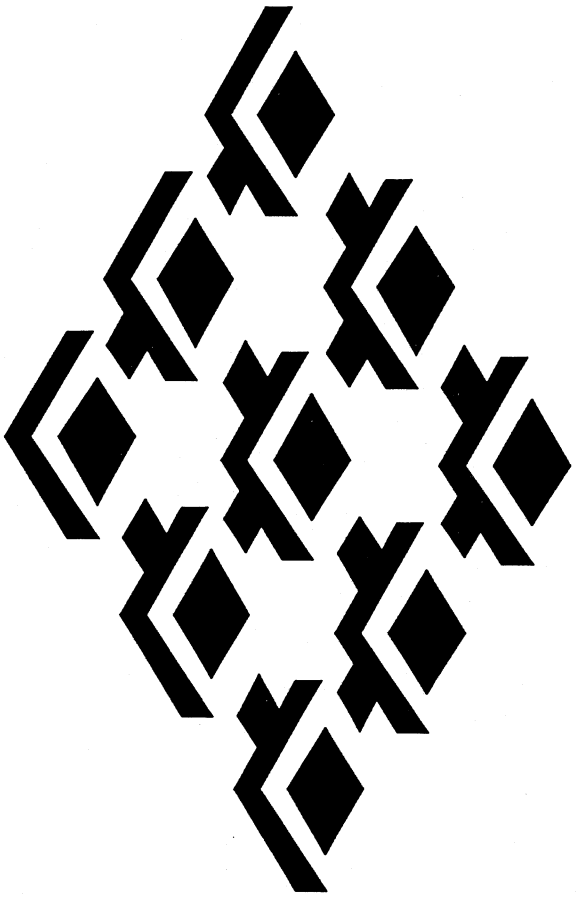
TRANSTECH TTGS

GRAPHICS SERVER
USER MANUAL

VERSION 1.0 4:9:1989



TRANSTECH DEVICES LIMITED

UNIT 17, WYE INDUSTRIAL ESTATE
LONDON ROAD
HIGH WYCOMBE
BUCKINGHAMSHIRE
HP11 1LH
ENGLAND

TELEPHONE (+44) 0494 464303
FAX (+44) 0494 463686

Contents

1 Introduction

The Transtech Graphics Server (**TTGS**), consists of about fifty elementary graphics primitives, which can be used to manipulate, *points, vectors, circles, arcs, rectangles,* and *polygons.* It also includes support for *text,* and *windows.*

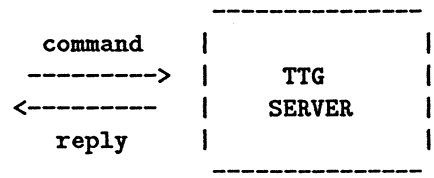
The Server is written exclusively in **occam** and is accessed via a channel interface. This ensures that the Server can be accessed from any of a wide range of high level languages, usually by means of a simple interface procedure (stub) which interfaces directly with the Server channels.

2 Function Overview

Table 1 lists all the functions supported in the TTG Server, their Op.Codes and parameters. The majority of these functions are compatible with the Inmos B007 library. Those functions not compatible with the B007 library are marked with an asterisk in table 1 . All the parameters are of type INT32 (signed integer) [1] unless indicated otherwise, e.g BYTE.

3 Server Interface

The TTG Server is accessed through a pair of transputer channels. The input channel is used to pass all requests, and the output channel is used by the Server to return results. The TTG Server actually consists of a stand alone process which serves these channels, and runs concurrently with the user application. This means that the TTG Server process can be used as a graphics server, processing requests from any number of processes, multiplexed on the command/reply channels.



4 Server software

The Server was developed in occam under the IMS D700d Transputer Development system. As such it is currently available in source form only on IBM PC $5\frac{1}{4}$" disks. However, the supplied bootable binary of TTGS (**TTGS.btl**) can be loaded onto a TTG TRAM running under any host, it is only necessary to transfer **TTGS.btl** to the appropriate medium.

To install the software, place the floppy in drive A of your PC and type 'a:install'. Further instructions will follow, then all files will be copied.

The supplied floppy contains a directory **source** in which can be found

TOPLEVEL.TOP this is a TDS (IMS D700d) top-level file, and contains the source of TTGS plus some examples. All other supplied formats can be derived from this source. Within this TDS toplevel file can be found

[1] Note that it is highly unlikely that a 16-bit version of this server will ever be produced, and for efficiency the TTG Server source uses INT throughout. However, if a 16-bit Transputer were to attempt to communicate with a 32-bit TTG Server, it would need to use INT32 explicitly.

Op.Code	Function	Parameters
1	c.plot.point	x1, y1
2	c.draw.line	x1, y1, x2, y2
3	c.draw.circle	xCentre, yCentre, radius
4	c.draw.arc	x1, y1, x2, y2, x3, y3
5	c.draw.rectangle	x, y, xLength, yLength
6	c.draw.polygon	numberOfSides, x1, y1, ..., xn, yn
7	c.fill.polygon	x, y
8	c.move	x, y
9	c.move.rel	deltaX, deltaY
10	c.clear.screen	colourValue
11	c.select.screen	screenNumber
12	c.display.screen	screenNumber
13	c.flip.screen	
14	c.copy.screen	destinationScreenNumber
15	c.clear.window	colourValue
16	c.select.window	windowNumber
17	c.display.window	windowNumber, x, y
18	c.set.window	xDimension, yDimension
19	c.set.draw.mode	mode
20	c.draw.char	characterNumber
21	c.define.char	charNumber, [8] BYTE bitMap
22	c.write.string	noBytesText, [ ] BYTE text
23	c.untyped.string	
24	c.write.number	number
25	c.scroll	
26	c.jump.scroll	
27	c.rotate	quarterTurns
28	c.reflect.x	
29	c.reflect.y	
30	c.line.feed	
31	c.carriage.return	
32	c.set.colour	colourValue, red, green, blue
33	c.select.fg.colour	colourValue
34	c.select.bg.colour	colourValue
35	c.select.colour.table	tableNumber
36	c.set.mask.reg	mask
37	c.set.xwidth	plot pixel width
38	c.set.yheight	plot pixel height
39	c.get.pixel.colour	
40	c.line.frequency	lineFrequency
41	c.frame.rate	frameRate
42	c.interlace	boolean
43	c.pixel.clock	pixelFrequency
47	c.display.x	xSize in pixels
48	c.display.y	ySize
44	c.init.crt	
45	c.quick.fill	xseed, yseed
46	c.write.file.to.screen	[ySize][xSize] BYTE image
* 50	c.fast.clear	colour
51	c.define.gcursor	[16][16] BYTE pixel, BYTE edgeChar, transparentChar, interiorChar, edge, interior
52	c.remove.gcursor	
53	c.draw.gcursor	x, y
54	c.wait.vblank	
255	c.terminate	

Table 1: Miscellaneous TTGS Functions

TTGconst.tsr this filed fold contains a library, which contains the definitions of the command op codes and return codes used by TTGS.

TTGS.tsr contains library this filed fold contains a library, which in turn contains the SC of TTGS. To use TTGS in an application it is necessary to include the lines

```
#USE "TTGconst.tsr"
#USE "TTGS.tsr"
```

within your application.

PROGRAM TTGS source this is the configuration harness used to generate the supplied **TTGS.btl** file. It is set up to accept commands down link 1. This can be changed if desired, and recompiled.

EXE TTG server as exe this is a TDS 'executable', and runs within the Inmos D700D. This will typically be used from the TDS to drive any of the Transtech range of graphics devices, and the application code will need to be included within the EXE. Note that to be usable in this form the TTG TRAM must the one running the TDS.

various folds labelled **EXE example** contain examples illustrating how to drive TTGS. To run the examples, first point at **PROGRAM TTGS source** and press ALT-4 to boot TTGS into the TTG TRAM, then get the example EXE (using ALT-5) and run the example (using ALT-6).

Visible at the root of the floppy is

TTGS.BTL this is a standalone bootable image. This is standalone version of the TTG Server can be booted direct into a Transtech Graphics module using the **afserver** or **iserver**. The supplied image assumes that commands will be received down link 1. Experienced users can use this file for dynamic code loading, and this is illustrated in the C examples later in this document. Dynamically loading the **.BTL** file is simpler than using the **.CSC**, but for true maniacs

TTGS.CSC this is an extracted code fold and resides within the library fold described above. This is suitable for dynamic code loading, and will typically be used from within occam or 'C' programs for dynamically loading the TTG Server process onto a TTG TRAM. Intimate knowledge of .CSC file format is required to use this.

A TTG Server can be driven by an occam program as follows:

```
[nparams]INT32 parameters:
INT32 command, reply:
SEQ
  to.ttglib ! command
  SEQ i = 0 FOR SIZE parameters
    to.ttglib ! parameters [i]
  from.ttglib ? reply
```

5 Initialisation

The main elements of the TTG Server which require initialisation are the Video Timing Generation parameters for your particular monitor. There are a total of six parameters which need to be defined, as follows:

```
line.frequency   the total number of scan lines per screen.
frame.rate       the total number of frames per second.
pixel.clock      the rate at which pixels are output (video data rate).
display.x        the number of displayable pixels per scan line.
display.y        the total number of displayed scan lines.
interlace        interlaced, or non interlaced display operation.
```

Typical values are listed below.

```
VAL line.frequency      IS 34800:       -- 34.8 KHz
VAL frame.rate          IS 50:          -- 50 Hz frame rate
VAL pixel.clock         IS 25000000:    -- 25 MHz pixel clock
VAL display.x           IS 512:
VAL display.y           IS 512:
VAL BOOL interlace      IS FALSE:
```

The TTG Server boots up in the most sensible state for any given graphics card - for example the TTG1 boots with the above values as default, whereas the TTG3 boots with x.size = y.size = 1024, pixel.clock = 85000000.

If different values are required (for more details refer to the relevant Transtech Graphics Device User Guide), then parameters are modified by individual calls to the TTG Server. The TTG Server maintains internal state for each of the definable parameters, and the VTG is only changed when the **init.crt** is sent. So for example:

```
err  = TTG(c_line_frequency, 38000);
err  = TTG(c_frame_rate, 52);
err  = TTG(c_pixel_clock, 25000000);
err  = TTG(c_interlace, FALSE);
err  = TTG(c_x_size, 512);  /* maps on to c.displayx */
err  = TTG(c_y_size, 512);  /* and displayy */

err  = TTG(c_init_crt);
```

can be used from PARALLEL C to initialise the VTG.

NB the TTG Server calls an internal routine **initCRTC()** on receiving the **init.crt** command, with the current parameter values. This routine is described in more detail for specific graphics modules in the relevant User Guide. So if the user wishes to access the device directly, refer to the User Guide and the **initCRTC()** example.

BITS	RANGE	EFFECT
0-1	0-3	green intensity
2-3	0-3	red intensity
4-5	0-3	blue intensity
6-7	0-3	intensity bias

Table 2: Pixel fields for colour table 0

colourValue	Colour Range	
0 - 15	grey scale	
16 - 31	null to red	
32 - 47	null to green	
48 - 63	null to blue	
64 - 79	null to yellow	
80 - 95	null to cyan	
96 - 111	null to magenta	
112 - 127	low blue	plus red to green
128 - 143	low red	plus green to blue
144 - 159	low green	plus blue to red
160 - 175	medium blue	plus red to green
176 - 191	medium red	plus green to blue
192 - 207	medium green	plus blue to red
208 - 223	full blue	plus red to green
224 - 239	full red	plus green to blue
240 - 255	full green	plus blue to red

Table 3: look-up table ranges for Table 1

6 Colour Tables

The TTGS library supports four internal colour tables. The desired current colour table can be selected by the function **select.colour.table**.

Table 0: IBM Standard
This has colours partitioned amongst the eight bits of each pixel, with 2 bits per colour and 2 bits to bias any of the colours. Bias value 0 has no effect, 1 intensifies the red, 2 intensifies the green, and 3 intensifies the blue.

Table 1: TTG Standard
The colour look-up table entries can be summarised as follows:

Table 2 Grey Scale
This table is the default table for monochrome images. It consists of 256 grey levels from 0 (dark) to 255(white).

Table 3 Dithered colour
This table is the default table for true colour. Each byte contains a 3-bit red field (bits 0..2) a 3-bit green field (bits 3..5) and a 2-bit blue field (bits 6 and 7). This table gives a good range of spectral hues and full-colour images can be efficiently dithered down to 8 bits using this table, the dithering operation requiring only bit masks, shifts and adds.

7 Screen Support

The TTG Server provides support for drawing into double buffered frame stores.

The following functions are provided for controlling the display screen: **select.screen, display.screen, clear.screen, flip.screen**, to select the screen to be drawn into and displayed, to clear the draw screen, and to flip between the draw screens. The **copy.screen** function copies the contents of one screen to another.

8 Windows

The TTG Server provides limited support for windows. Any selected window exists in all the available screens. **set.window** sets up a window with a given width and length and returns an I.D for the allocated window. **clear.window** and **select.window** perform the same operations on a window as the equivalent functions do on a screen. **display.window** defines where on the screen the window should be placed.

scroll and **jump.scroll** can be used to scroll the current window. The two functions give smooth scroll, and character scroll respectively.

9 Colour Selection

The look-up table can be dynamically modified using the **set.colour** function. This command overwrites the selected table entry. The foreground and background colours for all textual and primitive display functions is selected by the two functions **select.fg.colour** and **select.bg.colour**.

The function **get.pixel.colour** returns the value for any pixel on the current draw screen.

The **set.pixel.mask** function can be used to setup the mask register in the look-up table. This register is ANDed with all displayed pixels, so it can be used to rapidly select pixels drawn. Can be useful for selecting single bit images for rapid animation.

10 Text Support

The TTGS Server also provides limited text support. Simple functions are provided to write single characters (**draw.char**), strings (**write.text**, and numbers (**draw.number**). A total of 256 characters are supported. The text characters are all ASCII compatible, and there are an additional 31 graphics characters. Note that any of these characters can be redefined dynamically. The full character set is defined below:

```
0-31        graphics
32-57      "#$%&'()*+,+-./0123456789
58-96      ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
97-126     abcdefghijklmnopqrstuvwxyz{        |}~
127-255     graphics
```

The server also supports a number of text drawing modes, which enables the overlaying of text planes. This is discussed further in the definition of the **set.draw.mode** function.

11 Function Overview

draw.line

Parameters: x1, y1, x2, y2
Reply: e.ok on success else error code

Function: draw a straight line between the two specified points in the current draw screen.

plot.point

Parameters: x, y
Reply: e.ok on success else error code

Function: plot a single pixel at the defined postion within the current draw screen.

draw.circle

Parameters: xCentre, yCentre, radius
Reply: e.ok on success else error code

Function: draw a circle of defined radius at the defined postion within the current draw screen.

draw.arc

Parameters: x1,y1, x2,y2, x3,y3
Reply: e.ok on success else error code

Function: draw an arc specified by the three points defined postion within the current draw screen.

draw.rectangle

Parameters: x, y, xLength, yLength
Reply: e.ok on success else error code

Function: draw a rectangle. x,y defines the top left vertex, xLength,yLength, the size of the rectangle. The rectangle is drawn into the current draw screen.

draw.polygon

Parameters: numberOfSides, x1,y1,..., xn,yn
Reply: e.ok on success else error code

Function: Draw a polygon with the specified number of sides into the current draw screen.

fill.polygon

Parameters: x, y
Reply: e.ok on success else error code

Function: Fill an arbitrary polygon enclosing the specified point. The polygon is filled with the current foreground colour. Any point within the polygon may be selected. This algorithm fills an area bounded by the current foreground colour, so x,y must be within the boundary (not on the boundary) of the desired polygon, which has already been drawn in the current foreground colour.

quick.fill

Parameters: x, y
Reply: e.ok on success else error code

Function: Fast fill a *convex* polygon enclosing the specified point. The polygon is filled with the current foreground colour.

draw.char

Parameters: character.Number
Reply: e.ok on success else error code

Function: Draw the given ASCII character at the current cursor position.

write.string

Parameters: noBytesText, Text

mode	Foreground	Background
0	Replace	Replace
1	AND	AND
2	OR	OR
3	XOR	XOR
4	Replace	no change
5	AND	no change
6	OR	no change
7	XOR	no change

Table 4: Text display modes

Reply: e.ok on success else error code

Function: write a text string to the current cursor position. The string wraps around the edge of the screen, or window.

write.number

Parameters: number
Reply: e.ok on success else error code

Function: Write the given number (base 10) to the current cursor position.

define.char

Parameters: charNumber, rowOne, ...,rowEight
Reply: e.ok on success else error code

Function: Allows user to program characters. The function expects an 8*8 bit mapped image of the character, so a total of 8 bytes must be given. Each byte specifies eight consecutive pixels, starting with the top row. Bit 7 (i.e char AND 128) is the far left bit of the row.

set.draw.mode

Parameters: modeNumber
Reply: e.ok on success else error code

Function: The Server supports several text drawing modes. The mode defines the operation applied as new character grids (8*8 pixels) are combined with the existing screen text.

The colour values of the foreground and background pixels are modified according to the mode as defined in Table 4.

rotate

Parameters: quarterTurns
Reply: e.ok on success else error code

Function: Following this command, all characters displayed are rotated through the defined number of quarter turns. 0 = no rotation, 1 = 90deg, 2 = 180deg, 3 = 270deg.

reflect.x

Parameters: None
Reply: e.ok on success else error code

Function: Text is reflected in the X plane. This can be used in combination with reflect.y to reflect in both planes.

reflect.y

Parameters: None
Reply: e.ok on success else error code

Function: Text is reflected in the Y plane. This can be used in combination with reflect.x to reflect in both planes.

move

Parameters: xPosition, yPosition
Reply: e.ok on success else error code

Function: Move text cursor to the absolute position specified.

move.rel

Parameters: dx, dy
Reply: e.ok on success else error code

Function: Move text cursor relative to the current position.

carriage.return

Parameters: None
Reply: e.ok on success else error code

Function: Moves the cursor to the left hand edge of the currently selected window, or screen, on the current line.

line.feed

Parameters: None
Reply: e.ok on success else error code

Function: Moves the cursor down by the height of the characters being drawn. The screen, or window is scrolled when the command is issued on the bottom line.

select.screen

Parameters: screenNumber
Reply: e.ok on success else error code

Function: select a screen as the current draw screen. All future operations will draw into this screen.

display.screen

Parameters: screenNumber
Reply: e.ok on success else error code

Function: select a screen as the current display screen. The contents of the display screen will be displayed.

clear.screen

Parameters: colourValue
Reply: e.ok on success else error code

Function: clear the current draw screen to the defined colour.

fast.clear.screen

Parameters: colour
Reply: e.ok on success else error code

Function: certain TRANSTECH TTG cards support an ultra high-speed screen clear. This utilises a feature of modern video RAMs, but is subject to certain constraints - the function must take place during frame flyback, and the pixel mask register must be set to zero. This function clears a whole bank of video RAM to **colour**, meeting all these constraints, so the user need not **wait.vblank** beforehand. The function always clears the invisible screen, i.e the bank currently not selected for display. For a 1024x1024 display, this function takes less than 500μSeconds, i.e less than 500 picoSeconds per pixel.

flip.screen

Parameters: None
Reply: e.ok on success else error code

Function: flips between display screens.

copy.screen

Parameters: sourceScreenNumber, destinationScreenNumber
Reply: e.ok on success else error code

Function: copy the entire contents of one screen to another.

set.window

Parameters: xDimension, yDimension
Reply: window number

Function: Sets up a window with a given width and length. The number of the window is returned.

select.window

Parameters: windowNumber
Reply: window number

Function: Select a given window as the current draw window. All future operations draw into this window.

display.window

Parameters: windowNumber
Reply: e.ok on success else error code

Function: Select a given window as the current display window. The selected window is displayed.

clear.window

Parameters: colourValue
Reply: e.ok on success else error code

Function: clear the given window to the defined colour.

jump.scroll

Parameters: None
Reply: e.ok on success else error code

Function: The current draw window is scrolled vertically by one character height.

scroll

Parameters: None
Reply: e.ok on success else error code

Function: The current draw window is scrolled vertically by one line.

set.colour

Parameters: colourValue, red, green, blue
Reply: e.ok on success else error code

Function: writes a colour look-up table entry. Defines the red,green, and blue components of the given colourValue, which may be any of the 0..255 table locations. Each primary colour may have range 0..255.

select.fg.colour

Parameters: colourValue
Reply: e.ok on success else error code

Function: select the given colourValue as the current foreground colour. All subsequent text and primitives will appear in this colour.

select.bg.colour

Parameters: colourValue
Reply: e.ok on success else error code

Function: select the given colourValue as the current background colour. All subsequent text background will appear in this colour.

get.pixel.colour

Parameters: x, y
Reply: colourValue

Function: return the colourValue of the selected pixel on the current draw screen.

select.colour.table

Parameters: tableNumber
Reply: e.ok on success else error code

Function: select one of the three built in colour tables.

set.mask.register

Parameters: mask
Reply: e.ok on success else error code

Function: set the mask register in the look-up table to the defined value. This value is ANDed with input pixel before the pixel is presented to the look-up RAM.

set.xwidth

Parameters: width
Reply: e.ok on success else error code

Function: set the character pixel width. Defaults to 1, giving characters 8 pixels wide. When not set to 1, all plot operations e.g line drawing, text will generate fat pixels.

set.yheight

Parameters: yheight
Reply: e.ok on success else error code

Function: set the character pixel height. Defaults to 1, giving characters 8 pixels high. When not set to 1, all plot operations e.g line drawing, text will generate tall pixels.

line.frequency

Parameters: frequency
Reply: e.ok on success else error code

Function: change the line frequency used in all subsequent **init.crt** requests.

frame.rate

Parameters: frameRate
Reply: e.ok on success else error code

Function: change the frame rate, which is used in all subsequent **init.crt** requests.

interlace

Parameters: enabled
Reply: e.ok on success else error code

Function: a boolean which defines whether interlaced output is enabled. This defaults to FALSE on startup.

display.x

Parameters: xSize
Reply: e.ok on success else error code

Function: set number of pixels per scan line. Certain TTG cards only support a fixed number of pixels per line, in which case this function has no effect.

display.y

Parameters: ySize
Reply: e.ok on success else error code

Function: set number of scan lines to display. Certain TTG cards only support a fixed number of scan lines, in which case this function has no effect.

select.gcursor

Parameters: [16][16] BYTE cursor,
BYTE edgeChar, transparentChar, interiorChar,
BYTE colourOfEdge, colourOfInterior
Reply: e.ok on success else error code

Function: this function can be used to re-define the graphics cursor. The function expects an 16*16 BYTE array, each character defines a single pixel of the cursor. e.g.

```
char cursor[16][16] = {    "       ooo        ",
                           "       o+o        ",
                           "       o+o        ",
                           "       o+o        ",
                           "       o+o        ",
                           "  oooooo+oooooo   ",
                           " o++++++++++++++o ",
                           " o++++++++++++++o ",
                           "  oooooo+oooooo   ",
                           "       o+o        ",
                           "       o+o        ",
                           "       o+o        ",
                           "       o+o        ",
```

```
                                "       o+o       ",
                                "       ooo       " };
```

In this example:

```
edgeChar        = 'o';
transparentChar = ' ';
interiorChar    = '+';
colourOfEdge    =  10;
colourOfEdge    = 256;
```

The parameter *edgeChar* defines the character to be interpreted as the cursor edge, the *transparentChar* parameter defines the character used to denote the transparent background and the *interiorChar* parameter defines the character used to denote the interior of the cursor. The border colour of the cursor is given by *colourOfEdge*, and the background colour is given by *colourOfInterior*. The transparent colour is of course 0, so that the transputer's MOVE2D operations can be used.

remove.gcursor

Parameters: None
Reply: e.ok on success else error code

Function: remove the graphics cursor from the current display screen. Its previous draw location is stored by the Server.

draw.gcursor

Parameters: x, y
Reply: e.ok on success else error code

Function: draw the graphics cursor at the defined position. This position is stored for the following **remove.gcursor** operation.

wait.vblank

Parameters: None
Reply: e.ok

Function: do not proceed until vertical blanking is in progress. Used for synchronising frame flipping to prevent screen shearing.

12 Examples

Here are some program fragments illustrating use of the TTG Server. There are two ways of using the TTG server - it will either run concurrently with the application, if the application is to run on the graphics card, or it will run in isolation on the graphics card, with another application processor communicating with it via a link.

These examples assume the latter [2], that the TTG Server is running on another processor communicating with the one running the example code. The most common set-up will be - a TRANSTECH TMB08 motherboard (or equivalent) in a PC, with a TRANSTECH TTM 6 (or equivalent) in site 0, plus a TTG graphics TRAM (e.g TTG1, TTG3) in site 1. This is the assumed configuration, but different configurations are trivial.

The examples are given in occam (IMS D700D or IMS D705B) and C (3L Parallel C).

The first example sets up the system to a known state, selecting video parameters and a colour table. Here is the occam :-

```
CHAN OF ANY toTTG :
PLACE toTTG AT 2  :     -- on TMB08, site0 link2 ==> site1 link1
CHAN OF ANY fromTTG :
PLACE fromTTG AT 6  :

#USE "TTGconsts.tsr"
-- for the TOOLSET this would be #INCLUDE "TTGconsts.occ"

PROC TTGcommand ( INT reply, VAL INT command )
  -- only used for parameterless commands
  SEQ
    toTTG   ! command
    fromTTG ? reply
:

PROC TTGparams ( INT reply, VAL INT command, VAL [] INT params )
  -- used for all commands with INT parameters
  SEQ
    toTTG   ! command
    SEQ i = 0 FOR SIZE params
      toTTG   ! params [i]
    fromTTG ? reply
:

INT reply :
SEQ
  TTGparams  ( reply, c.line.frequency, [ 34800 ] )
  TTGparams  ( reply, c.frame.rate,     [ 60 ] )
  TTGparams  ( reply, c.interlace,      [ INT FALSE ] )
  TTGparams  ( reply, c.pixel.clock,    [ 25000000  ] )
```

[2] The version of TTGS supplied with each TRANSTECH graphics card includes examples of each method of operation.

```
  TTGparams  ( reply, c.x.size,     [ 512 ] )
  TTGparams  ( reply, c.y.size,     [ 512 ] )
  TTGcommand ( reply, c.init.crt )
  TTGparams  ( reply, c.select.screen, [0] )
  TTGparams  ( reply, c.display.screen, [0] )
```

This code would compile and run as an **EXE**. Here the equivalent C :-

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

#define SERVER_CODE "TTGS.btl"
#define SEEK_SET 0
#define SEEK_END 2

#define LINE_FREQ 34800
#define FRAME_RATE 50
#define PIXEL_CLOCK 25000000

#include "ttglib.h"

/* Define Library interface channels */
extern CHAN *to_ttglib, *from_ttglib;

/* Global data */

static CHAN *in_links[4] = {Link0Input,Link1Input,Link2Input,Link3Input};
static CHAN *out_links[4] = {Link0Output,Link1Output,Link2Output,Link3Output};

/* function declarations */
void error();
int read_binary();
int setup_vtg();


int main(argc, argv, envp, in_ports, ins, out_ports, outs)
int argc, ins, outs;
char *argv[], *envp[];
CHAN *in_ports[], *out_ports[];
{
        int result, fsize;
        char *code;
        int link;


        /* pickup server link argument */
        if (argc < 2) {
                error("SYNTAX: cdemo <link.no>",0);
                return(-1);
```

```
        }
        link = atoi(argv[1]);

        /* assign channel pointers */
        to_ttglib = out_links[link];
        from_ttglib = in_links[link];

        /* read library binary */
        if ((fsize = read_binary(&code)) <= 0) return(-1);

        chan_out_message(fsize, code, to_ttglib);       /* BOOT it */

        printf("TTG Server BOOTED through link %d!\n",link);

        free(code);     /* free code buffer */

        /* SETUP the VTG */

        if (! setup_vtg()) return(-1);


        /* select + display screen 0 */
        TTG ( SELECT_SCREEN,  0 );
        TTG ( DISPLAY_SCREEN, 0 );

        if ((result = TTG(c_terminate)) != e_ok) {
                error("failed to terminate TTGS",result);
                return(-1);
        }

        return(0);
}


void error(mssg, err)
char *mssg;
int err;
{
        printf("ERROR: %s [%d]\n",mssg,err);
}


int read_binary(codebuffer)
char **codebuffer;
{
        char *code;
        int  result, fsize;
        FILE *fp;

        /* read library */
        if ((fp = fopen(SERVER_CODE,"rb")) == (FILE *)NULL) {
```

```
                error("failed to open TTGServer code",fp);
                return(-1);
        }
        if ((result=fseek(fp,0,SEEK_END)) == -1) {
                error("fseek failed",result);
                return(-1);
        }
        if ((fsize=ftell(fp)) == -1) {
                error("ftell failed",result);
                return(-1);
        }
        if ((result=fseek(fp,0,SEEK_SET)) == -1 ) {
                error("fseek failed",result);
                return(-1);
        }
        if ((code = malloc(fsize+10)) == (char *)NULL) {
                error("Insufficient Memory for code",0);
                return(-1);
        }
        if ((result=fread(code,sizeof(char),(fsize),fp)) != (fsize)) {
                error("Failed to read whole file",result);
                return(-1);
        }
        fclose(fp);

        *codebuffer = code;     /* return code buffer address */
        return(fsize);
}


int setup_vtg()
{
        int result;
        if ((result = TTG(c_line_frequency, LINE_FREQ)) != e_ok) {
                error("error on line frequency",result);
                return(0);
        }
        if ((result = TTG(c_frame_rate, FRAME_RATE)) != e_ok) {
                error("error on frame rate",result);
                return(0);
        }
        if ((result = TTG(c_pixel_clock, PIXEL_CLOCK)) != e_ok) {
                error("error on frame rate",result);
                return(0);
        }
        if ((result = TTG(c_init_crt)) != e_ok) {
                error("error on init_crt",result);
                return(0);
        }
        return(1);
}
```

This is a substantially larger example than the preceding occam. The C example in fact dynamically boots the file **TTGS.btl** out of a link, specified at the command line. After compilation and linkage, the command **cdemo 2** will run the demo.

Now an occam example which fills a triangle parameters and a colour table. Here is the occam :-

```
... include constants, define procedures

PROC triangle ( VAL INT colour, VAL [2] INT p0, p1, p2 )
  INT reply, xcen, ycen :
  SEQ
    TTGparams ( reply, c.select.fg.colour, [ colour ])
    TTGparams ( reply, c.draw.line, [ p0[0], p0[1], p1[0], p1[1] ] )
    TTGparams ( reply, c.draw.line, [ p1[0], p1[1], p2[0], p2[1] ] )
    TTGparams ( reply, c.draw.line, [ p2[0], p2[1], p0[0], p0[1] ] )
    xcen := ((p0[0]+p1[0])+p2[0])/3
    ycen := ((p0[1]+p1[1])+p2[1])/3
    TTGparams ( reply, c.quick.fill, [ xcen, ycen ] )
:

INT reply :
SEQ
  TTGparams  ( reply, c.line.frequency, [ 34800 ] )
  TTGparams  ( reply, c.frame.rate,     [ 60 ] )
  TTGparams  ( reply, c.pixel.clock,    [ 25000000 ] )
  TTGcommand ( reply, c.init.crt )
  TTGparams  ( reply, c.select.screen,  [0] )
  TTGparams  ( reply, c.display.screen, [0] )
  TTGparams  ( reply, c.clear.screen,   [0] )
  triangle   ( 30, [10, 10], [400, 90], [180, 450] )
```

and again the equivalent C :-

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

#define SERVER_CODE "TTGS.btl"
#define SEEK_SET 0
#define SEEK_END 2

#define LINE_FREQ 34800
#define FRAME_RATE 50
#define PIXEL_CLOCK 25000000

#include "ttglib.h"
```

```
/* Define Library interface channels */
extern CHAN *to_ttglib, *from_ttglib;

/* Global data */

static CHAN *in_links[4] = {Link0Input,Link1Input,Link2Input,Link3Input};
static CHAN *out_links[4] = {Link0Output,Link1Output,Link2Output,Link3Output};

/* function declarations */
void error();
int read_binary();
int setup_vtg();


int main(argc, argv, envp, in_ports, ins, out_ports, outs)
int argc, ins, outs;
char *argv[], *envp[];
CHAN *in_ports[], *out_ports[];
{
        int result, fsize;
        char *code;
        int link;


        /* pickup server link argument */
        if (argc < 2) {
                error("SYNTAX: cdemo <link.no>",0);
                return(-1);
        }
        link = atoi(argv[1]);

        /* assign channel pointers */
        to_ttglib = out_links[link];
        from_ttglib = in_links[link];

        /* read library binary */
        if ((fsize = read_binary(&code)) <= 0) return(-1);

        chan_out_message(fsize, code, to_ttglib);       /* BOOT it */

        printf("TTG Server BOOTED through link %d!\n",link);

        free(code);     /* free code buffer */

        /* SETUP the VTG */

        if (! setup_vtg()) return(-1);


        /* select + display screen 0 */
        TTG ( SELECT_SCREEN,  0 );
```

```
        TTG ( DISPLAY_SCREEN, 0 );
        TTG ( CLEAR_SCREEN, 0 );

        triangle   ( 30, 10, 10, 400, 90, 180, 450 );

        if ((result = TTG(c_terminate)) != e_ok) {
                error("failed to terminate TTGS",result);
                return(-1);
        }

        return(0);
}


void triangle ( c, x0, y0, x1, y1, x2, y2 )
int c, x0, y0, x1, y1, x2, y2;
{
  int xcen, ycen;

  TTG ( SELECT_FG_COLOUR, c );
  TTG ( DRAW_LINE, x0, y0, x1, y1 );
  TTG ( DRAW_LINE, x1, y1, x2, y2 );
  TTG ( DRAW_LINE, x2, y2, x0, y0 );

  xcen = (x0+x1+x2)/3;
  ycen = (y0+y1+y2)/3;
  TTG ( QUICK_FILL, xcen, ycen );
}
```

For brevity, **setup_vtg**, **read_binary** and **error** were omitted from this second example.

13 Error Codes

The following error codes may be returned by the TTG Server.

Error	Code
e.ok	0
e.out.of.drawing.range	-1
e.invalid.screen	-2
e.invalid.window	-3
e.no.window.store	-4
e.too.many.windows	-5
e.unknown.drawing.mode	-6
e.invalid.rotation	-7
e.invalid.colour	-8
e.char.out.of.range	-9
e.string.length.exceeded	-10
e.invalid.colour.table	-11
e.invalid.command	-12

Table 5: Error Codes