



IMS F003C 2D graphics occam & C Libraries


**A software support package
for iq systems' graphics TRAMs.**

(IMS DX05B, DX205, DX214 toolset compatible)



INMOS is a member of the SGS-THOMSON Microelectronics Group

Copyright © INMOS Limited 1992. This document may not be copied, in whole or in part, without prior written consent of INMOS.

 [®], **inmos** [®], IMS and occam are trademarks of INMOS Limited.

INMOS is a member of the SGS-THOMSON Microelectronics Group.

Contents

1	Introduction to the IMS F003C	1
1.1	Prerequisites	1
1.1.1	Hardware	1
1.1.2	Software	1
1.2	Organisation of the manual	2
1.2.1	Manual conventions	2
2	Software installation	3
3	Overview of the IMS F003C	5
3.1	CGI display server	5
3.1.1	ANSI C and OCCAM libraries	8
3.1.2	Graphics board support libraries	8
4	CGI concepts	9
4.1	The IMS F003C CGI library	12
4.2	Screens	13
4.3	Colour representation	15
4.4	CGI drawing modes	15
4.4.1	Plot styles	15
4.4.2	Filler modes	16
4.4.3	Pixel replace modes	17
5	Graphics board concepts	19
5.1	Board initialisation	20
5.2	Video memory management	21
5.2.1	Mapping physical CGI screens to VRAM	22
5.3	Colour palette	23
5.4	The iq Systems graphics boards	23
5.4.1	IMS B419 graphics TRAM	24
5.4.2	IMS B437 compact display TRAM	24
6	CGI libraries	25
6.1	Initialisation and termination	25
6.2	Alphabetical list of CGI primitives	26
7	Graphics board functions	95
7.1	List of functions	95

8	ANSI C user guide	103
8.1	Toolset search path	103
8.1.1	IMS F003C library and include files	103
8.2	Invoking the CGI display server	104
8.2.1	Single processor, single program	104
8.2.2	Multiprocessor, multi program	106
8.3	Configuring transputer memory sizes	107
8.4	Opening the graphics board	108
8.5	Compiling and linking IMS F003C programs	109
8.5.1	Compiling	109
8.5.2	Linking	109
8.6	Example program	109
9	occam user guide	111
9.1	Toolset search path	111
9.1.1	IMS F003C library and include files	111
9.2	Invoking the CGI display server	112
9.2.1	Single processor, single program	112
9.2.2	Multiprocessor, multi program	114
9.3	Configuring transputer memory sizes	114
9.4	Opening the graphics board	114
9.5	Compiling and linking IMS F003C programs	116
9.5.1	Compiling	116
9.5.2	Linking	116
9.6	Example program	116
10	Further use of the CGI system	117
10.1	Using and defining text fonts	117
10.2	Using CGI screens for windowing	119
10.3	Simple animation techniques	122
10.4	Writing a board support library	124

Appendices

A	Directory structure	125
B	IMS B419 hardware overview	127
B.1	Description	127
B.1.1	Introduction	127
B.1.2	Screen sizes	128
B.1.3	SubSystem signals	128
B.1.4	Memory Map	129
B.1.5	Pixel clock selection	130
B.1.6	Jumper selection	130
B.2	Board layout	131
B.2.1	Video and sync outputs	132
C	IMS B437 hardware overview	133
C.1	Description	133
C.2	Memory map	134
C.3	Display formats	134
C.4	Colour video controller	134
C.5	Control register programming	136
C.6	Hardware cursor	137
C.7	Events	137
C.8	Board control registers	138
10.4.1	Colour mode select register	138
10.4.2	IMS G332 reset register	138
10.4.3	Startup procedure	139
C.9	Video outputs	139
C.10	Board layout	140
C.11	Accessories	140
D	References	141

1 Introduction to the IMS F003C

The IMS F003C is a 2D (two dimensional) graphics package for *iq* Systems graphics board products. It provides functional conformance with a subset of the Computer Graphics Interface (CGI) standard.

Applications can be developed in the C or Occam programming languages for an arbitrary network of transputers. Graphical output is obtained by installing an appropriate *iq* Systems graphics board somewhere in the network and programming it using the IMS F003C software package.

The IMS F003C is compatible with INMOS software development toolsets. Developers incorporate IMS F003C with their own application software using an appropriately selected C or Occam toolset.

1.1 Prerequisites

In order to develop applications with the IMS F003C the following environments are required:

1.1.1 Hardware

- IBM PC AT or compatible personal computer
- IMS B008 IBM PC AT TRAM motherboard
- IMS B419 graphics TRAM

OR

- IMS B437 compact display TRAM

1.1.2 Software

- IMS D7214 ANSI C Toolset for IBM PC AT

OR

- IMS D7205 Occam Toolset for IBM PC AT

1.2 Organisation of the manual

This manual is split into ten Chapters and four appendices.

Chapter 2 provides step by step instructions for installing the IMS F003C software on an IBM PC AT or compatible computer.

Chapter 3 contains an overview of the software components contained in the IMS F003C package and describes some potential development environments.

A detailed description of CGI concepts is provided in Chapter 4. Readers familiar with 2D computer graphics systems may choose to overlook this Chapter.

Chapter 5 contains a detailed explanation of graphics board concepts with particular reference to *iq* Systems graphics board products. Again, readers familiar with these concepts may wish to overlook this Chapter.

An alphabetical description of the CGI library and graphics board utility functions can be found in Chapters 6 and 7.

Chapter 8 describes how to develop software using IMS F003C in conjunction with an ANSI C toolset. OCCAM toolset users should instead read Chapter 9, which contains an equivalent OCCAM user guide. Both Chapters also contain annotated example source code and instructions for compiling and executing examples included on the installation disks.

The final Chapter looks into a number of more advanced topics. For example, an explanation of the text font format and a description of multi-frame animation techniques can be found in this Chapter.

Engineering data for the *iq* Systems graphics board products supported by IMS F003C can be found in the appendices. Memory and register address maps are provided together with more detailed hardware information.

1.2.1 Manual conventions

Throughout this manual, reference to software routines and constants will be made using ANSI C syntax. Equivalent OCCAM names may be derived by substituting occurrences of the '_' (underscore) character with a '.' (period) character as appropriate.

Source code fragments and operating system command lines will be printed in a teletype style font.

2 Software installation

The installation of IMS F003C requires at least 2Mbytes of free disk space be available on the host computer system hard disk.

IMS F003C is distributed on two 1.2Mbyte 5 1/4" floppy disks or on two 720K byte 3 1/2" diskettes. The disks can be found in a transparent wallet at the rear of the manual. Select the appropriate disks for your system.

To install IMS F003C from floppy disk drive **A:** onto hard disk drive **C:** of an IBM PC AT or compatible computer proceed as follows:

- 1 Insert the disk labelled **DISK 1 of 2**, into disk drive **A:**
- 2 Change current working directory to **C:**.
- 3 At the operating system command prompt, type **a:install a c.**
- 4 Respond as appropriate to prompts made by the install program.
- 5 Insert the second disk (labelled **DISK 2 of 2**) when prompted.

The installation procedure will create and install IMS F003C files under the directory **C:\F003C**. See Appendix A for details of the directory structure and a list of the files that should be present after installation.

The file **C:\F003C\INSTALL2.BAT** may be deleted after installation.

3 Overview of the IMS F003C

The IMS F003C software package consists of the following components:

- CGI display server
- ANSI C and occam interface libraries
- Include files
- *iq* Systems graphics board support libraries
- Source code of the board support libraries
- Example source code

3.1 CGI display server

The CGI display server is a process that runs in parallel with application software and provides access to a graphics board. It is responsible for programming the graphics board hardware and for performing CGI operations when requested by an application program. Graphical output is displayed on an output monitor connected to the graphics board. The CGI display server may be configured to run on any of the *iq* Systems graphics boards, it is linked with a board specific library that provides it a device independent interface to the hardware.

The CGI display server may run in parallel with application software on the same transputer (the graphics board), or with the application running on an adjacent transputer network or a mixture of the two. This arrangement is shown in the following diagrams, where a mixture of TRAM motherboards, general purpose compute TRAMs and graphics TRAMs are used to build various transputer systems capable of generating graphical output via an attached monitor.

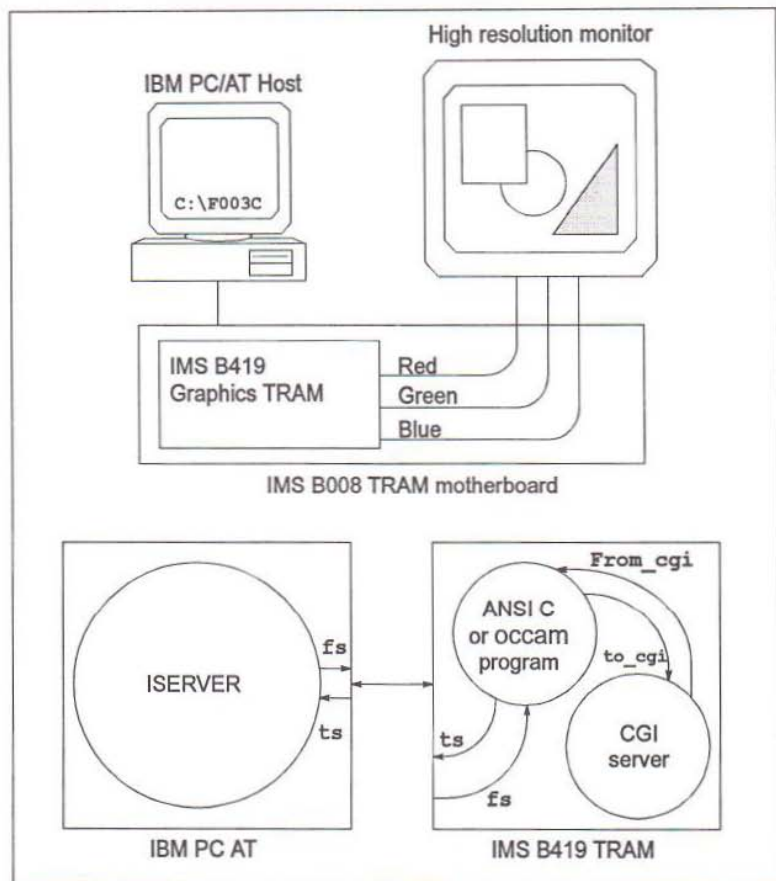


Figure 3.1

This shows an IMS B008 TRAM motherboard and an IMS B419 graphics TRAM. The graphics TRAM is connected to a high resolution monitor. The application program is hosted by an *ISERVER* running on an IBM PC AT development host and consists of a number of processes running in parallel with the CGI display server.

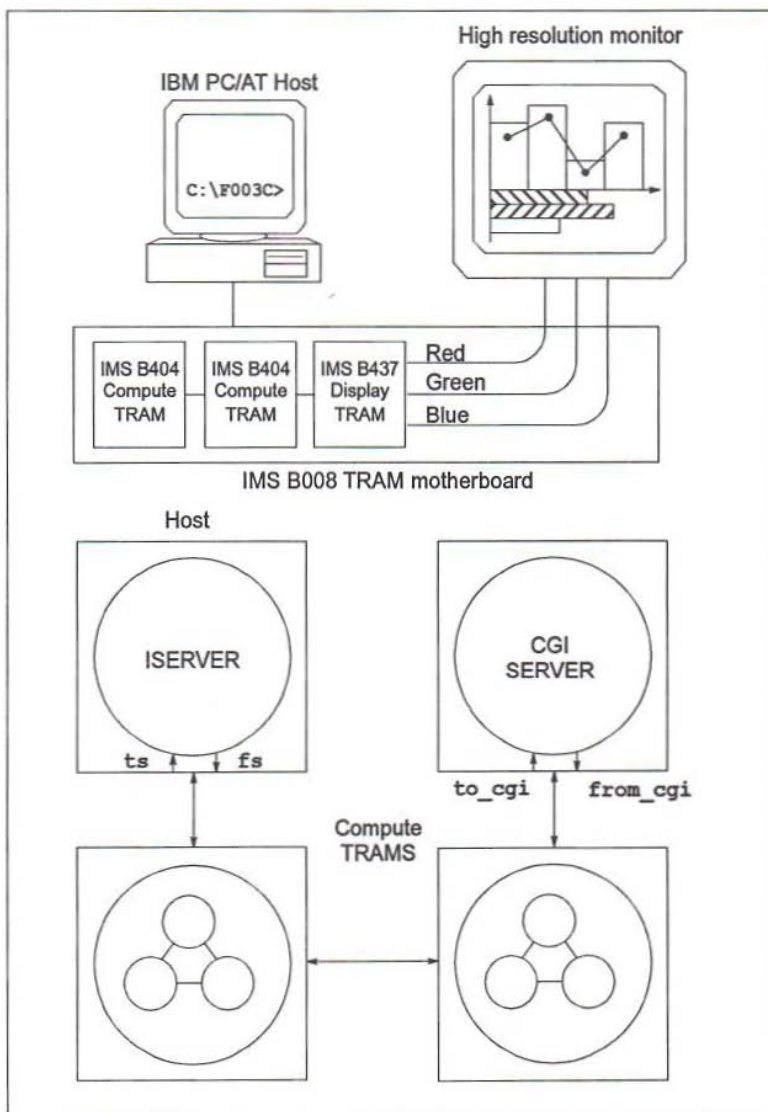


Figure 3.2

This also shows an IMS B008 TRAM motherboard. It is configured with two general purpose compute TRAMs and an IMS B437 compact display TRAM. Again, the

graphics TRAM is connected to a high resolution monitor. The CGI server runs by itself on the graphics TRAM while the application program runs in parallel on the other compute TRAMs. An `ISERVER` runs on the host.

3.1.1 ANSI C and occam libraries

The IMS F003C ANSI C and occam interface libraries contain an equivalent set of procedures. They all communicate with the CGI display server over a pair of transputer channels connected between the server and application software. The programmer's interface to the CGI system is defined by these interface libraries.

The libraries contain two sets of procedures. Those prefixed by `cgi_` belong to the family of Computer Graphics Interface primitives. There are a large number of these and collectively they define a two dimensional graphics package that is functionally conformant with a subset of the CGI standard. The CGI primitives are all device independent: they require no knowledge of the underlying graphics board architecture.

The other set of procedures are prefixed by `gs_`. These implement a device independent interface to the graphics hardware. The same procedures are used to program the graphics hardware regardless of the actual graphics board being used. The CGI display server programs the hardware correctly by calling elements of a board specific support library which is selected according to the graphics board present.

3.1.2 Graphics board support libraries

The device independent interface to the graphics hardware provided by the CGI server is implemented by a number of device dependent board support libraries. Libraries are supplied for a wide range of the *iq* Systems graphics board products, the appropriate one is linked with the CGI display server when building an application.

Monitor resolution and timing characteristics are completely programmable and the libraries also provide device independent colour palette setup and video memory management.

The board support libraries are supplied in source code form. If required, a variant for some other transputer based graphics board can be created by porting the source provided. This is described in Chapter 10.

4 CGI concepts

IMS F003C provides a functionally conformant subset of the Computer Graphics Interface standard (ISO TC97/SC21 N1179). The standard defines the interface between the device independent and device dependent parts of a two dimensional (2D) graphics system. IMS F003C implements the CGI graphical primitive functions, attribute functions and miscellaneous initialisation and error logging primitives.

CGI defines the functional behaviour of a number of graphical output primitives and attribute functions, in a way which is encoding and binding independent. This allows the same facilities to be provided in different languages while taking into account the syntax of that language. IMS F003C provides such bindings for the ANSI C and occam programming languages.

CGI graphical primitive functions are those functions that define the geometric components of a picture. The graphics primitive functions defined in the CGI standard fall into one of the following categories:

- Line
- Marker
- Text
- Filled area
- Image
- Generalised drawing primitive (GDP)

CGI attribute functions determine the appearance of the graphical primitive functions. Attributes are either individual or 'bundleable'. This means that either an attribute must be applied individually or that it may be combined with others, and then applied.

Readers seeking further information on the CGI standard should consult document: ISO TC97/SC21 N1179. The following tables show how the various CGI primitives are implemented by the IMS F003C libraries.

Line functions	
Polyline	<code>cgi_polyline</code>
Disjoint Polyline	<code>cgi_disjpolyline</code>
Circular Arc Centre	<code>cgi_arc</code>
Elliptical Arc	<code>cgi_arc</code>
Marker function	
Poly Marker	<code>cgi_dot, cgi_copy</code>
Text functions	
Text	<code>cgi_text, cgi_sptext</code>
Append Text	<code>cgi_addtext, cgi_addsptext</code>
Restricted Text	<code>cgi_text, cgi_sptext, cgi_chrbegin, cgi_chrspace</code>
Filled area functions	
Polygon	<code>cgi_polygon, cgi_paint</code>
Polygon Set	<code>cgi_polyline, cgi_disjpolyline, cgi_line, cgi_ftrap</code>
Rectangle	<code>cgi_rect, cgi_frect</code>
Circle	<code>cgi_circle, cgi_fcircle</code>
Circular Arc 3 Point Close	<code>cgi_arcc, cgi_strokearc, cgi_fanfill</code>
Circular Arc Centre Close	<code>cgi_arcc, cgi_strokearc, cgi_fanfill</code>
Ellipse	<code>cgi_circle, cgi_fcircle</code>
Elliptical Arc Close	<code>cgi_arcc, cgi_strokearc, cgi_fanfill</code>
Image function	
Cell Array	<code>cgi_frect, cgi_ftrap, cgi_copy</code>

Table 4.1 CGI graphical primitives vs. IMS F003C

Line attributes Line Type Line Width Line Colour	<code>cgi_setlinestyle,</code> <code>cgi_setdrawmode</code>
Marker attributes Marker Type Marker Size Marker Colour	<code>cgi_setpelstyle, cgi_copy,</code> <code>cgi_zoom</code>
Text attributes Text Font Index Text Precision Character Expansion Factor Character Spacing Text Colour Character Height Character Orientation Character Set Index	<code>cgi_setfont</code> <code>cgi_text, cgi_chrz, cgi_zoom</code> <code>cgi_chrz, cgi_zoom</code> <code>cgi_chrspace</code> <code>cgi_setfcol</code> <code>cgi_chrz</code> <code>cgi_setorient</code> <code>cgi_setfont</code>
Filled area attributes Interior Style Fill Colour Hatch Index Pattern Index Pattern Table Pattern Size	<code>cgi_setfillstyle, cgi_setfcol</code>

Table 4.2 CGI attribute primitives vs. IMS F003C

4.1 The IMS F003C CGI library

The IMS F003C CGI functions are supplied in an INMOS TCOFF compatible object library called `CGILIB.LIB`. Two variants are provided, one for ANSI C, the other for `occam`. The following lists summarise the CGI functions available:

Line functions

<code>cgi_line</code>	Straight line
<code>cgi_rect</code>	Rectangle outline
<code>cgi_polyline</code>	Consecutive line segments
<code>cgi_disjpolyline</code>	Straight line segments
<code>cgi_polygon</code>	Polygon outline
<code>cgi_circle</code>	Ellipsoid outline
<code>cgi_arc</code>	Partial ellipsoid outline
<code>cgi_arcc</code>	Closed partial ellipsoid outline
<code>cgi_strokearc</code>	Stroke ellipsoid outline
<code>cgi_dot</code>	Single point
<code>cgi_setlinestyle</code>	Setup custom line style

Text functions

<code>cgi_text</code>	Print text at position
<code>cgi_addtext</code>	Add text at current position
<code>cgi_sptext</code>	Print text with spacing control
<code>cgi_addsptext</code>	Add text with spacing control
<code>cgi_chrbegin</code>	Set character position
<code>cgi_chrspace</code>	Set character spacing
<code>cgi_chrz</code>	Print character with scaling
<code>cgi_setfont</code>	Setup character font

Filled area functions

<code>cgi_cls</code>	Clear screen
<code>cgi_frect</code>	Filled rectangle
<code>cgi_fcircle</code>	Filled ellipsoid
<code>cgi_fanfill</code>	Filled partial ellipsoid
<code>cgi_paint</code>	Area flood fill
<code>cgi_ftrap</code>	Filled trapezoid
<code>cgi_fhline</code>	Filled horizontal lines
<code>cgi_setfillstyle</code>	Set custom fill pattern

Image functions

<code>cgi_copy</code>	Image copy
<code>cgi_zoom</code>	Image copy with zoom
<code>cgi_rot</code>	Image rotation
<code>cgi_shear</code>	Image shear
<code>cgi_search</code>	Search for colour change
<code>cgi_setpelstyle</code>	Set custom pel pattern

Control functions

<code>cgi_init</code>	Initialise CGI system
<code>cgi_terminate</code>	Terminate CGI system
<code>cgi_setdrawmode</code>	Set drawing modes
<code>cgi_setdrawscreen</code>	Set current CGI screen
<code>cgi_setorient</code>	Set text and image orientation

Error handling

<code>cgi_errstat</code>	Expound current CGI error
--------------------------	---------------------------

4.2 Screens

All CGI operations are performed on an abstract data structure called a screen. A screen represents a bounded two dimensional area that contains the graphical output of CGI functions. Cartesian coordinates are used to address points located on a screen and all CGI operations, when applied to a screen, are clipped to its extent. The CGI system uses the screen abstraction to represent various types of graphical object. For example, screens are used to hold character images when expanded from a packed font.

In IMS F003C, the ANSI C and occam implementations of a screen are defined as follows:

ANSI C struct	occam INT array
<pre>struct { char *raster; int xsize; int ysize; int stride; int multiMode; } screen;</pre>	<pre>[SCREEN.SIZE]INT screen: screen[SCREEN.RASTER] screen[SCREEN.XSIZE] screen[SCREEN.YSIZE] screen[SCREEN.STRIDE] screen[SCREEN.MULTIMODE]</pre>

raster is the transputer address of a region of memory used to hold a two dimensional image, called a raster. It is **xsize** pixels wide by **ysize** pixels high. The **stride** value specifies the horizontal stride to take when stepping to an equivalent position on the next horizontal line. **multiMode** is used internally by the CGI system.

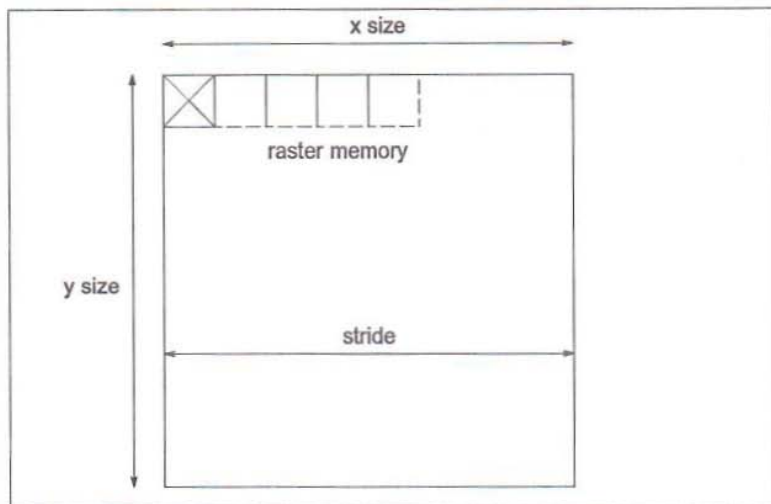


Figure 4.1 The CGI screen abstraction

The CGI system maintains the notion of a current drawing screen. This is a screen that has been identified as a target for future CGI operations: the majority of the CGI functions implicitly address the current drawing screen. It is assigned with `cgi_setdrawscreen`.

Any number of screens may exist in a system and some may be related to others. For example, to build a windowing system one could use the screen abstraction to represent the hierarchies that exist between parent windows and their child sub-windows. The only restriction concerning the use of the screen abstraction is that the memory associated with a screen must be located on the transputer running the CGI display server.

A screen may be displayed on an output monitor if its horizontal and vertical dimensions match the physical display resolution. Such a screen is referred to as a physical screen. Physical screens are usually implemented with video memory on the graphics board. When displayed on a monitor their cartesian origin (0,0) is located at the top left hand corner of the display.

CGI screens can be allocated statically, or dynamically, on the transputer that runs the CGI display server. A new screen may be derived from an existing one by referencing a sub-area of the existing screen's raster memory. Physical screens are al-

located dynamically using `fs_initscreen` and have their rasters stored in video memory on the graphics board.

4.3 Colour representation

The IMS F003C implementation of CGI uses 8 bit pixels resulting in screens containing up to 256 different colours. Pixel values are used to address a colour palette which generates the actual display colour from a possibly larger range. The resolution of the colour palette is graphics board dependent, see section 5.3 for specific details of this.

4.4 CGI drawing modes

The CGI system may operate in a number of different drawing modes that define the run-time behaviour of graphical primitives. Drawing modes are concerned with the following:

- Plot style
- Filler mode
- Pixel replace mode

Ultimately, most graphical primitives are implemented by plotting a sequence of pixels. The pixel replace mode defines how a pixel is written into screen memory. The plot style is used to control the generation of pixel values, for example, when drawing a line, and the fill mode relates to the different methods available for performing area flood fill.

The CGI system applies the current plot style, fill and pixel replace modes implicitly, during normal operation. They may be initialised with `cgi_setdrawmode` and depending on the CGI function may combine to produce a resultant visual effect. In other situations only a subset may have an impact.

4.4.1 Plot styles

Plot styles affect the outcome of the CGI plotting and outline functions, such as `cgi_dot`, `cgi_circle` or `cgi_polyline`.

When tracing the outline of an object, or when plotting a sequence of straight lines, the current plot style determines the size, shape and visibility of every point plotted. There are five plot styles:

- PIXEL
- PEL
- LINSTYLE

- **LINestyle-TRANSPARENT**
- **LINestyle-PEL**

PIXEL

A single pixel is plotted to represent each point. This gives solid outlines of minimum display thickness drawn in the current foreground colour. See `cgi_setfcol`.

PEL

Each point is represented by a small, two dimensional pattern, called a pel. The pel pattern is established with `cgi_setpelstyle` and used whenever a point would otherwise have been plotted. Pels are useful for repeatedly plotting customised shapes such as cursors or bullet marks.

The pixel values defined by a pel pattern determine its colour. In the default pixel replace mode (overwrite) only pixels which have non-zero values are plotted. This means that if the pel background colour is always zero, then the foreground can consist of any number of non-zero colours, all of which will be plotted normally. By selecting an appropriate pixel replace mode the zero-valued background can be plotted, or the foreground ignored.

LINestyle

A line style is a one dimensional array of pixel values that is used to determine the value of consecutive points on a line. The CGI system keeps track of which pixel value to use for the next point and cycles repeatedly through the pixel array assigning values to new points. A pixel value can be used a variable number of times before moving on to the next value, this is controlled by the line style zoom factor and achieves a stretch effect. Line styles are initialised with `cgi_setlinestyle` which defines the pixel array contents, and its zoom factor.

LINestyle-TRANSPARENT

This is a variant of the line style. A transparency effect is achieved by only plotting points that have non-zero values as defined by the line style array. All other points are plotted normally. Zero valued pixels define positions where background colours will be visible through the line style.

LINestyle-PEL

Another variant of the line style mode, this combines a line style with a pel pattern. Non-zero valued points defined by the line style are replaced by the pel pattern.

4.4.2 Filler modes

The CGI area fill primitives operate according to the current fill mode. This defines the method for filling areas created by functions such as `cgi_frext` or `cgi_fanfill`. There are two fill modes:

- SOLID
- PATTERN

SOLID

Areas are filled with a solid colour. The colour is defined by the current foreground colour, see `cgi_setfcol`.

PATTERN

A customised two dimensional pattern called a fill style is used. This is tiled over the fill area and clipped to it's boundary. The current fill style is initialised with `cgi_setfillstyle` to define the pixel values contained in the fill style pattern. By selecting a suitable pixel replace mode, zero valued pixels may be treated specially if required, otherwise they are written to the current screen along with the non-zero valued pixels.

4.4.3 Pixel replace modes

The pixel replace modes define how pixels are ultimately written into screen memory. They are fundamental to the operation of the CGI system: the result of every CGI primitive in conjunction with the higher level drawing modes is influenced by the current pixel replace mode. There are three types of pixel replace mode:

- OVERWRITE
- LOGICAL
- TRANSPUTER

OVERWRITE

The basic replace mode. Screen memory is overwritten with new pixel values.

LOGICAL

The logical replace modes are implemented by performing a read modify write operation on screen memory. An existing pixel is combined, using a logical operator, with the new pixel value and the resultant pixel written into screen memory. The logical modes supported are:

Operator	Result
AND	bitwise AND
OR	bitwise OR
XOR	bitwise XOR
NAND	bitwise NAND
NOR	bitwise NOR

TRANSPUTER

The transputer replace modes correspond directly to the two dimensional block move instructions supported by the transputer. They are:

Operator	Result
MOVE2DALL	block copy
MOVE2DZERO	zero block copy
MOVE2DNONZERO	non-zero block copy

5 Graphics board concepts

The *iq* Systems graphics board products supported by IMS F003C all have a similar hardware architecture. They all feature a transputer (of some sort), have normal dynamic random access memory for program and data storage and an additional area of special purpose video memory for image output to a graphics monitor. All the boards have a colour video controller (CVC) chip capable of driving a wide range of monitors at different pixel rates and at different display resolutions.

The IMS F003C CGI library contains a number of functions for initialising and controlling the hardware on a graphics board in a *device independent* way. This allows software developed for one graphics board to run on another without changing any source code. The programmer's interface to the graphics board hardware is defined by the following functions:

Function	Description
<code>fs_screenaddr</code>	Return the address of a screen's raster
<code>fs_displaybank</code>	Display a bank of video memory
<code>fs_initscreen</code>	Map a physical screen to video memory
<code>fs_setpalette</code>	Set colour palette entry
<code>fs_openboard</code>	Device independent board open function
<code>fs_closeboard</code>	Device independent board close function
<code>fs_writeregs</code>	Write graphics board registers

These functions cause the CGI display server to call a similar set of functions from a *device dependent* library that achieve an equivalent effect on whatever graphics hardware is actually present. The CGI server is linked against a device dependent library when building an application program for a particular graphics board. Device dependent libraries for the following *iq* Systems graphics board products are provided with IMS F003C:

<i>iq</i> Systems graphics board	IMS F003C library
IMS B419 graphics TRAM	B419.LIB
IMS B419 graphics TRAM with G300A	B419A.LIB
IMS B437 compact display TRAM	B437.LIB

5.1 Board initialisation

In order to use a graphics board an application must first open it with the `fs_open-board` function. This performs a number of operations to initialise the graphics hardware ready for use by the CGI system. The most important of these is the initialisation of the CVC chip. The CVC chip generates a display on an output monitor and must be programmed with a number of video timing parameters that specify the format and timing of signals used to control the monitor. Usually, this will depend on the desired display resolution and the timing characteristics of the chosen monitor.

The CVC is programmed with the contents of a data structure called the video timing generator (abbreviated VTG) parameter block. This contains a number of values that define elements of the video signals used to drive a monitor. The parameters are directly applicable to a range of CVC devices manufactured by INMOS and used on the *iq* Systems graphics boards. The ANSI C and occam definitions of the VTG parameter block are:

ANSI C struct	occam INT array
<pre>struct { int pll; int line_time; int half_sync; int back_porch; int display; int short_display; int v_display; int v_blank; int v_sync; int v_preequalise; int v_postequalise; int broad_pulse; int mem_init; int transfer_delay; int mask_register; int control; } vtg;</pre>	<pre>[VTG.SIZE]INT vtg: vtg[VTG.PLL] vtg[VTG.LINE.TIME] vtg[VTG.HALF.SYNC] vtg[VTG.BACK.PORCH] vtg[VTG.DISPLAY] vtg[VTG.SHORT.DISPLAY] vtg[VTG.V.DISPLAY] vtg[VTG.V.BLANK] vtg[VTG.V.SYNC] vtg[VTG.V.PREEQUALISE] vtg[VTG.V.POSTEQUALISE] vtg[VTG.BROAD.PULSE] vtg[VTG.MEM.INIT] vtg[VTG.TRANSFER.DELAY] vtg[VTG.MASK.REGISTER] vtg[VTG.CONTROL]</pre>

ANSI C and occam header files are supplied that define a number of constant VTG parameter blocks suitable for controlling a range of monitors at a number of commonly used display resolutions. In most situations, a parameter block that

matches the requirements of a particular application can be selected from the header file and used verbatim. If a suitable parameter block can't be found, or if special requirements dictate the use of other timing parameters, then consult *The graphics databook* [5].

This contains technical information about INMOS CVC devices. It includes an in depth discussion of video timing mechanisms and how to calculate video timing parameters. The reader should also consult the appendices, which contain engineering data for the *iq* Systems graphics board products supported by IMS F003C.

5.2 Video memory management

Transputers used on the *iq* Systems graphics boards have a linear address space. Within this space lies a region of normal dynamic random access memory (DRAM) and a region of special video memory (VRAM). The size and location of these memory areas is dependent on the architecture of the graphics board. The DRAM is always located at the bottom of the transputer address map (most negative address end) and is used for program and data storage. The VRAM is located elsewhere, usually at higher addresses, and is used as raster memory to store the output of graphical operations. A monitor display is produced by the CVC which reads the VRAM continuously to generate the appropriate output signals.

On some graphics boards the two memory areas are separate, on others they may be configured (with a jumper) to be either contiguous or non-contiguous. Spare video memory can be used for additional program and data storage, but only if it is contiguous with existing DRAM. Other boards have no normal DRAM at all and use VRAM for program, data and raster storage. The amount of VRAM required to generate output on a monitor is directly related to the monitor resolution. Since the CVC hardware allows this to be configured at run-time the available VRAM can serve a number of purposes:

- Depending on the amount of VRAM present it can be used to store a number of monitor sized rasters. The graphics hardware is programmed to display one of these rasters but can be switched, at any time, to display another.
- If the VRAM is contiguous with DRAM then part of it may be allocated to program storage effectively extending the amount of the DRAM available.

Video memory is managed by dividing it up into a number of equal sized regions, called banks. The size of a bank is determined by the display resolution and matches exactly the amount of raster memory needed to generate an output image at that resolution. The total number of video banks available on a particular graphics board therefore depends on two factors: the amount of VRAM present and the configured display resolution.

The memory architecture of a typical graphics board is shown in the diagram below:

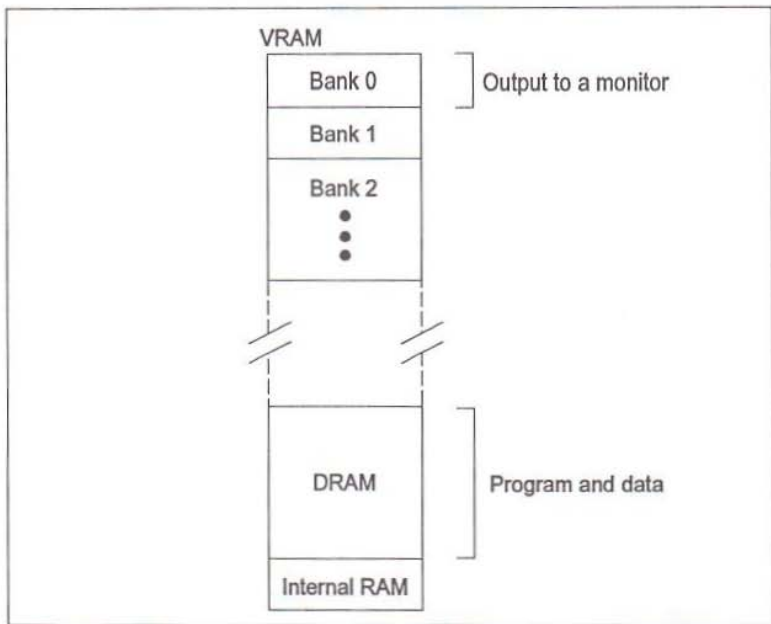


Figure 5.1 Memory architecture of a graphics board

Note that video memory banks are allocated from the top of video memory toward lower memory addresses. In the diagram, bank number 0 is positioned at the top most part of VRAM. Other banks are located at ever decreasing addresses beneath this. If the VRAM and DRAM are contiguous it is possible to extend the memory available for program and data storage by using up spare VRAM banks near the bottom. This is achieved by configuring an application program with a memory size that includes any spare VRAM banks nominated for program use. It is the programmer's responsibility to ensure that such VRAM banks will never be used for any other purpose.

5.2.1 Mapping physical CGI screens to VRAM

Physical CGI screens have their raster memory allocated from VRAM by initialising a screen data structure to reference a video memory bank. This is done with `fs_initscreen`, given the number of the video memory bank to use for the screen's raster it returns a screen structure.

There need be no correspondence between the current CGI drawing screen and the physical screen displayed on a monitor. Both can be selected independently.

A screen is made visible by programming the graphics hardware to output the corresponding video memory bank. `fs_displaybank` does this.

In the example below, a new screen is allocated and mapped to video memory bank 0. It is made the current CGI drawing screen and its video memory bank displayed on an output monitor. This has the effect of causing the CGI system to display all subsequent operations instantaneously, on the monitor.

```
{
  screen s;

  /* Allocate a physical screen and map it to bank 0 */

  fs_initscreen( from_cgi, to_cgi, &s, 0 );

  /* The screen is s.xsize pixels wide by s.ysize pixels
     high. These values correspond to the monitor resolution
     which is fixed at startup time with fs_openboard() */

  /* Assign the screen to the current draw screen */

  cgi_setdrawscreen( to_cgi, s );

  /* Display the screen on an output monitor */

  fs_displaybank( to_cgi, 0 ); /* Output video bank 0 */

  /* Do lots of drawing with the CGI functions ... */
}
```

5.3 Colour palette

The INMOS colour video controller chips used on all the *iq* Systems graphics board products generate colour displays with a programmable colour look-up table called a palette. This provides a mapping between pixel values and the actual colour generated on an output monitor. Colour values are described by three numbers that specify the red, green and blue components of the colour. For a given pixel value, the output colour is programmed with `fs_setpalette` by specifying what the red, green and blue colour components should be.

The colour components have an 8 bit resolution. When combined, they describe a colour from a 24 bit colour space that supports a palette of up to 16 million different colours. Because the IMS F003C CGI system manipulates 8 bit pixel values the colour palette can contains up to 256 different colours selected from the 16 million possible.

5.4 The *iq* Systems graphics boards

This section describes some specific features of *iq* Systems graphics boards that should be considered before using them. More detailed engineering data, on each, can be found in the appendices.

5.4.1 IMS B419 graphics TRAM

The IMS B419 has 2M bytes of DRAM and 2M bytes of VRAM. There is enough video memory to support display resolutions of up to 1280 by 1024 pixels with some left over. (Note that display resolutions any larger than this are not possible because of the very high pixel data rates required).

There are two variants of the IMS B419. The older has an IMS G300A CVC fitted, current production versions use the IMS G300B. The corresponding board support libraries are: **B419A.LIB** and **B419.LIB**.

The IMS B419 must be configured to make its DRAM and VRAM contiguous. This is a jumper option on the board and is described in Appendix B. Making the memory areas contiguous offers the possibility to extend program and data space into VRAM as previously described. Note that the board support libraries will not function correctly unless this is done.

5.4.2 IMS B437 compact display TRAM

The IMS B437 has 1M byte of VRAM and no DRAM. Because of the limited memory available a trade off situation must be reached to satisfy the requirements of program storage and the desired monitor resolution. For example, one screen with a resolution of 1024 by 768 pixels would leave approximately 256K bytes of memory left for program storage. Typically, the IMS B437 is used in configurations where only the CGI server runs on the IMS B437 and application software runs elsewhere in the transputer network.

The board support library for the IMS B437 is: **B437.LIB**.

The special purpose times one pixel clock frequencies available on the IMS B437 are not used by the board support library. For more information on these features see the Appendix C.

6 CGI libraries

6.1 Initialisation and termination

6.1.1 cgi_init

Initialise the CGI server.

C:

```
void cgi_init( Channel *to_cgi )
```

occam:

```
PROC cgi.init( CHAN OF ANY to.cgi )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server

Description:

cgi_init initialises the CGI system to the following state:

- No current text font
- No current pel, fill or line style patterns
- Pixel mode **FM_COL**
- Replace mode **RM_COL**
- Fill mode **FM_COL**

6.1.2 cgi_terminate

Terminate the CGI display server.

C:

```
void cgi_terminate( Channel *from_cgi, Channel *to_cgi )
```

occam:

```
PROC cgi.terminate( CHAN OF ANY from.cgi, to.cgi )
```

Parameters:

Parameter	Comment
from_cgi	Channel from CGI server
to_cgi	Channel to CGI server

Description:

cgi_terminate terminates the CGI display server.

6.2 Alphabetical list of CGI primitives

6.2.1 cgi_addsptext

Append text at current character position, with spacing control.

C:

```
void cgi_addsptext(
    Channel *to_cgi,
    int n, char *str,
    int *dx, int *dy )
```

occam:

```
PROC cgi.addsptext(
    CHAN OF ANY to.cgi,
    VAL INT n,
    VAL [ ]BYTE str,
    VAL [ ]INT dx, dy )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
n	Number of characters to plot
str	Character string
dx	X axis character spacing distances
dy	Y axis character spacing distances

Description:

`cgi_addsptext` plots `n` characters from the character string `str` according to the current font description. The first character is plotted at the current character position which is then incremented by X and Y axis offsets specified by the inter-character spacing vectors `dx` and `dy`, for the character. Subsequent characters are plotted in the same manner, using the next pair of spacing distances. The current character position after the operation completes is offset from the first character plotted by X and Y axis distances equal to the sum of the `dx` and `dy` spacing vectors respectively.

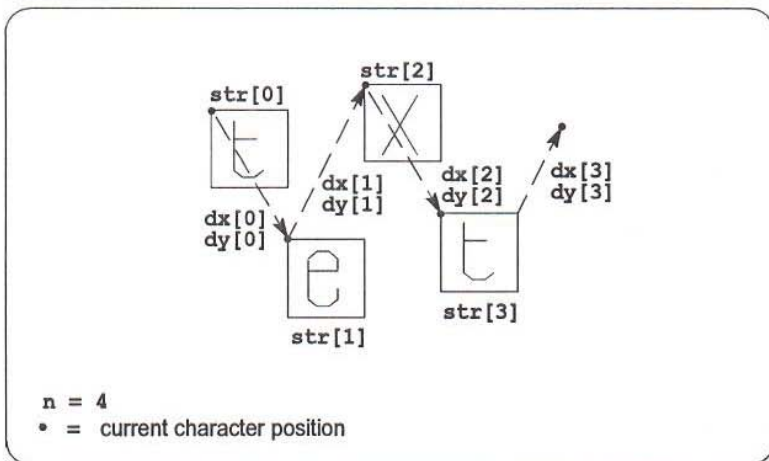
The spacing vectors should be set with respect to the current orientation, see `cgi_setorient`. Characters are plotted according to the current pixel replace mode, see `cgi_setdrawmode`.

Characters are reproduced at the size of their font, which should be initialised, see `cgi_setfont`. Each pixel of every character plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

For text display, the default pixel replace mode `RM_COL`, will cause characters to imprint within a rectangular bounding box of colour 0. In some cases this will not produce the desired effect. If only the foreground of the text is required and a pixel overwrite mode rather than a logical operation is desired then select pixel replace mode `RM_NZ`. This will cause only those pixels which are non-zero to be plotted.

Diagram:

Current screen



6.2.2 `cgi_addtext`

Append text at current character position.

C:

```
void cgi_addtext(  
    Channel *to_cgi,  
    int n, char *str )
```

occam:

```
PROC cgi.addtext(  
    CHAN OF ANY to.cgi,  
    VAL INT n,  
    VAL []BYTE str )
```

Parameters:

Parameter	Comment
<code>to_cgi</code>	Channel to CGI server
<code>n</code>	Number of characters to plot
<code>str</code>	Character string

Description:

`cgi_addtext` plots `n` characters from the character string `str` according to the current font description. Characters are plotted at the current character position which is then incremented by the currently defined X and Y axis inter-character spacing distances, see `cgi_chrspace`. The current character position after the operation completes is offset from the last character plotted by these distances.

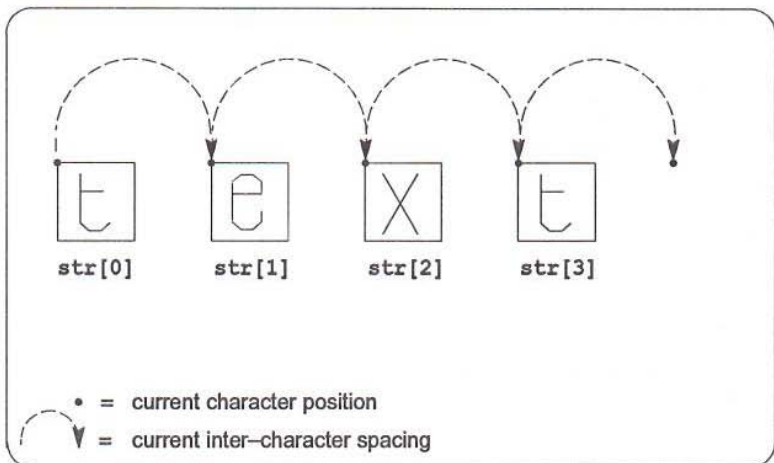
Characters are plotted according to the current pixel replace mode, see `cgi_setdrawmode` and the current orientation, see `cgi_setorient`.

Characters are reproduced at the size of their font which should be initialised, see `cgi_setfont`. Each pixel of every character plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

For text display, the default pixel replace mode `RM_COL`, will cause characters to imprint within a rectangular bounding box of colour 0. In some cases this will not produce the desired effect. If only the foreground of the text is required and a pixel overwrite mode rather than a logical operation is desired then select pixel replace mode `RM_NZ`. This will cause only those pixels which are non-zero to be plotted.

Diagram:

Current screen



6.2.3 cgi_arc

Outline part of an axis aligned ellipsoid.

C:

```
void cgi_arc(
    Channel *to_cgi,
    int Xc, int Yc, int A, int B,
    int DXs, int DYs, int DXe, int DYe )
```

occam:

```
PROC cgi_arc(
    CHAN OF ANY to_cgi,
    VAL INT Xc, Yc, A, B, DXs, DYs, DXe, DYe )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(Xc, Yc)	Centre coordinate
A	Length of X direction semi axis
B	Length of Y direction semi axis
(DXs, DYs)	Start vector
(DXe, DYe)	End vector

Description:

`cgi_arc` plots part of the outline of an axis aligned ellipsoid centred at (Xc, Yc) and with semi-axis lengths of **A** and **B** pixels. Both **A** and **B** must be positive, the larger of the two values is the semi-major axis length, while the lesser specifies the semi-minor axis length.

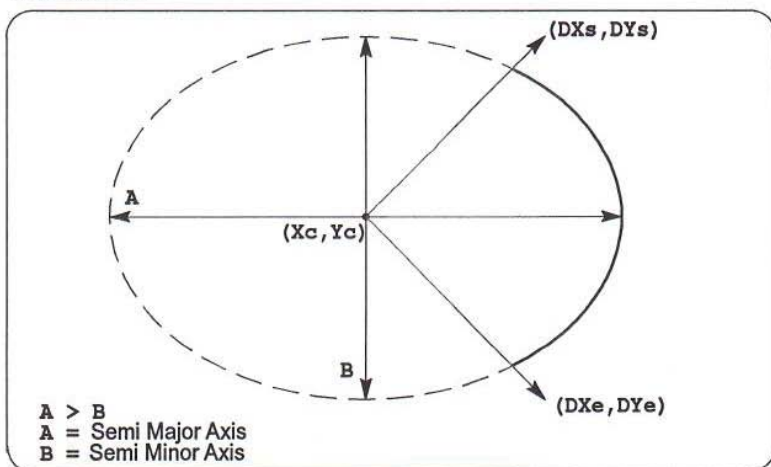
(DXs, DYs) and (DXe, DYe) define direction vectors emanating from the centre of the ellipse that specify which part of its outline to draw. Only points clockwise of the (DXs, DYs) vector and anti clockwise of (DXe, DYe) are plotted.

The outline is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and plot modes affect the appearance of the outline, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.4 cgi_arcc

Outline part of an axis aligned ellipsoid, closed with chord or segment lines.

C:

```
void cgi_arcc(
    Channel *to_cgi,
    int Xc, int Yc, int A, int B,
    int DXs, int DYs, int DXe, int DYe,
    int CloseMode )
```

occam:

```
PROC cgi.arcc(
    CHAN OF ANY to.cgi,
    VAL INT Xc, Yc, A, B, DXs, DYs, DXe, DYe,
    CloseMode )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(Xc, Yc)	Centre coordinate
A	Length of X direction semi axis
B	Length of Y direction semi axis
(DXs, DYs)	Start vector
(DXe, DYe)	End vector
CloseMode	Close mode

Description:

`cgi_arcc` plots part of the outline of an axis aligned ellipsoid centred at (X_c, Y_c) and with semi-axis lengths of A and B pixels. Both A and B must be positive, the larger of the two values is the semi-major axis length, while the lesser specifies the semi-minor axis length.

(DX_s, DY_s) and (DX_e, DY_e) define direction vectors emanating from the centre of the ellipse that specify which part of its outline to draw. Only points clockwise of the (DX_s, DY_s) vector and anti clockwise of (DX_e, DY_e) are plotted.

The partial outline is closed with either a single chord line, joining the two end points, or a pair of segment lines, connecting each end point to the centre of the ellipse at (X_c, Y_c) . The value of `CloseMode` determines which method is used, valid values are:

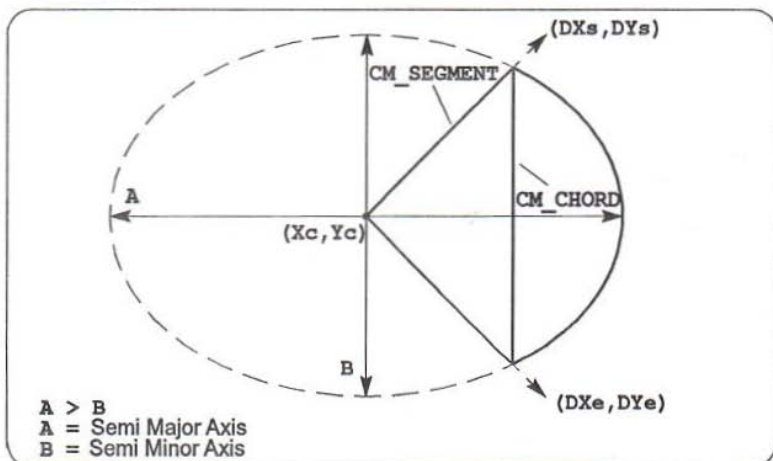
CloseMode	Comment
CM_CHORD	Close outline with a chord line
CM_SEGMENT	Close outline with two segment lines

The outline is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and plot modes affect the appearance of the outline, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.5 cgi_chrbegin

Set current character display position.

C:

```
void cgi_chrbegin(  
    Channel *to_cgi,  
    int X, int Y )
```

occam:

```
PROC cgi_chrbegin(  
    CHAN OF ANY to_cgi,  
    VAL INT X, Y )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(X, Y)	Character position coordinate

Description:

`cgi_chrbegin` sets the current character position to (X, Y). The next text operation will start plotting characters at this position. All text operations, other than `cgi_chrz`, update the current character position as characters are plotted.

Setting the current character position to a location outside the extent of the current screen definition is allowed. However, it should be remembered that all character plotting operations are clipped to the current screen definition, see `cgi_setdrawscreen`.

6.2.6 cgi_chrspace

Set current inter-character spacing.

C:

```
void cgi_chrspace(  
    Channel *to_cgi,  
    int dX, int dY )
```

occam:

```
PROC cgi.chrspace(  
    CHAN OF ANY to.cgi,  
    VAL INT dX, dY )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
dX	X axis character spacing distance
dY	Y axis character spacing distance

Description:

`cgi_chrspace` sets the current inter-character spacing distances. These values are used to increment the current character position, in the X and Y axis directions, after each character is plotted. `dX` specifies the inter-character spacing distance in the X axis direction, `dY` specifies the Y axis distance.

The inter-character spacing is independent of the current orientation and font size, see `cgi_setorient` and `cgi_setfont`.

6.2.7 cgi_chrz

Plot character with zoom scaling.

C:

```
void cgi_chrz(
    Channel *to_cgi,
    char ch,
    int zlenx, int zleny )
```

occam:

```
PROC cgi.chrz(
    CHAN OF ANY to_cgi,
    VAL BYTE ch,
    VAL INT zlenx, zleny )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
ch	Character to plot
zlenx	Width of scaled character on X axis
zleny	Height of scaled character on Y axis

Description:

`cgi_chrz` plots the single character `ch` according to the current font description and with independent scaling in the X and Y axis directions. `zlenx` specifies the width of the character, when plotted, in the X axis direction. The character's height, also when plotted, is given by `zleny` on the Y axis.

Depending on the current font size, see `cgi_setfont`, reduction or enlargement can be achieved independently in the X and Y axis directions by setting `zlenx` and `zleny` appropriately.

The current character position is NOT updated after the character is plotted.

The character is plotted according to the current pixel replace mode, see `cgi_setdrawmode`.

For text display, the default pixel replace mode `RM_COI`, will cause the character to imprint within a rectangular bounding box of colour 0. In some cases this will not produce the desired effect. If only the foreground of the character is required and a pixel overwrite mode rather than a logical operation is desired then select pixel replace mode `RM_NZ`. This will cause only those pixels which are non-zero to be plotted.

Each pixel of the character plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

6.2.8 cgi_circle

Outline an axis aligned ellipsoid.

C:

```
void cgi_circle(  
    Channel *to_cgi,  
    int Xc, int Yc, int A, int B )
```

occam:

```
PROC cgi.circle(  
    CHAN OF ANY to.cgi,  
    VAL INT Xc, Yc, A, B )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(Xc, Yc)	Centre coordinate
A	Length of X direction semi axis
B	Length of Y direction semi axis

Description:

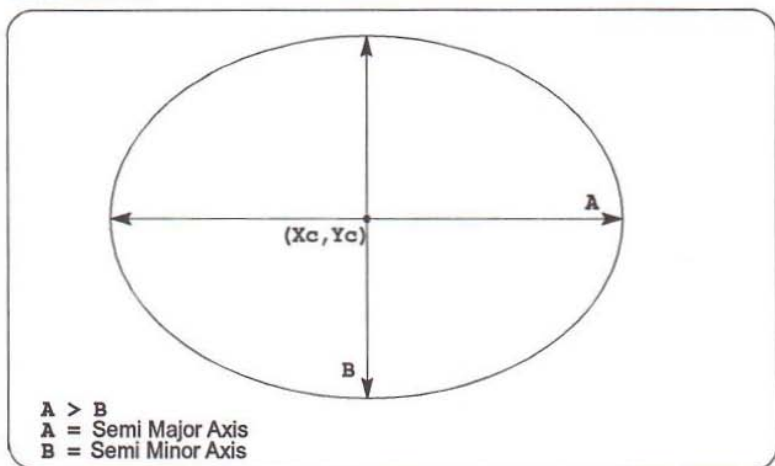
`cgi_circle` plots the outline of an axis aligned ellipsoid centred at (Xc, Yc) and with semi-axis lengths of **A** and **B** pixels. Both **A** and **B** must be positive, the larger of the two values is the semi-major axis length, while the lesser specifies the semi-minor axis length. An outline of a circle is plotted with a diameter equal to either **A** or **B**, if they have identical values.

The outline is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and plot modes affect the appearance of the outline, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.9 cgi_cls

Clear screen.

C:

```
void cgi_cls(  
    Channel *to_cgi,  
    screen s, int colour )
```

occam:

```
PROC cgi.cls(  
    CHAN OF ANY to.cgi,  
    VAL [SCREEN.SIZE]INT s,  
    VAL INT colour )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
s	Screen to clear
colour	Colour

Description:

`cgi_cls` clears the entire raster area associated with the screen `s` to the colour specified by `colour`.

The current fill and pixel replace modes are ignored.

6.2.10 cgi_copy

2D region block copy.

C:

```
void cgi_copy(
    Channel *to_cgi,
    screen s, int Xs, int Ys,
    int DX, int DY,
    screen d, int Xd, int Yd )
```

occam:

```
PROC cgi_copy(
    CHAN OF ANY to_cgi,
    VAL [SCREEN.SIZE]INT s,
    VAL INT Xs, Ys, DX, DY,
    VAL [SCREEN.SIZE]INT d,
    VAL INT Xd, Yd )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
s	Source screen
(Xs, Ys)	Source coordinate
DX	Size of region in X direction
DY	Size of region in Y direction
d	Destination screen
(Xd, Yd)	Destination coordinate

Description:

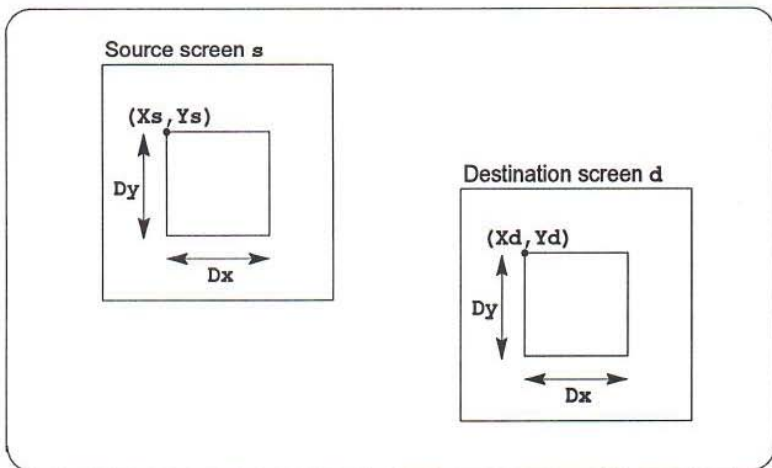
`cgi_copy` copies a rectangular, axis aligned, region from the source screen `s` to the destination screen `d`. The size of the region is specified by `DX` pixels in the X axis direction and `DY` pixels on the Y axis.

The coordinate `(Xs, Ys)` identifies the top left hand corner of the region on the source screen, it is copied to `(Xd, Yd)` on the destination screen.

The region is clipped to the destination screen definition. No scaling is performed.

The current orientation and pixel replace modes affect the resultant display, see `cgi_setorient` and `cgi_setdrawmode`.

Diagram:



6.2.11 cgi_disjpolyline

Draw a sequence of disjoint lines.

C:

```
void cgi_disjpolyline(
    Channel *to_cgi,
    int n, int *points )
```

occam:

```
PROC cgi.disjpolyline(
    CHAN OF ANY to_cgi,
    VAL INT n,
    VAL []INT points )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
n	Number of (X,Y) points
points	Line start and end points

Description:

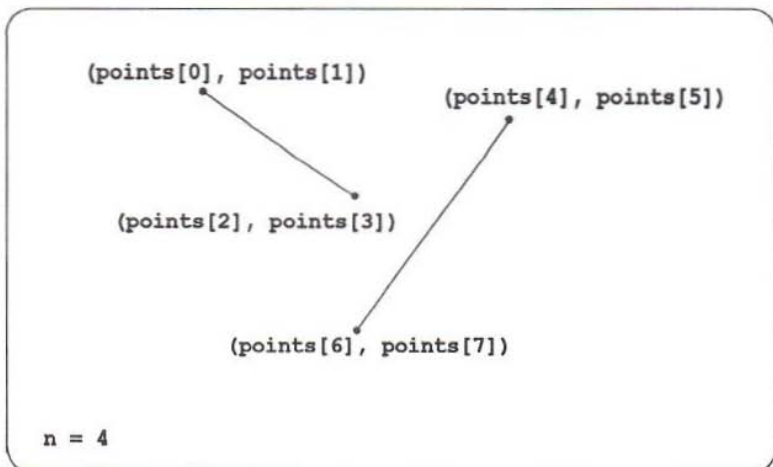
`cgi_disjpolyline` draws a sequence of disjoint (unconnected) straight lines between points defined by the integer vector `points`. The coordinate of a point is given by an integer pair (X,Y) and lines are drawn between a pair of coordinates specifying the line's start and end points. Each coordinate is used only once, as either a line start or as an end point. The first coordinate contained in `points` is always treated as a line start point and the next its corresponding end point. The number of points is given by `n` which will usually be even (because a line is described by two points). If `n` is odd a single point is plotted instead of the last line, if it is 1 only a single point is plotted.

Each line is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and plot modes affect the appearance of each line, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.12 cgi_dot

Plot a point.

C:

```
void cgi_dot(  
    Channel *to_cgi,  
    int X, int Y )
```

occam:

```
PROC cgi.dot(  
    CHAN OF ANY to_cgi,  
    VAL INT X, Y )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(X, Y)	Coordinate of point

Description:

cgi_dot plots a single point at (X, Y) .

The point is only plotted if it lies within the extent of the current screen definition, see cgi_setdrawscreen.

The current pixel replace and plot modes affect the appearance of the point, see cgi_setdrawmode.

Diagram:

Current screen



6.2.13 cgi_errstat

Expound the current CGI error.

C:

```
int cgi_errstat(
    Channel *from_cgi, Channel *to_cgi,
    char *errtext, int *errqual )
```

occam:

```
PROC cgi_errstat(
    CHAN OF ANY from.cgi, to.cgi,
    []BYTE errtext,
    INT errtext.len,
    INT errno, errqual )
```

Parameters:

Parameter	Comment
from_cgi	Channel from CGI server
to_cgi	Channel to CGI server
errtext	Text string indicating error
errtext.len	Length of error string (OCCAM only)
errno	CGI error code (OCCAM only)
errqual	CGI error qualifier

Note that `cgi_errstat` returns `errno`.

Description:

`cgi_errstat` returns the current CGI error status.

The CGI system records the reason for any error condition it encounters during normal operation, this consists of an error code and an error qualifier.

The error code `errno` indicates the reason for the current error and the qualifier `errqual` further qualifies it in a context sensitive way. For example, if the current error code describes an invalid pixel replace mode then the error qualifier will contain the offending mode value.

A textual description of the current error code is returned in `errtext`, this should contain at least `maxErrString` characters of storage. For OCCAM, `cgi_errstat` returns the length of the error string in `errtext.len`. The C variant returns a normal, null terminated, string.

The valid error number codes are:

errno	Comment
e_OK	No error
e_BADPELMODE	Invalid pixel plot mode, see <code>cgi_setdrawmode</code>
e_BADREPMODE	Invalid pixel replace mode, see <code>cgi_setdrawmode</code>
e_BADFILLMODE	Invalid fill mode, see <code>cgi_setdrawmode</code>
e_BADSEARCHDIRN	Invalid search direction, see <code>cgi_search</code>
e_BADSEARCHTEST	Invalid search test criteria, see <code>cgi_search</code>
e_BADFORIMODE	Invalid orientation, see <code>cgi_setorient</code>

The current error code and qualifier will be reset to indicate "No error".

6.2.14 cgi_fcircle

Plot a filled, axis aligned, ellipsoid.

C:

```
void cgi_fcircle(  
    Channel *to_cgi,  
    int Xc, int Yc, int A, int B )
```

occam:

```
PROC cgi_fcircle(  
    CHAN OF ANY to.cgi,  
    VAL INT Xc, Yc, A, B )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(Xc, Yc)	Centre coordinate
A	Length of X direction semi axis
B	Length of Y direction semi axis

Description:

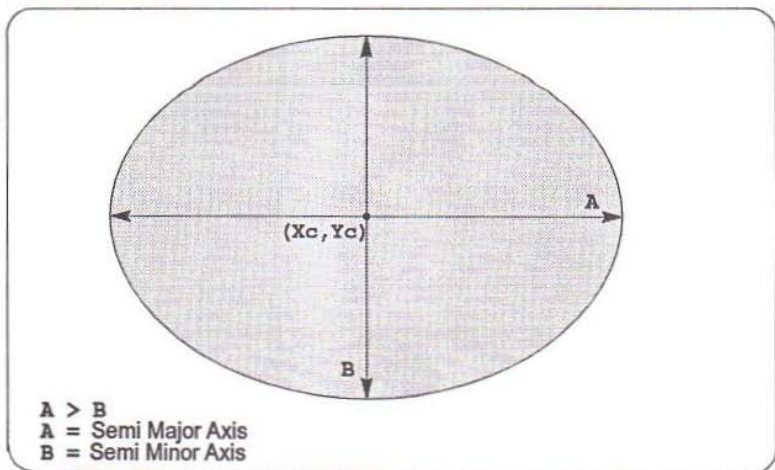
`cgi_fcircle` plots a filled, axis aligned, ellipsoid centred at (X_c, Y_c) and with semi-axis lengths of **A** and **B** pixels. Both **A** and **B** must be positive, the larger of the two values is the semi-major axis length, while the lesser specifies the semi-minor axis length. A filled circle is plotted with a diameter equal to either **A** or **B**, if they have identical values.

Every point plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and fill modes affect the appearance of the ellipse, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.15 cgi_fanfill

Plot a partially filled, axis aligned, ellipsoid. Closed with chord or segment lines.

C:

```
void cgi_fanfill(
    Channel *to_cgi,
    int Xc, int Yc, int A, int B,
    int DXs, int DYs, int DXe, int DYe,
    int CloseMode )
```

occam:

```
PROC cgi.fanfill(
    CHAN OF ANY to.cgi,
    VAL INT Xc, Yc, A, B, DXs, DYs, DXe, DYe,
    CloseMode )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(Xc, Yc)	Centre coordinate
A	Length of X direction semi axis
B	Length of Y direction semi axis
(DXs, DYs)	Start vector
(DXe, DYe)	End vector
CloseMode	Close mode

Description:

`cgi_fanfill` plots part of a filled, axis aligned, ellipsoid centred at (X_c, Y_c) and with semi-axis lengths of A and B pixels. Both A and B must be positive, the larger of the two values is the semi-major axis length, while the lesser specifies the semi-minor axis length.

(DX_s, DY_s) and (DX_e, DY_e) define direction vectors emanating from the centre of the ellipse that specify which part of its interior to fill. Only points clockwise of the (DX_s, DY_s) vector and anti clockwise of (DX_e, DY_e) are plotted.

The partial ellipse is bounded by either a single chord line, joining the two end points, or a pair of segment lines connecting each end point to the centre of the ellipse at (X_c, Y_c) . The value of `CloseMode` determines which method is used, valid values are:

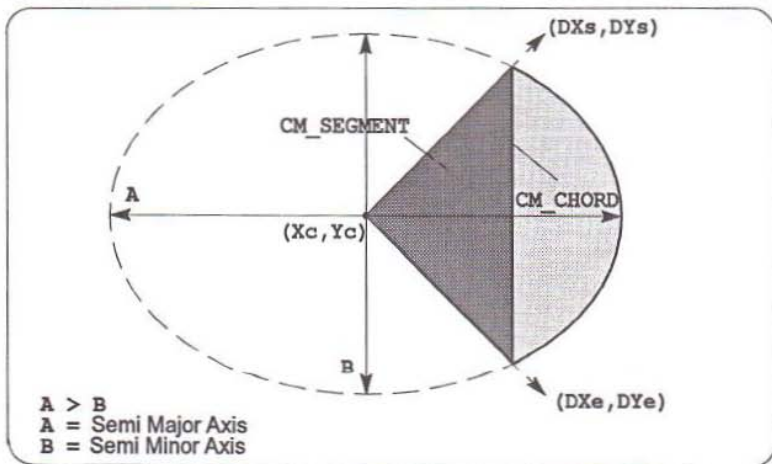
CloseMode	Comment
CM_CHORD	Close ellipsoid with a chord line
CM_SEGMENT	Close ellipsoid with two segment lines

Every point plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and fill modes affect the appearance of the ellipse, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.16 `cgi_fhline`

Plot a sequence of filled, horizontal, line segments.

C:

```
void cgi_fhline(
    Channel *to_cgi,
    int Y, int n, int *Xords )
```

occam:

```
PROC cgi_fhline(
    CHAN OF ANY to.cgi,
    VAL INT Y, n,
    VAL []INT Xords )
```

Parameters:

Parameter	Comment
<code>to_cgi</code>	Channel to CGI server
<code>Y</code>	Line segment Y ordinate
<code>n</code>	Number of X ordinates
<code>Xords</code>	Segment start and end X ordinates

Description:

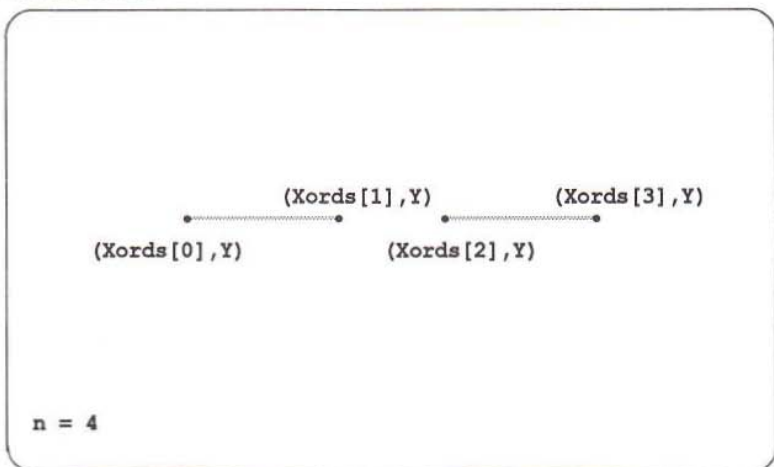
`cgi_fhline` plots a sequence of filled horizontal line segments between points defined by the integer vector `Xords` in conjunction with the single Y axis ordinate `Y`. The coordinate of a point is given by an integer pair (X,Y) and lines are filled between a pair of coordinates specifying the line's start and end points on the horizontal line Y. Each coordinate is used only once, as either a line start or as an end point. When combined with `Y`, the first X axis ordinate contained in `Xords` is always treated as a line start point and the next value used to define its corresponding end point. The number of X axis ordinates is given by `n` which must be even (because a line is described by two points).

Every line filled is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and fill modes affect the appearance of each line, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.17 cgi_frect

Plot a filled, axis aligned, rectangle.

C:

```
void cgi_frect(  
    Channel *to_cgi,  
    int X0, int Y0, int X1, int Y1 )
```

OCcam:

```
PROC cgi.frect(  
    CHAN OF ANY to.cgi,  
    VAL INT X0, Y0, X1, Y1 )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(X0,Y0)	Corner point coordinate
(X1,Y1)	Opposite point coordinate

Description:

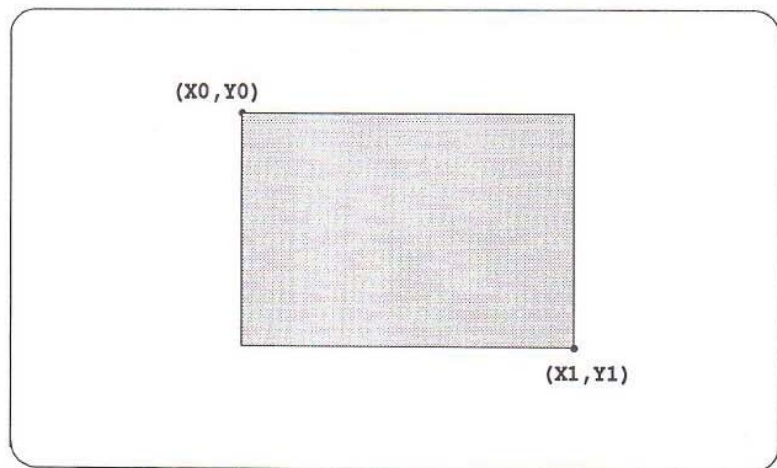
`cgi_frect` plots a filled, axis aligned, rectangle between two diagonally opposite points specified by the coordinates (X0,Y0) and (X1,Y1).

Every point plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and fill modes affect the appearance of the rectangle, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.18 cgi_ftrap

Plot a filled trapezoid.

C:

```
void cgi_ftrap(
    Channel *to_cgi,
    int X1, int Y1, int X2, int Y2,
    int X3, int Y3, int X4, int Y4,
    int Ys, int Ye )
```

occam:

```
PROC cgi.ftrap(
    CHAN OF ANY to.cgi,
    VAL INT X1, Y1, X2, Y2, X3, Y3, X4, Y4, Ys, Ye )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(X1, Y1)	First edge: Start point coordinate
(X2, Y2)	First edge: End point coordinate
(X3, Y3)	Second edge: Start point coordinate
(X4, Y4)	Second edge: End point coordinate
Ys	Top horizontal Y axis bound
Ye	Bottom horizontal Y axis bound

Description:

`cgi_ftrap` plots a filled trapezoid. The trapezoid is horizontally bounded by two non-horizontal edges, filling occurs between the left and right edge lines. The first edge is specified by a straight line between the points (X1, Y1) and (X2, Y2) and the second edge by a line between (X3, Y3) and (X4, Y4). The fill area is vertically bounded by two horizontal edges. The top edge is described by a horizontal line with a Y axis value equal to the larger of Ys and the smallest Y1, Y2, Y3 or Y4 ordinate. The bottom edge line has a Y axis value equal to the lesser of Ye and the largest Y1, Y2, Y3 or Y4 ordinate.

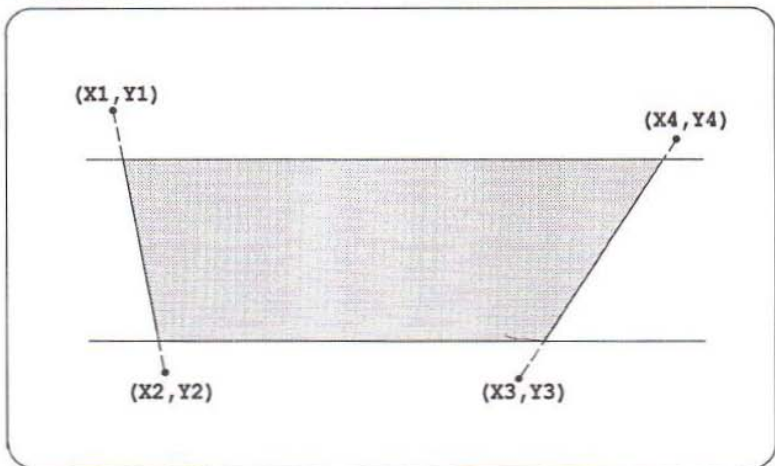
The left and right edge lines may intersect. If they do, an object similar in shape to an hour glass (two touching triangles) will be plotted.

Every point plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and fill modes affect the appearance of the trapezoid, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.19 cgi_line

Draw a straight line between two points.

C:

```
void cgi_line(  
    Channel *to_cgi,  
    int X0, int Y0, int X1, int Y1 )
```

occam:

```
PROC cgi.line(  
    CHAN OF ANY to.cgi,  
    VAL INT X0, Y0, X1, Y1 )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(X0,Y0)	Start point coordinate
(X1,Y1)	End point coordinate

Description:

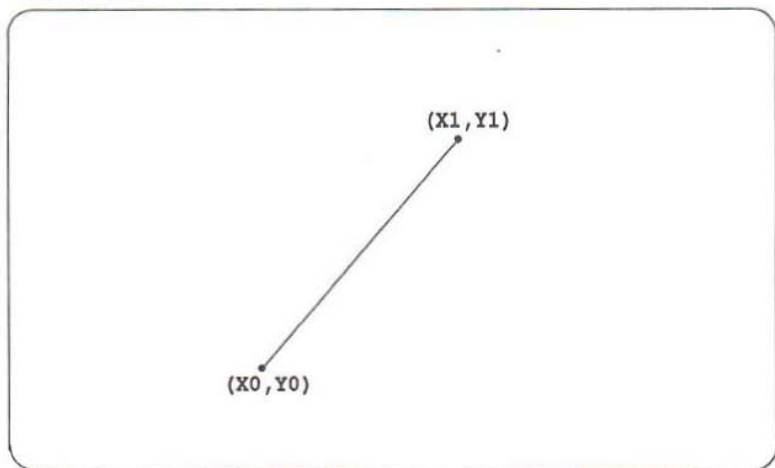
cgi_line plots a straight line between two points specified by the coordinates (X0,Y0) and (X1,Y1).

Every point plotted is clipped to the current screen definition, see cgi_setdrawscreen.

The current pixel replace and plot modes affect the appearance of the line, see cgi_setdrawmode.

Diagram:

Current screen



6.2.20 cgi_paint

Paint (flood fill) a bounded region.

C:

```
void cgi_paint(
    Channel *to_cgi,
    int Xs, int Ys, int Bcol )
```

occam:

```
PROC cgi.paint(
    CHAN OF ANY to.cgi,
    VAL INT Xs, Ys, Bcol )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(Xs, Ys)	Interior point coordinate
Bcol	Boundary colour

Description:

`cgi_paint` flood fills a bounded region. The region is specified by a boundary of constant colour `Bcol` and filling starts at an interior point given by the coordinate `(Xs, Ys)`. If the pixel at this point already has the value `Bcol` then no filling occurs.

The current pixel replace and fill modes affect the resultant display, see `cgi_setdrawmode`.

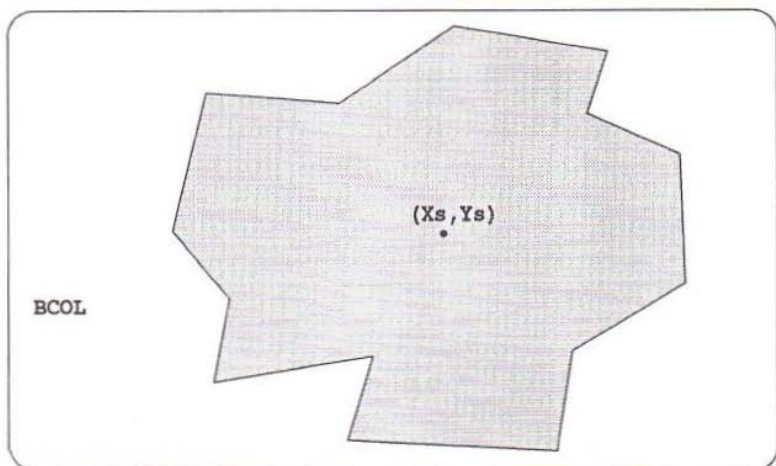
Filling with the current foreground colour (fill mode `FM_COL` and plot mode `PM_COL`) equal to the defined boundary colour produces a correct result. However, the use of a fill pattern which contains pixels of the boundary colour will almost certainly fail.

The fill algorithm guarantees correct behaviour when a logical pixel replace mode is active by plotting each pixel once only, see `cgi_setdrawmode`.

The fill region is clipped to the current screen definition, see `cgi_setdrawscreen`.

Diagram:

Current screen



6.2.21 cgi_polygon

Outline a polygon.

C:

```
void cgi_polygon(  
    Channel *to_cgi,  
    int n, int *points )
```

OCCAM:

```
PROC cgi.polygon(  
    CHAN OF ANY to.cgi,  
    VAL INT n,  
    VAL []INT points )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
n	Number of (X,Y) points
points	Polygon vertex points

Description:

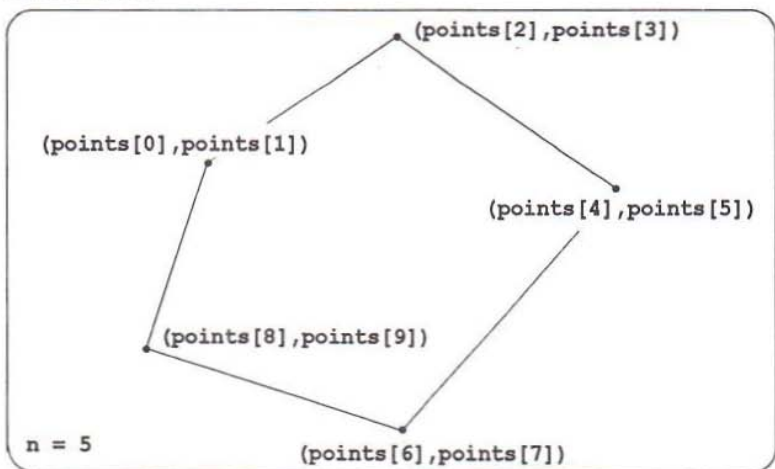
cgi_polygon plots the outline of a polygon by drawing a sequence of connected, straight lines, between its vertex points. The polygon's last vertex point is connected to its first to complete the outline. The coordinate of each point is given by an integer pair (X,Y) taken from the vector points, the number of points is specified by n. Lines are drawn in the order defined by each consecutive point contained in points. If only one coordinate is present, or if all the points are coincident, then a single point is plotted.

The outline is clipped to the current screen definition, see cgi_setdrawscreen.

The current pixel replace and plot modes affect the appearance of the outline, see cgi_setdrawmode.

Diagram:

Current screen



6.2.22 cgi_polyline

Draw a sequence of connected lines.

C:

```
void cgi_polyline(  
    Channel *to_cgi,  
    int n, int *points )
```

occam:

```
PROC cgi.polyline(  
    CHAN OF ANY to_cgi,  
    VAL INT n,  
    VAL []INT points )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
n	Number of (X,Y) points
points	Line start and end points

Description:

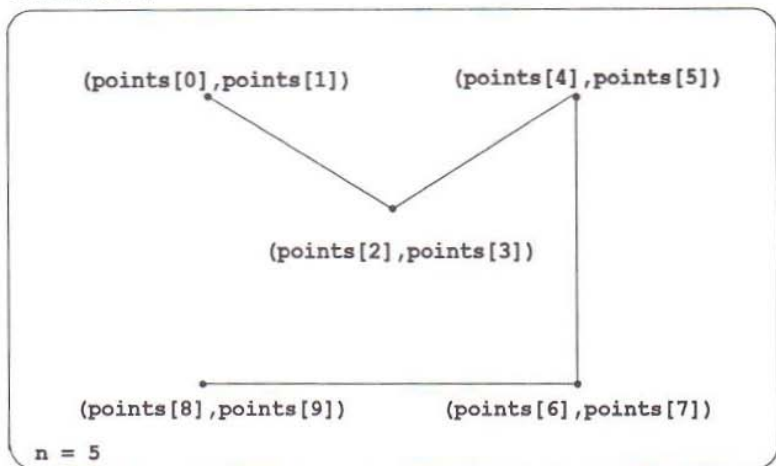
`cgi_polyline` draws a sequence of straight lines connecting the points defined by the integer vector `points`. The number of points is specified by `n`. The coordinate of a point is given by an integer pair (X,Y) and lines are drawn between a pair of coordinates specifying the line's start and end points. The drawing order is defined by each consecutive point contained in `points`. The resulting, continuous line, is called a `polyline`.

The `polyline` is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and plot modes affect the appearance of the `polyline`, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.23 cgi_rect

Outline an axis aligned rectangle.

C:

```
void cgi_rect(  
    Channel *to_cgi,  
    int X0, int Y0, int X1, int Y1 )
```

occam:

```
PROC cgi.rect(  
    CHAN OF ANY to.cgi,  
    VAL INT X0, Y0, X1, Y1 )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
(X0, Y0)	Corner point coordinate
(X1, Y1)	Opposite point coordinate

Description:

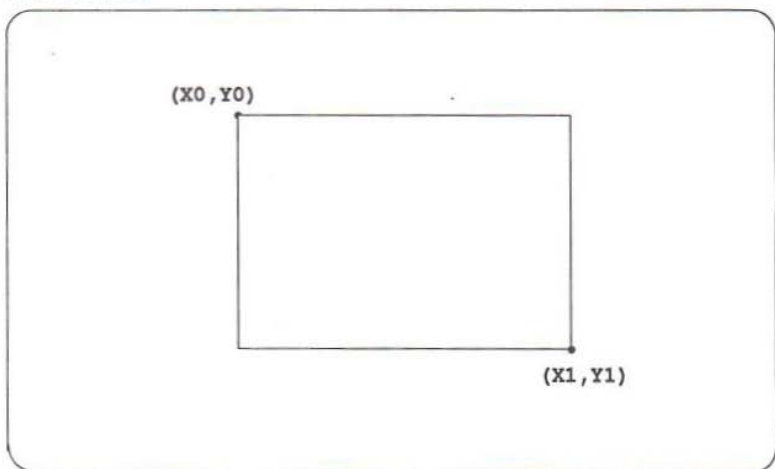
`cgi_rect` plots an outline of an axis aligned rectangle between two diagonally opposite points specified by the coordinates (X0, Y0) and (X1, Y1).

The outline is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace and plot modes affect the appearance of the outline, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.24 cgi_rot

2D region block rotation.

C:

```
void cgi_rot(
    Channel *to_cgi,
    screen s, int Xs, int Ys, int LSX, int LSY,
    int Xd, int Yd, float angle )
```

OCCAM:

```
PROC cgi.rot(
    CHAN OF ANY to.cgi,
    VAL [SCREEN.SIZE]INT s,
    VAL INT Xs, Ys, LSX, LSY, Xd, Yd,
    VAL REAL32 angle )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
s	Screen
(Xs, Ys)	Source coordinate
LSX	Size of region in X direction
LSY	Size of region in Y direction
(Xd, Yd)	Destination coordinate
angle	Radian angle of rotation

Description:

`cgi_rot` copies and rotates a rectangular, axis aligned, region from the source screen `s` to the current drawing screen. The size of the source region is specified by `DX` pixels in the X axis direction and `DY` pixels on the Y axis. It is rotated through an angle of `angle` radians, a positive value denotes an anti-clockwise angle of rotation.

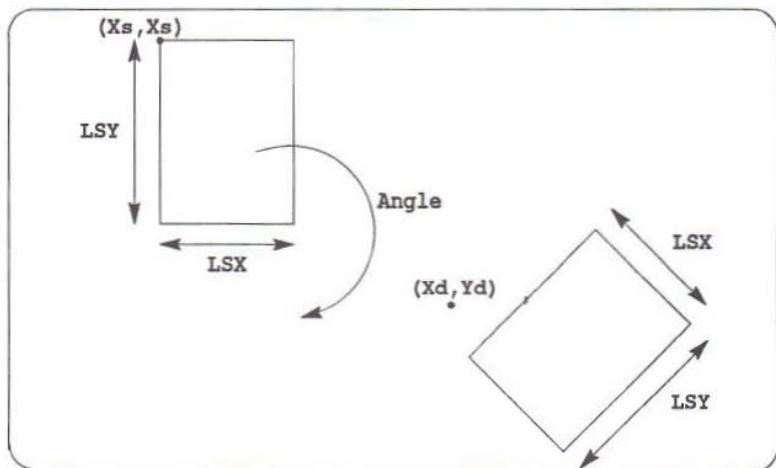
The coordinate `(Xs, Ys)` identifies the top left hand corner of the region on the source screen, its rotated copy is plotted at `(Xd, Yd)` on the current drawing screen.

The rotated region is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace mode affects the resultant display, see `cgi_setdrawmode`.

Diagram:

Source screen



6.2.25 cgi_search

Scan a horizontal line segment for colour change.

C:

```
int cgi_search(
    Channel *from_cgi, Channel *to_cgi,
    int Xs, int Ys, int Bcol,
    int sense, int dirn )
```

occam:

```
PROC cgi.search(
    CHAN OF ANY from.cgi, to.cgi,
    VAL INT Xs, Ys, Bcol, sense, dirn,
    INT xposn )
```

Parameters:

Parameter	Comment
from_cgi	Channel from CGI server
to_cgi	Channel to CGI server
(Xs, Ys)	Search point coordinate
Bcol	Colour transition
sense	Search criteria
dirn	Search direction
xposn	X axis result (OCCAM only)

Note that `cgi_search` returns `xposn`.

Description:

`cgi_search` is used to discover where on a horizontal line, a particular colour change occurs. The start point for the search is specified by the coordinate (Xs, Ys), the search occurs along a horizontal line drawn through it. The search proceeds in one of two directions: either to the left of the start point, or to its right, as specified by `dirn`. Searching continues until a pixel of the transition colour `Bcol` is discovered, or until a pixel of some other colour is found. The search criteria `sense` defines which method to use.

Valid search direction values are:

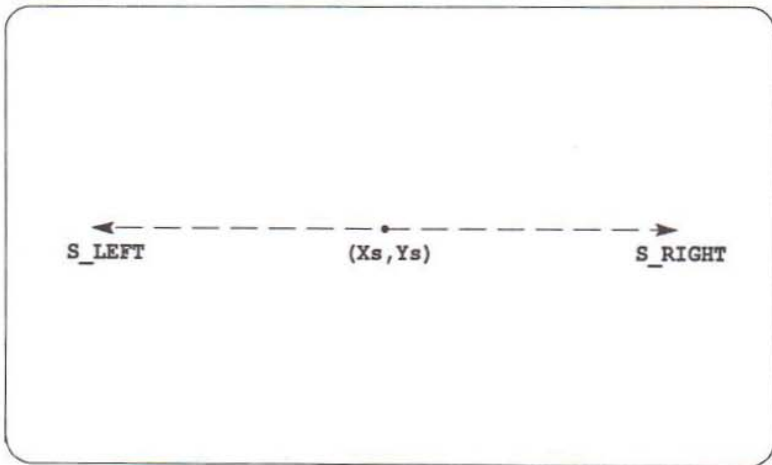
Search direction	Comment
S_LEFT	Search left of start point
S_RIGHT	Search right of start point

Valid search criteria values are:

Search test	Comment
S_WHILENOT	Search until a pixel of colour Bco1 is discovered
S_WHILEGOT	Search until a pixel not equal in colour to Bco1 is discovered

Diagram:

Current screen



6.2.26 cgi_setbcol

Set current background colour.

C:

```
oid cgi_setbcol(  
    Channel *to_cgi,  
    int Bcol )
```

occam:

```
PROC cgi.setbcol(  
    CHAN OF ANY to.cgi,  
    VAL INT Bcol )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
Bcol	Background colour

Description:

cgi_setbcol sets the current background colour to Bcol.

6.2.27 cgi_setdrawmode

Set current draw modes for plotting, filling and logical pixel operations.

C:

```
void cgi_setdrawmode(  
    Channel *to_cgi,  
    int pm, int rm, int fm )
```

occam:

```
PROC cgi.setdrawmode(  
    CHAN OF ANY to.cgi,  
    VAL INT pm, rm, fm )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
pm	Plot mode
rm	Replace mode
fm	Fill mode

Description:

`cgi_setdrawmode` sets the current pixel plot, replace and fill modes to `pm`, `rm` and `fm` respectively.

The pixel plot mode `pm` affects the result of most drawing operations, such as `cgi_polyline` or `cgi_arc`. Drawing operations are achieved by plotting a sequence of points according to the current plot mode. It defines whether a single pixel, the current picture element or the current line style pattern is used to determine how each point should be plotted. Valid pixel plot modes are:

Plot mode	Comment
PM_COL	Points are plotted as a single pixel in the current foreground colour, see <code>cgi_setfcol</code> .
PM_LINestyle	Points are plotted according to the current line style pattern, see <code>cgi_setlinestyle</code> .
PM_LINestyleTR	As PM_LINestyle, except that zero valued linestyle pattern pixels are not plotted. This achieves a transparency affect.
PM_PEL	Single points are represented by the current picture element pattern, see <code>cgi_setpelstyle</code> .
PM_LS_PEL	As PM_LINestyle, except that single points defined by the current line style pattern are replaced by the current picture element pattern.

The pixel replace mode `rm` affects the result of all drawing and fill operations. It defines how pixels are ultimately written into the current frame store and therefore the colour that each pixel will assume when displayed. Pixels can either be combined (using a bitwise operator) with the value of a pixel at the same location, or they can be written directly into the frame store. Valid pixel replace modes are:

Replace mode	Comment
RM_COL	Overwrite mode: pixel defined by the current foreground colour, see <code>cgi_setfcol</code> .
RM_AND	Colour defined by the bitwise AND of the new pixel value and the existing framestore pixel.
RM_OR	Colour defined by the bitwise OR of the new pixel value and the existing framestore pixel.
RM_XOR	Colour defined by the bitwise XOR of the new pixel value and the existing framestore pixel.
RM_NOR	Colour defined by the bitwise NOR of the new pixel value and the existing framestore pixel.
RM_NAND	Colour defined by the bitwise NAND of the new pixel value and the existing framestore pixel.
RM_Z	Overwrite mode: existing framestore pixel only over written with zero valued new pixels.
RM_NZ	Overwrite mode: existing framestore pixel only over written with non-zero valued new pixels.
RM_ALL	Overwrite existing pixel with new pixel value.

The fill mode `fm` affects only fill operations, such as `cgi_frect`. It defines how filling should be performed, valid fill modes are:

Fill mode	Comment
<code>FM_COL</code>	Fill with current foreground colour, see <code>cgi_setfcol</code>
<code>FM_PATTERN</code>	Fill with current fill style, see <code>cgi_setfillstyle</code>

6.2.28 cgi_setdrawscreen

Set current drawing screen.

C:

```
void cgi_setdrawscreen(  
    Channel *to_cgi,  
    screen s )
```

occam:

```
PROC cgi.setdrawscreen(  
    CHAN OF ANY to.cgi,  
    VAL [SCREEN.SIZE]INT s )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
s	Screen

Description:

cgi_setdrawscreen sets the current screen. The screen, specified by *s*, defines the size and location of the frame store raster to use for all subsequent CGI operations.

6.2.29 cgi_setfcol

Set current foreground colour.

C:

```
void cgi_setfcol(  
    Channel *to_cgi,  
    int Fcol )
```

occam:

```
PROC cgi.setfcol(  
    CHAN OF ANY to.cgi,  
    VAL INT Fcol )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
Fcol	Foreground colour

Description:

`cgi_setfcol` sets the current foreground colour to `Fcol`.

6.2.30 cgi_setfillstyle

Set current customised fill style.

C:

```
void cgi_setfillstyle(
    Channel *to_cgi,
    int fxsize, int fysize,
    char *fillmap )
```

occam:

```
PROC cgi.setfillstyle(
    CHAN OF ANY to.cgi,
    VAL INT fxsize, fysize,
    VAL [ ]BYTE fillmap )
```

Parameters:

Parameter	Comment
<code>to_cgi</code>	Channel to CGI server
<code>fxsize</code>	Width of fill style on X axis
<code>fysize</code>	Height of fill style on Y axis
<code>fillmap</code>	Fill style pixel map

Description:

`cgi_setfillstyle` sets the current fill style. Fill styles are represented by a two dimensional pattern which is used to tile a screen area during patterned fill operations: the pattern is replicated over the screen area to be filled. The size of the fill pattern is given by `fxsize` pixels in the X axis direction, and `fysize` pixels along the Y axis. The fill style is described by the vector `fillmap` which should contain, in horizontal line order, each row of pixels that make up the custom fill pattern. The maximum width and height of a fill pattern is `maxFillSize` pixels.

Note that the current fill style is only used for fill operations if the current fill mode is `FM_PATTERN`, see `cgi_setdrawmode`.

The current pixel replace mode affects the resultant display: pixels defined by the fill pattern are plotted according to the current pixel replace mode. See `cgi_setdrawmode`.

6.2.31 cgi_setfont

Set current text font.

C:

```
int cgi_setfont(
    Channel *from_cgi, Channel *to_cgi,
    unsigned int *packfont,
    int nchars, int famw, int fwpc, int flpw )
```

Occam:

```
PROC cgi.setfont(
    CHAN OF ANY from.cgi, to.cgi,
    VAL INT[] packfont,
    VAL INT nchars, famw, fwpc, flpw,
    BOOL ok )
```

Parameters:

Parameter	Comment
from_cgi	Channel from CGI server
to_cgi	Channel to CGI server
packfont	Encoded font
nchars	Number of characters in font
famw	Width of unpacked character in bits
fwpc	Number of 32 bit words per character
flpw	Number of encoded rows per 32 bit word
ok	Success status (OCCAM only)

Note that `cgi_setfont` returns non-zero if the font was loaded successfully, zero otherwise. `cgi.setfont` returns the boolean variable `ok` to indicate success or failure.

Description:

`cgi_setfont` loads a font description into the CGI server. Only one font may be active at any one instant so this operation will overwrite any existing font description held by the server. If there is insufficient memory for the new font `cgi_setfont` will return an error status.

Fonts are described by an integer vector which contains a packed representation of each character contained in the font. A font can contain any number of characters only limited by the memory restrictions of the CGI server. A bit mask is used to represent each character cell, this has a constant width and height for all the characters of the same font. Bits are listed in horizontal scan line order for each character: a one bit represents a coloured pixel and a zero bit the background. The

bit masks for each character are packed into a number of 32 bit words, these are then concatenated to produce the packed font. The order defines how characters are retrieved from the font: the integer value of a character is used as the index of the corresponding character cell within the font array, if ASCII text representation is required then the font should contain character descriptions at positions that correspond to the ASCII encoding.

`packfont` is an integer vector that describes a font containing `nchars` characters. Character cells are described by the parameters `famw`, `fwpc` and `flpw`. The width of the font is given by `famw`, this specifies the number of bits to use per horizontal row. Each bit defines whether a foreground or background pixel is plotted. `fwpc` is the number of 32 bit words required to encode one character cell and `flpw` is the number of horizontal rows encoded per word.

All text operations use the current font description.

Section 10.1 'Using and defining text fonts' describes this in more detail.

6.2.32 cgi_setlinestyle

Set current customised line style.

C:

```
void cgi_setlinestyle(
    Channel *to_cgi,
    int n, char *ls,
    int zoomFac )
```

occam:

```
PROC cgi.setlinestyle(
    CHAN OF ANY to.cgi,
    VAL INT n,
    VAL []BYTE ls,
    VAL INT zoomFac )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
n	Length of line style
ls	Line style pixel map
zoomFac	Zoom factor

Description:

`cgi_setlinestyle` sets the current line style. Line styles are represented by a one dimensional vector of pixels. During drawing operations, the current line style vector can be used to define the value of the next pixel to plot. Pixels are taken from the line style vector and used to plot a specific number of subsequent points, as defined by the line style zoom factor. When a pixel value has been exhausted the next pixel from the line style vector is used, if it was the last pixel then the first pixel is re-used.

`ls` describes a line style of length `n` pixels. The maximum length of a line style is `maxLineStyle` pixels, the minimum length is 1.

The number of times a line style pixel is plotted is given by the zoom factor `zoomFac`.

Note that the current line style is only used for drawing operations if the current pixel plot mode specifies one of the line style plot functions, see `cgi_setdrawmode`.

The current pixel replace mode affects the resultant display: pixels defined by the line style are plotted according to the current pixel replace mode. See `cgi_setdrawmode`.

6.2.33 cgi_setorient

Set current orientation for text and copy operations.

C:

```
void cgi_setorient(
    Channel *to_cgi,
    int orient )
```

occam:

```
PROC cgi.setorient(
    CHAN OF ANY to.cgi,
    VAL INT orient )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
orient	Orientation

Description:

`cgi_setorient` sets the current orientation mode to `orient`. This specifies one of eight, axis aligned, orientations to apply when plotting characters or performing two dimensional block copy operations (see `cgi_copy`). Valid orientation values are:

Orientation	Comment
TX_NORM	Normal orientation
TX_90	Rotate 90 degrees clockwise
TX_180	Rotate 180 degrees clockwise
TX_270	Rotate 270 degrees clockwise
TX_NORMFLIP	Flip top to bottom
TX_90FLIP	Rotate 90 degrees clockwise, then flip top to bottom
TX_180FLIP	Rotate 180 degrees clockwise, then flip top to bottom
TX_270FLIP	Rotate 270 degrees clockwise, then flip top to bottom

6.2.34 cgi_setpelstyle

Set current customised pel style.

C:

```
void cgi_setpelstyle(
    Channel *to_cgi,
    int pxsize, int pysize,
    char *pelmap,
    int xorg, int yorg )
```

occam:

```
PROC cgi.setpelstyle(
    CHAN OF ANY to.cgi,
    VAL INT pxsize, pysize,
    VAL []BYTE pelmap,
    VAL INT xorg, yorg )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
pxsize	Width of pel style on X axis
pysize	Height of pel style on Y axis
pelmap	Pel style pixel map
(xorg,yorg)	Offset to centre of Pel

Description:

`cgi_setpelstyle` sets the current pel style. Pel styles are represented by a two dimensional pattern which is copied into the current screen where a single point would otherwise have been plotted. Every operation that is implemented by drawing a sequence of points, such as `cgi_line`, can be configured to plot the pel style pattern instead.

The size of the pel pattern is given by `pxsize` pixels in the X axis direction, and `pysize` pixels along the Y axis. The pel style is described by the vector `pelmap` which should contain, in horizontal line order, each row of pixels that make up the custom pel pattern. The maximum width and height of a pel pattern is `maxPelSize` pixels.

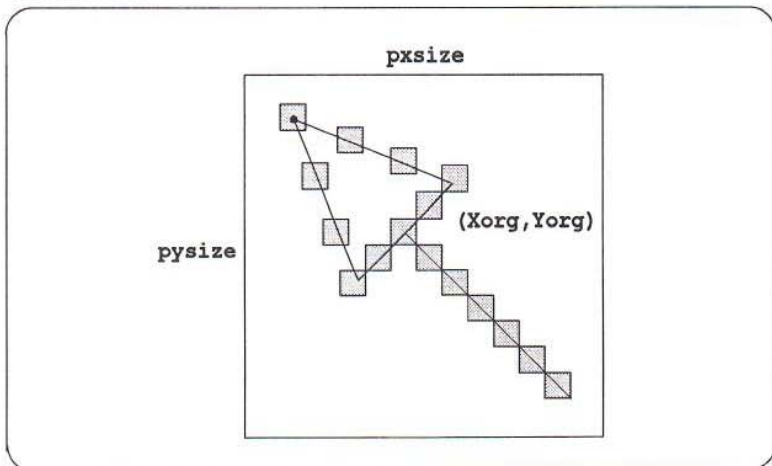
The pel style has a single point, located within its bulk, that identifies an origin. Pels are plotted such that their origins are positioned at the coordinate of the replaced point. `(xorg,yorg)` defines the origin of the Pel style as an offset from the base of its two dimensional pattern, (top left hand corner).

The current pixel replace mode affects the resultant display: pixels defined by the pel style are plotted according to the current pixel replace mode. See `cgi_setdrawmode`.

Note that the current pel style is only used for drawing operations if the current pixel plot mode specifies one of the pel functions, see `cgi_setdrawmode`.

Diagram:

Current screen



6.2.35 cgi_shear

2D region block shear.

C:

```
void cgi_shear(
  Channel *to_cgi,
  screen s, int Xs, int Ys, int LSX, int LSY,
  int Xd, int Yd,
  int LDXx, int LDYx, int LDYx, int LDYy )
```

OCCAM:

```
PROC cgi.shear(
  CHAN OF ANY to.cgi,
  VAL [SCREEN.SIZE]INT s,
  VAL INT Xs, Ys, LSX, LSY, Xd, Yd,
  LDXx, LDYx, LDYx, LDYy )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
s	Screen
(Xs, Ys)	Source coordinate
LSX	Size of region in X direction
LSY	Size of region in Y direction
(Xd, Yd)	Destination coordinate
LDXx, LDYx	LSX shear control
LDYx, LDYy	LSY shear control

Description:

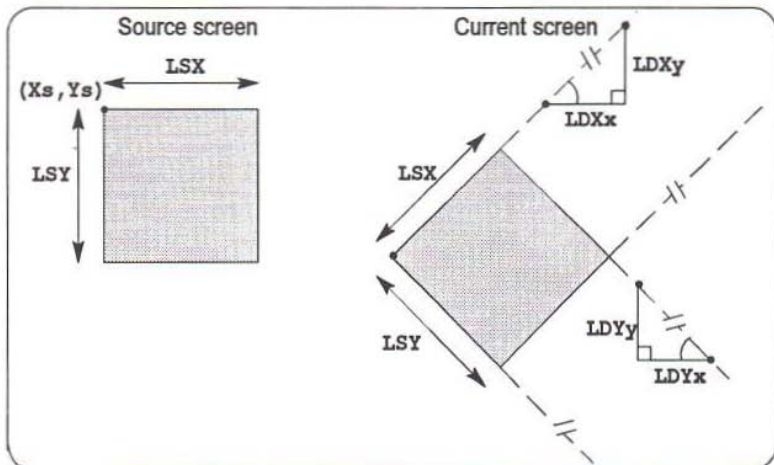
`cgi_shear` copies and shears a rectangular, axis aligned, region from the source screen `s` to the current drawing screen. The size of the source region is specified by `LSX` pixels in the X axis direction and `LSY` pixels on the Y axis. It is sheared according to the value of the four shear control parameters: `LDXx`, `LDYx`, `LDYx` and `LDYy`. `LDXx` and `LDYx` control the amount and direction of the shear along the `LSX` line: `LDXx` is the component of shear in the X axis direction, `LDYx` is the Y axis component. Similarly, `LDYx` and `LDYy` control the shear along the `LSY` line, `LDYx` is the X axis component and `LDYy` is the Y axis part. Each pair of shear control parameters define a right-angled triangle with axis aligned sides, the hypoteneus lines define the direction of the sheared horizontal or vertical edges of the original, rectangular, region.

The coordinate `(Xs, Ys)` identifies the top left hand corner of the region on the source screen, its sheared copy is plotted at `(Xd, Yd)` on the current drawing screen.

The sheared region is clipped to the current screen definition, see `cgi_setdrawscreen`.

The current pixel replace mode affects the resultant display, see `cgi_setdrawmode`.

Diagram:



6.2.36 `cgi_sptext`

Plot text at specified position, with spacing control.

C:

```
void cgi_sptext(
    Channel *to_cgi,
    int X, int Y,
    int n, char *str,
    int *dx, int *dy )
```

occam:

```
PROC cgi.sptext(
    CHAN OF ANY to.cgi,
    VAL INT X, Y, n,
    VAL []BYTE str,
    VAL []INT dx, dy )
```

Parameters:

Parameter	Comment
<code>to_cgi</code>	Channel to CGI server
<code>(X,Y)</code>	Start coordinate
<code>n</code>	Number of characters to plot
<code>str</code>	Character string
<code>dx</code>	X axis character spacing distances
<code>dy</code>	Y axis character spacing distances

Description:

`cgi_sptext` plots `n` characters from the character string `str` according to the current font description. The first character is plotted at the current character position, which is initially set to `(X,Y)`. It is then incremented by `X` and `Y` axis offsets specified by the inter-character spacing vectors `dx` and `dy`, for the character. Subsequent characters are plotted in the same manner, using the next pair of spacing distances. The current character position after the operation completes is offset from the first character plotted by `X` and `Y` axis distances equal to the sum of the `dx` and `dy` spacing vectors respectively.

The spacing vectors should be set with respect to the current orientation, see `cgi_setorient`. Characters are plotted according to the current pixel replace mode, see `cgi_setdrawmode`.

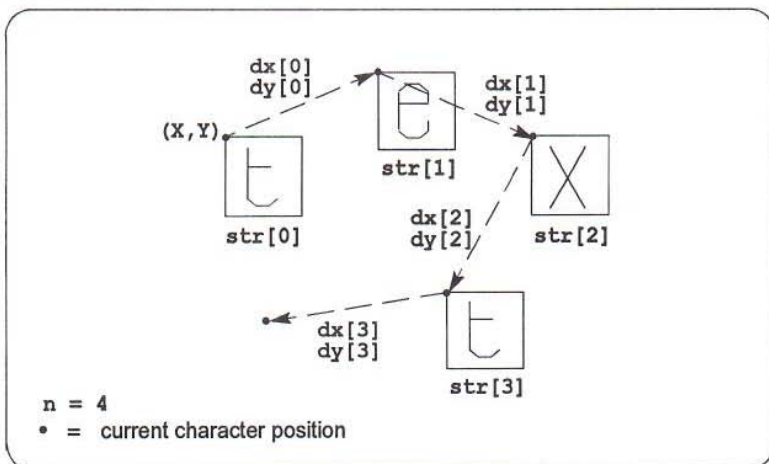
Characters are reproduced at the size of their font, which should be initialised, see `cgi_setfont`. Each pixel of every character plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

For text display, the default pixel replace mode `RM_COL`, will cause characters to imprint within a rectangular bounding box of colour 0. In some cases this will not

produce the desired effect. If only the foreground of the text is required and a pixel overwrite mode rather than a logical operation is desired then select pixel replace mode `RM_NZ`. This will cause only those pixels which are non-zero to be plotted.

Diagram:

Current screen



6.2.37 cgi_strokearc

Outline part of an axis aligned ellipsoid, closed with chord or segment lines.

C:

```
void cgi_strokearc(  
    Channel *to_cgi,  
    int Xc, int Yc, int A, int B,  
    int DXs, int DYs, int DXe, int DYe,  
    int CloseMode )
```

occam:

```
PROC cgi.strokearc(  
    CHAN OF ANY to.cgi,  
    VAL INT Xc, Yc, A, B, DXs, DYs, DXe, DYe,  
    CloseMode )
```

Parameters:

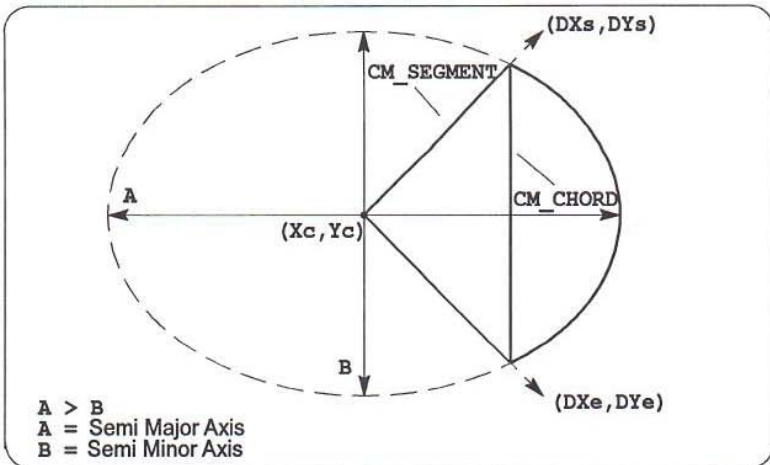
Parameter	Comment
to_cgi	Channel to CGI server
(Xc, Yc)	Centre coordinate
A	Length of X direction semi axis
B	Length of Y direction semi axis
(DXs, DYs)	Start vector
(DXe, DYe)	End vector
CloseMode	Close mode for arc

Description:

`cgi_strokearc` performs the same function as `cgi_arcc`. However, when drawing with a defined line style, `cgi_strokearc` achieves a more pleasing result. This is because `cgi_strokearc` uses a non-optimal algorithm and calculates individual points rather than using a faster (less accurate) technique.

Diagram:

Current screen



6.2.38 `cgi_text`

Plot text at specified position.

C:

```
void cgi_text(
    Channel *to_cgi,
    int X, int Y,
    int n, char *str )
```

Occam:

```
PROC cgi.text(
    CHAN OF ANY to.cgi,
    VAL INT X, Y, n,
    VAL [ ]BYTE str )
```

Parameters:

Parameter	Comment
<code>to_cgi</code>	Channel to CGI server
<code>(X, Y)</code>	Start coordinate
<code>n</code>	Number of characters to plot
<code>str</code>	Character string

Description:

`cgi_text` plots `n` characters from the character string `str` according to the current font description. Characters are plotted at the current character position which is then incremented by the currently defined X and Y axis inter-character spacing distances, see `cgi_chrspace`. The current character position after the operation completes is offset from the last character plotted by these distances.

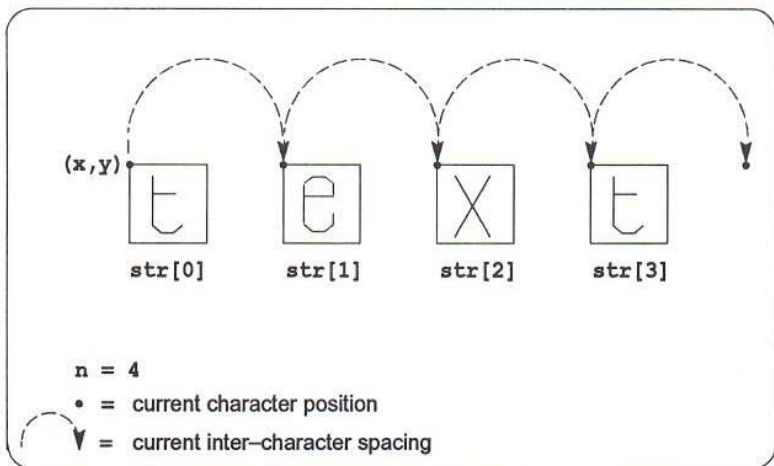
Characters are plotted according to the current pixel replace mode, see `cgi_setdrawmode` and the current orientation, see `cgi_setorient`.

Characters are reproduced at the size of their font, which should be initialised, see `cgi_setfont`. Each pixel of every character plotted is clipped to the current screen definition, see `cgi_setdrawscreen`.

For text display, the default pixel replace mode `RM_COI`, will cause characters to imprint within a rectangular bounding box of colour 0. In some cases this will not produce the desired effect. If only the foreground of the text is required and a pixel overwrite mode rather than a logical operation is desired then select pixel replace mode `RM_NZ`. This will cause only those pixels which are non-zero to be plotted.

Diagram:

Current screen



6.2.39 cgi_zoom

2D region block copy with zoom scaling.

C:

```
void cgi_zoom(
    Channel *to_cgi,
    screen s,
    int Xs, int Ys, int LSX, int LSY,
    screen d,
    int Xd, int Yd, int LDX, int LDY,
    int interpolate )
```

OCCAM:

```
PROC cgi.zoom(
    CHAN OF ANY to.cgi,
    VAL [SCREEN.SIZE]INT s,
    VAL INT Xs, Ys, LSX, LSY,
    VAL [SCREEN.SIZE]INT d,
    VAL INT Xd, Yd, LDX, LDY, interpolate )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
s	Source screen
(Xs, Ys)	Source coordinate
LSX	Size of source in X direction
LSY	Size of source in Y direction
d	Destination screen
(Xd, Yd)	Destination coordinate
LDX	Size of destination in X direction
LDY	Size of destination in Y direction
interpolate	Interpolated zoom flag

Description:

`cgi_zoom` copies a rectangular, axis aligned, region from the source screen `s` to the destination screen `d`. It performs arbitrary scaling independently in the X and Y axis directions to achieve a zoom effect.

The size of the source region is specified by `LSX` pixels in the X axis direction and `LSY` pixels on the Y axis. It is scaled to fit the size of the destination region given by `LDX` pixels in the X axis direction and `LDY` pixels on the Y axis.

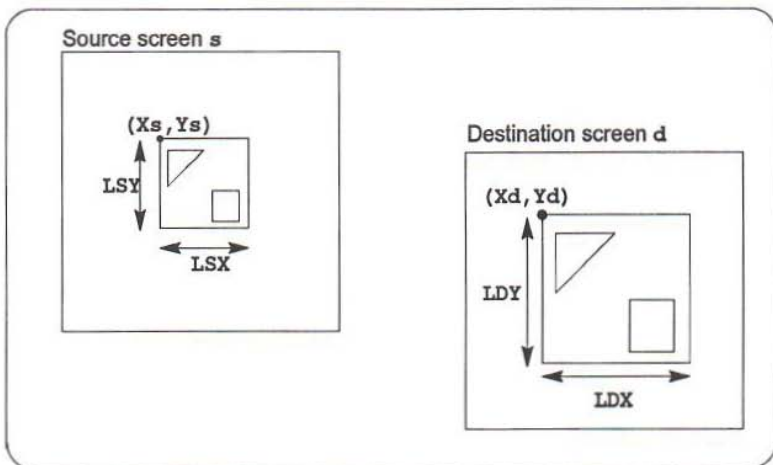
The coordinate `(Xs, Ys)` identifies the top left hand corner of the region on the source screen, it is copied to `(Xd, Yd)` on the destination screen.

`interpolate` controls whether an interpolated zoom will be performed. If it has the value zero, no interpolation will be performed. If it is non-zero an interpolation algorithm will be applied when copying pixels to the destination screen.

The scaled source region is clipped to the destination screen definition.

The current pixel replace mode affects the resultant display, see `cgi_setdrawmode`.

Diagram:



7 Graphics board functions

7.1 List of functions

7.1.1 fs_screenaddr

Return the raster address of a screen.

C:

```
char *fs_screenaddr(  
    Channel *from_cgi, Channel *to_cgi,  
    int bank )
```

Note:

There is no equivalent occam variant of this function because the language does not support indirect addressing.

Parameters:

Parameter	Comment
from_cgi	Channel from CGI server
to_cgi	Channel to CGI server
bank	Bank number

Description:

`fs_screenaddr` returns what would be the raster address of a physical screen if mapped to video memory bank `bank`. If called from the same processor as the CGI server this can be used for directly accessing the raster memory associated with a physical screen.

7.1.2 fs_displaybank

Display a video memory bank.

C:

```
void fs_displaybank(  
    Channel *to_cgi,  
    int bank )
```

occam:

```
PROC fs.displaybank(  
    CHAN OF ANY to.cgi,  
    VAL INT bank )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
bank	Bank number

Description:

`fs_displaybank` programs the graphics hardware to display a particular bank of video memory. The output subsequently generated on a monitor will correspond to the contents of the video memory bank identified by `bank`.

Physical CGI screens, which have their raster memory areas represented by video memory banks, are displayed in this way. See `fs_initscreen`.

The video memory bank displayed by `fs_displaybank` need not correspond to the current CGI drawing screen, see `cgi_setdrawscreen`.

7.1.3 fs_initscreen

Initialise a physical CGI screen.

C:

```
void fs_initscreen(  
    Channel *from_cgi, Channel *to_cgi,  
    screenptr s,  
    int bank )
```

occam:

```
PROC fs.initscreen(  
    CHAN OF ANY from.cgi, to.cgi,  
    [SCREEN.SIZE]INT s,  
    VAL INT bank )
```

Parameters:

Parameter	Comment
from_cgi	Channel from CGI server
to_cgi	Channel to CGI server
s	Screen
bank	Bank number

Description:

`fs_initscreen` creates and initialises a physical CGI screen ready for graphics operations. It is returned in `s`. The horizontal and vertical dimensions of the screen are determined by the graphics board display resolution, this is fixed when initialising the graphics board with `fs_openboard`. All physical screens have the same dimensions.

The physical screen has its raster memory area mapped to the video memory bank specified by `bank`. Depending on the display resolution and the total amount of video memory available, a variable number of video memory banks will be present. If the bank number is out of range then the screen returned will be mapped to bank zero (which is always available) and its X and Y axis dimensions set to zero. This renders the screen useless for normal CGI operations.

The screen can be made visible by displaying the video memory bank associated with it, see `cgi_displaybank`.

7.1.4 fs_setpalette

Set colour palette entry

C:

```
void fs_setpalette(
    Channel *to_cgi,
    int clutno, int red, int green, int blue )
```

occam:

```
PROC fs.setpalette(
    CHAN OF ANY to.cgi,
    VAL INT clutno, red, green, blue )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
clutno	Colour palette index
red	Red colour component
green	Green colour component
blue	Blue colour component

Description:

`cgi_setpalette` programs one entry in the colour palette. The CGI system uses a fixed size colour palette containing `maxPalette` colour entries. Each entry is 24 bits wide and consists of a red, a blue and a green component. The entry to program is given by `clutno` and the corresponding colour components by `red`, `green` and `blue`.

Colour component values range between 0 and 255. Small values indicate a low intensity and larger values a higher intensity.

The include files: `colours.h` and `colours.inc` contain red, green and blue colour component definitions for a number of interesting colours.

7.1.5 fs_openboard

Initialise a graphics board for use.

C:

```
void fs_openboard(  
    Channel *to_cgi,  
    VTG vtg )
```

occam:

```
PROC fs.openboard(  
    CHAN OF ANY to_cgi,  
    VAL [VTG.SIZE]INT vtg )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
vtg	Video timing parameters

Description:

`fs_openboard` initialises a graphics board. It causes the CGI display server to perform whatever device dependent actions are necessary to setup the graphics board ready for use, a graphics board must be opened before it can be used for displaying the output of CGI operations.

A single parameter is required: `vtg`. This should contain monitor and display resolution specific video timing parameters to initialise the CVC on the graphics board. It is important that these parameters match the capabilities of an attached monitor. Chapter 5 has a more detailed description of this. The include files: `video.h` and `video.inc` contain a number of constant video parameter block definitions that may be applicable.

In normal circumstances the control register field of the parameter block should be set to zero. This will cause the device dependent library associated with a particular graphics board to program the CVC control register in a board specific way. This can be overridden by specifying a non-zero value to write to the control register. In ANSI C, the field is `vtg.control`, in OCCam it is `vtg[VTG.CONTROL]`.

7.1.6 fs_closeboard

Terminate use of a graphics board.

C:

```
void fs_closeboard( Channel *from_cgi, Channel *to_cgi )
```

occam:

```
PROC fs.closeboard( CHAN OF ANY from.cgi, to.cgi )
```

Parameters:

Parameter	Comment
from_cgi	Channel from CGI server
to_cgi	Channel to CGI server

Description:

`cgi_closeboard` performs whatever device dependent operations are necessary to terminate use of the graphics board. The actual actions taken will depend on the graphics hardware being used.

7.1.7 fs_writeregs

Write graphics board registers.

C:

```
void fs_writeregs(  
    Channel *to_cgi,  
    int n, int *registers, int *contents )
```

occam:

```
PROC fs.writeregs(  
    CHAN OF ANY to.cgi,  
    VAL INT n,  
    VAL []INT registers, contents )
```

Parameters:

Parameter	Comment
to_cgi	Channel to CGI server
n	Number of registers
registers	Register addresses
contents	Register contents

Description:

fs_writeregs causes the CGI display server to program graphics board registers. This allows full access to the hardware control registers of any graphics board in a device dependent way. **registers** should contain the addresses of the graphics board registers to program, they will be written with the contents of **contents**. The number registers to program is given by **n**.

8 ANSI C user guide

This chapter contains a user guide for ANSI C toolset developers. It provides all the information necessary to develop graphics software for a transputer system, incorporating an iq Systems graphics board, with the IMS F003C and an ANSI C toolset. It should be read in conjunction with the appropriate toolset documentation.

8.1 Toolset search path

The toolset search path `ISEARCH` should be setup to include the following directories:

- `drive:\F003C\CLIB`
- `drive:\F003C\BOARDS`

For example, with:

```
SET ISEARCH=C:\F003C\CLIB\ C:\F003C\BOARDS\
```

8.1.1 IMS F003C library and include files

The following libraries will then be on the search path:

Library	Purpose
<code>CGILIB.LIB</code>	ANSI C CGI library
<code>B419.LIB</code>	IMS B419 board support library
<code>B419A.LIB</code>	IMS B419 (G300A) board support library
<code>B437.LIB</code>	IMS B437 board support library

and the following header files:

Include file	Purpose
<code>cgilib.h</code>	CGI library prototypes
<code>cgitypes.h</code>	CGI constant and type definitions
<code>colours.h</code>	Colour definitions
<code>video.h</code>	Video timing parameters

8.2 Invoking the CGI display server

The CGI display server has the following entry point:

```
CgiServer( Process *p, Channel *to_cgi, Channel *from_cgi )
```

It must be invoked as a transputer process from a program running on a suitable graphics board. The channels `to_cgi` and `from_cgi` are used to connect the server to application software running on the same transputer, or on some other transputer located elsewhere in the network.

The CGI server can be used in the following ways:

- By starting it from a C program and allowing the same program to engage in graphics operations. This is a single processor example where the application software and the CGI server run in parallel on the same transputer.
- By moving the invocation of the CGI server into a separate program and using the toolset configuration tools to place programs on different transputers. This technique can be used to build single and multiprocessor applications.

8.2.1 Single processor, single program

In this example, the CGI server is started with `ProcRun` and the main program continues in parallel. The main program calls functions from the CGI library to interact with the server, it can subsequently stop the server by calling `cgi_terminate`.

```
#include <stdio.h>
#include <process.h>
#include <channel.h>

#include <cgilib.h>
#include <cgitypes.h>

int main()
{
    Process *cgi;
    Channel *to_cgi, *from_cgi;

    /* Allocate the CGI channels */

    to_cgi = ChanAlloc();
    from_cgi = ChanAlloc();

    if ( (to_cgi == NULL) || (from_cgi == NULL) )
    {
        printf( "Failed to allocate channel\n" );
        abort();
    }
    /* Allocate the CGI server process */
```

```
/* The CGI server stack size is given by CGI_STACK_SIZE
   from the header file <cgitypes.h>. The server requires two
   parameters: the "to_cgi" and "from_cgi" channels. */

cgi = ProcAlloc( CgiServer, CGI_STACK_SIZE, 2, to_cgi,
                from_cgi );

if ( cgi == NULL )
{
    printf( "Failed to allocate process\n" );
    abort();
}

/* Start the CGI server, the main program continues */

ProcRun( cgi );

/* Use functions from the CGI library to interact with the
   CGI server. The first initialises the CGI system, others
   are used to perform graphics operations. The CGI server
   can be terminated with "cgi_terminate". */

cgi_init( to_cgi ); /* Initialise the CGI system */

/* Open the graphics board and do lots of graphics ...
   close the graphics board when done */

/* Application finished, time to terminate the CGI server */

cgi_terminate( from_cgi, to_cgi );

}
```

8.2.2 Multiprocessor, multi program

This example has two programs running in parallel. One is responsible for running the CGI server and the other is an application which communicates with the server using placed transputer channels. The toolset configuration utilities are used to declare and place the programs, and the channels connecting them, onto the available hardware.

```
#include <stdio.h>
#include <process.h>
#include <channel.h>

#include <misc.h> /* For get_param() */

#include <cgilib.h>
#include <cgitypes.h>

int main()
{
    Process *cgi;
    Channel *to_cgi, *from_cgi;

    /* Get the CGI channels from the configuration environment,
       these may have been mapped onto transputer links and
       connected to another processor. Alternatively, they may
       connect this program to another program running on the
       same processor. */

    to_cgi = get_param(3);
    from_cgi = get_param(4); /* Defined by interface mapping */

    /* Allocate the CGI server process */

    /* The CGI server stack size is given by CGI_STACK_SIZE
       from the header file <cgitypes.h>. The server requires two
       parameters: the "to_cgi" and "from_cgi" channels. */

    cgi = ProcAlloc( CgiServer, CGI_STACK_SIZE, 2, to_cgi,
                    from_cgi );

    if ( cgi == NULL )
    {
        printf( "Failed to allocate process\n" );
        abort();
    }

    /* Start the CGI server and wait until it is terminated by
       the application software, this program will then terminate
       as well */

    ProcPar( cgi, NULL );
}
```


This could be simplified by calling the CGI server function inline. If this is done the process descriptor parameter `Process *p` must be passed to the function explicitly. It can be set to any value, for example:

```
#include <stdio.h>
#include <channel.h>

#include <misc.h> /* For get_param() */

#include <cgilib.h>
#include <cgitypes.h>

int main()
{
    Channel *to_cgi, *from_cgi;

    /* Get the CGI channels from the configuration environment,
       these may have been mapped onto transputer links and
       connected to another processor. Alternatively, they may
       connect this program to another program running on the
       same processor. */

    to_cgi = get_param(3);
    from_cgi = get_param(4); /* Defined by interface mapping */

    /* Start the CGI server and wait until it is terminated by
       the application software, this program will then terminate
       as well */

    CgiServer( NULL, to_cgi, from_cgi );
}
```

8.3 Configuring transputer memory sizes

The amount of memory available on a transputer for program storage is specified by `IBOARDSIZE` (single transputer system) or by a configuration description. When specifying the amount of memory available on a graphics board with contiguous DRAM and VRAM, care should be taken to ensure that program code or data is not assigned to VRAM that will be used for graphics operations.

8.4 Opening the graphics board

Before any output can be displayed on a monitor the graphics board must be initialised. This is done by calling `fs_openboard` with a suitable set of video timing parameters. Parameters for varying display resolutions and different monitor types are provided in the include file `video.h`. The following example shows the typical steps taken by an application program during initialisation:

```
#include <stdio.h>
#include <process.h>
#include <channel.h>

#include <cgilib.h>
#include <cgitypes.h>

#include <video.h> /* For video timing parameters */

int main()
{
    /* Start the CGI server and allocate channels to it,
       these are "to_cgi" and "from_cgi". Alternatively, the
       CGI server may already be running in another program */

    /* Declare and initialise a video timing parameter block,
       V_1024_768 is defined in <video.h> and specifies a set
       of parameters for a 1024 by 768 pixel display. */

    VTG v = v_1024_768

    screen s; /* A CGI screen */

    /* Initialise the graphics board */

    fs_openboard( to_cgi, v );

    /* Initialise a physical screen and map it to video
       ram bank 0. */

    fs_initscreen( from_cgi, to_cgi, &s, 0 );

    /* Set the current drawing screen to s and display it
       on the output monitor. */

    cgi_setdrawscreen( to_cgi, s ); /* Now drawing in s */

    fs_displaybank( to_cgi, 0 ); /* Bank 0 now displayed */

    /* CGI drawing operations will now be displayed ... */
}
```

8.5 Compiling and linking IMS F003C programs

8.5.1 Compiling

There are no special compilation requirements for programs that use the IMS F003C libraries.

A default font can be enabled by including the header file `cgitypes.h` and compiling with the preprocessor flag `_FONT` defined. For example, with:

```
icc example.c /t8 /D_FONT
```

This uncomments an `unsigned int` array called `font8by8` that contains a simple font definition for use with `cgi_setfont`.

8.5.2 Linking

Software calling functions from the CGI library should be linked against `CGI-LIB.LIB`.

Programs which invoke the CGI server must be linked with one of the board support libraries.

8.6 Example program

The directory `\F003C\CLIB\EXAMPLES` contains an example program. It is designed to run on a single transputer configuration: one of the *iq* Systems graphics boards.

To build and run it:

```
icc example.c /ta /D_FONT
```

```
ilink example.tco cgilib.lib board.lib /f startup.lnk /ta
```

(Where *board* is the name of a specific graphics board).

```
icollect example.lku /t
```

```
iserver /se /sb example.btl
```


9 occam user guide

This chapter contains a user guide for OCCAM toolset developers. It provides all the information necessary to develop graphics software for a transputer system, incorporating an iq Systems graphics board, with the IMS F003C and an OCCAM toolset. It should be read in conjunction with the appropriate toolset documentation.

9.1 Toolset search path

The toolset search path `ISEARCH` should be setup to include the following directories:

- `drive:\F003C\OCCAMLIB`
- `drive:\F003C\BOARDS`

For example, with:

```
SET ISEARCH=C:\F003C\OCCAMLIB\ C:\F003C\BOARDS\
```

9.1.1 IMS F003C library and include files

The following libraries will then be on the search path:

Library	Purpose
<code>CGILIB.LIB</code>	occam CGI library
<code>LIBCRED.LIB</code>	Reduced C runtime library
<code>B419.LIB</code>	IMS B419 board support library
<code>B419A.LIB</code>	IMS B419 (G300A) board support library
<code>B437.LIB</code>	IMS B437 board support library

and the following header files:

Include file	Purpose
<code>cgilib.inc</code>	CGI constant definitions
<code>colours.inc</code>	Colour definitions
<code>video.inc</code>	Video timing parameters

9.2 Invoking the CGI display server

The reader may need to refer to the chapter entitled *Mixed language programming* in the appropriate OCCAM toolset user manual when reading this section.

The CGI display server is implemented as a C function, when calling it from an OCCAM program it has the following entry point:

```
CgiServer( VAL INT gsb, p, CHAN OF ANY to.cgi, from.cgi )
```

It must be invoked by a program running on a suitable graphics board. The channels `to.cgi` and `from.cgi` are used to connect the server to application software running on the same transputer, or on some other transputer located elsewhere in the network.

Because the CGI server is implemented in C it requires the invoking OCCAM program to setup a C environment. Support for this is provided by the OCCAM toolset with the `CALLC.LIB` library. This contains a number of procedures for setting up and initialising a C function call from OCCAM.

The CGI server requires a static and a heap area. These are allocated from an `INT` array declared by the calling OCCAM program. The array must be big enough to hold the static variables used by the CGI server and provide enough space for a heap. The heap is used to allocate dynamic storage for loadable fonts (see `cgi.setfont`), it should be large enough to hold the biggest font required by an application. The static space required by the CGI server is constant and can be satisfied with a 5000 word `INT` array, additional space in the array will be used for the heap. The workspace requirement of the CGI server is specified by a compiler `#PRAGMA` in the include file `cgilib.inc`.

The CGI server can be used in the following ways:

- By running it in parallel with an application contained in the same program. This is a single processor example where the application software and the CGI server run in parallel on the same transputer.
- By moving the invocation of the CGI server into a separate program and using the toolset configuration tools to place programs on different transputers. This technique can be used to build single and multiprocessor applications.

9.2.1 Single processor, single program

In this example, the CGI server is run in parallel with the application from a single program. The application calls procedures from the CGI library to interact with the server, it can subsequently stop the server by calling `cgi.terminate`.

```

#USE "hostio.lib"
#include "hostio.inc"

#USE "callc.lib" -- occam toolset library

#USE "cgllib.lib"
#include "cgllib.inc"

PROC example( CHAN OF ANY fs, ts, [ ]INT free.mem )

INT gsb, static.size: -- For the static area
CHAN OF ANY to.cgi, from.cgi:

SEQ

-- Determine the exact amount of static space required by
-- the CGI server.

init.static( [free.mem FROM 0 FOR 0], static.size, gsb )
IF
  ( static.size > (SIZE free.mem) )
  SEQ
    so.write.string.nl( fs, ts, "No memory for CGI statics" )
    so.exit( fs, ts, sps.failure )
    CAUSEERROR() -- Stop the transputer
  TRUE
  SKIP

-- Abbreviate the static and heap areas from the free.mem
-- INT array. "static.size" gives the amount of static space
-- required. The rest of free.mem is used as heap space.

static.area IS [free.mem FROM 0 FOR static.size]:
heap.area IS [free.mem FROM static.size FOR
              (SIZE free.mem) - static.size]:

SEQ

-- Initialise the static and heap areas.

INT unused.size: -- Don't need the size
init.static( static.area, unused.size, gsb )
init.heap( gsb, heap.area )

-- Run the CGI server in parallel with application software

PAR

  CgiServer( gsb, 0, to.cgi, from.cgi )

  SEQ

    -- Use procedures from the CGI library to interact with the
    -- CGI server. The first initialises the CGI system, others are
    -- used to perform graphics operations. The CGI server can be
    -- terminated with "cgi.terminate".

    cgi.init( to.cgi ) -- Initialise the CGI system

    -- Open the graphics board and do lots of graphics ...
    -- close the graphics board when done

    -- Application has finished, time to terminate the CGI server

    cgi.terminate( from.cgi, to.cgi )

```

9.2.2 Multiprocessor, multi program

This example has two programs running in parallel. One is responsible for running the CGI server and the other is an application which communicates with the server using placed transputer channels. The toolset configuration utilities are used to declare and place the programs, and the channels connecting them, onto the available hardware.

```
#USE "hostio.lib"
#include "hostio.inc"

#USE "callc.lib" -- occam toolset library

#USE "cgilib.lib"
#include "cgilib.inc"

PROC example( CHAN OF ANY fs, ts, from.cgi, to.cgi )

  INT gsb, static.size: -- For the static area

  [5000]INT static.area: -- Enough for the CGI server
  [4000]INT heap.area: -- Enough for an interesting font

  SEQ

  -- The CGI channels come from the configuration environment,
  -- these may have been mapped onto transputer links and
  -- connected to another processor. Alternatively, they may
  -- connect this program to another program running on the
  -- same processor.

  -- Initialise the static and heap areas

  init.static( static.area, static.size, gsb )
  init.heap( gsb, heap.area )

  -- Start the CGI server and wait until it is terminated by
  -- the application software, this program will then terminate
  -- as well.

  CgiServer( gsb, 0, to.cgi, from.cgi )

:
```

9.3 Configuring transputer memory sizes

The amount of memory available on a transputer for program storage is specified by IBOARDSIZE (single transputer system) or by a configuration description. When specifying the amount of memory available on a graphics board with contiguous DRAM and VRAM, care should be taken to ensure that program code or data is not assigned to VRAM that will be used for graphics operations.

9.4 Opening the graphics board

Before any output can be displayed on a monitor the graphics board must be initialised. This is done by calling `fs.openboard` with a suitable set of video timing pa-

rameters. Parameters for various display resolutions and different monitor types are provided in the include file `video.inc`. The following example shows the typical steps taken by an application program during initialisation:

```
#USE "cgilib.lib"
#include "cgilib.inc"

#include "video.inc" -- For video timing parameters

PROC example( CHAN OF ANY from.cgi, to.cgi )

-- Start the CGI server and declare channels to it,
-- these are "to.cgi" and "from.cgi". Alternatively, the
-- CGI server may already be running in another program.

-- Declare and initialise a video timing parameter block,
-- V.1024.768 is defined in video.inc and specifies a set
-- of parameters for a 1024 by 768 pixel display.

[VTG.SIZE]INT v:
[SCREEN.SIZE]INT s: -- A CGI screen

SEQ

  v := V.1024.768

  -- Initialise the graphics board

  fs.openboard( to.cgi, v )

  -- Initialise a physical screen and map it to video
  -- ram bank 0.

  fs.initscreen( from.cgi, to.cgi, s, 0 )

  -- Set the current drawing screen to s and display it
  -- on the output monitor.

  cgi.setdrawscreen( to.cgi, s ) -- Now drawing in s

  fs.displaybank( to.cgi, 0 ) -- Bank 0 now displayed

  -- CGI drawing operations will now be displayed ...

:
```

9.5 Compiling and linking IMS F003C programs

9.5.1 Compiling

There are no special compilation requirements for programs that use the IMS F003C libraries.

9.5.2 Linking

Software calling procedures from the CGI library should be linked against `CGI-LIB.LIB`.

Programs which invoke the CGI server must be linked with one of the board support libraries and also the OCCAM toolset mixed language support library: `CALLC.LIB`. In addition, the reduced C runtime library `LIBCRED.LIB` is also required. This file is normally supplied with an ANSI C toolset. However, because most OCCAM developers will probably not have a C toolset the library is also supplied with the IMS F003C software. It can be found in `\F003C\OCCAMLIB`

9.6 Example program

The directory `\F003C\OCCAMLIB\EXAMPLES` contains an example program. It is designed to run on a single transputer configuration: one of the iq Systems graphics boards.

To build and run it, type:

```
oc example.occ /ta
ilink example.tco cgilib.lib board.lib callc.lib
      hostio.lib convert.lib libcred.lib /f occama.lnk /ta
```

(Where *board* is the name of a specific graphics board).

```
icollect example.lku /t
iserver /se /sb example.btl
```

10 Further use of the CGI system

This chapter contains more detailed information concerning the use of various aspects of the CGI system.

10.1 Using and defining text fonts

Text fonts are downloaded to the CGI server with `cgi_setfont`. This defines bit-maps for the various character cells that make up the font. Because the CGI system uses heap space to hold a font definition it should be invoked with enough heap memory available to hold the largest font to be used. Only one font is held by the CGI server at a time, if an application requires the use of multiple fonts then it will have to load each one as and when needed.

A default font is supplied with the IMS F003C software. It contains a fixed size ASCII character set defined within an eight by eight pixel character cell. By including the file `cgilib.inc`, OCCAM programmers will have access to a `VAL []INT` array called `FONT.8.BY.8` which contains the font definition. ANSI C programmers should include `cgitypes.h` which if compiled with the preprocessor variable `_FONT` defined will un-comment an `unsigned int` array `font8by8` that contains an equivalent font.

When downloading a font, the `cgi_setfont` function requires various font characteristics to be defined. These specify, for example, the number of 32 bit words used to hold the bit pattern of a single character cell. There are four parameters required to define a font:

Font parameter	Purpose
<code>famw</code>	Font area memory width (in bits)
<code>fwpc</code>	Number of 32 bit words per character
<code>flpw</code>	Number of character lines per 32 bit word
<code>nchars</code>	Number of characters in the font

The default font is supplied with definitions for these values, for example, in C they are: `font_FAMW`, `font_FWPC`, `font_FLPW` and `font_NCHARS`.

If necessary, the programmer can define additional fonts or perhaps convert existing fonts from some other environment into this format. The following example shows how the font parameters relate to the bit mask used to represent each character cell defined by the font.

In the supplied 8 by 8 pixel font, the character 0 is represented by the following bit mask:

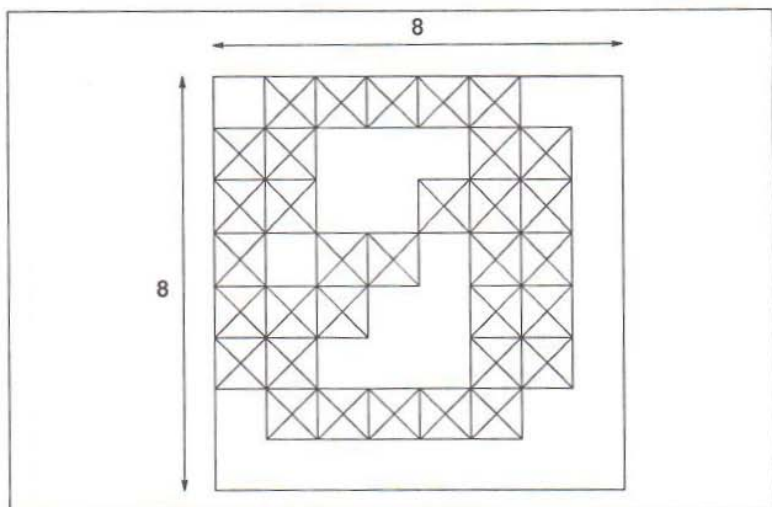


Figure 10.1 Character '0' representation in font 8 by 8

This is stored in two 32 bit words: `0xd6cec67c`, which describes the top half of the character cell, and `0x007cc6e6` for the bottom half. The origin of the character cell is defined to be the top left hand corner. The first word defines lines in horizontal row order, starting with the least significant bit. In this example, the least significant byte of the first word is `0x7c`, this represents the first row of the character cell with the bit mask `01111100`.

The complete font is represented by an array of 32 bit words, each pair of words is used to encode the definition of a single character. The byte value of a character is used as an index into this array when retrieving a character definition in order to plot it. The 8 by 8 font is defined by the following font parameters:

Font parameter	Value
famw	8 pixels wide
fwpc	2 x 32 bits per character
flpw	4 x character lines per 32 bit word
nchars	164 characters

10.2 Using CGI screens for windowing

The CGI screen abstraction can be used to form the basis of a windowing like, graphical user interface. Such interfaces typically use two dimensional screen areas to represent objects such as popup menus, dialogue boxes or text windows. These objects can all be implemented using facilities provided by the CGI library.

The CGI screen structure describes a two dimensional region of raster memory that the CGI system performs graphical operations within. The size of the screen defines the extent of drawing operations: drawing is clipped to the boundary of the screen. There are two types of screen. The first has a raster stored in normal memory and can never be displayed on a monitor, the second type is designed to be displayed on a monitor and has a raster stored in video memory. its screen dimensions match the resolution of the monitor. CGI primitives for copying, scaling or rotating screens can be used to copy a part of one screen to another.

An existing screen structure can be used to create another. If the new screen refers to an existing, but smaller area of the original, then it can be used to represent a window. When selected as the current screen, the CGI system will clip all further drawing operations to its extent. This will create the effect of drawing in a bounded window, the background will be protected. By combining this with the CGI copy or rotation primitives, simple windowing can be implemented. In the example, the background area would have to be copied elsewhere while the window is manipulated and then copied back again to restore it.

The following example demonstrates some of these techniques:

```
#include <stdio.h>
#include <math.h>
#include <math.h>
#include <stdlib.h>
#include <channel.h>
#include <process.h>

#include <cgilib.h>
#include <video.h>
#include <colours.h>

/*-----
 * sub_screen - create a sub screen from an existing screen
 *-----
 */

void sub_screen( screen *new, screen old,
                int xorg, int yorg,
                int xsize, int ysize )
{
    /* Ensure that the new screen fits on the old one */
```

```
if ( (xorg >= old.xsize) || (yorg >= old.ysize) )
    return;

/* Clip the new screen dimensions to the extent of the
old screen. The stride must remain the same as the old
screen because the new raster is not contiguous. */

new->raster = old.raster + (yorg * old.xsize) + xorg;
new->xsize = xorg + xsize > old.xsize ? old.xsize - xorg : xsize;
new->ysize = yorg + ysize > old.ysize ? old.ysize - yorg : ysize;
new->stride = old.stride;
new->multiMode = old.multiMode;
}

/*-----
 * main
 *-----
 */

int main ()
{
    Process *cgi;
    Channel *to_cgi, *from_cgi;

    VTG v = V_1024_768;

    screen s1, s2; /* Declare two screen structures */

    /* Allocate channels to the CGI server */

    to_cgi = ChanAlloc();
    from_cgi = ChanAlloc();

    if ( (to_cgi == NULL) || (from_cgi == NULL) )
    {
        printf( "Failed to allocate channel\n" );
        abort();
    }

    /* Allocate and start the CGI server */

    cgi = ProcAlloc( CgiServer, CGI_STACK_SIZE, 2, to_cgi, from_cgi );
    if ( cgi == NULL )
    {
        printf( "Failed to allocate process\n" );
        abort();
    }
    ProcRun( cgi );

    /* Initialise the CGI server and open the graphics board */

    cgi_init( to_cgi );
    fs_openboard( to_cgi, v );

    /* Program some simple colours into the palette */
}
```

```

fs_setpalette( to_cgi, 2, LINEN_R, LINEN_G, LINEN_B );
fs_setpalette( to_cgi, 3, SKYBLUE_R, SKYBLUE_G, SKYBLUE_B );

/* Initialise a physical screen and display it */

fs_initscreen( from_cgi, to_cgi, &s1, 0 );
printf( "Screen initialised\nraster = 0x%x xsize = %d ysize %d\n",
        (int)(s1.raster), s1.xsize, s1.ysize );
fs_displaybank( to_cgi, 0 );

/* Allocate a new screen, derived from s1, which represents a
   window like viewport into the original screen. */

sub_screen( &s2, s1, 100, 200, 200, 200 );

/* Clear both screens. Note the order in which this is done.
   The background screen s1 is cleared first, the window screen
   s2 is cleared afterwards. */

cgi_cls( to_cgi, s1, 2 ); /* Clear background to LINEN */
cgi_cls( to_cgi, s2, 3 ); /* Clear window screen to SKYBLUE */

/* Select the window screen as the current drawing screen and
   write some text into it. This could be used as a popup window
   or a menu selection etc. */

cgi_setdrawscreen( to_cgi, s2 );

cgi_setfcol( to_cgi, 2 ); /* Drawing colour is now LINEN */
cgi_setbcol( to_cgi, 0 ); /* Background colour is 0 */

/* Down load a font and initialise text attributes */

cgi_chrspace( to_cgi, 10, 0 );
cgi_setfont( from_cgi, to_cgi, font8by8,
             font_NCHARS, font_FAMW, font_FWPC, font_FLPW );
cgi_setorient( to_cgi, TX_NORM );

/* A pixel replace mode of move non-zero will cause text to
   be written in the current foreground colour while ignoring
   the background (because the background colour is now 0). */

cgi_setdrawmode( to_cgi, FM_COL, RM_NZ, FM_COL );

/* Print some text in the window, note that it will be clipped
   to the extent of s2 */

cgi_text( to_cgi, 20, 20, 13, "Hello World !" );

/* Close the graphics board and terminate the CGI server */

fs_closeboard( from_cgi, to_cgi );
cgi_terminate( from_cgi, to_cgi );
}

```

10.3 Simple animation techniques

Provided a graphics board has enough video memory to support more than one physical CGI screen, simple animation can be achieved. This is done by cycling the graphics board hardware through the available screens, displaying each in turn, whilst changing the contents of the screen just about to be displayed.

For example, a simple cube like object could be made to continuously spin around some axis of rotation. To do this, the cube would first have to be drawn at a starting position and displayed. Meanwhile, the CGI system would be instructed to draw a second cube in another, invisible, screen. It would be drawn with a small physical displacement from the first cube. When the second cube is complete the displayed screen and the invisible screen are toggled: the displayed screen becomes invisible, and the screen with the new cube, visible. If this process is continued an animation effect can be achieved as the cube continuously moves around on the display.

The technique is best demonstrated with an example. The following program uses a pair of physical CGI screens to animate a rotating disk, note the use the `cgi_fcircle` function which has its axis parameters altered after drawing every new circle, this combines to produce a three dimensional effect.

```

/*-----
 * main
 *-----
 */

int main ()
{
    Process *cgi;
    Channel *to_cgi, *from_cgi;

    /* Declare two screens in an array, bank will be used to
       alternate which screen is displayed, and which screen is
       drawn into. */

    screen s[2];
    int axis, bank, step;

    VTG v = V_1024_768;

    /* Allocate channels to the CGI server */

    to_cgi = ChanAlloc();
    from_cgi = ChanAlloc();

    if ( (to_cgi == NULL) || (from_cgi == NULL) )
    {
        printf( "Failed to allocate channel\n" );
        abort();
    }

    /* Allocate and start the CGI server */

    cgi = ProcAlloc( CgiServer, CGI_STACK_SIZE, 2, to_cgi, from_cgi );
    if ( cgi == NULL )
    {
        printf( "Failed to allocate process\n" );
        abort();
    }
}

```



```

}
ProcRun( cgi );

/* Initialise the CGI server and open the graphics board */
cgi_init( to_cgi );
fs_openboard( to_cgi, v );

/* Initialise a pair of physical screens */
fs_initscreen( from_cgi, to_cgi, &s[0], 0 );
printf( "Screen initialised\nraster = 0x%x xsize = %d ysize %d\n",
        (int)s[0].raster, s[0].xsize, s[0].ysize );

fs_initscreen( from_cgi, to_cgi, &s[1], 1 );
printf( "Screen initialised\nraster = 0x%x xsize = %d ysize %d\n",
        (int)s[1].raster, s[1].xsize, s[1].ysize );

/* Setup the palette with some simple colours */
fs_setpalette( to_cgi, 0, LINEN_R, LINEN_G, LINEN_B );
fs_setpalette( to_cgi, 1, YELLOW_R, YELLOW_G, YELLOW_B );

cgi_setfcol( to_cgi, 1 ); /* Drawing colour is YELLOW */

cgi_cls( to_cgi, s[0], 0 );
cgi_cls( to_cgi, s[1], 0 ); /* Clear both screens to LINEN */

axis = 0;
step = 5; /* Axis dimensions change in a step of 5 pixels */

/* Initially, screen[0] is drawn into, and screen[1]
   is displayed. bank is used to index each screen from the
   screen array. */
bank = 0;

fs_displaybank( to_cgi, bank ^ 1 );
cgi_setdrawscreen( to_cgi, s[bank] );

cgi_fcircle( to_cgi, 500, 350, axis, 200 - axis );

bank ^= 1; /* Toggle bank */
axis += step; /* Step the circle axis */

while (1) /* Do this continuously */
{
    fs_displaybank( to_cgi, bank ^ 1 );
    cgi_setdrawscreen( to_cgi, s[bank] );

    /* Wipe the old circle from the screen by clearing it,
       and draw a new circle. Alter the circle axis step if
       the axis has reached the end of its range. */

    cgi_cls( to_cgi, s[bank], 0 );
    cgi_fcircle( to_cgi, 500, 350, axis, 200 - axis );

    if ( (axis == 200) || (axis == 0) ) step = -step;

    bank ^= 1; /* Toggle bank */
    axis += step; /* Step the circle axis */
}
}

```

10.4 Writing a board support library

The source code of board support libraries for *iq* Systems graphics board products supported by the IMS F003C software is supplied in the directory: \F003C\BOARDS\SOURCE. This should allow a new version of a board support library to be created for some other transputer based graphics board.

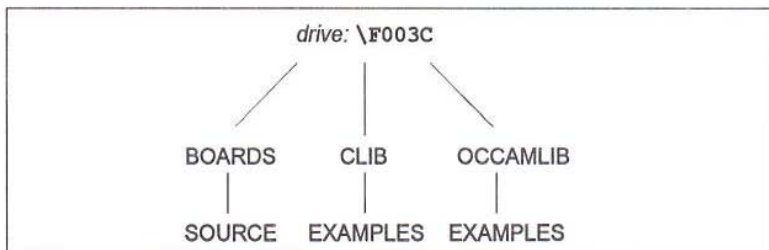
The interface required by the CGI server defines the functions that must be provided by a board support library. They are:

Function	Purpose
FS_SCREENADDR	Return the address of a physical screen
FS_DISPLAYBANK	Display a bank of video memory
FS_INITSCREEN	Initialise a screen, map it to video memory
FS_SETPALETTE	Program a colour palette location
FS_OPENBOARD	Do device specific initialisation
FS_CLOSEBOARD	Do device specific termination
FS_WRITEREGS	Write board control registers

The source code is well commented and should contain all the information necessary to port it to another graphics board.

A Directory structure

IMS F003C files are installed within the following directory structure:-



Which, after a successful installation, should contain the following files:-

drive: \F003C\CLIB

CGILIB.LIB
CGILIB.H
CGITYPES.H
COLOURS.H
VIDEO.H

drive: \F003C\CLIB\EXAMPLES

EXAMPLE.C

drive: \F003C\OCCAMLIB

CGILIB.LIB
LIBCRED.LIB
CGILIB.INC
COLOURS.INC
VIDEO.INC

drive: \F003C\OCCAMLIB\EXAMPLES

EXAMPLE.OCC

drive: \F003C\BOARDS

B419.LIB
B419A.LIB
B437.LIB

drive: \F003C\BOARDS\SOURCE

B419.C
B437.C
FSTORE.H
FSTOREI.H

B IMS B419 hardware overview

B.1 Description

The IMS B419 combines the IMS G300B Colour Video Controller (CVC) with the IMS T800 32 bit Floating Point Transputer to form a high performance graphics system. Two Mbytes of four cycle DRAM provides a general purpose store sufficient to run large applications such as windowing environments. Two Mbytes of Video RAM provide arbitrary screen resolutions up to a maximum of 1280 x 1024 8 bit/pixel with unrestricted screen formats at resolutions below this.

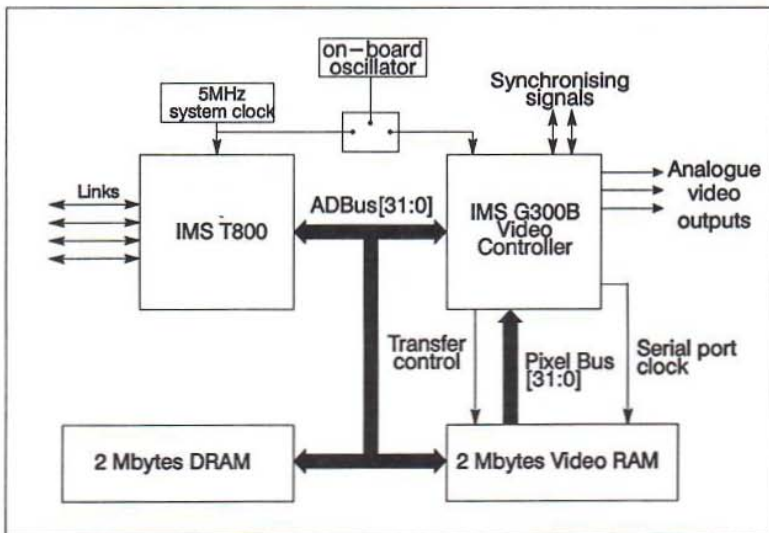


Figure B.1 Block diagram

B.1.1 Introduction

The IMS B419 is one of a range of INMOS TRANsputer Modules (TRAMs). TRAMs are board level transputers with a simple, standardised interface. They integrate processor, memory and peripheral functions allowing powerful, flexible, transputer based systems to be produced with the minimum of design effort¹.

¹Further details of the TRAM/motherboard philosophy and the full electrical and mechanical specification of TRAMs can be found in technical notes *Dual-In-Line Transputer Modules (TRAMs)* and *Module Motherboard Architecture* which are included later in this databook. The *Transputer Databook* may also be required. This is available as a separate publication from INMOS.

The IMS B419 implements a complete high performance graphics subsystem. The frame store consists of 2 Mbytes of dual ported Video RAM which supports displays of arbitrary resolution at 8 bit/pixel. The resolution of the system is programmable and is only limited by the CVCs maximum dot rate (100MHz). The CVC is configured by an IMS T800 which is provided with 2 Mbytes of 200ns cycle DRAM. This store is available for screen manipulation workspace and general program memory. The processor can be used to implement graphic primitives directly or as an intelligent channel, receiving image data from other transputers via its four bidirectional links at data rates of up to 6.8 Mbytes/sec. This makes the IMS B419 useful for applications from acting as part of an embedded system in industrial control, to a graphics output for a 3D graphical supercomputer.

B.1.2 Screen sizes

Screen sizes are set by writing to a few registers in the G300B CVC, and can be chosen to suit the application. Suppose, for instance, an 8.5 x 11 sheet of paper (in landscape), represented by a screen with 100 pixels per inch. This would need an 1100 x 850 display, a format not normally available from a hardware solution. The G300B gives a line width in multiples of 4 pixels, which makes it simple to produce this screen. As well as producing special screens such as 11 x 8.5, many of the standard screens can also be produced; indeed the user can switch between screen formats, the display clock frequency, and even the source of the input clock, all by simply changing the G300B registers and other registers on the board by software.

Some examples of possible screen sizes are given in Table B.1. All the screens in the table are for 8 bits per pixel.

Screen Size	Pixels	Aspect Ratio	Interlace
CGA	320 x 240	1.333	no
EGA	640 x 350	1.829	no
VGA	640 x 480	1.333	no
Enh VGA	800 x 600	1.333	no
Ext VGA	1024 x 768	1.333	no
11 x 8.5	1100 x 850	1.294	no
11 x 8.5	1164 x 900	1.293	no
	1024 x 1024	1.0	no
	1280 x 1024	1.25	no
A5	1216 x 860	1.414	no

Table B.1 A selection of possible screen sizes

B.1.3 SubSystem signals

The user may require the G300B Graphics TRAM to control a network of transputers and/or other TRAMs. A set of control signals are provided which enables the

master to control these slaves or subsystems. The SubSystem port consists of three signals: **SubSystemReset** and **SubSystemAnalyse**, which enables the master to reset and analyse its subsystem; and **SubSystemnotError**, which is used to monitor the error flag in the subsystem. These signals are accessible to the processor as a set of memory-mapped registers.

Register	Hardware byte address	Asserted state
SubSystemReset (Wr only)	#00000000	1
SubSystemAnalyse (Wr only)	#00000004	1
SubSystemError (Rd only)	#00000000	1

Table B.2

B.1.4 Memory Map

The video memory (VRAM) on the IMS B419 can be arranged to be either contiguous with the DRAM or separately mapped. The IMS F003C requires that the VRAM must be contiguous with the DRAM; so JP4 must be fitted, and JP5 removed when the IMS B419 is installed. The resulting memory map is shown in Figure B.2.

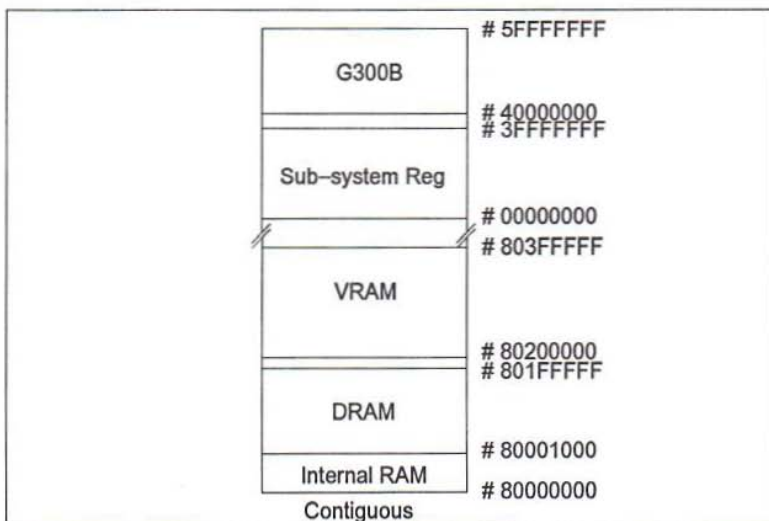


Figure B.2 Address map

Users are advised not to access the IMS G300B directly, but to use the routines provided by the IMS F003C.

B.1.5 Pixel clock selection

The IMS G300B requires a clock to control the movement of pixel data, and generate timing signals. It has a phase-locked loop (PLL) which can generate the high frequency pixel clock from a low frequency input clock. The PLL can generate frequencies from 25MHz upwards.

The IMS B419 provides a choice of clocking schemes which are described in detail in the *IMS B419 hardware reference guide*. The IMS F003C uses the 5MHz TRAM clock in conjunction with the IMS G300's on-chip phase locked loop. This allows the use of any clock frequency which is a multiple of 5MHz from 25MHz — 100MHz. If any other clock frequency is required, the nearest multiple of 5MHz should be used. This has been found to give satisfactory results with all types of video monitor, and screen resolution.

B.1.6 Jumper selection

Five jumper links are used to select the IMS G300B clock source and to configure the memory map of the IMS B419. Jumpers are labelled JPx where a jumper is either installed or absent between two pin posts.

Jumper	Function
JP1	Always remove on IMS B419-4
JP2	Do not fit
JP3	Always fit
JP4	Select contiguous VRAM
JP5	Select non-contiguous VRAM

Table B.3

For IMS F003C compatibility, JP4 must be fitted and JP5 removed.

B.2 Board layout

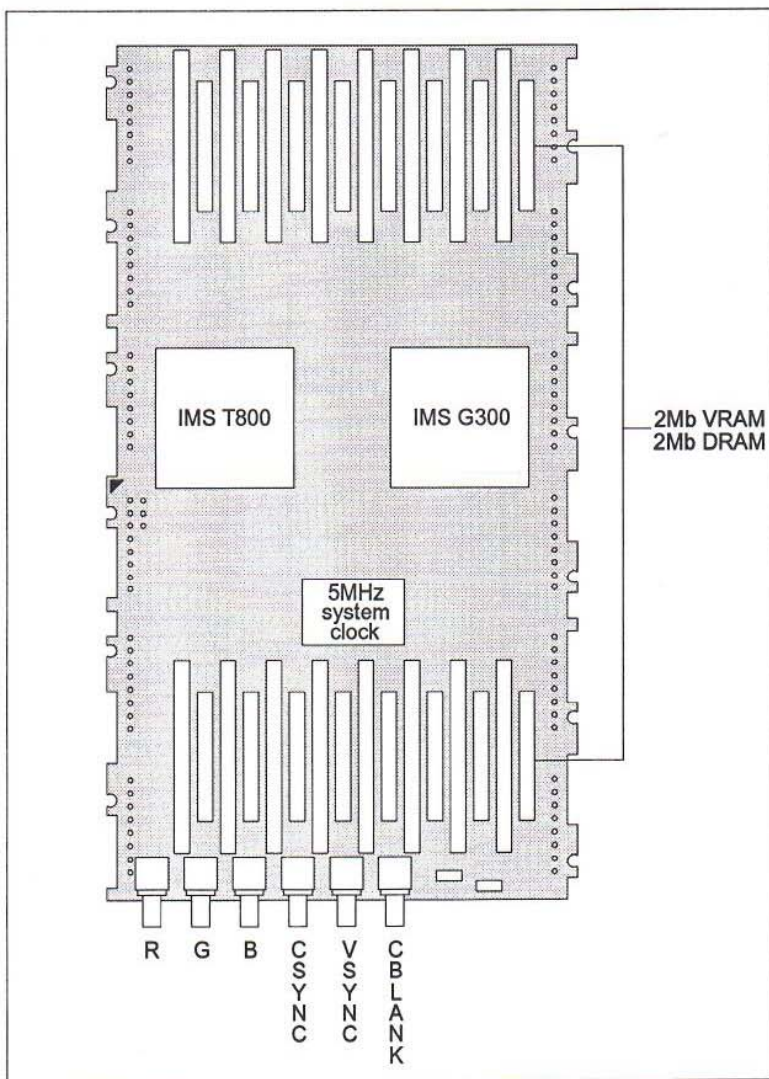


Figure B.3

B.2.1 Video and sync outputs

The G300B CVC can be programmed to generate timing which complies with both the RS170a and EIA-343 video standard. The outputs are designed to drive a 75R line directly. The RGB analogue outputs and synchronising signals are on five SMB connectors as shown below. If the display monitor accepts composite sync on one of its video inputs the sync outputs may be left unconnected. SMB identification from top to bottom of the board. Sync. information is output on all three video signals.

1 Composite blank	Input/Output
2 Vertical Sync	Output
3 Composite or Horizontal Sync	Output
4 Blue	Output 75R
5 Green	Output 75R
6 Red	Output 75R

C IMS B437 hardware overview

C.1 Description

The IMS B437 consists of an IMS T805 transputer; with 1Mbyte of dual port video RAM which is directly addressed by the transputer, and an IMS G332 colour video controller which is connected to the serial ports of the video RAMs.

The IMS G332 can be programmed by the transputer to generate almost any required video timing and display resolution; the only restriction being that maximum clock frequencies and memory size limits are not exceeded. Because of its ability to drive many types of display monitor at a wide range of resolutions, the IMS B437 is suitable for a variety applications. It is able to generate high resolution displays, VGA-type displays and TV standard images with correct sync patterns and interlacing. The 15 and 16 bit/pixel true colour modes provide highly realistic colour rendition.

The IMS B437 can be used with the IMS B429 to build a high performance image processing system, which fits on a single IMS B008 PC add-in card or IMS B014 VME card. It is also suitable for use in any transputer application where graphical output is required and space is limited.

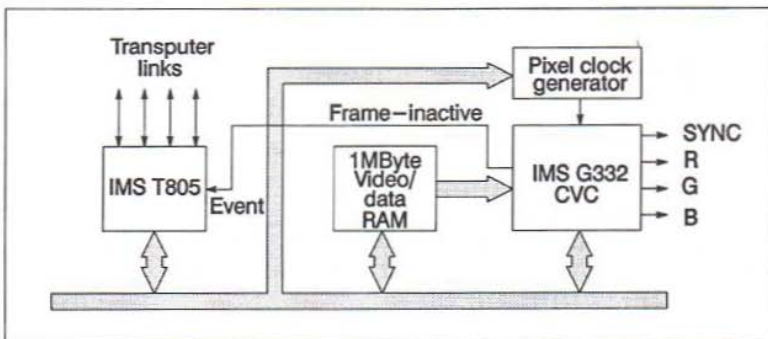


Figure C.1 Block diagram

C.2 Memory map

Feature	Address
DRAM (1Mb)	0x80001000 - 0x80100FFF
IMS G332	0x00000000 - 0x3FFFFFFF
Board control	0x40000000 - 0x4000001F

Table C.1 IMS B437 Memory Map

Users are advised that it is not necessary to write to the IMS G332 or the board control registers directly. They should use the IMS F003 to program the IMS G332.

C.3 Display formats

Pixel widths can be 1,2,4,8,15 or 16 bits. The 1,2,4, and 8 bit modes are pseudo-colour modes which use the IMS G332's Colour Look-Up Table to select from a much larger colour space (8 bits/DAC). The 15 and 16 bit modes drive the DACs directly (but use the Look-Up Tables to perform gamma-correction). In the 15 and 16 bit modes, the pixel clock speed is limited to a maximum of 50MHz. This is more than adequate for producing full colour displays at VGA and TV standard resolutions. The required pixel width is selected by programming the IMS G332, and board control register 5. Pixels are displayed from consecutively addressed words in memory, starting at the address specified by TopOfScreen/LineStart. The format of pixels within each word is that the pixels are output to the screen starting from the least-significant end of the word.

C.4 Colour video controller

The IMS B437 uses an IMS G332 Colour Video Controller. This device generates fully programmable video timing which allows the IMS B437 to drive a wide variety of display monitors with a wide variety of display resolutions. Examples of typical formats which are supported by the IMS B437, and the amount of memory remaining for program use, are:

- 1 screen of 1024 × 768 by 8 bit pixels, 256k program space
- 3 screens of 640 × 480 by 8 bit pixels, 124k program space
- 1 screen of 640 × 480 by 15/16 bit pixels, 424k program space

Table C.2 gives parameter lists for programming the IMS G332 to drive two typical monitors and display resolutions. The first is for a high resolution 8-bit/pixel display, on a monitor with 48kHz horizontal scan rate. The other is for a true-colour display

on a monitor with 31.25kHz horizontal scan rate. For details of how to determine the correct parameters for other combinations of monitor and resolution, refer to the IMS G332 datasheet [3].

Register	1024x768x8	640x480x15
Vertical Scan	60	60
Horizontal Scan	48kHz	31.25kHz
Pixel Clock	60.0MHz	25.0MHz
Half Sync	12	9
Back Porch	24	18
Display	256	160
Short display	100	60
Broad Pulse	132	82
VSync	6	6
VPreEqualise	6	6
VPostEqualise	6	6
VBlank	48	64
VDisplay	1536	960
LineTime	312	200
LineStart	*	*
MemInit	490	240
TransferDelay	22	16
Boot Location	44	37
Control Register A	#342015	#442015
Control Register B	0	0

Table C.2 Example parameter lists

The **LineStart/TopOfScreen** register must be programmed with the byte offset of the live screen from the bottom of memory. For example, if the screen has been placed at machine address #80020000, these registers must *both* be programmed with #20000. The video RAM serial output shift register on the IMS B437 is 512 words (2048 bytes) long. Hence, the IMS G332 must be configured to increment the VRAM transfer address by 512 after each transfer. This is independent of the pixel width selected. Note that the sum of **MemInit** and **TransferDelay** should equal the length of the video RAM serial output shift register *in pixels* divided by four. The requirement for each pixel width is summarised in Table C.3. The times defined by the other datapath registers are always specified in multiples of four pixels: i.e. in periods of $\text{PixelClock}/4$.

Pixel Width	Meminit + TransferDelay
15/16	256
8	512
4	1024
2	2048
1	4096

Table C.3

C.5 Control register programming

There are some features of the IMS G332 which must always be operated in a particular way on the IMS B437. These are set by programming Control Register A at start up, and are summarised in Table C.4. Control register B must always be programmed with 0. In particular, note that on the IMS B437, the IMS G332 must always be operated in *interleaved* mode: this has no effect on how the video timing parameters are calculated. Control register bits which are not specified in the Table will depend on the type of monitor being driven, number of bits per pixel, cursor enable/disable, etc. Users are recommended to use the routines provided in the main part of this user manual to program the IMS G332.

Bit	Function	Program With
0	Enable VTG	
1	Enable Interlace	
2	Interface Format	
3	Mode	0 (master)
4	Plain Sync	
5	Separate Sync	0 (composite)
6	Sync On Video	
7	Pedestal	
8	Blank I/O	0 (output)
9	Blank Function	0
10	Force Blanking	0 (Unblanked)
11	Disable Blanking	0 (enabled)
12	Address Increment	0
13	Address Increment	1
14	Disable Xfer cycles	0
15	Pixel delay	0
16	Pixel delay	0
17	Pixel delay	0
18	Enable Interleaving	1
19	Delayed Sampling	0
20	Bits/pixel	
21	Bits/pixel	
22	Bits/pixel	
23	Disable Cursor	

Table C.4 IMS G332 Control Register A

C.6 Hardware cursor

The IMS G332 provides a 64×64 hardware cursor, the location of which is specified by the Cursor Position register in the IMS G332. The cursor may be blanked by setting bit 23 in IMS G332 control register A.

C.7 Events

The IMS B437 uses the IMS T805's Event Channel input to allow application software to synchronise to the vertical flyback portion of the video display cycle. The rising edge of the IMS G332's `FrameInactive` signal sets the event latch which asserts `EventReq` to the IMS T805. The event latch is cleared by `EventAck` from the

transputer which occurs when a user-provided event handler process is scheduled, and also by a hardware reset applied to the IMS B437.

Software can synchronise to `Framelnactive` by performing a channel input from the IMS T805's Event channel. It is recommended that all accesses to the IMS G332 are performed during vertical blanking.

C.8 Board control registers

This set of 8 one-bit registers is used to set up the board, reset the IMS G332, and select between true colour and pseudo-colour modes. All of these functions must be set up as part of an initialisation procedure, while the IMS G332 is not active. Hence, registers 0–6 should only be written while the IMS G332 is held in reset; ie when register 7 is 0. The recommended startup procedure is described below.

Register Number	Function	Address
0	Control 0	0x40000000
1	Control 1	0x40000004
2	Control 2	0x40000008
3	Control 3	0x4000000C
4	Control 4	0x40000010
5	Colour Mode	0x40000014
6	Control strobe	0x40000018
7	Reset IMS G332	0x4000001C

Table C.5 Board Control Registers

10.4.1 Colour mode select register

The IMS B437 operates in two distinct modes: pseudo-colour mode, and true colour mode. The pseudo-colour modes use 1,2,4 or 8 bits/pixel; the true colour modes use 15 or 16 bits/pixel. These modes are selected by programming the IMS G332 and also by board control register 5. This register must be written with 0 to use any of the pseudo-colour modes, and with 1 to use either of the true colour modes. The register is cleared (to pseudo-colour mode) by an external hardware reset to the IMS B437.

10.4.2 IMS G332 reset register

The IMS G332 can be reset at any time by writing 0 to board control register 7, waiting for a minimum of 20µs, and then writing 1 to this register. The IMS G332 is held in the reset state until this register is written with 1. This register is cleared (to 0) by an external hardware reset to the IMS B437, holding the IMS G332 in reset.

10.4.3 Startup procedure

The recommended initialisation procedure for the IMS B437 is as follows:

- 1 Assert reset to the IMS G332 to stop all of its activity, by writing 0 to board control register 7. The IMS G332 will be in the reset state after a hardware reset to the board, but it is recommended that it should always be reset explicitly.
- 2 Write 0 to control register 6.
- 3 Write the pattern 01000 to registers 0–4 respectively.
- 4 Write 1 to control register 6.
- 5 Write 0 to control register 6.
- 6 Write the appropriate value to the Colour Mode register.
- 7 De-assert reset to the IMS G332, by writing 1 to board control register 7.
- 8 Continue with the initialisation procedure for the IMS G332, as described in the IMS G332 data sheet.

C.9 Video outputs

The video outputs are terminated by 75Ω to ground on the IMS B437, to match a terminated 75Ω line. Clamping diodes to both supply rails protect the IMS G332 video outputs against the application of hostile voltages. The IMS B437 drives 1.0V video signals (including sync) into a properly terminated 75Ω line. Three SMR connectors carry the Red, Green, and Blue video signals, with sync available on all outputs. Another SMR connector carries the composite sync output from the IMS G332 which allows monitors that require a separate sync input to be driven easily. Sync output on the video signals can be turned off by appropriate programming of the IMS G332.

C.10 Board layout

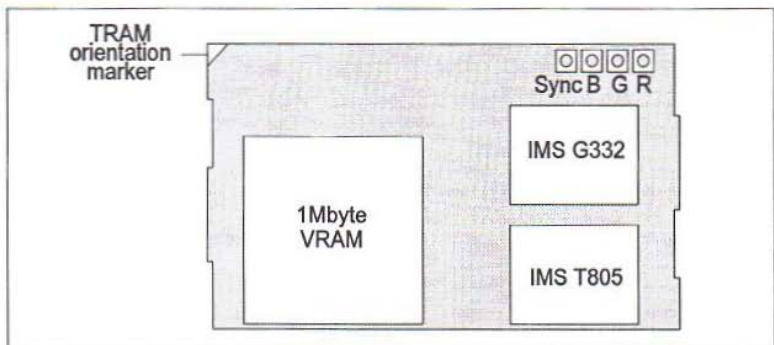


Figure C.2 IMS B437 board layout

C.11 Accessories

The IMS B437 is supplied with a set of four cables which fit the SMR connectors on the IMS B437 and are terminated at their free end with BNC male connectors. The cables are 1m in length.

D References

- 1 *Dual-In-line Transputer Modules (TRAMs)*, INMOS Technical note 29, INMOS Ltd.
- 2 *The Transputer Development and iq Systems Databook*, INMOS Ltd, 1991
- 3 *IMS G332 Colour Video Controller datasheet*, INMOS Ltd 1991.
- 4 *Crystal Oscillator Module (Appendix A.3)*, IMS B419-4 Graphics TRAM User Manual, INMOS Limited, 1990.
- 5 *Graphics Databook*, 2nd Edition, INMOS Limited, 1990.

Sales Offices

EUROPE

DENMARK

2730 HERLEV
Herlev Torv, 4
Tel: (45-42) 94.85.33
Tel: 35411
Telefax: (45-42) 948694

FINLAND

LOHJA SF-08150
Karjalankatu, 2
Tel: 12.155.11
Telefax: 12.155.66

FRANCE

94253 GENTILLY Cedex
7, Avenue Gallieni - BP 93
Tel: (33-1) 47.40.75.75
Tel: 632570 STMHQ
Telefax: (33-1) 47.40.79.10

67000 STRASBOURG

20, Place des Halles
Tel: (33) 88.75.50.66
Tel: 870001F
Telefax: (33) 88.22.29.32

GERMANY

6000 FRANKFURT
Gullesstrasse, 322
Tel: (49-69) 237492
Tel: 178997 689
Telefax: (49-69) 231957
Telefax: 6997689-STVBP

8011 GRASBRUNN

Bretenschers Ring, 4
Neukirchli Technopark
Tel: (49-89) 45006-0
Tel: 528211
Telefax: (49-89) 4605454
Telefax: 8971075-STDISTR

5000 HANNOVER 51

Rotenburgstrasse, 28A
Tel: (49-511) 616900
Tel: 175118418
Telefax: (49-511) 6151243

8500 NÜRNBERG 20

Erlenstegenstrasse, 72
Tel: (49-911) 59893-0
Tel: 525243
Telefax: (49-911) 598701

5200 SIEGBURG

Frankfurter Str. 22a
Tel: (49-2241) 660 84-86
Tel: 859510
Telefax: (49-2241) 67584

7000 STUTTGART

Oberer Kirchhaldenweg, 135
Tel: (49-711) 692041
Tel: 721718
Telefax: (49-711) 691408

ITALY

20090 ASSAGO (MI)
V.le Milano - Strada 4 -
Palazzo A/4A
Tel: (39-2) 85213 1 (10 lines)
Tel: 330131 - 330141
SGSAGS
Telefax: (39-2) 8250449

40033 CASALECCHIO DI RENO

(BO)
Via R. Fuorti, 12
Tel: (39-51) 591914
Tel: 512442
Telefax: (39-51) 591305

00161 ROMA

Via A. Torlonia, 15
Tel: (39-6) 8443341
Tel: 620653 SGSATE I
Telefax: (39-6) 8444474

NETHERLANDS

5652 AM EINDHOVEN
Meerrenakkerweg, 1
Tel: (31-40) 550015
Tel: 51186
Telefax: (31-40) 528835

SPAIN

86021 BARCELONA
Calle Platan, 8, 4th Floor, 5th Door
Tel: (34-3) 4143300 - 4143361
Tel: (34-3) 2021461

28027 MADRID

Calle Albarceles, 5
Tel: (34-1) 4051615
Tel: 27050 TCCEE
Telefax: (34-1) 4031134

SWEDEN

S-16421 KISTA
Borgarforsdagsatan, 13 - Box 1094
Tel: (46-8) 7939220
Tel: 12078 THSWG
Telefax: (46-8) 7504950

SWITZERLAND

1218 GRAND-SACONNEX
(GENEVA)
Chemin François-Lehmann 18/A
Tel: (41-22) 7986452
Tel: 415493 STM CH
Telefax: (41-22) 7984869

United Kingdom and Eire

MARLOW, BUCKS SL7 1YL
Planar House, Parkway
Globe Park
Tel: (44-628) 890800
Tel: 847458
Telefax: (44-628) 890391

AMERICAS

BRAZIL

05413 SÃO PAULO
R. Henrique Schaumann
286-C/33
Tel: (55-11) 883-5455
Tel: (391) 11-37988 "UMBR
BP"
Telefax: 11-551-128-22367

CANADA

BRAMPTON, ONTARIO
341, Main St. North
Tel: (416) 455-0505
Telefax: 416-455-2606

USA

NORTH & SOUTH AMERICAN
MARKETING HEADQUARTERS
1000, East Bell Road
Phoenix, AZ 85022
(1)-(602) 867-6100

SALES COVERAGE BY STATE

ALABAMA

303, Williams Avenue,
Suite 1031,
Huntsville, AL 35801-5104
Tel: (205) 533-5995

ARIZONA

1000, East Bell Road
Phoenix, AZ 85022
Tel: (602) 867-6100

CALIFORNIA

200 East Sandpointe,
Suite 120,
Santa Ana, CA 92707
Tel: (714) 957-6018

2055, Gateway Place,

Suite 300
San José, CA 95110
Tel: (408) 452-9122

COLORADO

1898, S. Platten Cl.
Boulder, CO 80301
Tel: (303) 449-9000

FLORIDA

902 Clint Moore Road
Congress Corporate Plaza II
Bldg 3 - Suite 220
Boca Raton, FL 33487
Tel: (407) 997-7233

GEORGIA

6025, G Atlantic Blvd
Norcross, GA 30071
Tel: (404) 242-7444

ILLINOIS

600, North Meacham
Suite 304,
Schaumburg, ILL. 60173-4941
Tel: (708) 517-1890

INDIANA

1716, South Plate St
Kokomo, IN 46902
Tel: (317) 459-4700

MASSACHUSETTS

55, Old Bedford Road
Lincoln North
Lincoln, MA 01773
Tel: (617) 259-0300

MICHIGAN

17197, N. Laurel Park Drive
Suite 253,
Livonia, MI 48152
Tel: (313) 462-4030

MINNESOTA

7805, Telegraph Road
Suite 112
Bloomington, MN 55438
Tel: (612) 944-0098

NEW JERSEY

Staffordshire Professional Ctr
1307, White Horse Road Bldg. F,
Voorhees, NJ 08043
Tel: (609) 772-6222

NEW YORK

2-4, Austin Court
Poughkeepsie, NY 12603-3633
Tel: (914) 454-8813

NORTH CAROLINA

4505, Fair Meadow Lane
Suite 220
Raleigh, NC 27607
Tel: (919) 787-6555

TEXAS

1310, Electronics Drive
Carrollton, TX 75006
Tel: (214) 466-7402

ASIA/PACIFIC

AUSTRALIA

NSW 2027 EDGECLIFF
Suite 211, Edgecliff Centre
203-233, New South Head Road
Tel: (61-2) 327.39.22
Tel: 0711 126911 TCALUS
Telefax: (61-2) 327.61.76

HONG KONG

WANCHAI
22nd Floor - Hopewell Centre
183, Queen's Road East
Tel: (852-5) 8615788
Tel: 60955 ESGIES HX
Telefax: (852-5) 8656589

INDIA

NEW DELHI 110001
Liaison Office
52, Upper Ground Floor
World Trade Centre
Barakhamba Lane
Tel: 3715191
Tel: 031-66816 STMI IN
Telefax: 3715192

KOREA

SEOUL 121
8th Floor Shinwon Building
823-14, Yuksam-Dong
Kang-Nam-Gu
Tel: (82-2) 553-0399
Tel: SCSKOR K29998
Telefax: (82-2) 552-1051

MALAYSIA

PULAU PINANG 10400
4th Floor, Suite 4-03
Bangunan FOP, 123D Jalan Anson
Tel: (04) 379735
Telefax: (04) 379616

SINGAPORE

SINGAPORE 2056
28 Ang Mo Kio - Industrial Park, 2
Tel: (65) 48214 11
Tel: RS 55201 ESGIES
Telefax: (65) 4820240

TAIWAN

TAIPEI
12th Floor
571, Tun Hua South Road
Tel: (886-2) 755-4111
Tel: 10310 ESGIE TW
Telefax: (886-2) 755-4008

JAPAN

TOKYO 168
Nisseki Takawawa Bld. 4F
2-19-10 Takawawa
Mnato-ku
Tel: (81-3) 3280-4125
Telefax: (81-3) 3280-4131