


inmos®

IMS F003
2D graphics

C libraries

Copyright © INMOS Limited 1990

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.

72 OEK 244 00

July 1990

Contents

1	IMS F003 overview and environment	3
1.1	Files comprising the graphics library	4
2	IMS F003 detail description	5
2.1	CGI conformance	5
	Table of graphical primitive functions	6
	Table of CGI attribute functions	7
2.2	Summary of functions	7
2.2.1	Plotting functions	8
2.2.2	Fill functions	8
2.2.3	Text functions	8
2.2.4	Image functions	9
2.2.5	Setup functions	9
2.2.6	Auxiliary functions	9
2.2.7	Hardware-dependent functions	9
2.3	Application areas	10
2.4	General concepts	10
2.4.1	Plot styles	10
2.4.2	Fill mode	12
2.4.3	Logical modes	12
2.5	Using the library	13
2.5.1	Choosing your monitor parameters	13
2.5.2	Compilation requirements	13
2.5.3	Linking requirements	13
2.5.4	An example program	14
2.5.5	Interpolation	15
2.5.6	Achieving transparency	16
2.6	General notes	16
2.6.1	The graphics card	16
2.6.2	Architecture	17
2.6.3	Initialization	18
2.6.4	Multiple plotting	18
2.6.5	Performance considerations	19
2.6.6	System extensions	19
3	Library functions	21
3.1	Alphabetic list of implemented functions	21
3.2	CGILIB.LIB functions	21
3.2.1	cgi_addsptext	22
3.2.2	cgi_addtext	23
3.2.3	cgi_arc	24
3.2.4	cgi_arcc	25

3.2.5	cgi_chrbegin	26
3.2.6	cgi_chrspace	27
3.2.7	cgi_chrz	28
3.2.8	cgi_circle	29
3.2.9	cgi_cls	30
3.2.10	cgi_copy	30
3.2.11	cgi_disjpolyline	31
3.2.12	cgi_dot	31
3.2.13	cgi_errstat	32
3.2.14	cgi_fcircle	33
3.2.15	cgi_ffan	34
3.2.16	cgi_fhline	35
3.2.17	cgi_frect	35
3.2.18	cgi_ftrap	36
3.2.19	cgi_init	37
3.2.20	cgi_line	39
3.2.21	cgi_paint	40
3.2.22	cgi_polygon	41
3.2.23	cgi_polyline	42
3.2.24	cgi_rect	43
3.2.25	cgi_rot	44
3.2.26	cgi_search	45
3.2.27	cgi_setbcol	47
3.2.28	cgi_setdrawmode	48
3.2.29	cgi_setdrawscreen	49
3.2.30	cgi_setfcol	50
3.2.31	cgi_setfillstyle	51
3.2.32	cgi_setfont	52
3.2.33	cgi_setlinestyle	53
3.2.34	cgi_setorient	54
3.2.35	cgi_setpelstyle	55
3.2.36	cgi_sptext	56
3.2.37	cgi_strokearc	57
3.2.38	cgi_text	58
3.2.39	cgi_zoom	59
3.3	B419.LIB functions	60
3.3.1	fs_displaybank	60
3.3.2	fs_initscreen	61
3.3.3	fs_initVTG	62
3.3.4	fs_screenaddr	63
3.3.5	fs_setpalette	64

1 IMS F003 overview and environment

The IMS F003 software provides a two dimensional graphics library, functionally conformant with a subset of the Computer Graphics Interface (CGI) standard. This library is sufficiently flexible to allow it to be used as a building block to implement additional 2-dimensional graphics operations, or indeed, 3-dimensional graphics systems.

Functions in the library fall into a number of clearly defined areas. A family of functions exists for drawing points, lines, and arcs. Another family handles two dimensional drawing operations and fills. The text support provided by another family is expandable by the programmer to accommodate personalized font information. Two dimensional screen-pixel operations offer block copying, zooming, and rotation. An auxiliary category handles miscellaneous initialization of the graphics card and data structures used by the rest of the library.

In order to separate those parts of the graphics software that are hardware dependent, a supplementary library called B419.LIB provides framestore-specific initialization, multi-frame display buffering, and colour control, for the INMOS IMS B419 graphics card.

The CGI library will be useful in application areas from engineering drawing to mimic diagrams, from interactive drawing and modelling to concept visualization.

For brevity, the software will be referred to as the CGI library throughout this document.

The IMS F003 CGI software has been implemented in ANSII C, using the INMOS TCOFF Toolset C compiler.

There are two libraries provided. The main library, CGILIB.LIB, handles all the CGI drawing operations which are independent of the chosen graphics card. It is supplied in binary form as a library, compatible with the INMOS TCOFF toolset.

The supplementary library, B419.LIB, adapts the behaviour of the main CGI library to run on the IMS B419 graphics TRAM (or compatible). This TRAM can form part of an arbitrary transputer network. The IMS B419 can support a wide range of display resolutions and pixel rates. The CGI library can be used with all of these, by simply altering initialization parameters to functions in the B419.LIB library, at run-time. A major benefit accruing from this design decision is that the library can be used with other hardware by simply linking in a different hardware library. Further, facilities for altering the display parameters at run-time, and for convenient multi-frame buffer access and colour control, are all isolated from the main CGI library with clean documented interfaces.

A programmer can access functions in the libraries from a C application. For the C programmer, a source header file, CGIDEFS.H, defines the prototypes of func-

tions that can be accessed from user applications. This header file is included with the users' C source prior to compilation.

1.1 Files comprising the graphics library

There are six files on the diskette:

B419.LIB

CGIDEFS.H

CGITYPES.H

CGILIB.LIB

EXAMPLE.C

README

These should be copied to the appropriate directory.

2 IMS F003 detail description

2.1 CGI conformance

The IMS F003 offers a functionally conformant subset of the CGI (ISO TC97/SC21 N119) standard. The CGI standard defines the interface between device-independent and device-dependent parts of a graphics system. CGI Graphical Primitive Functions, Attribute Functions, and miscellaneous initialization and error logging functions have been included.

The 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, for example, while taking into account the syntax of that language. The IMS F003 is implemented in ANSI C, and is supplied with suitable C-language function bindings.

Throughout this document, reference is made to the areas where the functions provided by the IMS F003 relate to the CGI standard functions. In some cases, for performance reasons, it has been necessary to provide a subset of capabilities, for example, the arc drawing functions provided produce only axis-aligned arcs.

CGI Graphical Primitive Functions are those functions that define the geometric components of a picture. The Graphics Primitive Functions defined in the CGI can be grouped into the following categories : line, marker, text, filled area, image, and GDP (Generalized Drawing Primitive).

CGI Attribute Functions determine the appearance of these Graphical Primitive Functions. Attributes are classified as either individual or bundleable.

While it is not the purpose of this document to discuss the CGI standard, the following table shows how various CGI (ISO TC97/SC21 N1179) Graphical Primitive Functions and attribute Functions are implemented in terms of IMS F003 CGI library functions.

Table of graphical primitive functions

Line functions	
POLYLINE	cgi_polyline
DISJOINT POLYLINE	cgi_disjpolyline
CIRCULAR ARC CENTER	cgi_arc
ELLIPTICAL ARC	cgi_arc
Marker function	
POLY MARKER	cgi_dot, cgi_copy, Pel operations
Image function	
CELL ARRAY	cgi_frect, cgi_ftrap, cgi_copy
GDP function	
GENERALIZED DRAWING PRIMITIVE miscellaneous, e.g. some logical pixel (hardware specific) modes use specialized transputer block-move instructions.	
Filled area functions	
POLYGON	cgi_polygon, cgi_ftrap, cgi_fhline, cgi_paint, cgi_search
POLYGON SET	cgi_polyline, cgi_disjpolyline, cgi_line, cgi_ftrap, cgi_fhline
RECTANGLE	cgi_rect, cgi_frect
CIRCLE	cgi_circle, cgi_fcircle
CIRCULAR ARC 3 POINT CLOSE	cgi_arcc, cgi_strokearc, cgi_ffan
CIRCULAR ARC CENTER CLOSE	cgi_arcc, cgi_strokearc, cgi_ffan
ELLIPSE	cgi_circle, cgi_fcircle, cgi_strokearc
ELLIPTICAL ARC CLOSE	cgi_arcc, cgi_strokearc, cgi_ffan
Text functions	
TEXT	cgi_text, cgi_sptext, cgi_chr, cgi_chrbegin, cgi_chrspace
APPEND TEXT	cgi_addtext, cgi_addsptext, cgi_chr, cgi_chrspace
RESTRICTED TEXT	cgi_text, cgi_sptext, cgi_chr, cgi_chrbegin, cgi_chrspace

Table 2.1 CGI (ISO TC97/SC21 N1179) Example from IMS F003

Table of CGI attribute functions

The following table shows how various CGI (ISO TC97/SC21 N1179) Attribute Functions can be implemented in terms of IMS F003 CGI library functions.

LINE TYPE	<code>cgi_setlinestyle, cgi_setdrawmode</code>
LINE WIDTH	<code>cgi_setlinestyle, cgi_setdrawmode</code>
LINE COLOR	<code>cgi_setlinestyle, cgi_setdrawmode</code>
MARKER TYPE	<code>cgi_setpelstyle, cgi_copy, cgi_zoom</code>
MARKER SIZE	<code>cgi_setpelstyle, cgi_copy, cgi_zoom</code>
MARKER COLOUR	<code>cgi_setpelstyle, cgi_copy, cgi_zoom</code>
TEXT FONT INDEX	<code>cgi_setfont</code>
TEXT PRECISION	<code>cgi_text, cgi_chr, cgi_zoom</code>
CHARACTER EXPANSION FACTOR	<code>cgi_chr, cgi_zoom</code>
CHARACTER SPACING	<code>cgi_chrspace</code>
TEXT COLOR	<code>cgi_setfcol</code>
CHARACTER HEIGHT	<code>cgi_chr</code>
CHARACTER ORIENTATION	<code>cgi_setorient, cgi_rot</code>
CHARACTER SET INDEX	<code>cgi_setfont</code>
ALTERNATE CHARACTER SET INDEX	<code>cgi_setfont</code>
INTERIOR STYLE	<code>cgi_setfillstyle, cgi_setfcol</code>
FILL COLOR	<code>cgi_setfcol</code>
HATCH INDEX	<code>cgi_setfillstyle</code>
PATTERN INDEX	<code>cgi_setfillstyle</code>
PATTERN TABLE	<code>cgi_setfillstyle</code>
PATTERN SIZE	<code>cgi_setfillstyle</code>

Table 2.2 CGI (ISO TC97/SC21 N1179) Implementation in IMS F003

2.2 Summary of functions

The CGI graphics functions are divided into the following related groups:

- plotting points, lines, and arcs
- fills rectangles, ellipsoids, trapezoids, painting
- text multiple orientation and scaling
- image copying, rotating, and zooming at pixel level
- setup cgi structure initialization
- auxiliary error handling

A further group handles the hardware-dependent aspects of the graphics system, provided in a supplementary library.

A summary each function is given next. This should highlight the flexibility of the implementation, and provide a foundation for describing first some application areas, and then some of the unfamiliar terms and capabilities.

2.2.1 Plotting functions

<code>cgi_line</code>	Line between 2 end points
<code>cgi_rect</code>	Rectangle outline
<code>cgi_polyline</code>	Consecutive line segments
<code>cgi_disjpolyline</code>	Several separate lines
<code>cgi_polygon</code>	Arbitrary polygon
<code>cgi_circle</code>	Axis-aligned ellipsoids
<code>cgi_arc</code>	Partial ellipsoids
<code>cgi_arcc</code>	Partial ellipsoids with segment and chord termination option
<code>cgi_strokearc</code>	Single stroke arc
<code>cgi_dot</code>	Single point

2.2.2 Fill functions

<code>cgi_cls</code>	Clear screen to given solid colour
<code>cgi_frect</code>	Filled rectangle
<code>cgi_fcircle</code>	Filled ellipsoid
<code>cgi_ffan</code>	Filled fan
<code>cgi_paint</code>	Region filling
<code>cgi_ftrap</code>	Filled trapezoid
<code>cgi_fhline</code>	Filled horizontal line segments

2.2.3 Text functions

<code>cgi_text</code>	Text display at explicit position
<code>cgi_addtext</code>	Append text to current text position
<code>cgi_sptext</code>	Text with individual character kerning control
<code>cgi_addsptext</code>	Append text with kerning control
<code>cgi_chrbegin</code>	Set character position

<code>cgi_chrspc</code>	Set inter-character default spacings
<code>cgi_chr</code>	Zooming of a given character

2.2.4 Image functions

<code>cgi_copy</code>	2D copy with logical operations
<code>cgi_zoom</code>	Arbitrary scaling of 2D block
<code>cgi_rot</code>	2D block rotation

2.2.5 Setup functions

<code>cgi_init</code>	Set up cgi static variables
<code>cgi_setdrawmode</code>	Sets pixel mode, fill mode, and logical replace mode
<code>cgi_setfont</code>	Init a user's packed font for text display
<code>cgi_setorient</code>	Select an orientation for text and copy operations
<code>cgi_setpelstyle</code>	Sets up a user pel design, up to 32 by 32 pixels
<code>cgi_setlinestyle</code>	Sets up a user line design, up to 1024 pixels long
<code>cgi_setfillstyle</code>	Sets up a user fill design, up to 32 by 32 pixels
<code>cgi_setdrawscreen</code>	Select a screen for drawing on

2.2.6 Auxiliary functions

<code>cgi_errstatus</code>	Expound current error
<code>cgi_search</code>	Scan horizontally for colour changes.

2.2.7 Hardware-dependent functions

These functions are contained in a separate library called B419.LIB, and allow the main CGI library functions to be used with the IMS B419 graphics card.

<code>fs_initVTG</code>	Initialize g300.
<code>fs_initscreen</code>	Initialize the framestore screen structures
<code>fs_setpalette</code>	Set colour palette entries
<code>fs_screenaddr</code>	Determine address of given screen bank
<code>fs_displaybank</code>	Select screen bank for display

2.3 Application areas

The CGI library offers particular support in the engineering drawing field, where the ability to customize line styles allows an unlimited expression of traces, the fill definitions allow hatching and shading to be implemented, and multiple scale/orientation font manipulation allows for diagram dimensioning. Enough support is provided for implementing three dimensional projections; the demonstration programs illustrate the technique.

The library could be used to implement the visual end of an interactive drawing, modelling, and CAD package.

2.4 General concepts

The CGI library has a number of drawing modes which define the run-time behaviour of most of the commands. These modes concern the following:

- Plot Style use pixels, Pels, or Linestyles for outlining ?
- Filling mode solid or patterned ?
- Logical mode replace pixel or do a logical operation ?

These modes are collectively initialized using the `cgi_setdrawmode` function.

The Plot Style and Filling mode apply only to specific commands, whereas the currently defined Logical Mode applies to everything plotted on a screen. A description of these modes, and their applicability, is given now.

2.4.1 Plot styles

All the Plotting commands conform to the Plotting style modes. This includes commands for drawing lines, dots, and arcs etc. The Plot style affects every point to be plotted to render the required object, in terms of the size, shape, and visibility of the plotted point.

There are five Plot styles:

- Pixel
- Pel
- Linestyle
- Linestyle-Transparent
- Linestyle-Pel

Pixel mode means that a single pixel is plotted for each point on a line or arc to be drawn. This gives solid outlines of minimal displayable thickness, drawn in the current foreground colour.

Pel mode means that a 'fat' pixel, called a Pel, is plotted for every point for the given command. Pels are useful for plotting cursors, markers, bullets etc. The program-

mer can specify the shape of a Pel, on a square grid, up to 32 by 32 pixels. The Pel can be thought of as having two colours, foreground and background. In Pel mode, the default logical pixel overwrite mode will only plot the non-zero entries from the Pel definition. This means that if the background colour is always zero, then the foreground can actually consist of any number of non-zero colours, all of which will be plotted normally. By choosing a suitable logical pixel mode, the zero-valued background colour can be plotted, or the foreground can be ignored. Plotting a line or arc in Pel mode can result in thick traces, for example, if the Pel is defined as some globular shape.

The programmer can define a Linestyle, in preference to the default line which is solid colour. Up to 128 bytes of information describe the Linestyle, which can have any number of colours in it. These colours are determined when the Linestyle is initialized. However, it is best to think of the Linestyle as having a foreground and a background colour, because the LineStyle-Pel mode, discussed next, makes use of the current Linestyle value to determine whether to plot a Pel. The current system foreground and background colours are ignored when using Linestyle mode.

A pointer and count are associated with an active Linestyle. The pointer advances along the Linestyle, at a rate determined by the counter. This means that it is possible to control how many points are plotted before the Linestyle pointer is incremented, giving a stretch effect. If Linestyle mode is engaged, the outcome of every plot command depends on the current position of the Linestyle pointer.

This mode is the same as Linestyle mode, except the background colour, taken to be zero, is not plotted. Normally, in Linestyle mode, the current system foreground and background colours are ignored and a value from the Linestyle vector is always used. This mode, however, allows a non-destructive effect with a zero background colour, without having to compromise the appearance of the foreground colour by using one of the logical pixel operations. This is useful in engineering drawing applications when plotting traces and general construction/sectioning lines.

The Pel and Linestyle mode can be combined, to create the Linestyle-Pel mode. This mode combines the attributes of both component modes. The effect is that for every point to be plotted in the given drawing command, the currently defined Pel is plotted in accordance with the Linestyle mode.

If current Linestyle value is non-zero then a Pel will be plotted at the current position. The colours of the Linestyle are ignored; only a zero entry in the Linestyle vector is used to signify the non-plotting of a Pel. This is why the Linestyle should be thought of as having two colours, foreground and background, even though in actuality it may have many. Note that this behaviour can be thought of as Linestyle-Pel transparent. If it is necessary to cause pixel overwrite in background areas, and still obtain the same visual effect, then the applications programmer must scribe the area in Pel mode in a suitable colour, and *then* use Linestyle-Pel mode to write the foreground.

2.4.2 Fill mode

All Fill operations are influenced by the Fill mode. This includes commands such as rectangle fill, circle fill, and region paint etc. The Fill mode permits either a solid fill colour or a user-defined pattern fill to be employed when filling.

Solid filling uses the current foreground colour.

A user-defined fill pattern can be up to 32 by 32, and can have any number of colours in it. If the patterned Fill mode is selected, the user's active fill pattern is tiled over the area to be filled. In this case, the current foreground and background colours are ignored; the actual colour values specified in the pattern are used. This allows patterns to have up to 256 different colours in them. By choosing a suitable logical pixel mode, zero values may be treated specially if required. The origin of the fill pattern is coincident with the graphics screen origin, at (0,0).

2.4.3 Logical modes

The Logical mode operates in the screen pixel domain, by performing logical bitwise operations on the colour to be plotted, and the existing colour at that pixel. In this way, the resultant colours plotted can be made to depend on the colours that were plotted previously. The Logical mode affects every drawing operation on the screen, independently of the other plot/fill modes.

The basic pixel overwrite mode is:

- overwrite pixel overwrite; ignore present pixel colour

This mode is the fastest, because it does not require reading the framebuffer prior to plotting, and does not require a logical byte operation before plotting.

The logical modes are:

- AND bitwise AND with existing pixel
- OR bitwise OR with existing pixel
- XOR bitwise XOR with existing pixel
- NAND bitwise NAND with existing pixel
- NOR bitwise NOR with existing pixel

These modes are slower than the basic pixel replace mode, because extra reads are required from the framebuffer, and a logical operation and writeback has to be performed for each pixel plotted.

There are additionally three overwrite modes called MOVE2DALL, MOVE2DZERO, MOVE2DNONZERO. These modes directly correspond to the two dimensional block move instructions on T800 series transputers, and significantly increase the performance of plotting and filling. The basic pixel overwrite mode corresponds to the MOVE2DALL mode.

There is one exception to this, concerning the plotting of a Pel (in either Pel or Line-style-Pel mode). Here, the default overwrite mode corresponds to the transputers'

MOVE2DNONZERO mode. In other words, the zero elements of the Pel are not written in overwrite mode in this case. To write the zero elements of a Pel in overwrite mode, one would select the MOVE2DALL overwrite mode.

2.5 Using the library

It is assumed in this discussion that the programmer is using the TCOFF toolset C compiler, and an IMS B419 graphics TRAM. Note that the ellipsis (three consecutive dots, thus ...) is used to represent hidden detail not relevant to the current points being discussed.

2.5.1 Choosing your monitor parameters

The CGI library is totally independent of the resolution of the output display device. All hardware dependent aspects have been isolated in the supplementary library B419.LIB.

The timing specifications for the G300 on the IMS B419 are expressed in an integer vector, passed to the `fs_initVTG` function from the B419.LIB library. It is straight forward for alternative parameters to be selected; one simply consults the IMS G300 CVC documentation for the parameters relevant to your monitor.

2.5.2 Compilation requirements

To use the CGI library functions, the C programmer includes a header file in the source of each separately compiled C module which references any of the functions in the CGI library.

The C compiler's preprocessor directive `#include` is used like this:

```
#include "cgidefs.h"
```

This header file, `cgidefs.h`, defines:

- important system constants
- macros for fastest execution of common commands
- function prototypes, conformant to ANSI C.

Note that `cgidefs.h` also references another C header file, `cgitypes.h`, which must be available at compile time of the C module.

To compile the program is straightforward :

```
icc myprog.c /t8/g
```

2.5.3 Linking requirements

The library file `cgilib.lib` should be included in the programmers' specification of files to be linked together. For example,

```
ilink myprog.tco cgilib.lib b419.lib /f startup.lnk
```

Of course, the same CGI library code can be shared by any number of separately compiled modules. It is therefore only necessary to link in the CGI library code once.

Also, it is important that calls to the CGI library functions are sequential, because of the static variables used to hold graphics state information. Parallel access to the CGI library could result in a corruption of this static state information.

2.5.4 An example program

The following example program shows how to use and initialize some of the CGI library features.

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

#include "cgidefs.h" /* include system definitions */

screen screens [2]; /* two screen structures, banks 0 and 1 */
int visible_screen, invisible_screen;

void mypalette()
{
    int i=0;
    for (i=0; i<255; i++)
        fs_setpalette ( i, i, i, i ) ; /* Grey scale palette */
}

/* These parms are suitable for 1024 by 1024
   on 50-60KHz linescan monitor */
int g3parms[] = { 17, 17, 43, 256, 87, 164,
                 6, 56, 2048, 338, 0, 491, 21, 255 };

int main ()
{
    char linestyle [64], pel [64], pfill [64];

    cgi_init(); /* init the CGI library */
    fs_initVTG ( g3parms ); /* init the framestore */
    visible_screen = 0;
    invisible_screen = 1;
    fs_initscreen ( &screens [ visible_screen],
                   visible_screen, 1024, 1024 );
    fs_initscreen ( &screens [invisible_screen],
```



```

invisible_screen, 1024, 1024 );
fs_displaybank ( visible_screen );
cgi_setdrawscreen (&screens[visible_screen]);
mypalette();

... set up Linestyle, Pel, and fills

cgi_setlinestyle ( 32, &linestyle[0], 8 );
/* Tell CGI about your */
cgi_setpelstyle ( 8, 8, &pel[0], 4, 4 );
/* lines, fills, and pels */
cgi_setfillstyle ( 8, 8, &pfill[0] );
cgi_setdrawmode ( PM_LS_PEL, RM_COL, FM_COL );
cgi_setfont (font8by8, 8, 2, 4); /* system font */
/* 8bit wide, 2 words/char, 4 lines/word */
cgi_setorient (TX_NORM); /* upright text and copy */
cgi_chrspace (10, 0);
cgi_chrbegin (100, 100);

... initialization over, do rest of program
cgi_line (0,0,100,100);
cgi_fcircle (500,500,300,200);
}

```

Suppose this source is placed in a file called `myprog.c`, and we want to build a system to run on a single IMS B419 G300 TRAM connected directly to the host computer, then to build the system the following commands are suitable:

```

icc myprog.c /t8 /g -> myprog.tco
ilink myprog.tco cgilib.lib b419.lib/f startup.lnk /t8 ->
myprog.lkuicollect myprog.lku /t /s 800 -> myprog.btl

```

We can boot the application using

```
iserver /sb myprog.btl /se
```

If the G300 TRAM is not the root transputer, we can `iskip` load the program. For example, if the IMS B419 is connected to link 2 of the root transputer, we can boot our single processor application like this:

```
iskip 2 /e/r
iserver /sc myprog.btl /ss /se
```

2.5.5 Interpolation

The `zoom` command can perform an interpolation during the zoom operation. The purpose of this is to create a larger or smaller copy of the original that more closely resembles that original. When an image is enlarged, each source pixel will corre-

spond to several in the destination image. This can give a jagged appearance. Interpolation helps to smooth these out. Also, when an image is reduced, parts of it can disappear because several pixels in the source correspond to only one in the destination. Also, reducing an image can affect the coloration, particularly if used in conjunction with a logical pixel mode.

This implementation uses a look-up table version of the reciprocal distance method that is suitable for monochrome and colour images. Both image types are handled identically. The technique involves thresholding the colours in the source image, to determine how to interpolate neighbouring pixels. This threshold mapping is key to the interpolation process. This implementation tests the most significant bit of the colour on each pixel. Thus, all colours greater than 128 will be interpolated with all colours less than 128. Adjacent colours that do not transgress this threshold will not interpolate.

The intensity of any point depends on the intensities of the 4 closest neighbouring points, weighted according to their distance from that point. A mapping is performed to threshold the colours of the four corner pixels corresponding to the point to be plotted. Based on the results of this mapping, an interpolation table can be selected. The position of the drawing point is then determined in the table, and the corresponding table entry indicates which corner point is closest in terms of its reciprocal distance. This colour is always one of the original corner colours, and because of this, the interpolated image will have a coloration closely resembling that of the original.

2.5.6 Achieving transparency

The `LineStyle-Transparent` mode can be used to give a transparency of the background colour in plotting a linestyle line. In addition, many other operations can use transparency, by using one of the logical pixel modes `RM_Z` and `RM_NZ`. These two modes correspond to the transputer's 2D block move instructions. Operations involving copying, such as `cgi_copy`, and the text operations, can take advantage of these modes.

For example, the `RM_NZ` mode used with `cgi_copy` gives a transparency of the zero colour. In other words, the zero colour regions in the source block are never written into the destination, but all other colours are. Conversely, the `RM_Z` mode gives transparency for all non-zero colours. A copy operation in the `RM_Z` mode will only write the zero colour into the destination; all non-zero colours are ignored.

Use of the `RM_NZ` mode with text commands like `cgi_atext`, will only write the foreground text colour. And using `RM_Z` mode will write the text in inverse video, by writing only the zero background colour.

2.6 General notes

2.6.1 The graphics card

The CGI library is independent of the graphics hardware used as a framestore. The supplementary library `B419.LIB` adapts the CGI library to this specific graphics

TRAM. This TRAM uses the INMOS G00 Colour Video Controller chip, which integrates programmable video timing, video memory management, memory interfaces and colour look-up table on a single chip. The device has two modes of operation. One mode, the so-called 8-bit mode, uses the video RAM as a pointer to an entry in the colour look up table (CLUT). This mode can display any 256 colours from more than 16 million, and is the mode that the CGI library operates with. The G300's other mode, the 24-bit mode, uses one byte of video memory for each of Red, Green, and Blue. While this offers many more displayable colours on-screen, the CGI library does not support this mode of operation.

The flexibility of the IMS G300 device in driving a wide range of video output devices in terms of resolution and pixel rate, is fully exploitable with the CGI library.

The library can operate to any of the G300s pixel rates and screen resolutions. The run-time initialization using `fs_initVTG(int *g3parms)` from B419.LIB, establishes the G300's clock rate, and sets the device into CLUT mode.

Note that on the IMS B419, the CGI library places the graphics origin at the top left hand corner of the screen.

2.6.2 Architecture

Fundamental to the implementation and operation of the CGI library is the definition of a SCREEN. All graphics operations are performed on a screen.

A SCREEN is defined as a C structure with the following fields:

<code>char *raster;</code>	a pointer to a byte map (normally a 2d raster field)
<code>int Xsize;</code>	Maximum X value on this screen
<code>int Ysize;</code>	Maximum Y value on this screen
<code>int stride;</code>	The stride for the current screen

The system performs most operations on the current screen. A static pointer, called `_CURRSCREEN`, points to the screen structure to be used. This is normally set using the `cgi_setdrawscreen` function, as follows:

```
cgi_setdrawscreen (&screens[visible_screen]);
```

The raster area pointed to by this screen structure is used as the current drawing field. This need not be a currently visible raster area, or indeed, it need not be one of the framebuffer screens. All drawing operations are clipped to the current drawing area.

The screen structure concept is used to represent not only video memory blocks, but also Pel maps, unpacked Font, and pattern fill areas. This results in a convenient way to be able to apply many operations to different items. For example, 2D clipping, logical plotting, zoom and rotate are all applied to SCREENS, and are all used within the library in this way.

A programmer can define any number of screens, for many purposes. One purpose might be to implement any number of software windows on an area of display memory, with a view to clipping operations to this extent.

2.6.3 Initialization

It is the programmers' responsibility to initialize any structures necessary before calling any commands that use these structures. The example program shows how to initialize the main structures used by the CGI library and set up the G300 on the graphics card.

```

cgi_init();
fs_initVTG( g3parms );
visible_screen = 0;
invisible_screen=1;
fs_initscreen ( &screens [ visible_screen],
                visible_screen, 1024, 1024 );
fs_initscreen ( &screens [invisible_screen],
                invisible_screen, 1024, 1024 );
fs_displaybank ( visible_screen );
cgi_setdrawscreen (&screens[visible_screen]);

```

Most graphics operations read/write on the RAM area pointed to by `_CURRSCREEN`. It is important that this pointer is correctly initialized, otherwise the graphics operations could write over unexpected areas of memory on the graphics card. Note in particular that calling `cgi_init` will reset the `_CURRSCREEN` pointer to NULL. This pointer is usually initialized to point to a screen structure responsible for one of the framestore screen buffers, of which only one is visible at any given moment. The function `cgi_setdrawscreen` can be used to do this. This allows graphics operations to be performed on an invisible buffer, then instantly switched for animation effects.

Providing this amount of initialization has been correctly performed, most of the commands will fail safely if they have not been correctly initialized. For example, attempting to display a character when the font has not been initialized with `cgi_setfont`, will safely default to no action. Illegal plotting/fill modes default to basic versions, ie, solid fill, pixel overwrite Logical mode, and basic Pixel Plotstyle mode. All plotting is clipped to the current screen dimension.

2.6.4 Multiple plotting

In some cases, the Plot algorithms (lines, arcs, points etc.) will plot some screen pixels more than once. In most cases, this is unlikely to cause a problem, but certain combinations of Plot mode and Logical pixel mode can produce undesired visual effects. For example if a Logical pixel mode, such as XOR is selected, then pixels that happen to be drawn twice will vanish.

This scenario can only happen with the Plot and Image algorithms when the XOR Logical Pixel mode is engaged. The other major families for filling and text display operate only in horizontal line segments, and so will not plot points more than once.

Consider the sorts of situations where multiple plotting in XOR mode can give rise to visual artifacts:

- 1 Linestyle-Pel drawing. In drawing a Pel-line, successive Pels will overlap in most cases.

- 2 Flattened ellipsoids. Ellipse dimensions are specified in terms of the length of their semi-major and semi-minor axes. If one of the axes is more than about ten times as long as the other, then at the sharper end of the ellipse, where the bend radius is tight, multiple plotting can occur. This effect is exacerbated where large Pels are involved.
- 3 Intersecting polylines. If a polyline crosses over itself, then the area of overlap will be plotted more than once. Again, large Pels will give a more marked effect.
- 4 Small vertex angles. When drawing polygons and polylines, if the angle at the vertex formed by two consecutive line segments is small, then some degree of overlap is possible. This effect can also be observed in some cases with the closed arc command, where the segment lines joining the two end points on the arc to the center can form a small vertex angle described above.
- 5 Image zoom, shearing or rotation can result in some points being plotted more than once.

2.6.5 Performance considerations

The transputer's 2D blockmove instructions are used where-ever possible, for maximum performance. For example, in drawing solid or patterned areas without regard to logical operations, the blockmoves significantly increase performance. This is akin to the CGI standards' Generalized Drawing Primitive, where the use of certain logical pixel modes offer implicit enhanced library performance in the transputer implementation.

All transputers offer a small amount of very fast one-cycle on-chip RAM. On the t800 series, this is 4 KBytes. The CGI library attempts to use this RAM for temporary storage of small vectors, used for fast region filling.

2.6.6 System extensions

The prototypes file CGIDEFS.H defines the functions in CGILLIB.LIB, and B419.LIB. The CGI library defines a number of static variables which are accessible from the user's application. The most important of these are announced in the prototypes file. Direct use of most of these is intended only as low-level support hooks for programmers to build their own extensions.

The useful ones are:

```
extern int _COL0;                system background colour
extern int _COL1;                system foreground colour
extern screenptr _CURRSCREEN;    pointer to current drawing screen
extern unsigned int font8by8[];  system font
```

The `_CURRSCREEN` pointer is normally set using a call of this form:

```
cgi_setdrawscreen (&screens[visible_screen]);
```

But it can be directly manipulated if desired.

Certain graphics operations make use of a stack. Storage for this is defined in the prototypes file, with a definition similar to this:

```
int gstack[1000];
```

It is primarily used by the `cgi_paint` algorithm. It was decided to make this stack available to the programmer, so its size can be adapted. For example, if the painter algorithm runs out of stack space when filling, it is a simple matter to allocate more stack space for it in a controlled way. Had the stack been concealed within the library, this would not have been such a trivial extension. A stack size of 1000 here is sufficient for filling about 200 concurrent horizontal line segments.

3 Library functions

3.1 Alphabetic list of implemented functions

Unless otherwise stated, all the CGI library functions operate on the current active screen structure. This is pointed to by the static `_CURRSCREEN`, but is normally set using `cgi_setdrawscreen`, to point to a framestore screen structure, to allow the bank switching to be utilized. Again, unless otherwise stated, the graphics functions draw in the current foreground colour, `_COL1`, selected with `cgi_setfcol`. Remember that on the IMS B419 graphics TRAM, the library places the graphics origin at the top left hand corner of the screen.

This list has been divided into two parts, corresponding to the CGILIB library and the B419 library.

3.2 CGILIB.LIB functions

The CGILIB.LIB functions, which are hardware independent, are prefixed with `cgi_`. They are accessed by textually including the prototype definition file `cgidefs.h` with the C source, and then linking with CGILIB.LIB.

3.2.1 `cgi_addsptext`

Function: Append text at current position, with spacing control

Calling syntax:

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

Parameters:

<code>int n</code>	number of characters to plot
<code>char *str</code>	Pointer to valid data
<code>int *dx</code>	Pointer to array of integer X spacings
<code>int *dy</code>	Pointer to array of integer Y spacings

Description:

`n` characters from the data string are plotted according to the current font. The character coordinates are updated after each character, according to the inter-character spacings defined by the vectors `dx` and `dy`. The programmer must set the spacing vectors `dx` and `dy` bearing in mind the selected orientation. The current logical pixel mode and orientation affect the displayed result. The characters are reproduced at the size of their defined font. Each pixel of every character is clipped to the current screen. Note that the font required must first be initialized using `cgi_setfont`. Note that for text display, the default logical pixel replace mode, `RM_COL`, causes the text to imprint with a rectangular box of colour 0. In some cases, this may not produce the desired effect. If only the foreground of the text is required, and a pixel overwrite mode (i.e. not a logical operation with the current frame buffer) is required, then select pixel mode `RM_NZ`. This corresponds to the transputers `MOVE2DNONZERO` capability.

3.2.2 `cgi_addtext`

Function: Append text to current text position.

Calling syntax:

```
void cgi_addtext ( int n, char *str );
```

Parameters:

<code>int n</code>	number of characters to plot
<code>char *str</code>	Pointer to valid data

Description:

`n` characters from the data string are plotted according to the current font, starting at the current character position. The character coordinates are updated after each character, according to the inter-character spacings defined by `cgi_chrspace`. The current logical pixel mode and orientation affect the displayed result. The characters are reproduced at the size of their defined font. Each pixel of every character is clipped to the current screen. Note that the font required must first be initialized using `cgi_setfont`. Note that for text display, the default logical pixel replace mode, `RM_COL`, causes the text to imprint with a rectangular box of colour 0. In some cases, this may not produce the desired effect. If only the foreground of the text is required, and a pixel overwrite mode (i.e. not a logical operation with the current frame buffer) is required, then select pixel mode `RM_NZ`. This corresponds to the transputers `MOVE2DNONZERO` capability.

3.2.3 `cgi_arc`

Function: Outline part of an axis-aligned ellipsoid.

Calling syntax:

```
void cgi_arc ( int Xc, int Yc, int A, int B,  
              int DXs, int DYs, int DXe, int DYe );
```

Parameters:

<code>int Xc</code>	Center X coord
<code>int Yc</code>	Center Y coord
<code>int A</code>	Length of semi-axis in X direction
<code>int B</code>	Length of semi-axis in Y direction
<code>int DXs</code>	X Direction of start vector
<code>int DYs</code>	Y Direction of start vector
<code>int DXe</code>	X Direction of end vector
<code>int DYe</code>	Y Direction of end vector

Description:

This function plots part of the outline of an axis-aligned ellipsoid, centered at (X_c, Y_c) , with semi-axes A and B . Both A and B must be positive. If $A > B$, then A is the semi-major axis and B is the semi-minor axis. Otherwise, B is the semi-major axis and A is the semi-minor axis. Four way symmetry is used for drawing non-circular ellipses. The points (DX_s, DY_s) and (DX_e, DY_e) define direction vectors from the center of the ellipse. Only points clockwise of the (DX_s, DY_s) vector and ANTI-clockwise of (DX_e, DY_e) are plotted. Drawing order is otherwise the same as for the `cgi_circle`. Every point on the outline is clipped to the `_CURRSCREEN` definition. The Plot style and Logical pixel mode affect the appearance of the outline.

3.2.4 `cgi_arcc`

Function: Outline part of an axis-aligned ellipsoid, with chord or segment lines.

Calling syntax:

```
void cgi_arcc ( int Xc, int Yc, int A, int B,
int DXs, int DYs, int DXe, int DYe, int CloseFlag );
```

Parameters:

<code>int Xc</code>	Center X coord
<code>int Yc</code>	Center Y coord
<code>int A</code>	Length of semi-axis in X direction
<code>int B</code>	Length of semi-axis in Y direction
<code>int DXs</code>	X Direction of start vector
<code>int DYs</code>	Y Direction of start vector
<code>int DXe</code>	X Direction of end vector
<code>int DYe</code>	Y Direction of end vector
<code>int CloseFlag</code>	Close with two segment lines or a chord

Description:

This function plots part of the outline of an axis-aligned ellipsoid, centered at (X_c, Y_c) , with semi-axes A and B . Both A and B must be positive. If $A > B$, then A is the semi-major axis and B is the semi-minor axis. Otherwise, B is the semi-major axis and A is the semi-minor axis. Four way symmetry is used for drawing non-circular ellipses. The points (DX_s, DY_s) and (DX_e, DY_e) define direction vectors from the center of the ellipse. Only points clockwise of the (DX_s, DY_s) vector and ANTI-clockwise of (DX_e, DY_e) are plotted. Drawing order is otherwise the same as for the `cgi_circle`. Every point on the outline is clipped to the `_CURRSCREEN` definition. The Plot style and Logical pixel mode affect the appearance of the outline.

The `CloseFlag` determines whether the part-outline drawn is closed with either a segment or chord. Valid `CloseFlag` values are:

<code>CF_SEGMENT</code>	Draw two segment lines to the center
<code>CF_CHORD</code>	Draw a chord between the two endpoints.

3.2.5 `cgi_chrbegin`

Function: Set current character display position.

Calling syntax:

```
void cgi_chrbegin ( int X, int Y );
```

Parameters:

<code>int X</code>	X coord of current character position
<code>int Y</code>	Y coord of current character position

Description:

This function initializes the current text position, by defining where the next character will be plotted. The co-ordinates are in pixels, from the graphics origin. The character origin is taken to be top left. Note that all text commands other than `cgi_chrz` update the current character position.

3.2.6 `cgi_chrspc`

Function: Set current inter-character spacing.

Calling syntax:

```
void cgi_chrspc ( int dX, int dY );
```

Parameters:

<code>int dX</code>	X step added to current character position per character
<code>int dY</code>	Y step added to current character position per character

Description:

This command sets the inter-character default spacings, which are added to the current character position after a character is plotted. The parameters are independent of the orientation selected, in that `dX` is always added to the screen X position, and `dY` is always added to the screen Y position. It is the programmers' responsibility to set the appropriate values here, depending on the current orientation and font size.

3.2.7 `cgi_chrz`

Function: Plot character with scaling.

Calling syntax:

```
void cgi_chrz (char data, int zlenx, zleny );
```

Parameters:

<code>char data</code>	Character to display
<code>int zlenx</code>	Width in X screen pixels to display
<code>int zleny</code>	Width in Y screen pixels to display

Description:

Any character in the current font can be independently enlarged/reduced in screen X / screen Y direction. The actual amount of scaling depends on the ratios of parameters `zlenx` and `zleny`, to the font width (`famw`) as defined in `cgi_setfont`. The current logical pixel mode and orientation affect the displayed result. The scaling is performed without interpolation. Note that the font required must first be initialized using `cgi_setfont`. Note that for text display, the default logical pixel replace mode, `RM_COL`, causes the text to imprint with a rectangular box of colour 0. In some cases, this may not produce the desired effect. If only the foreground of the text is required, and a pixel overwrite mode (i.e. not a logical operation with the current frame buffer) is required, then select pixel mode `RM_NZ`. This corresponds to the transputers `MOVE2DNONZERO` capability.

3.2.8 `cgi_circle`

Function: Outline an axis-aligned ellipsoid.

Calling syntax:

```
void cgi_circle (int Xc, int Yc, int A, int B );
```

Parameters:

<code>int Xc</code>	Center X coord
<code>int Yc</code>	Center Y coord
<code>int A</code>	Length of semi-axis in X direction
<code>int B</code>	Length of semi-axis in Y direction

Description:

This function plots the outline of an axis-aligned ellipsoid, centered at (X_c, Y_c) , with semi-axes A and B . Both A and B must be positive. If $A > B$, then A is the semi-major axis and B is the semi-minor axis. Otherwise, B is the semi-major axis and A is the semi-minor axis. Four way symmetry is used for drawing non-circular ellipses. Drawing order starts on the horizontal axis passing through the center point, heading outwards towards the vertical axis. Then, the Y-axis phase begins on the vertical axis passing through the center point, heading to where the X-axis phase ended. Every point on the outline is clipped to the `_CURRSCREEN` definition. The Plot style and Logical pixel mode affect the appearance of the outline.

3.2.9 `cgi_cls`

Function: Optimized screen clear.

Calling syntax:

```
void cgi_cls ( screenptr s, int colour );
```

Parameters:

<code>screenptr s</code>	pointer to the screen structure to be cleared.
<code>int colour</code>	solid colour to initialize whole of screen to.

Description:

This function clears an entire screen structure's raster area, to the specified solid colour. It has been optimized to use the transputer's 2D block move instructions. The current Fill mode and Logical pixel modes are ignored.

3.2.10 `cgi_copy`

Function: 2D block copy.

Calling syntax:

```
void cgi_copy ( screenptr s, int Xs, int Ys,
               int LSX, int LSY,
               screenptr d, int Xd, int Yd );
```

Parameters:

<code>screenptr s</code>	pointer to source screen
<code>int Xs</code>	Top left source co-ord X
<code>int Ys</code>	Top left source co-ord Y
<code>int LSX</code>	X size of source to copy
<code>int LSY</code>	Y size of source to copy
<code>screenptr d</code>	pointer to destination screen
<code>int Xd</code>	Top left destination co-ord X
<code>int Yd</code>	Top left destination co-ord Y

Description:

This command copies a 2D block of specified dimensions, from the source screen to the destination screen. No scaling is performed, although the current Logical pixel mode and selected orientation are observed. 2D block moves are used where possible, and the block are clipped as necessary to fit. The source and destination screen can be the same, but if the areas overlap then the effect is undefined. All length parameters must be larger than 1.

3.2.11 cgi_disjpolyline

Function: Plot a series of disjoint lines.

Calling syntax:

```
void cgi_disjpolyline ( int n, int *params );
```

Parameters:

int n	The number of coord pairs
int *params	Pointer to an array containing points line X1,Y1, X2,Y2, line X3,Y3, X4,Y4, line Xn-1,Yn-1, Xn,Yn

Description:

This function takes a pointer to an integer vector containing a sequence of at least n (X,Y) co-ordinate pairs. n is normally even, because each pair of (X,Y) data is joined by a separate line segment. This results in $n/2$ distinct line segments being drawn. Every point on every line is clipped to the `_CURRSCREEN` definition. The plotstyle and pixel replace mode affect the appearance of the outline. Drawing order is the order of the params array. If only one co-ord pair is given, a single point is plotted. If n is otherwise odd, a single point is plotted for the last line segment.

3.2.12 cgi_dot

Function: Plot a point.

Calling syntax:

```
void cgi_dot ( int X, int Y );
```

Parameters:

int X	X coord
int Y	Y coord

Description:

Plot a single point at (x,y), clipped to the `_CURRSCREEN` definition. The plotstyle and pixel replace mode affect the appearance of the point.

3.2.13 cgi_errstat

Function: Expound the current CGI error.

Calling syntax:

```
int cgi_errstat ( char *errtext, int *errqual );
```

Parameters:

```
char *errtext  A text string returned, indicating the error.
int errqual    The error qualifier.
```

Returns:

```
errno          Error code number
```

Description:

This function should be called when one wishes to examine the current error status. The function returns a non-zero error code if there has been an error in any of the previous function calls to the library. The error codes are defined in `cgitypes.h`, and are as follows:

<code>e_BADPELMODE</code>	invalid Plot mode
<code>e_BADREPMODE</code>	invalid Logical pixel mode
<code>e_BADFILLMODE</code>	invalid Fill mode
<code>e_BADSEARCHDIRN</code>	invalid search direction
<code>e_BADSEARCHTEST</code>	invalid search test criteria
<code>e_BADFORIMODE</code>	invalid orientation

The `errtext` parameter returns a textual description of the error and it is the programmer's responsibility to allocate sufficient storage for this error message ; 40 bytes is sufficient.

3.2.14 `cgi_fcircle`

Function: Draw a filled axis-aligned ellipsoid.

Calling syntax:

```
void cgi_fcircle ( int Xc, int Yc, int A, int B );
```

Parameters:

<code>int Xc</code>	Center X coord
<code>int Yc</code>	Center Y coord
<code>int A</code>	Length of semi-axis in X direction
<code>int B</code>	Length of semi-axis in Y direction

Description:

This function fills an axis-aligned ellipsoid, centered at (X_C, Y_C) , with semi-axes A and B . Both A and B must be positive. If $A > B$, then A is the semi-major axis and B is the semi-minor axis. Otherwise, B is the semi-major axis and A is the semi-minor axis. Four way symmetry is used for drawing non-circular ellipses. Drawing order starts on the horizontal axis passing through the center point, heading outwards towards the vertical axis. Then, the Y-axis phase begins on the vertical axis passing through the center point, heading to where the X-axis phase ended. Fills are done on horizontal line segments, with each endpoint clipped to the `_CURRS-CREEN` definition. The Fill style and Logical pixel mode affect the appearance of the outline.

3.2.15 `cgi_ffan`

Function: Draw a filled partial ellipsoid.

Calling syntax:

```
void cgi_ffan ( int Xc, int Yc, int A, int B,
               int DXs, int DYs, int DXe, int DYe,
               int CloseFlag );
```

Parameters:

<code>int Xc</code>	Center X coord
<code>int Yc</code>	Center Y coord
<code>int A</code>	Length of semi-axis in X direction
<code>int B</code>	Length of semi-axis in Y direction
<code>int DXs</code>	X Direction of start vector
<code>int DYs</code>	Y Direction of start vector
<code>int DXe</code>	X Direction of end vector
<code>int DYe</code>	Y Direction of end vector
<code>int CloseFlag</code>	Close with two segment lines or a chord

Description:

This function fills part of an axis-aligned ellipsoid, centered at (X_c, Y_c) , with semi-axes A and B . Both A and B must be positive. If $A > B$, then A is the semi-major axis and B is the semi-minor axis. Otherwise, B is the semi-major axis and A is the semi-minor axis. Four way symmetry is used for drawing non-circular ellipses. Drawing order starts on the horizontal axis passing through the center point, heading outwards towards the vertical axis. Then, the Y -axis phase begins on the vertical axis passing through the center point, heading to where the X -axis phase ended. Fills are done on horizontal line segments, with each endpoint clipped to the `_CURRSCREEN` definition. The Fill style and Logical pixel mode affect the appearance of the outline.

The `CloseFlag` determines whether the fill is bounded by a chord or a pair of segments. Valid `CloseFlag` values are:

<code>CF_SEGMENT</code>	Draw two segment lines to the center
<code>CF_CHORD</code>	Draw a chord between the two endpoints

3.2.16 `cgi_fhline`

Function: Fill a list of horizontal line segments.

Calling syntax:

```
void cgi_fhline ( int Y, int n, int *params );
```

Parameters:

```
int Y          Y line on which to fill segments
int n          Number of points defining start/stop X positions
int *params    Data array listing X values
```

Description:

This routine takes a pointer to a sequence of (start,stop) X values on a given Y line, and fills them. There are $n/2$ horizontal segments drawn. Each segment endpoint is clipped to the `_CURRSCREEN` definition. The Fill style and logical pixel mode affect the appearance of the line.

3.2.17 `cgi_frect`

Function: Draw a filled rectangle.

Calling syntax:

```
void cgi_frect ( int X0, int Y0, int X1, int Y1);
```

Parameters:

```
int X0        X coord of first point
int Y0        Y coord of first point
int X1        X coord of second point
int Y1        Y coord of second point
```

Description:

This fills an axis-aligned rectangle, between two diagonally opposite corners, using blockmoves where possible. The end points of each blockmove stage are clipped to the `_CURRSCREEN` definition. The Fill mode and Logical pixel mode affect the appearance of the fill. If the Fill mode is solid colour, and the pixel replace mode is overwrite, the fill area is 2D block-moved for speed. If the fill area is greater than a certain threshold, and Fill mode is `FM_PATTERN`, and the Logical pixel replace mode is moverwrite, patterned fills are 2D block-moved into position.

3.2.18 cgi_ftrap**Function:** Filled a trapezoid.**Calling syntax:**

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

Parameters:

int X1, int Y1	First point on first edge
int X2, int Y2	Second point on first edge
int X3, int Y3	First point on second edge
int X4, int Y4	Second point on second edge
int Ys	Y bounding value
int Ye	Y bounding value

Description:

This routine fills a trapezoid in horizontal sections. The trapezoid is bounded horizontally by the two (non-horizontal) lines (X1,Y1) to (X2,Y2), (X3,Y3) to (X4,Y4). The fill area is vertically restrained on the top by the largest Y value of Ys and the smallest Y of the endpoints of the non-horizontal lines; and on the bottom by the smallest of Ye and the largest of the other endpoints:

$$Y_{top} = \max [Y_s, \min(Y1,Y2), \min (Y3,Y4),]$$

$$Y_{bot} = \min [Y_e, \max(Y1,Y2), \max (Y3,Y4)]$$

where Ytop and Ybot are the upper and lower y bounds, and max() and min() refer to the largest/smallest of their list of arguments. Filling is performed in horizontal strips, with each endpoint clipped. Fillmode and logical pixel operation are accommodated. The non-horizontal lines may self-intersect without problem.

Note that if Ytop is equal to Ybot, a single line is drawn. If Ybot is smaller than Ytop, nothing is drawn.

3.2.19 `cgi_init`

Function: Initialize the cgi library static variables.

Calling syntax:

```
void cgi_init(void);
```

Parameters:

None.

Description:

This initialization function should be called before any other `cgi` functions, otherwise paranormal behaviour should be expected. It need only be called once. After calling this initialization function, the pointer to the current screen structure, `_CURRSCREEN`, is set to `NULL`. This means that attempting to do any drawing immediately afterwards will have no effect. Therefore, one should set the `_CURRSCREEN` pointer to a valid framestore screen structure, before further drawing.

Note that it is still necessary to use functions in the SET family to initialize particular features one intends using.

Example useage:

```

main()
{
    ... declarations
    cgi_init(); /* init the CGI library */
    fs_initVTG ( g3parms ); /* init the framstore */
    visible_screen = 0;
    invisible_screen = 1;
    fs_initscreen ( &screens [ visible_screen],
                   visible_screen, 1024, 1024 );
    fs_initscreen ( &screens [invisible_screen],
                   invisible_screen, 1024, 1024 );
    fs_displaybank ( visible_screen );
    cgi_setdrawscreen (&screens[visible_screen]);
                                /* Don't forget this! */
    ... set colour palette
    ... set up Linestyle, Pel, and fills

    cgi_setlinestyle ( 32, &linestyle[0], 8 );
                                /* Tell CGI about your */
    cgi_setpelstyle ( 8, 8, &pel[0], 4, 4 );
                                /* lines, fills, and pels */
    cgi_setfillstyle ( 8, 8, &pfill[0]);
    cgi_setdrawmode ( PM_LS_PEL, RM_COL, FM_COL );
    cgi_setfont (font8by8, 8, 2, 4); /* system font */
    /* 8bit wide, 2 words/char, 4 lines/word */
    cgi_setorient (TX_NORM); /* upright text and copy */
    cgi_chrspace (10, 0);
    cgi_chrbegin (100, 100);

    ... do the rest of the program
}

```


3.2.20 `cgi_line`

Function: Draw a straight line between (and including) the two end points.

Calling syntax:

```
void cgi_line ( int X0, int Y0, int X1, int Y1 );
```

Parameters:

<code>int X0</code>	X coord of first point
<code>int Y0</code>	Y coord of first point
<code>int X1</code>	X coord of second point
<code>int Y1</code>	Y coord of second point

Description:

This function plots a straight line between the given end points, ensuring no pixel gaps. Drawing order always proceeds from the first point, to the second point. Every point on the line is clip tested to the `_CURRSCREEN` definition. The `plotstyle` and logical pixel mode affect the appearance of the line.

3.2.21 `cgi_paint`

Function: Paint an existing area.

Calling syntax:

```
void cgi_paint ( int Xs, int Ys, int Bcol );
```

Parameters:

<code>int Xs</code>	X coord of interior point
<code>int Ys</code>	Y coord of interior point
<code>int Bcol</code>	Boundary colour

Description:

This routine flood-fills an area bounded by colour `Bcol`, starting from an interior point (X_s, Y_s) . In order to operate correctly with logical pixel operations, the algorithm guarantees to paint each pixel only once. Filling is affected by the Fill mode, and the logical pixel operation. Filling with the same colour as `Bcol` gives correct results, except that points inside closed regions of `Bcol` are not filled. Filling with a pattern which includes `Bcol` is dangerous, and will almost certainly fail to operate. If the initial point (X_s, Y_s) happens to be on a `Bcol` coloured pixel, the paint aborts immediately.

3.2.22 cgi_polygon

Function: Plot a polygon outline.

Calling syntax:

```
void cgi_polygon ( int n, int *params );
```

Parameters:

int n	The number of coordinate pairs to use
int *params	Pointer to an array containing points X1,Y1,X2,Y2, ..., Xn,Yn

Description:

This function takes a pointer to an integer vector containing a sequence of at least n (X,Y) co-ordinate pairs. The function will plot a polyline through all the points, and then join the first to the last point. Every point on every line is clipped to the _CURRSCREEN definition. The plotstyle and pixel replace mode affect the appearance of the outline. Drawing order is the order of the params array. If only one co-ord pair is given, a single point is plotted. If all points are coincident, a single point is plotted.

3.2.23 `cgi_polyline`

Function: Plot a polyline through all the coordinate pairs.

Calling syntax:

```
void cgi_polyline ( int n, int *params );
```

Parameters:

<code>int n</code>	The number of coordinate pairs to use
<code>int *params</code>	Pointer to an array containing points X1,Y1,X2,Y2, ..., Xn,Yn

Description:

This function takes a pointer to an integer vector containing a sequence of at least `n` (X,Y) co-ordinate pairs. The function will plot a polyline through all the points, plotting the endpoints only once. Every point on every line is clipped to the `_CURRSCREEN` definition. The `plotstyle` and `pixel replace` mode affect the appearance of the outline. Drawing order is the order of the `params` array. If one or fewer co-ord pairs are given, there is no action.

3.2.24 `cgi_rect`

Function: Draw an axis-aligned rectangle outline, between two diagonally opposite corners.

Calling syntax:

```
void cgi_rect ( int X0, int Y0, int X1, int Y1 );
```

Parameters:

<code>int X0</code>	X coord of first point
<code>int Y0</code>	Y coord of first point
<code>int X1</code>	X coord of second point
<code>int Y1</code>	Y coord of second point

Description:

An axis-aligned rectangle outline is drawn on the current screen, between the two diagonally opposite corners. If both points are the same, no action is performed. Otherwise, drawing order is horizontally from the first point to the X coord of the second point, then vertically to the second point, until the first point is reached again. Every point drawn is clip tested to the `_CURRSCREEN` definition. The plot-style and pixel replace mode affect the appearance of the line.

3.2.25 `cgi_rot`

Function: 2D block rotation.

Calling syntax:

```
void cgi_rot ( screenptr s, int Xs, int Ys,  
              int LSX, int LSY,  
              int Xd, int Yd, float angle );
```

Parameters:

<code>screenptr s</code>	pointer to source screen
<code>int Xs</code>	Top left source co-ord X
<code>int Ys</code>	Top left source co-ord Y
<code>int LSX</code>	X size of source to rotate
<code>int LSY</code>	Y size of source to rotate
<code>int Xd</code>	Top left destination co-ord X
<code>int Yd</code>	Top left destination co-ord Y
<code>float angle</code>	radian angle for rotation

Description:

This command copies a 2D block of specified dimensions, from the source area to the destination area, while performing a rotation of a specified radian angle. The current Logical pixel mode is observed. The resultant block is clipped as necessary to fit. If the source and destination areas overlap then the effect is undefined. All length parameters must be larger than 1. Positive angles denote anticlockwise rotations.

3.2.26 `cgi_search`

Function: Scan horizontal line segment for colour changes.

Calling syntax:

```
int cgi_search (int Xs, int Ys, int Bcol,  
               int sense, int dirn);
```

Parameters:

int Xs	X start point
int Ys	Selected scan line
int Bcol	Transition colour
int sense	Search test
int dirn	Search direction

Returns:

```
int xposn    X value corresponding to search result
```

Description:

This function is used to indicate where on the current scan line, a particular colour change occurs. For example, one may wish to know where the first/last pixel of a given colour, going left/right from the start point occurs on a scan line previously drawn. The `cgi_search` function can be used to establish the position of such colour changes on a specified horizontal line. It has two search modes.

One mode allows left or right searching while a specified colour is maintained. In this mode, the X value returned is the last pixel of that colour, unless the starting point is not the test colour. In this case, the X value returned is one pixel to the side of the start point, so as to be beyond the expected search range. For example, searching left from X position 500 while a specified colour is maintained, but such that the first colour tested is different, will return position 501. This number is not inside the expected search range of 500 to 0, thereby indicating that the first pixel tested was not suitable. Searching right with the same example would return 499, for the same reason.

The other mode allows left or right searching UNTIL a specified colour is located. In this mode, the X value returned is the last X position BEFORE the specified colour is located, and of course depends on whether a left or right search is employed. This has the effect of returning an X value one pixel closer to the starting point.

If invalid search directions or test conditions are passed, the function returns X position 0 and sets a `cgi` error.

Valid parameter values for `sense` are:

S_WHILENOT	continue while current pixel is not Bcol
S_WHILEGOT	continue while current pixel is Bcol

Valid parameter values for `dirn` are:

<code>S_LEFT</code>	search left of current pixel
<code>S_RIGHT</code>	search right of current pixel

Example usage:

The following example clears the screen, draws a vertical line at X position 1000 in colour 255. A search is to be done right and left from (500,500), while the colour is not 255 :

```
cgi_cls(0);
cgi_setfcol (255);
cgi_line (1000, 0, 1000, 1000);
xposn1 = cgi_search (500, 500, 255, S_WHILENOT, S_RIGHT);
xposn2 = cgi_search (500, 500, 255, S_WHILENOT, S_LEFT);
```

The results are `xposn1=999`, showing the last x position before Bcol is encountered in a right search, and `xposn2=0`, showing that the left hand screen edge is found before an instance of colour 255 on that line.

3.2.27 cgi_setbcol

Function: Select current background drawing colour

Calling syntax:

```
void cgi_setbcol( int Bcol );
```

Parameters:

int Bcol **Background colour**

Description:

This function selects the colour index in the look-up table, to use as the background colour, in those drawing operations that make use of a background colour. This integer should be in the range 0 to 255 inclusive. If the number is outside this range, the current background colour is not altered.

3.2.28 `cgi_setdrawmode`

Function: Sets current drawing modes for plotting, filling, and logical pixel

Calling syntax:

```
void cgi_setdrawmode ( int pm, int rm, int fm );
```

Parameters:

```
int pm      Plot mode
int rm      Logical pixel mode
int fm      Fill mode
```

Description:

Sets Plot mode, Logical replace mode, and Fill mode. Plot mode affects only Plot operations. Fill mode affects only fill operations. Logical pixel mode affects all drawing operations.

File `cgitypes.h` defines valid parameter entries:

Valid Plot style modes are:

```
PM_COL      Basic mode
PM_LINestyle  Linestyle mode
PM_LINestyleTR  Linestyle-Transparent mode
PM_PEL      Pel (fat pixel) mode
PM_LS_PEL   Linestyle Pel mode
```

Valid pixel replace modes are:

```
RM_COL      Overwrite mode
RM_AND      Logical bitwise AND
RM_OR       Logical bitwise OR
RM_XOR      Logical bitwise XOR
RM_NOR      Logical bitwise NOR
RM_NAND     Logical bitwise NAND
```

The following additional replace modes correspond directly to the transport's 2D blockmove instructions:

```
RM_ALL      MOVE2DALL
RM_Z        MOVE2DZERO
RM_NZ       MOVE2DNONZERO
```

Valid Fill modes are:

```
FM_COL      Solid colour fill
FM_PATTERN   Patterned fill
```

3.2.29 cgi_setdrawscreen

Function: Select current drawing screen.

Calling syntax:

```
void cgi_setdrawscreen ( screenptr sp );
```

Parameters:

screenptr sp Pointer to screen structure to draw into

Description:

This function sets the drawing area to use for all further CGI operations, or until a further `cgi_setdrawscreen` is issued.

3.2.30 `cgi_setfcol`

Function: Select current foreground drawing colour

Calling syntax:

```
void cgi_setfcol( int Fcol );
```

Parameters:

`int Fcol` **Foreground colour**

Description:

This function selects the colour index in the look-up table, to use for all further drawing operations. This integer should be in the range 0 to 255 inclusive. If the number is outside this range, the current foreground colour is not altered.

3.2.31 `cgi_setfillstyle`

Function: Establishes a custom fill design.

Calling syntax:

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

Parameters:

<code>int fxsize</code>	pattern size in X dirn
<code>int fysize</code>	pattern size in Y dirn
<code>char *fillmap</code>	pointer to pattern info

Description:

A fill pattern can be up to 32 by 32 pixels. This function initializes the system fill pattern statics. A Fill pattern has a foreground and a background colour. Generally, 0 is used for background and any other colour can be used for foreground. However, the actual colour plotted for the fill pattern is the current foreground colour, set with `cgi_setcol`. Note that the fill pattern will only be used in Plot functions if the appropriate Fill mode has been selected (`FM_PATTERN`) using `cgi_setdrawmode`.

Example usage :

```
char pfill [64];  
... init the pfill vector  
cgi_setfillstyle ( 8, 8, &pfill[0]);  
/* This defines an 8 by 8 pattern. */
```

3.2.32 `cgi_setfont`

Function: Establish a font for text display

Calling syntax:

```
void cgi_setfont( unsigned int *packfont,
                  int famw, int fwpc, int flpw );
```

Parameters:

```
unsigned int *packfont  a 32bit integer array,
                        assumed to contain an encoded font.
famw                   width of the font in bits (e.g. 8, 16, 24, 32) when unpacked
fwpc                   no of 32bit words used to encode one character (e.g. 2)
flpw                   no of rows encoded per 32 bit word (e.g. 4)
```

Description:

A font is stored in a packed form, as a sequence of 32-bit integers. Each character is unpacked as it is required. However, the system maintains a number of static variables that describe the current font to use. This function initializes the structures necessary to display any character from the font. The font supplied with the CGI library, called `font8by8`, is an 8 by 8 bit character set, packed 4 rows (of 8 bits) to one 32bit word. It consists of 256 characters, packed in a 512 word vector. The example below shows how to initialize this font for use with the text functions. A font can contain any number of characters. If one intends the text reproduction functions to result in ASCII character output, then one must ensure that the font array has the component characters in the correct position.

Example usage:

```
cgi_setfont (font8by8, 8, 2, 4);
/* Packed font is 8 bits wide when unpacked */
/* 2 32bit words represent one character */
/* There are four rows of the font per 32bit word */
```

Here, `font8by8` is assumed to hold font information.

See also: `expand_char`, `text` etc.

3.2.33 `cgi_setlinestyle`

Function: Establish a custom line design.

Calling syntax:

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

Parameters:

<code>int n</code>	length of linestyle
<code>char *ls</code>	pointer to linestyle data array
<code>int zoomFac</code>	current zoom factor for incrementing position

Description:

A Linestyle design can be up to 128 bytes long. This function initializes the system Linestyle statics. A Linestyle has a foreground and a background colour. Generally, 0 is used for background and any other colour can be used for foreground. However, the actual colour plotted for the Linestyle is the current foreground colour set with `cgi_setcol`. Note that the Linestyle will only be used in Plot functions if the appropriate Plot mode has been selected (PM_LS, for example) using `cgi_setdrawmode`.

Example usage:

```
char linestyle [64];  
... init the linestyle vector  
cgi_setlinestyle ( 32, &linestyle[0], 8 );
```

This sets up a linestyle of length 32 units, with a zoom factor of 8. This means that the next point on the linestyle is selected after the current point has been used eight times. The drawing commands that make use of symmetry (e.g. `cgi_circle`), automatically ensure the correct zoom factor on each line segment.

3.2.34 `cgi_setorient`

Function: Select current orientation for text and image copy operation.

Calling syntax:

```
void cgi_setorient( int orient );
```

Parameters:

`int orient` orientation value, selected from list below.

Description:

One of eight axis-aligned orientations can be applied when rendering a character from the current font, and when performing a block copy operation using `cgi_copy`.

Valid orientations are defined in `cgitypes.h`:

<code>TX_NORM</code>	Normal orientation
<code>TX_90</code>	Rotate 90 degrees clockwise
<code>TX_180</code>	Rotate 180 degrees clockwise
<code>TX_270</code>	Rotate 270 degrees clockwise
<code>TX_NORMFLIP</code>	Flip top to bottom
<code>TX_90FLIP</code>	Rotate 90 deg then top to bottom flip
<code>TX_180FLIP</code>	Rotate 180 deg then top to bottom flip
<code>TX_270FLIP</code>	Rotate 270 deg then top to bottom flip

3.2.35 `cgi_setpelstyle`

Function: Establish a custom Pel design.

Calling syntax:

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

Parameters:

<code>int pxsize</code>	Pel size in X dirn
<code>int pysize</code>	Pel size in Y dirn
<code>char *pelmap</code>	pointer to Pel info
<code>int xorg</code>	The x offset into the Pel for the Pel's center
<code>int yorg</code>	The y offset into the Pel for the Pel's center

Description:

A Pel design can be up to 32 by 32 pixels. A Pel has a foreground and a background colour. Generally, 0 is used for background and any other colour can be used for foreground. However, the actual colour plotted for the Pel is the current foreground colour, set with `cgi_setcol`. This function initializes the system Pel statics. Note that the Pel will only be used in Plot functions if the appropriate Plot mode has been selected (PM_PEL, for example) using `cgi_setdrawmode`.

Example usage:

```
char pel [64];  
... set up pel array  
cgi_setpelstyle ( 8, 8, &pel[0], 4, 4 );
```

This sets up a Pel of 8 by 8 pixels. The origin of the Pel is at (4,4), which is the center of it's bulk. By moving the (xorg,yorg) values, the origin of the Pel can be moved, with respect to the Pel. This would allow, say, an arrow-shaped cursor Pel to have it's origin at the tip of the arrow.

3.2.36 `cgi_sptext`

Function: Plot text at specified position with individual character spacing.

Calling syntax:

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

Parameters:

<code>int X</code>	Screen X co-ord of top left corner
<code>int Y</code>	Screen Y co-ord of top left corner
<code>int n</code>	number of characters to plot
<code>char *str</code>	Pointer to valid data
<code>int *dx</code>	Pointer to array of integer X spacings
<code>int *dy</code>	Pointer to array of integer Y spacings

Description:

The current character position is first set to (X, Y). Then, n characters from the data string are plotted according to the current font. The character coordinates are updated after each character, according to the inter-character spacings defined by the vectors dx and dy. The programmer must set the spacing vectors dx and dy bearing in mind the selected orientation. The current logical pixel mode and orientation affect the displayed result. The characters are reproduced at the size of their defined font. Each pixel of every character is clipped to the current screen. Note that the font required must first be initialized using `cgi_setfont`. Note that for text display, the default logical pixel replace mode, `RM_COL`, causes the text to imprint with a rectangular box of colour 0. In some cases, this may not produce the desired effect. If only the foreground of the text is required, and a pixel overwrite mode (i.e. not a logical operation with the current frame buffer) is required, then select pixel mode `RM_NZ`. This corresponds to the transputers `MOVE2DNONZERO` capability.

3.2.37 `cgi_strokearc`

Function: Outline an arc without using symmetry techniques.

Calling syntax:

```
void cgi_strokearc ( int Xc, int Yc, int A, int B,  
                    int DXs, int DYs, int DXe, int DYe );
```

Parameters:

<code>int Xc</code>	Center X coord
<code>int Yc</code>	Center Y coord
<code>int A</code>	Length of semi-axis in X direction
<code>int B</code>	Length of semi-axis in Y direction
<code>int DXs</code>	X Direction of start vector
<code>int DYs</code>	Y Direction of start vector
<code>int DXe</code>	X Direction of end vector
<code>int DYe</code>	Y Direction of end vector

Description:

This function plots part of the outline of an axis-aligned ellipsoid, centered at (X_c, Y_c) , with semi-axes A and B . Both A and B must be positive. If $A > B$, then A is the semi-major axis and B is the semi-minor axis. Otherwise, B is the semi-major axis and A is the semi-minor axis. The points (DX_s, DY_s) and (DX_e, DY_e) define direction vectors from the center of the ellipse. Only points clockwise of the (DX_s, DY_s) vector and ANTIClockwise of (DX_e, DY_e) are plotted. Drawing order is from the positive X axis, anticlockwise, in one sweep. This is clearly much slower than the normal arc, because no advantage of symmetry is used. This function is provided to offer (more) pleasing arcs when using a defined linestyle. Every point on the outline is clipped to the `_CURRSCREEN` definition. The Plot style and Log-pixel mode affect the appearance of the outline.

3.2.38 `cgi_text`

Function: Display text at specified coordinates.

Calling syntax:

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

Parameters:

<code>int X</code>	Screen X co-ord of top left corner
<code>int Y</code>	Screen Y co-ord of top left corner
<code>int n</code>	number of characters to show
<code>char *str</code>	Pointer to valid data

Description:

The current character position is first set to (X, Y). Then, n characters from the data string are plotted according to the current font. The character coordinates are updated after each character, according to the inter-character spacings defined by `cgi_chrspace`. The current logical pixel mode and orientation affect the displayed result. The characters are reproduced at the size of their defined font. Each pixel of every character is clipped to the current screen. Note that the font required must first be initialized using `cgi_setfont`. Note that for text display, the default logical pixel replace mode, `RM_COL`, causes the text to imprint with a rectangular box of colour 0. In some cases, this may not produce the desired effect. If only the foreground of the text is required, and a pixel overwrite mode (i.e. not a logical operation with the current frame buffer) is required, then select pixel mode `RM_NZ`. This corresponds to the transputers `MOVE2DNONZERO` capability.

3.2.39 `cgi_zoom`

Function: Arbitrary X and Y scaling of a 2D block.

Calling syntax:

```
void cgi_zoom ( screenptr s, int Xs, int Ys,
               int LSX, int LSY,
               screenptr d, int Xd, int Yd,
               int LDX, int LDY,
               int interpolate );
```

Parameters:

<code>screenptr s</code>	pointer to source screen area
<code>int Xs</code>	Top left source co-ord X
<code>int Ys</code>	Top left source co-ord Y
<code>int LSX</code>	X size of source to copy
<code>int LSY</code>	Y size of source to copy
<code>screenptr d</code>	pointer to destination screen area
<code>int Xd</code>	Top left destination co-ord X
<code>int Yd</code>	Top left destination co-ord Y
<code>int LDX</code>	X size of destination
<code>int LDY</code>	Y size of destination
<code>int interpolate</code>	select between normal and interpolated zoom

Description:

This command copies a 2D block of specified dimensions, from the source screen to the destination screen, while performing arbitrary scaling independently in X and Y directions. A DDA technique is used to effect the scaling. The current Logical pixel mode is observed. Since some points will be drawn more than once, it is best not to select the XOR Logical mode. The blocks are clipped as necessary to fit. The source and destination screen can be the same, but if the areas overlap then the effect is undefined. All length parameters must be larger than 1.

`interpolate` determines whether an interpolated zoom is to be performed. If this parameter has value zero, no interpolation will be performed. If this parameter has value one, the interpolation will be performed. Note that the interpolated zoom is much slower than the non-interpolated zoom. Also remember that only colours on opposite sides of the 128 value threshold will be blended together (in a way that does not affect the coloration of the original); adjacent colours on the same side of 128 threshold will appear as if they have been non-interpolated.

3.3 B419.LIB functions

The B419.LIB functions, which correspond to the IMS B419 graphics TRAM bindings, are prefixed with `fs_`. They are accessed by linking compiled C with B419.LIB. Note that the function prototypes for these functions are included in the `cgidefs.h` file, which will be used to reference the `cgilib` functions.

3.3.1 `fs_displaybank`

Function: Select which display bank to view.

Calling syntax:

```
void fs_displaybank ( int bank);
```

Parameters:

`int bank` The screen number, usually 0 or 1, to display.

Description:

This function should be called before attempting to view anything on an output device. By calling this function and changing the bank number, a multi-buffered screen output, suitable for animation, can be achieved. By setting the value of `_CURRSCREEN` to the address of the invisible screen structure, the CGI library can operate on the invisible frame buffer, then the display banks can be instantly switched over.

Example usage:

```
fs_initVTG ( g3parms );
fs_initscreen ( &screens[0], 0, 1024, 1024);
/* frame buffer 0 */
fs_initscreen ( &screens[1], 1, 1024, 1024);
/* and bank 1 */
fs_displaybank (0); /* screen 0 */
```

3.3.2 fs_initscreen

Function: Correspond a framestore screen structure with a bank number

Calling syntax:

```
void fs_initscreen ( screenptr s, int bank,
                   int xSize, int ySize );
```

Parameters:

screenptr s	A pointer to a screen structure.
int bank	The bank number of this screen.
int xSize	The X pixel dimension of the screen.
int ySize	The Y pixel dimension of the screen.

Description:

This function is used to initialize a framestore screen structure. There is an important distinction to be made between a framestore screen and a non-framestore screen. A framestore screen is the correct shape and size for a total coverage of the graphics card display. The graphics card will only display framestore screens, and the memory position of the raster area is obviously dependent on the address decoding of the graphics card. All CGI library operations are written into a screen of the appropriate dimensions, with a raster field at a suitable position for the graphics card. This function is used to correctly define the dimensions and raster area for framestore screens, which can then be subsequently selected by `fs_displaybank(bank)`. It expects to have all framestore screens the same shape and size. It is vital that this function is called with the correct X and Y framestore screen sizes, otherwise subsequent frame buffer address calculations will be wrong.

Remember that normal screen structures can be of any size, can have their raster area at an arbitrary memory address range, and can be used for various intermediate purposes. Such screens are almost certainly unrelated to the dimensions of a framestore screen.

Example usage:

```
fs_initVTG (g3parms);
fs_initscreen ( &screens[0], 0, 1024, 1024);
/* frame buffer 0 */
fs_initscreen ( &screens[1], 1, 1024, 1024);
/* and bank 1 */
fs_displaybank (0);
cgi_setdrawscreen (&screens[0]);
fs_setpalette ( 42, 255, 0, 0 );
/* entry 42 to full red */
```

3.3.3 fs_initVTG

Function: Initialize the G300 to your screen resolution.

Calling syntax:

```
void fs_initVTG ( int *timingdata );
```

Parameters:

int *timingdata a vector of G300 timing values.

Description:

The IMS G300 has a number of run-time programmable registers which initialize the device for one of many possible screen resolutions and frame rates. The fourteen elements of `timingdata` are used by `fs_initVTG` to configure the G300. These elements are as follows:

0 PLLfactor,	5 broad_pulse,	10 line_start,
1 half_sync,	6 v_sync,	11 mem_init,
2 back_porch,	7 v_blank,	12 transfer_delay,
3 display,	8 v_display,	13 mask_register
4 short_display,	9 line_time,	

This function should be called before attempting to view anything on an output device. Suitable values to reflect the timing parameters required by your monitor can be determined from the IMS B419 manual.

Example usage:

```
int g3parms[] = { 17, 17, 43, 256, 87, 164,
6, 56, 2048, 338, 0, 491, 21, 255 };
main()
{
fs_initVTG ( g3parms );
... rest of program
}
```

These values would be suitable for a high resolution 1024 by 1024 monitor at around 60 frames/second, such as a TAXAN ULTRAVISION 1000, or an HITACHI 4219 etc.

3.3.4 fs_screenaddr

Function: Return the raster field address of a framestore screen bank.

Calling syntax:

```
char* fs_screenaddr (int bank );
```

Parameters:

int bank The screen number, usually 0 or 1, to point to.

Description:

This low-level function is used to determine the address of a framestore screen raster area. It is used by the `fs_initscreen` function. It determines the address by using the bank number, and knowledge of the framestore X and Y dimensions as specified by a previous call to `fs_initscreen`. It is important that calls to `fs_initscreen` specify the total dimensions of a frame buffer, otherwise all further address calculations, used when switching visible screen banks, will be wrong.

3.3.5 `fs_setpalette`

Function: Set up the colour look-up table entries.

Calling syntax:

```
void fs_setpalette ( int clutno, int red,  
                    int green, int blue );
```

Parameters:

<code>int clutno</code>	A number from 0 to 255 corresponding to the CLUT entry.
<code>int red</code>	The red content for this colour, range 0 to 255.
<code>int green</code>	The green content for this colour, range 0 to 255.
<code>int blue</code>	The blue content for this colour, range 0 to 255.

Description:

Generally, this function would be called for value of `clutno` to be viewed, before attempting to view that colour. The palette entries can be changed dynamically at any time, and obviously will effect everything already onscreen, as well as features still to be drawn.

Example usage:

```
fs_setpalette ( 42, 255,0,0 ); /* entry 42 to full red */
```