing and synchronization may be critical in studies involving eyeposition and so an internal check is made on the timestamp for each eyedat structure. If the eye data is more than 50mS old then a warning message will be generated as shown below:-

```
>> cgtracker('open','ASL5000',1,50,57600,46,55,214,219)
>> eyedat=cgtracker('eyedat');
WRN cgTracker:EyeDat Eyetracker data is 0.065 seconds old
>> cgtracker('shut')
```

If no serial data has been received you will get an error message:-

```
>> cgtracker('open','ASL5000',1,50,57600,46,55,214,219)
>> eyedat=cgtracker('eyedat');
ERR cgTracker:EyeDat No serial connection
>> cgtracker('shut')
```

You can also record eye position data in the background for a period of time and then retrieve the information later.

To start recording, use the command:-

>> **cgtracker('start')**

Then, at the end of the period you can retireve the data using the following command:-

>> **eyearray = cgtracker('stop')**

'**eyearray**' will then contain an array of "eyedat" structures, each one corresponding to a separate eye position. You may store up to twenty minutes of data if the eye camera is running at 60Hz, 10 minutes at 120Hz or 5 minutes at 240Hz.

Matlab scripts

Sample scripts

Matlab script files have a ".m" file extension. Matlab scripts are text files containing matlab commands which can be run as matlab programs. The Cogent Graphics library is a set of additional matlab commands and these can therefore be used in matlab scripts. You may download a set of sample scripts from the Cogent Graphics website. When you decompress this file you should have a new folder named "Samples" which contains the matlab sample scripts. If you want to run one of these scripts you should run matlab and then change directory to your "Samples" folder.

The sample programs were modified in v1.17 to take into account the bug-fix of Cogent Graphics timing and so it is worthwhile to make sure you have downloaded the latest version of the sample programs from the website.

Animation examples

There are several animation examples which have direct mode and palette mode variants.
Some examples give accurate timing information along the top of the screen indicating the total time elapsed, the number of frames drawn, the refresh rate and the number of dropped frames. These examples are therefore of use to check that your version of Cogent is working properly on your machine.

The timing statistics are shown at the top of the screen:-

```
ProgName vN.NN P:NORMAL Tim:00:00:05 Frm:199 Av:75.12Hz Drp:0
```

The components are as follows:-

`ProgName vN.NN`	Program name and version number
`P:NORMAL`	Current priority class
`Tim:00:00:05`	Time elapsed in hours minutes and seconds since script was started
`Frm:199`	Number of frames since script was started
`Av:75.12Hz`	Average frame rate in Hertz
`Drp:0`	The number of frames dropped since script was started

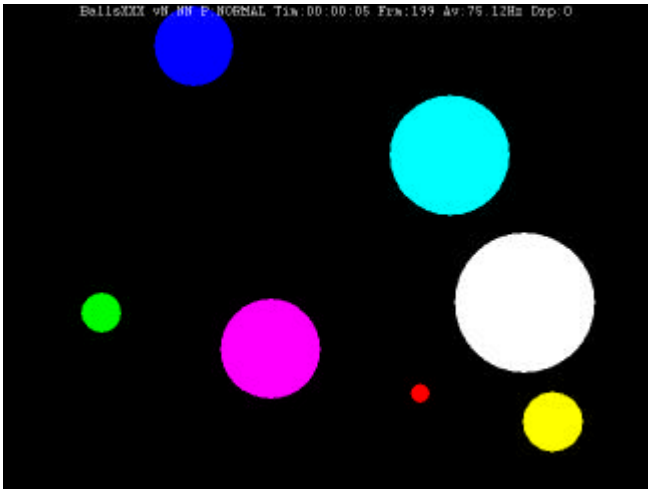
The sample scripts time each frame and calculates the average frame rate in hertz. This gives an average frame time. Each frame time is compared to the average and if it differs by more than 2 mS (0.002 seconds) it is counted as a "Dropped Frame".

You should read the section on "Display Timing" in this manual if you are concerned about timing for your experiment.

The following examples can be viewed in the matlab editor to see how the script achieves the animations. It is also possible to pass arguments to each of the example scripts to control display parameters but a detailed description of these examples will not be presented here. Hit the escape key to terminate most of these scripts.

Balls

The first example, “ballsrgb”/”ballspal” draws seven moving coloured circles on the display:-

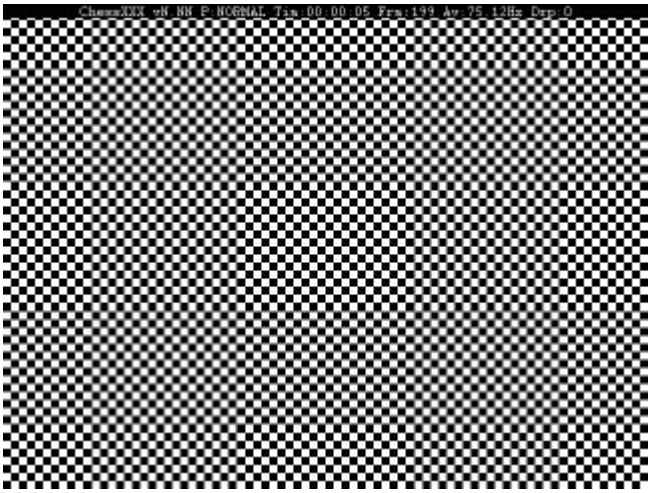


Scroll

The second example, “scrollrgb”/”scrollpal” draws a smoothly scrolling random grid on the display:-

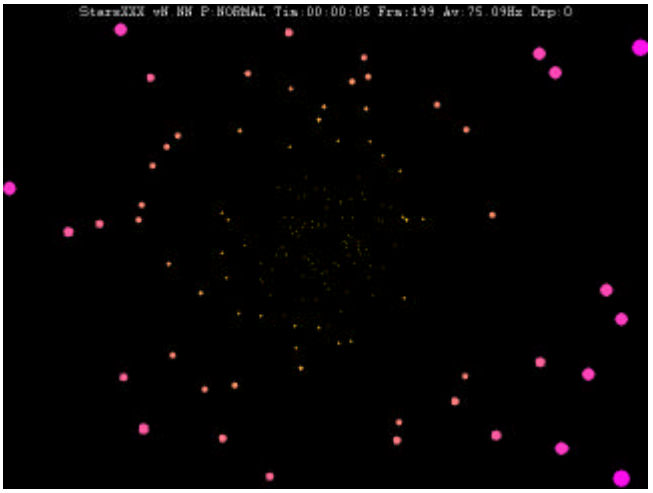
Chess

The third example, “chessrgb”/”chesspal” draws a reversing black and white chessboard on the display:-



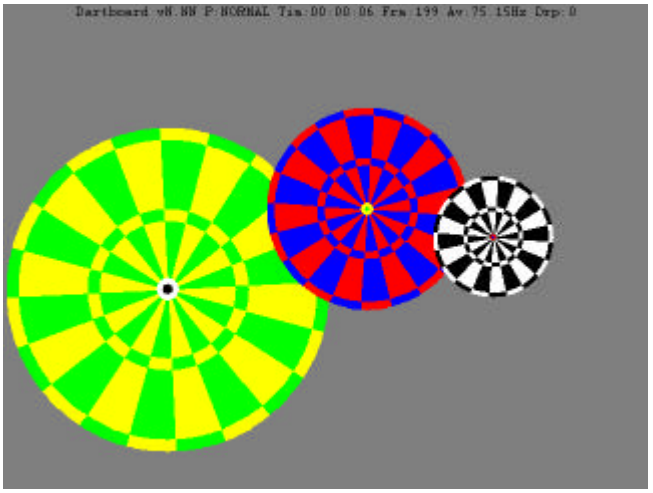
Stars

This script “starsrgb/starspal” draws an animated starfield with the colour of the objects changing from yellow in the centre to magenta at the edges:-



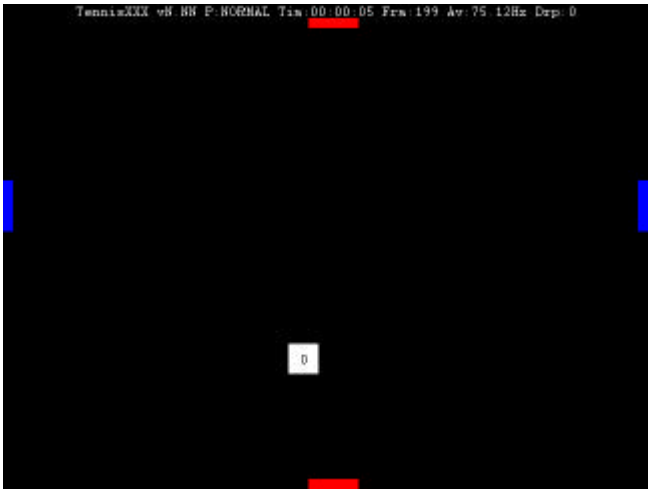
Dartboard

Another sample script, named “Dartboard”, uses palette animation to draw three spinning animated dartboards on the screen:-



Tennis

This script “tennisrgb/tennispal”plays a simple tennis game, controlled by the mouse:-

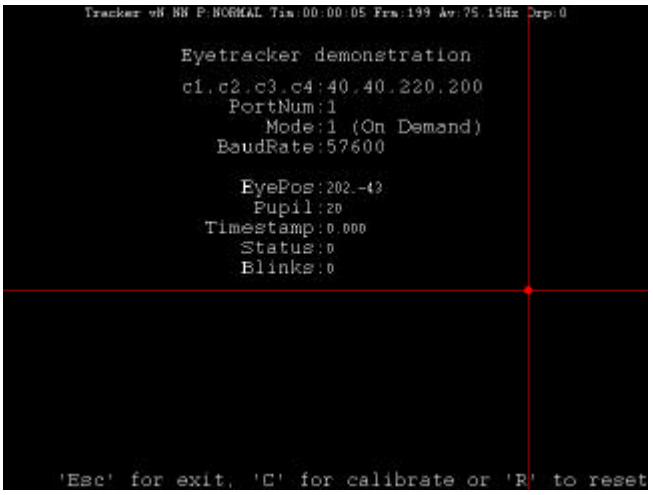


Equipment examples

This next script demonstrates the use of the eyetracker.

Tracker

This scripts gives a real-time demonstration of the eyetracker:-



The screen shows the setup values for the eyetracker (c1,c2,c3,c4, PortNum, Mode and BaudRate) at the top. Next comes the latest eye data from the eyetracker (EyePos, Pupil, Timestamp and Status). Underneath that comes a count of the number of blinks detected.

If there is valid eyedata (Pupil > 0), crosshairs and a spot in red indicate the point of gaze on the screen. Otherwise "No Eye Data" appears in red under the blink count.

You can press the 'R' key to reset the blinks count and timestamp values to zero or the 'C' key to display the eyetracker calibration screen. The 'Esc' key exits the script.

The script takes the following arguments:-

Tracker(c1,c2,c3,c4,PortNum,Mode,BaudRate)

c1,c2,c3,c4	= Calibration co-ordinates c3 > c1, c4 > c2. Default values are 40,40,220,200
PortNum	= Serial port (COM port) number 1 to 8. Default value is 1
Mode	= Eyetracker mode 1=On Demand, 0=Continuous. Default is 1 (On Demand)
BaudRate	= Serial port baud rate 9600/19200/38400/57600. Default value is 57600

You may use any of the forms of the command shown below. Missing values assume the default setting:-

Tracker
Tracker(c1,c2,c3,c4)
Tracker(c1,c2,c3,c4,PortNum)
Tracker(c1,c2,c3,c4,PortNum,Mode)
Tracker(c1,c2,c3,c4,PortNum,Mode,BaudRate)

Examples from this manual

There are some examples from this manual which have been included with the samples. These demonstrate aspects of graphics which use loops and are not suitable for interactive command line usage.

Mouse

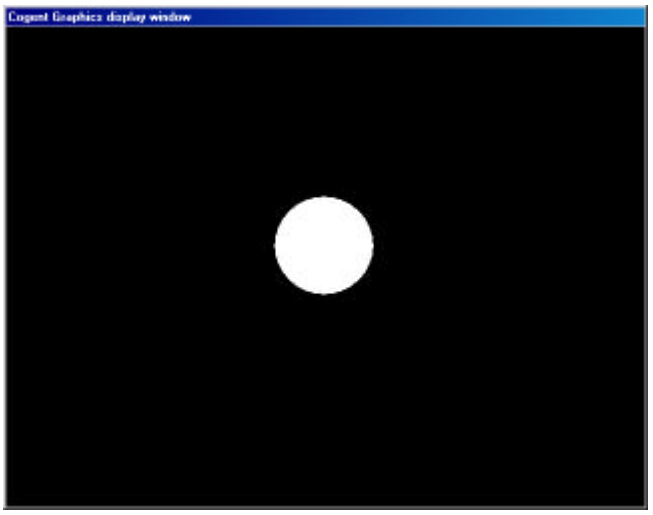
This script puts up a black display screen with a white circle that follows the mouse position. Hit a mouse button to exit:-

>> **Mouse**

Move the cursor into the display.

Hit a mouse button to exit

GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 60.14Hz
>>



KeyMap

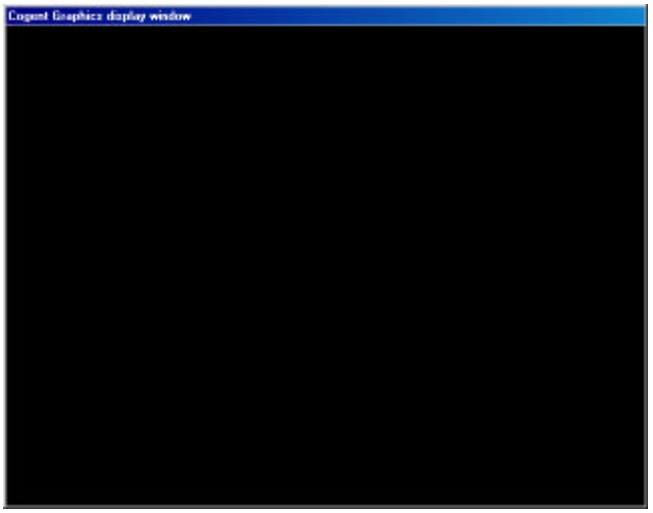
This script puts up a black display screen. Click on the screen with the mouse to select it and then use the keyboard. The key number of each button you press will be printed on the matlab console screen. Hit the Esc key to exit.

>>**Keymap**

Click in the display window to activate it
and then press any key to see its keycode.

Hit Esc to exit

GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Display:640x480x32 60.17Hz
Key:36
Key:37
. . .
. . .
Key:1
>>

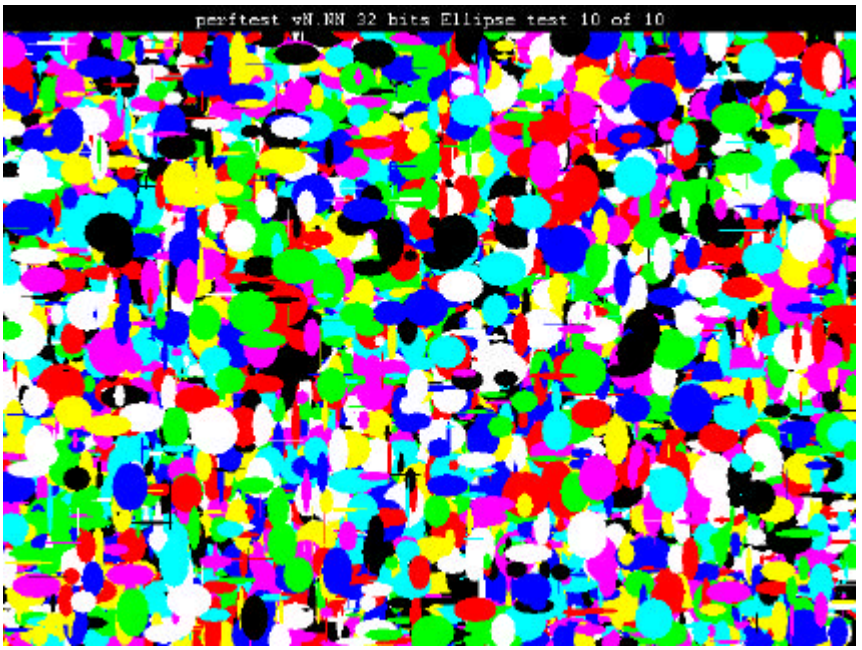


Utilities

Some useful utility scripts are also included with the samples.

PerfTest

Perftest runs a Cogent Graphics performance test, displays the results on screen and also saves them to a file named 'perfdata.txt'. Drawing of dots, lines, rectangles and ellipses are tested for 8, 16, 24 and 32 bit graphics. The test should take less than five minutes:-



During testing some messages may appear on-screen as graphics screens are opened. You may also see the line "Number of array elements exceeds 16384" which is produced while testing whether Student Matlab is being used. These messages can be ignored. The first proper lines of output give the time of the test and identify the test program and machine ID of the PC as shown below:-

```
perftest v1.01 10-Sep-2001
Machine:mustard00 128.40.206.2
GPrim  v1.24 Compiled:Oct 23 2002
GLib DirectDraw v7 Compiled:Feb  6 2002
Display 640x480 85.40Hz
```

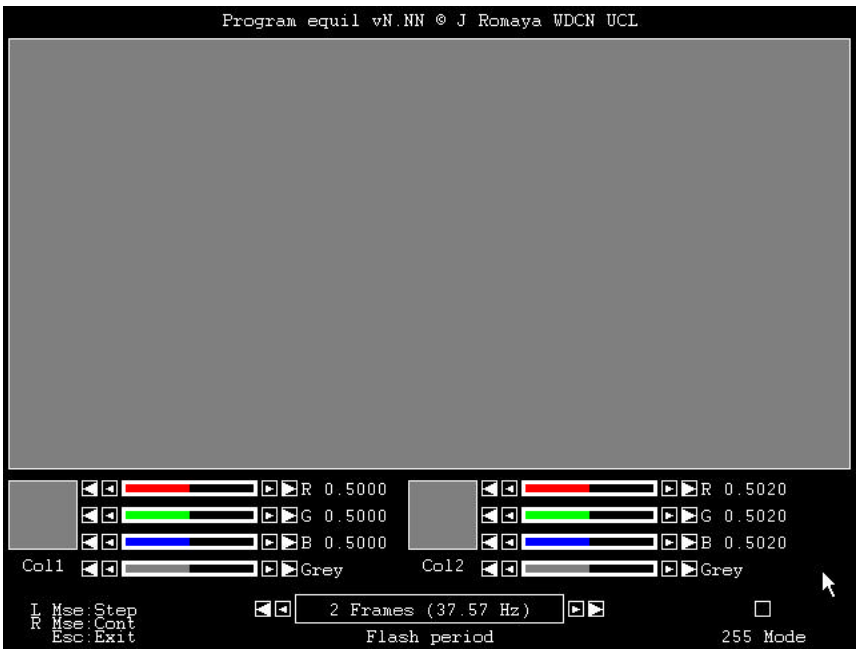
Then for each of the graphics modes (8, 16, 24 and 32 bit) you get lines as follows:-

```
 Cogent Graphics perfomance test (32 bit graphics)
    Dots per second:126448  StdErr:87
   Lines per second:32447  StdErr:31
   Rects per second:60442  StdErr:999
Ellipses per second:9941   StdErr:3
```

This gives the number of graphics objects drawn per second for dots, lines, rectangles and ellipses as well as a measure of the standard error in each measurement. The variation in these measurements is due to disk activity, virtual memory operations and task swapping which are part of the normal running of the operating system.

Equil

This script can be used to determine equiluminance points for different colours using a flicker method.



The large upper rectangular region flashes alternately between two colours named Colour 1 and Colour 2.

Underneath there are controls for colour 1 on the left and for colour 2 on the right. There is a square showing the colour and then four slider controls for red, green, blue and grey respectively. Move the mouse into the triangle boxes and then press a mouse button to increase or decrease a colour level. Use the left mouse button to change a step per click or the right button for continuous change. The big triangle changes the value in steps of 10 and the small triangle in steps of 1. Colour levels are displayed as a number from 0 to 1 by default but if you select '255 Mode' they are displayed as numbers from 0 to 255.

You may also change the flash period in multiples of 2 frames. Only even periods are possible because the program displays colour 1 for 'n' frames and then colour 2 for 'n' frames giving a period of '2n'.

You should aim to reduce the flicker to a minimum. Sometimes it helps to desaturate the colour by adding grey as it can be hard to find the equiluminance point with a highly saturated colour.

To stop the script press the Esc key

Function specifications

cgalign(XAlign,YAlign)

XAlign	'l', 'c' or 'r' for left, centre or right alignment respectively.
YAlign	't', 'c' or 'b' for top, centre or bottom alignment respectively.

This function sets the alignment mode for the cgtext, cgrect and cgdrawsprite commands:-

If left alignment is chosen the text, rectangle or sprite will have its left edge aligned with the x co-ordinate.

If centre alignment is chosen the text, rectangle or sprite will be centred horizontally on the x co-ordinate.

If right alignment is chosen the text, rectangle or sprite will have its right edge aligned with the x c-ordinate.

In a similar way the YAlign argument controls the vertical placement of the sprite with respect to the y co-ordinate.

cgalign('l','t')	Selects left and top alignment for **cgtext**, **cgrect** and **cgdrawsprite** commands

cgarc(cx,cy,w,h,a1,a2,<,Col><,ArcType>)

cx,cy	Co-ordinates of the centre of the ellipses for which the arcs are drawn.
w,h	Widths and heights of the ellipses for which the arcs are drawn.
a1,a2	Start and finish angles for the arcs. Angles are measured in degrees, increasing anticlockwise. Zero is a horizontal line going to the right from the ellipse centre.
Col	(Optional) The colours for the arcs. When in palette mode, Col must be an (n x 1), (single column), array containing the palette index for each item. When in direct mode, Col must be an (n x 3) array containing the RGB levels (0 to 1) for each item.
ArcType	(Optional) 'A' (default) draws hollow arcs, 'S' draws filled sectors.

This function draws hollow arcs or filled sectors, with centres on the co-ordinates (x,y) and widths and heights defined by w and h and start and finish angles defined by a1 and a2.

The arcs are drawn on the current destination as set by **cgsetsprite**.

If you include the Col argument, each arc or sector will be drawn in the requested colour, otherwise the current drawing colour (as set by **cgpencol**) will be used.

If the optional ArcType argument is 'S', filled sectors will be drawn. Otherwise, or if ArcType is 'A', hollow arcs are drawn using the current line width as set by **cgpenwid**.

The arrays (cx, cy, w, h and Col) must have the same number of entries. When in direct mode where Col is an (n x 3) array, 'n' gives the number of entries.

cgarc(0,0,20,10,45,90)

Draws a hollow arc of an ellipse centred on (0,0) with width 20 and height 10. The start and finish angles of the arc are 45 and 90 degrees respectively.

cgarc(0,0,10,10,90,135,'S')

Draws a filled sector of an ellipse of radius 10, centred on (0,0). The start and finish angles of the sector are 90 and 135 degrees respectively.

cgarc([100 100],[-100 100],[100 50],[50 100],[50 70],[100 200],[1 0 0;0 0 1],'S')

Draws two filled sectors in direct colour mode. One is drawn for an ellipse with centre (100,-100) with width 100 and height 50 in colour 1,0,0 (red). The other is drawn for an ellipse with centre (100,100) with width 50 and height 100 in colour 0,0,1 (blue). Start and finish angles for the two sectors are 50,100 and 70,200 respectively.

cgarc([100 100],[-100 100],[100 50],[50 100],[50 70],[100 200],[1;2])

Draws two hollow arcs in palette mode. One is drawn for an ellipse with centre (100,-100) with width 100 and height 50 in palette index 1. The other is drawn for an ellipse with centre (100,100) with width 50 and height 100 in palette index 2.). Start and finish angles for the two arcs are 50,100 and 70,200 respectively.

cgblitsprite(Key,srcx,srcy,srcw,srch,dstx,dsty<,dstw,dsth>)	
Key	Identification number of the sprite to draw.
srcx,srcy	Co-ordinates defining the position of the rectangle on the source sprite.
srcw,srch	The width and height of the rectangle on the source sprite
dstx,dsty	Co-ordinates defining the position of the rectangle on the destination.
dstw,dsth	You may optionally define the width and height of the rectangle on the destination if it differs from the source sprite dimensions.

This function copies an arbitrary rectangle from a sprite to the current destination as set by **cgsetsprite**. The destination may be scaled to the specified width and height if required. The sprite placement with respect to (x,y) depends on the current alignment mode as set by **cgalign**.

cgblitsprite(1,0,0,60,70,50,80)	Copy a rectangle 60 units wide by 70 units high from position (0,0) in sprite 1 to the co-ordinates (50,80) on the current destination.
cgblitsprite(2,0,0,60,70,50,80,120,140)	Copy a rectangle 60 units wide by 70 units high from position (0,0) in sprite 2 to the co-ordinates (50,80) on the current destination and double the width and height to 120 units wide by 140 units high.

cgcoltab(PI,R,G,B) or cgcoltab(PI,RGB)

PI	Palette index for first colour. This argument can take values from 0 to 255.
R,G,B	These arrays contain the colour of each colour to be loaded into the colour table. The R,G,B arrays contain the red, green and blue components respectively. The R,G,B components take values from 0 to 1. The R,G and B arrays must each contain the same number of elements and the number of elements taken together with the first palette index must not exceed the palette size (normally 256).
RGB	This (n x 3) array contains the same information as the separate R, G and B arguments above.

This command loads colours into the colour table. The colours are not displayed until a call is made to the **cgnewpal** command.

cgcoltab(1,1,0,0)	Sets colour 1 in the palette to maximum red
cgcoltab(2,[0 1 0])	Sets colour 2 in the palette to maximum green

cgdraw(x,y<,x2,y2><,Col>)

x,y	Co-ordinates of the points to draw, or, if (x2,y2) are present, the co-ordinates of the start of the lines to draw.
x2,y2	(Optional) Co-ordinates of the ends of the lines to be drawn.
Col	(Optional) The colour for each point or line. When in palette mode, Col must be an (n x 1), (single column), array containing the palette index for each item. When in direct mode, Col must be an (n x 3) array containing the RGB levels (0 to 1) for each item.

This function draws points at the co-ordinates (x,y), or, if (x2,y2) are present, draws lines from (x,y) to (x2,y2).

Lines are drawn with a width as set by **cgpenwid**.

The points or lines are drawn into the destination currently selected by **cgsetsprite**.

If you include the Col argument, each point or line will be drawn in the requested colour, otherwise the current drawing colour (as set by **cgpencol**) will be used.

All arrays (x, y, x2, y2 and Col) must have the same number of entries. When in direct mode where Col is an (n x 3) array, 'n' gives the number of entries.

cgdraw(0,0)	Draws a point at co-ordinates (0,0)
cgdraw(0,0,10,10)	Draws a line from (0,0) to (10,10)

cgdraw([100 100],[-100 100],[1;2])

Draws two points in palette mode. The first point is drawn at (100,-100) in palette index 1 and the second point is drawn at (100,100) in palette index 2.

cgdraw([50 100],[-100 50],[150 100],[-100 150],[1 0 0;0 0 1])

Draws two lines in direct colour mode. The first line is drawn from (50,-100) to (150,-100) in colour 1,0,0 (red). The second line is drawn from (100,50) to (100,150) in colour 0,0,1 (blue).

cgdrawsprite(Key,x,y<,w,h>)	
Key	Identification number of the sprite to draw.
x,y	Co-ordinates defining the target position for the sprite.
w,h	You may optionally define a width and height for the destination if you want to scale the sprite.

This function draws the selected sprite at the co-ordinates (x,y) on the current destination as set by **cgsetsprite**. The destination may be scaled to the specified width and height if required. The sprite placement with respect to (x,y) depends on the current alignment mode as set by **cgalign**.

cgdrawsprite(1,0,0)	Draws sprite number 1 at co-ordinates (0,0) on the current destination.
cgdrawsprite(2,0,0,10,10)	Draws sprite number 2 at co-ordinates (0,0) on the current destination and scales it to fit in a box 10 units square.

cgellipse(cx,cy,w,h<,Col><,'f'>)

cx,cy	Co-ordinates of the centre of the ellipses.
w,h	Widths and heights of the ellipses.
Col	(Optional) The colours for the ellipses. When in palette mode, Col must be an (n x 1), (single column), array containing the palette index for each item. When in direct mode, Col must be an (n x 3) array containing the RGB levels (0 to 1) for each item.
'f'	(Optional) draws filled rather than hollow ellipses.

This function draws ellipses, with centres on the co-ordinates (x,y) and widths and heights defined by w and h.

The ellipses are drawn on the current destination as set by **cgsetsprite**.

If you include the Col argument, each ellipse will be drawn in the requested colour, otherwise the current drawing colour (as set by **cgpencol**) will be used.

If the optional 'f' argument is used filled ellipses will be drawn. Otherwise hollow ellipses are drawn using the current line width as set by **cgpenwid**.

The arrays (cx, cy, w, h and Col) must have the same number of entries. When in direct mode where Col is an (n x 3) array, 'n' gives the number of entries.

cgellipse(0,0,20,10)

Draws a hollow ellipse centred on (0,0) with width 20 and height 10.

cgellipse(0,0,10,10,'f')

Draws a filled circle of radius 10, centred on (0,0).

cgellipse([100 100],[-100 100],[100 50],[50 100],[1 0 0;0 0 1],'f')

Draws two filled ellipses in direct colour mode. One is drawn with centre (100,-100) with width 100 and height 50 in colour 1,0,0 (red). The other is drawn with centre (100,100) with width 50 and height 100 in colour 0,0,1 (blue).

cgellipse([100 100],[-100 100],[100 50],[50 100],[1;2])

Draws two hollow ellipses in palette mode. One is drawn with centre (100,-100) with width 100 and height 50 in palette index 1. The other is drawn with centre (100,100) with width 50 and height 100 in palette index 2.

<S =>cgflip<(VBLFlag)> or <S =>cgflip<(R,G,B)> (Direct colour mode) or <S =>cgflip<(PI<,ImFlag>)> (Palette mode)	
S	(Optional) Timestamp in seconds for the page-flip. It is not possible to obtain a timestamp in immediate mode.
VBLFlag	(Optional) If present, no pageflip will occur but the function will wait until the next vertical blanking period. This argument must be 'v' or 'V'. Used for animation timing.
R,G,B	(Optional) If present, the offscreen buffer will be cleared to this colour. R,G and B represent the red, green and blue components of the colour respectively and take values from 0 to 1. This argument is only valid in direct colour mode.
PI	(Optional) If present, the offscreen buffer will be cleared to this palette index. PI represents the palette index and takes values from 0 to 255. This argument is only valid in palette mode.
ImFlag	(Optional) If present, the pageflip will occur immediately rather than waiting for the next vertical blanking period. This is appropriate when the flip immediately follows a **cgnewpal** command which automatically synchronises with the VBL anyway. This argument is only valid in palette mode. This argument can only be 'i' or 'I'. If your hardware does not support immediate flip mode the command will return an error.

This command can be used to copy the offscreen buffer to the screen, making it visible. However, the VBLFlag mode does not update the screen but simply waits until the next display frame.

You may optionally receive a timestamp indicating the precise time that the offscreen buffer became visible using the S variable. This timestamp gives you a time in seconds since you made your initialising call to **cgopen**. Currently the precision of the timestamp is 0.001S (= 1mS). The returned S value is always –1 in immediate mode. A value of –2 indicates that an error has occurred. Prior to v1.17 this function returned a timestamp in microseconds but there was a bug with this value which became negative after 35 minutes. This bug has now been fixed and the function returns its timestamp in units of seconds.

If you supply the optional R,G,B values the offscreen area will be cleared after the pageflip to that colour. If you are in palette mode you can use the PI value to clear the offscreen area to that palette index after the pageflip. In palette mode you can also specify the immediate mode for the flip if it immediately follows a **cgnewpal**.

cgflip	Makes the offscreen buffer visible.
S=cgflip	Makes the offscreen buffer visible and records the time it became visible in "S".
S=cgflip('V')	Waits for the next display frame and records the time in "S".
cgflip('I')	Makes the offscreen buffer visible immediately without waiting for the vertical blanking interval. This form is only available in palette mode.
S=cgflip(.5,.5,.5)	Makes the offscreen buffer visible, records the time it became visible in "S" and then clears the offscreen buffer to colour RGB=0.5, 0.5, 0.5 (mid-grey). This form is only available in direct colour mode.
S=cgflip(10)	Makes the offscreen buffer visible, records the time it became visible in "S" and then clears the offscreen buffer to palette index=10. This form is only available in palette mode.

cgfont(Fontname,Fontheight)

Fontname	The name of the font you want to use.
Fontheight	The height of the font you want to use.
This function loads the specified font. It will be used in subsequent calls to **cgtext**.	
cgfont('Arial',20)	Loads up the font named 'Arial' at a size of 20 ready for **cgtext**.

cgfreesprite(Key)

Key	The name of the font you want to use.
This function deletes the specified sprite.	
cgfreesprite(2)	Deletes sprite number 2.

dat = cggetdata('DataType',Selector)

or specifically:-

csd = cggetdata('CSD') or
gpd = cggetdata('GPD') or
ras = cggetdata('RAS',RASKey) or
dib = cggetdata('DIB',DIBKey) or
gsd = cggetdata('GSD') or
spr = cggetdata('SPR',SPRKey) or
mvd = cggetdaa('MVD',MOVKey) or
mve = cggetdata('MVE',MVEKey)

dat	(returned) - A matlab structure containing the data requested
DataType	A character string requesting a particular type of data;- 'CSD' requests the CogStdData structure 'GPD' requests the GPrimData structure 'RAS' requests a RAS structure 'DIB' requests a DIB structure 'GSD' requests the GScndData structure 'SPR' requests a Sprite structure 'MVD' requests a MOVData structure 'MVE' requests a Movie structure
Selector	If a RAS, DIB, Sprite, MOVData or Movie structure is requested this value contains the Key identifier for the specific structure required.

This function returns data from the Cogent Graphics libraries in the form of a matlab structure. The structure elements are described in the Data Values chapter of this manual.

csd = cggetdata('CSD')	Obtains the CogStdData structure.
gpd = cggetdata('GPD')	Obtains the GPrimData structure.
ras = cggetdata('RAS',3)	Obtains the RAS structure for raster number 3.
dib = cggetdata('DIB',7)	Obtains the DIB structure for DIB number 7.
gsd = cggetdata('GSD')	Obtains the GScndData structure.
spr = cggetdata('SPR',2)	Obtains the Sprite structure for sprite number 2.
mvd = cggetdata('MVD',8)	Obtains the MOVData structure for MOV number 8.
mve = cggetdata('MVE',9)	Obtains the Movie structue for movie number 9.

<[ks<,kp>]=>cgKeyMap()	
ks	Returned array of 95 elements, representing the current state of keys on the keyboard. Each element can be 1 indicating the key is pressed or 0 indicating it is not pressed.
kp	Returned array of 95 elements, showing the keys that have been pressed (and indicating the key has been pressed or 0 indicating it has not pressed.
This function reads the keyboard state.	
cgKeyMap	This call on its own simply clears the state of the key pressed (kp) data. All keys are reset to not having been pressed
ks = cgKeyMap	Return the state of keyboard keys in the array ks
[ks,kp] = cgKeyMap	Return the state of keyboard keys in the array ks and a record of all keys that have been pressed since the previous call to cgKeyMap in array kp.

Key codes are as follows:-

Key	No.
ESC	1
1!	2
2”	3
3£	4
4$	5
5%	6
6^	7
7&	8
8*	9
9(	10
0)	11
-_	12
=+	13
BK SP	14
TAB	15
Q	16
W	17
E	18
R	19
T	20
Y	21
U	22
I	23
O	24
P	25

Key	No.
[{	26
]}	27
ENTER	28
ENTER£	28
L CTRL	29
R CTRL£	29
A	30
S	31
D	32
F	33
G	34
H	35
J	36
K	37
L	38
;:	39
‘@	40
#~	41
L SHIFT	42
|	43
Z	44
X	45
C	46
V	47
B	48

Key	No.
N	49
M	50
,<	51
.>	52
/?	53
GREY/£	53
R SHIFT	54
PRT SCR	55
L ALT	56
R ALT£	56
SPACE	57
CAPS	58
F1	59
F2	60
F3	61
F4	62
F5	63
F6	64
F7	65
F8	66
F9	67
F10	68
F11	87
F12	88
NUM	69

Key	No
SCROLL	70
HOME	71
HOME£	71
UP	72
UP£	72
PGUP	73
PGUP£	73
GREY-	74
LEFT	75
LEFT£	75
CENTRE	76
RIGHT	77
RIGHT£	77
GREY+	78
END	79
END£	79
DOWN	80
DOWN£	80
PGDN	81
PGDN£	81
INS	82
INS£	82
DEL	83
DEL£	83

Keys marked with a following £ symbol are only available on extended keyboards.

<RASKey=>cgloadarray(Key,aw,ah,PixVal<,PalRGB<,StartIndex>><,sw,sh>)	
RASKey	(Optional) This gives the ID number of the raster that has been created for this sprite. The raster is created by the underlying GPrim library and should not be required by most users of the GScnd suite.
Key	The identification number to use for this sprite.
aw,ah	The width and height of the image array.
PixVal	This array defines the pixel value for the image for each pixel in turn starting at the top left of the image and then moving across rightwards and then downwards. An rgb image is defined by an (nx3) array with values ranging from 0 to 1. A palette image requires a (1xn) array containing palette indices from 0 to 255 in which case the PalRGB argument is also required.
PalRGB	This (mx3) array (m = 1 – 256) defines the palette colours for the array. The red green and blue components of each palette entry take values from 0 to 1.
StartIndex	When the display is in palette mode the array must be of palette type and also this variable must be included to indicate where in the display palette the array indices should start.
sw,sh	(Optional) The width and height of the sprite to create. If omitted, the sprite will be created so that its pixel size is the same as the supplied image array. Otherwise you may optionally set the sprite width and height using these values.

This function loads up a sprite with identification number Key with an image from the matlab workspace. You supply the image using the aw,ah and PixVal arguments. If you want you can scale the sprite using the sw and sh values to be whatever you please. If you want to use the underlying GPrim library functions you may optionally use the returned RASKey value to identify the raster used for this sprite. This function may be used in palette display mode in which case the PalRGB and StartIndex arguments must be present.

cgloadarray(3,100,100,PixVal)

Creates a new sprite with ID number 3. The sprite is the same size as the supplied image which is 100 pixels square. The colour of each pixel in the image is defined by the PixVal array which contains 10,000 x 3 elements (100x100 =10,000).

RASKey = cgloadarray(3,100,100,PixVal,PalRGB,0,50,20)

Creates a new sprite with ID number 3. The sprite will be 50 units wide and 20 units high (The units depend on the current co-ordinate system as selected by **cgscale**). The image is palette based and the palette is contained in the PalRGB argument. The supplied image is 100x100 pixels and PixVal must therefore contain 10,000 elements.
Array palette indices will be loaded with no offset (StartIndex = 0). The underlying GPrim raster identification key for this sprite is stored in RASKey.

RASKey = cgloadarray(3,100,100,PixVal,PalRGB,35,50,20)

Creates a sprite as in the previous example but this time the display is in palette mode and so the StartIndex argument has been added (=35). This means that array palette indices from 0 to n will be offset to palette entries 35 to (n + 35) in the sprite.

<Args=>cgloadbmp(Key,Filename<,Width,Height>) or <Args=>cgloadbmp(Key,Filename,StartIndex<,Width,Height>)

Args = RASKey or [RASKey,RGBPal]

RASKey	(Optional) This gives the ID number of the raster that has been created for this sprite. The raster is created by the underlying GPrim library and should not be required by most users of the GScnd suite.
RGBPal	(Optional) This returns the palette of the loaded bitmap file if it has one. The palette is returned as an array of (n x 3) values where n is the palette size.
Key	The identification number to use for this sprite.
Filename	The name of the BMP image file to use for the sprite.
StartIndex	(Required in palette mode) This gives the starting palette index for the image colours.
Width,Height	(Optional) The width and height of the sprite to create. If omitted, the sprite will be created so that its pixel size is the same as the supplied image array. Otherwise you may optionally set the sprite width and height using these values. If either Width OR Height is zero, the aspect ratio of the image is used to calculate the value as a ratio from the other, non-zero dimension.

This function loads up a sprite with identification number Key with an image from the named file. If you want you can scale the sprite using the Width and Height values to be whatever you please. If you want to use the underlying GPrim library functions you may optionally use the returned RASKey value to identify the raster used for this sprite.

The BMP file should be in uncompressed Windows bitmap (BMP) or Windows Device Independent Bitmap (DIB) format.

cgloadbmp(3,'Demo.bmp')

Creates a new sprite with ID number 3. The sprite is the same size as the supplied image file Demo.bmp.

RASKey = cgloadbmp(3,'Demo.bmp',30,40)

Creates a new sprite with ID number 3. The sprite will be 30 units wide and 40 units high (The units depend on the current co-ordinate system as selected by **cgscale**). The underlying GPrim raster identification key for this sprite is stored in RASKey.

cgloadbmp(3,'Demo.bmp',100,0)

Creates a new sprite with ID number 3. The sprite is 100 units wide and the height will be calculated so that aspect ratio of the original image is maintained.

cgloadbmp(3,'Demo.bmp',0,100)

Creates a new sprite with ID number 3. The sprite is 100 units in height and the width will be calculated so that aspect ratio of the original image is maintained.

cgloadlib<('U')>	
'U'	(Optional) This form of the command is used to unload all the graphics libraries from the matlab workspace. It should not be necessary to use this command under normal circumstances.
This function loads all the graphics libraries into the Matlab workspace and makes them available for use. You should start every matlab script with this command if you want to use graphics.	
cgloadlib	Loads up the graphics libraries into the Matlab workspace, ready for use.
cgloadlib('U')	Unloads the graphics libraries from the Matlab workspace. It should not be necessary to use this command under normal circumstances.

<RASKey=>cgmakesprite(Key,Width,Height<,R,G,B>) (Direct colour mode) or <RASKey=>cgmakesprite(Key,Width,Height<,PI>) (Palette mode)	
RASKey	(Optional) This gives the ID number of the raster that has been created for this sprite. The raster is created by the underlying GPrim library and should not be required by most users of the GScnd suite.
Key	The identification number to use for this sprite.
Width,Height	The width and height of the sprite to create.
R,G,B	(Optional) If present, the sprite will be cleared to this colour. R,G and B represent the red, green and blue components of the colour respectively and take values from 0 to 1. This argument is only valid in direct colour mode.
PI	(Optional) If present, the sprite will be cleared to this palette index. PI represents the palette index and takes values from 0 to 255. This argument is only valid in palette mode.
This function creates a new sprite with identification number Key with dimensions Width and Height. You may also optionally set the whole sprite to a colour using R,G,B or to a palette index using PI. If you want to use the underlying GPrim library functions you may optionally use the returned RASKey value to identify the raster used for this sprite.	
cgmakesprite(5,640,320) Creates a new sprite with ID number 5 and size 640 x 480 units.	
cgmakesprite(5,640,320,1,1,1) Creates a new sprite with ID number 5 and size 640 x 480 units. The whole sprite is cleared to colour 1,1,1 (white). This form is only available in direct colour mode.	
cgmakesprite(5,640,320,12) Creates a new sprite with ID number 5 and size 640 x 480 units. The whole sprite is cleared to palette index 12. This form is only available in palette mode.	

<[x,y,bd,bp]> = cgmouse(<sx,sy>)

Sx,sy	(Optional) Set the mouse position to sx,sy. Sx,sy are in local co-ordinates as set by **cgscale**.
x,y,bd,bp	(Optional) Mouse position and button state. Mouse position at the time of the call to **cgmouse** is given by x,y in local co-ordinates as set by **cgscale**. The button state is given by bd, bp, where bd gives the mouse buttons down when **cgmouse** was called and bp gives the buttons that had been pressed (and possibly released) since the previous time **cgmouse** was called. Buttons are given as the arithmetic sum of the following values:- 1 – left button, 2 – middle button, 4 – right button.

This function reads (and optionally sets) the mouse. When you are reading you may request as many of x,y,bd,bp as you require. This function is not yet fully integrated with the rest of cogent. In particular there may be undesirable interactions between this function and the cogent logging service for mouse events. Use this function with caution.

cgmouse(0,0)

Sets the mouse position to (0,0).

[x,y] = cgmouse;

Reads the current position of the mouse.

[x,y,bd] = cgmouse;

Reads the current position and button state of the mouse.

<S =>cgnewpal<(ImFlag = 'I' or 'i')> (Palette mode)

ImFlag	(Optional) If present, the colour table will be made visible immediately rather than waiting for the next vertical blanking period. This is appropriate when the flip immediately follows a **cgflip** command which automatically synchronises with the VBL anyway.
S	(Optional) Timestamp in seconds for displaying the new palette.

This command makes the current colour table visible. It is only available in palette mode.

You may optionally receive a timestamp indicating the precise time that the colour table became visible using the S variable. This timestamp gives you a time in seconds since you made your initialising call to **cgopen**. Currently the precision of the timestamp is 0.001S (= 1mS).

cgnewpal	Makes the colour table visible at the next VBL.
cgnewpal('I')	Makes the colour table visible immediately.
S=cgnewpal	Makes the colour table visible and records the time it became visible in variable S.

cgopen(Res,BPP,RefRate,Monitor)	
Res	Graphics resolution. Values are:- 1 = 640 x 480 3 = 1024 x 768 5 = 1280 x 1024 2 = 800 x 600 4 = 1152 x 864 6 = 1600 x 1200
BPP	Bits per pixel. 0 bits per pixel is used when you do not know what values are available. It will first try 32, then 24, then 16 bits in that sequence until it successfully opens a screen. 8 bits per pixel gives only 256 different colours on the screen at any time. This mode is memory-efficient and you can perform special graphics effects with it. 16 bits per pixel gives thousands of colours on screen at any time and is fairly memory efficient but is limited in its capability to render subtle colours. 24 and 32 bits per pixel gives 16 million colours on screen at one time and are suitable for photo-quality images. Inefficient use of memory. Bits per pixel is ignored if sub-window (Monitor = 0) is used.
RefRate	Refresh rate in hertz. Specify a rate or specify zero to select an optimal rate for your monitor. In order to select the refresh rate using this command you must have a graphics card and monitor that allow DirectX to control refresh rate. If you do not, use zero for this value and DirectX will use an "optimal" rate. You have some control over this "optimal" rate if you have the DirectX control panel which is mentioned briefly in the "Introduction" section of this manual. This argument is ignored if sub-window mode (Monitor = 0) is used.
Monitor	Which monitor to use. Possible values are:- -2 List all resolutions. Do not open a screen. -1 List all possible monitors. Do not open a screen. 0 Open a sub-window on the main desktop 1 Take over the whole desktop on the main monitor. 2-n Take over subsidiary monitors If sub-window (Monitor=0) is used the bits per pixel and refresh rate of the main desktop are not changed.

Opens a new graphics screen with a specified resolution on the chosen monitor. A refresh rate may also be selected depending on the capabilities of your hardware.

cgopen(1,0,0,0)	Open a new graphics screen with resolution 640 x 480 pixels as a sub-window on the desktop of the main monitor.
cgopen(1,0,0,-2)	Lists available resolutions. Do not open a screen.
cgopen(1,0,0,-1)	Lists available monitors. Do not open a screen.
cgopen(2,24,0,1)	Take over the whole desktop of the main monitor. Set resolution 800x600 pixels, 24 bit pixels. Use an "optimal" refresh rate.
cgopen(3,16,60,1)	Take over the whole desktop on the main monitor for graphics. Make the graphics screen 1024 pixels wide by 768 pixels high and use 16 bits per pixel. Set the refresh rate to 60 Hz.

cgopenmovie(Key,Filename)

Key	The identification number to use for this movie.
Filename	The name of the movie file to load.

This command loads a movie file for later use.

cgopenmovie(7,'movie.avi')

Open the movie file 'movie.avi' and associate it with ID number 7.

cgpencol(R,G,B) (Direct colour mode)

or

cgpencol(PI) (Palette mode)

R,G,B	Drawing colour to use in subsequent drawing operations (commands **cgdraw**, **cgellipse**, **cgpolygon**, **cgrect**, **cgtext**). R, G and B set the red, green and blue components of the colour respectively. They can take values between zero and one. This argument is only valid in direct colour mode.
PI	Palette index to use in subsequent drawing operations (commands **cgdraw**, **cgellipse**, **cgpolygon**, **cgrect**, **cgtext**). PI sets the palette index to use (0 to 255). This argument is only valid in palette mode.

Set the drawing colour for use in subsequent drawing operations.

cgpencol(1,0,0)	Set the drawing colour to be 1,0,0 (maximum red).
cgpencol(22)	Set the drawing palette index to 22.

cgpenwid(Width)

Width	Line width to use in subsequent drawing operations (commands **cgdraw** and **cgellipse**). Units depend on the current co-ordinate system as selected by **cgscale**. A value of zero always sets the line width to a single pixel.

Set the line width for use in subsequent drawing operations.

cgpenwid(5)	Set the line width to be 5 units.

cgphotometer('Open','PhotometerID',PortNum) or cgphotometer('Shut') or str=cgphotometer('ID') or xyz = cgphotometer('XYZ') or spc = cgphotometer('SPC')

PhotometerID	Currently the only photometer supported is:- 'PR650' – PhotoResearch PR-650 Spectra-Colorimeter
PortNum	Serial port to which the photometer is connected. Takes values from 1 to 8.
str	Array of two strings. The first string identifies the version number of the cgphotometer dll. The second string identifies the photometer.
xyz	Array with three elements containing the CIE (1931) X,Y and Z values from the last measurement.
spc	(n x 2) array containing the radiant spectrum from the last measurement. spc(:,1) contains the wavelength in nanometres for each measurement, spc(:,2) contains the spectral radiance in $Wm^{-2}sr^{-1}nm^{-1}$

Communication with photometer.

You must first open communications with the 'Open' command and finish by closing communications with the 'Shut' command.

The 'XYZ' command measures light and returns the CIE (1931) XYZ values. You may then also download the radiant spectrum of the measurement using the 'SPC' command.

cgphotometer('open','pr650',1)	Open communications with the photometer. The PR650 should be connected to serial port COM1.
cgphotometer('shut')	Shut communications with the photometer.
idstr = cgphotometer('ID')	Obtain the identification strings. Typically:- GScnd:cgPhotometer v1.24 Compiled:Oct 23 2002 PhotoResearch Spectra-Colorimeter Model PR-650 SN:60954201
xyz = cgphotometer('XYZ')	Measure light and return the CIE (1931) XYZ values.
spc= cgphotometer('SPC')	Return the spectrum of the last measurement.

cgplaymovie(Key<x,y,w,h>)

Key	The identification numberof the movie to play.
x,y,w,h	Where it should appear on the display.

This command plays a movie that has been opened with the cgopenmovie command. If you omit the x,y,w,h parameters the movie will play full screen. Otherwise it appears at the requested position. The position is affected by the **cgalign** command. You cannot interrupt the movie once it has been started.

cgplaymovie(7)	Play movie number 7.
cgplaymovie(33,0,0,320,240)	Play movie number 33 at position 0,0 with width 320 units and height 240 units. Position is dependent on the **cgalign** command settings.

cgpolygon(x,y<,XOffset,Yoffset>)	
x,y	These arrays hold the co-ordinates of the vertices (corners) of the polygon. There must be equal numbers of elements in arrays x and y.
XOffset,Yoffset	(Optional) Offset to be applied to the x,y co-ordinates. This provides a quick way of moving the polygon without having to adjust all the x,y array elements.
Draws a filled polygon on the current destination (as set by **cgsprite**) in the current drawing colour as set by **cgpencol**.	
x=[10 0 15]; **y=[0 20 5];** **cgpolygon(x,y)**	Draw a filled polygon in the current drawing colour. The polygon has the following vertices:- (10,0) (0,20) (15,5)
x=[10 0 15]; **y=[0 20 5];** **cgpolygon(x,y,-10,20)**	Draw a filled polygon in the current drawing colour offset by –10 units in the x axis and 20 units in the y axis. The resulting polyon has the following vertices:- (0,20) (-10,40) (5,25)

cgrect<(x,y,w,h<,Col>)>)	
x,y,w,h	(Optional) Position (x,y), width(w) and height (h) of the rectangles. If omitted, fill the whole destination. The position of the rectangle is defined by (x,y) but is modified by the **cgalign** setting.
Col	(Optional) The colours for the rectangles. When in palette mode, Col must be an (n x 1), (single column), array containing the palette index for each item. When in direct mode, Col must be an (n x 3) array containing the RGB levels (0 to 1) for each item.

Draws filled rectangles on the current destination (as set by **cgsetsprite**). If you include the Col argument, each ellipse will be drawn in the requested colour, otherwise the current drawing colour (as set by **cgpencol**) will be used.

The arrays (x, y, w, h and Col) must have the same number of entries. When in direct mode where Col is an (n x 3) array, 'n' gives the number of entries.

cgrect	Fill the whole of the current destination with the current drawing colour.
cgrect(0,0,100,200)	Fill in a rectangle 100 units wide by 200 units high aligned on the point (0,0).

cgrect([100 100],[-100 100],[100 50],[50 100],[1 0 0;0 0 1])

Draws two filled rectangles in direct colour mode. One is drawn with aligned on (100,-100) with width 100 and height 50 in colour 1,0,0 (red). The other is drawn aligned on (100,100) with width 50 and height 100 in colour 0,0,1 (blue).

cgrect([100 100],[-100 100],[100 50],[50 100],[1;2])

Draws two filled rectangles in palette mode. One is drawn aligned on (100,-100) with width 100 and height 50 in palette index 1. The other is drawn aligned on (100,100) with width 50 and height 100 in palette index 2.

cgscale or cgscale(ScrWidDeg) or cgscale(SrcWidmm,ObsDstmm)	
No arguments	Selects pixel co-ordinates.
ScrWidDeg	(Optional) Specifies the width of the screen in degrees of visual angle. Selects degrees of visual angle as the co-ordinate system for subsequent drawing operations.
ScrWidmm,ObsDstmm	(Optional) Specifies the width of the screen and the observer distance in millimetres. Selects degrees of visual angle as the co-ordinate system for subsequent drawing operations.

Sets the screen scaling and co-ordinate system for subsequent drawing commands including **cgdraw**, **cgdrawsprite**, **cgellipse**, **cgloadarray**, **cgmakesprite**, **cgpenwid**, **cgpolygon**, **cgrect**, **cgtext**.

cgscale	Set pixel co-ordinates.
cgscale(25.4)	Set the screen width to be 25.4 degrees of visual angle and select visual angle as the co-ordinate system to use.
cgscale(400,650)	Set the screen width to be 400mm and the observer distance to be 650mm and select visual angle as the co-ordinate system to use.

cgscrdmp or cgscrdmp(Filename) or cgscrdmp(Defname,Defnumber)

When called without arguments this command creates a screen dump of the current display and saves it in a file with the current default name. The default name is of the form DefRootNNNNN.BMP where DefRoot is a text prefix and NNNNN is a five digit number. The initial value of DefRoot is 'CGSD' (pseudo-acronym for **C**o**G**ent **S**creen**D**ump) and the initial value of NNNNN is 00001. The value of NNNNN is incremented each time such a file is requested. So the first file is named CGSD00001.BMP, the second is named CGSD00002.BMP and so on.

When called with just the Filename argument this command creates a screen dump of the current display and saves it in a file named Filename.BMP.

When called with both the Defname and Defnumber arguments no screen dump is created but the default filename DefRoot and NNNNN values as described above are set to Filename and Filenumber respectively.

Cgscrdmp	Save the current display as an image file. The first file created will be called (by default) CGSD00001.BMP, the second CGSD00002.BMP and so on.
cgscrdmp('jpr')	Save the current display as an image file named 'jpr.BMP'.
cgscrdmp('Picts/aa',10)	Do not create an image file but set the default filenaming to be:- Picts/aa00010.BMP, Picts/aa00011.BMP etc... This puts all the screendump files in a sub-folder named 'Picts' (You must create the Picts subfolder beforehand).

cgsetsprite(Key)

Key	The ID number of the sprite to use for subsequent drawing operations. Allowable values are 1 to 10000 or if you specify zero the offscreen area will be used.

Sets the destination for subsequent drawing commands including **cgdraw**, **cgdrawsprite**, **cgellipse**, **cgpolygon**, **cgrect**, **cgtext**.

cgsetsprite(17)	From now on, draw into sprite number 17.
cgsetsprite(0)	From now on, draw into the offscreen area.

cgshut

Close down graphics, closing the screen, freeing all memory and cleaning up before exit.

cgshut	Closing the screen and cleaning up before exit.

cgshutmovie(Key)	
Key	Identification number of the movie to be closed.
Close a movie, freeing up the memory it uses.	
cgshutmovie(23)	Closes up movie number 24

cgtext(Text,x,y)	
Text	The text to draw.
x,y	Where to draw the text.
Draws the specified text on the current destination (as selected by **cgsetsprite**) in the current font (as selected by **cgfont**) in the current colour (as selected by **cgpencol**) at the point (x,y), using the current alignment mode (as selected by **cgalign**).	
cgtext('abcdef',0,0)	Draws the text abcdef at the point (0,0).

cgtracker('Open',TrackerID,PortNum,Mode,BaudRate<,c1,c2,c3,c4>) or cgtracker('Shut') or cgtracker('Calibrate'<BckCol,ForCol>) or eyedat = cgtracker('eyedat') or cgtracker('start') or eyearray = cgtracker('stop')	
TrackerID	Currently the only eyetracker supported is:- 'ASL5000' – ASL model 5000 control unit.
PortNum	Serial port to which the eyetracker is connected (1 to 8).
Mode	Set this value to 1 if the eyetracker is operating in "On Demand" mode (the currently recommended mode). Set it to 0 if the eyetracker is operating in "Continuous" mode.
BaudRate	Baud rate for the serial port. Default value is 57600 although the following other values are also technically possible:- 9600, 19200, 38400.
c1,c2,c3,c4	Calibration co-ordinates of Target points 1 and 9. If these optional values are used the eye data will be in cogent screen co-ordinates. Otherwise they will be in arbitrary units.
BckCol, ForCol	Optional colour definitions for the calibration screen. These should be specified as red, green and blue colour levels, 0 to 1.
eyedat	This is a structure containing the current eye data:- eyedat.X – Eye X co-ordinate. eyedat.Y – Eye Y co-ordinate. eyedat.Pupil – Pupil radius eyedat.Status – Status as returned by ASL5000 control unit. Zero means normal reading. Other values indicate special conditions (see ASL manual) If the calibration co-ordinates **c1,c2,c3,c4** were supplied when communications were opened, the X and Y values will be in cogent screen co-ordinates. Otherwise they will be in arbitrary units. A pupil radius of zero means that the pupil was not discriminated. The subject was either looking away or had closed their eye.

Communication with eyetracker.

You must first open communications with the 'Open' command and finish by closing communications with the 'Shut' command. The 'Calibrate' commands displays the standard calibration screen, the 'eyedat' command returns the current eye position. The 'Start' and 'Stop' commands are used to retrieve a sequence of positions.

cgtracker('Open','ASL5000',1,1,57600,46,55,219,214)	
Opens communications with the ASL5000 eyetracker on serial port 1 (COM1) in "On Demand" mode at 57600 baud with calibration co-ordinates 46,55,219,214	
cgtracker('shut')	Closes communications with the eyetracker.
cgtracker('calibrate',[1 0 0],[0 0 1])	Displays the eyetracker calibration screen with a red [1 0 0] background and blue [0 0 1] characters.
eyedat = cgtracker('eyedat')	Obtains the latest eyedata from the eyetracker.
cgtracker('start')	Starts recording eye data
eyearray = cgtracker('stop')	Stops recording eyedata and returns an array of eyedat structures.

cgtrncol(Key<,TCol>) (Direct colour mode)

or

cgtrncol(Key<,PI>) (Palette mode)

Key	The ID number of the sprite.
TCol	Transparent colour for the sprite. Omit this argument to set no transparency. Valid values are as follows:- **'n'** selects black (0,0,0) **'r'** selects maximum red (1,0,0) **'g'** selects maximum green (0,1,0) **'y'** selects maximum yellow (1,1,0) **'b'** selects maximum blue (0,0,1) **'m'** selects maximum magenta (1,0,1) **'c'** selects maximum cyan (0,1,1) **'w'** selects maximum white (1,1,1) This argument is only valid in direct colour mode.
PI	Transparent palette index for the sprite (0 to 255). Omit this argument to set no transparency. Only valid in palette mode.

Set the transparent colour of a sprite.

cgtrncol(23,1,0,0)	Set the transparent colour of sprite 23 to be 1,0,0 (maximum red).
cgtrncol(17,22)	Set the transparent palette index of sprite 17 to 22.
cgtrncol(32)	Sets no transparency for sprite 32.

cgvers<(Mode)>

Mode	(Optional) Mode can be 'U' for usage or 'C' for copyright.

Prints out version information for all GScnd commands on the matlab console. This is useful for checking what version of the library you are using. It also prints out a warning if there are any inconsistencies in the version numbers of the various components. When called with the 'U' argument you get a usage guide as well for all commands giving a synopsis of the arguments that can be passed to each command. The 'C' argument prints a copyright message for each command.

cgvers	Prints out version information for all GScnd commands on the console.
cgvers('U')	Prints out a full usage guide for all GScnd commands on the console.
cgvers('C')	Prints out a copyright message for all GScnd commands on the console.

Data Values

Introduction

The **cgGetData()** function can be used to obtain matlab data structures from the underlying Cogent Graphics libraries. This data may be of use in some script applications so this method has been provided to access it. A deeper knowledge of the libraries may be required to understand some of the data members of these structures and in that case you should consult the programmer's manuals for the CogStd and GPrim libraries which are available from the web distribution pages.

The cggetdata() function has the following forms:-

csd = cggetdata('CSD')	Requests the CogStdData structure
gpd = cggetdata('GPD')	Requests the GPrimData structure
ras = cggetdata('RAS',RASKey)	Requests the data for RAS number RASKey
dib = cggetdata('DIB',DIBKey)	Requests the data for DIB number DIBKey
gsd = cggetdata('GSD')	Requests the GscndData structure
spr = cggetdata('SPR',SPRKey)	Requests the data for Sprite number SPRKey
mvd = cggetdata('MVD',MVDKey)	Requests the MOVData structure for MOV number MVDKey
mve = cggetdata('MVE',MVEKey)	Requests the Movie structure for movie number MVEKey

In general you can use the cgGetData() function in the following way:-

```
>> csd=cggetdata('CSD');
```

This command returns the CogStdData structure to the matlab variable csd. You can look at the individual members of the CogStdData structure by typing **csd** as below:-

```
>> csd

csd =

    Version: 124
CogStdString: 'CogStd v1.24 Compiled:Oct 23 2002'
```

Here you can see that the csd structure has two members. The first is called 'Version' and it has the value 124. The second is called 'CogStdString' and it has the value:- 'CogStd v1.24 Compiled:Oct 23 2002'.

The rest of this manual gives a description of the different data structures and the members they contain.

CogStdData structure

The CogStdData structure contains data from the CogStd library. It contains the following members:-

Version	The version number of the CogStd library multiplied by 100. Thus, if the version number is 1.24, the value of 'Version' will be 124.
CogStdString	A character string identifying the CogStd library version and compilation date.

GPrimData structure

Version	The version number of the GPrim library multiplied by 100. Thus, if the version number is 1.24, 'Version' will be 124.
GPrimString	A string identifying the GPrim library version and compilation date.
GLibString	A string identifying the underlying graphics library version and compilation date.
PixWidth	The width of the display screen in pixels.
PixHeight	The height of the display screen in pixels.
BitDepth	The number of bits for each pixel.
RefRate100	The display refresh rate in Hertz, multiplied by 100. Thus if the refresh rate is 75.34 Hz, the value of 'RefRate100' will be 7534.
TranCOL	'COL' structure (see below) containing the current transparent colour.
DrawCOL	'COL' structure (see below) containing the current drawing colour.
Fontname	The name of the currently selected font.
PointSize	The pointsize for text drawing.
LineWidth	The current line width.
CurrentRASKey	The number of the currently selected raster.
CurrentDIBKey	The number of the currently selected DIB.
AlignX	Horizontal alignment mode:- 0 = left justified, 1 = centred, 2 = right justified
AlignY	Vertical alignment mode:- 0 = top, 1 = centre, 2 = bottom
ColTable	An array of COLORREF structures (described below) containing the current palette. Size of the array is given by PalSize.
PalSize	The number of entries in the palette.
Flags	This contains binary flag values. Currently the only flag used is:- 2 If this is set then the flip function cannot be synchronized with the vertical blanking interrupt.
NextRASKey	The ID number of the next Raster.
NextDIBKey	The ID number of the next DIB.
MouseX,MouseY	Mouse position in pixels. (0,0) is the top left of the screen, increasing to the right and downwards. These values are updated only when the mouse is read.
MouseS,MouseP	Mouse button state. MouseS has the mouse buttons that were down when the mouse was last read. MouseP has the buttons that have been pressed (and possibly released) since the previous time the mouse was read. Button values are the arithmetic sum of :- 1 - left button, 2 - middle button, 4 - right button MouseS can also include:- 8 - Control key down, 16 - Shift key down
KeyS,KeyP	Key state; each is an array of three values containing the bitwise map of keys down and pressed since the keys were last read.
SDFilename, SDFilenumber	These two variables are combined to create a filename for the screendump file when none is supplied to the cgScrDmp() command. The filename is SDFilenameNNNNN.BMP where NNNNN is the five-digit value of SDFilenumber. These variables are initialized to 'CGSD' and 1 respectively.
MouseOffsetX, MouseOffsetY	These two variables have been added to fix a bug with the cgMouse function when operating in dual-monitor mode. They represent a correction factor that has to be applied internally.

The 'COL' structure contains the following members:-

CR	When in direct colour mode this member is a 'COLORREF' structure (described below) which contains the colour definition in terms of red, green and blue components.
PV	When in palette mode this member contains the palette index number of the colour.

The 'COLORREF' structure contains the following members:-

Red	The red component of the colour – ranging from 0 to 255.
Grn	The green component of the colour – ranging from 0 to 255.
Blu	The blue component of the colour – ranging from 0 to 255.

You can access these substructures in the following way:-

```
>> gpd.DrawCOL.CR

ans =

    Red: 255
    Grn: 255
    Blu: 255
```

Here we are looking at the DrawCOL.CR substructure of the gpd GPrimData structure.

RAS structure

The RAS structure also comes from the GPrim library. It describes a Raster and contains the following members:-

w	The width of the raster in pixels.
h	The height of the raster in pixels.

For example:-

```
>> ras=cggetdata('RAS',0)

ras =

    w: 640
    h: 480
```

Here we can see that Raster 0 (the display screen) has a width of 640 pixels and a height of 480 pixels.

DIB structure

The DIB structure also comes from the GPrim library. It describes a DIB (Device-Independent Bitmap) and contains the following members:-

FName	The filename of the DIB.
w	The width of the DIB in pixels.
h	The height of the DIB in pixels.
C	The number of colours in the DIB palette.

For example:-

```
>> dib=cggetdata('DIB',1)

dib =

   FName: 'demo'
      w: 320
      h: 240
      c: 256
```

Here we can see that DIB 1 was loaded from file ‘demo.bmp’. It has a width of 320 pixels and a height of 240 pixels and is a palette-based image with 256 colour entries.

MOVData structure

The MOVData structure also comes from the GPrim library. It describes a movie and contains the following members:-

Filename	The name of the movie file.

For example:-

```
>> mvd=cggetdata('MVD',1)

mvd =

   Filename: ‘movie.avi’
```

Here we can see that MOVData 1 was created from movie file ‘movie.avi’.

GScndData structure

The GScndData structure contains data from the GScnd library (named cgData.dll). It contains the following members:-

Version	The version number of the GScnd library multiplied by 100. Thus, if the version number is 1.24, the value of 'Version' will be 124.
GScndString	A character string identifying the GScnd library version and compilation date.
CurrentRAS	The currently selected Raster ID number.
Flags	This contains binary flag values. Currently the only flag used is:- 1 This flag is set to indicate that the GScnd library has opened successfully.
AlignX	Horizontal alignment mode:- 0 = left justified 1 = centred 2 = right justified
AlignY	Vertical alignment mode:- 0 = top 1 = centre 2 = bottom
ScreenWidth	The width of the display screen in pixels.
ScreenHeight	The height of the display screen in pixels.
ScreenBits	The number of bits for each pixel.
PixScale	This conversion factor is used to multiply GScnd co-ordinates to convert them into GPrim co-ordinates.
PixOffsetX,PixOffsetY	These offsets are added to GScnd co-ordinates to convert them into GPrim co-ordinates.

Sprite structure

The Sprite structure also comes from the GScnd library. It describes a Sprite and contains the following members:-

RASKey	The identification number of the underlying GPrim Raster.
TCol	A 'COL' structure (described below) containing the transparent colour for the sprite.
Width	The width of the sprite in pixels.
Height	The height of the sprite in pixels.

The 'COL' structure contains the following members:-

CR	When in direct colour mode this member is a 'COLORREF' structure (described below) which contains the colour definition in terms of red, green and blue components.
PV	When in palette mode this member contains the palette index number of the colour. Otherwise this member is set to –1.

The 'COLORREF' structure contains the following members:-

Red	The red component of the colour – ranging from 0 to 255.
Grn	The green component of the colour – ranging from 0 to 255.
Blu	The blue component of the colour – ranging from 0 to 255.

Movie structure

The Movie structure also comes from the GScnd library. It describes a movie and contains the following members:-

MOVKey	The ID number of the underlying GPrim MOVData structure.
Filename	The name of the movie file.

For example:-

```
>> mve=cggetdata('MVE',3)

mve =

   MOVKey: 1
   Filename: 'movie.avi'
```

Here we can see that movie structure 3 has an underlying GPrim MOV of ID number 1 and that it was created from movie file 'movie.avi'.

Multiple Displays

Introduction

You can set up Cogent Graphics so that the display appears on a different monitor to your matlab screen. This can be a somewhat tricky operation and I have found that different systems need fiddling about with until the display is right. In this section I will describe how I have set up the displays on a number of computers in my lab. I hope that one of these case studies resembles your setup.

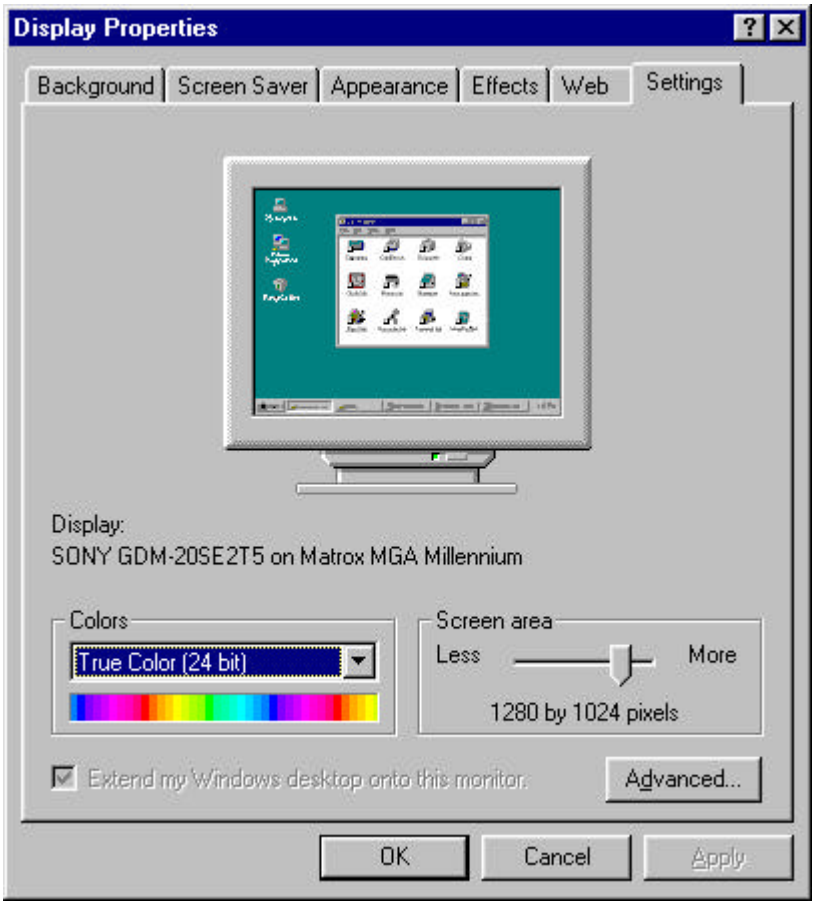
Recommendations

Your multi-monitor system should be set up so that all screens are in direct colour mode (15, 16, 24 or 32 bit colour). Otherwise, if you use palette mode Cogent Graphics, any other screen that is in 256 colour mode will be affected by the Cogent Graphics palette.

The cgMouse function works correctly with single and dual monitor systems. If you have more than two monitors however, cgMouse may behave unpredictably. To avoid this, use the display control panel to position your Cogent Graphics monitor so that it is the right-most and bottom-most monitor on the system.

Single Monitor Display

If you open the "Displays" control panel and then click the "Settings" tab you should see the "Display Properties" dialog box:-



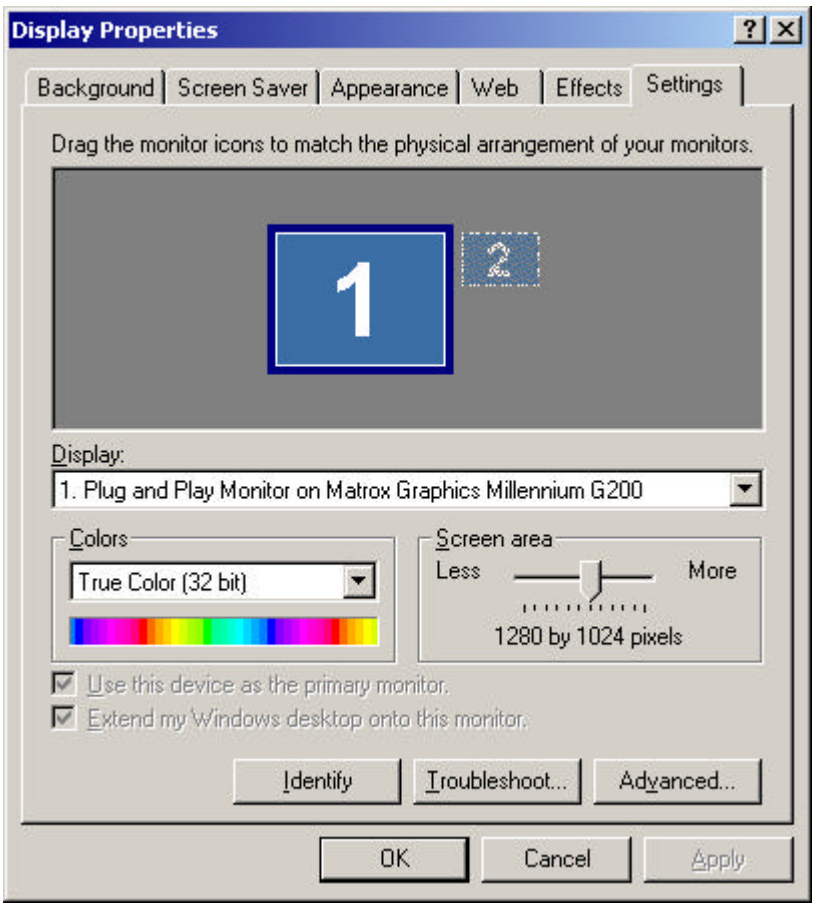
This shows the display properties on a system that has only one display. When you use the **cgopen(1,0,0,-1)** command to list the available devices, you will see a single device listed; the "Primary Display Driver".

```
>> cgloadlib
>> cgopen(1,0,0,-1)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Available devices:-
    1:display (Primary Display Driver)
```

On a system like this you can only open a Cogent Graphics display on the one screen. You may open a sub-window on the display using zero as the device number for **cgopen()**, or you may open a full screen display using one as the device number for **cgopen()**.

Single graphics card, dual display

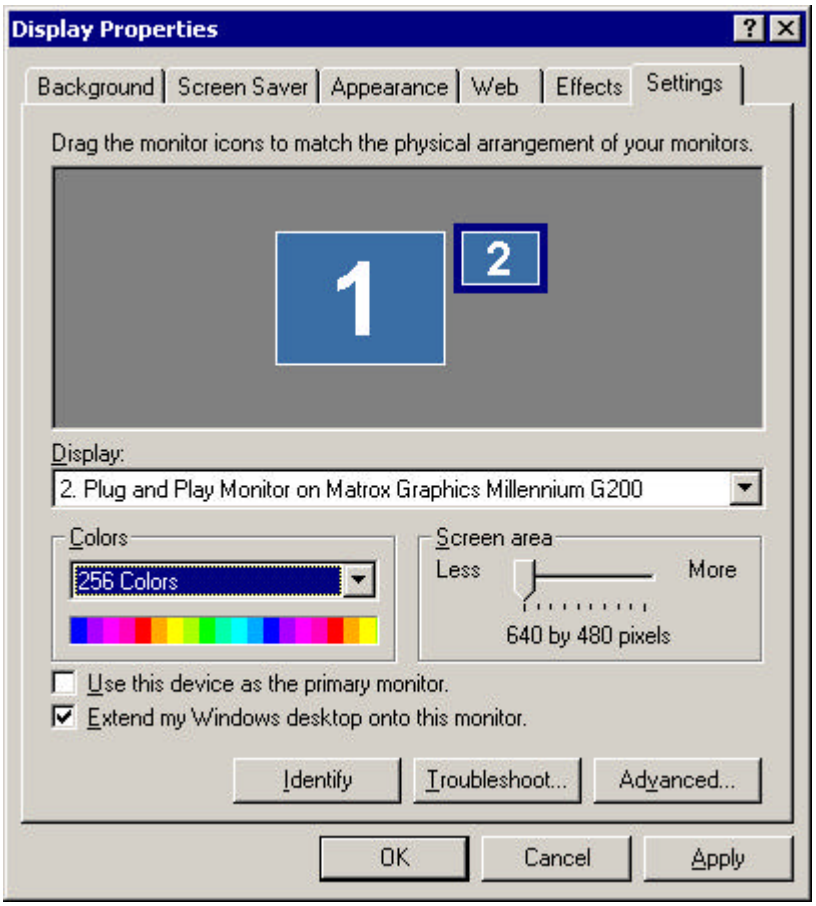
Some graphics cards such as the Matrox Millenium G200 have a multiple display capability. On such systems the "Display Properties" dialog box looks like this:-



The display properties box shows that there are two displays on this system, numbered 1 and 2. However, only display number 1 is enabled because display number 2 is ghosted out. When we use the **cgopen(1,0,0,-1)** command to list available displays we get something like this:-

```
>> cgloadlib
>> cgopen(1,0,0,-1)
GPrim   v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Available devices:-
    1:\\.\DISPLAY1 (Matrox Graphics Millennium G200)
```

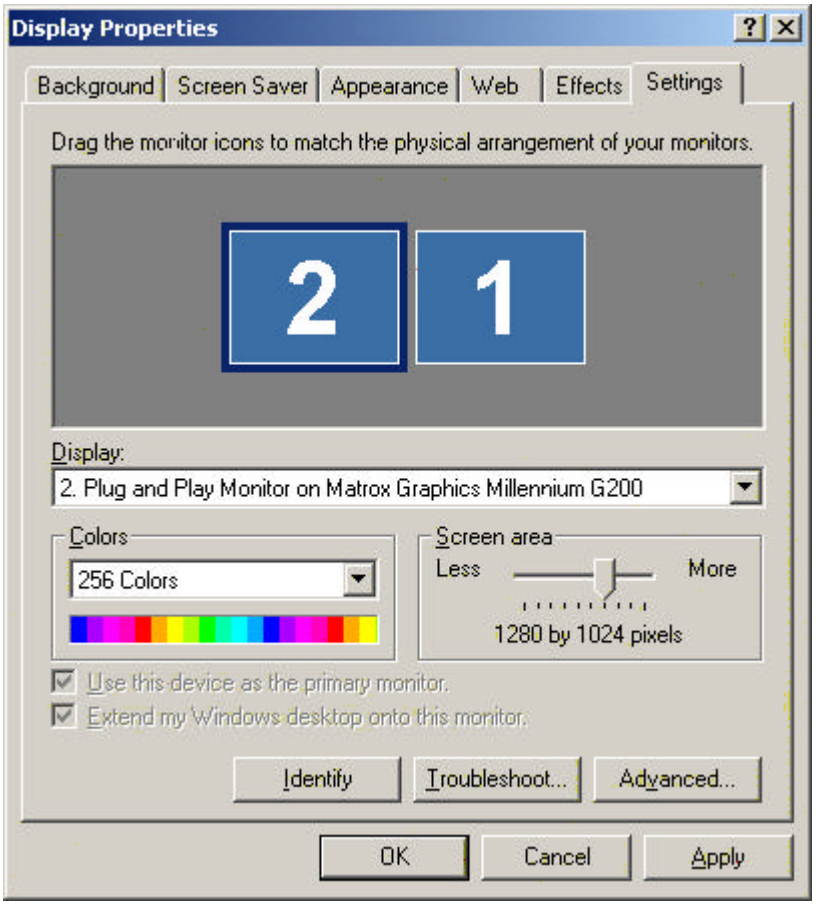
Once again it appears that only one display is available for use. In order to use both displays we must first click on the ghosted-out "2" display to select it and then select the "Extend my Windows desktop onto this monitor" box:-



Next, click on the "OK" button to accept this setting. When we next use **cgopen(1,0,0,-1)** we find that two displays are now available:-

>> **cgopen(1,0,0,-1)**
GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Available devices:-
1:\\.\DISPLAY1 (Matrox Graphics Millennium G200)
2:\\.\DISPLAY2 (Matrox Graphics Millennium G200)

However, this is not the end of the story. Unfortunately, hardware accelerated graphics are only available on display 1 so we must use display 1 for our Cogent Graphics and use display 2 for our windows screen. To do this, select display 2 as shown above and then select "Use this device as the primary monitor". You will probably also want to adjust the "Screen area" and "Colors" settings as you will be using display 2 for your windows screen. You may find that you are limited in what you can choose for display 2 and you may for example have to settle for 256 colours on your Windows screen. At this point you can also move the icon for display 2 (your Windows screen) to a different position relative to display 1 by dragging it with the mouse:-



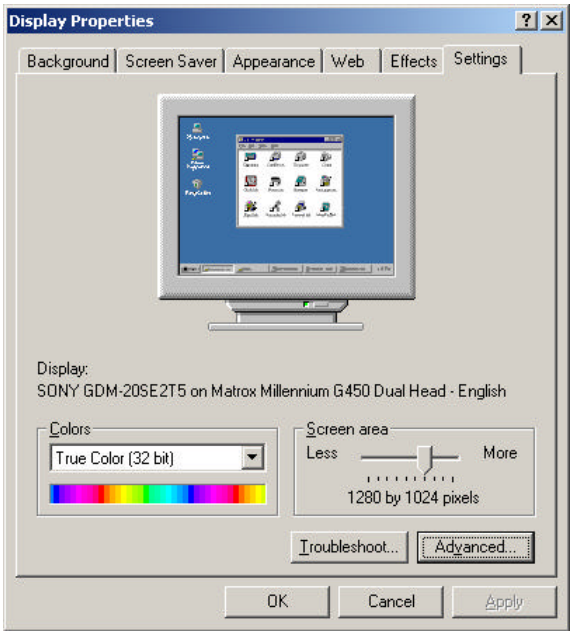
Then click the "OK" button. You may need to confirm the new settings within a certain time period if you change the display dimensions. You must then drag the windows taskbar onto your #2 display using the mouse and reboot your system. You must have a monitor connected to your #2 video output to complete this stage.

If you set up your multi-display system in this way you can open your Cogent Graphics screen on device1 so that it will run with full graphics speed by selecting 1 as the device number in the **cgopen** command. If you set up your system in this way then it will be optimized for Cogent Graphics.

Of course, other configurations are also possible; you can deselect display 2 in the control panel and just use display 1. Or you could keep display 1 as your windows screen and open your Cogent Graphics to run without hardware acceleration in display 2 by using 2 as the device number in the **cgopen** command.

Single display, dual monitor

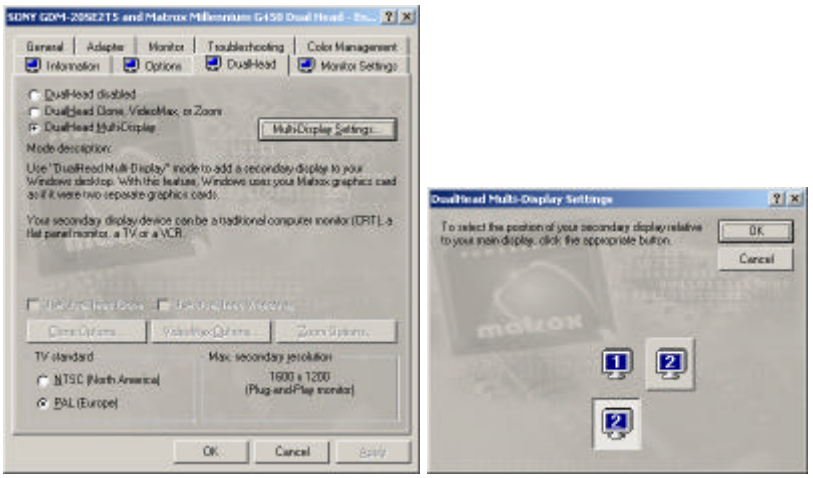
Some graphics cards such as the Matrox Millenium G450 Dual Head allow you to create an extra large windows desktop which spreads over two monitors. Unfortunately this facility is of little use for Cogent Graphics as there is only one display controller and so effectively this is just a single display system. The display properties box looks like this:-



When you use **cgopen** to list the available displays you get the following:-

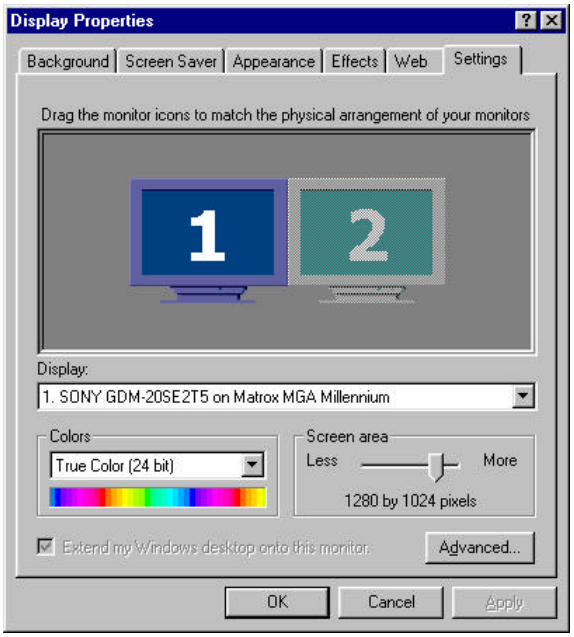
>> **cgloadlib**
>> **cgopen(1,0,0,-1)**
GPrim v1.24 Compiled:Oct 23 2002 (GLib DirectDraw v7 Compiled:Oct 23 2002)
Available devices:-
1:display (Primary Display Driver)

When you click on the "Advanced" button you get a custom G450 dialog box which has a "DualHead" tab. You can select "DualHead Multi-Display" and when you click on the "Multi-Display Settings" button you get another custom dialog box which lets you configure two monitors. However, there is still only a single graphics controller so unfortunately all this is of little use with regard to Cogent Graphics.

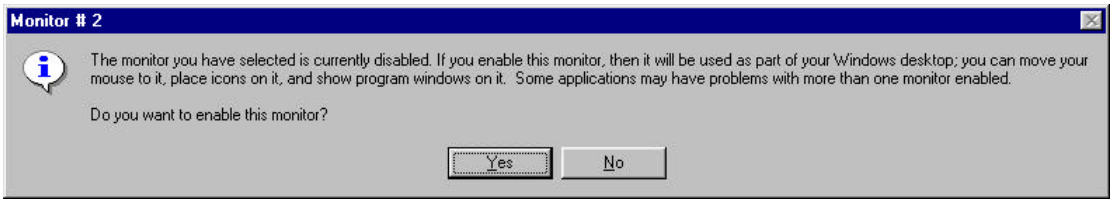


Two graphics cards, one disabled

Some computers are supplied with a built-in graphics card on the motherboard. When you add another graphics card the motherboard graphics are disabled. Although it seems that the system has two graphics cards it is in fact impossible to use the motherboard graphics card. When you open the Display Settings you get something like this:-



Two displays are shown, one ghosted out. If you click on the ghosted out display you get a message similar to this:-



If you click on "Yes" the display seems to be enabled but when you click on "Apply" on the Display settings box the display becomes ghosted again to indicate that it cannot be enabled.

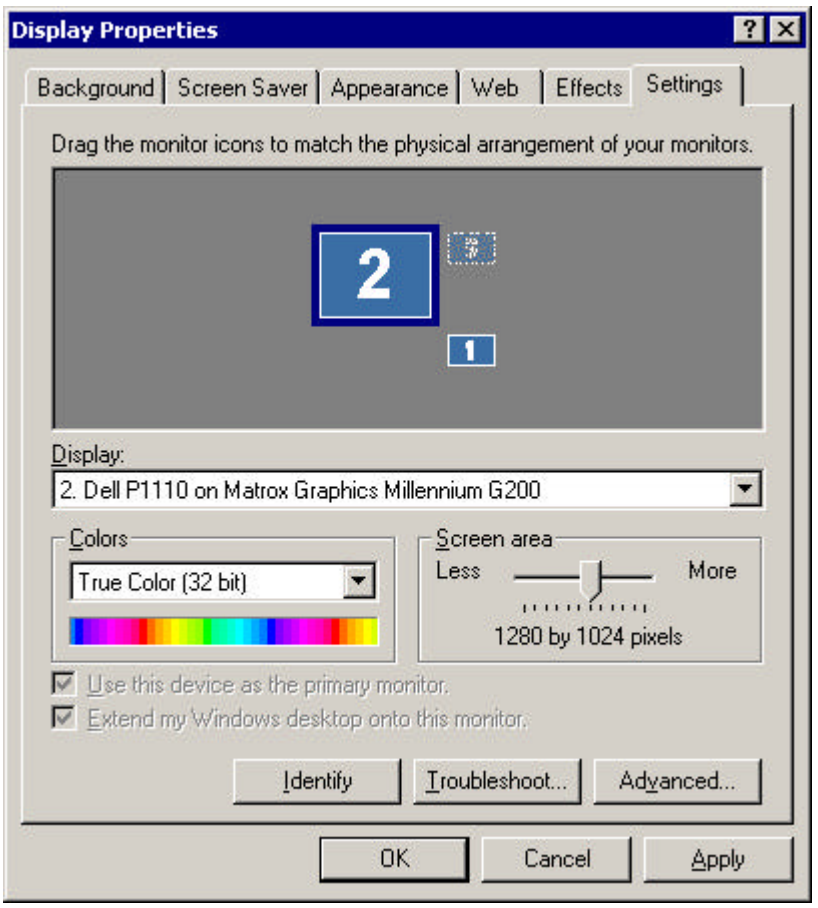
When you use **cgopen** to list the available displays you get something like this:-

```
>> cgloadlib
>> cgopen(1,0,0,-1)
GPrim  v1.24 Compiled:Oct 23 2002  (GLib DirectDraw v7 Compiled:Oct 23 2002)
Available devices:-
    1:\\.\Display1 (Matrox MGA Millenium)
```

Only the enabled display is listed.

Two graphics cards

If your system has two graphics cards it should be set up so that both displays are enabled. You may get better performance if you use display number 1 for Cogent Graphics and switch the windows to display 2, or it may not make any difference. You will have to experiment to see what configuration is best. Typically your Display Properties settings will look like this:-



Remember to click "Apply" and to reboot the system before using your new settings.

Eyetracker Setup

Eyetracker introduction

Cogent Graphics can communicate with the departmental eyetrackers using the Applied Science Laboratories (ASL) model 5000 control unit.

This manual is not intended to document the use of the eyetracker itself and you should acquaint yourself with the operation of the eyetracker before attempting to use it with cogent.

The sections which follow deal with setting up the eyetracker and connecting it to your cogent PC. This information is correct as of 30th July 2002.

Eyetracker software

The eyetracker requires a separate PC for its operation and there is an ftp site where you can download the latest ASL eyetracker software for it:-

ftp://asluser:aslftp@a-s-l.com/pub/

You can download the latest manual for your eyetracker from:-

ftp://asluser:aslftp@a-s-l.com/pub/manuals/

You should also download the latest version of the "eyepos" program from the following directory:-

ftp://asluser:aslftp@a-s-l.com/pub/eyepos/e5win/

The file to download is currently named **eyps121.exe**. It is a self-expanding archive file which will create a folder containing the program files when you run it.

The controller program for the eyetracker is named **e5Win.exe**.

Controlling the eyetracker

You need a separate PC to control the eyetracker. This PC should be connected to the ASL model 5000 control unit using the PC serial port (usually a male 9 pin D-type connector on the back of the PC). This connects to a similar male 9 pin D-type connector on the back of the ASL model 5000 control unit labeled "Controller (4)".

The cable wiring between these connectors should be as follows:-

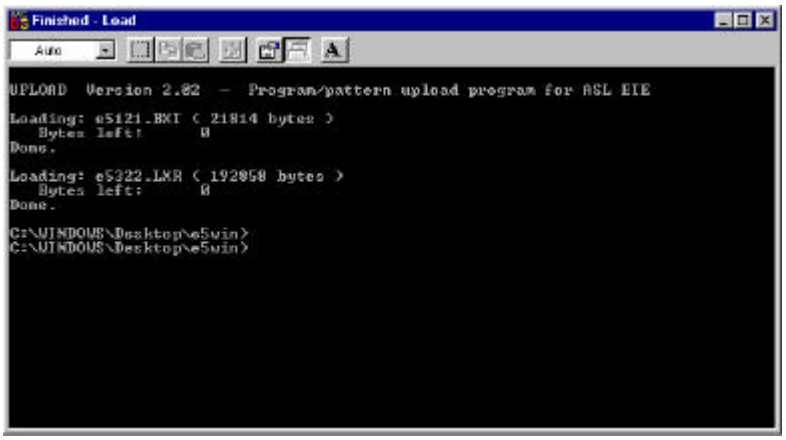
Pin	Pin
2	2
3	3
5	5

The wiring should be exactly as shown above with pins 2, 3 and 5 connected to corresponding pins on both connectors; i.e. no reversal of pins 2 and 3.

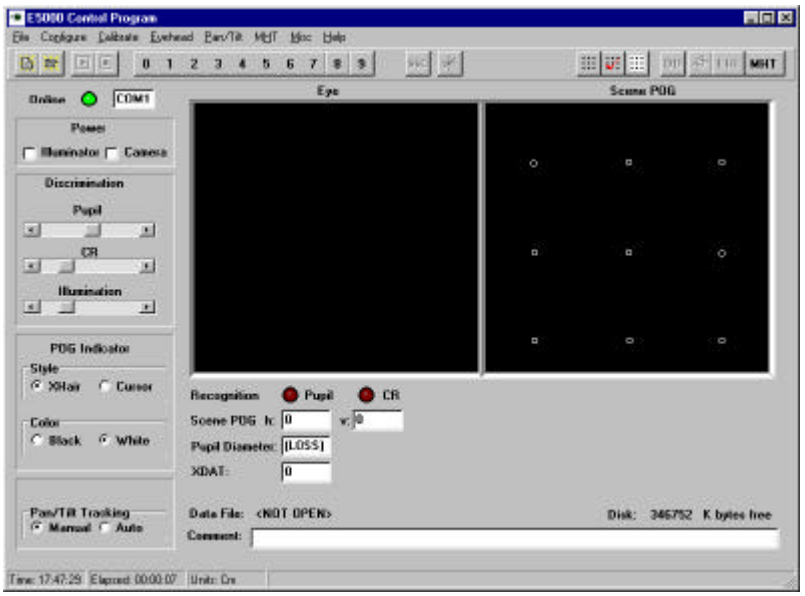
Typically the following ports on the back of the ASL model 5000 should also be connected:-

Port	**Connected to**
DC In	Power adaptor
Camera	Eye camera
Eye Out	CRT monitor
Scene Out	CRT monitor
Remote Scene **OR** Svid In	Camera or computer display via VGA convertor

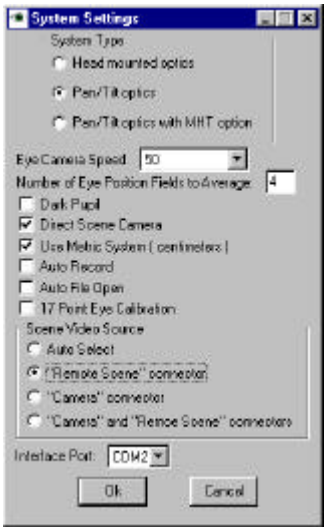
You should then switch on the ASL model 5000 and the eye camera. Then you should upload some software to the ASL model 5000 from the eyetracker PC. You have to do this each time you switch on the ASL model 5000. You do this by running a batch file which you will find in the same folder as the **e5Win.exe** program. The file you run depends on the operating system of the eyetracker PC. If you are using Windows 95 or Windows 98 you should run **load.bat**. If you are using Windows 2000 you should run **load_nt.bat**. A console window will appear on the desktop and two files should upload to the ASL model 5000:-



If there is a problem with uploading, switch the ASL model 5000 off, wait 15 seconds, switch it on and start again. Once you have uploaded the software you can run the controller program **e5Win.exe**:-



You should check that the "Online" button is green which indicates that the link has been established. If it is red then there is a problem. Check the cables into the ASL model 5000 again and check that you have uploaded the software correctly. Check also that you are using the correct serial port on the controller PC. You should be using the serial port for "COM1". If you need to use another serial port you should select "System Settings" from the "Configure" menu and set the "Interface port" appropriately as shown below:-



You should also check with your computer support staff that the other settings are correct for the apparatus you are using. In particular you should set the Eye Camera Speed to the correct value for the camera that you are using and you should select the appropriate "System Type" for your equipment.

If that still does not work then ask a member of your computer support staff for assistance.

You should now be able to control your eyetracker as described in the appropriate manual. Make sure that you are familiar with the operation of your eyetracker before you attempt to interface with your cogent PC.

Connecting to your cogent PC

Your cogent PC should be also be connected to the ASL model 5000 control unit using the PC serial port (usually a male 9 pin D-type connector on the back of the PC). This connects to a similar male 9 pin D-type connector on the back of the ASL model 5000 control unit labeled "Serial Out (2)".

The cable wiring between these connectors should be identical to the other cable:-

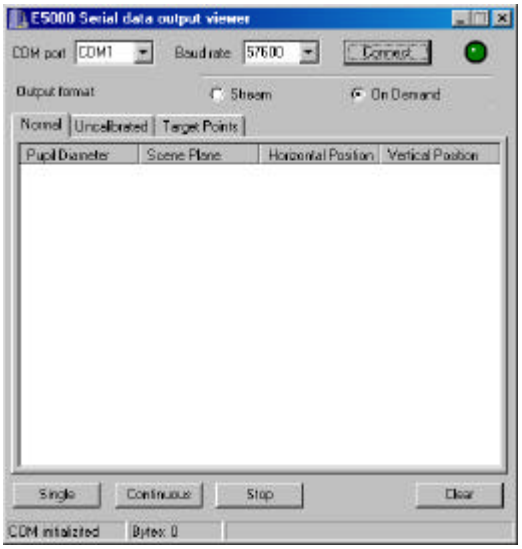
Pin	Pin
2	2
3	3
5	5

The wiring should be exactly as shown above with pins 2, 3 and 5 connected to corresponding pins on both connectors; i.e. no reversal of pins 2 and 3.

Once you have connected your cogent PC to the ASL model 5000 and you have started the **e5Win.exe** program you should check that your cogent PC is connected properly. You should download the file **spv9906.exe** from the ASL website at:-

ftp://asluser:aslftp@a-s-l.com/pub/spviewer/

Copy this file into a new folder and expand it. You should then run the file **Viewer.exe**:-



Click on the "On Demand" button, check that the "COM port" setting is correct and that "Baud rate" is set to 57600 and then click on the "Connect" button. The circle next to it should go green to indicate the connection is working correctly. Then click on the "Continuous" button and you should see a steady stream of numbers in the window. If this does not happen then ask a member of your computer support staff for assistance.

The probable cause is that a setting in the **e5000.cfg** in the eyepos folder on the controller PC is incorrectly set. The line should read:-

serial_data_output_format=1

This sets the ASL model 5000 to send eye data only when requested. If you see the following line:-

serial_data_output_format=129

then the ASL model 5000 will send eye data continuously. Although the cogent interface can deal with this, this mode is undesirable because there is currently a bug with the ASL software; when you perform the subject eye calibration the data mode switches back to "On Demand" mode. This causes the data stream to stop and you have to restart it by going into the "System" item of the "Configuration" menu and clicking on "OK". It is a lot more convenient to operate entirely in "On Demand" mode right from the start.